Universidad Zaragoza
1542

Trabajo de Fin de Grado

Navegación autónoma de un multi-rotor: control automático del avance

Autora:

Inés Portolés García

Director:

Luis Montano Gella

Escuela de Ingeniería y Arquitectura

Zaragoza, Septiembre de 2017

TRABAJOS DE FIN DE GRADO / FIN DE MÁSTER

Escuela de Ingeniería y Arquitectura Universidad Zaragoza

DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. Inés Portolés García,

con nº de DNI 73162835L en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) Grado en ingeniería electrónica y automática, (Título del Trabajo) Navegación autónoma de un multi-rotor: control automático del avance

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, 1 de Septiembre de 2017

Fdo: Inés Portolés García

Índice

Capítulo 1

Introducción

1.1. Contexto y estado del arte

Los vehículos aéreos no tripulados (UAVs, *Unmanned Aerial Vehicles*), comúnmente conocidos como drones, son aviones que viajan sin pasajeros a bordo ni tripulación, y que pueden volar a un nivel controlado de velocidad y altura durante largos períodos de tiempo.

El comienzo de los drones se remonta a la primera guerra mundial, cuyos primeros modelos eran lanzados mediante catapultas o volaban por radio-control. Sin embargo, ya en 1989, Nikola Tesla fue el primero en inventar un barco controlado por radio, dando el primer paso hacia esta tecnología.

En 1918 se construyeron los primeros torpedos, pero no llegaron a usarse gracias al fin de la primera guerra mundial. Se cree que el término *drone* apareció en el período de entreguerras, durante el cual se desarrollaron algunas aeronaves con fines de entrenamiento y que eran manejadas por radio. A partir de entonces los drones fueron utilizados para fines militares.

Figura 1.1: 1935, the DH.82B Queen Bee [1]

En 1993 los drones empezaron a utilizarse para la monitorización del clima y el medio ambiente, y en la actualidad se utilizan en actividades muy distintas, como vídeo y fotografía, juguetes, y en el envío de paquetes a domicilio de la mano de Amazon.

La variedad de aplicaciones que tienen actualmente los drones ha dado lugar a diferentes configuraciones. Este trabajo de centra en un *quadrotor* comercial. Se trata de un drone con cuatro rotores colocados en cruz, y cuyo movimiento se basa en la potencia de rotación de los motores (de la cual depende su altura) y en la diferencia de velocidad entre unos y otros. De este modo, es fácil imaginar una situación en la que los motores traseros actúen más rápidamente que los delanteros, lo que proporcionaría al drone una fuerza en la dirección positiva de X. De la misma forma, dependiendo de la fuerza que ejerzan los motores derechos o izquierdos, el *quadrotor* se inclinará a un lado u otro. En la figura 1.2 se pueden ver los distintos movimientos que realiza el drone, siendo *roll* el giro en torno al eje X, *pitch* en torno al eje Y, y *yaw* el giro en torno a Z.

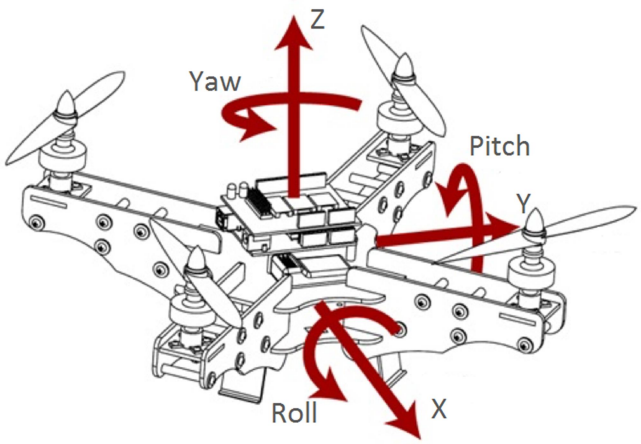

Figura 1.2: Grados de libertad de un drone [2]

Este proyecto se lleva a cabo dentro del grupo de Robótica, Percepción y Tiempo Real de la Universidad de Zaragoza. Este es uno de los grupos de investigación del Instituto Universitario de Investigación en Ingeniería de Aragón (I3A) y es considerado Grupo de Investigación por el Gobierno de Aragón. Dicho grupo tiene las siguientes líneas de trabajo:

- Localización y Mapeado Simultáneo.
- Visión por Computador y Percepción.
- Comunicaciones y redes ad-hoc.
- Exoesqueletos y procesamiento de bioseñales.
- Aprendizaje: en robótica, optimización Bayesiana, interfaces cerebro-ordenador...
- Robótica Móvil. Planificación y navegación.

Este proyecto se centra en la última línea.

1.2. Objetivos

El objetivo de este proyecto es la realización del control de avance del quadrotor *Flame Wheel F450*. La finalidad es, por tanto, controlar la velocidad del drone a partir de un perfil de velocidades y aceleraciones, de modo que se le haga avanzar de forma estable durante un determinado lapso de tiempo. Se utilizará el autopiloto comercial *Naza M-Lite* para el control de los demás grados de libertad y para la estabilización del sistema.

Para realizar un control de velocidad, en primer lugar es necesaria una medida de dicha variable. Frente a la incapacidad de realizar esta medición en pleno vuelo, se deberá encontrar un modelo que se aproxime al comportamiento real del *quadrotor*, y que a partir de las variables que sí es posible medir (aceleraciones lineales, velocidades angulares, inclinación) mediante el uso de sensores, nos dé una estimación de dicha velocidad. Para tal fin, se utilizará un sensor inercial (IMU, *Inertial Measurement Unit*) que aportará tales medidas.

Una vez se obtenga el modelo, se deberá implementar el control deseado. Se realizará la programación de dicho control en la placa de desarrollo *LAUNCHXL-F28377S*. Se integrará en ella el IMU y el módulo de comunicaciones inalámbricas, que será un XBee PRO, conectados a dos puertos de comunicación serie. El control se realizará mediante la comunicación de la consigna de velocidad angular al autopiloto, codificada en un PWM. El entorno de desarrollo utilizado para la programación del software será *Code Composer Studio* y el sistema operativo en tiempo real de *Texas Instruments* (*SYS/BIOS*).

El esquema general del control a realizar es el siguiente:

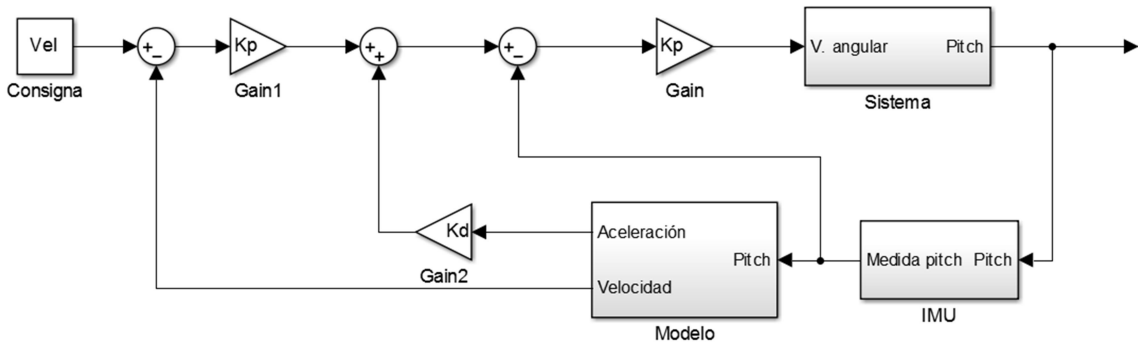

Figura 1.3: Diagrama de bloques del sistema a realizar

1.3. Organización de la memoria

Se ha dividido la memoria en seis capítulos, el primero de ellos la introducción y los cinco restantes que se explican a continuación:

- Hardware y software.
  Se detalla el hardware facilitado por la Universidad y el software utilizado en el PC para la realización del proyecto.

- Análisis del modelo.
  En este apartado se explican los pasos anteriores a la puesta en práctica del proyecto, consistentes en la simulación por ordenador del comportamiento de un drone y la búsqueda de un modelo que lo aproxime.

- Implementación.
  Una vez terminada la parte teórica, se pasa a implementar nuestro software en el drone real, realizando así mismo las conexiones necesarias y la configuración de sus respectivos módulos.

- Pruebas.
  Se procede en este apartado a explicar las pruebas reales realizadas con el drone físico, y las expectativas cumplidas en la fase final del proyecto.

- Conclusiones.
  En este último apartado se exponen las conclusiones a las que se han llegado durante la realización del proyecto.

Capítulo 2

Hardware y software

2.1. Hardware

En este apartado se explica cada uno de los componentes del hardware.

2.1.1. Estructura del quadrotor

El *Flame Wheel F450* es un quadrotor diseñado por DJI. Es una estructura formada por cuatro barras y dos PCBs integradas para facilitar la conexión de los motores y la batería.

Figura 2.1: Flame Wheel F450 [3]

Las barras están fabricadas a partir de un material ultrarresistente, que absorbe la energía de los impactos. Dos de ellas son rojas, color que indica la parte delantera del *quadrotor*. Las otras dos son de color blanco y señalan la parte trasera. Las placas por su parte están colocadas paralelamente entre sí en la parte alta y baja de la estructura, proporcionando un espacio intermedio en el que colocar los componentes electrónicos de nuestro drone, como el autopiloto. Toda esta estructura tiene un peso de 282g y puede soportar entre 800 y 1200g al despegar.

Además, durante este proyecto se ha construido un nuevo tren de aterrizaje robusto que soporte las caídas y permita hacer de soporte para la batería y otros elementos.

2.1.2. Motores y ESCs

Los motores que mueven las hélices del *quadrotor* son de la gama *2212 920KV Brushless Motor* de DJI, se alimentan entre 7 y 12V y pueden llegar a rotar hasta 15416rpm. Estos motores consumen un máximo de 30A, aunque suelen hacerlo entre 15 y 25A. Se controlan a partir de los ESC, que son controladores de velocidad para motores. En este caso se utiliza el modelo *OPTO ESC* de 30A de DJI.

2.1.3. Microprocesador

Se ha utilizado el *TMS320F28377S*. Tiene una frecuencia de bus de 200MHz y una CPU de 32 bits. Es un microprocesador diseñado para el control en bucle cerrado como servo-control, con un subsistema de control en tiempo real. Para este proyecto los periféricos más interesantes serán el módulo PWM, que controlará la velocidad angular del pitch; las salidas de comunicación serie, que servirán para intercambiar información con el ordenador a través del XBee y recoger las medidas del sensor inercial; y la salida a uno de los leds de la placa, que indicará si las tareas se están ejecutando correctamente y si el reloj funciona de la forma requerida, parpadeando una vez por segundo.

2.1.4. Módulo de comunicaciones

El microprocesador se comunica con el ordenador gracias a un XBee Pro, que mediante radiofrecuencia aportará información en tiempo real de las variables que queramos conocer, como por ejemplo el *pitch* medido por el IMU en cada instante, y al mismo tiempo nos servirá para enviar órdenes al drone. Es capaz de comunicarse con el ordenador hasta una distancia de 550m en interior, y a varias decenas de km en exterior. Transmite a una frecuencia de 9600bps.

Figura 2.2: XBee PRO [4]

2.1.5. Multiplexor

En este proyecto van a ser necesarios dos modos de vuelo. El primero de ellos es el modo automático, en el cual el control del drone se hará completamente desde el mando

de radiofrecuencia. El segundo modo es el manual, donde el microprocesador será el que controle los grados de libertad que se precisen.

Para realizar el cambio entre los dos modos de vuelo, automático y manual, será necesario un multiplexor que ceda el control al mando o al microcontrolador respectivamente. Para ello se ha elegido el *Pololu RC Switch*, en concreto el multiplexor *Pololu 4-Channel RC Servo*, que permite cambiar la entrada entre dos fuentes diferentes durante el vuelo. Este multiplexor tiene cuatro *inputs* denominadas *master* (M1-4), donde se conectará la señal proveniente del mando, otros cuatro *slaves* (S1-4), a cuya entrada estará la salida de nuestro PWM, además de las cuatro salidas y una señal de SEL que vendrá del receptor de radiofrecuencia. Para esta aplicación se utilizará solamente un *master* y un *slave* para hacer el cambio en el control del *pitch*.

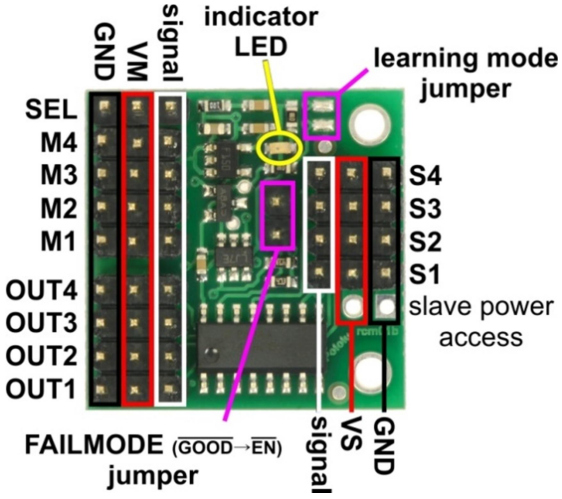

Figura 2.3: Pololu 4-Channel RC Servo

2.1.6. Sensor de medición inercial

El IMU que se ha utilizado es el *9DOF Razor*, que está compuesto por un giróscopo, un magnetómetro y un acelerómetro en la misma placa, y cuyas medidas se integran en un microcontrolador. Este último se encarga de leer las medidas de los sensores y realiza los debidos cálculos para devolver, por la línea SCI, las aceleraciones, velocidades angulares y orientaciones. Este sensor se alimenta a 3.3V, y por tanto se conectará directamente a la alimentación de la placa. Además de alimentación y tierra, se deberán conectar la salida de transmisión y la entrada de recepción a dos pines de recepción y transmisión respectivamente de nuestro microprocesador.

2.1.7. Batería y reguladores de tensión

El fabricante del *quadrotor* nos indica que es recomendable el uso de una batería LiPO (batería de polímero de Litio) de tres o cuatro celdas. Por tanto, la batería utilizada es una *Turnigy* 5.0 de cuatro celdas, 14.8V y 5000mAh.

Se utiliza también un módulo de potencia que proporciona una tensión estable de 5.37V y una corriente de 2.25A. Ha sido necesaria la obtención de una fuente de 3.3V para alimentar tanto a la placa como al IMU y XBee. Para ello se ha construido una pequeña placa, con un regulador de tensión que alimente a algunos pines, y que al mismo tiempo ha servido para organizar el cableado.

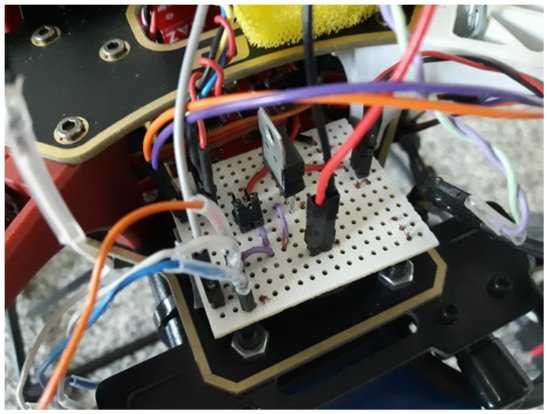

Figura 2.4: Regulador 3.3V

2.1.8. Mando y receptor de radiofrecuencia

El modelo de mando es el *Futaba 6J*. Es un transmisor de 6 canales que emite a una frecuencia de 2.4GHz, y cuya señal es recogida por un receptor de radiofrecuencia de las mismas características. Se va a utilizar 5 de los 6 canales posibles, que servirán para el control del *roll*, *pitch*, *throttle*, *yaw*, y por último para el cambio del modo de vuelo. Los cuatro primeros serán dirigidos a través de dos *sticks*, mientras que el quinto será un *switch* de dos posiciones para los dos modos de vuelo utilizados. Estas señales pasarán directamente del receptor al autopiloto.

Figura 2.5: Futaba 6J

2.1.9. Autopiloto comercial

Se ha utilizado el autopiloto comercial *NAZA-M V1 Lite* de DJI. De los tres modos de vuelo que incorpora, se llevará a cabo el control del drone mediante dos de ellos:

automático y manual. El cambio de uno a otro se realiza a partir de la señal enviada a la entrada U del autopiloto, que se ha acoplado a una de las salidas del receptor de radiofrecuencia.

El control se realiza enviando señales a las cuatro entradas principales del autopiloto, designadas por las letras A (*Aileron*), E (*Elevator*), T (*Throttle*) y R (*Rudder*), que representan las variables *roll*, *pitch*, acelerador y *yaw* respectivamente. Dependiendo de la señal que llega al autopiloto por cada una de estas líneas, se enviará a la salida de los cuatro motores una potencia u otra (codificada para que los ESCs la ejecuten).

Figura 2.6: Naza-M V1 Lite [5]

Al ser el *pitch* la variable a controlar, se conectará a la salida del multiplexor que, según sea el modo automático o manual, dará el control al mando de radiofrecuencia o al microprocesador. El resto de variables serán siempre controladas por el mando, y por tanto se conectarán directamente al receptor.

2.2. Software

Los pasos de simulación, programación y las pruebas necesarias se han llevado a cabo utilizando el software que se detalla a continuación.

2.2.1. Matlab

Matlab es un software de cálculo especializado para ingeniería, que permite expresar cálculos y algoritmos, realizar gráficos, visualizar datos, y en esencia resolver problemas de ingeniería. Se va a utilizar el entorno de diagramas de bloque *Simulink* para la simulación del comportamiento del drone y la realización de un control de *pitch* y de velocidad, que nos dé una aproximación al que finalmente se incorporará. También se utilizará para la búsqueda de un modelo que aproxime el funcionamiento del drone, permitiendo calcular una velocidad que se asimile a la real, a partir de las variables medibles por los sensores. [7][8].

2.2.2. Code Composer Studio

Code Composer Studio (CCS) es un entorno de desarrollo integrado desarrollado por *Texas Instruments*, que permite la programación de sus propios microcontroladores y procesadores integrados. Incluye un compilador C/C++, editor de código y *debugger* entre otras funcionalidades. Es el entorno de desarrollo que servirá para la programación del *F28377S*. [9].

2.2.3. SYS/BIOS

SYS/BIOS es un sistema operativo de tiempo real, diseñado por *Texas Instruments* e incorporado a CCS para la realización de aplicaciones que requieran de sincronización o programación en tiempo real. Este sistema operativo ayuda a minimizar los requerimientos de memoria y CPU, permite interrupciones hardware y software, tareas, funciones periódicas, e incluye estructuras para la comunicación entre tareas como semáforos, buzones y mensajes de longitud variable.

2.2.4. XCTU

Es el programa de ordenador que realiza la comunicación con el XBee. Tiene dos consolas de lectura de mensajes. En una de ellas se visualizan los caracteres enviados y recibidos en hexadecimal, mientras que en la otra se leen los caracteres directamente. Permite crear un protocolo propio para comunicarse con el XBee, y tiene múltiples posibilidades de comunicación (frecuencia, bits de stop, etc.). Resulta una herramienta muy útil para la realización de pruebas del control.

Capítulo 3

Análisis del modelo

Para la realización de un control de velocidad es necesario conocer dicha velocidad, puesto que se precisa de ella para el cierre del bucle de realimentación. Sin embargo, son pocos los sensores inerciales capaces de dar una medida de velocidad a partir de medidas de aceleración y magnetismo, además de ser sensores muy caros. Es por eso que se hace necesario encontrar un modelo similar al del drone que permita conocer la velocidad de forma aproximada, mediante la medida de otras variables.

3.1. Variables a medir

El primer paso es decidir con qué variables es posible aproximar el modelo. Las dos ideas iniciales son las siguientes:

A) Realizar la medida de la aceleración e integrarla para obtener la velocidad. Para ello es necesario conocer también la inclinación, puesto que el IMU proporciona medidas de aceleraciones relativas a su propia posición, y no absolutas como se precisa. Una vez conocidas las aceleraciones y orientaciones es posible crear una matriz de transformación, que pase de coordenadas relativas a absolutas dando una medida finalmente de la velocidad horizontal del drone.

B) Medir la orientación del drone y, mediante simulación, obtener una relación entre ésta y su velocidad. Esto es posible gracias a que es la inclinación la que induce el movimiento al drone, ya que un *pitch* negativo dará como resultado una aceleración positiva. Se puede apreciar esta relación en la figura 3.1, en la que se indica la diferencia de potencia entre los motores delanteros y traseros necesaria para el avance y la inclinación. Se explica de forma más detallada en el anexo 1. En este caso no sería necesaria la matriz de transformación, ya que una inclinación se relaciona directamente con la aceleración absoluta.

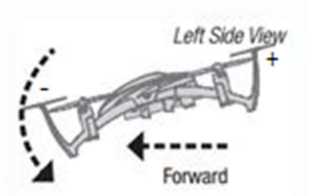

Figura 3.1: Relación del pitch con el avance

Para esta elección se realizaron distintas pruebas.

3.2. Medida de la aceleración

Los primeros experimentos consistieron en el movimiento del IMU de forma manual, es decir, haciendo que el IMU avanzase empujándolo con la mano. Se hizo esta prueba de distintas formas: haciendo avanzar el IMU sobre una superficie plana; elevando el IMU en el aire con la mano y haciendo que avanzara sin ninguna inclinación; elevándolo y avanzando después con una inclinación *pitch*. Tras muchas pruebas se llegó a la conclusión de que la medida de la aceleración era demasiado ruidosa como para que al integrar saliese una medida de velocidad fiable. A esto hay que añadir que en el caso de inclinación no nula había que hacer un cambio de orientación mediante una matriz de transformación, lo cual introducía aún más error en la medida. En la figura 3.2 se puede apreciar una prueba en la que el IMU era arrastrado por una superficie plana para asegurar una inclinación nula. Se puede apreciar el ruido de la aceleración, que, tal y como se puede ver en las figuras 3.3 y 3.4, dan como resultado unas velocidades también ruidosas y con un error bastante notable.

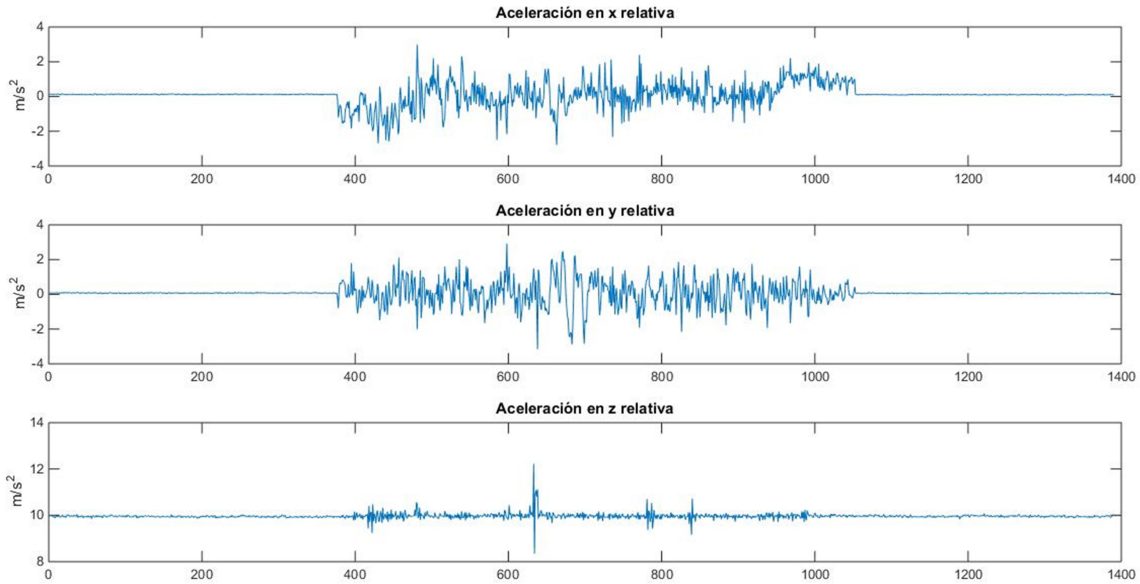

Figura 3.2: Medidas del IMU en el avance

Es interesante destacar que en el último tramo (a partir del instante de muestro 1050 aproximadamente) la velocidad del IMU en todas las direcciones era en realidad nulo. Sabiendo que el periodo de muestreo es de 200Hz, podemos calcular que tras unos 4 segundos de movimiento el error de velocidad es bastante elevado. Además, la velocidad en z es siempre cero, ya que el IMU se mueve en horizontal, y por tanto debería ser siempre nula y no es así. Es posible que este último problema se solucionara suponiendo un offset en la medida de la aceleración, ya que parece que la velocidad cae de forma casi lineal. Sin embargo ha resultado imposible predecir este offset, puesto que de una prueba a otra éste no era constante, y dependía de factores como la orientación.

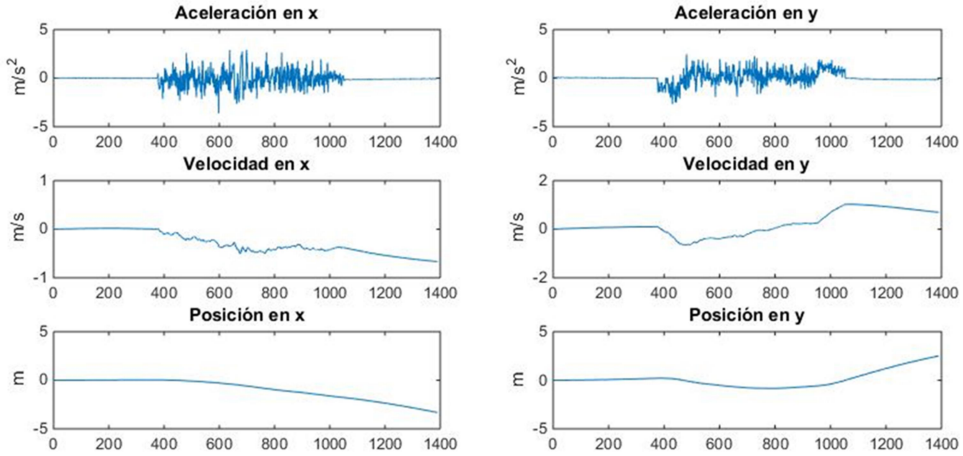

Figura 3.3: Cálculos absolutos en X e Y

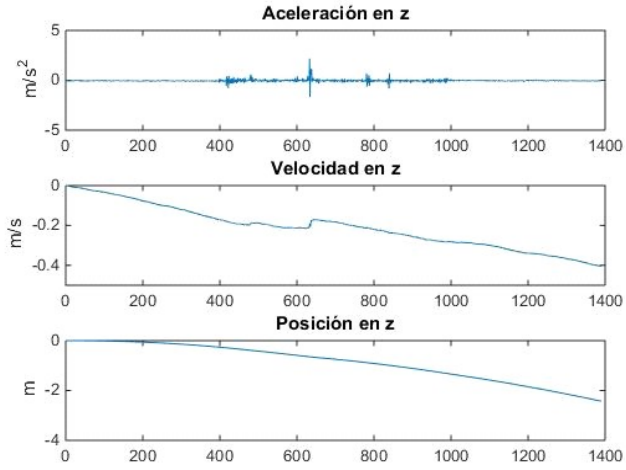

Figura 3.4: Cálculos absolutos en Z

Se pudo concluir, por tanto, que no era posible conseguir una buena medida de la velocidad mediante la integración de la aceleración.

A partir de estas conclusiones se procedió al cálculo del modelo a partir del *pitch*.

3.3. Medida del pitch

Ante la necesidad de encontrar una relación entre el *pitch* y la velocidad, se ha procedido a la utilización de una simulación en *Simulink* de un drone real. En esta simulación es posible visualizar los 12 grados de libertad del drone: *roll*, *pitch*, *yaw*, sus tres velocidades angulares, x, y, z, y sus velocidades lineales.

El diagrama de bloques es el de la figura 3.5. Se distinguen cuatro zonas a simple vista: se representa con el color naranja el control de velocidad en las direcciones x e y; le sigue el control de *roll* y *pitch* de color azul, cuyas consignas provienen del primero; en color negro se controla el ángulo *yaw*; y por último en rojo se distingue el control de altura. El bloque Mixer realiza la conversión entre las acciones de los reguladores en movimiento de los motores, y el bloque Quadrotor simula el sistema del drone, dando lugar a las variables de salida. Todo esto se explica más detalladamente en el Anexo 2.

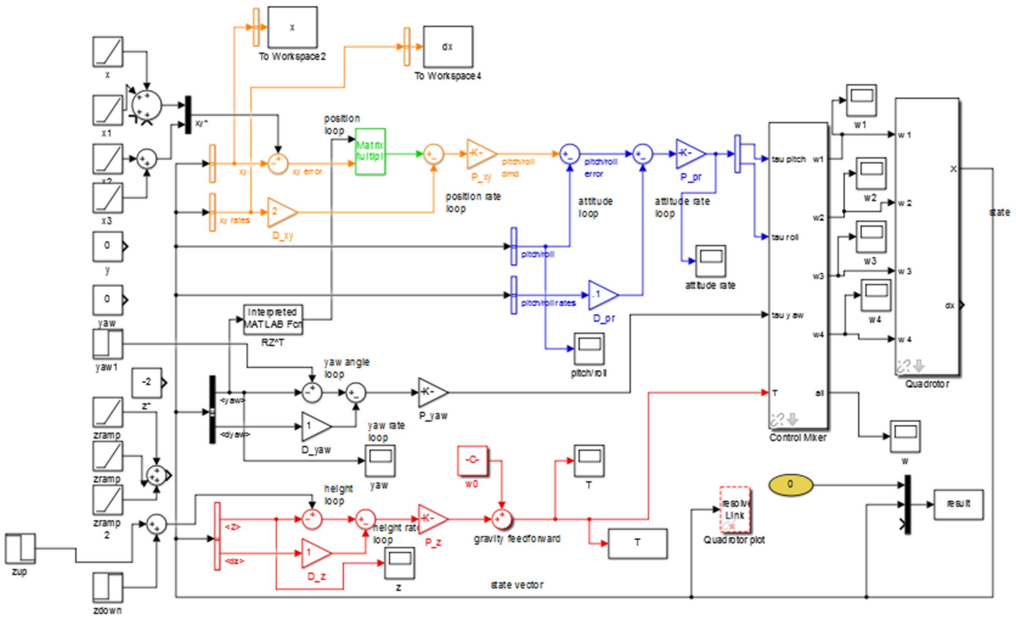

Figura 3.5: Diagrama de bloques del control [11]

Antes de proceder a la simulación del sistema, se realizan algunos cálculos que pueden ayudar a reconocer el modelo más fácilmente. La figura 3.6 muestra la relación de aceleraciones que se proporciona al *quadrotor*, y el cálculo de la aceleración se realiza en las ecuaciones (3.1), (3.2) y (3.3).

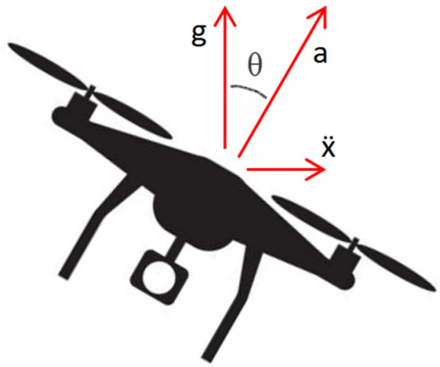

Figura 3.6: Diagrama de aceleraciones

En esta imagen se representa con la letra *a* la aceleración que los motores provocan en el drone, que incluye una aceleración en la dirección del eje z que compensará la acción de la gravedad, y otra en la dirección x que servirá para dar movimiento al drone. Podemos descomponer por tanto la aceleración *a* en sus dos componentes, indicadas en la ecuación (3.1) y desarrolladas a continuación.

$$a \cdot \sin(\theta) = \ddot{x} \qquad a \cdot \cos(\theta) = g \quad (3.1)$$

$$\frac{\ddot{x}}{\sin(\theta)} = \frac{g}{\cos(\theta)} \qquad (3.2)$$

$$\ddot{x} = g \cdot \tan(\theta) \qquad (3.3)$$

Las inclinaciones necesarias para el movimiento de un drone son muy pequeñas, y es por eso que en la ecuación (3.3) se puede aproximar la tangente de θ por el ángulo θ. De este modo queda una función mucho más sencilla y más fácil de integrar, en la que la aceleración es directamente proporcional al *pitch*. Por otro lado, hay que tener en cuenta que el ángulo representado en la imagen corresponde a un *pitch* negativo, así que la aceleración es en realidad proporcional al pitch en negativo.

$$\ddot{x} = -g \cdot \theta \qquad (3.4)$$

Una vez conocida la relación entre *pitch* y aceleración, comprobamos los datos en el simulador.

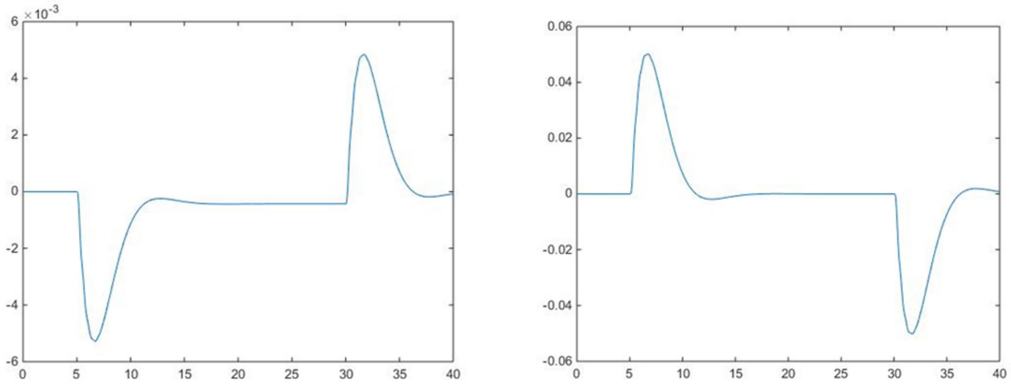

Figura 3.7: Pitch y aceleración lineal en X

En la figura 3.7 se hace evidente esta relación, ya que la forma de ambas gráficas parece la misma pero negativa. El siguiente paso es encontrar la constante que relaciona estas dos variables, que según los cálculos ya expuestos es el módulo de la gravedad. Para ello se exportan ambas variables al *workspace* de *Matlab* y se utiliza la función *polyfit*, que nos devuelve una relación polinómica entre ambas. Sabiendo que la aceleración es directamente proporcional al *pitch*, se piden los dos coeficientes del polinomio de primer grado que más aproxime una función a la otra. Esta relación es la siguiente:

$$a = -9.6892 \cdot pitch - 0.0026 \qquad (3.5)$$

En este punto ya se puede realizar una aproximación de la velocidad a partir del *pitch*, y así se hace. El resultado es una función que se asemeja mucho a la de la aceleración real, pero no es exacta.

Esta aceleración calculada a partir del *pitch* tiene un sesgo, y por tanto aparece un *offset* en todo el resultado, incluido el inicio y el fin del recorrido, en los que la aceleración debería ser cero. Se calcula por tanto la aceleración sin tener en cuenta el sesgo, ya que es imprescindible que la aceleración sea nula en estos instantes. Para ver el error cometido, se resta a la aceleración calculada la aceleración real del drone. La figura 3.8 muestra este error, que es sospechosamente parecido a la forma de la gráfica de la velocidad representada en la misma figura.

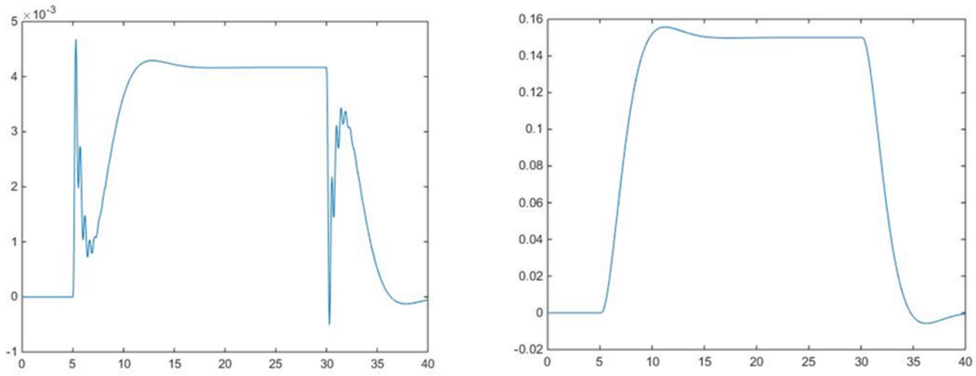

Figura 3.8: Error de aproximación de (3.5) y velocidad lineal

Estudiando el resultado, es fácil suponer que la aceleración dependa, no solo del *pitch* que se le administre, sino también de la velocidad a la que se mueva. Es probable que a velocidades más altas, este error se vea acrecentado por el rozamiento con el aire y otros efectos sobre el rozamiento, que puedan ser determinados por la forma y características del drone. La consecuencia es que, al subir la velocidad, el drone tenga que superar esa fuerza de rozamiento con una pequeña aceleración constante, lo cual significa un mínimo *pitch* para seguir avanzando sin frenarse.

Para que la función de la aceleración sea más realista, es necesario por tanto tener en cuenta el efecto que en ella tiene esta fuerza de rozamiento dependiente de la velocidad. Así mismo vuelve a ser necesario el uso de la función *polyfit* de *Matlab*, que proporcionará la relación entre el error de (3.5) y la velocidad. Se repite el procedimiento anterior y el resultado final es la función (3.6).

$$a = -9.6892 \cdot pitch \quad 0.02436 \cdot velocidad \qquad (3.6)$$

Finalmente, la diferencia entre la función aproximada y la real es la que se indica en la figura 3.9.

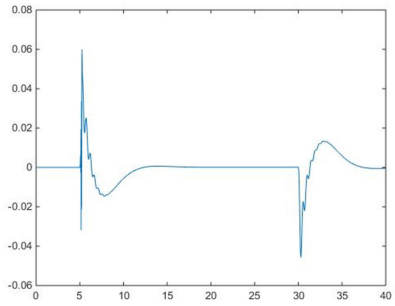

Figura 3.9: Error de aproximación de (3.6)

Este error en la aproximación de la aceleración se supone asumible. De este modo se ha llegado a una estimación de la aceleración que depende tanto del *pitch* como de la propia velocidad. Sin embargo se puede presumir, frente a la primera opción de medida,

que a pesar de ser necesario de igual manera integrar la aceleración, no dependemos de la orientación del drone y por tanto de su matriz de rotación. Además, la medida del *pitch* es más precisa que la de aceleración.

3.4. Simulación de la medida de velocidad

El último paso para decidir si este método de medición-aproximación de la velocidad es válido, es implementarlo en la simulación del drone. Para ello, en lugar de realimentar la velocidad real del drone, se realimenta una aproximación dependiente del *pitch* y de la velocidad. De ese modo, en el bloque "quadrotor" de *Simulink* (figura 3.10), se cambian las medidas directas de velocidad y posición por la salida (y la integral de ésta) de un subsistema cuya única entrada es el *pitch*. La salida de este subsistema, dx, es la velocidad estimada por el modelo calculado. En X se agrupan todas las medidas de las variables que se realimentan en el control, así que se sustituye la medida real por la estimación que se hace a partir del *pitch*.

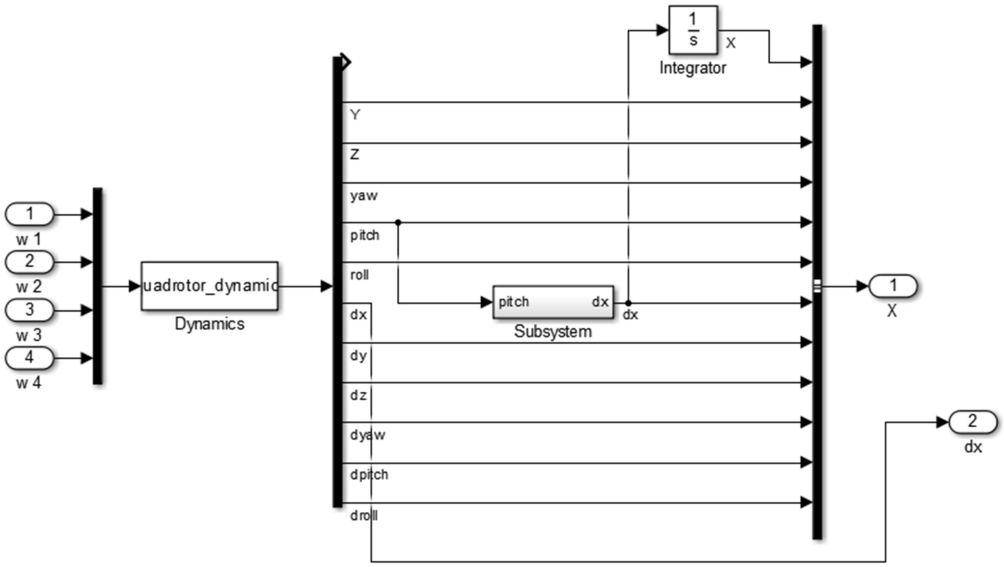

Figura 3.10: Bloque "quadrotor"

La figura 3.11 es el diagrama de bloques que se encuentra en el interior del subsistema. En este caso Gain será la constante que relaciona la aceleración con el *pitch*, y Gain1 representa la de velocidad. Se hace la diferencia entre estos dos términos y se integra, dando lugar a la velocidad, que en la siguiente iteración será la que se realimente.

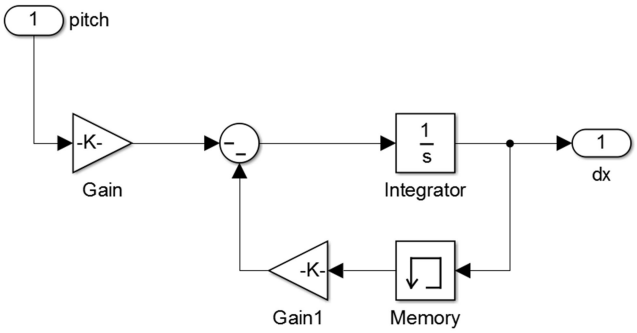

Figura 3.11: Subsistema de cálculo de la velocidad

El resultado es el que se muestra en la figura 3.12. La gráfica de arriba es la velocidad real que lleva el quadrotor, mientras la de abajo representa la velocidad calculada mediante aproximación. Esta prueba se ha realizado a partir una consigna de 0.2m/s tras los primeros 5s.

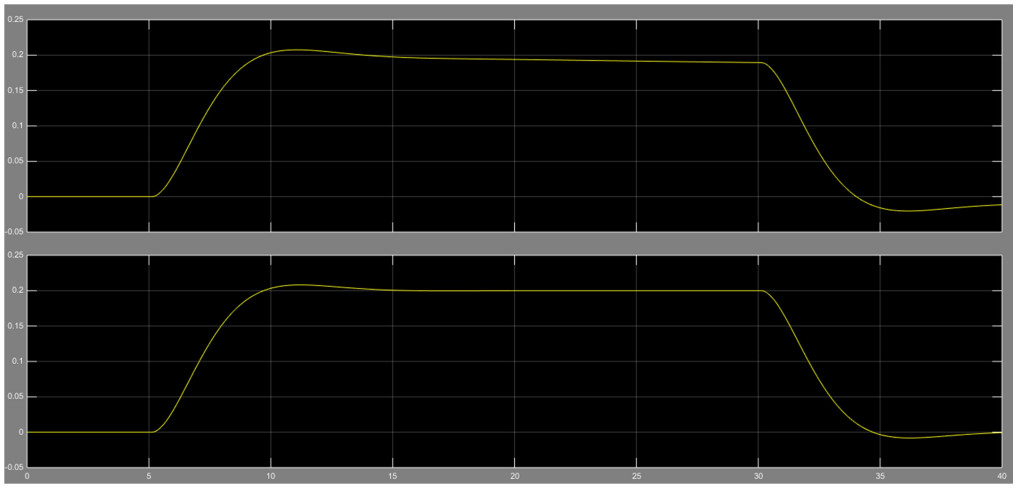

Figura 3.12: Velocidad real y aproximada

Es evidente que hay un pequeño error entre las dos medidas. Este error provoca que el control, creyendo haber llegado a la consigna de velocidad, proporcione una acción algo menor a la debida, dando lugar a una disminución de la velocidad real. Sin embargo, este error no es importante puesto que en un recorrido de 20 segundos la velocidad ha caído en solamente un 5%, y no se pretende realizar recorridos mucho más largos debido a la inestabilidad del drone.

Una vez demostrado que la medida es fiable, se ha cambiado el control de posición de la simulación inicial por un control de velocidad, ya que es el control que se implementará en la realidad. Tras el ajuste de los parámetros el resultado es parecido al caso anterior. En la figura 3.13 se puede observar la consigna de *pitch* en rojo y su seguimiento por parte del sistema en azul.

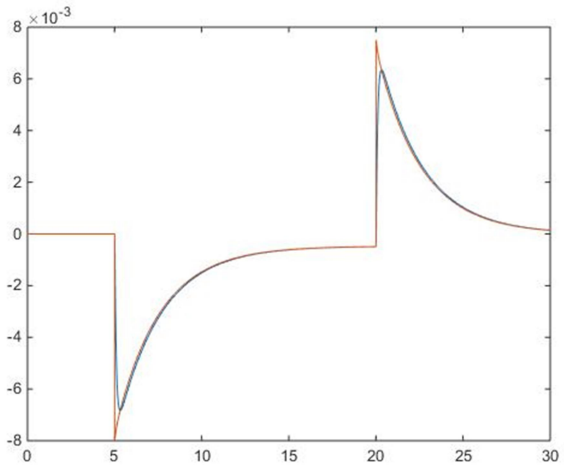

Figura 3.13: Pitch y consigna de Pitch expresados en radianes

Llegados a este punto, tras observar que el resultado es satisfactorio, se decide utilizar la aproximación (3.6) para la medida de la velocidad, aunque es posible que los coeficientes de simulación deban ser ajustados al modelo real del drone en el momento de realizar el control.

Capítulo 4

Implementación

En este apartado se explica la implementación del control en el microprocesador. En primer lugar se explica de forma global la organización de las tareas, y seguidamente se detallan por separado.

Tras realizar las pruebas de simulación y determinar el modelo y el control más propicios, los dos últimos pasos son la implementación en primer lugar del control de *pitch*, y en segundo del control de velocidad que englobaría al primero. La realidad es que el primer objetivo ha sido satisfecho, mientras que el segundo, aunque sí ha sido implementado en el microcontrolador, no ha sido probado ni ajustado de forma experimental. Esto es debido a la dedicación de una buena parte del tiempo en el ajuste del hardware, que abarca la búsqueda de la mejor disposición de los elementos en el drone, algunos problemas debidos a interferencias en el propio sistema receptor (motivo principal de tal consumo de tiempo), la obtención de un regulador de tensión que proporcionase una tensión de 3.3V en varios dispositivos que la requieren, y la introducción del IMU.

Por tanto, esta parte del trabajo se ha centrado en el control de *pitch*, bucle interno para el control de avance que se ha dejado para futuros proyectos.

El control de pitch se ha basado en un regulador proporcional, cuya constante ha sido ajustada mediante pruebas. La implementación de este control incluye la instalación en el microcontrolador de las tareas necesarias para la comunicación con el ordenador, con el IMU y el envío de señales codificadas al autopiloto que representan la consigna de velocidad angular.

La figura 4.1 muestra el conexionado de todos los dispositivos utilizados.

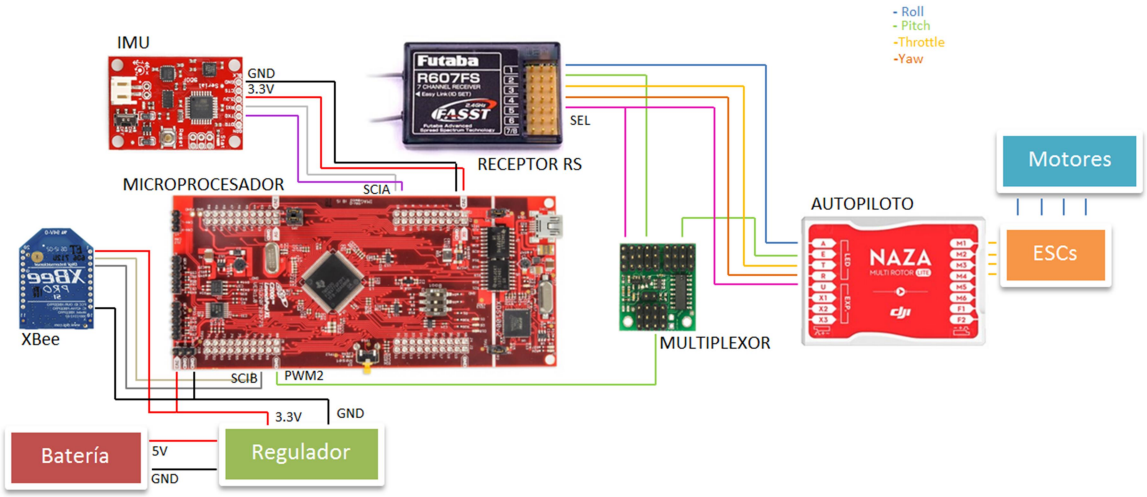

Figura 4.1: Conexionado global

En la imagen no se ha representado la alimentación de los motores, que vendrá directamente de la batería. La línea de 5V es a su vez la salida de un regulador propio de la batería, ya que su tensión es de entre 14 y 16V (dependiendo de si está más o menos cargada). Esta tensión se lleva directamente a las PCBs integradas del drone, y de ahí alimentan a los motores.

Por último, se ha indicado, en la esquina superior derecha, los colores con los que se ha representado cada una de las líneas de comunicación entre el receptor y el autopiloto.

4.1. Diseño del software

Antes de entrar en el funcionamiento individual de cada tarea, se va a explicar qué es una tarea y cómo opera.

Lo principal que hay que saber sobre las tareas en SYS/BIOS es que no son funciones secuenciales que se ejecuten en un orden, sino que, según si la prioridad de cada una es mayor o menor a la de las demás, tendrá mayor preferencia. Se va a diferenciar dos tipos de tareas: periódicas y esporádicas.

Las tareas periódicas son aquellas que se ejecutan una vez por cada ciclo de periodo que les corresponda. Es decir, a una tarea se le asigna un tiempo pasado el cual la tarea se ejecutará, a menos que haya ejecutándose una tarea de mayor prioridad. En este caso la prioridad de las tareas esporádicas vendrá dada por el periodo, dándole mayor prioridad a las tareas que se ejecuten cada menos tiempo.

Las tareas esporádicas no se ejecutan periódicamente como las anteriores, sino que se fuerza su ejecución a partir de una interrupción, ya sea software o hardware.

El fichero *main.c* del proyecto está dividido en una función *main* (principal) que se ejecutará al inicio, cuatro tareas periódicas y una tarea esporádica.

Cuando la placa sea encendida por primera vez o se haya producido un *reset*, se ejecutará la primera función que inicializará los puertos, los módulos que se van a utilizar y el reloj, no sin antes crear e inicializar todas las tareas dándoles su prioridad y periodo (este último solamente en las tareas periódicas).

Como ya se ha comentado, en este programa se han implementado cinco tareas en total:

- LED: consiste en el parpadeo de un led cada segundo. Permite conocer el estado correcto o incorrecto de las tareas y el reloj.
- COM: recibe la información enviada desde el ordenador y le transmite el estado de algunas variables. En esta tarea se decide el estado (*stop*, *mission* o *calibration*) dependiendo de los mensajes que recibe.
- CONTROL: se encarga del control del PWM, de *pitch* y de velocidad.
- PERFIL: desde el momento de inicio de la misión, crea un perfil de velocidad que servirá de consigna para el control.
- IMU: recibe las velocidades angulares, aceleraciones y orientaciones del sensor, las procesa y las comparte con el resto de tareas.

Para que las tareas puedan comunicarse sin que se solapen acciones de lectura o escritura de las variables, es imprescindible el uso de servidores. Cada servidor contendrá funciones de lectura y escritura protegidas por un *mutex* (algoritmo de exclusión mutua), que no permitirá a la CPU pasar de una tarea a otra sin antes terminar de interaccionar con la variable.

Se han utilizado seis servidores en este proyecto: IMU, Status, Consigna, VelReal y Constante.

La estructura final de tareas es la siguiente:

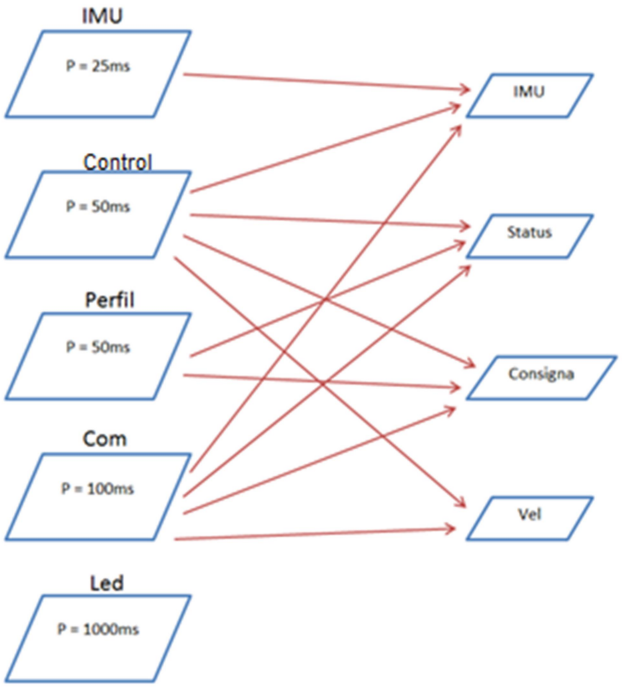

Figura 4.2: Diagrama de tareas

4.2. Tareas

En este apartado se concretará la función de cada una de las tareas.

4.2.1. Led

La tarea del led es una herramienta útil, ya que a pesar de su simpleza, aporta la posibilidad de saber si, en caso de mal funcionamiento del programa, se trata de un problema de mal funcionamiento de las tareas. Puede servir por ejemplo en caso de que el reloj no funcione correctamente, que el programa no se haya cargado de forma satisfactoria en el microprocesador, o que se haya quedado atascado en alguna otra tarea.

Es una tarea periódica a la que se ha dado la menor prioridad, dado que es la que tiene un periodo mayor (1 segundo), y no es imprescindible para el funcionamiento del control.

4.2.2. Comunicaciones

Esta es otra tarea periódica, que envía datos como el *pitch*, la velocidad o la consigna al ordenador para que puedan visualizarse. El envío de esta información se realiza con un periodo de 100 ms, dejando a esta tarea en el penúltimo puesto en prioridad. Además, cada vez que se ejecuta esta tarea, se lee si ha habido algún mensaje recibido.

La comunicación con el ordenador se ha habilitado en uno de los módulos SCI, en concreto en el SCIb. La frecuencia de envío es de 9600 bps, y para configurar el micro para que se comunique de forma satisfactoria hay que tener en cuenta que la frecuencia de reloj es de 200MHz.

El XBee no tiene un protocolo concreto. Se envían mensajes desde el ordenador de la forma que uno quiera, así que hay que adaptar el código de recepción sabiendo cómo se quiere enviar los mensajes. La forma de los mensajes se ha estructurado de la siguiente forma:

- Mensajes transmitidos desde el ordenador.
  Los mensajes que se envían dan la orden de cambiar de estado. Estos mensajes son los siguientes: "#MSN", "#CALI", "#STOP". Cada uno de ellos sirve para indicar en qué estado se desea que se encuentre en drone, y terminan con un salto de línea. Durante las pruebas se ha utilizado un mensaje auxiliar "#SUMK" que indica al micro que se desea cambiar alguna constante del control. Esto ha servido de ayuda para la búsqueda del mejor regulador para el *pitch*.

- Mensajes transmitidos desde el drone.
  Periódicamente se envía un mensaje para visualizar el *pitch*, la velocidad y la consigna desde el ordenador. Este mensaje tiene la estructura:
  #PITCH 1025 #VEL 25 #CONS 0
  Donde las unidades son grados, mm/s, y una de las dos anteriores dependiendo del control que se esté realizando. Los ángulos están multiplicados por 1000 para facilitar el envío de la información. En este caso el pitch sería de 1.025º.

En el bucle de la tarea, el primer paso es recoger la información de *pitch*, velocidad y consigna de sus respectivos servidores, para después enviarlos por la línea de comunicación serie. Una vez hecho esto, la tarea se pregunta si ha recibido alguna orden, y si ha sido así recoge el mensaje y realiza el cambio de estado escribiendo en el servidor.

Los tres estados posibles son:

- *Stop*, en el que la consigna será nula y cuyo objetivo es mantener el drone paralelo al suelo.
- *Mission*, en el que la consigna será la que se haya definido en ese estado. En el caso del control de *pitch*, la consigna será dicho ángulo. En el caso del control de velocidad, la consigna será el perfil de velocidad calculado.
- *Calibrate*. Este estado es casi instantáneo, ya que desde el PC se envía el mensaje cuando se sabe que el drone está apoyado en el suelo y por tanto se tiene una inclinación de 0º. Se guarda como *offset* la medida actual de *pitch*, y

a partir de ese momento se resta ese valor a todas las medidas. Además se inicializa la velocidad a un valor nulo.

Siempre que haya pasado un ciclo del bucle desde la calibración, se pasa directamente al estado STOP.

4.2.3. Perfil

La tarea que calcula el perfil de velocidades no ha sido utilizada finalmente en el control del drone, ya que no ha sido posible la realización del control de velocidad mediante los sensores y el tiempo disponible para este trabajo. Sin embargo sí que fue programado, ya que la idea inicial era llegar a utilizar un control de velocidad real.

En esta tarea se comienza a calcular un perfil que aumenta la velocidad con el tiempo hasta llegar a un valor máximo, y a partir de ese momento se mantiene la consigna constante durante un tiempo determinado. El último paso es hacer descender la velocidad hasta llegar a cero. Este proceso comienza cuando el estado pasa a MISSION, y al terminar cambia el estado de nuevo a STOP.

4.2.4. Control

Para la realización del control de *pitch* se ha utilizado la salida del PWM2. Este control se basa en el envío de una señal de periodo 13’6ms y cuyo tiempo en alto varía entre 1 y 2ms, representando estos las máximas velocidades angulares en positivo y negativo respectivamente, y correspondiendo un tiempo en alto de 1’5ms a una velocidad angular nula.

Para las pruebas de control del *pitch* con el drone se lee el estado y, según si es MISSION o STOP, la consigna es una u otra. Una vez elegida la consigna, se realiza un control proporcional, midiendo el *pitch* y cerrando el bucle. La acción de este controlador es la velocidad angular, que se envía de la forma especificada al PWM.

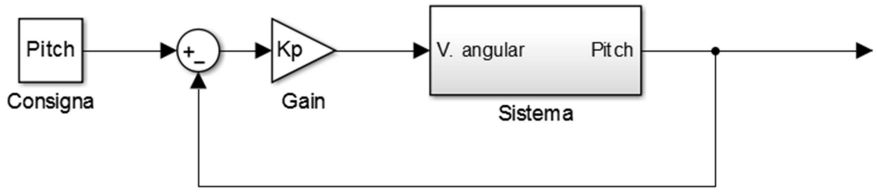

Figura 4.3: Regulador de Pitch

El control de velocidad es un proporcional derivado. Esto proviene de los controladores ajustados en simulación, y la velocidad se calcula gracias al modelo calculado anteriormente. Esta última no es una medida exacta, ya que se calcula a partir del ángulo. De este modo, la medida de la velocidad no es muy fiable, además de estar calculada para un drone distinto al que disponemos, pero podría servir como primera aproximación ajustable mediante pruebas en las que la velocidad sea conocida.

Este regulador ha sido programado en el microcontrolador. Corresponde a una función que se ejecuta en el bucle de la tarea y que devuelve la consigna de *pitch* del sistema, que será el input de la función de control de *pitch*. Sin embargo no ha sido finalmente utilizado, ya que las pruebas del control de *pitch* no han dado el resultado esperado, y ha sido comentado el código que llama a esta función. En caso de haber sido probado, se habrían ajustado empíricamente los parámetros de simulación con el drone real.

También el cálculo de la velocidad a partir del modelo del drone ha sido implementado en el código, pero al no haber sido probado el control de velocidad tampoco se ha precisado su uso. Ambos quedan registrados como idea inicial para un control real del avance que pueda ser ajustado o modificado en un proyecto futuro.

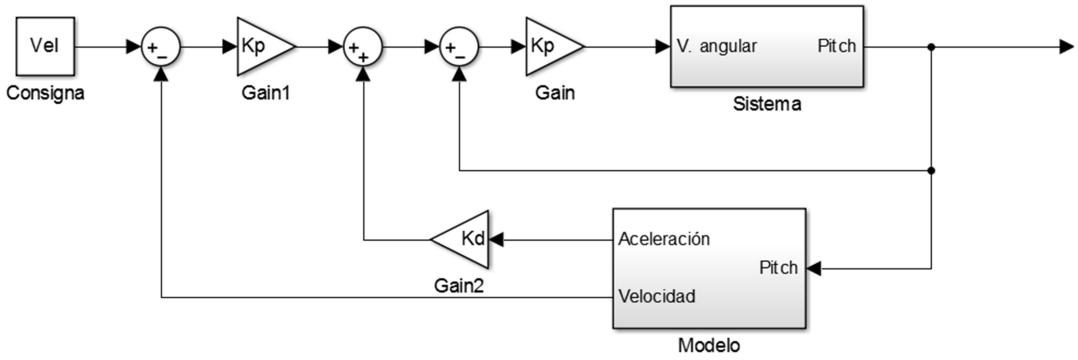

Figura 4.4: Regulador de velocidad

Por último, la consigna se envía a las comunicaciones para que sea enviada al ordenador.

4.2.5. IMU

El IMU se comunica con el micro a partir del módulo SCIa, con una frecuencia de 57600bps y el envío de un mensaje cada 25ms.

En esta tarea se lee el *pitch* medido por el sensor, y se ha implementado una mediana móvil de las cinco últimas medidas. Una vez conocido el valor del *pitch*, este se envía al servidor correspondiente.

Cada mensaje aporta información de las tres aceleraciones lineales, las tres velocidades angulares, y los tres ángulos del quadrotor, en este orden. Sin embargo lo que interesa para este control son los ángulos, así que se busca en cada trama la parte en la que se encuentra esta información. La estructura es la siguiente:

YPR=12.3251,0.5254,1.8961

Este es solo un ejemplo en el que se envía en primer lugar el inicio "YPR=" seguido de los tres ángulos *yaw*, *pitch* y *roll*. Por tanto, se busca en la trama este inicio y se recogen los datos que se reciben a continuación. El resto de información se desecha.

4.3. Servidores

Cada servidor se ha utilizado para almacenar una variable que se lee y escribe desde distintas tareas, y cuya lectura y escritura se protegen mediante un semáforo. Estos sensores y las variables que almacenan son los siguientes:

- IMU: almacena el valor del *pitch*. Esta variable se escribe desde la tarea del IMU, y es leída tanto por las comunicaciones, que enviarán esta información al ordenador, como por la tarea de control que utilizará esta medida para la realización del control.
- Status: como bien indica su nombre, guarda el valor del estado, que es escrito normalmente por las comunicaciones pero que podría hacer el cambio a STOP desde la tarea PERFIL al terminar el recorrido. Esta información será leída tanto por el perfil como por la tarea de control.
- Consigna: dependiendo del control que se realice, se utilizará para la consigna de *pitch* o de velocidad, que será escrita por el control y leída por las comunicaciones en el primer caso, o escrita por el perfil y leída tanto por el control como por las comunicaciones en el segundo.
- VelReal: sirve para enviar la velocidad calculada en la tarea de control a la tarea de comunicación.
- De forma auxiliar se ha utilizado el servidor Constante, que ha sido de ayuda para el cambio de las constantes de control durante las pruebas sin necesidad de cargar nuevamente el programa en el microcontrolador.

4.4. Análisis de tiempo real

La tabla 4.1 muestra los tiempos de cómputo de cada tarea (C), sus plazos de respuesta (D) y sus periodos (P). Los primeros se miden directamente mientras se ejecuta el programa, mientras que los segundos se equiparan a los terceros, ya que no es necesario que la tarea se realice antes de un tiempo concreto, sino como mínimo antes de su siguiente ejecución.

Aunque la tarea del IMU es esporádica, puede calcularse como una tarea periódica ya que se ejecuta cada 25ms, que es el tiempo que el IMU tarda en enviar una nueva medida.

Por su parte, B_{HP} es el tiempo de bloqueo por herencia de prioridad. Esto es, el tiempo que cada tarea puede ser bloqueada por otra (u otras) de menor prioridad debido a la utilización de un servidor común.

TAREA	Prioridad	C (µs)	P=D (ms)	B_{HP} (µs)
IMU	5	56	25	1.8
Control	4	11.32	50	3.6
Perfil	3	1.825	50	1.8
Com	2	996	100	0
Led	1	0.23	1000	0

Para que todas las tareas se ejecuten en el tiempo marcado es necesario que cumplan sus plazos. Una tarea que no cumple plazos no se ejecutará en los tiempos fijados, o no será posible asegurar que lo hace. Para determinar si eso es así se realiza la planificación de tareas.

Como ya se ha comentado, la prioridad es mayor para las tareas que se ejecutan con mayor frecuencia. Las tareas Control y Perfil sin embargo tienen el mismo periodo, y se ha elegido la tarea Control como la de mayor prioridad entre ellas por su importancia, ya que es la que realiza el control del drone. Se utiliza el teorema de *Liu & Layland,* que dice que los plazos se cumplen si se cumple la siguiente relación:

$$U = \sum_{i=1}^{n} \frac{C_i}{P_i} \leq n\left(2^{\frac{1}{n}} - 1\right) = U_0(n) \qquad (4.1)$$

Donde n es el número de tareas y U la utilización del procesador. Para las cinco tareas de que se dispone $U_0(5) = 0.743$. El cálculo de U es el siguiente:

$$U = \frac{0.056}{25} + \frac{0.01132}{50} + \frac{0.001825}{50} + \frac{0.996}{100} + \frac{0.00023}{1000} = 0.01245 \leq 0.743 \qquad (4.2)$$

Se cumple el teorema y, por tanto, las tareas cumplen sus plazos de respuesta. Este teorema es suficiente pero no necesario, lo que significa que en el caso de no haberse cumplido, no se podrían garantizar los plazos y se habría procedido a utilizar otro teorema que sí fuera necesario.

Capítulo 5

Pruebas

Tras las pruebas de simulación y la implementación del software en el microprocesador, se ha procedido a probar el control del *pitch* en el drone real.

Las primeras pruebas consistieron en el control del PWM sin conectar el micro al autopiloto. Para ello, se le dio una consigna de *pitch* al controlador y se inclinó manualmente el drone hacia un lado y otro para ver la respuesta del regulador. Tras algunos cambios, el control proporcional implementado funcionó perfectamente, y para una consigna inicial nula, el PWM aumentaba o disminuía el tiempo en alto dependiendo de la inclinación positiva o negativa del drone. Al encontrarse con una consigna de *pitch* distinta se vio cómo el PWM también cambiaba de forma proporcional.

En la figura 5.1 se observa la variación del PWM según el *pitch* para una consigna de *pitch* nula. La primera gráfica presenta el ángulo, y en la segunda se observa como el PWM sigue la forma de la primera. Como ya se ha comentado, el PWM proporciona una señal que permanece entre 1 y 2ms en alto y con un periodo de 13,6ms. La variable de la gráfica es el registro de comparación del PWM, que da 1ms de tiempo en alto si el valor es de 3125, que corresponde a una consigna de velocidad angular negativa máxima, y 2ms si es 6250, que expresa la máxima velocidad angular positiva. Un valor en el registro de 4687 sería el punto medio, enviando al autopiloto una consigna de velocidad angular nula. Al mismo tiempo se ha comprobado que el PWM proporcione la señal correcta mediante un osciloscopio.

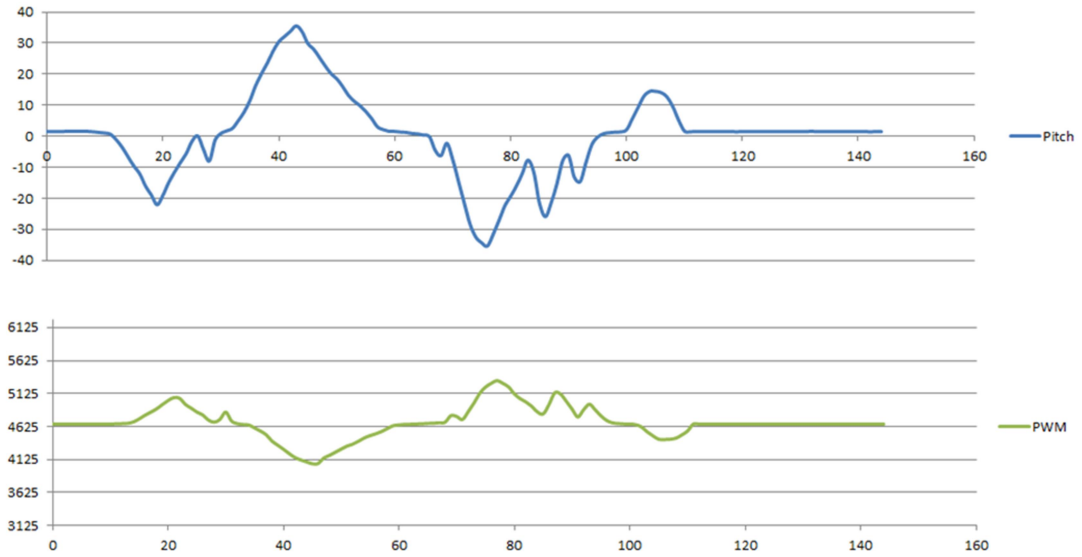

Figura 5.1: Control de PWM con consigna de pitch nula

Una vez se comprobó que el control era correcto, se conectó el PWM al autopiloto y se probó el control de los motores sin hélices. Esta prueba consistía solamente en constatar que los motores delanteros y traseros aumentaban o disminuían la velocidad al recibir una consigna de *pitch* positiva o negativa.

Debido a que las pruebas de avance del drone pueden resultar peligrosas, y dado que se quiere verificar solamente en esta implementación el correcto control del *pitch*, se procedió a sujetar el drone mediante cuerdas que le permitían pequeños desplazamientos, a la vez que evitaban posibles colisiones en el Laboratorio. Se puede ver en la figura 5.2.

Figura 5.2: Imagen del drone asegurado mediante cuerdas

La primera prueba fue, desde el suelo, elevar el drone utilizando el control de *pitch*. La finalidad era que el pitch fuese cero al elevarse, y que lo hiciese de forma estable.

Sin embargo no siempre funcionaba y dependía mucho de lo cargada que estuviese la batería.

La segunda prueba fue elevar el drone utilizando el mando de radiocontrol, y después pasar al modo manual en el que el micro controlara el ángulo. La idea era probar distintas constantes para el regulador proporcional y encontrar aquella para la que el control fuese suave y llegase a la consigna deseada. Sin embargo no fue posible llegar de forma limpia a la consigna de *pitch*, y el resultado fue oscilante y con errores bastante elevados. La figura 5.3 es la representación de una de estas pruebas.

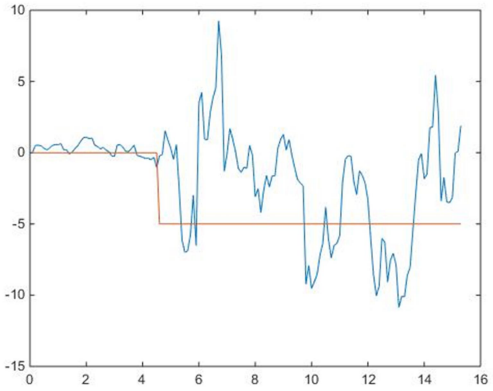

Figura 5.3: Prueba del control de pitch, consigna: 5º

Con el drone parado y apoyado en el suelo, idealmente no debería variar la medida del *pitch*. Sin embargo ésta oscilaba entre valores distantes entre sí en 0.1º. En las pruebas de simulación, un control de velocidad proporciona al drone inclinaciones poco mayores de 1º, como se puede comprobar en la figura 5.4, que corresponde a una consigna de velocidad de 0.2m/s en la primera gráfica y de 0.5m/s en la segunda. El primer caso se corresponde mejor con la realidad, ya que no se va a proporcionar al drone velocidades tan altas como en el segundo. Sin embargo es útil ver su comportamiento para velocidades altas y comprobar los máximos. Esto indica que, en el mejor de los casos, el error de la medida será de casi un 10% sobre la consigna de *pitch*, y en condiciones normales será hasta del 25%. Por tanto, es bastante probable que la medida de este sensor no sea lo suficientemente precisa para la realización del control.

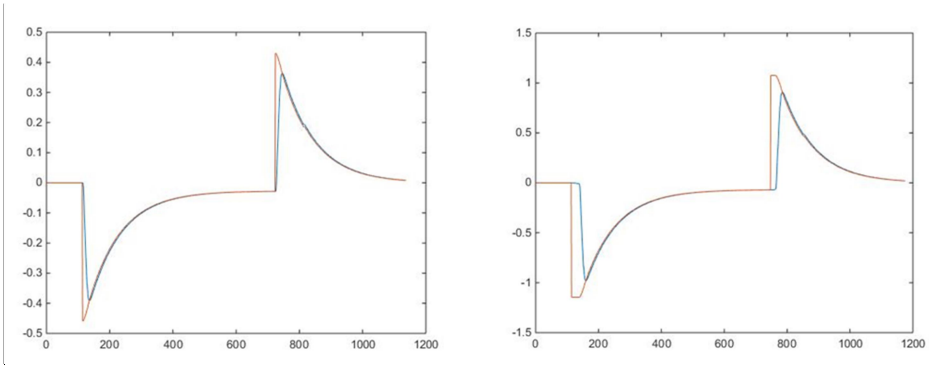

Figura 5.4: Pitch y consigna de Pitch expresadas en grados (simulación)

Se ha comprobado que las constantes del regulador necesarias para el control varían dependiendo de si la batería se encuentra más o menos cargada, puesto que, aunque la consigna es la velocidad angular, el autopiloto no da suficiente potencia a los motores incluso para la misma consigna.

Las pruebas de *pitch* no han sido del todo insatisfactorias ya que, en pleno vuelo, el control es capaz de proporcionar una inclinación al drone hacia el lado que se pide, y que oscila en torno al valor de pitch deseado. Es cierto que las gráficas que se ha conseguido guardar son bastante oscilantes y poco precisas, pero se ha llegado a realizar pruebas en las que se apreciaba perfectamente el giro del drone hacia la posición indicada, en las que finalmente el drone ha aguantado en esa posición de manera más o menos estable.

Capítulo 6

Conclusiones

Como ya se ha expuesto al inicio de esta memoria, el objetivo de este trabajo era la realización de un control de avance en un drone. Los tres pasos necesarios son la medida de la velocidad, la implementación de un control de posición angular *pitch*, y la de un control de velocidad que englobara al anterior. Sin embargo este último no ha podido ser abordado debido al tiempo invertido en la puesta a punto del IMU y del receptor, y en el ajuste del controlador del *pitch*, hasta conseguir el control de esta variable.

Sin embargo, se ha sentado un precedente gracias a las pruebas realizadas con el IMU, que demuestran que no es recomendable la utilización de las medidas de aceleración para la medida de la velocidad, y que esta se puede aproximar de una forma más sencilla mediante la medida del ángulo *pitch*.

Además, se ha evidenciado la necesidad de usar un sensor inercial más preciso en esta aplicación, puesto que, según los datos de simulación, las inclinaciones necesarias son demasiado pequeñas para utilizar un sensor de poca precisión.

Por último, se ha diseñado un control de *pitch*, absolutamente necesario como primer paso para el control de velocidad de avance. Se ha diseñado además el control de velocidad que, aunque no ha sido probado, ha quedado registrado en el código del proyecto y puede ser reutilizado.

Por tanto, con el trabajo que ya ha sido adelantado, es factible proponer la finalización del control de velocidad para un TFG próximo, en el que este trabajo pueda ser reutilizado y terminado con sensores más precisos y un tiempo del que no se dispone en un único TFG.

Finalmente añadir que en el transcurso de este trabajo he aprendido mucho en cuanto a la programación de microprocesadores en tiempo real, y algo muy importante como es encontrar soluciones por mí misma a los problemas que se han planteado, dejando atrás la costumbre de que los docentes tengan la respuesta como puede ocurrir en el resto de asignaturas.

Apéndice A

Principios del movimiento

En este primer apéndice se van a abordar los principios del movimiento de un drone, es decir, qué factores intervienen en su movimiento y, en esencia, cómo se mueve.

La razón por la que un drone se eleva, se inclina o se desplaza es la tracción que realizan sus cuatro motores. No solo eso, sino la diferencia de tracción entre unos y otros. La figura A.1 presenta estas tracciones como T_1, T_2, T_3 y T_4.

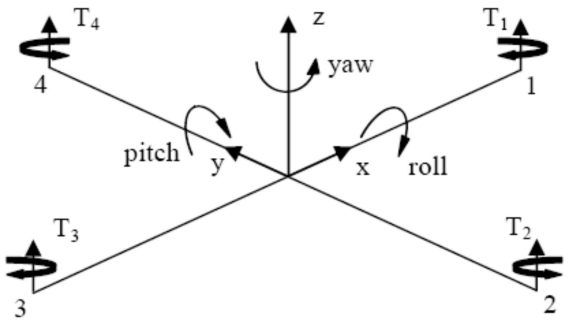

Figura A.1: Posibles giros de un quadrotor [12]

Hay dos formas de controlar el movimiento de un quadrotor. La primera de ellas es el movimiento en 'I' o movimiento en '+', representado a la izquierda de la figura A.2 y que trata de realizar el desplazamiento en la dirección de uno de los motores. La segunda forma, a la derecha de la imagen, es el movimiento en 'X', y su desplazamiento se realiza en la dirección que marcan dos de sus motores.

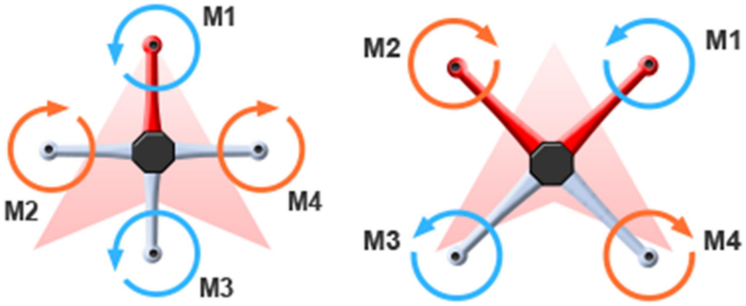

Figura A.2: Mixer en '+' y Mixer en 'x'

Estos dos tipos de movimiento son los dos tipos de Mixer posibles para un quadrotor. El Mixer de un drone es el sistema de reparto de fuerzas sobre los motores. Según cuál sea el tipo de Mixer, las fuerzas que se ejercen sobre cada uno de ellos serán diferentes. En esencia se puede decir que es la dirección en la que mirar el drone, qué dirección imponer para x. Esto influye directamente en las variables a controlar, ya que el ángulo *pitch*, *roll* o *yaw* será uno u otro dependiendo de la dirección de x e y.

El primer tipo de Mixer es el más inestable para movimientos frontales y laterales, ya que la tracción necesaria para su desplazamiento la realiza solamente con la diferencia entre dos motores, y son estos por tanto los que aportan toda la fuerza que lleva al movimiento. En la figura A.3 se observa como ejemplo el movimiento frontal. La tracción de los motores 2 y 4 es constante, mientras que el avance o retroceso dependen de la diferencia entre el motor 1 y el 3. En el caso de un movimiento lateral, se invertirían los roles de las dos parejas de motores.

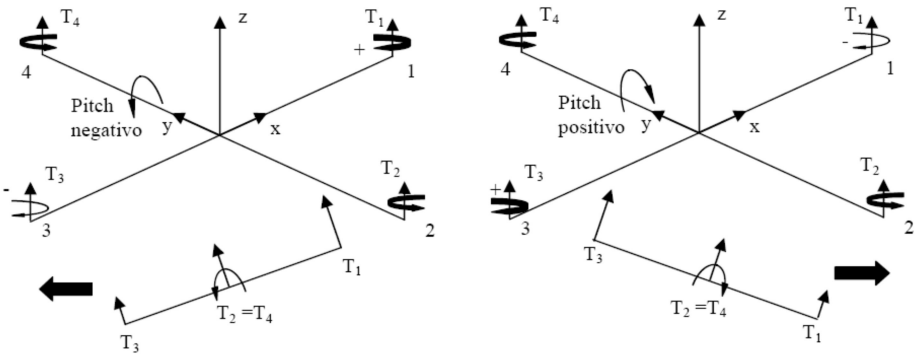

Figura A.3: Movimiento frontal de un quadrotor en '+' [12]

El Mixer en 'X' sin embargo realiza un control diferencial entre los dos pares de motores. Para un movimiento hacia delante, los motores 1 y 2 realizan una fuerza menor y los motores 3 y 4 hacen elevarse la parte trasera del drone. Por tanto, todos los motores intervienen en el desplazamiento del drone, y todos ellos son parte del control en las direcciones frontal y lateral, que son las que más interesan al control de este trabajo. Por tanto, es este el Mixer utilizado para la realización de este TFG.

En las representaciones anteriores se ha supuesto que la dirección de z es hacia arriba. Sin embargo, dependiendo de la fuente, se puede encontrar de las dos formas. Para esta explicación se va a suponer una z en el sentido contrario.

Las ecuaciones de este Mixer son las siguientes, donde T son las tracciones debidas a los cuatro tipos de control: *pitch* (θ), *roll* (ϕ), *yaw* (ψ) y T para la aceleración en vertical. Se representa por *w* la acción de cada uno de los motores.

$$w_1 = T_\theta + T_\psi + T \qquad \text{(A.1)}$$

$$w_2 = -(T_\theta - T_\phi - T_\psi + T) \qquad \text{(A.2)}$$

$$w_3 = -T_\phi + T_\psi + T \qquad \text{(A.3)}$$

$$w_4 = -(-T_\psi + T) \qquad \text{(A.4)}$$

El signo negativo de 2 y 4 se deben a que sus motores giran en el sentido contrario que 1 y 3. Si obviamos estos signos, se puede intuir la relación entre la tracción necesaria para cada movimiento y el giro de los rotores.

De las ecuaciones expuestas se deduce que si se requiere un *pitch* mayor, el resultado es una tracción mayor sobre los motores 1 y 2, es decir, los motores delanteros. De ese modo la parte delantera del drone se elevará sobre la trasera provocando un ángulo positivo. En caso de un giro negativo, la tracción realizada por estos dos motores disminuirá, dando el resultado contrario.

De la misma forma, las ecuaciones (A.2) y (A.3) indican que para un giro *roll* serán los motores 2 y 3 los que alteren su velocidad de giro, pero en este caso en sentido negativo. Así, para un giro *roll* positivo, los motores que corresponden a la mitad izquierda del drone girarán más despacio, y en caso contrario más deprisa.

Los dos giros *pitch* y *roll* se explican en la figura A.4. El color rojo indica los motores que controlan el movimiento, y el grosor de la línea significa una mayor o menor tracción de los motores.

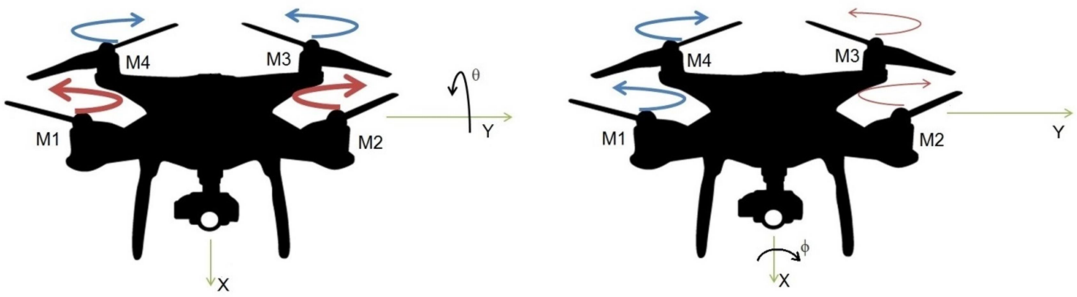

Figura A.4: Giro pitch y roll

El giro *yaw* se consigue con el cambio de velocidad de todos los motores. Si lo que se busca es que el drone gire en sentido positivo, los motores 1 y 3 recibirán mayor potencia y los restantes menor. La razón de este fenómeno radica en el sentido de giro de cada uno de los motores. Los dos primeros rotan en la dirección positiva de *yaw*, mientras que 2 y 4 giran en el sentido contrario. Una potencia mayor por parte de dos de ellos provoca un giro en el sentido contrario a la rotación de los motores, lo cual es debido al rozamiento con el aire. Esto se explica más fácilmente observando la figura A.5.

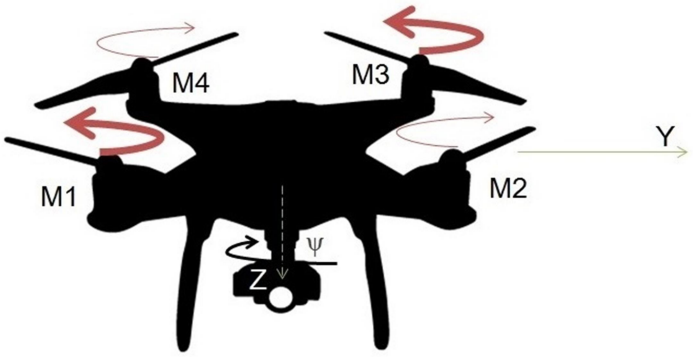

Figura A.5: Giro yaw

Por último, el hecho de que el drone se eleve o descienda es mucho más simple. Se aplica una tracción a todos los motores por igual, que deberá contrarrestar el efecto de la gravedad para una velocidad nula, y cuyo aumento o disminución provocará su ascenso o descenso.

Apéndice B

Entorno de simulación

Este apéndice se va a centrar en explicar más detalladamente el entorno diseñado por Peter Corke en *Simulink* para la simulación del comportamiento del drone, y los cambios realizados en el mismo para la realización del proyecto.

En la figura B.1 se expone nuevamente el diagrama de bloques.

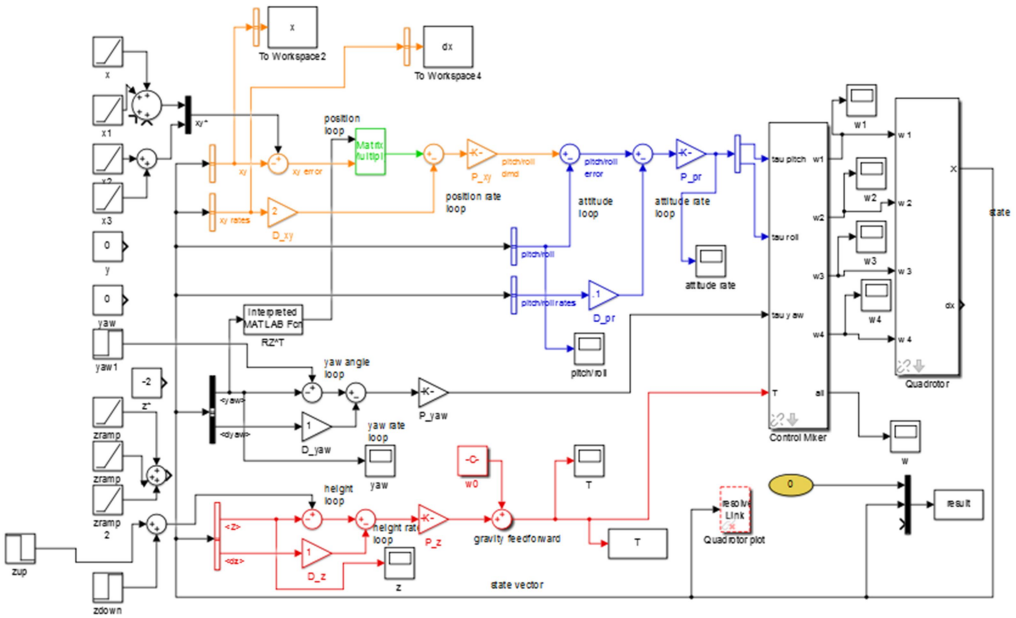

Figura B.1: Diagrama de bloques del control [11]

En este sistema se observan cuatro controladores, un bloque Mixer y un bloque Quadrotor. Se va a proceder a explicar cada uno de ellos.

Arriba a la izquierda se aprecia en color naranja el control de velocidad. Inicialmente, este era un control de posición cuya consigna consistía en una rampa de pendiente 0.2m/s. Sin embargo, la dificultad para medir de forma precisa la velocidad indica que una medida fiable de la posición, calculada como la integral de la primera,

sería incluso más difícil. Por ello, se ha optado por un control de velocidad en el que la variable realimentada sea más fiel a la realidad.

En la figura B.2 se observa de forma más precisa dicho control. La consigna no es una rampa como era inicialmente, sino un escalón de velocidad en el caso de x. Vx1 es un escalón cuya amplitud es la consigna de velocidad requerida y cuya duración es de 20s. El drone necesita un tiempo inicial para elevarse, así que se le resta durante los primeros 5s la misma amplitud, obteniendo un escalón que dura desde t=5 hasta t=20. Se calcula el error de velocidad realimentando dicha variable, y se utiliza la matriz de transformación dependiente del ángulo *yaw*. Este último paso no afecta, ya que se va a controlar que tanto el *roll* como el *yaw* sean cero. La segunda realimentación es de aceleración, que le da al control la parte derivada. Tanto x2 como x3 (que aportan la consigna en la dirección y) se han puesto a cero.

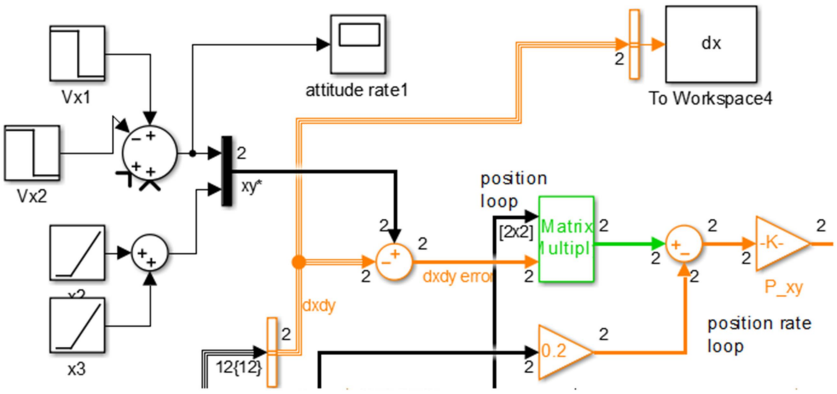

Figura B.2: Control de velocidad en simulación

El control de *pitch* se observa en la figura B.3. El control de velocidad aporta la consigna de *pitch* a este control, y del mismo modo se realimenta tanto el *pitch* como su velocidad angular, dando lugar a un control proporcional derivado. Las acciones de este regulador son dos tracciones que dependen del *pitch* y el *roll* deseados.

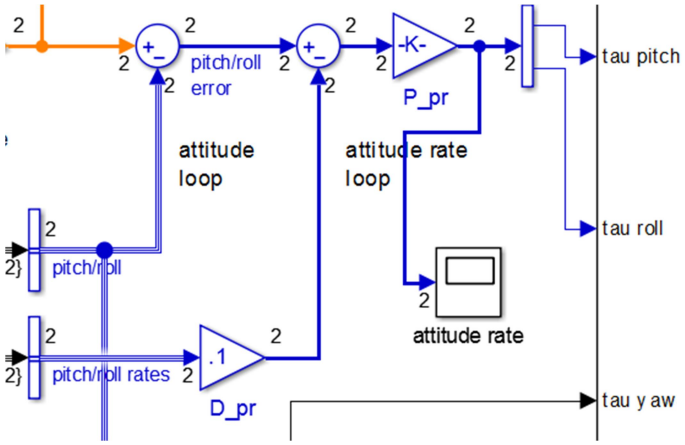

Figura B.3: Control de pitch en simulación

En cuanto al control de *yaw* y de altura, no han sido modificados. Esto es debido a que el propósito de este proyecto es el control del *pitch* y la velocidad. De esta forma, no merece la pena adentrarse demasiado en detalles. Es suficiente con saber que se ha dado una consigna nula a *yaw* para que el drone siga una línea recta en la dirección de x, y que los primeros 5 segundos es el control de altura el que se encarga de elevar el drone.

En el anexo A se ha explicado el funcionamiento de los motores dependiendo del movimiento que se pretenda realizar en el drone. El bloque Mixer realiza esa función de reparto de tracciones a cada uno de los motores. Las ecuaciones de dicho anexo son plasmadas en este bloque (figura B.4). Antes de enviar estas acciones al drone se saturan a un valor máximo.

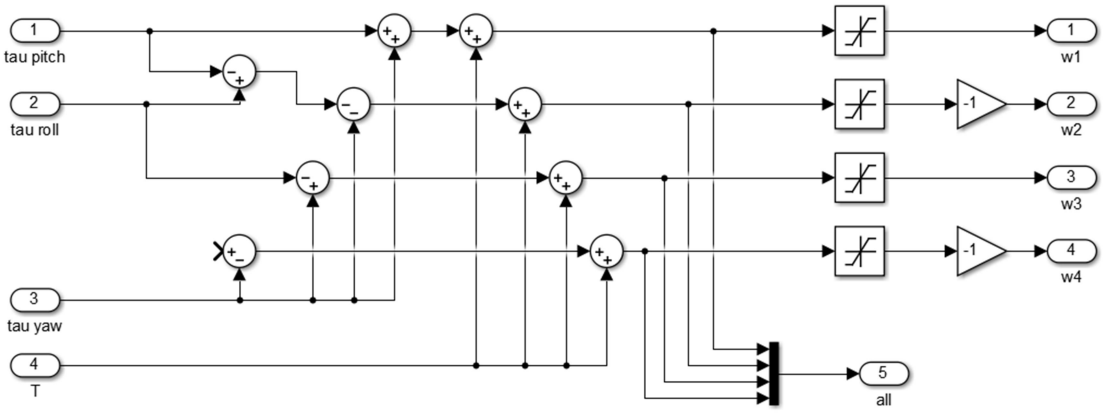

Figura B.4: Bloque mixer

Finalmente, el bloque quadrotor incluye el modelo real del drone, y tiene como salida las posiciones y velocidades tanto lineales como angulares.

Bibliografía

[1] A Brief History of Drones. URL http://www.iwm.org.uk/history/a-brief-history-of-drones

[2] Nothing Beats a Clean Signal. URL: http://www.qualtre.com/2015/10/nothing-beats-a-clean-signal

[3] Flame Wheel ARF KIT. URL: https://www.dji.com/flame-wheel-arf/feature

[4] Datasheet XBEE. URL: https://www.sparkfun.com/datasheets/Wireless/Zigbee/XBee-Datasheet.pdf

[5] Datasheet NAZA. URL: http://download.dji-innovations.com/downloads/naza-m%20lite/en/NAZA-M%20LITE_User_Manual_v1.00_en.pdf

[6] The commercial drones market. URL http://www.businessinsider.com/a-history-of-commercial-drones-2016-12

[7] Simulink. URL: https://es.mathworks.com/products/simulink.html

[8] Matlab. URL: https://es.mathworks.com/products/matlab.html

[9] Code Composer Studio. URL: http://www.ti.com/tool/ccstudio

[10] Futaba Radio System. URL: http://www.futabarc.com/systems/futk6000.html

[11] Corke, P.: Robotics, Vision and Control. Fundamental Algorithms in MATLAB®. Second, Completely Revised, Extended and Updated Edition. Springer Tracts in Advanced Robo-itcs, vol. 118 (2017)

[12] Beatriz Frisón Alegre: Modelado y control de un helicóptero de cuatro rotores. *Universidad de Zaragoza, CPS, 2009.*

[13] Kilian Nicolás Pascual Latorre: Planificación de misiones y navegación autónoma de un quadcopter. *Universidad de Zaragoza, EINA, 2016.*