UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Control remoto bifrecuencia de un cuadricóptero

TRABAJO FIN DE MÁSTER

Máster Universitario en Automática e Informática Industrial

Autor: Carbonell Lázaro, Rafael

Tutor: Salt Llobregat, Julián José

Curso 2018-2019

Agradecimientos

Agradezco a mi tutor Julián Salt por su guía y ayuda durante esta investigación, al profesor Vicente Casanova por su inestimable colaboración, y a mi novia, Paula, por su paciencia y compañía, junto a mis padres, Rafa y Ángeles y mi hermana Lourdes.

Resum

Aquest treball final de màster té com a objectiu controlar un dron mitjançant tècniques de bifreqüència en orientació. El bucle de control es compon bàsicament de tres tasques: mesurament, càlcul de control i actuació. En primer lloc, començarem per a un controlador PD en monofreqüència, i des d'aquest punt, investigarem si és possible aplicar la bifreqüència i quines són les seus límits de funcionament tant en simulació com la implementació real. Un punt a destacar és que la comunicació de paràmetres i referència es fa de forma remota a través del bluetooth.

Paraules clau: dron, control, bifreqüència, remot, bluetooth

Resumen

Esta trabajo final de máster tiene como objetivo controlar un dron mediante técnicas de bifrecuencia en orientación. El bucle de control se compone básicamente de tres tareas: medición, cálculo de control y actuación. En primer lugar, empezaremos por a un controlador PD en monofrecuencia, y desde este punto, investigaremos si es posible aplicar la bifrecuencia y cuales son sus limites de funcionamiento tanto en simulación como la implementación real. Un punto a destacar es que la comunicación de parámetros y referencia se hace de forma remota a través del bluetooth.

Palabras clave: dron, control, bifrecuencia, remoto, bluetooth

Abstract

This final master's work aims to control a drone using dual rate techniques in orientation. The control loop basically consists of three tasks: measurement, control calculation and actuation. First, we will start with a mono frequency PD controller, and from this point, we will investigate if it is possible to apply the dual rate and what are its limits of operation both in simulation and actual implementation. A point to note is that the communication of parameters and reference is done remotely via bluetooth.

Key words: drone, control, dual rate, remote, bluetooth

Índice general

Índice de figuras

CAPÍTULO 1
Introducción

Este trabajo fin de máster consiste en el control de cuadricóptero, comercialmente conocido como «dron» o «drone», con la intención de mantener la estabilidad en orientación.

La particularidad de este sistema de control es que en primera instancia se basara en un sistema de control monofrecuencia y posteriormente se transformara en un sistema bifrecuencia con la intención de reducir el coste computacional, incluso introducir control «offboard».

1.1 Motivación

La motivación de este trabajo nace de la industria de los vehículos aéreos no tripulados al ser un sector al alza, y que en consecuencia, creará miles de puestos de trabajo en un futuro. Actualmente e históricamente el campo de los UAV ha ido acompañado de la industria militar, aunque en esta última década su uso a nivel civil ha ido en aumento.

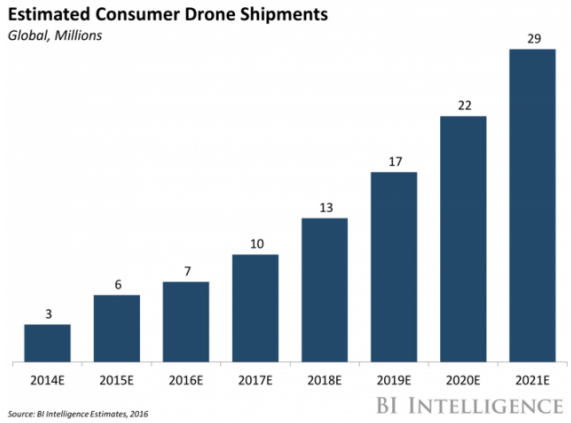

Figura 1.1: Gráfica del consumo comercial estimado de drones basada en el año 2017

Otra de las motivaciones por este proyecto viene dada de retomar un proyecto anterior, cuando se despertó el interés por formar parte de la comunidad «Hazlo tu mismo», el término es más conocido en ingles «Do It Yourself (DIY)».

El primer proyecto se realizó en la construcción por piezas de un dron. En aquel momento, empecé con el montaje mirando tutoriales por Internet, viendo

drones construidos por otra gente, hasta el momento que opté por realizar el desarrollo del drone y compré todas piezas pensando en que seria fácil realizar el proyecto, no resulto ser nada trivial. Después de todo, alguna vez, conseguí volar hacer volar el dron, aunque sufrió algún que otro accidente.

Con esto quiero decir que he querido retomar este proyecto en algún momento. Hablé con mi tutor de sobre el proyecto anterior, y teniendo más de conocimiento sobre las materias de control, sentí la necesidad de adentrarme en el mundo del control en el vuelo, aunque dándole otro enfoque, realizando mi propio software de vuelo, aprendiendo los pilares básicos de este tipo de aeronaves y aportando mi grano de arena.

1.2 Objetivos

El objetivo principal del proyecto es la realización de un software de control de estabilidad para un dron de cuatro hélices sobre la plataforma de desarrollo para microcontroladores arduino. Este objetivo está relacionado con los siguientes objetivos académicos:

- Investigar sobre aspectos de control básico aplicados a los drones.
- Programación de un software de vuelo de un drone.
- Construcción por piezas de un drone.
- Investigar sobre el control bifrecuencia.
- Aplicar aspectos de programación.
- Aplicar conocimientos de tiempo real para la planificación del sistema.

1.3 Estructura de la memoria

La estructura de la memoria se compone de 7 secciones. Estas secciones son la introducción, la mecánica de vuelo que nos introduce a una aproximación matemática del modelo del sistema, el posterior apartado corresponde al propio modelo matemático estudiado.

El siguiente apartado ya corresponde al diseño del cuadricóptero que va desde los componentes, un análisis de tiempo real, el control, la sensorización y los actuadores. Los penúltimos apartados constan de la simulación y la implementación del drone. Y por último las conclusiones obtenidas de este trabajo.

CAPÍTULO 2
Mecánica de vuelo

La mecánica de vuelo de un cuadricóptero, como su propio nombre indica se basa en la descomposición de su nombre «cuadri» y «cóptero» al disponer de cuatro rotores con hélices que le permiten realizar los movimientos de desplazamiento horizontal y vertical.

El control del movimiento se logra variando la velocidad angular de cada uno de los cuatro motores. Los cuadricópteros poseen tres tipos de movimiento: alabeo o «roll», cabeceo o «pitch» y guiñada o «yaw». Estos movimientos les concede 6 grados de libertad a los cuadricópteros.

Un cuadricóptero consiste en una estructura central en la cual se encuentran anclados la batería y la electrónica de vuelo. Las placas centrales se unen a los cuatro brazos equidistantes los cuales respectivamente tienen un motor unido a un hélice en el extremo. Estos forman una cruz brindando la posibilidad de sustentarse en el aire controlando su orientación y traslación. El movimiento básico de un cuadricóptero se basa en controlar la secuencia de empujes que generan sus rotores.

El movimiento de traslación del cuadricóptero requiere inclinar la plataforma hacia el eje deseado generando un desequilibrio en la secuencia de empujes de sus rotores. Por lo tanto, de manera similar a los helicópteros tradicionales, el movimiento traslación y rotación están acoplados. Básicamente, cambiar la velocidad de un motor puede causar un movimiento en tres grados de libertad. Como consecuencia tenemos la posibilidad de movernos en seis grados de libertad controlando los cuatro motores del cuadricóptero.

2.1 Configuración

El cuadricóptero propuesto en este documento se desarrolla en un configuración en cruz «x». Normalmente, los rotores se sitúan simétricamente formando una cruz «+» o una equis «x» como podemos ver en las figuras 2.1 y 2.2. Aunque podemos encontrar otro tipo de configuraciones como los drones de carreras «V» o en configuración en hache «H».

En todas estas configuraciones, los motores giran dos en sentido horario y dos en sentido antihorario con la razón de cancelar entre si el par de fuerzas genera-

dos. Los motores con el mismo sentido de giro están dispuestos en el mismo eje del chasis estando alineados como podemos ver en las figuras 2.1 y 2.2.

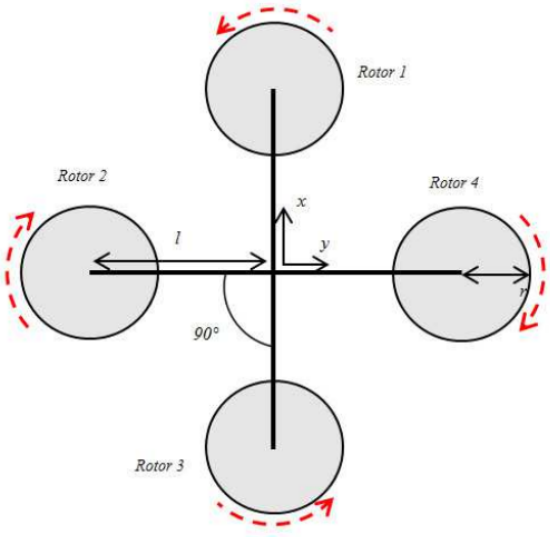

Figura 2.1: Configuración en cruz «+»

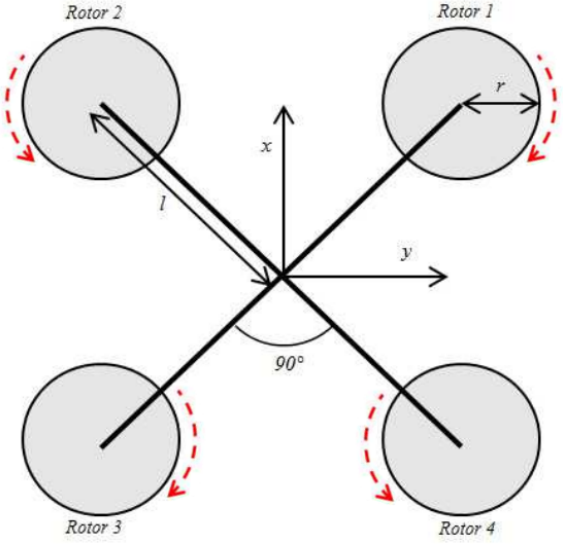

Figura 2.2: Configuración en equis «x»

Como ya hemos dicho, la configuración que emplearemos es la disposición en equis «x», ya que como veremos más adelante, esta emplea todos los motores para realizar todo la serie de movimientos a los largo de sus 3 ejes. En la figura 2.4 podemos ver la vista del perfil con el sistema inercial «X, Y y Z» haciendo referencia el eje «X» al Norte o «North», el eje «Y» al Este o «East» y el eje «Z» hacia Abajo o «Down», siguiendo el sistema de coordenadas NED. El sistema NED esta asociado al sistema tierra con respecto al objeto.

Por otra parte, tenemos asociado el sistema inercial al cuerpo del drone que implicaría una traslación y rotación desde este sistema NED principal XYZ asociando a cada eje xyz un ángulo de giro, siendo asociado al eje «x» el roll (ϕ), al eje «y» el pitch (θ) y al eje «z» el yaw (ψ).

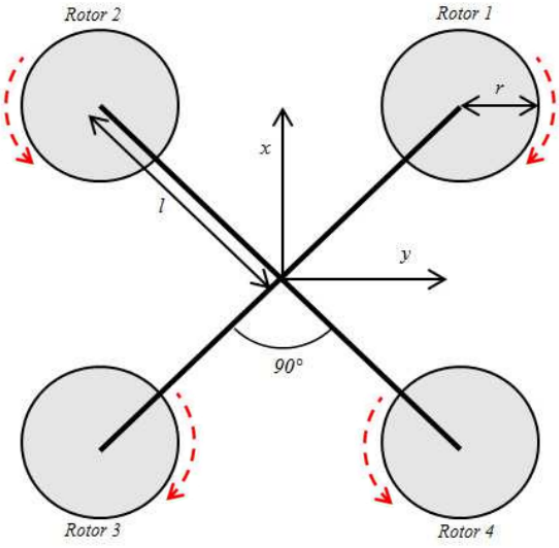

Figura 2.3: Vista de planta

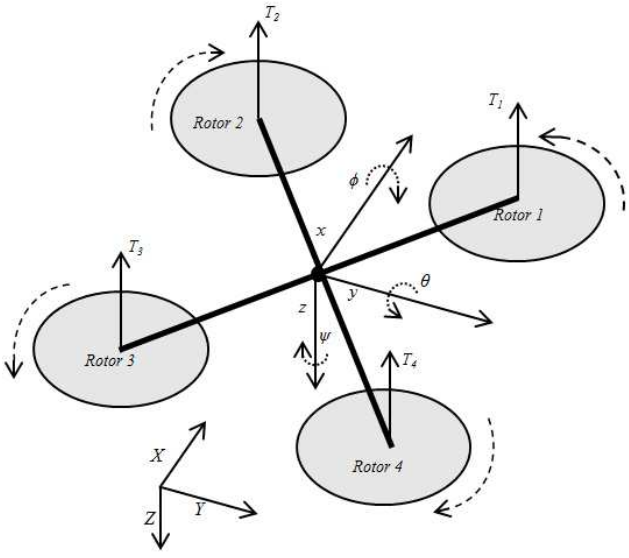

Figura 2.4: Vista de perfil

2.2 Movimientos

Antes de seguir con esta sección, cabe destacar una aclaración que puede llevar a confusión, un aspecto son los ejes asociados a un objeto, que este es fijo con el objeto y otro es el ángulo asociado al eje como son el «Roll» (ϕ) para el eje «x», el «Pitch» (θ) para eje «y», y el «Yaw» (ψ) para el eje «z».

Y cualquier variación del ángulo asociado a un eje, provoca como consecuencia que los otros dos ejes roten respecto a este, provocando un movimiento del sistema de ejes asociado al objeto, y por tanto, moviendo todo el objeto respecto este eje. Cuando esto se produce a lo largo del tiempo, se habla de velocidad angular respecto a un eje, usado para medir la variación a lo largo del tiempo.

Para generar los movimientos básicos, los rotores deben variar la velocidad de giro para inducir empuje en un cierto orden, dependiendo de la posición y el movimiento deseado del cuadricóptero. Como habíamos indicado anteriormente, el control del movimiento se logra variando la velocidad angular de cada uno de los cuatro motores. Los cuadricópteros poseen tres tipos de movimiento: alabeo o «roll», cabeceo o «pitch» y guiñada o «yaw». Además del movimiento de ascender o descender, en ingles conocido como «Thrust».

En el caso principal para tener los movimientos de elevación, aceleración , descenso o inclusive un vuelo estático en el aire; es necesario trabajar con los cuatro rotores al mismo tiempo cambiando todo la velocidad angular en la misma cantidad. El empuje en la dirección vertical es producido por la suma de todas las fuerzas de los rotores. Todo esto con la intención de vencer a la fuerza generada por la masa de la aeronave y la gravedad, como podemos observar en la figura 2.5.

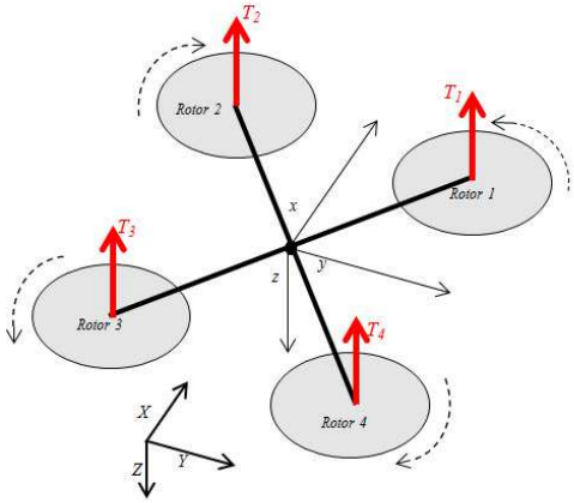

Figura 2.5: Vista del empuje principal

Para vuelo estacionario, el empuje de todos los rotores debe ser equivalente. Pequeñas diferencias pueden causar el cuadricóptero inclinarse e incluso volverse inestable. Aunque el mecanismo de control de vuelo cuadricóptero puede parecer simple, en realidad en vuelo, es casi humanamente imposible volar sin la ayuda de un controlador. Para ayudar al piloto, se utilizará un sensor inercial que ayuda a regular la orientación del cuadricóptero.

El movimiento de cabeceo o pitch (θ) le permite realizar movimientos hacia adelante y atrás aplicando torque alrededor del eje «y». Al variar la velocidad de los rotores delanteros contra los motores traseros hará que la aeronave se incline alrededor del eje «y», y se desplace hacia adelante y viceversa si se quiere que avance hacia detrás.

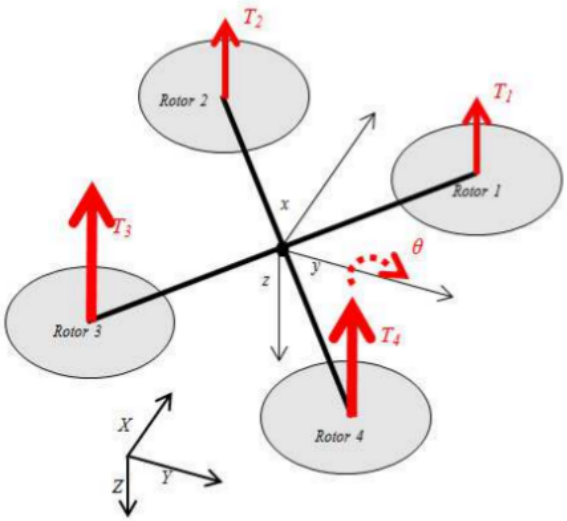

Figura 2.6: Vista del movimiento de pitch hacia delante

El movimiento de alabeo sigue el mecanismo que el movimiento de cabeceo pero alrededor del eje «x». El movimiento de alabeo o roll (ϕ) permite al cuadricóptero desplazarse a la izquierda o derecha provocada por la diferencia del par de los rotores izquierdos y derechos.

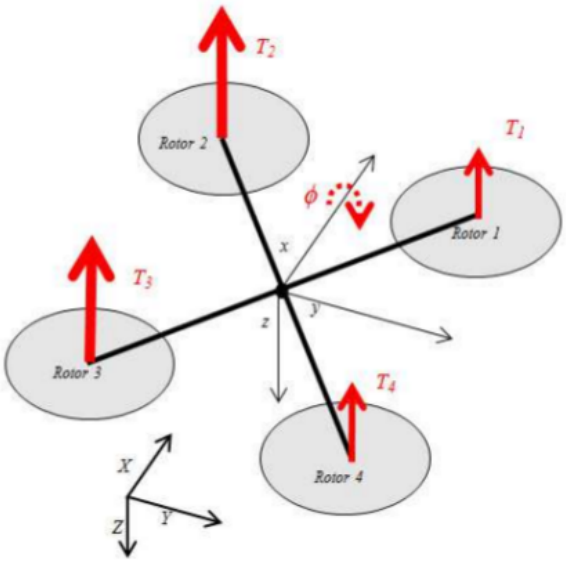

Figura 2.7: Vista del movimiento de roll hacia derecha

El movimiento de guiñada o yaw (ψ permite al cuadricóptero girar respecto al eje «z» o eje vertical. Para lograr un giro sobre el eje vertical, se tiene que aumentar el empuje de los dos rotores que giran en el mismo sentido. En este caso, serían el de la parte frente-derecha y el de la parte atrás-izquierda al mismo tiempo que debemos disminuir el de frente-izquierda y el de atrás-derecha. En consecuencia,

tenemos un giro en sentido horario. Por otra parte, para dar un giro en dirección contraria tendríamos que hacer lo opuesto.

En el sentido más técnico, el movimiento de guiñada se introduce al ejercer par alrededor del eje «z». La aplicación de diferentes velocidades de rotación al par de motores contrarrotativos, provoca un cambio de rumbo. De hecho, el movimiento de guiñada es generado por el par reactivo de los rotores.

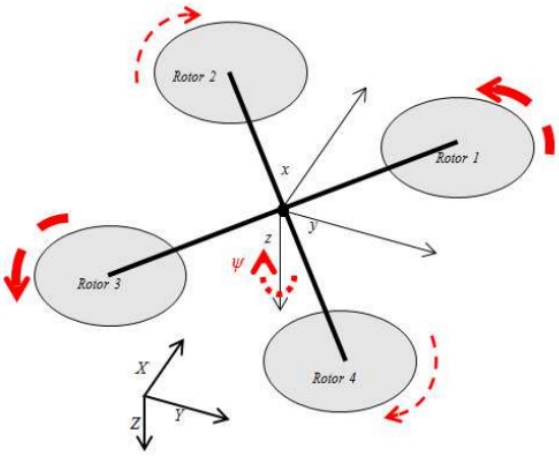

Figura 2.8: Vista del movimiento de yaw en sentido horario

Por último, destacar que una vez entendidas estos conceptos básicos asociados al vuelo de un cuadricóptero, podremos entender mejor el funcionamiento del modelo matemático.

CAPÍTULO 3
Modelado matemático

Este capítulo describe el modelo matemático de un cuadricóptero en configuración en equis «X».[4] En primer lugar, hay que definir las ecuaciones dinámicas y cinemáticas del drone. El modelo dinámico describe como actúan las fuerzas y resultan los torques en la rotación y la translación del drone. Las ecuaciones cinemáticas describen la relación entre la posición y velocidad del vehículo. El sistema inercial y el sistema objeto como drone son introducidos en la figura 3.1.

El origen del sistema drone se asume que es el centro de gravedad ubicado el centro de la estructura del drone. Por otra parte, el sistema inercial sigue el sistema de coordenadas NED (North-East-Down) explicado anteriormente. El sistema inercial y el sistema drone se describen como $F^I = (e^x, e^y, e^z)$ y $F^B = (e^{xb}, e^{yb}, e^{zb})$ respectivamente.

La orientación del sistema drone con respecto al sistema inercial NED se define como $\eta \triangleq [\phi, \theta, \psi]^T$, conocido como roll, pitch, yaw respectivamente; y la velocidad angular se define como $\omega^b \triangleq [p, q, r]^T$. La posición con respecto al sistema inercial se define como $\xi \triangleq [x, y, z]^T$ mientras que las velocidades del drone se definen como $v^b \triangleq [u, v, w]^T$.

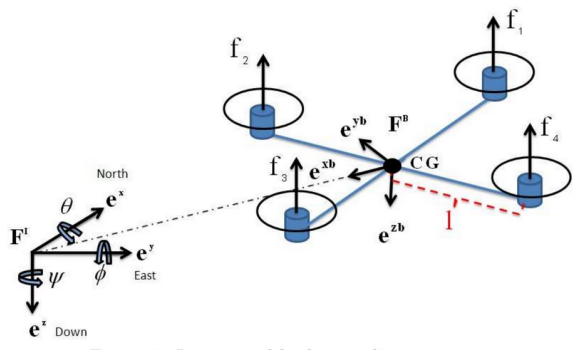

Figura 3.1: Sistemas de coordenadas: el inercial y el drone

La ecuación dinámica del drone sometida a fuerzas y momentos de actuación como un cuerpo rígido en el esquema de Newton-Euler es:

$$\begin{bmatrix} mI_{3x3} & 0_{3x3} \\ 0_{3x3} & J \end{bmatrix} \begin{pmatrix} \dot{v}^b \\ \dot{\omega}^b \end{pmatrix} + \begin{bmatrix} \omega^b \times (mv^b) \\ \omega^b \times (J\omega^b) \end{bmatrix} = \begin{pmatrix} F^b \\ \tau^b \end{pmatrix} \tag{3.1}$$

Donde $m[kg]$ es la masa de la plataforma, I es una matriz identidad y J es la matriz de momentos de inercias de la plataforma del cuadricóptero. La plataforma del drone tiene un estructura simétrica por lo tanto podemos asumir la matriz de inercias es esencialmente simétrica y constante en los tres ejes, lo que implica que $J = diag(J_{xx}, J_{yy}, J_{zz})$. F^b y τ^b son la fuerza total y el torque aplicado a la estructura del vehículo.

La velocidad del cuerpo se proyecta respecto al sistema de referencia inercial a través de la matriz rotacional, $\dot{\xi} = Rv^b$ siendo R:

$$R = \begin{pmatrix} c_\theta c_\psi & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{pmatrix} \tag{3.2}$$

Donde $c_k = cos(k)$ y $s_k = sin(k)$. La matriz de transformación de $\dot{\eta}$ a ω^b dado por la siguiente expresión $\dot{\eta} = \Gamma\omega^b$, donde:

$$\Gamma = \begin{pmatrix} 1 & sin(\phi)\,tan(\theta) & cos(\phi)\,tan(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & \frac{sin(\phi)}{cos(\theta)} & \frac{cos(\phi)}{cos(\theta)} \end{pmatrix} \tag{3.3}$$

Las principales fuerzas que actúan sobre el drone son su peso, los empuje del rotor y la resistencia de la carrocería. El empuje del motor f_i es proporcional al cuadrado de la velocidad angular de la hélice Ω_i y actúa a lo largo de e^z como podemos ver en la figura 3.1 y en la ecuación 3.4.

$$f_i = K_\Omega \Omega_i^2 \tag{3.4}$$

La velocidad de rotación del rotor es controlada a través del voltaje v_i:

$$\Omega_i = \frac{K_v}{1 + \tau_f s} v_i \tag{3.5}$$

Donde K_Ω y K_v son valores constantes, y τ^f es una constante de tiempo dado por modelo de empuje del rotor. A medida que la plataforma se mueve por el aire, produce resistencia. Sin embargo, desde que nuestra plataforma se movió como un pequeño el drone a escala es lento, el arrastre generado es bastante pequeño. La fuerza de resistencia de la carrocería se puede escribir como:

$$D = -\frac{1}{2}\rho C_d |v^b| v^b \tag{3.6}$$

Donde ρ es la densidad del aire y C_d es el coeficiente de rozamiento. Mientras que la gravedad también ejerce una fuerza sobre el cuadricóptero. La fuerza actúa sobre el centro de gravedad CG y en el sistema de referencia inercial viene dada por:

$$f_g^I = \begin{pmatrix} 0 & 0 & mg \end{pmatrix}^T \tag{3.7}$$

Sin embargo, debemos definir como actúan todas estas fuerzas sobre la estructura del drone. Del sistema inercial al sistema drone aplicamos la matriz de rotación a la ecuación 3.7 obteniendo como resultado:

$$f_g^b = R^T f_g^I = \begin{pmatrix} -mg\,sin(\theta) \\ mg\,cos(\theta)\,sin(\phi) \\ mg\,cos(\theta)\,cos(\phi) \end{pmatrix} \tag{3.8}$$

Como resultado, obtenemos la siguiente ecuación para describir todas las fuerzas que actúan sobre el sistema drone:

$$F^b = -\frac{1}{2}\rho C_d |v^b| v^b + f_g^b - \sum_{i=1}^{4} f_i e^z \tag{3.9}$$

En cambio para los torques para el control del roll, pitch y yaw se definen como $(\tau_\phi, \tau_\theta, \tau_\psi)$ y son producidos principal por una diferencia de empuje en los rotores. Mientras que el roll y pitch se produce por una diferencia de motores opuestos, el yaw se produce por empuje diferencial entre los dos pares de motores contrarrotativos Q, y viene definido por:

$$Q_i = d\Omega_i \tag{3.10}$$

Donde d es el coeficiente de resistencia al torque inducido por el rotor que depende principalmente de las especificaciones de la hélice. Entonces, actuando sobre $\tau^b = [\tau_\phi, \tau_\theta, \tau_\psi]$ en el sistema drone según la siguientes ecuaciones.

$$\tau_\phi = \frac{\sqrt{2}}{2} l \left[(f_2 + f_3) - (f_1 + f_4)\right] \tag{3.11}$$

$$\tau_\theta = \frac{\sqrt{2}}{2} l \left[(f_1 + f_2) - (f_3 + f_4)\right] \tag{3.12}$$

$$\tau_\psi = Q_1 - Q_2 + Q_3 - Q_4 \tag{3.13}$$

El modelo dinámico completo del cuadricóptero se puede obtener sustituyendo las ecuaciones 3.9, 3.11, 3.12 y 3.13 en la ecuación 3.1 y proyectando el estado del sistema drone al sistema inercial con las matrices de transformación 3.2 y 3.3.

Aunque determinar todos los parámetros de la aeronave es un proceso arduo y complejo, por ello, las ecuaciones se simplifican a la entradas que podemos controlar, donde U es la potencia normalizada aplicada al motor i, y τ_{te} es un potencia nominal dada por los cuatro motores para compensar la gravedad, de estar forma se simplifica significativamente la ley de control.

$$\tau_{te} = U_1 + U_2 + U_3 + U_4 \tag{3.14}$$

$$\tau_\phi = (U_2 + U_3) - (U_1 + U_4) \tag{3.15}$$

$$\tau_\theta = (U_1 + U_2) - (U_3 + U_4) \tag{3.16}$$

$$\tau_\psi = U_1 - U_2 + U_3 - U_4 \tag{3.17}$$

CAPÍTULO 4
Diseño

Con respecto al apartado de diseño, es la sección más extensa del trabajo ya que consta de varias disciplinas como son parte mecánica, la eléctrica, la electrónica y la informática; teniendo en cuenta que no entramos en detalle en cada una de ellas, ya que el grueso es el control.

El apartado de diseño consta de las siguiente partes: los componentes que forman el cuadricóptero, la conexiones eléctricas y electrónicas como son los buses de comunicación y el esquema de conexión de los componentes eléctricos, también se contemplan aspecto de tiempo real en el cual hacemos un análisis, aproximación y explicación de porque este sistema cumple las especificaciones de tiempo real, luego pasamos al control, explicando cada una de las partes del control empleado, sus tipos y la justificación de que control emplear y por último damos una pequeña pincelada de la fusión sensorial debido a que el sensor inercial proporciona los datos en crudo y tiene la necesidad de ser fusionados para una mejor precisión.

4.1 Componentes

4.1.1. Motores y hélices

Los motores «brushless» son motores eléctricos sin escobillas. Estos motores se emplean en drones porque no sufren el desgaste de las escobillas y son eficientes al no tener tanta perdida de calor en relación con la potencia de entrada y de salida, aunque no suelen estar sensorizados por su alta velocidad, si se podría establecer cierto cierto control mediante la corriente suministrada y la devuelta por los tres grupos de bobinas.

El grupo de motores son los mostrados en la figura 4.3. Estos vienen en un lote de un motor a sentido horario o antihorario con una pareja de hélices horaria y antihoraria, al ser un lote diseñado con el fin de ser empleado para drones.

Además de tener el cono del motor distintivo donde el cono negro corresponde al apriete a izquierdas y el rojo a derechas, la intención es que mientras la tuerca se apriete en el sentido opuesto al de giro del motor para favorecer que la tuerca se mantenga apretada como medida de seguridad.

Figura 4.1: Motores «brushless» EMAX MT2213 935KV CW y CCW con pareja de hélices 10x4.5"

Las características del lote motor son las siguientes:

- Una pareja de hélices 10x4.5 pulgadas (normal e inversa)
- Diámetro: 27.9 mm
- Longitud: 39.7 mm
- Peso: 55 g
- Kv: 935
- Batería: 3-4 s
- Esc: 18-20 A

4.1.2. Controlador electrónico de velocidad (ESC)

El controlador electrónico de velocidad o «Electronic speed controller» es un circuito electrónico integrado que se emplea a modo de etapa de potencia entre la batería y el motor teniendo una referencia dada la energía suministrada para controlar la velocidad del motor. El ESC empleado es un controlador SimonK 30A 2-3S Brushless ESC con las siguientes características:

- Batería: 2-3 s
- Amperaje nominal: 30 A
- Con regulador de voltaje (BEC) de 5V y 3A
- Tamaño 50 x 23 x 8 mm
- Peso: 25 g

- Frecuencia de control del motor a 16 KHz.
- Frecuencia máxima de entrada PWM a 490 Hz
- Conector deans/T-plug

Figura 4.2: SimonK 30A 2-3 s Brushless ESC

4.1.3. Tarjeta de distribución energética (PDB)

La tarjeta de distribución energética o «power distribution board» es un una simple pcb que contiene un conector de entrada de la batería y cuatro conectores de salida para cada uno de los ESC con algunos cables adicionales que pueden servir para alimentar otros elementos como cámaras, la controladora de vuelo o medir el nivel de la batería.

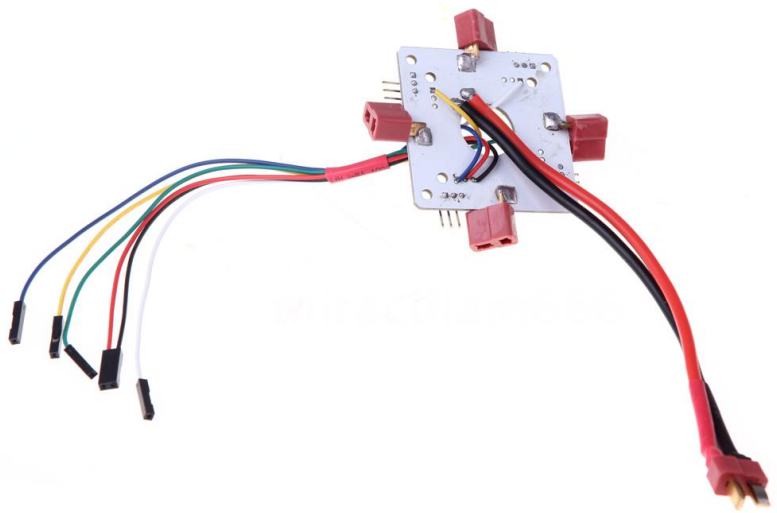

Figura 4.3: Power distribution board con conectores deans/T-plug

4.1.4. Controladora de vuelo - Arduino

El Arduino Due es la primera placa arduino basada en un microcontrolador de núcleo ARM en una arquitectura de 32 bits con una frecuencia de 84 Mhz (Atmel SAM3X8E ARM Cortex-M3). Cuenta con 54 pines digitales de entrada y salida (de los cuales 12 se pueden utilizar como salidas PWM), 12 entradas analógicas, 4 UARTs (puertos serie de hardware), una conexión capaz USB OTG, 2 DAC (de digital a analógico) , 2 TWI, un conector de alimentación, un conector de SPI, un conector JTAG, un pulsador de reset y un pulsador de borrado.

Hemos elegido la Arduino Due por los siguientes beneficios:

- Posee un núcleo de 32 bits.
- Reloj de 84 MHz.
- Entorno amigable y depurado de programación.
- Facilidad de programación.
- Gran soporte por la comunidad.
- Gran cantidad de librerías.
- 2 puertos serie USB. Permitiendo tener uno puerto serie para comunicación y otro para debugging.

En definitiva, un seria de comodidades y ventajas ofrecidas al programador que permite una programación cómoda, sencilla y comprensible. Además que para la aplicación dada con la potencia que nos suministra es más que suficiente.

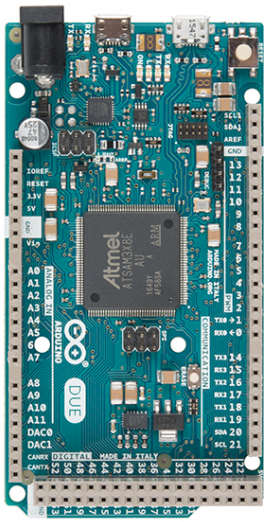

Figura 4.4: Arduino Due

4.1.5. Sensor inercial

La MPU 9250 es un sensor inercial de 9 grados de libertad fabricado por Invensense. Este sensor internamente esta compuesto por 3 sensores de 3 grados de libertad cada uno, un giroscopio, un acelerómetro y un magnetómetro que sirven para medir la velocidad angular, la aceleración lineal y el campo magnético de la tierra respectivamente.

Los sensores internamente se componen de unas membranas que según su disposición y forma son capaces de proporcionarnos un tipo de información. Pues estas membranas están conectadas a un conversor analógico digital de un resolución de 16 bits con un primer filtrado de la señal.

Con respecto a la resolución de estos sensores tenemos que el acelerómetro tiene disponibles ± 2 g, ± 4 g, ± 8 g y ± 16 g siendo g, la aceleración gravitatoria de la tierra. Mientras que el giroscopio tiene disponibles ± 250 dps, ± 500 dps, ± 1000 dps y ± 2000 dps (grados por segundo). Con respecto al magnetómetro tiene una única resolución de ± 4800 uT (microtesla).

Los buses de comunicación que incorpora el sensor son el bus de comunicación I2C y SPI. La electrónica funciona a 3,3 V manteniendo la compatibilidad eléctrica con los 5 V con un consumo de 3,5 mA con todo el conjunto de sensores internos en funcionamiento junto con el DMP activo. El DMP es un procesador digital integrado para la fusión de los ángulos.

En este proyecto descartaremos el uso del magnetómetro, ya que los motores generan demasiadas interferencias e introducen más ruido en la fusión que beneficio en la orientación relativa al norte magnético. Otro aspecto que descartamos es el uso del DMP porque la fusión de los sensores se hace dependiente

de esta imu, haciendo el código dependiente de esta imu, así pues apostamos el uso del filtro de Madgwick. Por ultimo, elegimos esta imu por su relación coste-rendimiento.

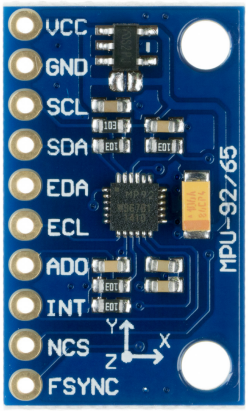

Figura 4.5: MPU 9250

4.1.6. Módulo de comunicación

El módulo HC-06 es un módulo bluetooth compatible con arduino. El módulo de bluetooth HC-06 es un módulo esclavo, quiere decir que puede de recibir conexiones desde un ordenador o tablet, es decir, un nodo maestro.

El HC-06 tiene 4 pines de conexión: VCC, GND, RX y TX. Los pines RX y TX son los pines empleados para la lectura y escritura del puerto serie. El pin TX es el pin de transmisión de datos, por este pin el HC-06 transmite los datos que le llegan desde el ordenador o móvil mediante bluetooth, este pin debe ir conectado al pin RX del arduino, en cambio el pin RX es el pin de recepción, a través de este pin el HC-06 recibirá los datos del arduino los cuales se transmitirán por bluetooth, este pin va conectado al Pin TX del arduino.

Especificaciones:

- Frecuencia: banda ISM de 2,4 GHz.
- Modulación: GFSK.
- Potencia de transmisión : menos de 4 dBm, Clase 2.
- Sensibilidad: Menos de -84 dBm en el 0,1 BER.
- Ratio asíncronos: 2.1 Mbps / 160 kbps.
- Ratio síncrono: 1 Mbps / 1 Mbps.
- Perfiles de la ayuda : puerto serie bluetooth (esclavo).
- Fuente de alimentación: + 3.3 VDC 50 mA.

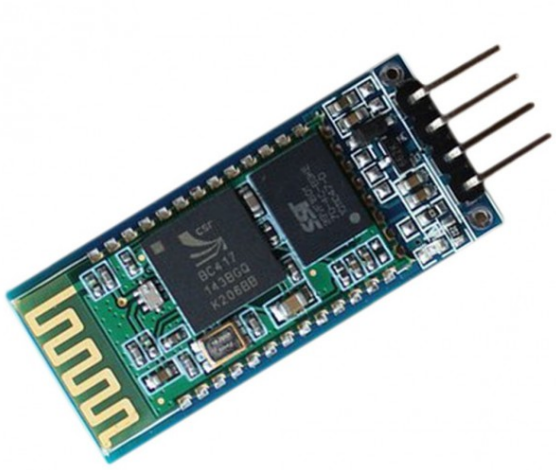

Figura 4.6: Módulo bluetooth HC-06

4.1.7. Circuito impreso para arduino

El circuito impreso es consecuencia de dos aspectos, un aspecto de desarrollo personal para aprender más sobre el diseño de pcbs y otro aspecto más importante es la organización y saneamiento de las conexiones entre los sensores y actuadores.

Debido que hasta el momento de la decisión del desarrollo del circuito integrado, las conexiones se realizaban mediante cables «dupont» conectados a los diferentes conectores, teniendo el riesgo de desconectarse y poder causar algún problema. Además de la farragosa tarea de volver a conectarlo, ya que había muchos y entorpecía esta labor.

Por ello, se decidió que realizar un circuito integrado seria de gran ayuda para mejorar la estética y organización de los cables, teniendo los conectores necesarios para interconectar todos los elementos de una forma cómoda.

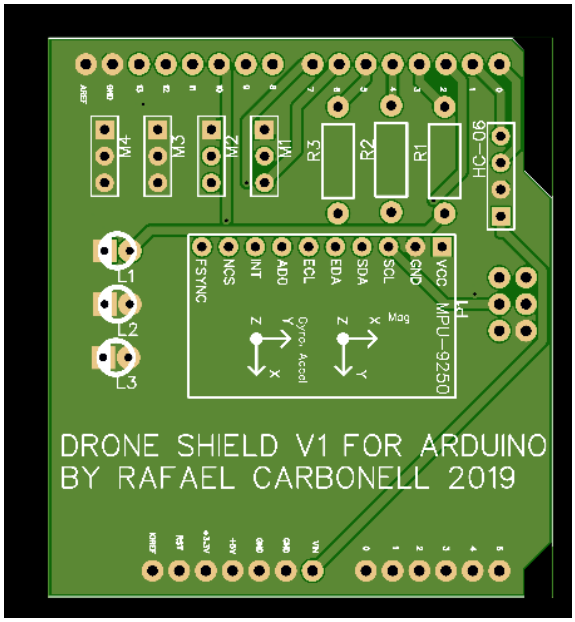

Figura 4.7: PCB shield

4.1.8. Chasis

El chasis es la estructura F450 para drones de cuatro rotores diseñado por DJI en 2015, una de las empresas más importante en el sector de los drones comerciales. Se eligió por su coste y sencillez de montaje, ya que consta únicamente de cuatro brazos y dos placas que se unen con unos tornillos.

Figura 4.8: Chasis DJI F450

4.2 Conexiones eléctricas y electrónicas

Este seria el esquema de conexionado de los elementos del drone, por una parte, tenemos la batería o fuente de alimentación que va conectado a la PDB, a esta están conectados los cuatro controladores de motor, estos tienen diversas conexiones.

El controlador de motor en primer lugar se conecta a su correspondiente motor, por otra parte, se conecta a la shield. Como el controlador de motor también regula el voltaje de la alimentación a cinco voltios, con este voltaje podemos ali-

mentar el arduino, el sensor inercial y el módulo de bluetooth. Por último, en este mismo conector tenemos otro cable dedicado al control PWM que emite el arduino hacia el controlador de motor.

Por ultimo, tenemos el sensor inercial, que va soldado a la shield y se conecta al arduino mediante el bus de comunicación SPI. Y el módulo bluetooth que va conectado a la salida serie de la shield.

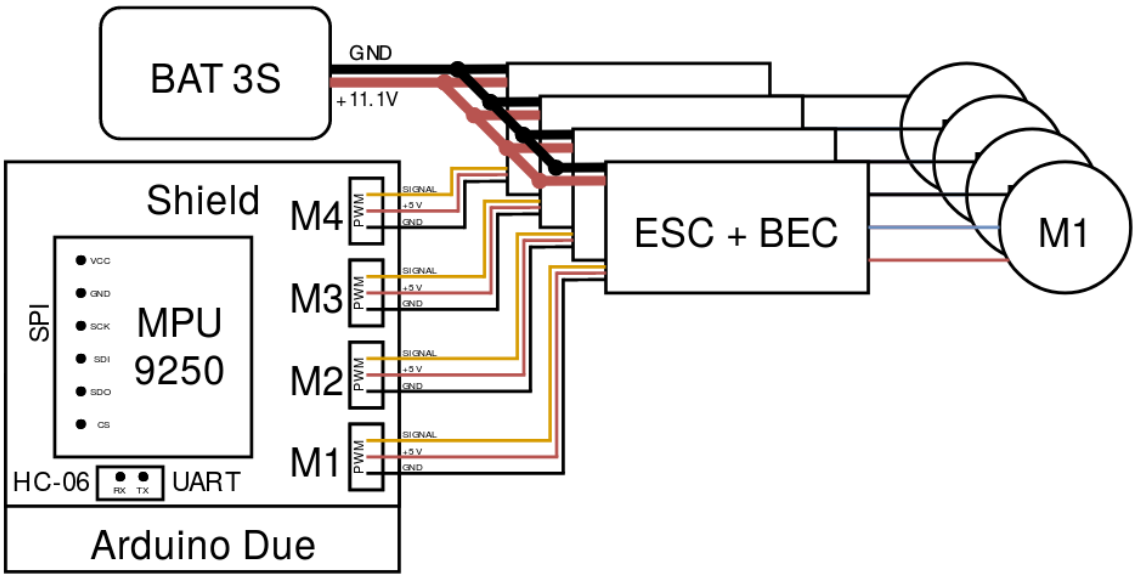

Figura 4.9: Esquema de conexión de los elementos eléctricos del drone

4.2.1. Bus Serie

El bus serie o «UART» (Universal Asynchronous Receiver/Transmitter) es un protocolo de bajo nivel en el que los datos son enviados, bit a bit y se reconstruyen por medio de registros o rutinas. Está formado por pocos conductores y su ancho de banda depende de la frecuencia.

Aunque originalmente fueron usados para conectar dispositivos lentos (como el teclado o un ratón), actualmente se están usando para conectar dispositivos mucho más rápidos como discos duros, unidades de estado sólido, tarjetas de expansión e incluso para el bus del procesador.

Destacar que el bus uart es bastante configurable, como es la frecuencia, el numero de bit por paquete («dataframe»), el bit de paridad y el tipo de parada del paquete. En la siguiente figura podemos ver un ejemplo de paquete 8N1. La frecuencia es indistinta en este tipo de figuras, la frecuencia que hemos empleado en este es 57600 baudios, ya que es lo suficientemente lenta para ser robusta frente interferencias.

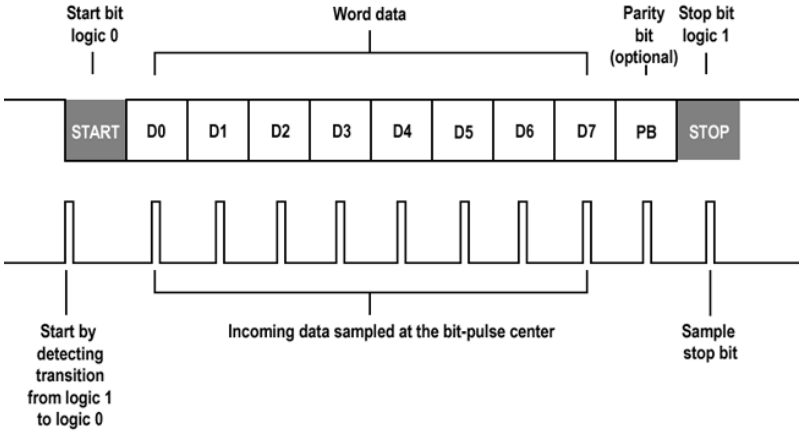

Figura 4.10: Ejemplo de paquete de bits

4.2.2. Bus SPI

El bus SPI (Serial Peripheral Interface) es un estándar de comunicación entre circuitos integrados, mas concretamente entre microcontroladores y periféricos.

Se compone de una bus con dos roles de maestro y «n» esclavos. Se compone de 3 lineas principales: una linea de frecuencia de reloj marcada por el maestro, una linea dedicada para la comunicación de maestro a esclavo y otra a la inversa del esclavo al maestro. A estas 3 lineas, se suman las «n» lineas de selección, siendo una linea de selección por cada esclavo conectado al maestro, con la intención de ser esta linea la que marca que esclavo tiene permitido comunicarse con el maestro.

La mayor ventaja es que es una comunicación «full duplex» y permite una mayor velocidad de comunicación con respecto al I2C. Además de no estar limitado a transferencias de bloques de 8 bits. Por tanto, es la elección empleada con el sensor inercial para este proyecto.

4.2.3. Modulación por ancho de pulso

La modulación por ancho de pulsos (PWM) es la modulación de una señal analógica por el ancho de un pulso. Esta técnica se empleaba para generar señal analógica mediante el uso de una salida digital.

Esta técnica ha cambiado en el uso de los drones, desde la aparición de los servos, el PWM se ha utilizado para el control de la posición de los servomotores a un frecuencia de 50 Hz, siendo 1 milisegundos el ancho de pulso mínimo y 2 milisegundos en ancho de pulso máximo.

La evolución hacia los controladores de motor de los drones ha sido emplear esta técnica para controlar la tensión transmitida a los motores, siendo proporcional al tamaño del pulso, midiendo la duración del pulso.

Las ventajas de medir la duración del ancho de pulso son que la señal pasa a ser una codificación digital de la señal de actuación siendo menos vulnerable

al ruido y teniendo la ventaja de poder aumentar la frecuencia a la cual enviamos ordenes al controlador, como podemos observar en la siguiente figura **??**. Conservando el tamaño mínimo y máximo de los anchos de pulso y variando la frecuencia en función de la velocidad de respuesta necesaria.

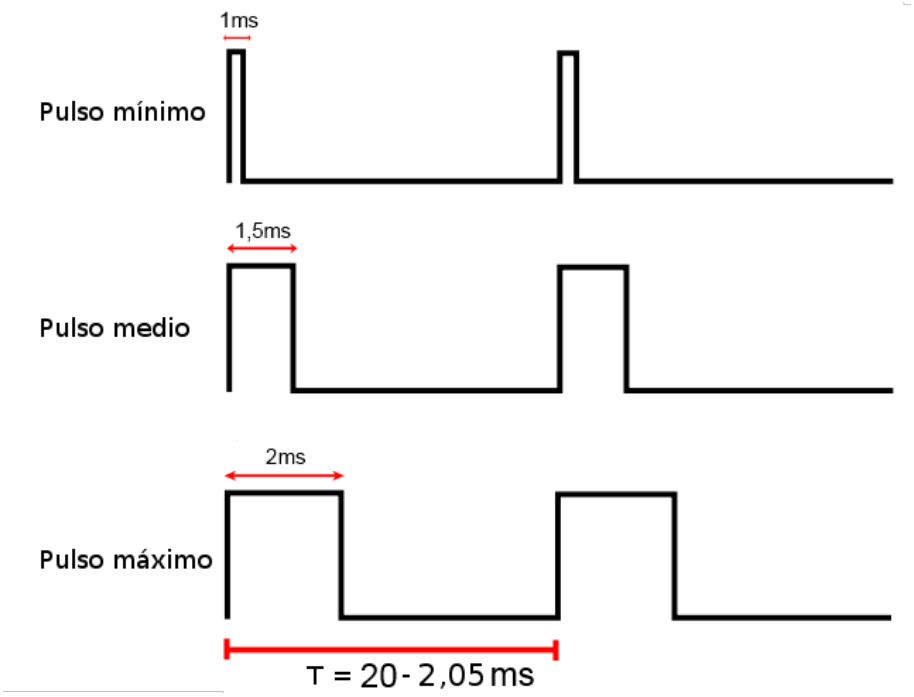

Figura 4.11: PWM de 50 Hz a 490 Hz

4.3 Tiempo Real

Esta sección corresponde a un análisis de planificabilidad de tiempo real del sistema completo. Para este cometido hemos recogido estadísticas del coste de cada tarea, unas 10000 iteraciones para su estimación. En este sistema tenemos tres tareas definidas: la tarea de control, la tarea de envió y la tarea de recepción.

El sistema se compone de dos partes, una fase de inicialización y una fase de bucle infinito. La fase de inicialización tiene un coste de 65 milisegundos, ya que la calibración del sensor inercial se tiene que realizar siempre, este tiempo forma parte de la etapa inicial, y por tanto, no influye en este análisis.

Por otra, para considerar que la fase de bucle infinito se comporta como un sistema basado en tareas hemos realizado la siguientes pasos. Cada tarea tiene una condición de disparo asociada, es decir, un timer asociado a un interrupción o un evento genera la activación de la tarea, y la segunda condición es que en caso de activarse algunas casi simultáneamente, estas tienen una prioridad entre estas, siendo más prioritaria la de menor período. Además, se da la condición de que no se pueden producir interrupciones anidadas. Por último, no tiene un paso de datos muy robusto, ya que se realiza por variables compartidas globales y protegidas frente a la concurrencia.

La granularidad de la medición del sistema es 1 milisegundo. Como ya hemos dicho antes, la tarea más prioritaria es la de menor período, y en caso de tener el

mismo período, es elegido por el diseñador, en este caso, el criterio que se ha seguido es el siguiente para las dos tareas con el período de 100 milisegundos.

Como la tarea de enviar es periódica a 100 milisegundos, esta tiene mayor prioridad, ya que la tarea de recibir es una tarea aperiódica, con el mínimo período de 100 milisegundos, ya que es el tiempo mínimo de recepción es de 100 milisegundos. El WCET es el «worst case execution time» que es el peor y mayor tiempo de ejecución de la tarea.

Task	WCET	Período	Prioridad
Control	1	10	1
Send	1	100	2
Receive	1	100	3

El análisis de planificabilidad se puede mostrar observar es este cronograma. El hiper-período de todo el conjunto de tareas es 100 milisegundos, aunque viendo como referencia la figura 4.13, se puede observar que con simular 10 milisegundos, el sistema es planificable, ya que las 3 tareas se han ejecutado dentro del hiper-período.

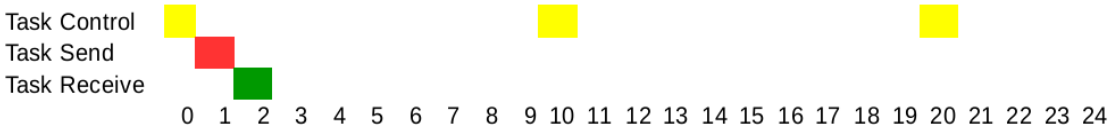

Figura 4.12: Cronograma

Podemos verificar la afirmación, si entendemos como que este sistema se aproxima gratamente a un sistema con un planificador Rate Monotonic y aplicamos el siguiente calculo:

$$U = \sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{\frac{1}{n}} - 1) \tag{4.1}$$

donde Ci es el WCET, Ti es el período y n es el número de tareas.

E incluso hay un método más sencillo:

$$U = \lim_{n \to \infty} n(\sqrt[n]{2} - 1) = ln2 \approx 0{,}693147 \tag{4.2}$$

Y como resultado obtenemos que la utilización del sistema es $U = 0{,}12$ y por tanto, siendo planificable según la política de plantificación de Rate Monotonic.

4.4 Control

Muchos de los sistemas actuales como coches, centrales eléctricas, motores, etc, emplean mecanismos para controlar estos sistemas. Hay diversos mecanismos de control, uno muy empleado es el controlador PID, sus siglas hacen referencia a que hay tres términos que se tienen en cuenta como son los términos proporcional, integral y derivativo.

En nuestro caso, si intentáramos controlar un drone para mantenerlo nivelado en la horizontal controlando nosotros mismos visualmente los motores, lo mas probable es que no lleguemos a levantar el drone sin causar un accidente, y ahí esta la importancia de emplear un controlador.

Entonces podemos aplicar un o varios controladores PID para mantener nivelado en la horizontal un drone o mantenerlo a cierta altitud o ambos juntos. Durante un vuelo, es posible que el drone sufra rachas de ciento, por tanto, genera que el drone tenga un ángulo distinto al ángulo cero en la horizontal.

En este punto, el controlador se encarga de medir la diferencia y en consecuencia calcular en base a esta diferencia y aplicar una acción a los motores para compensar el error de orientación del drone. También ocurre incluso si nosotros variamos la referencia siendo esta distinta de la horizontal, por ejemplo, un mando de radio control con el cual controlamos el ángulo, el drone junto con el controlador intentara tener el ángulo deseado.

Otra situación seria el control de la posición junto con un sensor GPS o el control de un motor con un encoder. Los controladores PID se emplean en muchas áreas del control automático, la mecatrónica y los procesos industriales.

En resumen, el controlador PID tiene una referencia dada que compara con el valor real, la diferencia entre la referencia y el valor real se llama error. Y aplicando los términos proporcional, integral y derivativo con respecto al error, tenemos en consecuencia una acción de control a aplicar a los actuadores.

4.4.1. Termino proporcional

El término proporcional produce un valor de salida que es proporcional al valor de error actual. La respuesta proporcional se puede ajustar multiplicando el error por una constante Kp, llamada la constante de ganancia proporcional.

El término proporcional viene dado por:

$P = K_p e(t)$

Una alta ganancia proporcional resulta en un gran cambio en la salida para un cambio dado en el error. Si la ganancia proporcional es demasiado alta, el sistema puede volverse inestable. En contraste, una pequeña ganancia resulta en una pequeña respuesta de salida a un gran error de entrada, y un controlador menos sensible o menos sensible. Si la ganancia proporcional es demasiado baja, la acción de control puede ser demasiado pequeña cuando se responde a perturbaciones del sistema.

Error de estado estacionario

El error de estado estacionario es la diferencia entre la salida final deseada y la real. Debido a que se requiere un error distinto de cero para impulsarla, un controlador proporcional generalmente opera con un error de estado estacionario. El error de estado estacionario (SSE) es proporcional a la ganancia de proceso e inversamente proporcional a la ganancia proporcional.

4.4.2. Termino integral

La contribución del término integral es proporcional tanto a la magnitud del error como a su duración. La integral en un controlador PID es la suma del error instantáneo en el tiempo y da el offset acumulado que debería haber sido corregido previamente. El error acumulado se multiplica por la ganancia integral (Ki) y se añade a la salida del controlador.

El término integral viene dado por:

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

El término integral acelera el movimiento del proceso hacia el punto de ajuste y elimina el error residual de estado estacionario que ocurre con un controlador proporcional puro. Sin embargo, dado que el término integral responde a errores acumulados del pasado, puede causar que el valor actual supere el valor del punto de ajuste.

4.4.3. Termino derivativo

La derivada del error de proceso se calcula determinando la pendiente del error a lo largo del tiempo y multiplicando esta tasa de cambio por la ganancia derivada Kd. La magnitud de la contribución del término derivado a la acción de control global se denomina ganancia derivada, Kd.

El término derivado viene dado por:

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

La acción derivada predice el comportamiento del sistema y por lo tanto mejora el tiempo de asentamiento y la estabilidad del sistema. Un derivado ideal no es causal, por lo que las implementaciones de los controladores PID incluyen un filtro adicional de paso bajo para el término derivado para limitar la ganancia y el ruido de alta frecuencia.

4.4.4. Proceso de control

El controlador PID se llama en general al conjunto de controladores que usan los términos explicados anteriormente. De hecho, si alguna de las contantes de Ki o Kd o ambas las hacemos nulas según nuestras necesidades, obtenemos las siguiente combinaciones:

- Control P
- Control PI
- Control PD
- Control PID

Y podemos encontrar más variaciones con alguna modificación de como actúan los términos, a esta serie de combinaciones, al grupo en si se les llama controladores PID. En este proyecto, emplearemos un control PD porque el proceso

tiene el controlador P actuara para corregir el error, mientras que la D actuara para eliminar el error de estado estacionario generado por el controlador P.

Con respecto al ajuste de parámetros en la simulación se puede ajustar manualmente sin correr demasiado peligro, en cambio en la implementación real hemos tenido ayuda de un operador experto para su ajuste, Vicente Balaguer Garín, a fecha del trabajo, doctorando de la UPV sobre aspectos de control más avanzados en drones.

Por lo que se refiere a la estructura de control del drone, hemos aplicado el modelo matemática del proceso explicado en capítulo 3 y el control PD, tenemos tres sistemas SISO linealizados de control de la orientación en el eje XYZ del drone. Al linealizar el modelo, encontramos que en el Roll y Pitch tenemos unos ángulos de control entre -20 y 20 grados, mientras que en el Yaw nuestra referencia dada sera a 0. A este tipo de control del drone se le llama control de actitud. Mientras que no disponemos control de altitud con respecto al plano del suelo y por tanto este lo realiza el operador humano, a este acción la llamaremos empuje.

Como resultado quedaría el siguiente esquema de control:

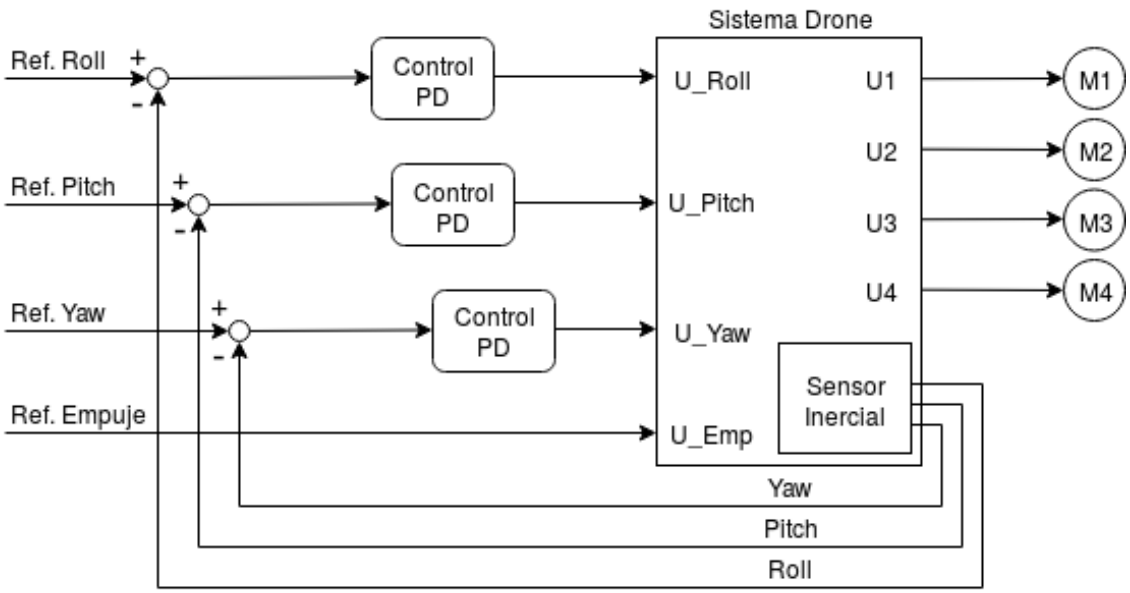

Figura 4.13: Esquema de control

4.5 Fusión sensorial

El algoritmo de Madgwick [5] es un algoritmo de fusión de los principales sensores que componen un sensor inercial. Este se basa en la estimación de los ángulos a partir de la velocidad angular estimada de los sensores.

A grandes rasgos el algoritmo funciona de la siguiente forma, se estima la velocidad angular a partir del giroscopio a través de una serie de estados. Estos estados son realimentados con la salida anterior de los ángulos, que inicialmente es nula.

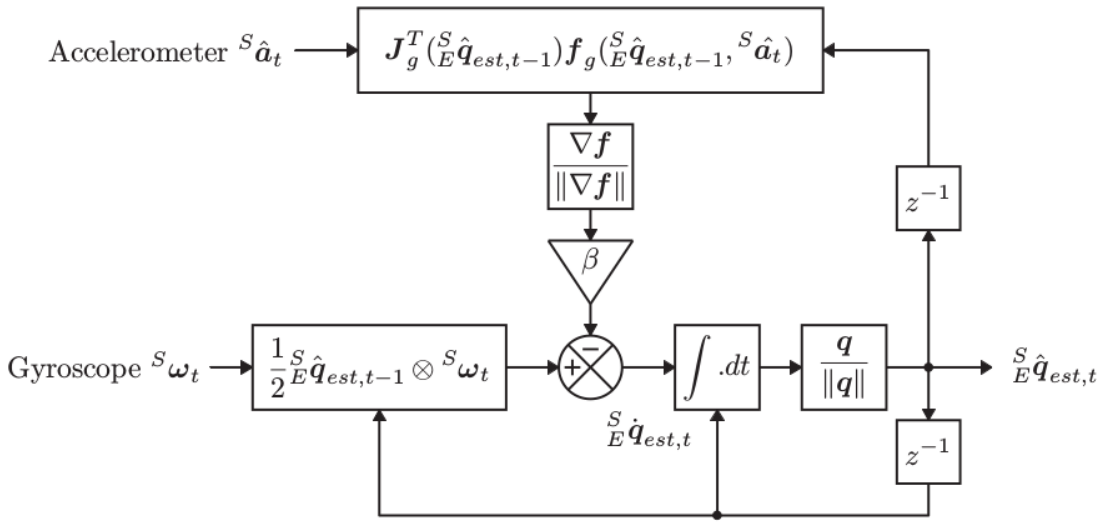

Figura 4.14: Filtro de Madgwick

Por otro lado, también se estima la deriva del giroscopio estimando la velocidad angular a partir del acelerómetro. Este también se obtiene a partir de una serie de estados realimentados por la salida de los ángulos.

A partir de estos valores, el valor del giroscopio estimado es compensado con el valor del acelerómetro eliminando la deriva de la medida. Como resultado obtenemos el valor filtrado y depurados de la velocidad angular. A este le aplicamos la integral y como resultado obtenemos la posición angular. Este valor acaba por realimentar la estimación de estados del giroscopio y acelerómetro por separado.

Este filtro es tan potente por su bajo coste computacional frente a un filtro de Kalman, aunque sacrificando en prestaciones frente a este. Este filtro también dispone de una versión más completa y ampliada para la estimación con respecto al norte magnético terrestre.

CAPÍTULO 5
Simulación

Por lo que se refiere a la simulación se presenta como una herramienta idónea para ensayar, conocer el funcionamiento de determinados sistemas o anticiparse a problemas. Estos sistemas de simulación facilitan conocer qué tipo de respuestas se pueden ofrecer ante determinadas situaciones, sin ningún tipo de riesgo físico ni para los humanos ni para las máquinas.

El primer paso a llevar a cabo es el modelado y ensamblaje de los elementos para tener una modelo del cuadricóptero. Para ello, emplearemos herramientas como Simulink junto con modelo 3D de la piezas similares al cuadricóptero con la intención de ensamblarlas y obtener un modelo matemático semejante con la realidad.

El objetivo final es poder realizar distintas pruebas observando el comportamiento final ya que una buena simulación nos permite entender mejor el comportamiento del sistema y en consecuencia abordar con una mejor estrategia la resolución del problema.

5.1 Desarrollo del modelo en Simulink

Para esta sección emplearemos la herramienta de Simulink junto con un librería adicional «Simscape Multibody», que nos permiten realizar un entorno de simulación adecuado con cierta comodidad con diagramas de bloques, además de poder modelar los sistemas con sistemas mecánicos en 3D introduciendo los elementos, las uniones, restricciones, elementos de fuerza y sensores.

Esta funcionalidad nos la proporciona importando los ensamblajes CAD completos, junto con sus masas para poder generar un modelado matemático preciso junto a una animación 3D generada automáticamente permitiendo visualizar el sistema de forma dinámica.

5.1.1. Ensamblado y Diseño CAD

Las partes del cuadricóptero en 3D fueron obtenidas desde el repositorio de librerías de piezas GRABCAD, para tener un punto de partida para realizar el diseño. Con las partes obtenidas en formato STL, las introducimos al programa simulink, como podéis observar en la figura 5.1. En esta figura podemos observar

las uniones de las piezas por medio de bloques, las articulaciones de revolución, las fuerzas generadas por los motores unidos a la hélices junto con algunos plotters para observar su correcto funcionamiento.

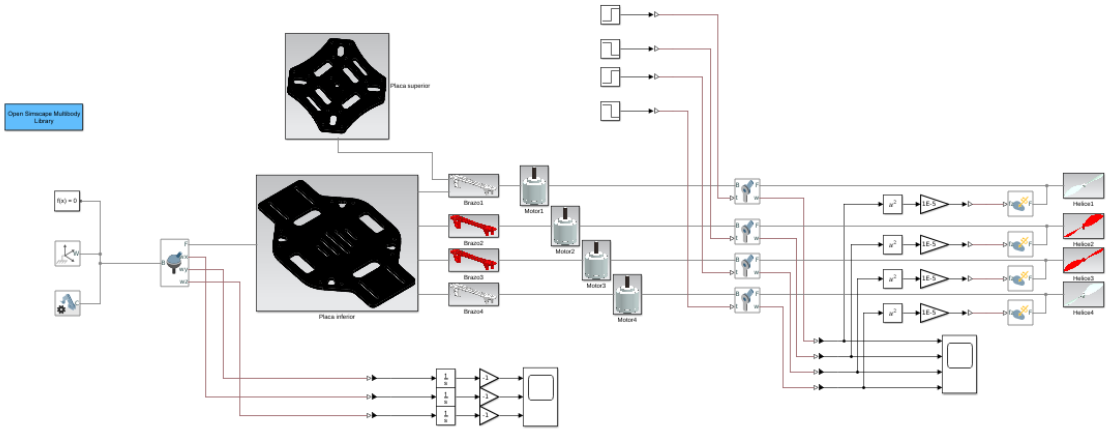

Figura 5.1: Modelado del drone

Como resultado obtenemos el siguiente modelo en 3D de la figura 5.3.

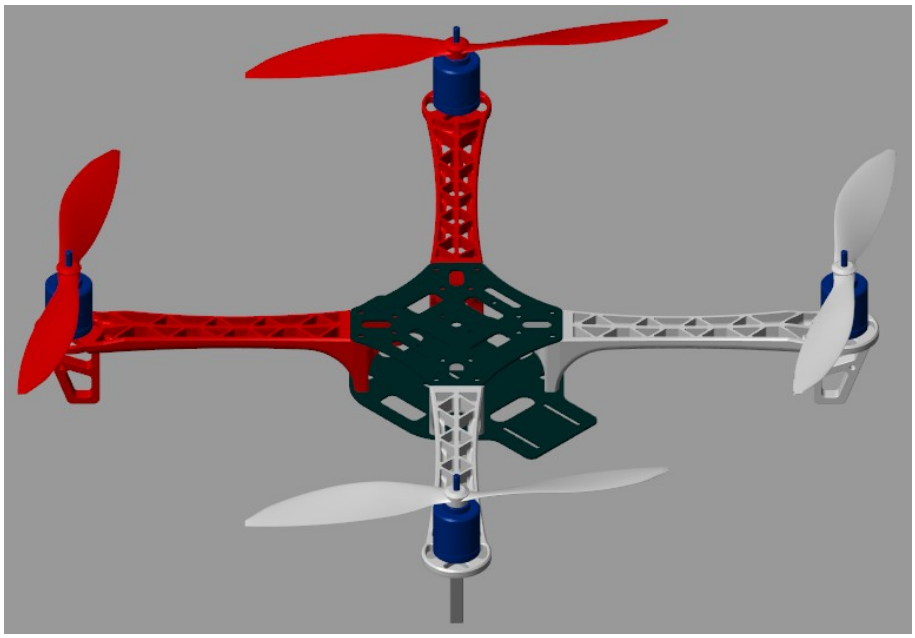

Figura 5.2: Ensamblado final

Una vez tenemos implementado el modelo con el paquete de Simscape Multibody lo incluiremos como parte de otro elemento en Simulink y desarrollaremos el sistema de control que se encargara de controlar y regular el comportamiento del sistema.

Los resultados de la simulación lo visualizaremos a través de gráficas que nos muestran la respuesta en función del tiempo de las variables de interés además de una animación 3D que es generada automáticamente para visualizar la dinámica del sistema.

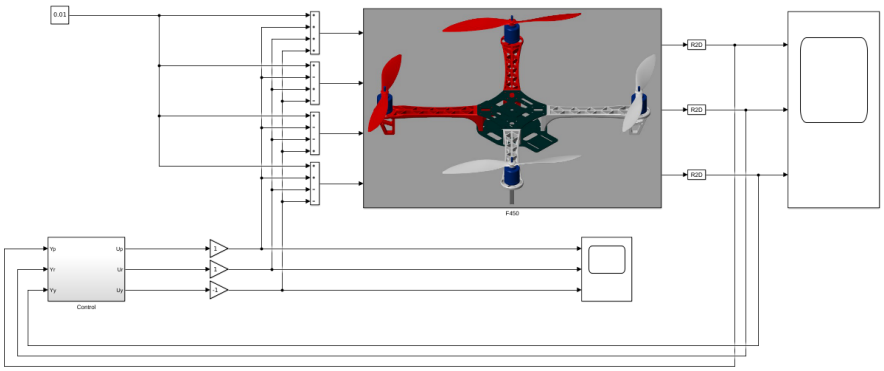

Figura 5.3: Sistema de simulación final ensamblado

Una vista más completa del modelado en Simulink donde se muestran todas las piezas sólidas y las uniones entre estas y las piezas importadas además de incluir las restricciones necesarias para el correcto funcionamiento, cabe destacar que cada solido tiene sus correspondientes características como su masa, densidad, geometría, etc.

Entre las variables de interés que nos interesa analizar y ver su evolución en el tiempo, tenemos los ángulos de «pitch», «roll» y «yaw» (cabeceo, alabeo y guiñada), que con estas variables realizamos el lazo cerrado de control.

5.2 Sistema de control

El sistema de control se basa en un controlador PD como ya hemos explicado anteriormente. La implementación llevaba a cabo en un controlador PD de tipo digital con una realimentación de la orientación de su respectivo eje y con una prealimentación para la acción derivativa para evitar los valores de infinito al producirse un escalón en la variable de referencia así como una discretización de la posición para la obtención de la velocidad angular.

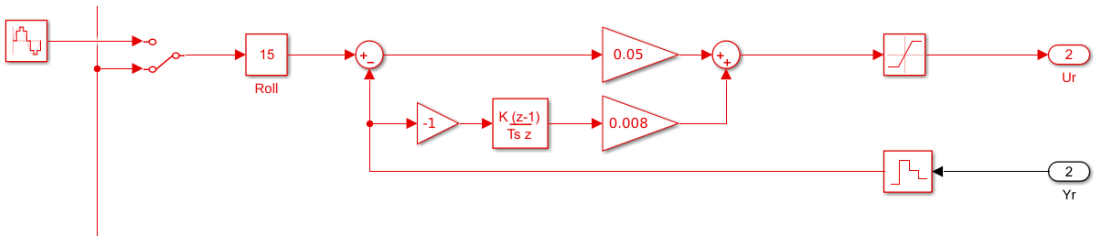

Figura 5.4: Control PD

En este punto, podemos ver que la figura siguiente cambia de color con respecto a la figura anterior. Es consecuencia de la introducción del control bifrecuencia, que se basa en reducir la frecuencia de muestreo del sensor «n» veces proporcional al tiempo de control del sistema con la intención de reducir el coste energético o a las latencias de comunicación.

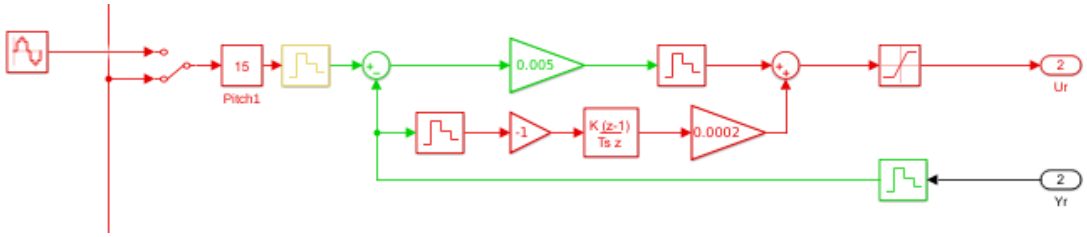

Figura 5.5: Control PD bifrecuencia

Por último, comentar que el control a realizar lo realizaremos únicamente sobre el eje «roll», ya que el control sobre el eje «pitch» al ser un drone simétrico en «X» los parámetros son prácticamente los mismos para el controlador del otro eje. En cambio, el control del «yaw» resulta algo más complejo debido a la discontinuidad del sensor inercial, además que este control comercialmente se suele realizar sobre una referencia de velocidad angular de giro si se controla por un operador humano.

5.3 Referencias y Resultados

En este punto, empezaremos con un controlador PD monofrecuencia, en este probaremos una entrada de referencia determinada y evaluaremos los resultados hasta encontrar unos parámetros de Kp y Kd, que se ajusten al control con el período de T = 10 milisegundos. Siendo Kp = 0.005 y Kd = 0.0005. Para este apartado de sintonización incluiremos las siguientes figuras 5.6 y 5.7.

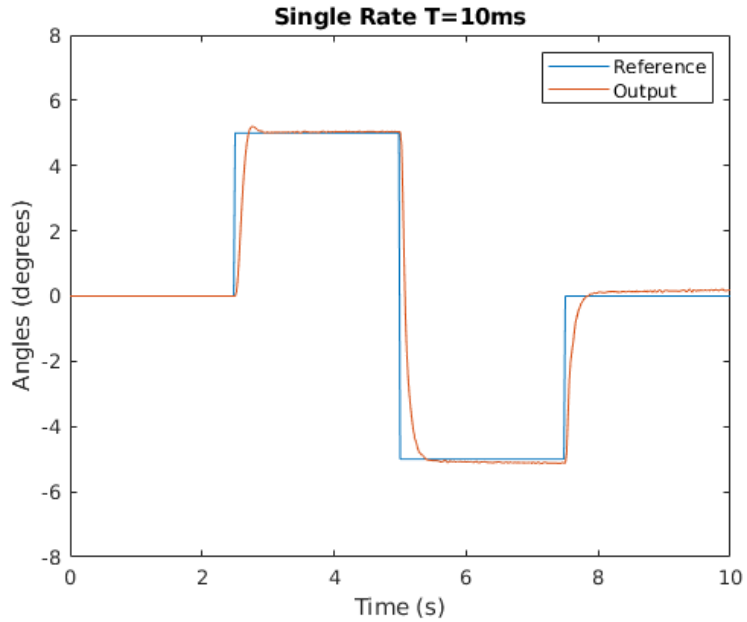

Figura 5.6: Referencia cuadrada entre +/- 5 grados

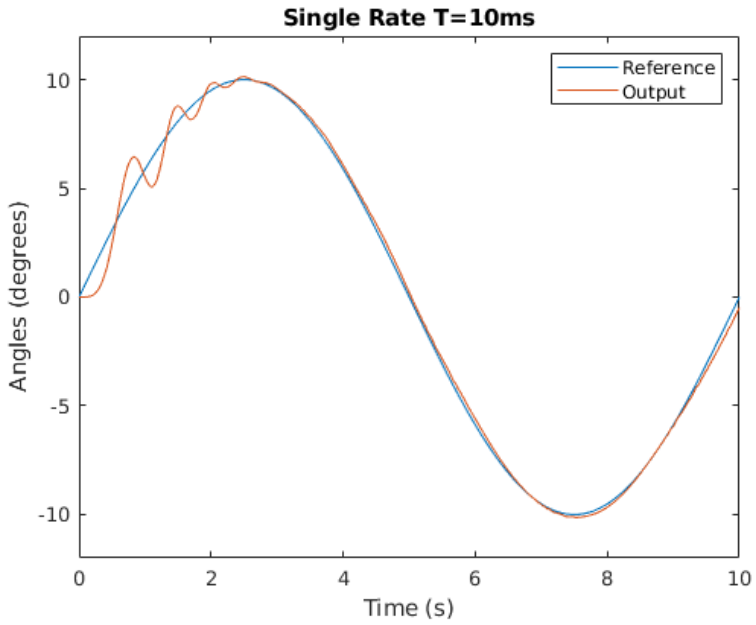

Figura 5.7: Referencia sinusoidal entre +/- 10 grados

Los siguientes pasos son probar el limite del control en bifrecuencia y resintonizar los parámetros para mejorar la bifrecuencia del control. Primero probaremos con la bifrecuencia con n=2, pasando a tener un control inestable como podemos ver en la figura 5.8.

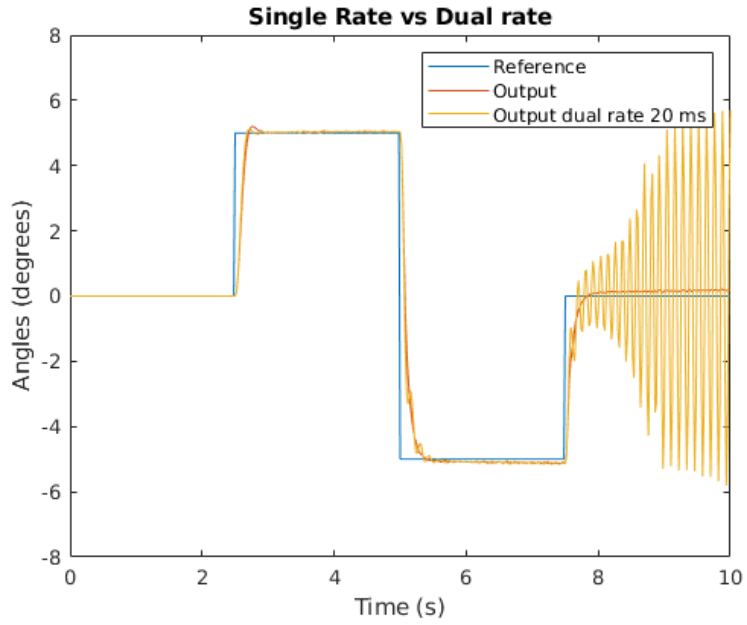

Figura 5.8: Referencia cuadrada entre +/- 5 grados - antes de la resintonización

Para solventar esta situación, realizamos una resintonización de los parámetros con el cual obtenemos un control estable en n = 2 con un período de muestreo del sensor de posición de T = 20 milisegundos, siendo Kp = 0.005 y Kd = 0.00025, como podemos observar en la figura 5.9.

Como consecuencia de la resintonización, el sistema de control PD gana en sobreoscilación y lentitud, pero en consecuencia ganamos en robustez, pudiendo subir el tiempo de muestreo del control bifrecuencia, aliviando carga computacional en el microcontrolador.

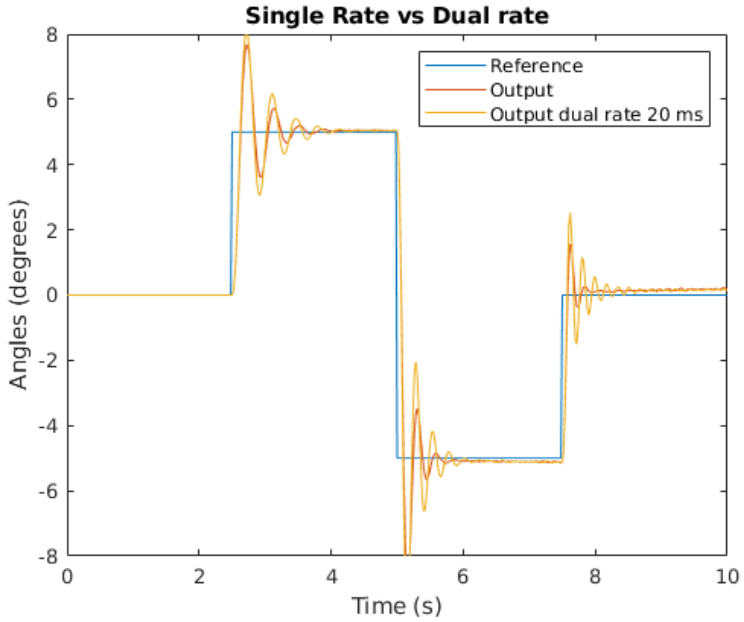

Figura 5.9: Referencia cuadrada entre +/- 5 grados - después de la resintonización

Por otra parte, sabemos que el limite de la bifrecuencia se estable en n=3, porque al retocar los parámetros de control, intentando resintonizarlos, no obtenemos un buen resultado. La siguiente figura 5.10 se muestra con los términos Kp y Kd establecidos anteriormente. Hemos introducido una variabilidad sobre estos valores y no hemos encontrado ningún resultado satisfactorio.

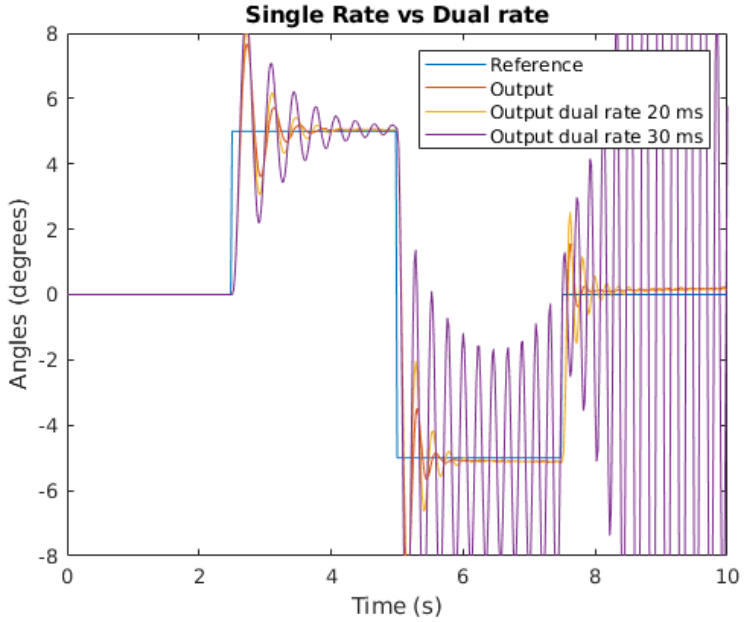

Figura 5.10: Limite de la bifrecuencia en n = 3

CAPÍTULO 6
Implementación

Para la implementación real hemos acondicionado el cuadricóptero para tener ciertas medidas de seguridad. Estas medidas consisten en una plataforma de hierro unida a una rotula de cámara para permitirle el movimiento en los tres ejes, pero que no salga volando, por otra parte hemos puesto un palo y en el extremo una pelota de ping pong para poder coger el «drone» por la parte superior y así poder realizar el ajuste de parámetros del control con seguridad.

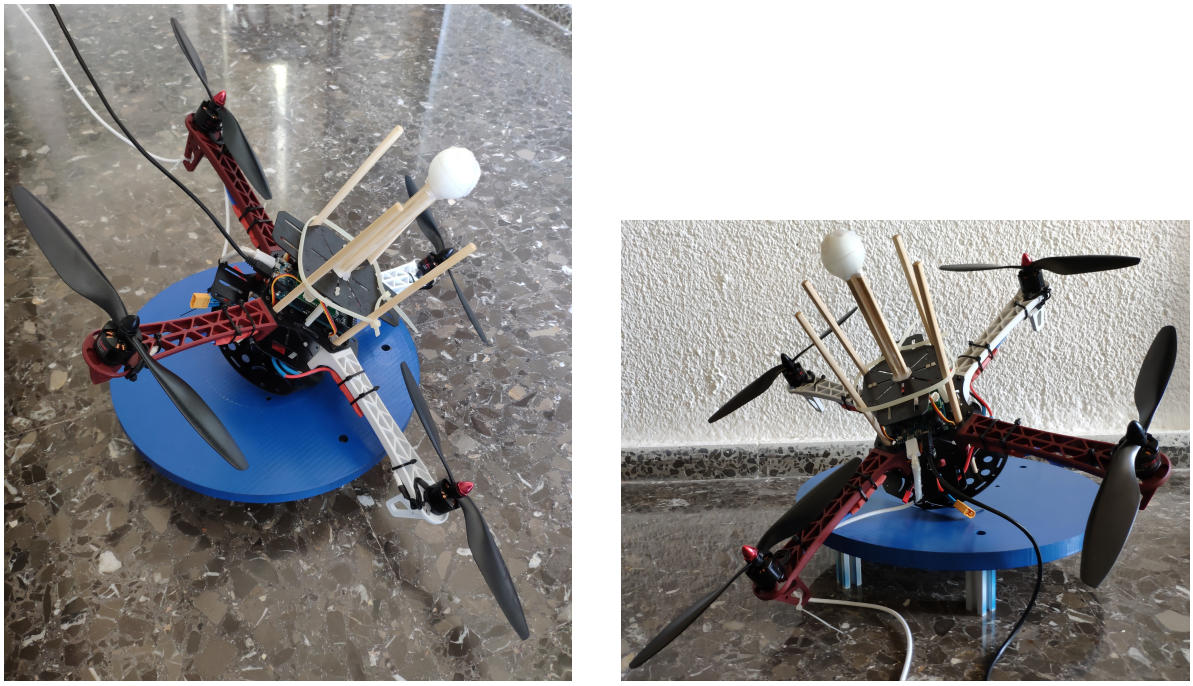

Figura 6.1: Implementación real del cuadricóptero

6.1 Programación y comunicación

Esta sección es complementaria a la sección 4.3, en esta ya se explican detalles técnicos de la programación. En esta ampliaremos con aspectos de comunicación y los aspectos de programación triviales. Y para más detalles de implementación ver el anexo A.

Como decíamos en el sección de tiempo real, «El sistema se compone de dos partes, una fase de inicialización y una fase de bucle infinito. La fase de inicialización tiene un coste de 65 milisegundos, ya que la calibración del sensor inercial se tiene que realizar siempre, este tiempo forma parte de la etapa inicial...». Consideramos ahora la fase bucle infinito, se compone de tres tareas: la tarea de control, la tarea de envió y la tarea de recepción de mensajes.

Las tareas de control y la tarea de envió que se activan respectivamente cada 10 y 100 milisegundos. Estas son disparadas mediante un timer periódico que hace ejecutarse una rutina de interrupción. Esta rutina de interrupción tiene únicamente una asignación a una variable, actuando de flag en el bucle principal y disparando la ejecución de la respectiva tarea. La asignación a esta variable es atómica, debido al tipo de variable.

Por otra, parte tenemos la tarea de recepción que una tarea en segundo plano, cuando esta comprueba que alguna de las dos tareas no esta en ejecución, comprueba si hay algún mensaje. Cuando termina, se duerme hasta el siguiente ciclo de 100 milisegundos. Aunque, también se activa por la interrupción del «buffer», si este recibe algún contenido, se activa asíncronamente.

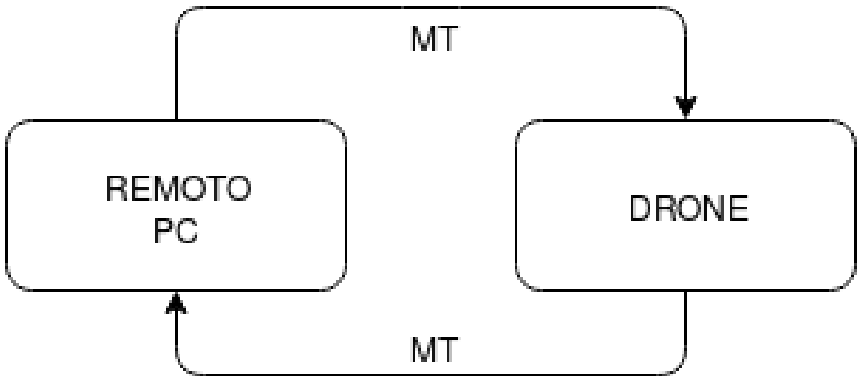

Figura 6.2: Concepto de remoto

Las características de la tarea de recepción van acorde con el ordenador remoto, ya que esta también se basa en las mismas características, envía las nuevas referencias y recibe los datos cada 100 milisegundos. Asíncronamente, puede enviar los parámetros a los controladores, incluso enviar un mensaje de armado o desarmado al drone por aspecto de seguridad.

Como decíamos, el período de comunicación entre el drone y el ordenador se realiza cada 100 milisegundos, siendo MT el período mostrado en la figura 6.2. Este factor también es importante, ya que el drone, si pierde más de 5 paquetes consecutivos, es decir que durante 500 milisegundos, no recibe ningún paquete, entiende que hay algún problema y se apaga completamente, es decir, vuelve al estado de desarmado.

Por último, destacar que como el bus de comunicación es serie a través de bluetooth la pruebas se pueden realizar tanto por cable como bluetooth. Las primeras se realizaron por cable para verificar el buen funcionamiento del código y posteriormente, se realizaron con bluetooth para los experimentos presentados.

6.2 Pruebas y Resultados

6.2.1. Control en monofrecuencia

El siguiente apartado corresponde a una serie de pruebas realizas en el laboratorio las cuales las realizamos en monofrecuencia para verificar que el control. Los parámetros que hemos empleado han sido diversos hasta encontrar manualmente los apropiados, con una Kp = 0.01 y una Kd = 0.004. Los test son diversos desde la simple prueba de mantener el drone estable durante al menos 2 minutos hasta una referencia sinusoidal con cambio en la amplitud, todo esta para verificar la robustez del control.

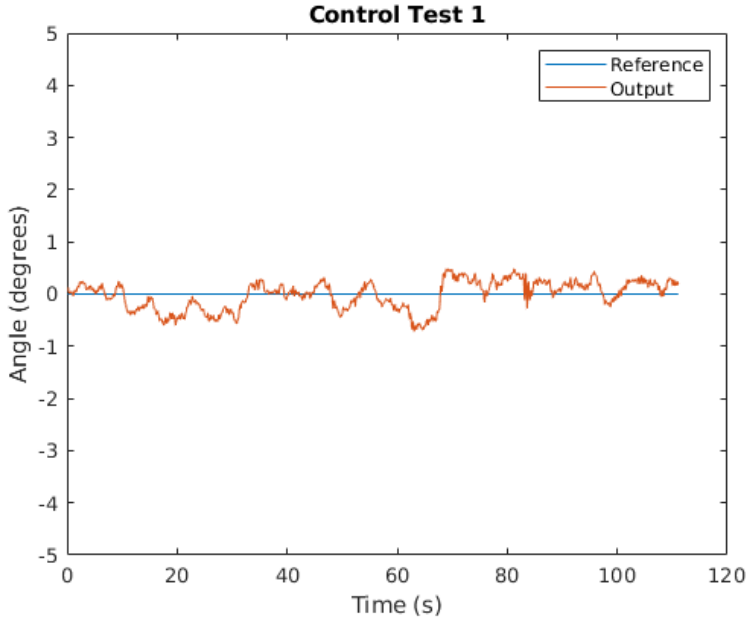

Figura 6.3: Test de referencia a 0 grados

Las perturbaciones introducidas las realizamos manualmente, realizando una serie de golpes a la pelota de ping pong situada en la parte superior del cuadricóptero.

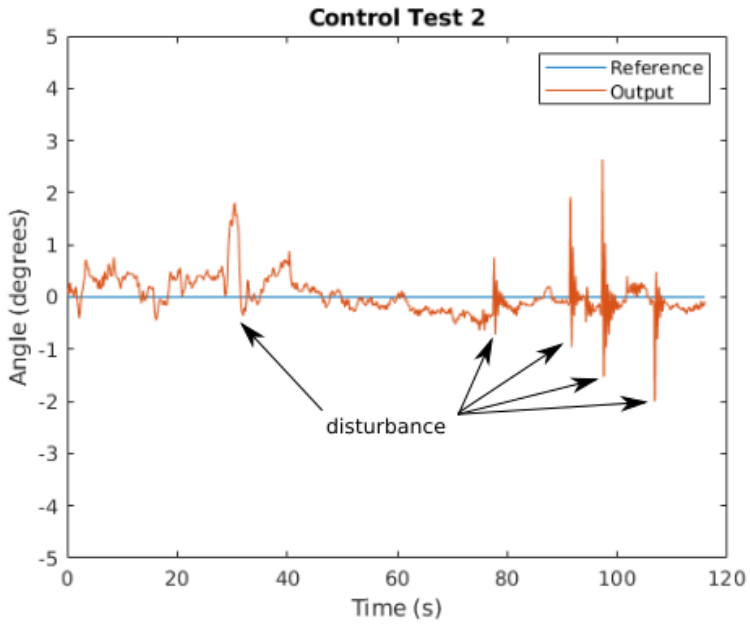

Figura 6.4: Test de referencia a 0 grados con perturbaciones

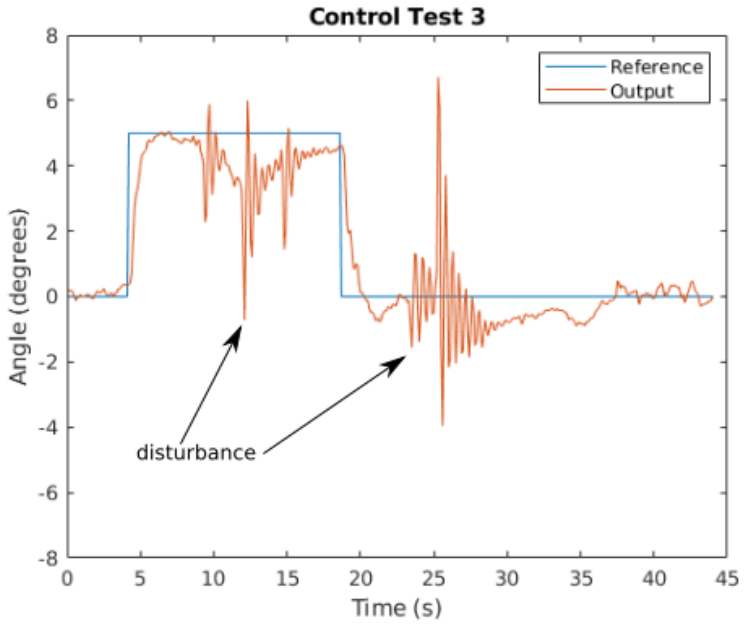

Figura 6.5: Test de referencia escalón de 5 grados con perturbaciones

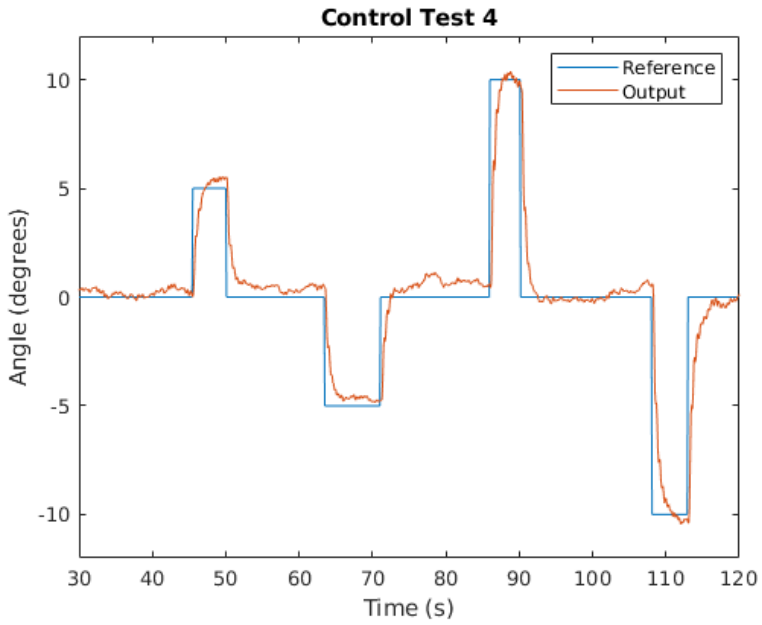

Figura 6.6: Test 1 con entradas escalón variadas

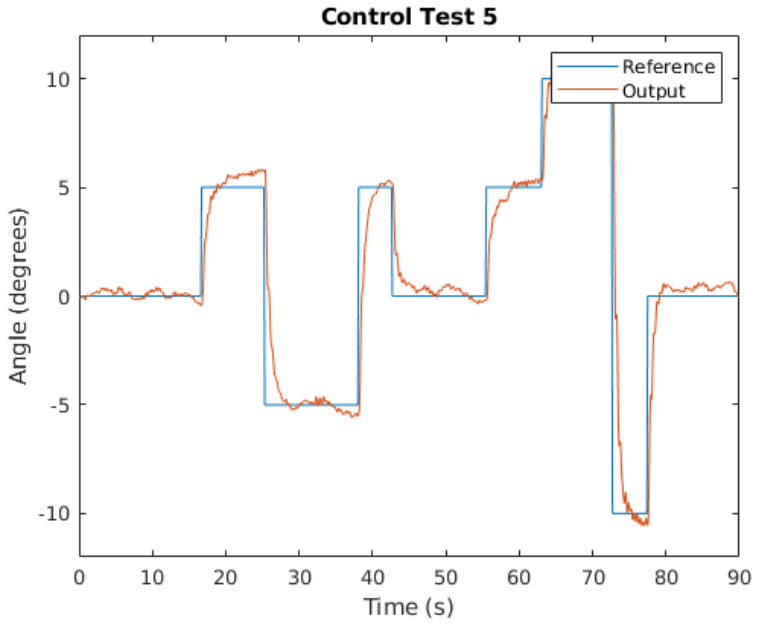

Figura 6.7: Test 2 con entradas escalón variadas

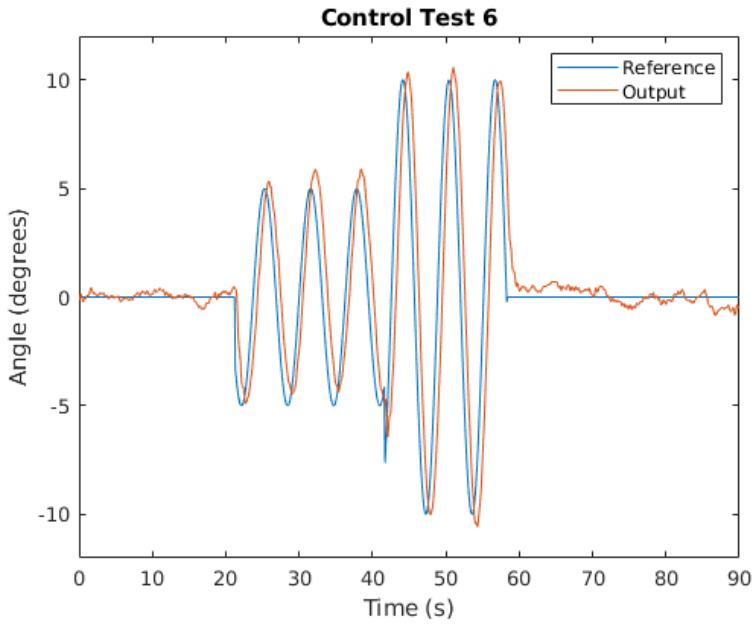

Figura 6.8: Test con referencia sinusoidal con cambio de amplitud

Como podemos ver en la gráficas anteriores de 6.3 a la 6.8, esta serie de test valida el control PD en robustez frente a cambios de referencias de varios tipos y perturbaciones, además de poder observar que la señal de salida no es totalmente limpia ya que el sensor inercial contiene ruido, y junto con la inestabilidad del sistema, el control se vuelve algo más complejo frente al sistema simulado. Esta serie de test sirve para validar el sistema de control en monofrecuencia, en la siguiente sección pasaremos a realizar pruebas de funcionamiento en bifrecuencia.

6.2.2. Control en bifrecuencia

A la hora de realizar el control, debemos tener en cuenta los aspectos de seguridad como es cortar la energía del sistema de forma voluntaria o cortar la comunicación para que el sistema se caiga, obviamente este sistema ante errores se programa para el laboratorio por seguridad, en un entorno de vuelo comercial, esta plataforma debería estar más que probada, y este no es el caso.

La razón por la cual introduzco esta serie de comentarios sobre la seguridad es consecuencia del comportamiento del drone, como podemos observar en la gráficas el drone tiende a oscilar, cosa que puede volverse peligroso para el operador humano por la agresividad de los movimientos, aunque se llevaba guantes de seguridad anticorte, no hay que olvidar que los motores con las hélices a altas velocidades, como es el caso, se convierten en cuchillas y pueden realizar cortes.

Dicho esto, para el estudio del control con bifrecuencia hemos empleado los mismo parámetros para el controlador, y hemos usado las mismas entradas tanto para la monofrecuencia, como para la bifrecuencia con n=2, pasando el tiempo de muestreo a 20 milisegundos en bifrecuencia. En la siguiente gráfica 6.9, podemos observar que es un sistema lento en estabilizarse y si llega régimen permanente.

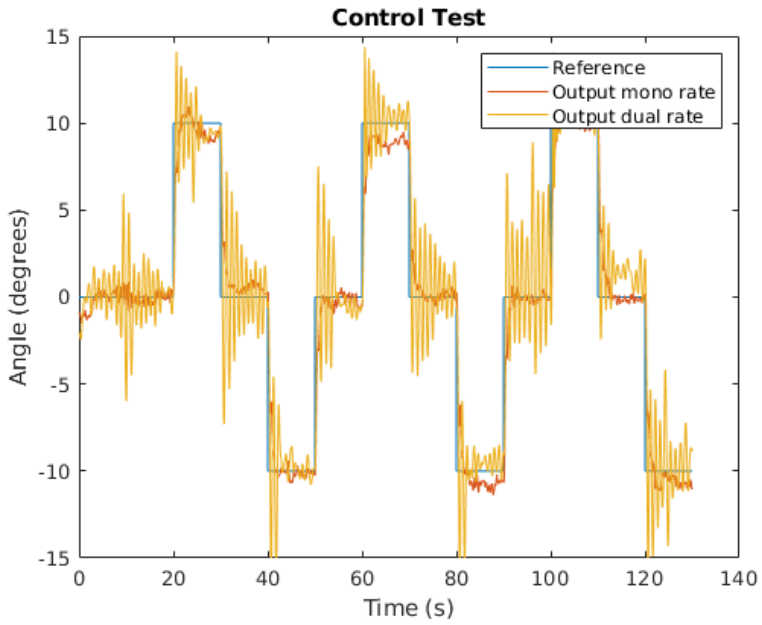

Figura 6.9: Test comparando la bifrecuencia con la monofrecuencia

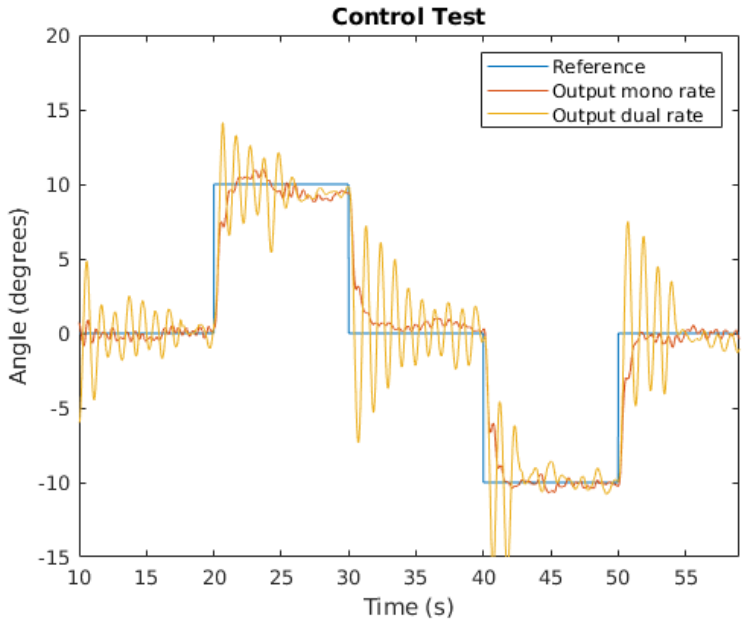

Figura 6.10: Test comparando la bifrecuencia con la monofrecuencia con zoom

Llegados a este punto, viendo que los resultados no son concluyentes, no seguimos con la investigación con bifrecuencia para el control en orientación a causa del tiempo necesario para obtener un mejor control frente al tiempo disponible para realizar el trabajo.

CAPÍTULO 7
Conclusiones

En cuanto al resultado del trabajo hemos cumplido objetivos parcialmente, ya que por una parte hemos sido capaces de llegar a realizar todo el montaje y diseño del drone, tanto a nivel hardware como a nivel software. El control monofrecuencia se ha completado sobre un rotula, ya que sí que era capaz de sostenerse sobre un rotula y cumplir con la referencia del ángulo. Así mismo, es capaz de hacer frente a la perturbaciones introducidas y mantener el control en orientación.

En cambio, los primeros resultados de la investigación del control en bifrecuencia no han sido totalmente satisfactorios y hay que proceder a una resintonización de los parámetros y a un nuevo diseño del control.

Otro aspecto a tener en cuenta es la idealidad del sistema modelado en la simulación. Hay que tener en cuenta que en la simulación, la orientación del sensor es prácticamente perfecta, mientras que en la realidad el sensor inercial se caracteriza por tener bastante ruido en su medición.

En consecuencia provoca que el sensor introduzca una serie de ruido en el sistema de control que en la reducción de la frecuencia de control tiene como consecuencia una perdida de robustez considerable, como hemos observado en la bifrecuencia.

Llegados a este punto, cabe destacar que el campo del control bifrecuencia en drones en el control en orientación es un campo no explorado y que por tanto, en un principio puede que la investigación lleve a un resultado no del todo satisfactorio.

Otro aspecto que apenas tiene una observación directa y/o es difícil de proyectar en un trabajo es la labor de la programación de código, ya que la planificación y programación de este lleva bastante tiempo.

Por último, destacar que estoy satisfecho por el trabajo realizado, que me resulta apasionante la área de control aplicada al campo de la robótica en general junto la aplicación de la informática a este campo.

Bibliografía

[1] Julián J. Salt Llobregat, Angel Cuenca Lacruz, Vicente Casanova Calvo, Antonio Correcher Salvador. *Control automático. Tiempo continuo y tiempo discreto*. Editorial Reverté, 26 de Febrero de 2015 .

[2] Castillo, P; Lozano, R; Dzul, A. - Modelling and Control of Mini-Flying Machines. - *Advances in Industrial Control*, 2005.

[3] Randal Beard. - Quadrotor Dynamics and Control Rev 0.1. - *Brigham Young University*, May, 2008.

[4] Ali Reza Partovi, Ang Zong Yao Kevin, Hai Lin, Ben M. Chen and Guowei Cai. - Development of a Cross Style Quadrotor. - *National University of Singapore, Singapore*, August, 2012.

[5] Sebastian O. H. Madgwick ; Andrew J. L. Harrison ; Ravi Vaidyanathan. - Estimation of IMU and MARG orientation using a gradient descent algorithm. - *Zurich, Switzerland*, July, 2011.

[6] Sandoval Gío, J.; Salt Llobregat, J. - Diseño de regulador bifrecuencia mediante compensación de tipo adelanto-atraso de fase. - *Universidad Autónoma de Yucatán, México*, 2011.

[7] Peñacoba Hornillos, VM; García Gil, P. - Plataforma de desarrollo para la configuración del control empotrado en helicópteros cuatri-rotor. - *Proyecto Final de Carrera en la Universidad Politécnica de Valencia*, 2013.

[8] Ródenas Lorda, L; García Gil, P. - Plataforma de desarrollo para el control de estabilidad en tiempo real de un vehículo aéreo tipo quadrotor. - *Proyecto Final de Carrera en la Universidad Politécnica de Valencia*, 2013.

[9] Ardupilot. - Consultado a http://ardupilot.org/. [Septiembre 2019]

[10] Articulo sobre las tendencias de venta de los drones. - Consultado a https://www.businessinsider.com/drone-industry-analysis-market-trends-growth-forecasts-2017-7?IR=T. [Septiembre 2019]

APÉNDICE A
Código

```
#define SerialData SerialUSB // Native USB
#define SerialDebug Serial   // Programming USB
#define SerialIMU Serial1

#define USING_SERVO_LIB  true
#include <DueTimer.h>

#include "Madgwick.h"

/*******************/
/* Control         */
/*******************/
// Time
float T = 0.01; // 10 ms
int nT = 0;
int cnt = nT;

// Throttle
volatile float u_nom = 1000;

// Pitch
volatile float ref_pitch = 0;
volatile float err_pitch = 0;
volatile float err_ant_pitch = 0;
volatile float up_pitch;
volatile float ud_pitch;
volatile float u_pitch;
volatile float kp_pitch = 0;
volatile float kd_pitch = 0;

// Roll
volatile float ref_roll = 0;
volatile float err_roll = 0;
volatile float err_ant_roll = 0;
volatile float up_roll;
volatile float ud_roll;
volatile float u_roll;
volatile float kp_roll = 0;
volatile float kd_roll = 0;

// Yaw
volatile float ref_yaw = 0;
volatile float err_yaw = 0;
```

```
volatile float err_ant_yaw = 0;
volatile float up_yaw;
volatile float ud_yaw;
volatile float u_yaw;
volatile float kp_yaw = 0;
volatile float kd_yaw = 0;

#define M_2PI (M_PI * 2.0)

/********************/
/* PWM              */
/********************/

const byte pwm[4] = {6, 7, 8, 9};
const float cv = 0.4096; // 4096 / 10.000 us

const int minPWM = 1000;
const int maxPWM = 2000;

void initPWM() {
  analogWriteResolution(12);
}

void writePWM(int p, float us) {
  if (us > maxPWM) us = maxPWM;
  else if (us < minPWM) us = minPWM;

  analogWrite(p, cv * us);
}

/********************/
/* Serial Bluetooth */
/********************/

/* Comun */
const uint8_t header = 0x7e;
const uint8_t end_header = 0x7d;

/* Envio */
const uint8_t send_n_data = 4; // numero de datos
const uint16_t send_buffer_length = sizeof(uint16_t) * send_n_data
    ; // Tipo de datos  * numero de datos
volatile uint8_t send_buffer_data[send_buffer_length];
volatile int16_t data; // Auxiliar var to data
volatile int newSend;  // New value to send

/* Recepcion */
volatile uint8_t sync = 0;
volatile uint8_t recv_buffer_data[9];
volatile int16_t recv_data[4];

const int16_t rpy_id = 3000;
const int16_t roll_id = 4000;
const int16_t pitch_id = 5000;
const int16_t yaw_id = 6000;
const int16_t unom_id = 7000;
const int16_t disarm_id = 8000;
const int16_t arm_id = 9000;

```

```
102 /*******************/
103 /* IMU             */
104 /*******************/
105
106 //#include "uNavAHRS.h"
107 #include "MPU9250.h"
108
109 float AccelBiasX =  0.0000;
110 float AccelScaleFactorX =  1.0000;
111 float AccelBiasY =  -0.0692;
112 float AccelScaleFactorY =  1.0002;
113 float AccelBiasZ =  0.0000;
114 float AccelScaleFactorZ =  1.0000;
115 float MagBiasX =  15.2209;
116 float MagScaleFactorX =  1.2557;
117 float MagBiasY =  -16.3533;
118 float MagScaleFactorY =  1.1633;
119 float MagBiasZ =  25.4352;
120 float MagScaleFactorZ =  0.7440;
121
122
123 // an MPU-9250 object on SPI bus 0 with chip select 10
124 MPU9250 Imu(SPI, 10);
125 int status;
126
127 //uNavAHRS Filter; // a uNavAHRS object
128 Madgwick Filter;
129
130 // a flag for when the MPU-9250 has new data
131 volatile int newData;
132
133 // Load accel and mag bias
134 // and scale factors from CalibrateMPU9250.ino
135 float axb, axs, ayb, ays, azb, azs;
136 float hxb, hxs, hyb, hys, hzb, hzs;
137 // Pitch
138 volatile float pitch;
139 volatile float apitch;
140 volatile float dpitch;
141 volatile float offpitch;
142 // Roll
143 volatile float roll;
144 volatile float aroll;
145 volatile float droll;
146 volatile float offroll;
147 volatile float alfaR = 0.5;
148 // Yaw
149 volatile float yaw;
150 volatile float dyaw;
151 // Heading
152 //volatile float head;
153 volatile float aux;
154
155
156 const float radToDeg = 180.0f / PI;
157
158 /*******************/
159 /* Task            */
160 /*******************/
```

```
161
162 // timers to measure performance
163 volatile unsigned long t[5] = {0, 0, 0, 0, 0};
164
165 boolean arm = false;
166 boolean imuStarted = false;
167 boolean sendStarted = false;
168 boolean recvStarted = false;
169
170 volatile boolean readed = false;
171
172
173 /*******************/
174 /* Methods         */
175 /*******************/
176
177 void setup() {
178   // serial to display data
179   SerialData.begin(57600);
180   SerialDebug.begin(57600);
181   //while(!SerialData) {}
182
183   //Serial.println("Started...");
184   // start communication with IMU
185   status = Imu.begin();
186   if (status < 0) {
187     SerialDebug.println("IMU initialization unsuccessful");
188     SerialDebug.println("Check IMU wiring or try cycling power");
189     SerialDebug.print("Status: ");
190     SerialDebug.println(status);
191     while (1) {}
192   }
193
194   axb = AccelBiasX;
195   axs = AccelScaleFactorX;
196   ayb = AccelBiasY;
197   ays = AccelScaleFactorY;
198   azb = AccelBiasZ;
199   azs = AccelScaleFactorZ;
200   hxb = MagBiasX;
201   hxs = MagScaleFactorX;
202   hyb = MagBiasY;
203   hys = MagScaleFactorY;
204   hzb = MagBiasZ;
205   hzs = MagScaleFactorZ;
206
207   Imu.setAccelCalX(axb, axs);
208   Imu.setAccelCalY(ayb, ays);
209   Imu.setAccelCalZ(azb, azs);
210
211   Imu.setMagCalX(hxb, hxs);
212   Imu.setMagCalY(hyb, hys);
213   Imu.setMagCalZ(hzb, hzs);
214
215   // setting a 41 Hz DLPF bandwidth
216   Imu.setDlpfBandwidth(MPU9250::DLPF_BANDWIDTH_41HZ);
217   // setting SRD to 9 for a 100 Hz update rate
218   Imu.setSrd(9);
219   // enabling the data ready interrupt
```

```
220   Imu.enableDataReadyInterrupt();
221   // attaching the interrupt to microcontroller pin 1
222   pinMode(2, INPUT);
223   attachInterrupt(2, runFilter, RISING);
224   //Serial.println("Running...");
225
226   //Filter.setInitializationDuration(30 * 1000 * 1000); // 60
        segundos
227   Filter.begin(100);
228
229   send_buffer_data[0] = 0;
230   send_buffer_data[1] = 0;
231   send_buffer_data[2] = 0;
232   send_buffer_data[3] = 0;
233   send_buffer_data[4] = 0;
234   send_buffer_data[5] = 0;
235   send_buffer_data[6] = 0;
236   send_buffer_data[7] = 0;
237
238   initPWM();
239   writePWM(pwm[0], minPWM);
240   writePWM(pwm[1], minPWM);
241   writePWM(pwm[2], minPWM);
242   writePWM(pwm[3], minPWM);
243
244     int n = 0;
245   while(n < 5000){
246     while (newData != 1) {}
247     newData = 0;
248     t[0] = micros();
249     Imu.readSensor();
250
251     Filter.updateIMU(Imu.getGyroX_rads(), Imu.getGyroY_rads(), Imu
        .getGyroZ_rads(),
252       Imu.getAccelX_mss(), Imu.getAccelY_mss(), Imu.getAccelZ_mss
        ());
253
254     n++;
255   }
256
257   n = 0;
258   while(n < 1000){
259     while (newData != 1) {}
260     newData = 0;
261     t[0] = micros();
262     Imu.readSensor();
263
264     Filter.updateIMU(Imu.getGyroX_rads(), Imu.getGyroY_rads(), Imu
        .getGyroZ_rads(),
265       Imu.getAccelX_mss(), Imu.getAccelY_mss(), Imu.getAccelZ_mss
        ());
266
267     aroll = roll;
268     roll = Filter.getRoll();
269     if (roll > 0) roll -= 180.0f;
270     else roll += 180.0f;
271     offroll += roll;
272     offpitch += Filter.getPitch();
273
```

```
274       n++;
275     }
276
277     offroll /= 1000;
278     offpitch /= 1000;
279
280     Timer1.attachInterrupt(mySender);
281     Timer1.start(100000); // Calls every 100ms
282
283     t[0] = micros();
284     t[1] = micros();
285     t[2] = micros();
286     //SerialDebug.println("Running...");
287 }
288
289 void loop() {
290     if (newData == 1) {
291       newData = 0;
292       t[0] = micros();
293       Imu.readSensor();
294
295       droll = Imu.getGyroX_rads();
296       dpitch = Imu.getGyroY_rads();
297       dyaw = Imu.getGyroZ_rads();
298
299       // update the filter
300       Filter.updateIMU(droll, dpitch, dyaw,
301         Imu.getAccelX_mss(), Imu.getAccelY_mss(), Imu.getAccelZ_mss
      ());
302
303       if(nT == cnt){
304         cnt = 0;
305         droll = droll * radToDeg;
306         dpitch = dpitch * radToDeg;
307         dyaw = dyaw * radToDeg;
308
309         roll = Filter.getRoll();
310         pitch = Filter.getPitch();
311         yaw = Filter.getYaw();
312
313         if (roll > 0) roll = roll - 180.0f;
314         else roll = roll + 180.0f;
315       } else {
316         cnt++;
317       }
318
319
320         roll = roll-offroll;
321         pitch = pitch-offpitch;
322
323         err_pitch = ref_pitch - pitch;
324         up_pitch = -1 * (kp_pitch * err_pitch );
325         ud_pitch = -1 * (dpitch * kd_pitch);
326         u_pitch = up_pitch + ud_pitch;
327         err_ant_pitch = err_pitch;
328
329         err_roll = ref_roll - roll;
330         up_roll = -1 * (kp_roll * err_roll );
331         ud_roll = -1 * (droll * kd_roll);
```

```
332       u_roll = -1 * up_roll + ud_roll;
333       err_ant_roll = err_roll;
334
335       err_yaw = ref_yaw - yaw;
336       up_yaw = (kp_yaw * err_yaw );
337       ud_yaw = dyaw * kd_yaw;
338       u_yaw = up_yaw + ud_yaw;
339       err_ant_yaw = err_yaw;
340
341         if (arm == true) {
342           writePWM(pwm[0],  u_nom +     u_pitch  +     u_roll  +
        u_yaw  );
343           writePWM(pwm[1],  u_nom +     u_pitch  + (-1 * u_roll) +
      (-1 * u_yaw) );
344           writePWM(pwm[2],  u_nom + (-1 * u_pitch) + (-1 * u_roll)
      +     u_yaw  );
345           writePWM(pwm[3],  u_nom + (-1 * u_pitch) +     u_roll  +
      (-1 * u_yaw) );
346         }
347         else {
348           u_nom = minPWM;
349           writePWM(pwm[0], minPWM);
350           writePWM(pwm[1], minPWM);
351           writePWM(pwm[2], minPWM);
352           writePWM(pwm[3], minPWM);
353
354           kp_roll = 0;
355           kd_roll = 0;
356
357           kp_pitch = 0;
358           kd_pitch = 0;
359
360           kp_yaw = 0;
361           kd_yaw = 0;
362         }
363
364         data = rpy_id;
365         send_buffer_data[0]=(uint8_t)(data);
366         send_buffer_data[1]=(uint8_t)(data >> 8);
367
368         data = compress(roll);
369         send_buffer_data[2]=(uint8_t)(data);
370         send_buffer_data[3]=(uint8_t)(data >> 8);
371
372         data = compress(pitch);
373         send_buffer_data[4]=(uint8_t)(data);
374         send_buffer_data[5]=(uint8_t)(data >> 8);
375
376         data = compress(dyaw);
377         send_buffer_data[6]=(uint8_t)(data);
378         send_buffer_data[7]=(uint8_t)(data >> 8);
379     }
380
381     if (newSend == 1) {
382       newSend = 0;
383       sendStarted = true;
384       t[1] = micros();
385       SerialData.write(header);
```

```
386     for (int i = 0; i < send_buffer_length; i++) SerialData.write(
        send_buffer_data[i]);
387     SerialData.write(end_header);
388   }
389   while (newData != 1 && newSend != 1) { // Background task
390 //
        ////////////////////////////////////////////////////////////////////////////////

391     if (SerialData.available() > 0)
392     {
393       recvStarted = true;
394       t[2] = millis();
395       sync = SerialData.read();
396       if (sync == header) {
397         if (SerialData.available() >= 9) {
398           for (int i = 0; i < 9; i++) recv_buffer_data[i] =
        SerialData.read();
399           if (recv_buffer_data[8] == 125) { // Paquete
        syncronizado
400             recv_data[0] = (int16_t)(((uint16_t)recv_buffer_data[1
        ] << 8) | recv_buffer_data[0]);
401             recv_data[1] = (int16_t)(((uint16_t)recv_buffer_data[3
        ] << 8) | recv_buffer_data[2]);
402             recv_data[2] = (int16_t)(((uint16_t)recv_buffer_data[5
        ] << 8) | recv_buffer_data[4]);
403             recv_data[3] = (int16_t)(((uint16_t)recv_buffer_data[7
        ] << 8) | recv_buffer_data[6]);
404
405             switch (recv_data[0]) {
406               case rpy_id:
407                 ref_roll = decompress(recv_data[1]);
408                 ref_pitch = decompress(recv_data[2]);
409                 //ref_yaw = decompress(recv_data[3]);
410                 break;
411
412               case roll_id:
413                 kp_roll = decompress3(recv_data[1]);
414                 kd_roll = decompress3(recv_data[2]);
415                 break;
416
417               case pitch_id:
418                 kp_pitch = decompress3(recv_data[1]);
419                 kd_pitch = decompress3(recv_data[2]);
420                 break;
421
422               case yaw_id:
423                 kp_yaw = decompress3(recv_data[1]);
424                 kd_yaw = decompress3(recv_data[2]);
425                 break;
426
427               case unom_id:
428                 u_nom = recv_data[1];
429                 // SerialDebug.println("unom_id");
430                 break;
431
432               case disarm_id:
433                 // SerialDebug.println("disarm_id");
434                 arm = false;
435                 u_nom = 1000;
```

```
436                   break;
437
438                 case arm_id:
439                   // SerialDebug.println("arm_id");
440                   arm = true;
441                   break;
442
443                 default:
444                   // SerialDebug.println("err");
445                   break;
446               }
447
448             }
449           }
450         }
451       }
452 //
      ////////////////////////////////////////////////////////////////////////////////

453       if(recvStarted == true){
454         if((millis() - t[2]) > 500){
455           //SerialDebug.println((millis() - t[2]));
456           arm = false;
457           u_nom = minPWM;
458           writePWM(pwm[0], minPWM);
459           writePWM(pwm[1], minPWM);
460           writePWM(pwm[2], minPWM);
461           writePWM(pwm[3], minPWM);
462           SerialDebug.println("Apagado");
463         }
464       }
465     }
466
467 }
468
469 int16_t compress(float data) {
470   return (int16_t)(data*10);
471 }
472
473 float decompress(int16_t data) {
474   return ((float)data)/10.0;
475 }
476
477 float decompress3(int16_t data) {
478   return ((float)data)/1000.0;
479 }
480
481
482 void runFilter() {
483   newData = 1;
484 }
485
486 void mySender() {
487   newSend = 1;
488 }
489
490 void fatalError() {
491   writePWM(pwm[0], minPWM);
492   writePWM(pwm[1], minPWM);
```

```
493   writePWM(pwm[2], minPWM);
494   writePWM(pwm[3], minPWM);
495   while (1) {}
496 }
```

Listing A.1: Código de control

```
1  /* Libraries */
2  import processing.serial.*;
3  import controlP5.*;
4
5  import java.text.DateFormat;
6  import java.text.SimpleDateFormat;
7  import java.util.Date;
8  import java.io.FileWriter;
9
10 /* Graphical constansts */
11 static final int sizeX = 1200; // windows size
12 static final int sizeY = 600; // windows size
13
14 static final int padgx = 25; // General Padding X
15 static final int padgy = 25; // General Padding Y
16 static final int spacing = 25; // spacing between elements
17
18 static final int btsx = 80; // Button size X - And textfield
19 static final int btsy = 30; // Button size Y  - And textfield
20
21 static final int btsx2 = 160; // Button size X - And textfield
22
23 /* Graphical variables */
24 ControlP5 cp5;
25 Plotter received;
26 Plotter sended;
27 DropdownList ddl;
28 CheckBox checkbox;
29 Button connect;
30 Textfield kp_roll;
31 Textfield kd_roll;
32 Textfield kp_pitch;
33 Textfield kd_pitch;
34 Textfield kp_yaw;
35 Textfield kd_yaw;
36 Slider u_nom;
37
38 RadioButton ref;
39 Textfield ref_roll;
40 Textfield ref_pitch;
41 Textfield trim_roll;
42 Textfield trim_pitch;
43 int state = 0;
44 int n = 0;
45 float amp_roll;
46 float amp_pitch;
47 float trimval_roll;
48 float trimval_pitch;
49
50 float tri_roll;
51 float tri_pitch;
```

```
52 boolean stb_roll;
53 boolean stb_pitch;
54
55 /* Serial variables */
56 Serial myPort;    // The serial port
57 int bufferSize = 10; // Buffer size event in bytes
58 int baudRate = 57600;
59 String serialSelected = "";
60 float revbuffer[] = new float[4];
61 float sendbuffer[] = new float[4];
62 boolean sendWave = false;
63
64 /* Serial constrains */
65 static final short rpy_id   = 3000;
66 static final short roll_id  = 4000;
67 static final short pitch_id = 5000;
68 static final short yaw_id   = 6000;
69 static final short unom_id  = 7000;
70 static final short disarm_id = 8000;
71 static final short arm_id    = 9000;
72 static final short zero = 0;
73
74 /* File variables */
75 boolean fileOpen = false;
76 FileWriter fw;
77 static final DateFormat sdf = new SimpleDateFormat("dd-MM-yyyy_HH-
     mm");
78 static final String d = ";";
79 static final String end = "\n";
80 static final String header = "iRoll"+d+"iPitch"+d+"iYaw"+d+"oRoll"
     +d+"oPitch"+d+"oYaw"+end;
81
82 /* Methods */
83 void setup() {
84   size(1200, 600);
85
86   cp5 = new ControlP5(this);
87
88   PFont pfont = createFont("Arial", 20, true); // use true/false
      for smooth/no-smooth
89   ControlFont font = new ControlFont(pfont, 20);
90   ControlFont font2 = new ControlFont(pfont, 14);
91
92   // create a new button
93   cp5.addButton("DISARM")
94     .setValue(0)
95     .setPosition(sizeX-padgx-btsx2, padgy)
96     .setSize(btsx2, btsy)
97     .setFont(font)
98     .setId(1)
99     ;
100
101  // create a new button
102  cp5.addButton("ARM")
103    .setValue(0)
104    .setPosition(sizeX-padgx-2*btsx2-spacing, padgy)
105    .setSize(btsx2, btsy)
106    .setFont(font)
107    .setId(2)
```

```
108     ;
109
110   // create a new button
111   cp5.addButton("Update Roll")
112     .setValue(0)
113     .setPosition(sizeX-padgx-btsx2, sizeY-padgy-4*btsy-4*spacing)
114     .setSize(btsx2, btsy)
115     .setFont(font2)
116     .setId(3)
117     ;
118
119   // create a new button
120   cp5.addButton("Update Pitch")
121     .setValue(0)
122     .setPosition(sizeX-padgx-btsx2, sizeY-padgy-3*btsy-3*spacing)
123     .setSize(btsx2, btsy)
124     .setFont(font2)
125     .setId(4)
126     ;
127
128   // create a new button
129   cp5.addButton("Update Yaw")
130     .setValue(0)
131     .setPosition(sizeX-padgx-btsx2, sizeY-padgy-2*btsy-2*spacing)
132     .setSize(btsx2, btsy)
133     .setFont(font2)
134     .setId(5)
135     ;
136
137   // create a new button
138   cp5.addButton("Update U_nominal")
139     .setValue(0)
140     .setPosition(sizeX-padgx-btsx2, sizeY-padgy-btsy)
141     .setSize(btsx2, btsy)
142     .setFont(font2)
143     .setId(6)
144     ;
145
146
147   // add a vertical slider
148   u_nom = cp5.addSlider("")
149     .setPosition(sizeX-padgx-btsx2-2*spacing-2*btsx, sizeY-padgy-
        btsy)
150     .setSize(2*btsx+spacing, btsy)
151     .setRange(1000, 2000)
152     .setValue(1000)
153     .setId(7)
154     ;
155
156
157   ////////////////////////////////////////////////////
158   // ROLL
159
160
161   kp_roll = cp5.addTextfield("  ")
162     .setPosition(sizeX-padgx-3*btsx-2*spacing-btsx2+btsx, sizeY-
        padgy-4*btsy-4*spacing)
163     .setSize(btsx, btsy)
164     .setAutoClear(false)
```

```
165     .setFont(font)
166     .setId(8)
167         .setText("0");
168     ;
169
170   kd_roll = cp5.addTextfield("    ")
171     .setPosition(sizeX-padgx-2*btsx-1*spacing-btsx2+btsx, sizeY-
      padgy-4*btsy-4*spacing)
172     .setSize(btsx, btsy)
173     .setAutoClear(false)
174     .setFont(font)
175     .setId(9)
176         .setText("0");
177     ;
178
179   ////////////////////////////////////////////////////////////
180   // PITCH
181
182   kp_pitch = cp5.addTextfield("     ")
183     .setPosition(sizeX-padgx-3*btsx-2*spacing-btsx2+btsx, sizeY-
      padgy-3*btsy-3*spacing)
184     .setSize(btsx, btsy)
185     .setAutoClear(false)
186     .setFont(font)
187     .setId(10)
188     .setText("0");
189     ;
190
191   kd_pitch = cp5.addTextfield("      ")
192     .setPosition(sizeX-padgx-2*btsx-1*spacing-btsx2+btsx, sizeY-
      padgy-3*btsy-3*spacing)
193     .setSize(btsx, btsy)
194     .setAutoClear(false)
195     .setFont(font)
196     .setId(11)
197         .setText("0");
198     ;
199
200   ////////////////////////////////////////////////////////////
201   // YAW
202   kp_yaw = cp5.addTextfield("       ")
203     .setPosition(sizeX-padgx-3*btsx-2*spacing-btsx2+btsx, sizeY-
      padgy-2*btsy-2*spacing)
204     .setSize(btsx, btsy)
205     .setAutoClear(false)
206     .setFont(font)
207     .setId(12)
208         .setText("0");
209     ;
210
211   kd_yaw = cp5.addTextfield("        ")
212     .setPosition(sizeX-padgx-2*btsx-1*spacing-btsx2+btsx, sizeY-
      padgy-2*btsy-2*spacing)
213     .setSize(btsx, btsy)
214     .setAutoClear(false)
215     .setFont(font)
216     .setId(13)
217         .setText("0");
218     ;
```

```
219
220 //
     ////////////////////////////////////////////////////////////////
221
222 cp5.addTextlabel("K")
223   .setText("K")
224   .setPosition(sizeX-padgx-btsx2-2*btsx-1*spacing, sizeY-padgy-5
    *btsy-4*spacing)
225   .setFont(font)
226   .setColorValue(0)
227   .setId(14)
228   ;
229
230 cp5.addTextlabel("D")
231   .setText("D")
232   .setPosition(sizeX-padgx-btsx2-btsx, sizeY-padgy-5*btsy-4*
    spacing)
233   .setFont(font)
234   .setColorValue(0)
235   .setId(15)
236   ;
237
238 sended = new Plotter( cp5, "Send")
239   .setPosition(padgx, padgy)
240   .setSize(sizeX-2*padgx-3*spacing-btsx2-2*btsx, (sizeY-2*padgy-
    spacing)/2 )
241   .setRange(-22, 22)
242   .setView(Plotter.HOLDLINE) // use Chart.LINE, Chart.PIE, Chart
    .AREA, Chart.BAR_CENTERED
243   .setStrokeWeight(1.5)
244   .setColorCaptionLabel(color(40))
245   .setDecimalPrecision(1)
246   .setId(16)
247   ;
248
249
250 received = new Plotter( cp5, "Receive")
251   .setPosition(padgx, sizeY/2+spacing/2)
252   .setSize(sizeX-2*padgx-3*spacing-btsx2-2*btsx, (sizeY-2*padgy-
    spacing)/2)
253   .setRange(-22, 22)
254   .setView(Plotter.HOLDLINE) // use Chart.LINE, Chart.PIE, Chart
    .AREA, Chart.BAR_CENTERED
255   .setStrokeWeight(1.5)
256   .setColorCaptionLabel(color(40))
257   .setDecimalPrecision(1)
258   .setId(17)
259   ;
260
261 // create a DropdownList,
262 ddl = cp5.addDropdownList("SerialList")
263   .setPosition(sizeX-padgx-2*btsx2-spacing, 2*padgy+spacing)
264   .setSize(btsx2, 2*btsy)
265   .setId(18)
266   ;
267 ddl.addItems(Serial.list());
268
269
```

```
270   // create a new button
271   connect = cp5.addButton("Connect")
272     .setValue(0)
273     .setPosition(sizeX-padgx-btsx2, 2*padgy+spacing)
274     .setSize(btsx2, btsy)
275     .setFont(font)
276     .setId(19)
277     ;
278
279
280   checkbox = cp5.addCheckBox("checkBox")
281     .setPosition(sizeX-padgx-btsx2-2*btsx-2*spacing, sizeY-padgy-6
       *btsy-4*spacing)
282     .setSize(20, 20)
283     .setItemsPerRow(3)
284     .setSpacingColumn(110)
285     .setSpacingRow(20)
286     .addItem("  Export data to csv", 0)
287     .addItem("  Send ref", 1)
288     .setId(20)
289     ;
290
291   //
       ////////////////////////////////////////////////////////////////////////////
292
293   ref = cp5.addRadioButton("radioButton")
294       .setPosition(sizeX-btsx, padgy+3*btsy+2*spacing)
295       .setSize(40,20)
296       .setColorForeground(color(120))
297       .setColorActive(color(255))
298       .setColorLabel(color(255))
299       .setItemsPerRow(1)
300       .setSpacingColumn(10)
301       .setSpacingRow(10)
302       .addItem("  ZERO",0)
303       .addItem("  REF",1)
304       .addItem("  SEN",2)
305       .addItem("  TRI",3)
306       ;
307
308   cp5.addTextlabel("Roll")
309     .setText("Roll")
310     .setPosition(sizeX-padgx-btsx2-2*btsx-1*spacing, padgy+2*btsy+
       2*spacing)
311     .setFont(font)
312     .setColorValue(0)
313     ;
314
315   cp5.addTextlabel("Pitch")
316     .setText("Pitch")
317     .setPosition(sizeX-padgx-btsx2-btsx, padgy+2*btsy+2*spacing)
318     .setFont(font)
319     .setColorValue(0)
320     ;
321
322    cp5.addTextlabel("Amplitud")
323     .setText("Amplitud")
324     .setPosition(sizeX-padgx-btsx2, padgy+3*btsy+2*spacing)
```

```
325     .setFont(font)
326     .setColorValue(0)
327     ;
328
329   ref_roll = cp5.addTextfield("                    ")
330     .setPosition(sizeX-padgx-3*btsx-2*spacing-btsx2+btsx, padgy+3*
        btsy+2*spacing)
331     .setSize(btsx, btsy)
332     .setAutoClear(false)
333     .setFont(font)
334     .setText("0")
335     ;
336
337   ref_pitch = cp5.addTextfield("                                ")
338     .setPosition(sizeX-padgx-2*btsx-1*spacing-btsx2+btsx, padgy+3*
        btsy+2*spacing)
339     .setSize(btsx, btsy)
340     .setAutoClear(false)
341     .setFont(font)
342     .setText("0")
343     ;
344
345   trim_roll = cp5.addTextfield("                    ")
346     .setPosition(sizeX-padgx-3*btsx-2*spacing-btsx2+btsx, padgy+4*
        btsy+3*spacing)
347     .setSize(btsx, btsy)
348     .setAutoClear(false)
349     .setFont(font)
350     .setText("0")
351     ;
352
353   trim_pitch = cp5.addTextfield("                                ")
354     .setPosition(sizeX-padgx-2*btsx-1*spacing-btsx2+btsx, padgy+4*
        btsy+3*spacing)
355     .setSize(btsx, btsy)
356     .setAutoClear(false)
357     .setFont(font)
358     .setText("0")
359     ;
360
361    cp5.addTextlabel("trim")
362     .setText("Trim")
363     .setPosition(sizeX-padgx-btsx2, padgy+4*btsy+3*spacing)
364     .setFont(font)
365     .setColorValue(0)
366     ;
367
368   // create a new button
369   cp5.addButton("Update Ref")
370     .setValue(0)
371     .setPosition(sizeX-padgx-btsx2, sizeY-padgy-5*btsy-5*spacing-1
        0)
372     .setSize(btsx2, btsy)
373     .setFont(font2)
374     .setId(30)
375     ;
376
```

```
377   //
       ////////////////////////////////////////////////////////////////////////////
378
379   sended.addDataSet("roll");
380   sended.setColors("roll", color(255, 0, 0));
381   sended.setData("roll", new float[10*10]);
382   sended.addDataSet("pitch");
383   sended.setColors("pitch", color(0, 255, 0));
384   sended.setData("pitch", new float[10*10]);
385   sended.addDataSet("yaw");
386   sended.setColors("yaw", color(0, 255, 255));
387   sended.setData("yaw", new float[10*10]);
388
389   received.addDataSet("roll");
390   received.setColors("roll", color(255, 0, 0));
391   received.setData("roll", new float[10*10]);
392   received.addDataSet("pitch");
393   received.setColors("pitch", color(0, 255, 0));
394   received.setData("pitch", new float[10*10]);
395   received.addDataSet("yaw");
396   received.setColors("yaw", color(0, 255, 255));
397   received.setData("yaw", new float[10*10]);
398
399   frameRate(10);
400 }
401
402 void draw() {
403   send();
404   sended.push("roll", sendbuffer[1]);
405   sended.push("pitch", sendbuffer[2]);
406   sended.push("yaw", sendbuffer[3]);
407   received.push("roll", revbuffer[1]);
408   received.push("pitch", revbuffer[2]);
409   received.push("yaw", revbuffer[3]);
410   if (fileOpen == true) {
411     try {
412       fw.write(sendbuffer[1]+d+sendbuffer[2]+d+sendbuffer[3]+d+
413         revbuffer[1]+d+revbuffer[2]+d+revbuffer[3]+end);
414       fw.flush();
415     }
416     catch(Exception e) {
417       try {
418         fw.close();
419       }
420       catch(Exception e2) {
421       }
422       fw = null;
423       fileOpen = false;
424     }
425   }
426   background(color(190));
427 }
428
429 // Definir que enviar
430 void send() {
431   sendbuffer[0] = rpy_id;
432   //sendbuffer[1] = sin((frameCount+20)*0.01)*10;
433   //sendbuffer[2] = cos(frameCount*0.1)*10;
```

```
434   //sendbuffer[3] = sin(frameCount*0.1)*10;
435
436
437   switch(state){
438     case(0):
439       sendbuffer[1] = trimval_roll;
440       sendbuffer[2] = trimval_pitch;
441       sendbuffer[3] = 0;
442     break;
443     case(1):
444       sendbuffer[1] = amp_roll+trimval_roll;
445       sendbuffer[2] = amp_pitch+trimval_pitch;
446       sendbuffer[3] = 0;
447     break;
448     case(2):
449       sendbuffer[1] = sin(n*0.1)*amp_roll+trimval_roll;
450       sendbuffer[2] = sin(n*0.1)*amp_pitch+trimval_pitch;
451       sendbuffer[3] = 0;
452     break;
453     case(3):
454
455
456       if(stb_roll) tri_roll = tri_roll + 0.1;
457       else tri_roll = tri_roll - 0.1;
458       if(amp_roll > tri_roll) stb_roll = false;
459       else stb_roll = true;
460
461       if(stb_pitch) tri_pitch = tri_pitch + 0.1;
462       else tri_pitch = tri_pitch - 0.1;
463
464
465
466       if(amp_pitch > tri_pitch) stb_pitch = false;
467       else stb_pitch = true;
468
469       //sendbuffer[1] = tri_roll+trimval_roll;
470       //sendbuffer[2] = tri_pitch+trimval_pitch;
471       sendbuffer[1] = trimval_roll;
472       sendbuffer[2] = trimval_pitch;
473       sendbuffer[3] = 0;
474     break;
475   }
476   n++;
477   if (myPort != null && sendWave == true) sender((short)sendbuffer
       [0], compress(sendbuffer[1]), compress(sendbuffer[2]),
       compress(sendbuffer[3]));
478 }
479
480 void keyPressed() {
481   if (key == ' ') {
482     println("SPACE");
483     if (myPort != null) sender(disarm_id, zero, zero, zero);
484   }
485 }
486
487 void sender(short i1, short i2, short i3, short i4) {
488   byte sendbuffer[] = new byte[10];
489   sendbuffer[0]=126;
490   sendbuffer[1]= (byte)(i1 & 0xff);
```

```
491   sendbuffer[2]= (byte)((i1 >> 8) & 0xff);
492   sendbuffer[3]= (byte)(i2 & 0xff);
493   sendbuffer[4]= (byte)((i2 >> 8) & 0xff);
494   sendbuffer[5]= (byte)(i3 & 0xff);
495   sendbuffer[6]= (byte)((i3 >> 8) & 0xff);
496   sendbuffer[7]= (byte)(i4 & 0xff);
497   sendbuffer[8]= (byte)((i4 >> 8) & 0xff);
498   sendbuffer[9]=125;
499   myPort.write(sendbuffer);
500 }
501
502 void controlEvent(ControlEvent theEvent) {
503
504   if (theEvent.isFrom(checkbox)) {
505     if (theEvent.getArrayValue()[0] == 1) {
506       Date date = new Date();
507       String name = "data." + sdf.format(date) + ".csv";
508       try {
509         fw=new FileWriter(name);
510         File f = new File(name);
511         System.out.println(f.getAbsolutePath());
512         fw.write(header);
513         fw.flush();
514         fileOpen = true;
515       }
516       catch(Exception e) {
517         try {
518           fw.close();
519         }
520         catch(Exception e2) {
521         }
522         fw = null;
523         fileOpen = false;
524       }
525     } else if (theEvent.getArrayValue()[0] == 0) {
526       fileOpen = false;
527       if (fw != null) try {
528         fw.close();
529       }
530       catch(Exception e2) {
531       }
532     }
533     if (theEvent.getArrayValue()[1] == 1) sendWave = true;
534     else if (theEvent.getArrayValue()[1] == 0) sendWave = false;
535   } else if (theEvent.isFrom(ddl)) {
536     serialSelected = (String) ddl.getItem((int)theEvent.value()).
      get("name");
537   } else if(theEvent.isFrom(ref)) {
538     state = (int) theEvent.getValue();
539   } else {
540     Float a_kp;
541     Float a_kd;
542     switch(theEvent.getController().getId()) {
543       case(1):  // DISARM
544       println("DISARM "+theEvent.getController().getId());
545       if (myPort != null) sender(disarm_id, zero, zero, zero);
546       break;
547
548       case(2):  // ARM
```

```
549         println("ARM "+theEvent.getController().getId());
550         if (myPort != null) sender(arm_id, zero, zero, zero);
551         break;
552 
553         case(3):  // Update Roll
554         println("Update Roll "+theEvent.getController().getId());
555         try {a_kp = Float.valueOf(kp_roll.getText()); }
556         catch(Exception e2) {println("Error parsing"); return;}
557         try {a_kd = Float.valueOf(kd_roll.getText()); }
558         catch(Exception e2) {println("Error parsing"); return;}
559         println("OK -> " + a_kp + " " + a_kd);
560 
561         if (myPort != null) sender(roll_id, compress3(a_kp),
    compress3(a_kd), zero);
562         break;
563 
564         case(4):  // Update Pitch
565         println("Update Pitch "+theEvent.getController().getId());
566         try {a_kp = Float.valueOf(kp_pitch.getText()); }
567         catch(Exception e2) {println("Error parsing"); return;}
568         try {a_kd = Float.valueOf(kd_pitch.getText()); }
569         catch(Exception e2) {println("Error parsing"); return;}
570         println("OK -> " + a_kp + " " + a_kd);
571 
572         if (myPort != null) sender(pitch_id, compress3(a_kp),
    compress3(a_kd), zero);
573         break;
574 
575         case(5):  // Update Yaw
576         println("Update Yaw "+theEvent.getController().getId());
577         try {a_kp = Float.valueOf(kp_yaw.getText()); }
578         catch(Exception e2) {println("Error parsing"); return;}
579         try {a_kd = Float.valueOf(kd_yaw.getText()); }
580         catch(Exception e2) {println("Error parsing"); return;}
581         println("OK -> " + a_kp + " " + a_kd);
582 
583         if (myPort != null) sender(yaw_id, compress3(a_kp), compress
    3(a_kd), zero);
584         break;
585 
586         case(6):  // Update U_nominal
587         println("Update U_nominal "+theEvent.getController().getId()
    );
588         if (myPort != null) sender(unom_id, (short)u_nom.getValue(),
     zero, zero);
589         break;
590 
591         case(19):
592         if (!serialSelected.equals("") && myPort == null) {
593           myPort = new Serial(this, serialSelected, baudRate);
594           myPort.buffer(bufferSize);
595         }
596         break;
597 
598         case(30):  // Update U_nominal
599         println("Update Ref "+theEvent.getController().getId());
600 
601         try {amp_roll = Float.valueOf(ref_roll.getText()); }
602         catch(Exception e2) {println("Error parsing"); return;}
```

```
603
604       try {amp_pitch = Float.valueOf(ref_pitch.getText()); }
605       catch(Exception e2) {println("Error parsing"); return;}
606
607       try {trimval_roll = Float.valueOf(trim_roll.getText()); }
608       catch(Exception e2) {println("Error parsing"); return;}
609
610       try {trimval_pitch = Float.valueOf(trim_pitch.getText()); }
611       catch(Exception e2) {println("Error parsing"); return;}
612
613       break;
614     default:
615       break;
616     }
617   }
618 }
619
620 void serialEvent(Serial p) {
621   //println("Event");
622   if (myPort.read() == 126) {
623     if (myPort.available() >= 9) {
624       byte bytes[] = myPort.readBytes(9);
625       if (bytes[8] == 125) {
626         revbuffer[0] = (float)(((bytes[1] & 0xFF) << 8) | (bytes[0
      ] & 0xFF));
627         revbuffer[1] = decompress((short)(((bytes[3] & 0xFF) << 8)
      | (bytes[2] & 0xFF)));
628         revbuffer[2] = decompress((short)(((bytes[5] & 0xFF) << 8)
      | (bytes[4] & 0xFF)));
629         revbuffer[3] = decompress((short)(((bytes[7] & 0xFF) << 8)
      | (bytes[6] & 0xFF)));
630       }
631     }
632   }
633 }
634
635 short compress(float data) {
636   return (short)(data*10);
637 }
638
639 float decompress(short data) {
640   return ((float)data)/10.0;
641 }
642
643 short compress3(float data) {
644   return (short)(data*1000);
645 }
646
647 float decompress3(short data) {
648   return ((float)data)/1000.0;
649 }
650
651
652
653
654 //void keyPressed() {
655 //  // some key events to change the properties of DropdownList d1
656 //  if (key=='1') {
```

```
657 //    // set the height of a pulldown menu, should always be a
        multiple of itemHeight
658 //    ddl.setHeight(210);
659 //  } else if (key=='2') {
660 //    // set the height of a pulldown menu, should always be a
        multiple of itemHeight
661 //    ddl.setHeight(120);
662 //  } else if (key=='3') {
663 //    // set the height of a pulldown menu item, should always be
        a fraction of the pulldown menu
664 //    ddl.setItemHeight(30);
665 //  } else if (key=='4') {
666 //    // set the height of a pulldown menu item, should always be
        a fraction of the pulldown menu
667 //    ddl.setItemHeight(12);
668 //    ddl.setBackgroundColor(color(255));
669 //  } else if (key=='5') {
670 //    // add new items to the pulldown menu
671 //    int n = (int)(random(100000));
672 //    ddl.addItem("item "+n, n);
673 //  } else if (key=='6') {
674 //  } else if (key=='7') {
675 //    ddl.clear();
676 //  }
677 //}
```

Listing A.2: Interfaz gráfica de control

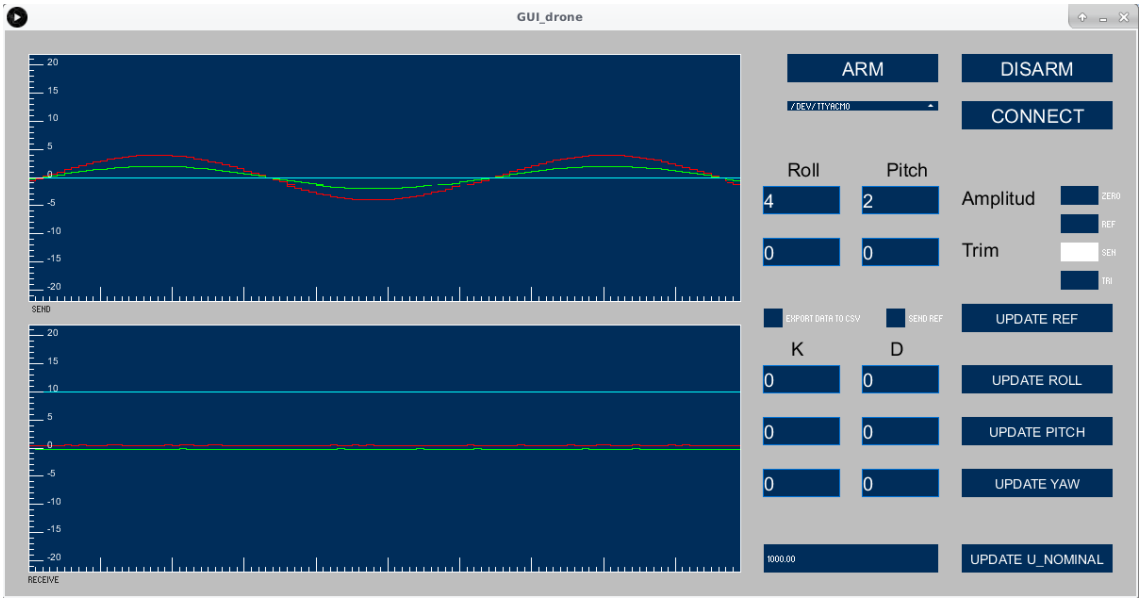

Figura A.1: Interfaz gráfica de control