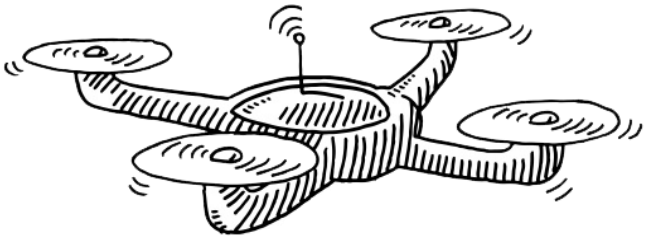
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

DISEÑO E IMPLEMENTACIÓN DE UN CUADRICÓPTERO BASADO EN EL MICROCONTROLADOR STM32F4

Grado en Ingeniería Electrónica Industrial y Automática



Realizado por:

Aleks Emilov Goranov

Dirigido por:

Carlos Pascual Domínguez Montagud

Fecha: 06/2016

Resumen

El objetivo de este proyecto es construir un cuadricóptero y realizar pruebas de vuelo. Esto implica el diseño e implementación de un sistema de control para conseguir la estabilidad y variación de orientación. Para ello, se utiliza el microcontrolador STM32F4 como unidad de procesamiento central junto a los sensores encargados de la unidad de medición inercial. Previamente se recurre a una selección de componentes y montaje de los mismos para definir el sistema final. Posteriormente se describe la cinemática y la dinámica del cuadricóptero para poder realizan simulaciones utilizando Simulink y Matlab con la finalidad de predecir el comportamiento de la aeronave antes de ponerla en funcionamiento. Más adelante se ha procedido a la programación del microcontrolador, esto incluye la configuración de puertos y pines, diseño de funciones, uso de librerías, etc. También se incluye el algoritmo de control siendo este una acción de control formada por tres controladores PID aplicada a cada uno de los rotores. Finalmente se realizan pruebas de vuelo, se estiman las ganancias de los tres PIDs y a partir de los resultados se han incluido una serie de mejoras para definir el comportamiento final del cuadricóptero.

Palabras clave: cuadricóptero, sistema de control, programación de microcontroladores, STM32F4

Abstract

The objective of this project is to build a quadcopter and do flight tests. This involves the design and implementation of control system for stability and variation of the orientation. To do this, the STM32F4 microcontroller is used as central processing unit beside the sensors responsible for the inertial measurement unit. Previously is resorted to a selection of components and assembly thereof to define the final system. Subsequently, the kinematics and dynamics of quadcopter are described to perform simulations using Simulink and Matlab in order to predict the behavior of the aircraft before putting it into operation. Later we proceeded to program the microcontroller, this includes configuration of ports and pins, design of functions, use of libraries, et cetera. Also, the control algorithm is included, which is basically a control action that consists of three PID controllers applied to each one of the rotors. Finally, flight tests are performed, the gains of the three PIDs are estimated and from the results, a number of improvements are included to define the final performance of the quadcopter.

Key words: quadcopter, control system, microcontroller programming, STM32F4

ÍNDICE GENERAL

MEMORIA

PRESUPUESTO

PLANOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1. INTRODUCCIÓN

"Una vez hayas probado el vuelo siempre caminarás por la Tierra con la vista mirando al Cielo, porque ya has estado allí y allí siempre desearás volver."

Wilbur Wright (1867-1948)

Un vehículo aéreo no tripulado (VANT) se define como una aeronave que vuela sin tripulación, capaz de mantener de manera autónoma un nivel de vuelo controlado y sostenido. Últimamente, los que más destacan son los cuadricópteros, un tipo de helicóptero propulsado por cuatro rotores. La popularidad de los cuadricópteros se debe a las características de despegue, vuelo estacionario y aterrizaje vertical que se pueden alcanzar.

Este trabajo se centra en el desarrollo de un sistema de control para conseguir la estabilidad y manejo del mismo mediante la variación de orientación. De modo que, a través de un control remoto se puedan conseguir todo tipo de movimientos, aunque siempre atendiendo a las limitaciones físicas o del propio sistema. Para llevar a cabo este trabajo, se ha estructurado de la siguiente manera:

En primer lugar, se ha realizado un estudio para conocer cuáles son los componentes que forman el cuadricóptero; una vez determinados dichos componentes se han seleccionado productos existentes del mercado. Para ello, nos hemos guiado por algunos factores como podrían ser el coste, calidad, disponibilidad y lo más importante las prestaciones o características que ofrece.

En segundo lugar, se describe el comportamiento dinámico del cuadricóptero mediante la modelación teórica y experimentar. Gracias a un programa de código libre para Matlab se ha realizado la simulación. Dicho programa dispone de todas las ecuaciones que describen el comportamiento dinámico, así como de herramientas para especificar los parámetros propios del cuadricóptero permitiendo así simplificar los cálculos.

En tercer lugar, se diseñan los tres controladores PID para controlar las variaciones de ángulo sobre los ejes X, Y, Z. El controlador se encarga de comparar el error, diferencia entre los valores obtenidos del receptor y del sensor, y ejecutar la acción de control sobre cada uno de los rotores para conseguir que ese error sea nulo.

En cuarto lugar, se procede a la programación del microcontrolador STM32F4. Esta tarea consiste en, configurar todos los puertos y pines e implementar el código en C encargado de extraer información, filtrar señales, realizar el control, etc.

Por último, se comprueba el correcto funcionamiento mediante el análisis de los resultados obtenidos en las etapas anteriores, y se proponen nuevas soluciones con el fin de mejorar la plataforma del cuadricóptero para trabajos futuros.

1.1. Objetivos

Los objetivos del presente proyecto radican en la implementación de un sistema de control para la estabilidad y orientación de un cuadricóptero. Para ello, se han de cumplir los siguientes objetivos:

- Seleccionar los componentes electrónicos que construyen el cuadricóptero.
- Analizar el comportamiento dinámico de la aeronave mediante el modelado físico de su sistema.
- Diseñar tres controladores para cada uno de los ejes X, Y, Z con el fin de controlar variaciones de ángulos de orientación sobre dichos ejes.
- Implementar el algoritmo de control para cada uno de los rotores.
- Realizar todas las conexiones entre componentes, así como, configurar, programar el microcontrolador y verificar el correcto funcionamiento durante las pruebas de vuelo.

1.2. Justificación

Con el avance de la tecnología y los precios reducidos de componentes electrónicos, cada vez existe una mayor tendencia a que los aficionados del mundo de la electrónica desarrollen sus propios dispositivos. Entre ellos, los cuadricópteros están siendo cada vez más sujeto de estudio por sus diversas aplicaciones y sobre todo por los conocimientos que aporta. Conocimiento que abarcan muchas ramas de la ingeniería como automática, electrónica, informática, aeronáutica e incluso diseño. Los cuadricópteros son dispositivos exigentes, de modo que, necesitan un sistema que se encarga de la mayor parte de tareas, que en general hablamos de microcontroladores. A razón de esto, uno de los aspectos más importantes sin lugar a duda es la configuración y programación del microcontrolador. Así mismo, veremos que el comportamiento del cuadricóptero depende de las técnicas de control y los métodos empleados, aunque es verdad que, por muy buenas que sean las técnicas de control, si no se dispone de sensores precisos, a veces, resulta complicado llegar al resultado deseado.

Con este proyecto se pretende poner a prueba los conocimiento adquiridos en el grado de Ingeniería Electrónica Industrial y Automática y además profundizar en diversas materias relacionadas con el presente proyecto para conseguir los objetivos previamente establecidos.

CAPÍTULO 2. SELECCIÓN DEL HARDWARE

"El experimentador que no sabe lo que está buscando no comprenderá lo que encuentra."

Claude Bernard (1813-1878)

En este apartado se describen e indican los componentes que forman parte del cuadricóptero, así como, algunas de las consideraciones que se han tenido en cuenta durante el proceso de selección del hardware.

Las decisiones que se tomen a la hora de seleccionar los componentes son muy importantes ya que influyen en el sistema final, es decir, los componentes son los que definen el sistema que pretendemos controlar.

Es importante mencionar que, en este apartado, sólo se hablará de los componentes necesarios para resolver el problema planteado y no se hablará de alternativas ni de otros componentes destinados a otros fines como podrían ser por ejemplo un módulo GPS, cámara, GPRS, etc.

2.1. Estructura

Es la parte más importante porque en gran medida el resto de componentes dependen de esta, por ejemplo, si el tamaño de la estructura es grande significará que necesitaremos unos motores más potentes y más grandes. Por esta misma razón, dependiendo de los motores elegidos, tendremos que escoger las hélices y ESCs más adecuados.

Figura 1. *Quanum Falcon Billet Block FPV Racing Frame.*

De modo que, finalmente se ha decidido implementar un cuadricóptero pequeño con estructura de diámetro 250 mm, para ello se ha elegido la estructura Quanum Falcón Billet (figura 1). Es de fibra de carbono con un espesor de 6 mm, de alta resistencia, muy ligera y cuenta con un montón de espacios y ranuras para colocar la electrónica y otros componentes. Además, viene con una PCB de distribución de alimentación y dos placas con LEDs para la iluminación delantera y trasera.

Especificaciones	
Parámetro	**Descripción**
Tamaño de los motores	1704 - 2204
Batería	1300-2200 mAh (3s LiPo)
ESCs	7 -20 A
Hélices	5 – 5.5 pulgadas

Tabla 1. *Especificaciones de la estructura Quanum*.

2.2. Motores DC sin escobillas y hélices

Actualmente en el mercado de los helicópteros de radio control podemos encontrar una amplia gama de motores y hélices como medio de propulsión. En primer lugar, es importante mencionar que existen dos tipos de motores: eléctricos y de combustión. Los motores de combustión ofrecen una buena relación potencia-peso y suponen una adecuada alternativa para aeronaves de tamaño grande. Sin embrago, este proyecto se centra en el desarrollo y diseño de un cuadricóptero de un tamaño pequeño de modo que los motores de combustión no son buena opción.

En cuanto a los motores eléctricos podemos distinguir o clasificarlos en dos categorías principales:

- *Motores eléctricos con escobillas*: son considerados más simple, son fácil de controlar y en general son baratos. Sin embargo, su eficiencia es inferior, su vida útil es menor y es complicado conseguir altas potencias con motores de tamaño reducido.
- *Motores eléctricos sin escobillas*: son más eficiente, su vida útil es mayor y se consiguen altas potencias con motores de tamaño reducido. En general son más caros y necesitan un circuito eléctrico que los maneje.

Dicho lo anterior, finalmente se han escogido motores eléctricos sin escobillas por las prestaciones que ofrecen, aunque suponga un costo superior.

Figura 2. *Motor eléctrico sin escobillas DYS BE1806.*

Los motores se han elegido en base a tres criterios. En primer lugar, el precio, al necesitar cuatro motores el precio final podría resultar excesivo. El segundo criterio se corresponde a las revoluciones por minuto que el motor debe girar por voltio aplicado sin carga, llamado factor de potencia (Kv), el cual se ha establecido en el valor 2300 Kv. Finalmente, el último criterio es el voltaje con el que operan los motores siendo este de 11,1 V correspondiendo a una batería de polímero de litio de tres celdas. En la tabla 1 se detallan las especificaciones del motor elegido.

Especificaciones	
Parámetro	**Descripción**
Kv	2300 Kv
Peso	18 g
Potencia máxima	84,4 W
Accesorios	Si

Tabla 2. *Especificaciones del motor BYS BE1806.*

Por otro lado, en cuanto a las hélices, se han elegido unas de diámetro de 5 pulgadas y en concreto unas de la marca Gemfan. Las hélices de Gemfan son conocidas por su durabilidad, gran equilibrio y bajo coste a cambio de alta calidad. Las hélices se venden en un pack de dos, una para giros en sentido horario y la otra para giros en sentido anti-horario.

Figura 3. *Hélices Gemfan 5x3 Black*.

2.3. Controladores electrónicos de velocidad (ESC)

Los controladores electrónicos de velocidad son los encargados de manejar los motores, a través de un canal receptor aceptan una señal de entrada tipo PWM de entre 50 y 450 Hz y dependiendo de la longitud del ancho de pulso (entre 1 ms parado y 2 ms a máxima potencia) entregarán más o menos potencia al motor.

Figura 4. *Controladores electrónicos de velocidad DYS 20A.*

Los motores consumen bastante energía, por ese motivo, los ESC deben ser capaces de suministrar la corriente máxima exigida por los motores. Los motores DYS BE1806 dependiendo de las hélices que utilicen y tipo de batería pueden estar consumiendo máximo 10 amperios. De modo que, los ESC deben ser capaces de suministrar mínimo 10 amperios, aunque siempre es aconsejable dejar un margen para cubrir posibles picos de corriente. Dicho lo anterior, se han escogido los controladores electrónicos de velocidad de la marca DYS de 20 A.

Especificaciones	
Parámetro	Descripción
Voltaje de entrada	2 – 4s LiPo
Frecuencia de señal	20-500 Hz
Frecuencia de salida PWM	18 kHz
Firmware	BlHeli
Peso	7,6 g

Tabla 3. *Especificaciones del ESC DYS 20A.*

2.4. Control remoto

En cuanto al control remoto, las dos tecnologías más utilizadas son la comunicación mediante señales infrarrojas y la comunicación por medio de señales de radio. Actualmente la más utilizada para aeronaves es mediante señales de radio y será la que emplearemos en este proyecto.

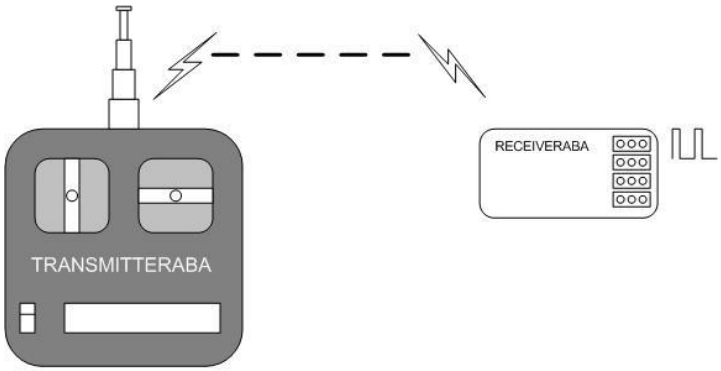


Figura 5. *Comunicación por radio control.*

El emisor es el aparato que se encarga de hacer de interfaz entre el piloto y el mando de la aeronave. El funcionamiento, de este aparato consiste en interpretar los movimientos que ejerce el usuario sobre los "*sticks*", pulsadores o interruptores y convertirlos en una señal de radio, para así ser emitida a la aeronave.

Hay diferentes sistemas de emisión en AM, FM y 2.4 GHz y diferentes métodos de codificación PCM y PPM. Actualmente las emisoras de radio control se han generalizado en la emisión de la frecuencia 2.4Ghz, dicha frecuencia funciona totalmente diferente a las antiguas que utilizaban un cristal con una frecuencia fija, estas más modernas emiten en una banda más ancha llamada DSS (Distribución dinámica de espectro) también utilizada por el estándar Bluetooth o Wi-Fi.

Por otro lado, el receptor es un pequeño aparato alojado en la aeronave que se encarga de descodificar las señales que recibe del radio-mando y convertirlos en impulsos o señales PWM que posteriormente serán interpretados por el microcontrolador.

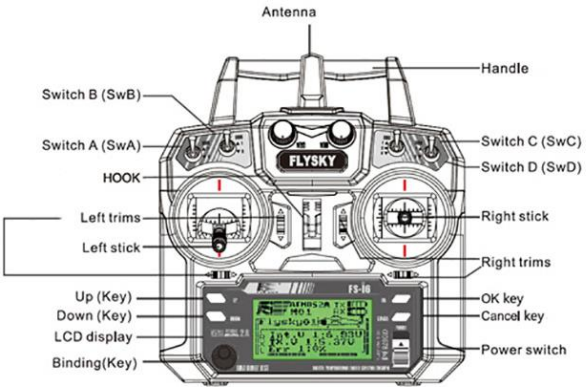


Figura 6. *Emisor RC marca FLYSKY FS-i6.*

Para este proyecto se ha escogido el emisor y receptor RC de la marca FLYSKY y en concreto el modelo FS-i6. Incluye un receptor de seis canales, más que suficiente para este proyecto. En caso de necesitar más canales, siempre se puede utilizar cualquiera otro compatible. En la tabla 3 se detallan sus especificaciones.

Características			
Canales:	6 Canales	**Tipo de modelo:**	Planeador / Heli / Avión
Rango de RF:	2.40-2.48GHz	**Ancho de banda:**	500KHz
Banda:	142	**Energía RF:**	Menos de 20dBm
Sistema 2.4ghz:	AFHDS 2A y AFHDS	**Code Type:**	GFSK
Sensibilidad:	1024	**Baja Tensión Advertencia:**	menos de 4.2V
Puerto DSC:	PS2; salida: PPM	**Puerto del cargador:**	No
Longitud ANT:	26mm * 2 (doble antena)	**Peso:**	392g
Poder:	6V 1.5AA*4	**Modo de visualización:**	Transflectiva tipo positivo STN, 128 * 64 puntos VA73 matriz * 39mm, luz de fondo blanco.
Tamaño:	174x89x190mm	**On-line al día:**	sí
Certificado:	CE0678,FCC	**Memorias de modelos:**	20

Tabla 4. *Especificaciones del emisor y receptor RC marca FLYSKY FS-i6.*

2.5. Sensores

Uno de los componentes más importantes para la aviónica en un vehículo aéreo no tripulado es la unidad de medición inercial (IMU); por medio de esta, es posible conocer el estado de la velocidad angular y la aceleración en los ejes X, Y, Z. La unidad de medición inercial provee la funcionalidad de tomar la información entregada por los sensores, ya sea un giroscopio, acelerómetro, barómetro o una brújula digital, para luego interpretar dicha información pudiendo estimar la posición u orientación y a partir de ahí realizar las acciones definidas por el usuario.

En cuanto al acelerómetro, debemos saber que es un dispositivo electromecánico que mide las fuerzas de aceleración. Estas fuerzas pueden ser estáticas, como la fuerza constante de la gravedad que tira hacia el centro de la Tierra, o podrían ser dinámicas, causadas por el movimiento o vibración del acelerómetro. A la hora de elegir el más adecuado los parámetros en los que nos debemos fijar son, por una parte, el rango de medición, o rango de escala completa, siendo la velocidad angular máxima que el giroscopio puede leer y por otra parte, la sensibilidad que se mide en mV por grado por segundo (mV / ° / s).

Un giróscopo es un sensor que se utiliza para medir la velocidad angular de un cuerpo. Las unidades de velocidad angular se miden en grados por segundo (° / s) o revoluciones por segundo (RPS). Cada canal del giroscopio mide la rotación alrededor de uno de los ejes. Por ejemplo, un giroscopio de 3 ejes medirá la rotación alrededor de los ejes X, Y, Z. La sensibilidad y rango de medición también son factores propios para este sensor.

2.5.1. MPU6050

Para este proyecto se ha utilizado el módulo GY-86 el cual incorpora tres chips: MPU6050, MS5611 (barómetro), HMC5883L (magnetómetro). De los tres chips, sólo utilizaremos el MPU6050. Este chip incluye dos sensores; un acelerómetro y un giróscopo (ambos de 3 ejes) y fabricados con tecnología MEMS. El MPU6500 se caracteriza por poseer convertidores analógico-digitales para cada canal. Esto quiere decir que, seremos capaces de capturar mediciones sobre los ejes X, Y, Z al mismo tiempo.

Figura 7. *Módulo Gy-86 incorpora 4 sensores. Acelerómetro, giroscopio, barómetro y brújula digital.*

Su peso es de tan solo 12 gramos, un tamaño reducido 2.2x1.7 cm y emplea una interfaz de comunicación por el bus I2C. Una de las limitaciones de una interfaz digital es la velocidad de muestreo máxima. El I2C tiene una frecuencia de muestreo máxima de 400 Hz que para esta aplicación es más que suficiente.

Especificaciones			
Acelerómetro		**Giroscopio**	
Rango de medición:	Programable ±2g, ±4g, ±8g o ±16g	**Rango de medición:**	Programable ±250, ±500, ±1000, o ±2000 °/seg
Salida:	Digital (ADC 16bit)	**Salida:**	Digital (ADC 16bit)
Uso de corriente:	500 uA	**Uso de corriente:**	3.6 mA
		Alimentación:	5 o 3.3 V

Tabla 5. *Especificaciones del sensor MPU6050.*

Por último, hay que tener en cuenta que los campos eléctricos o magnéticos pueden inferir en el correcto funcionamiento del sensor, por lo cual, se recomienda en la medida de lo posible, colocarlo en un sitio alejado de cualquier elemento que pueda generar alguno de estos campos.

2.6. Batería

Uno de los principales problemas asociados con la autonomía de un vehículo aéreo, es su alimentación eléctrica debido al alto consumo de energía de los motores. En general, sólo seremos capaces de mantener el vuelo durante unos cuantos minutos. A la hora de escoger la batería más adecuada, nos basaremos en dos criterios:

- La tensión de salida: viene determinada por el número de celdas que constituyen la batería.
- La capacidad de carga: es la capacidad de carga que puede almacenar el elemento o capacidad del acumulador, se mide en amperios-hora (Ah)

Para este proyecto se ha utilizado un batería de tipo LiPo al ser más ligera y caracterizarse por una mayor resistencia a perder carga con los ciclos de trabajo (mantienen su carga nominal más tiempo).

Figura 8. *Batería MultiStar Racer Series 1400 mAh.*

La batería en concreto, es de la marca MultiStar tiene una capacidad de carga de 1400 mAh, posee 3 celdas lo que equivale a una tensión de salida de 11.1 voltios y la capacidad de descarga es de 40C lo que equivale a una descarga de 56 amperios por hora, es decir, sabiendo que la batería es de 1400 mAh esto quiere decir que podríamos descargar la batería en 1.5 minutos. Hay que tener **mucha precaución** con ese tipo de baterías ya que, si se sobre-descargan o si se sobrecargan podrían hincharse y en el peor caso inflamarse.

2.7. Otros

Además de los componentes anteriormente citados, existen otros, tales como, un regulador de tensión para la alimentación del microcontrolador y el receptor. Se necesitan 5 voltios estables, el regulador de tensión suministra esta tensión a partir de los 11.1 voltios de la batería LiPo.

Figura 9. *Regulador de tensión YwRobot 5 o 3.3 V.*

Se dispone también de un medidor de voltaje, que como su nombre indica es capaz de medir la tensión de salida. El que se ha elegido en concreto, puede medir la tensión de salida para baterías tipo LiPo que tengan entre 1 y 8 celdas (1-8s).

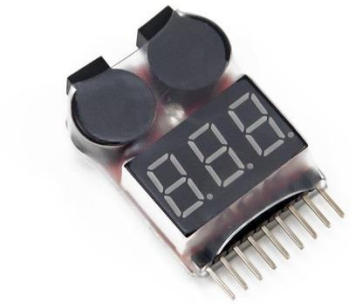


Figura 10. *Voltage teaser 1-8s.*

Por último, muchos cables para efectuar las conexiones entre componentes, tubos termo retráctiles para aislar cables y mallas de nylon para agrupar cables.

Figura 11. *Material diverso: cables, tubos termo retráctiles y mallas de nylon.*

CAPÍTULO 3. DISEÑO Y MONTAJE DEL CUADRICÓPTERO

"El arte es "yo"; la ciencia es nosotros."
Claude Bernard (1813-1878)

En este apartado se describe el procedimiento completo seguido para unir todas las piezas y obtener el modelo físico final del cuadricóptero. Esto incluye diseño de piezas, soldadura, conexiones entre componentes etc.

3.1. Diseño CAD

Dado que, algunos componentes necesitan una estructura que los sujete sobre la base del cuadricóptero y para obtener un diseño más compacto, único y personal, se han diseñado piezas que cumplan con esta función. El diseño de esas piezas infiere en algunos factores como el diseño final, el peso total o incluso en la aerodinámica y, por lo tanto, son consideraciones que se ha tenido en cuenta a la hora de diseñar. El programa utilizado es SolidWorks 2015 siendo un programa bastante intuitivo y fácil de manejar.

Figura 12. *Logotipo de SolidWorks.*

La primera pieza diseñada es un soporte para la batería. Esta pieza está diseñada para encajar entre la separación de las dos bases de la estructura del cuadricóptero, esto lo veremos en el proceso de montaje. También veremos que hay muchas conexiones de cables y gracias a esta pieza la mayoría se ocultan. El diseño es agradable y aerodinámico; el peso de esta pieza es de 38 g.

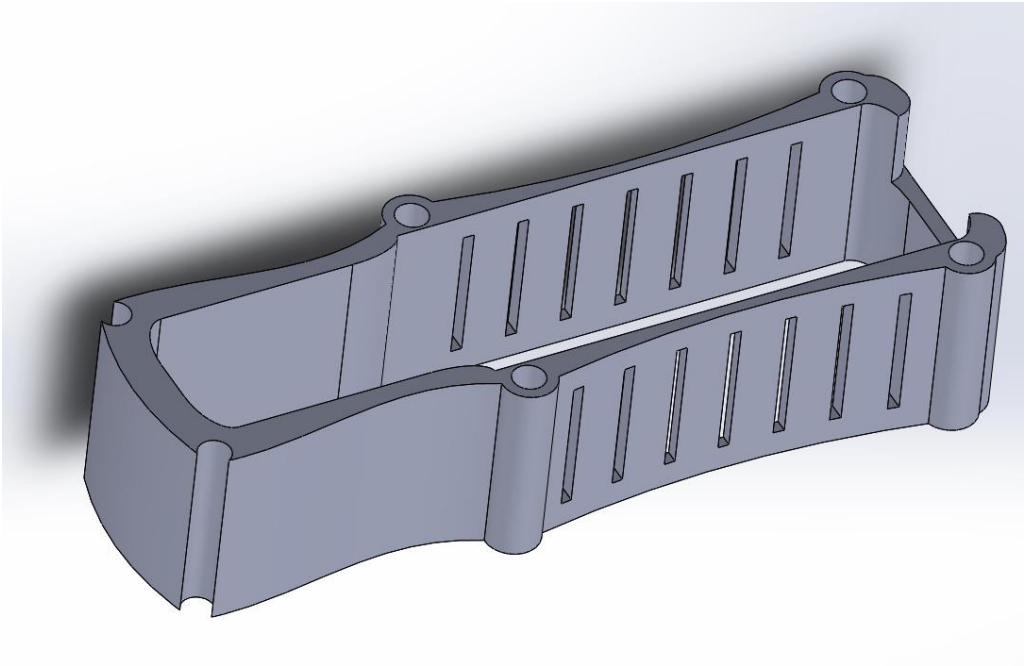


Figura 13. *Soporte batería.*

La segunda pieza diseñada es un soporte para el microcontrolador y el regulador de tensión. Las conexiones de cables con los *pines* del microcontrolador se realizan por la parte inferior de la placa. Por lo tanto, a la hora de diseñar esta pieza se han dejado espacios abiertos por donde puedan pasen los cables y una profundidad suficiente entre la placa y la base de esta pieza. El tamaño es lo más reducido posible teniendo en cuenta las limitaciones del tamaño del microcontrolador y del regulador de tensión. El peso de esta pieza es de 46 g.

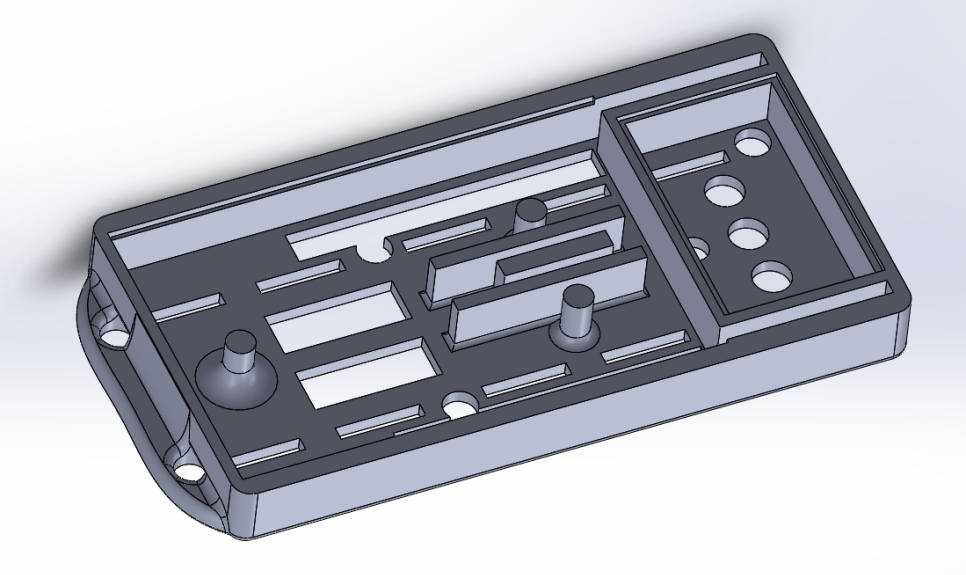


Figura 14. *Soporte para el microcontrolador y el regulador de tensión.*

La tercera pieza es un soporte para el receptor del mando y el medidor de tensión. Algunas de las consideraciones que se han tenido en cuenta en el diseño, es que las hélices invaden una zona de esta pieza, y esa es la razón por la cual presenta unas curvaturas en el diseño, tratando evitar cualquier posibilidad de contacto. El peso total es de 43 g.

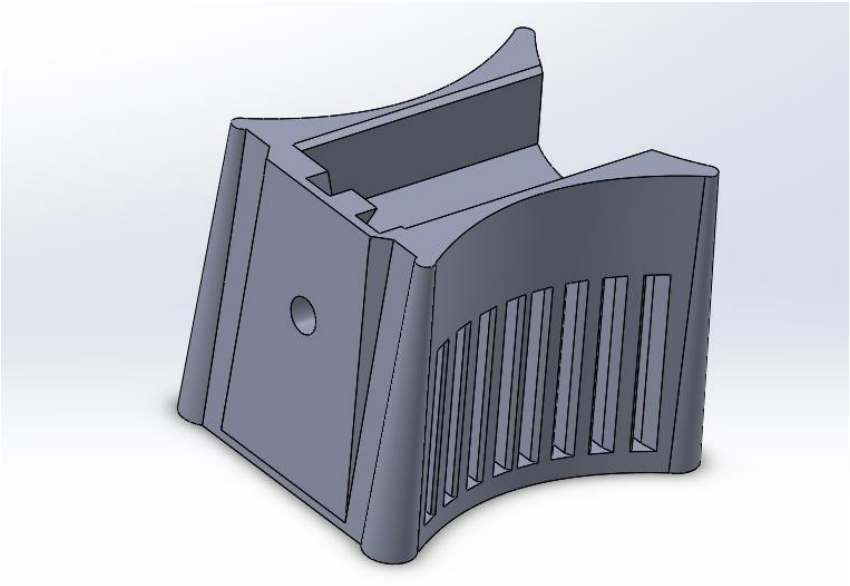


Figura 15. *Soporte para el receptor y el medidor de tensión.*

Finalmente, un soporte para el módulo gy-86, una pieza bastante simple pero muy útil. El módulo se debe colocar en el centro del cuadricóptero siendo la posición más adecuada. Se aprovechan los dos botones del microcontrolador y se utilizaran para sujetar esta pieza. El peso total de esta pieza es de tan solo 3 g.

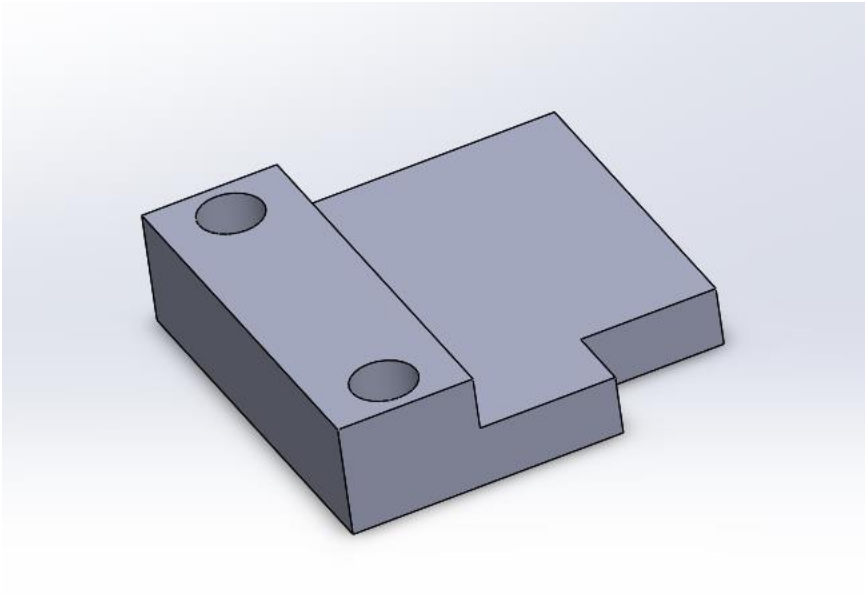


Figura 16. *Soporte para el módulo Gy-86.*

Una vez diseñadas las piezas, estas han sido imprimidas con una impresora 3D y utilizadas en el proceso de montaje. Para ello, estas piezas se han tenido que exportar en formato STL, formato que define la geometría de objetos 3D, excluyendo información como color, texturas o propiedades físicas. Utilizando un programa para impresión en 3D, se calculan los movimientos necesarios de cada capa para conseguir la pieza. De modo que, se entiende que se necesita una impresora 3D y además bien calibrada, para que las dimensiones sean las diseñadas y todo encaje. Los detalles del proceso de impresión no se explicarán en la presente memoria.

3.2. Montaje

Puesto que la mayor parte de los componentes son comprados, sólo tendremos que efectuar las conexiones correspondientes en cada caso. En cuanto a la parte de la electrónica, para una mejora visualización de cómo se han de conectar los módulos entre sí, nos guiaremos a partir del diagrama de conexiones (figura 17).

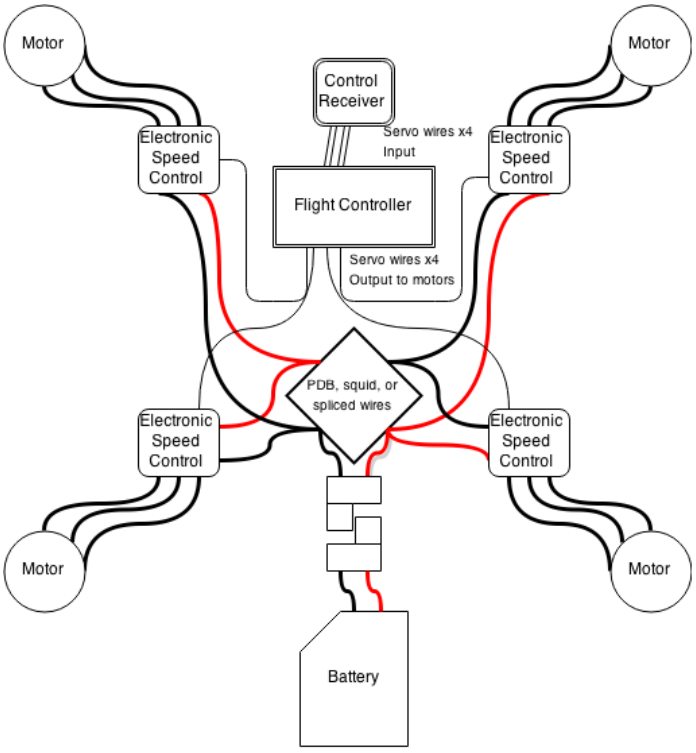


Figura 17. *Diagrama de conexiones.*

Por otro lado, en cuanto a la estructura del cuadricóptero, además de los marcos de fibra de carbón y las piezas diseñadas, necesitaremos una serie de componentes auxiliares, como varios tipos de tornillos para las uniones y unas cuantas varitas de aluminio para los espaciados entre componentes; todos ellos igualmente necesarios (figura 18).

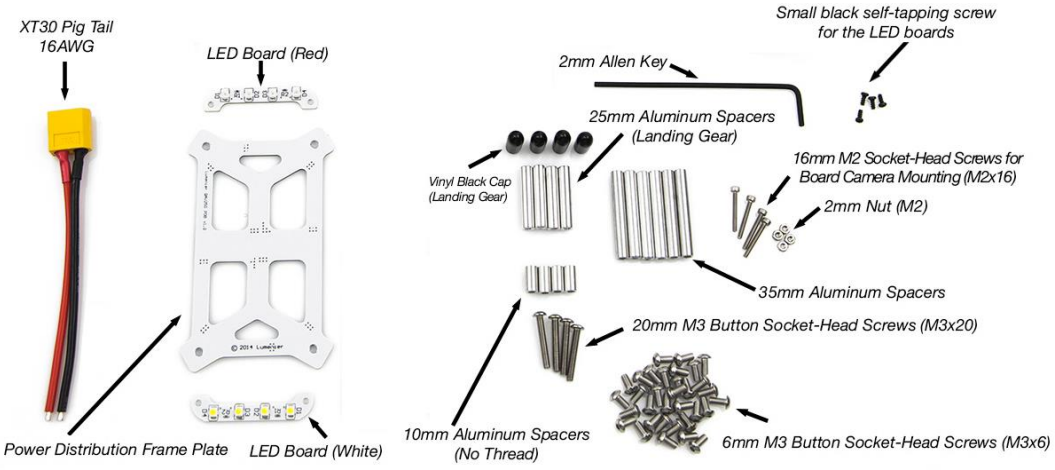


Figura 18. *Material auxiliar.*

A continuación se detalla el orden y los pasos seguidos en el montaje. El primer paso es unir la PCB de distribución de alimentación con las dos placas de leds. Para ello, se han soldado dos cables (positivo y negativo) en cada lado respetando la polaridad y haciendo las uniones correspondientes. También se ha soldado un cable que incluye el conector XT30 para posteriormente conectar la batería y otro cable bipolar para la alimentación del regulador de tensión.

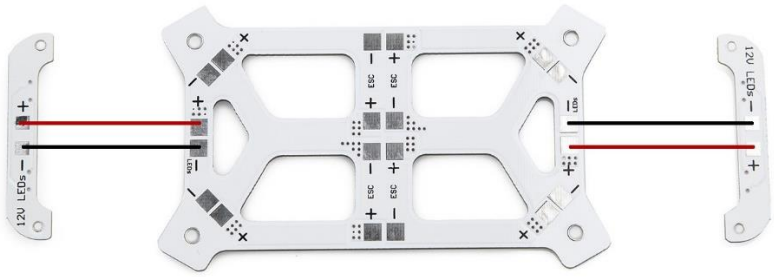


Figura 19. *Conexión entre PCBs.*

El segundo paso es soldar los ESCs sobre la placa PCB, para ello, en primer lugar debemos prepararlos. De manera que, se han cortado los dos cables de alimentación a la medida deseada y seguidamente, con la ayuda del soldador y aplicando estaño, se han soldado sobre la PCB respetando la polaridad. Esta operación se ha repetido para los cuatro ESCs.

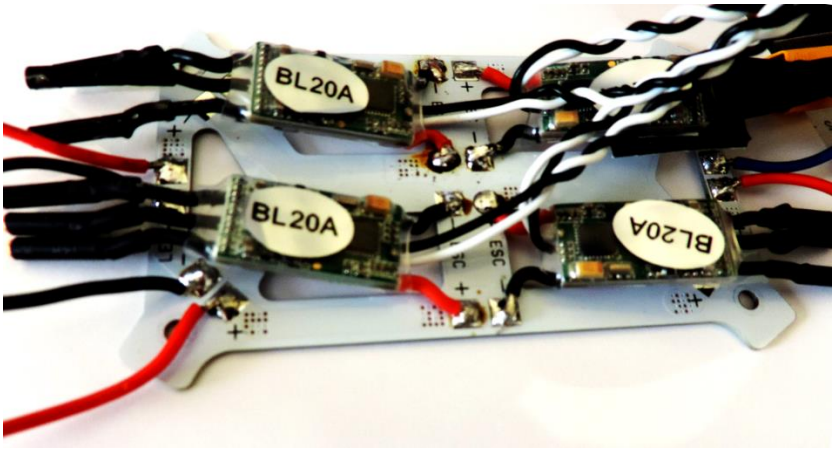


Figura 20. *Soldadura de los ESCs sobre la PCB de alimentación.*

El tercer paso es preparar las conexiones entre los ESCs y los motores. Al utilizar motor sin escobillas, contamos con 3 cables para las 3 fases. Dependiendo de cómo se conecten los cables con los ESCs, el motor girará en un sentido o en otro. Por lo tanto, realizar las conexiones de forma permanente no es buena solución, por ello, utilizaremos unos conectores macho-hembra. En caso de que cuando estemos realizando las pruebas, el motor girase en el sentido erróneo, simplemente intercambiaremos el orden de conexión para conseguir el sentido correcto. Los conectores se han soldado con los cables y luego se han aislado utilizando tubos termo retractiles. Finalmente, se han cubierto y agrupado con una malla de nylon dejando un aspecto final como el que se muestra en la figura 21.

Figura 21. *Motor BE1806, cables cubiertos con malla de nylon.*

Más adelante, se ha montado la PCB de distribución de alimentación, las dos placas con leds y los cuatro motores sobre el marco de fibra de carbono principal (inferior). Para fijar los motores y las placas leds se han utilizado los tornillos pequeños M1X5. Para la PCB de distribución de alimentación se han utilizado los tornillos M3x20 y los separadores de aluminio de 10 mm. El resultado se muestra en las figuras 22 y 23.

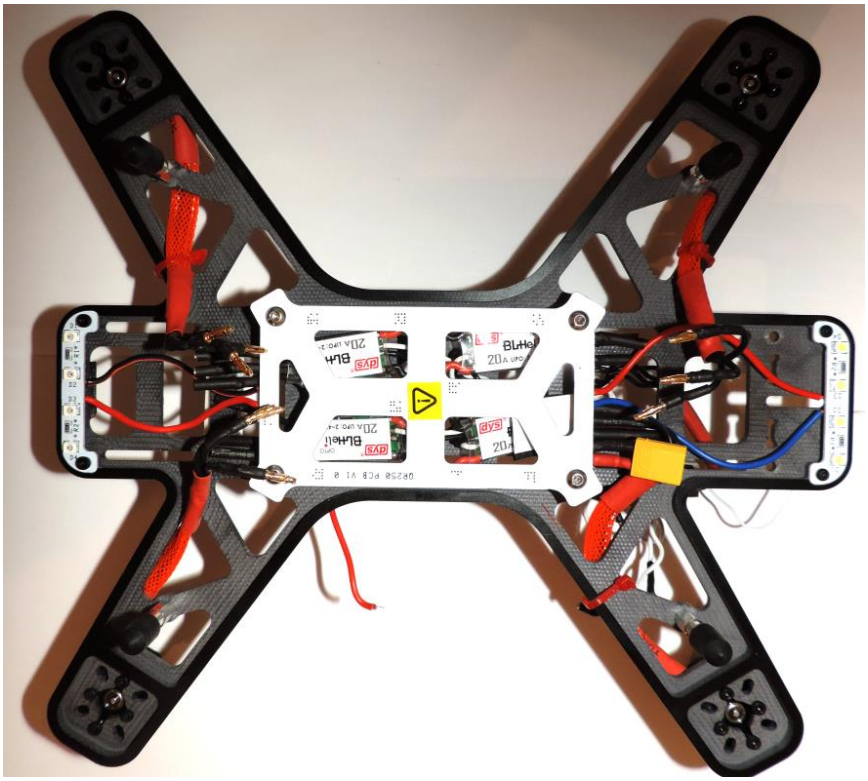


Figura 22. *Montaje PCB y motores parte inferior.*

Figura 23. *Montaje PCB y motores parte superior.*

A continuación, colocamos el soporte de la batería y sacamos los cables de señal de los ESCs y el cable de alimentación del regulador de tensión.

Figura 24. *Montaje PCB y motores parte superior.*

El siguiente paso consiste en colocar la batería y el soporte del receptor en el cual se ha montado previamente el receptor y el medidor de voltaje. También se han efectuado las conexiones de cables tanto del receptor como del medidor de tensión.

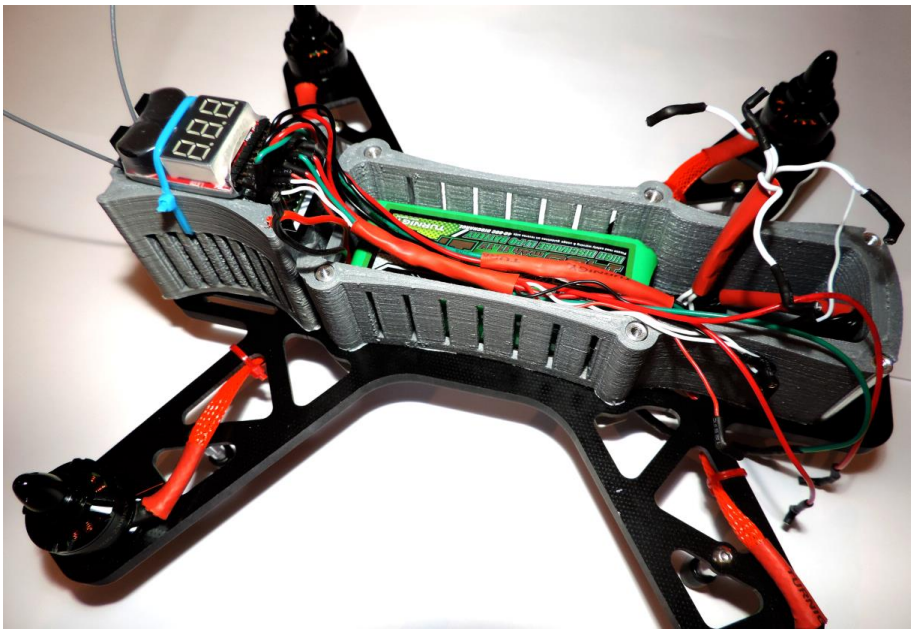


Figura 25. *Colocación batería, soporte receptor y medidor de tensión.*

En este paso se ha colocado la base superior junto al soporte del microcontrolador y el regulador de tensión. Para sujetar la base se han utilizado los tornillos M3x10 y las varitas de aluminio de 35 mm que actúan como tuercas para los tornillos. También se han sacado los cables y preparado para posteriormente conectarlos con el microcontrolador.

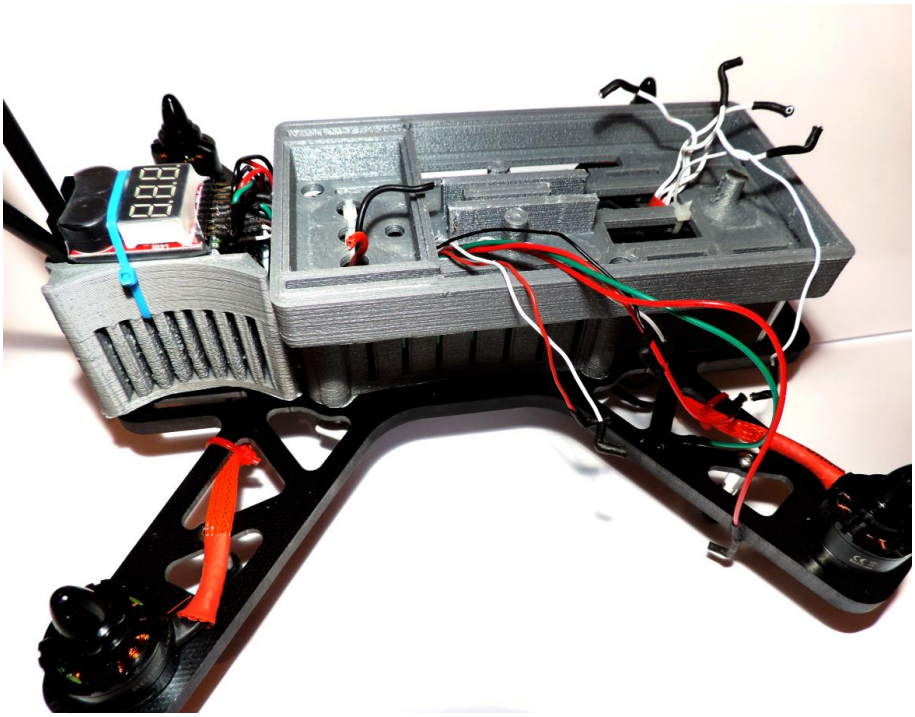


Figura 26. *Colocación del soporte del microcontrolador.*

Finalmente se realizan todas las conexiones de cables con el microcontrolador y el regulador de tensión. Se coloca el sensor sobre la pieza diseñada que a su vez se monta sobre el microcontrolador. El micro y el regulador de tensión se colocan en su sitio y con la ayuda de una pistola de silicona se fijan sobre la pieza de soporte. Cumplidos todos los pasos anteriores y algunos otros pequeños detalles, como colocación de las hélices o las dos antenas, el diseño final es el que se muestra en la figura 27.

Figura 27. *Colocación del resto de componentes. Diseño final.*

CAPÍTULO 4. MODELO MATEMÁTICO DEL CUADRICÓPTERO

"Es posible volar sin motores, pero no sin conocimiento y habilidad. Considero que es esto algo afortunado, para el hombre, por causa de su mayor intelecto, ya que es más razonable la esperanza de igualar a los pájaros en conocimiento, que igualar a la naturaleza en la perfección de su maquinaria."

Wilbur Wright (1867-1948)

El modelo matemático **describe la dinámica de un sistema y es necesario para poder simular y probar las técnicas de control**. El comportamiento de dicho sistema dependerá de los factores influyentes incluidos, es decir, de lo detallado que este el sistema. Sin embargo, no siempre es deseable incluirlos todos ya que, esto es muy exigente y requiere a menudo una gran cantidad de cálculos. Además, puede resultar complejo para el diseño de los controladores, por lo cual se debe obtener un modelo simplificado que describa de manera aproximada el comportamiento de la aeronave.

La estructura del cuadricóptero se muestra en la figura 28 incluyendo las correspondientes velocidades angulares, pares y fuerzas creadas por los cuatro rotores.

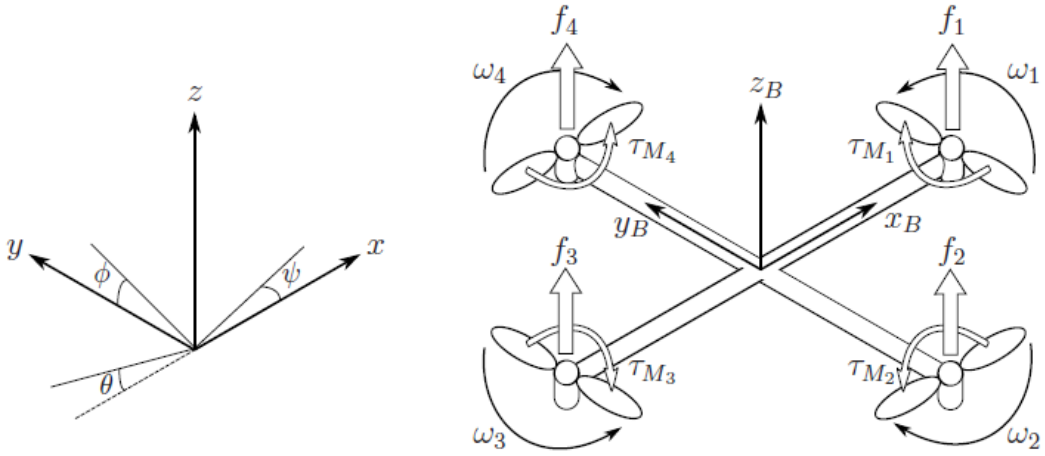


Figura 28. *Sistemas de referencia fijo y el sistema de referencia del cuadricóptero.*

La posición lineal absoluta del cuadricóptero se define en el sistema de referencia fijo ejes x, y, z con ξ. La posición angular, se define en el sistema de referencia fijo con tres ángulos de Euler η. El ángulo *pitch "* ϕ *"* determina la rotación del cuadricóptero sobre el eje Y. El ángulo *roll "* θ *"* determina la rotación sobre el eje X y el ángulo *yaw "* ψ *"* sobre el eje Z. El vector q contiene la posición lineal y angular.

$$\xi = \begin{bmatrix} x \\ y \\ z \end{bmatrix}, \qquad \eta = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix}, \qquad q = \begin{bmatrix} \xi \\ \eta \end{bmatrix} \qquad (1)$$

El origen del sistema de referencia del cuadricóptero está situado el centro de masas del mismo. Las velocidades lineales son determinadas por V_B y las velocidades angulares por υ.

$$V_B = \begin{bmatrix} v_{x,B} \\ v_{y,B} \\ v_{z,B} \end{bmatrix}, \qquad \upsilon = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \qquad (2)$$

La matriz de rotación del sistema de referencia del cuadricóptero respecto al sistema de referencia fijo viene determinada por la expresión R

$$R = \begin{bmatrix} C_\psi C_\theta & C_\psi S_\theta S_\phi - S_\psi C_\phi & C_\psi S_\theta C_\phi - S_\psi S_\phi \\ s_\psi C_\theta & C_\psi S_\theta S_\phi + S_\psi C_\phi & S_\psi S_\theta C_\phi - C_\psi S_\phi \\ -S_\theta & C_\theta S_\phi & C_\theta C_\phi \end{bmatrix} \qquad (3)$$

siendo $S_x = \sin(x)$ y $C_x = \cos(x)$. La matriz de rotación R es ortogonal, $R^{-1} = R^T$ siendo la matriz de rotación del sistema de referencia fijo al sistema de referencia del cuadricóptero.

La matriz de transformación[i] para las velocidades angulares del sistema de referencia fijo al sistema de referencia del cuadricóptero es W_η, y para el sistema de referencia del cuadricóptero al sistema de referencia fijo es ${W_\eta}^{-1}$

$$\dot{\eta} = {W_\eta}^{-1}\,\upsilon, \qquad \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & S_\phi T_\theta & C_\phi T_\theta \\ 0 & C_\phi & -S_\phi \\ 0 & S_\phi / C_\theta & C_\phi / C_\theta \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \qquad (4)$$

$$\upsilon = {W_\eta}^{-1}\,\dot{\eta}, \qquad \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & 0 & -S_\theta \\ 0 & C_\phi & C_\theta S_\phi \\ 0 & -S_\phi & C_\theta C_\phi \end{bmatrix} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \qquad (5)$$

donde $T_x = \tan(x)$. La matriz W_η es invertible si $\theta \neq (2k-1)\phi/2\,, (k\,\epsilon\,\mathrm{Z})$.

Se supone que el cuadricóptero tiene una estructura simétrica, los cuatro brazos están alineados respecto a los ejes X e Y. De modo que, la matriz de inercia es una matriz diagonal I en la que $I_{xx} = I_{yy}$.

$$I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \qquad (6)$$

La velocidad angular del rotor i, denotada con ω_i, crea una fuerza f_i en dirección del eje del rotor. La velocidad angular y la aceleración del rotor crean una fuerza de par τ_{M_i} alrededor del eje del rotor.

$$f_i = k{\omega_i}^2, \qquad \tau_{M_i} = b{w_i}^2 + I_M \dot{\omega}_\iota \qquad (7)$$

[i] T. S. Alderete, "Simulator aero model implementation." NASA Ames Research Center, Moffett Field, California, http://www.aviationsystemsdivision.arc.nasa.gov/publications/hitl/rtsim/Toms.pdf

en la que la constante de elevación es k, la constante de arrastre es b y el momento de inercia del rotor es I_M. En general, el efecto de $\dot{\omega}_\iota$ se considera pequeño y por lo tanto se omite.

La combinación de fuerzas del rotor crea una fuerza de empuje T en la dirección del eje Z del cuerpo. La fuerza de par τ_B está formada por los pares τ_ϕ, τ_θ y τ_ψ en la dirección correspondiente a los ángulos del sistema de referencia del cuadricóptero.

$$T = \sum_{i=1}^{4} f_i = k \sum_{i=1}^{4} {\omega_i}^2 , \qquad T^B = \begin{bmatrix} o \\ o \\ T \end{bmatrix} \tag{8}$$

$$\tau_B = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} l\,k\,(-\omega_2^2 + \omega_4^2) \\ l\,k\,(-\omega_1^2 + \omega_3^2 \\ \sum_{i=1}^{4} \tau_{M_i} \end{bmatrix} \tag{9}$$

donde l es la distancia entre el rotor y el centro de masas del cuadricóptero. Por lo tanto, el movimiento de *roll* se consigue disminuyendo la velocidad del segundo rotor y aumentando la velocidad del cuarto rotor. Del mismo modo, el movimiento de *pitch* se consigue disminuyendo la velocidad del primer rotor y aumentando la velocidad del cuarto rotor. Por último, el movimiento *yaw* se consigue aumentando la velocidad angular de los dos rotores opuestas o disminuyendo la de los otros dos opuestos.

4.1. Ecuaciones de Newton-Euler

El cuadricóptero se considera un cuerpo rígido y por lo tanto las ecuaciones de Newton-Euler se pueden utilizar para describir su dinámica. En el sistema de referencia del cuadricóptero las fuerzas requeridas para la aceleración de la masa $m\dot{V}_B$ y la fuerza centrifuga $\upsilon \times (m\,V_B)$ son iguales a la gravedad $R^T G$ y la fuerza de empuje total de los cuatro rotores T_B.

$$m\dot{V}_B + \upsilon \times (m\,V_B) = R^T G + T_B \tag{10}$$

En el sistema de referencia fijo, la fuerza centrífuga es anulada. Por lo tanto, sólo la fuerza de gravedad, la magnitud y la dirección de empuje contribuyen a la aceleración del cuadricóptero.

$$m\ddot{\xi} = G + RT_B \tag{11}$$

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = -g \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} + \frac{T}{m} \begin{bmatrix} C_\psi S_\theta C_\phi - S_\psi S_\phi \\ S_\psi S_\theta C_\phi - C_\psi S_\phi \\ C_\theta C_\phi \end{bmatrix}$$

En el sistema de referencia del cuadricóptero, la aceleración angular de la inercia $I\dot{v}$, las fuerzas centrípetas $v \times (Iv)$ y las fuerzas giroscópicas Γ son iguales al par externo τ

$$I\dot{v} + v \times (Iv) + \Gamma = \tau \qquad (12)$$

$$\dot{v} = I^{-1}\left(-\begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} I_{xx}\, p \\ I_{yy}\, q \\ I_{zz}\, r \end{bmatrix} - I_r \begin{bmatrix} p \\ q \\ r \end{bmatrix} \times \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \omega_\Gamma + \tau \right)$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} (I_{yy} - I_{zz})q\ \ r/I_{xx} \\ (I_{zz} - I_{xx})p\ \ r/I_{yy} \\ (I_{xx} - I_{yy})p\ \ q/I_{zz} \end{bmatrix} - \begin{bmatrix} q/I_{xx} \\ -p/I_{yy} \\ 0 \end{bmatrix} \omega_\Gamma + \begin{bmatrix} \tau_\phi/I_{xx} \\ \tau_\theta/I_{yy} \\ \tau_\psi/I_{zz} \end{bmatrix}$$

donde $\omega_\Gamma = \omega_1 - \omega_2 + \omega_3 - \omega_4$. Las aceleraciones angulares en el sistema de referencia fijo son entonces atraídas a partir de las del sistema de referencia del cuadricóptero con la matriz de transformación ${W_\eta}^{-1}$ y su derivada temporal.

$$\ddot{\eta} = \frac{d}{dt}\left({W_\eta}^{-1}\, v\right) = \frac{d}{dt}\left({W_\eta}^{-1}\right)v + {W_\eta}^{-1}\, \dot{v} \qquad (13)$$

$$= \begin{bmatrix} 0 & \dot{\phi}C_\phi T_\theta + \dot{\theta}S_\phi/C_\theta^2 & \dot{\phi}S_\phi C_\theta + \dot{\theta}C_\phi/C_\theta^2 \\ 0 & -\dot{\phi}S_\phi & -\dot{\phi}C_\phi \\ 0 & \dot{\phi}C_\phi/C_\theta + \dot{\phi}S_\phi T_\theta/C_\theta & \dot{\phi}S_\phi/C_\theta + \dot{\phi}C_\phi T_\theta/C_\theta \end{bmatrix} v + {W_\eta}^{-1}\, \dot{v}$$

4.2. Ecuaciones de Euler-Lagrange

La función de Lagrange $\mathcal{L}$ es la suma de las energías de traslación E_{trans} y rotación E_{rot} menos la energía potencial E_{pot}

$$\mathcal{L}(q, \dot{q}) = E_{trans} + E_{rot} - E_{pot} = (m/2)\dot{\xi}^T\dot{\xi} + (1/2)v^T Iv - mgz \qquad (14)$$

La ecuación de Euler-Lagrange[ii] con fuerzas externas y pares es

$$\begin{bmatrix} f \\ \tau \end{bmatrix} = \frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}}\right) - \frac{\partial \mathcal{L}}{\partial q} \qquad (15)$$

Los componentes lineales y angulares no dependen entre sí, por lo tanto, pueden estudiarse por separado. La fuerza lineal externa es el empuje total de los rotores. La ecuación lineal de Euler-Lagrange es

$$f = RT_B = m\ddot{\xi} + mg\begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \qquad (16)$$

que es equivalente con la ecuación (11).

[ii] P. Castillo, R. Lozano, and A. Dzul, "Stabilisation of a mini rotorcraft with four rotors," IEEE Control Systems Magazine, pp. 45–55, Dec. 2005.

La matriz Jacobiana $\mathcal{J}(\eta)$ desde υ hasta $\dot{\eta}$ es

$$\mathcal{J}(\eta) = \mathcal{J} = W_\eta^T \, I \, W_\eta$$
$$= \begin{bmatrix} I_{xx} & 0 & -I_{xx}S_\theta \\ 0 & I_{yy}C_\phi^2 + I_{zz}S_\phi^2 & (I_{yy} - I_{zz}C_\phi S_\phi C_\theta) \\ -I_{xx}S_\theta & (I_{yy} - I_{zz}C_\phi S_\phi C_\theta) & I_{xx}C_\theta^2 + I_{yy}S_\phi^2 C_\theta^2 + I_{zz}C_\phi^2 C_\theta^2 \end{bmatrix} \qquad (17)$$

Por lo tanto, la energía de rotación E_{rot} puede expresarse en el sistema de referencia como

$$E_{rot} = (1/2)\upsilon^T I \, \upsilon = (1/2)\ddot{\eta}^T \mathcal{J} \ddot{\eta} \qquad (18)$$

La fuerza angular externa es la de los pares de los rotores. Las ecuaciones angulares de Euler-Lagrange son

$$\tau = \tau_B = \mathcal{J}\,\ddot{\eta} + \frac{d}{dt}(\mathcal{J})\dot{\eta} - \frac{1}{2}\frac{\partial}{\partial \eta}(\dot{\eta}^T \mathcal{J}\dot{\eta}) = \mathcal{J}\ddot{\eta} + C(\eta, \dot{\eta})\dot{\eta} \qquad (19)$$

donde la matriz $C(\eta, \dot{\eta})$ es el término de Coriolis, que contiene los términos giroscópicos y centrípetas.

La matriz $C(\eta, \dot{\eta})$[iii] tiene la forma

$$C(\eta, \dot{\eta}) = \begin{bmatrix} C_{11} & C_{12} & C_{12} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \qquad (20)$$

donde

$$C_{11} = 0$$

$$C_{12} = (I_{yy} - I_{zz})(\dot{\theta}C_\phi S_\phi + \dot{\psi}S_\phi^2 C_\theta) + (I_{zz} - I_{yy})(\dot{\psi}C_\phi^2 C_\theta - I_{xx}\dot{\psi}C_\theta)$$

$$C_{12} = (I_{zz} - I_{yy})\dot{\psi}C_\phi S_\phi C_\theta^2$$

$$C_{21} = (I_{zz} - I_{yy})(\dot{\theta}C_\phi S_\phi + \dot{\psi}S_\phi C_\theta) + (I_{yy} - I_{zz})(\dot{\psi}C_\phi^2 C_\theta - I_{xx}\dot{\psi}C_\theta)$$

$$C_{22} = (I_{zz} - I_{yy})\dot{\phi}C_\phi S_\phi$$

$$C_{23} = -I_{xx}\dot{\psi}S_\theta C_\theta + I_{yy}\dot{\psi}S_\phi^2 S_\theta C_\theta + I_{zz}\dot{\psi}C_\phi^2 S_\theta C_\theta$$

$$C_{31} = (I_{yy} - I_{zz})\dot{\psi}C_\theta^2 S_\phi C_\phi - I_{xx}\dot{\theta}C_\theta$$

$$C_{32} = (I_{zz} - I_{yy})(\dot{\theta}C_\phi S_\phi S_\theta + \dot{\phi}S_\phi^2 C_\theta) + (I_{yy} - I_{zz})\dot{\phi}C_\phi^2 C_\theta + I_{xx}\dot{\psi}S_\theta C_\theta - I_{yy}\dot{\psi}S_\phi^2 S_\theta C_\theta - I_{zz}\dot{\psi}C_\phi^2 S_\theta C_\theta$$

$$C_{33} = (I_{yy} - I_{zzz})\dot{\phi}C_\phi S_\phi C_\theta^2 - I_{yy}\dot{\theta}S_\phi^2 C_\theta S_\theta - I_{zz}\dot{\theta}C_\phi^2 C_\theta S_\theta + I_{xx}\dot{\theta}C_\theta S_\theta$$

[iii] G. V. Raffo, M. G. Ortega, and F. R. Rubio, "An integral predictive/nonlinear H1 control structure for a quadrotor helicopter," Automatic, vol. 46, no. 1, pp. 29–39, 2010.

La ecuación (19) conduce a las ecuaciones diferenciales para las aceleraciones angulares que son equivalentes con las ecuaciones (12) y (13)

$$\ddot{\eta} = \mathcal{J}^{-1}(\tau_B - C(\eta, \dot{\eta})\dot{\eta}) \qquad (21)$$

4.3. Efectos aerodinámicos

El modelo anterior es una simplificación de interacciones dinámicas complejas. Para que el comportamiento sea más realístico, la fuerza de arrastre generada por la resistencia del aire es incluida. Esto se ideó con las Ecuaciones (10) y (15) con la matriz de coeficientes diagonal asociando las velocidades lineales con la fuerza de desaceleración del movimiento.[iv]

$$\begin{bmatrix}\ddot{x}\\ \ddot{y}\\ \ddot{z}\end{bmatrix} = -g\begin{bmatrix}0\\0\\1\end{bmatrix} + \frac{T}{m}\begin{bmatrix}C_\psi S_\theta C_\phi + S_\psi S_\phi\\ S_\psi S_\theta C_\phi - C_\psi S_\phi\\ C_\theta C_\phi\end{bmatrix} - \frac{1}{2}\begin{bmatrix}A_x & 0 & 0\\ 0 & A_y & 0\\ 0 & 0 & A_z\end{bmatrix}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\end{bmatrix}) \qquad (22)$$

donde A_x, A_y y A_z son los coeficientes de fuerza de arrastre para las velocidades en las direcciones correspondientes del sistema de referencia inicial.

[iv] H. Bouadi and M. Tadjine, "Nonlinear observer design and sliding mode control of four rotors helicopter," Proceedings of World Academy of Science, Engineering and Technology, vol. 25, pp. 225–230, 2007.

CAPÍTULO 5. SIMULACIÓN

"Saber y saberlo demostrar, es valer dos veces."
Baltasar Gracián (1601-1658)

Antes de poner en funcionamiento cualquier proyecto debemos asegurarnos de que se han planeado todos los procesos para que el diseño del sistema sea exitoso y su comportamiento no suponga peligro alguno. La simulación es una de las herramientas más eficaces a la hora de verificar el correcto funcionamiento. De modo que, muchos de los problemas y errores frecuentemente encontrados en el arranque de un nuevo sistema pueden ser evitados. Lo más importante de todo es que mejorar el entendimiento de cómo opera el sistema.

5.1. Modelo en Simulink

En este apartado de simulación utilizaremos el paquete "Quadcopter Dynamic Modeling and Simulation (Quad-Sim) v1.00[v]" siendo este un programa para Matlab. Incorpora todas las ecuaciones que definen la dinámica del cuadricóptero (modelo matemático previamente analizado en el capítulo 4) y el modelo para Simulink; un entorno de programación visual, que funciona sobre el entorno de programación Matlab.

A través de este modelo, indicando los parámetros que definen el cuadricóptero, así como las variables de entrada, seremos capaces de visualizar por pantalla la respuesta del sistema. Los resultados pueden ser simples gráficas o, gracias a su GUI (interfaz gráfica de usuario), interpretaciones en 3D de los movimientos predefinidos por el usuario.

Dicho lo anterior, procederemos a analizaremos los bloques que lo componen y explicaremos el funcionamiento de cada uno.

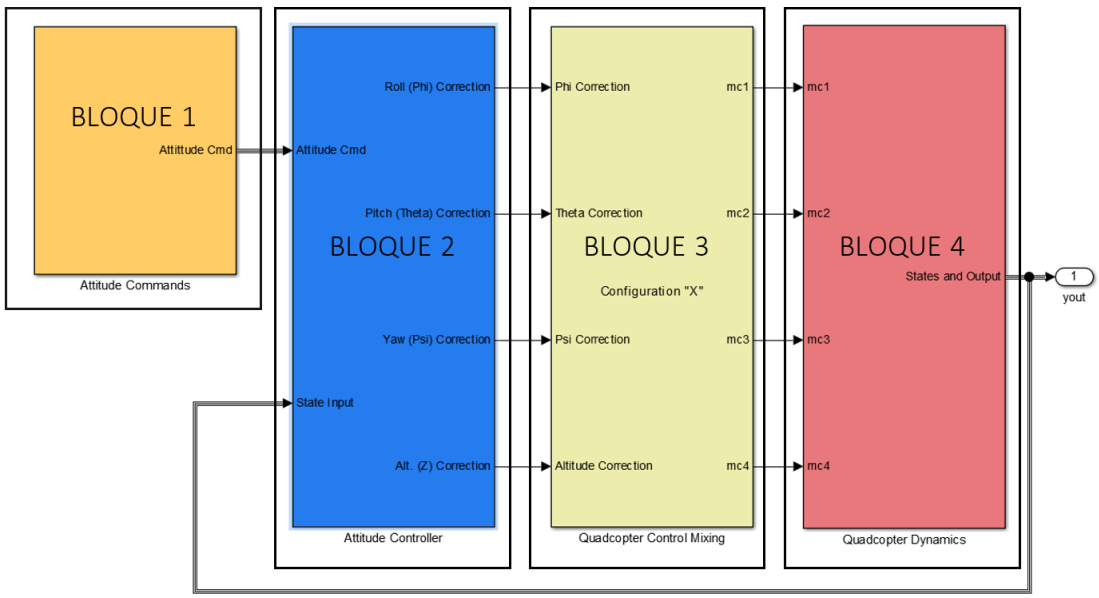


Figura 29. *Estructura del modelo en Simulink.*

[v] Quadcopter Dynamic Modeling and Simulation (Quad-Sim) v1.00 https://github.com/dch33/Quad-Sim

El **primer bloque** "*Attitude Cmd*" se encarga de dar órdenes que, el cuadricóptero debe realizar en un determinado instante; son los parámetros de entrada y equivale a órdenes de lo que podría ser un mando RC. Los cuatro tipos de comandos son altura y las tres posiciones angulares ϕ, θ, ψ sobre los ejes X,Y,Z del sistema de referencia fijo. Para este ejemplo asignaremos los valores que se muestra en la figura 30. Los ángulos están en radianes y la altura en "*feet*" (pies) siendo 1 feet equivalente a 0,3048 metros.

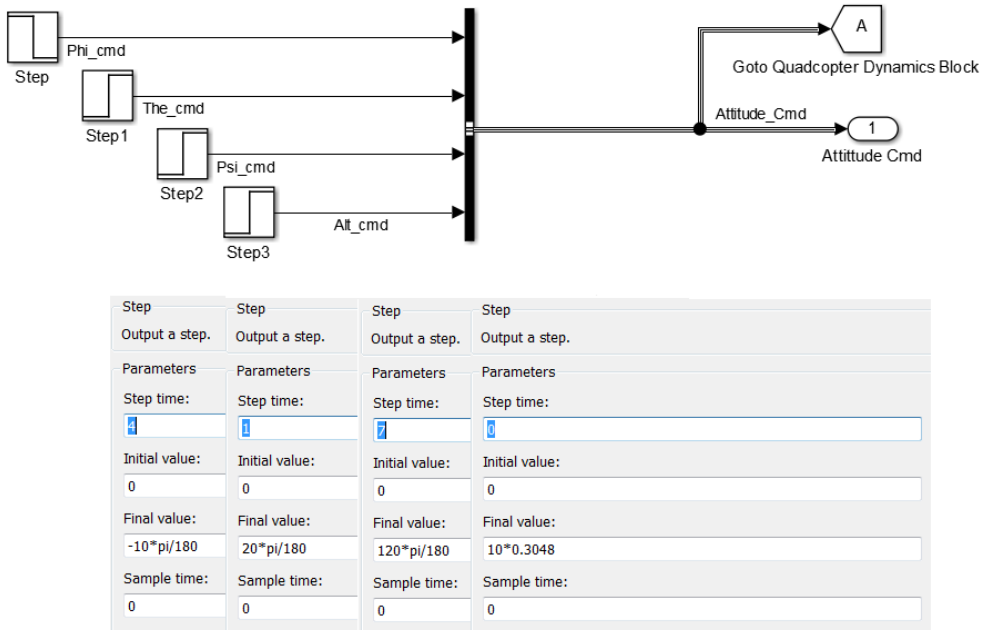


Figura 30. *Estructura interna del Bloque 1. Parámetros de entrada.*

El **segundo bloque** "*Attitude Controller*" se corresponde con la parte de control, está formado por cuatro controladores PID cada uno de los cuales se encarga de realizar las correcciones necesarias para conseguir que el sistema llegue al estado deseado. Dicho de otra forma, cada uno de los controladores calcula el error entre un valor medido y un valor deseado. Éste error será utilizado en el algoritmo de control para calcular la acción de control que consiga que el error sea nulo.

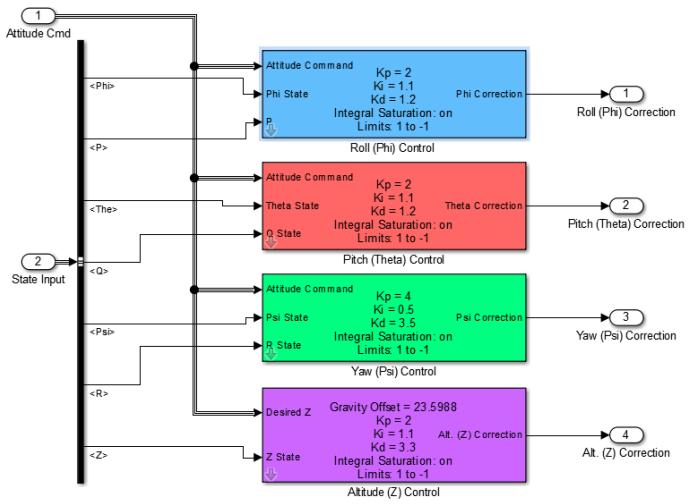


Figura 31. *Estructura interna del Bloque 2. Parámetros de ganancia para los controladores.*

La figura 31 representa la estructura interna del Bloque 2 que, como ya mencionamos antes, está formado por cuatro controladores o lo que equivale a cuatro sub-bloques. Explicaremos el funcionamiento del primer sub-bloque siendo el de los otros dos siguientes muy similar por no decir idéntico. El control de la altura lo explicaremos por separado.

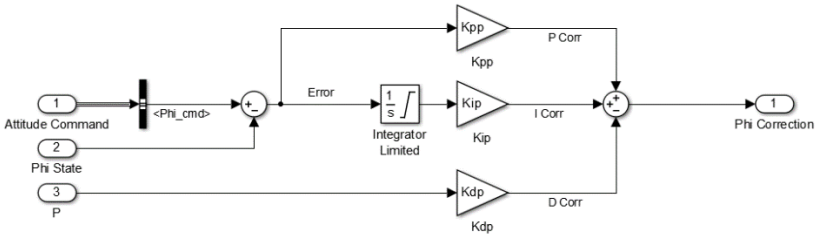


Figura 32. *Roll control. Bloque 2, estructura interna del sub-bloque 1.*

El primer sub-bloques es el controlador Phi, se corresponde con la corrección de phi o acción de control calculada en base al error entre ϕ del bloque 1 equivalente a la posición angular deseada y ϕ del bloque 4 equivalente a la respuesta del cuadricóptero para la acción de control actual (posición angular calculado en base a las ecuaciones del modelo matemático, en el modelo real este valor equivale al valor obtenido del sensor).

Este controlador es tipo PID, como se puede apreciar en la figura 32, y consiste de tres parámetros distintos: el proporcional, el integral, y el derivativo. El valor Proporcional depende del error actual. El Integral depende de los errores pasados y el Derivativo es una predicción de los errores futuros. En concreto para este PID, la corrección o acción de control para Phi es:

- Rol control:

$$PID_{\phi} = Kpp(\phi_{cmd} - \phi_{State}) + \left(Kip \times (\phi_{cmd} - \phi_{State}) + Kip \times \left(\phi_{cmd_{-1}} - \phi_{State_{-1}}\right)\right) - P \times Kdp$$

donde P es la velocidad angular en el eje X o lo que equivale a la derivada de la posición angular respecto del tiempo; Kpp, Kip y Kdp son ganancias y son parámetros diseñados por el usuario.

La acción de control para los otros dos PIDs es:

- *Pitch control*:

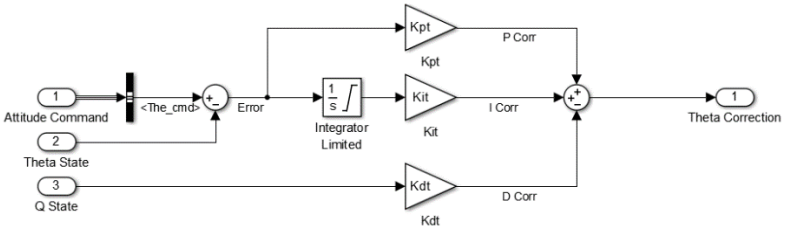


Figura 33. *Pitch control. Bloque 2, estructura interna del sub-bloque 2.*

$$PID_{\theta} = Kpt(\theta_{cmd} - \theta_{State}) + \left(Kit \times (\theta_{cmd} - \theta_{State}) + Kit \times \left(\theta_{cmd_{-1}} - \theta_{State_{-1}}\right)\right) - Q \times Kdt$$

donde Q es la velocidad angular en el eje Y; Kpt, Kit y Kdt son ganancias y son parámetros diseñados por el usuario.

- *Yaw control*:

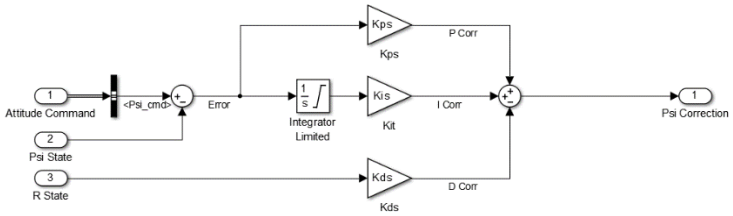


Figura 34. *Yaw control. Bloque 2, estructura interna del sub-bloque 3.*

$$PID_{\psi} = Kps(\psi_{cmd} - \psi_{State}) + \left(Kis \times (\psi_{cmd} - \psi_{State}) + Kis \times \left(\psi_{cmd_{-1}} - \psi_{State_{-1}}\right)\right) - R \times Kds$$

donde S es la velocidad angular en el eje Z; Kps, Kis y Kds son ganancias y son parámetros diseñados por el usuario.

En cuanto al control de la altura, en la simulación no se tiene en cuenta el estado de la batería, es muy importante tener en cuenta que cuando la batería reduce su capacidad, la tensión de salida disminuye, por tanto, la corriente baja y la fuerza de empuje disminuye. Sin embargo, estos factores en la simulación no nos interesan mucho y sólo los tendremos en cuenta en el sistema real que, solucionaremos por medio de software.

- *Altitude control*:

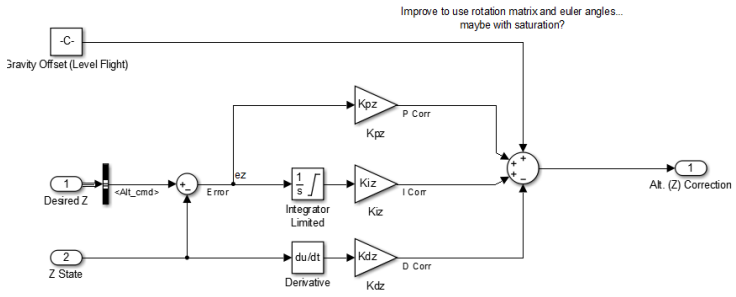


Figura 35. *Altitude control. Bloque 2, estructura interna del sub-bloque 4.*

$$PID_{Z} = Kpz(Z_{cmd} - Z_{State}) + \left(Kiz \times (Z_{cmd} - Z_{State}) + Kiz \times \left(Z_{cmd_{-1}} - Z_{State_{-1}}\right)\right) - Kds \times \left((Z_{cmd} - Z_{State}) - \left(Z_{cmd_{-1}} - Z_{State_{-1}}\right)\right) + Gravity_{offset}$$

El **tercer bloque** "*Quadcopter Control Mixing*" se encarga de aplicar la acción de control correspondiente a cada motor. Cabe destacar que existen dos tipos de configuraciones de vuelo que dependen del sistema de referencia del cuadricóptero elegido. En la figura 14 se muestran las dos configuraciones.

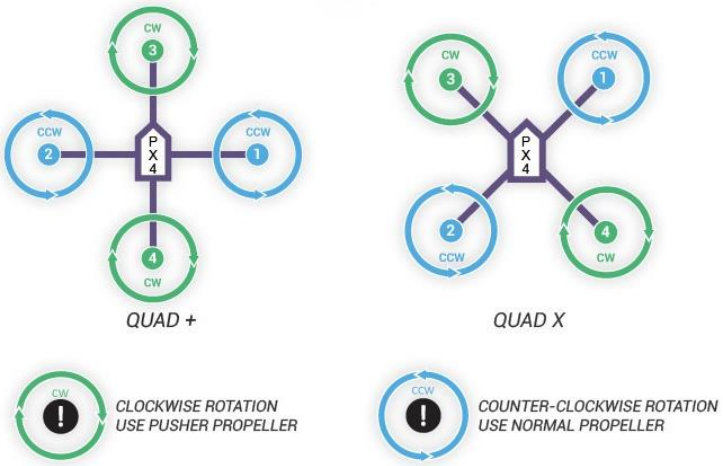


Figura 36. *Izquierda configuración +, derecha configuración X.*

Tanto en la simulación como en el proyecto real se ha utilizado la configuración tipo X. Por lo tanto, debemos escoger la configuración X, siendo esta la del sub-bloque 2 del bloque 3 y tiene la siguiente estructura.

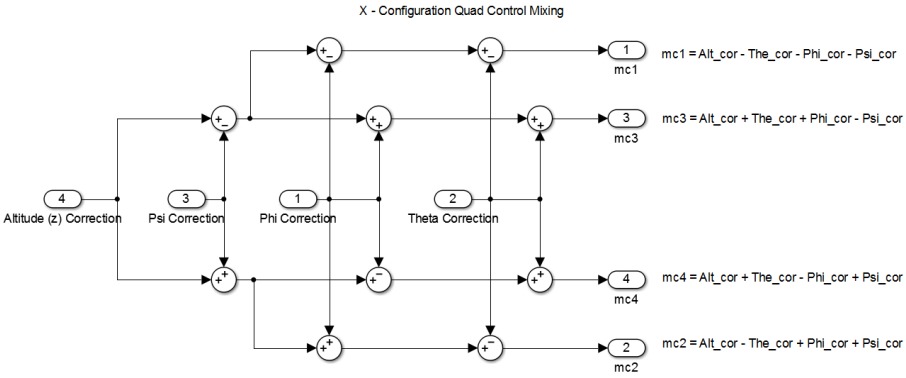


Figura 37. *Quad Control Mixing. Estructura interna Bloque 3.*

Se obtiene que el algoritmo de control o acción de control para cada uno de los rotores es:

$$M1 = PID_Z - PID_\theta - PID_\phi - PID_\psi$$

$$M2 = PID_Z + PID_\theta + PID_\phi - PID_\psi$$

$$M3 = PID_Z + PID_\theta - PID_\phi + PID_\psi$$

$$M4 = PID_Z - PID_\theta + PID_\phi + PID_\psi$$

El cuarto bloque "*Quadcopter Dynamics*" describe el comportamiento del cuadricóptero en cada instante de tiempo en relación a las causas o factores capaces de producir las alteraciones en el sistema. Estos factores se corresponden con la velocidad de los rotores que dependen de la acción de control aplicada a cada motor.

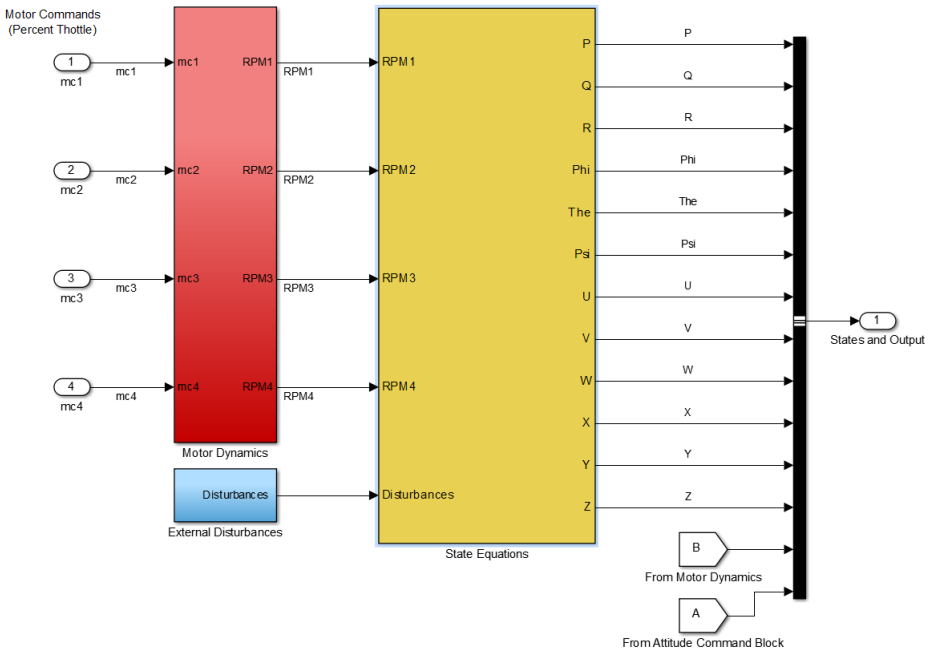


Figura 38. *Quadcopter dynamics. Estructura interna del Bloque 4.*

La acción de control o aceleración de los motores esta expresada en porcentajes, es decir está limitada a un rango de valores entre 0 y 100. Estos valores se deben transformar a revoluciones por minuto. De esta tarea se encarga el primer sub-bloque.

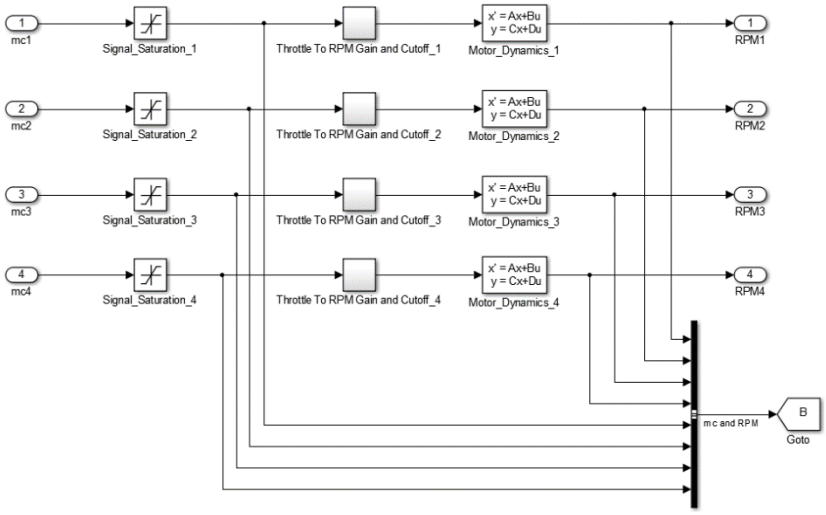


Figura 39. *Motor dynamics. Bloque 4, estructura interna del sub-bloque 1.*

Una vez obtenida la velocidad angular de cada uno de los rotores, el sub-bloque 2, haciendo uso de las ecuaciones definidas en el modelo matemático, calculará el estado del cuadricóptero. Esto incluye la posición angular, velocidad angular, posición lineal y la velocidad lineal.

5.2. Resultados

Los resultados obtenidos, para las variables de entrada y ganancias de los PIDs anteriormente señaladas (figura 30 y figura 31), se muestran a continuación.

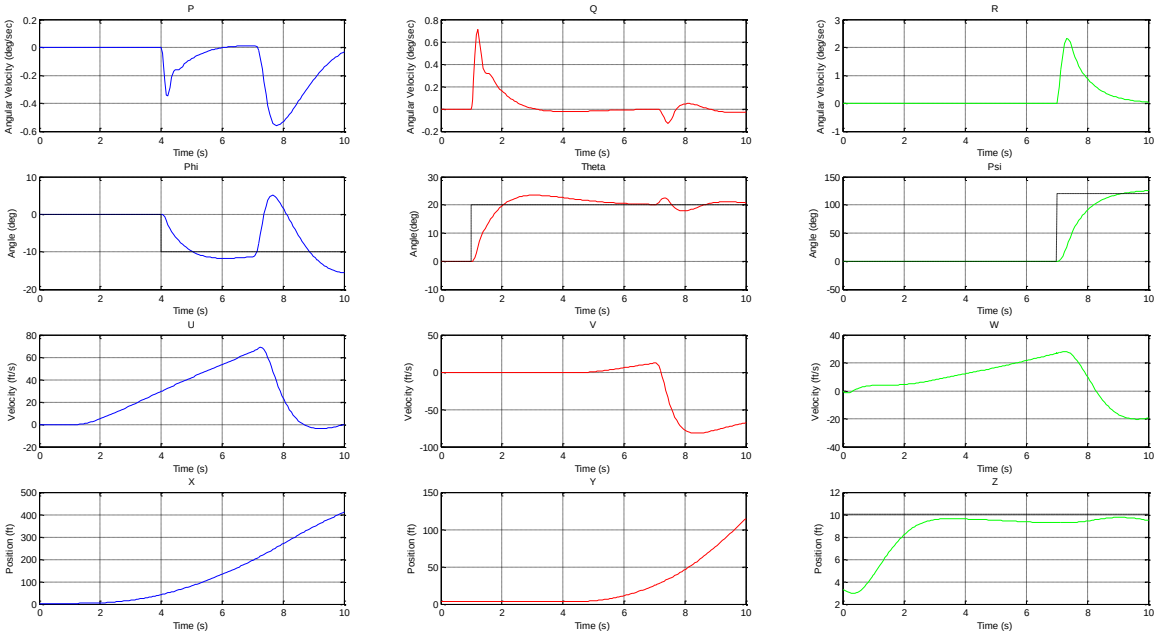


Figura 40. *Resultados de la simulación. Posición angular, velocidad angular, posición lineal y velocidad lineal del cuadricóptero.*

A partir de la figura 40, se observa que el control de posición angular para *Theta* y *Psi* es bastante bueno mientras que el control de Phi no tanto. El control de la altura consigue posicionar la aeronave en la posición deseada o de referencia. Para mejorar el control, debemos modificar las ganancias de los controladores realizando varias simulaciones hasta conseguir los valores óptimos.

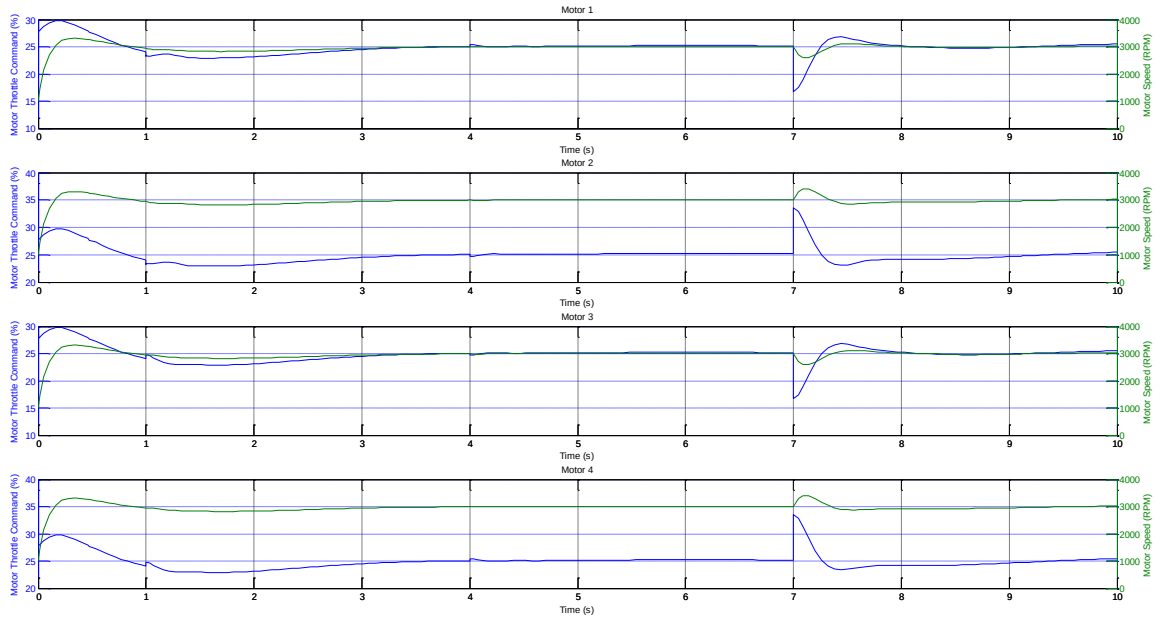


Figura 41. *Resultados de la simulación. Aceleración (%) y velocidad angular de los motores.*

A continuación, se muestran los resultados de la simulación en 3D. A la izquierda se muestra el comportamiento del cuadricóptero en cada instante y a la derecha la posición en la que está. En la figura 42 se muestra el estado del cuadricóptero en el instante 6 segundos y en la figura 43 en el instante 10 segundos.

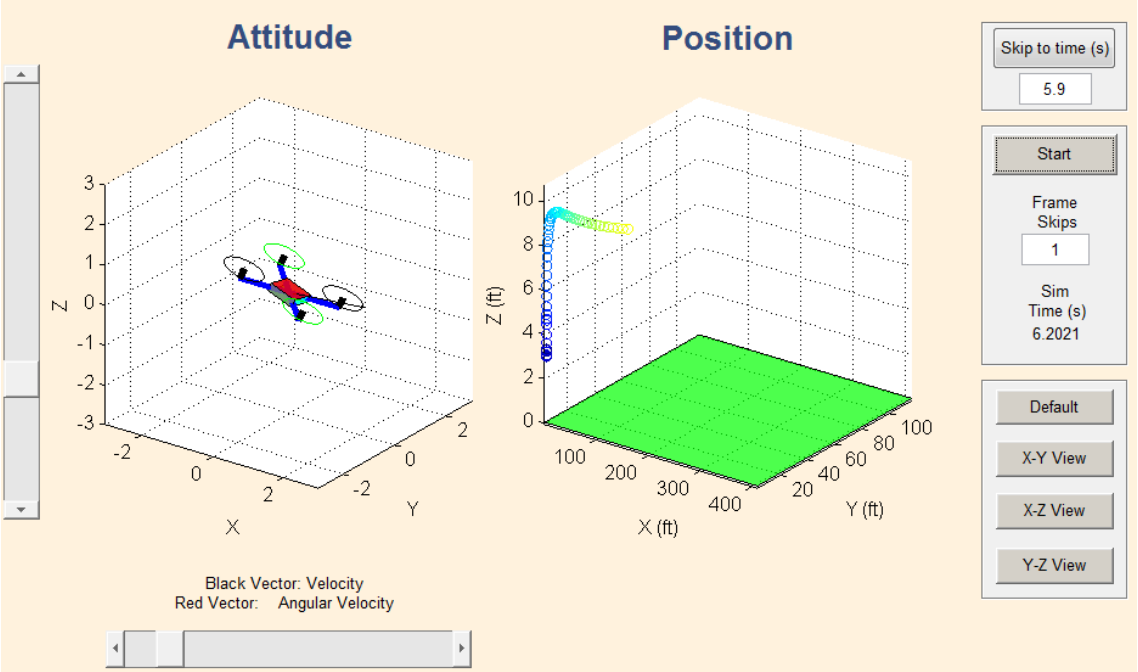


Figura 42. *Visualización 3D. Instante 6 segundos.*

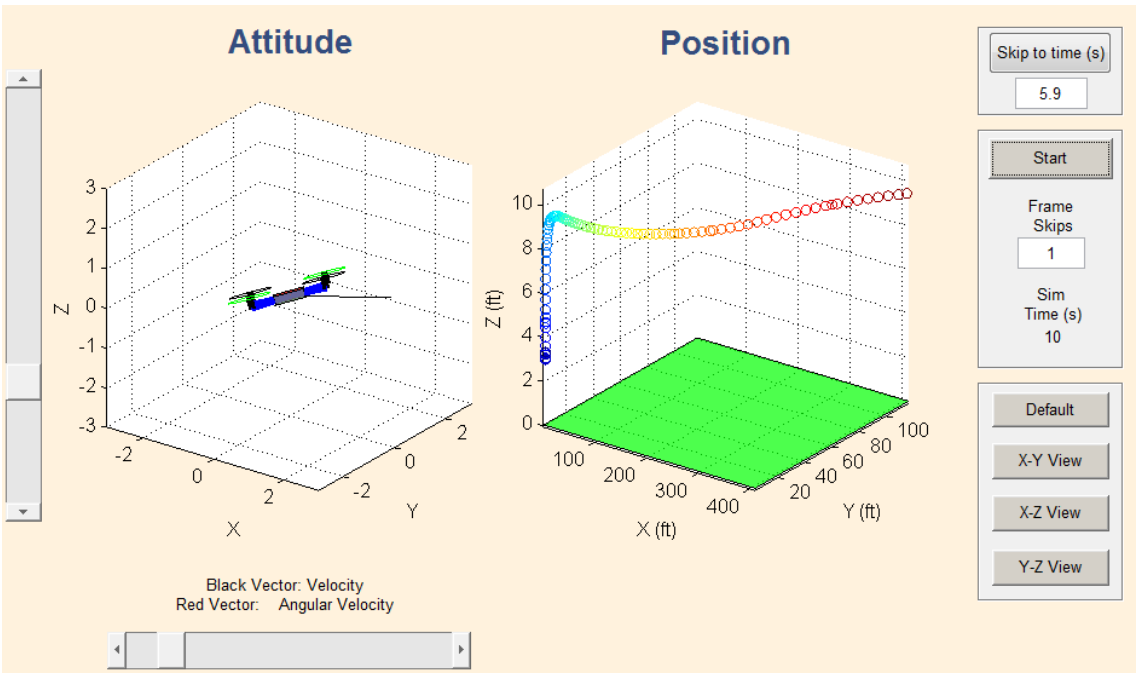


Figura 43. *Visualización 3D. Instante 10 segundos.*

CAPÍTULO 6. DISEÑO DEL SOFTWARE

"El problema de fondo es que el software contiene errores, y cuanto más software hay, más errores contiene."
Andrew S. Tanenbaum

En este apartado hablaremos de la parte de programación, para ello es importante conocer el entorno de programación y las limitaciones del microcontrolador con el cual se esté trabajando. En efecto, para este proyecto se ha utilizado el microcontrolador STM32F4 y el entorno de programación es µVision 5 de Keil.

Antes de empezar a programar, por una parte, hay que añadir la librería CMSIS (una interfaz estándar de software independiente del fabricante del micro) que permite acceder a los periféricos del mismo. Por otra parte, también es importante añadir la librería Standard Peripheral ya que, ofrece un conjunto de funciones para manejar los periféricos de la familia STM32F4 sin tener que tratar con registros.

6.1. Configuración de puertos y pines

Para que el microcontrolador pueda interactuar con el mundo externo, se dotan de puertos o pines que se conectan a otros sistemas o componentes diversos. Muchos de los pines de entrada/salida están multiplexados con funciones alternativas.

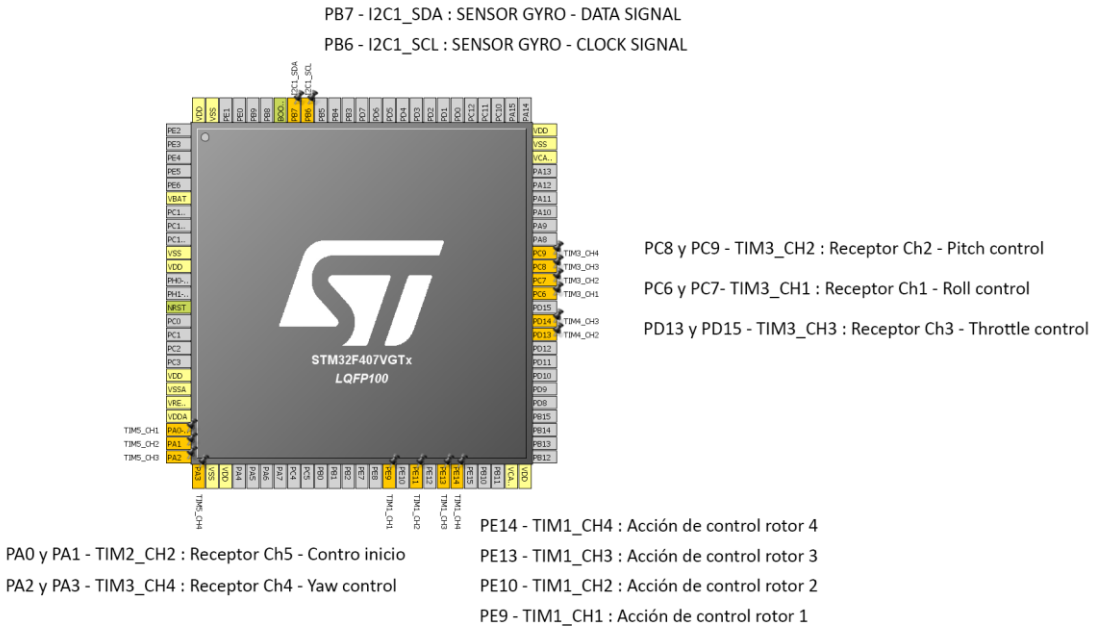


Figura 44. *Asignación de pines.*

En la figura 44 se muestran todos los pines utilizados de los cuales, 12 son de entrada y 4 son de salida. Un total de 16 pines, sin contar los pines de alimentación, cada uno de ellos configurado para cumplir con una misión. Se han utilizado cuatro *timers*, cada uno posee cuatro canales configurables para generar o captar señales PWM.

6.1.1. Configuración del receptor RC.

El primer paso sería definir las estructuras con las que vamos a trabajar. Estas estructuras están predefinidas e incorporan todos los parámetros que se pueden aplicar a la nueva estructura. Por ejemplo, para la estructura `GPIO_InitTypeDef`, los parámetros serían los pines que se desean asignar, el puerto, el modo (ya sea entrada, salida, modo alternativo), velocidad del reloj, etc.

```
/* Definir estructuras */
      GPIO_InitTypeDef GPIO_InitStructure;
      TIM_TimeBaseInitTypeDef TIM_TimeBaseInitStruct;
      TIM_ICInitTypeDef TIM_ICInitStruct;
      NVIC_InitTypeDef   NVIC_InitStructure;
```

A continuación, habilitamos el reloj del periférico APB1 y del periférico AHB1. Este paso es muy importante, equivale a darle vida al puerto con el que se esté trabajando. En este caso, el puerto GPIOC y el TIM3.

```
/* Inicializamos reloj del TIM3 */
      RCC_APB1PeriphClockCmd (RCC_APB1Periph_TIM3, ENABLE);

/* Inicializamos reloj del puerto GPIOC */
      RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOC, ENABLE);
```

En este paso configuramos la nueva estructura; los pines PC6, PC7, PC8, PC9 en modo alternativo.

```
/* Configuramos puertos y pines */
      GPIO_InitStructure.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7 |
      GPIO_Pin_8 | GPIO_Pin_9; // PC6, PC7, PC8, PC9
      GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
      GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
      GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
      GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
      GPIO_Init (GPIOC, &GPIO_InitStructure); // aplicamos la config.
```

En este paso multiplexamos los pines PC6, PC7, PC8, PC9 con el TIM3

```
/* Configuramos los pines en modo alternativo */
      GPIO_PinAFConfig (GPIOC, GPIO_PinSource6, GPIO_AF_TIM3);
      GPIO_PinAFConfig (GPIOC, GPIO_PinSource7, GPIO_AF_TIM3);
      GPIO_PinAFConfig (GPIOC, GPIO_PinSource8, GPIO_AF_TIM3);
      GPIO_PinAFConfig (GPIOC, GPIO_PinSource9, GPIO_AF_TIM3);
```

Puesto que las señales del receptor son más prioritarias que el resto de señales que llegan al micro, éstas deben realizase mediante interrupciones. Cuando se produce una interrupción básicamente la ejecución del programa principal se interrumpe para atender la interrupción que en este caso es `TIM3_IRQn`.

```
/* Configuramos las interrupciones */
      NVIC_InitStructure.NVIC_IRQChannel = TIM3_IRQn;
      NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
      NVIC_InitStructure.NVIC_IRQChannelSubPriority = 1;
      NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
      NVIC_Init(&NVIC_InitStructure); // aplicamos la configuración
```

El siguiente paso sería configurar el *timer*, se tiene que indicar el periodo que, a su vez, equivale al número de cuentas máximas antes de que se produzca la auto-recarga o inicio desde cero; el *prescaler*, para definir la velocidad a la que irán las cuentas e indicar si irán incrementando o decrementando.

```
/* Configuramos timer */
      TIM_TimeBaseInitStruct.TIM_Period = 19999; // Queremos periodo
de 20 ms = 200000 us
      TIM_TimeBaseInitStruct.TIM_Prescaler = ((SystemCoreClock /2) /
1000000) - 1; // Para ello las cuentas han de incrementar cada 1us,
la frecuencia de entrada al prescaler es 100 MHz
      TIM_TimeBaseInitStruct.TIM_ClockDivision = 1;
      TIM_TimeBaseInitStruct.TIM_CounterMode = TIM_CounterMode_Up;

      TIM_TimeBaseInit (TIM3, &TIM_TimeBaseInitStruct);
```

Para establecer la velocidad de las cuentas debemos conocer la estructura interna de los *timer*s. Por tanto, de acuerdo con la Figura 45 la velocidad de las cuentas depende tanto de la velocidad del reloj que llega como del *prescaler*.

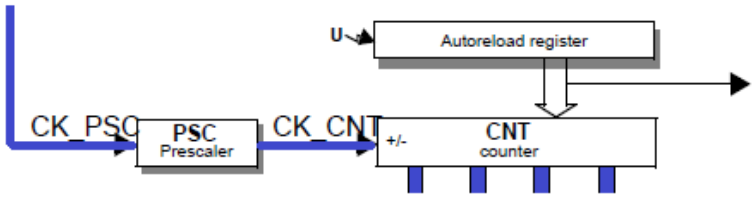


Figura 45. *Diagrama de bloques de un timer de propósito general.*

Si queremos que la velocidad de las cuentas sea de 1 micro segundo y sabiendo la velocidad del reloj que le llega por defecto es *SystemCoreClock/2* el *prescaler* se calcula de la siguiente forma:

$$PSC = \frac{SystemCoreClock/2}{1.000.000} - 1$$

Para un periodo de 20 ms, teniendo en cuenta que la precisión o las cuentas varían cada 1 micro segundo el periodo sería:

$$20000 \times 1\,\mu s = 20\,ms$$

$$Periodo = 20000 - 1 = \mathbf{19999}\ \ (\boldsymbol{cuentas\ empiezan\ en\ 0})$$

En este paso, configuramos los cuatro canales del *timer* en modo captura; las capturas detectan tanto flancos de subida como flancos de bajada. En el código a continuación se muestra la configuración para captar señales PWM solo para el canal 1 del receptor siendo el de los otros 6 muy parecido.

```
      //--------------CHANNEL1---------------------
      TIM_ICInitStruct.TIM_Channel = TIM_Channel_1;
      TIM_ICInitStruct.TIM_ICPolarity = TIM_ICPolarity_Rising;
      TIM_ICInitStruct.TIM_ICSelection = TIM_ICSelection_DirectTI;
      TIM_ICInitStruct.TIM_ICPrescaler = TIM_ICPSC_DIV1;
      TIM_ICInitStruct.TIM_ICFilter = 0x0;
      TIM_ICInit(TIM3,&TIM_ICInitStruct);
      //--------------CHANNEL2---------------------
      TIM_ICInitStruct.TIM_Channel = TIM_Channel_2;
      TIM_ICInitStruct.TIM_ICPolarity = TIM_ICPolarity_Falling;
      TIM_ICInitStruct.TIM_ICSelection = TIM_ICSelection_DirectTI;
      TIM_ICInitStruct.TIM_ICPrescaler = TIM_ICPSC_DIV1;
      TIM_ICInitStruct.TIM_ICFilter = 0x0;
      TIM_ICInit(TIM3,&TIM_ICInitStruct);
```

Como se pude observar, se han utilizado dos canales del *timer* 3 para captar una señal PWM, esto es debido al hecho de que uno detecta los flancos de subida y otro detecta los flancos de bajada de la señal en cuestión. Esto se puede conseguir con un solo canal detectando tanto flancos de subida como flancos de bajada, de hecho se hizo, pero al realizar varias pruebas, se pudo observar que a veces daba problemas y este es el motivo por el cual se ha realizado con dos canales.

Por último, habilitamos el *timer* 3 y habilitamos las interrupciones para los cuatro canales del mismo *timer*.

```
/* Habilitamos el timer 3 */
      TIM_Cmd (TIM3, ENABLE);

/* Habilitamos las interrupciones */
      TIM_ITConfig (TIM3, TIM_IT_CC1, ENABLE);
      TIM_ITConfig (TIM3, TIM_IT_CC2, ENABLE);
      TIM_ITConfig (TIM3, TIM_IT_CC3, ENABLE);
      TIM_ITConfig (TIM3, TIM_IT_CC4, ENABLE);
```

Con estos 4 canales del *timer* 3 capturaremos las señales responsables de los comandos: *pitch* y *roll*. Sin embargo, el receptor RC tiene 6 canales y el *timer* 3 solo 4, de modo que, también se han configurado los *timers* 4 y 5 para ocuparse de las señales PWM de los canales 3, 4, 5 y 6 del receptor. La configuración de esos *timers* no la explicaremos ya que es igual a la configuración del *timer* 3 que acabamos de explicar.

6.1.2. Configuración del sensor

La comunicación entre el sensor y el microcontrolador se realiza a través del bus I2C que se caracteriza por necesitar solamente dos líneas, la de datos (SDA) y la de reloj (SCL). El primer paso es, habilitar el reloj para los pines del puerto B y el reloj de la comunicación I2C1. Seguidamente configuramos los pines PB6 y PB7 en modo alternativo y serán los pines a través de los cuales se realizará la comunicación I2C.

```
// Estructuras
      GPIO_InitTypeDef GPIO_InitStruct;
      I2C_InitTypeDef I2C_InitStruct;

// Habilitamos APB1 peripheral clock para la comunicación I2C1
      RCC_APB1PeriphClockCmd (RCC_APB1Periph_I2C1, ENABLE);
// Habilitamos el reloj para los pines de SCL y SDA
      RCC_AHB1PeriphClockCmd (RCC_AHB1Periph_GPIOB, ENABLE);

      GPIO_InitStruct.GPIO_Pin = GPIO_Pin_6 | GPIO_Pin_7;
      GPIO_InitStruct.GPIO_Mode = GPIO_Mode_AF;
      GPIO_InitStruct.GPIO_Speed = GPIO_Speed_50MHz;
      GPIO_InitStruct.GPIO_OType = GPIO_OType_OD;
      GPIO_InitStruct.GPIO_PuPd = GPIO_PuPd_UP;
      GPIO_Init (GPIOB, &GPIO_InitStruct);

// Configuramos los pines en modo alternativo, con I2C1
      GPIO_PinAFConfig(GPIOB, GPIO_PinSource6, GPIO_AF_I2C1); // SCL
      GPIO_PinAFConfig(GPIOB, GPIO_PinSource7, GPIO_AF_I2C1); // SDA
```

A continuación, configuramos los parámetros de la comunicación I2C1, tales como la velocidad de comunicación, el modo, ciclo de trabajo, etc. Finalmente habilitamos la comunicación I2C1.

```
// Configure I2C1
      I2C_InitStruct.I2C_ClockSpeed = 100000;
      I2C_InitStruct.I2C_Mode = I2C_Mode_I2C;
      I2C_InitStruct.I2C_DutyCycle = I2C_DutyCycle_2;
      I2C_InitStruct.I2C_OwnAddress1 = 0x00;
      I2C_InitStruct.I2C_Ack = I2C_Ack_Disable;
      I2C_InitStruct.I2C_AcknowledgedAddress =
I2C_AcknowledgedAddress_7bit;
      I2C_Init (I2C1, &I2C_InitStruct);
// Enable I2C1
      I2C_Cmd (I2C1, ENABLE);
```

6.1.3. Configuración de los ESCs

Los ESCs necesitan una señal PWM con un pulso entre 1ms y 2ms y un periodo de 4ms (250 Hz). Estos parámetros pueden variar, sobretodo la parte del periodo que equivale al *refresh rate* o el tiempo de actualización. A partir del ancho del pulso los motores giraran más o menos revoluciones.

De forma similar a la configuración del receptor (apartado 6.1.1.) empezamos configurando los *pines* y el puerto, habilitamos el reloj para ese puerto y para el *timer* correspondiente y enlazamos los pines con el *timer*.

```
/* TIM1 clock enable */
   RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);
/* GPIOE clock enable */
   RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOE, ENABLE);
/* TIM1 channel 2 pin (PE.9) configuration */
   GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_11 |
GPIO_Pin_13 | GPIO_Pin_14;;
   GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
   GPIO_InitStructure.GPIO_Speed = GPIO_Speed_100MHz;
   GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
   GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;
   GPIO_Init(GPIOE, &GPIO_InitStructure);

/* Connect TIM pins to AF2 */
   GPIO_PinAFConfig(GPIOE, GPIO_PinSource9, GPIO_AF_TIM1);
   GPIO_PinAFConfig(GPIOE, GPIO_PinSource11, GPIO_AF_TIM1);
   GPIO_PinAFConfig(GPIOE, GPIO_PinSource13, GPIO_AF_TIM1);
   GPIO_PinAFConfig(GPIOE, GPIO_PinSource14, GPIO_AF_TIM1);
```

A continuación, configuramos el *timer* con un periodo de 4 ms y cuentas incrementando.

```
   TIM_TimeBaseStructInit (&TIM_TimeBaseStructure);
   TIM_TimeBaseStructure.TIM_Prescaler = ((SystemCoreClock) /
1000000) - 1;
   TIM_TimeBaseStructure.TIM_Period = 4000; // Periodo 4 ms
   TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
   TIM_TimeBaseInit(TIM1 , &TIM_TimeBaseStructure);
```

En este paso configuramos lo canales para que generen una señal PWM con un pulso de 1 ms que es cuando los motores están parados. Para el caso del canal 1 la configuración es:

```
// Canal 1
   TIM_OCStructInit (& TIM_OCInitStructure);
   TIM_OCInitStructure.TIM_Pulse = 1000;
   TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_PWM1;
   TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
   TIM_OC1Init(TIM1 , &TIM_OCInitStructure);
```

Por último, algunas otras configuraciones necesarias y habilitamos el *timer*.

```
   TIM_BDTRStructInit(&TIM_BDTRInitStruct);
   TIM_BDTRInitStruct.TIM_AutomaticOutput =
TIM_AutomaticOutput_Enable;
   TIM_BDTRConfig(TIM1, &TIM_BDTRInitStruct);

   TIM_ITConfig(TIM1, TIM_IT_Update, ENABLE);
   TIM_Cmd(TIM1 , ENABLE);
```

Todos los ESCs internamente incorporan un microcontrolador y es esa la razón por la cual pueden aceptar distintos pulsos con un periodo que puede variar. Para configurar los ESCs tenemos que, con los ESCs desconectados de la alimentación, aplicar la señal PWM con el pulso máximo, seguidamente conectar los ESCs a la alimentación y esperar unos segundos hasta escuchar unos pitidos de confirmación. Finalmente aplicamos el pulso mínimo y ya estarían configurados.

6.2. Obtención de la información

6.2.1. Información del receptor

La información del receptor se extrae por medio de interrupciones de modo que hay un manejador que al producirse la interrupción se encarga de ejecutar un código más prioritario. El manejador del *Timer* 3 es el TIM3_IRQHandler. A continuación, se expone un trozo de código que podríamos encontrar dentro de éste. Se corresponde con la extracción de información del canal 1 del receptor. Siendo Time 1 el pulso de la señal PWM.

```
//---------------------- ROLL RECEIVER CHANNEL 1 ---------------
// ------------------ CHANNEL 1 TIM3 ------------------------
   if(TIM_GetFlagStatus(TIM3,TIM_FLAG_CC1)==SET)
   {
      TIM_ClearFlag(TIM3,TIM_FLAG_CC1);
      counter1 = TIM_GetCapture1(TIM3);
   }

// ------------------ CHANNEL 2 TIM3 ------------------------
   if(TIM_GetFlagStatus(TIM3,TIM_FLAG_CC2)==SET)
   {
      TIM_ClearFlag(TIM3,TIM_FLAG_CC2);
      counter2 = TIM_GetCapture2(TIM3);
   }

   if(counter2>counter1){
      Time1=counter2-counter1;
   }
   else{
      Time1=20000-(counter1-counter2);
   }
```

El funcionamiento es simple, el canal 1 y 2 del *timer* 3 están configurado en modo captura, el primero detecta flancos de subida y el segundo detecta flancos de bajada. Se sabe que la señal capturada tiene un periodo de 20 ms y se desea medir el ancho del pulso t_{on} también conocido como ciclo de trabajo, que sabemos que puede variar entre 1 ms y 2 ms.

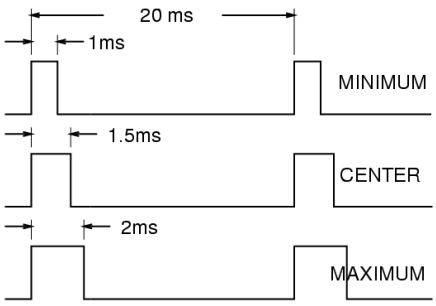


Figura 46. *Señal PWM capturada, ciclo de trabajo mínimo y máximo.*

Para calcular el ciclo de trabajo, lo lógico es hacer la diferencia entre el instante en el que se ha detectado el flanco de subida y el tiempo en el que se ha detectado el flanco de bajada; resultado en valor absoluto. El único problema que nos podríamos encontrar es cuando se produce la auto-recarga de las cuentas del *Timer* 3.

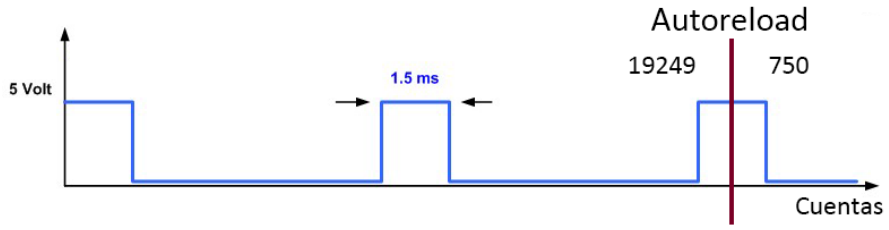


Figura 47. *Problema de la auto-recarga.*

Tal y como se muestra en la figura 47 si hacemos la diferencia entre las cuentas (750-19249) veremos que el valor no es un valor comprendido entre 1000 y 2000 (lo que viene a ser valores entre 1 y 2 ms). El código implementado incluye las condiciones necesarias para resolver este problema y tras realizar varias pruebas se ha comprobado que funciona correctamente.

Acabamos de analizar el código para extraer información del canal 1 del receptor que se corresponde con el movimiento *roll*, para el resto de canales el procedimiento es exactamente el mismos, sólo que algunos utilizan otro *timer* y por tanto otro manejador en el que introducir el código. Finalmente obtendremos cinco variables *Time1, Time2, Time3, Time4* y *Time5*. Cada uno incluye el ancho de pulso de los cinco canales del receptor que utilizaremos para implementar el control.

6.2.2. Información del sensor

Para extraer la información del sensor se han creado una serie de funciones que explicaremos a lo largo de las siguientes líneas. Las más importantes son de lectura y escritura, para entender el funcionamiento de las funciones de escritura y lectura primero debemos comprender el funcionamiento del bus I2C, siendo este el método empleado para la comunicación.

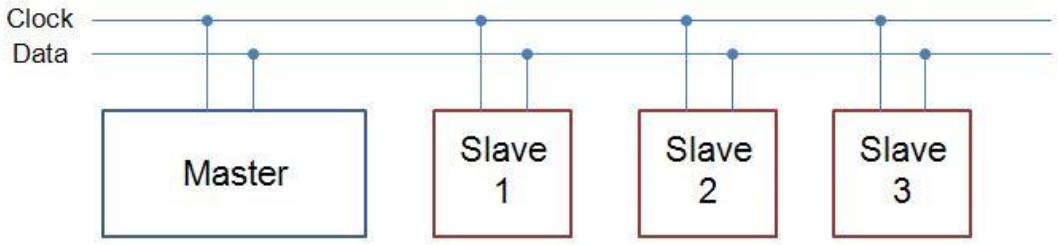


Figura 48. *Configuración del bus I2C.*

La idea general es que la comunicación se realiza entre un maestro y un esclavo, se entiende por maestro el dispositivo que inicia la transferencia en el bus y genera la señal de *Clock*; lo normal es que sea un microcontrolador. El esclavo es cualquier dispositivo electrónico que pueda ofrecer alguna información, en general son sensores. El funcionamiento se detalla en el siguiente diagrama de flujos.

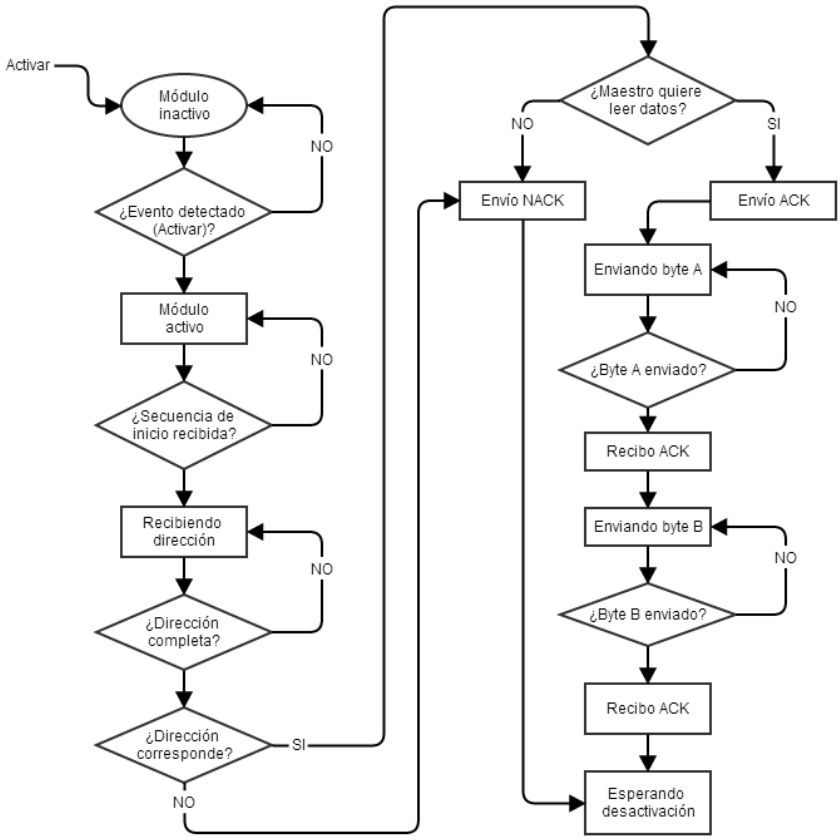


Figura 49. *Diagrama de flujos del bus I2C.*

El código correspondiente a la función de escritura, a la que se le pasa dirección del esclavo, la dirección del registro que se quiere modificar y el valor en hexadecimal que se quiere sobrescribir, es el siguiente:

```
void MPU5060_I2C_ByteWriteC (I2C_TypeDef* I2Cx, uint8_t slaveAddr,
uint8_t* pBuffer, uint8_t WriteAddr)
{
   // Comprueba si el bus está ocupado
   while (I2C_GetFlagStatus (I2Cx, I2C_FLAG_BUSY));
   // Habilita el bus
   I2C_GenerateSTART (I2Cx, ENABLE); //
   // Comprueba si somos el master
   while (!I2C_CheckEvent (I2Cx, I2C_EVENT_MASTER_MODE_SELECT));
   // Indicamos la dirección del esclavo en la que queremos escribir
  I2C_Send7bitAddress (I2Cx, slaveAddr, I2C_Direction_Transmitter);
   // comprueba que es el master quien va transmitir
   while (!I2C_CheckEvent (I2Cx,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));
   // Indicamos el registro que queremos modificar
   I2C_SendData (I2Cx, WriteAddr);
   // comprueba si se está transmitiendo el dato
   while(!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTING));
   //Modificamos el registro
   I2C_SendData (I2Cx, *pBuffer);
   // comprueba que se ha transmitido correctamente el dato
   while (!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
   // Liberamos el bus
   I2C_GenerateSTOP (I2Cx, ENABLE);
}
```

En cuanto a la función de lectura, le pasamos la dirección del esclavo, el número de bytes que queremos leer, la dirección del parámetro que queremos leer y almacenamos ese valor en una variable.

```
void MPU6050_I2C_BufferRead (I2C_TypeDef* I2Cx, uint8_t slaveAddr,
uint8_t* pBuffer, uint8_t ReadAddr, uint16_t NumByteToRead)
{
   // comprueba si el bus está ocupado
   while (I2C_GetFlagStatus (I2Cx, I2C_FLAG_BUSY));
   // Habilita el bus
   I2C_GenerateSTART (I2Cx, ENABLE);
   // Comprueba si somos el master
   while (!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_MODE_SELECT));
   // Indicamos la dirección del esclavo en la que queremos escribir
   I2C_Send7bitAddress (I2Cx, slaveAddr, I2C_Direction_Transmitter);

   // Indicamos la dirección del esclavo en la que queremos escribir
   while(!I2C_CheckEvent(I2Cx,
I2C_EVENT_MASTER_TRANSMITTER_MODE_SELECTED));

   I2C_Cmd(I2Cx, ENABLE);
   // indica el registro que queremos leer, giroscopio en X, Y, Z
cada uno cuenta con 2 bits = X_H, X_L, Y_H, Y_L, Z_H, Z_L
   I2C_SendData (I2Cx, ReadAddr);

   while (!I2C_CheckEvent(I2Cx, I2C_EVENT_MASTER_BYTE_TRANSMITTED));
   I2C_GenerateSTART (I2Cx, ENABLE);

   while (!I2C_CheckEvent (I2Cx, I2C_EVENT_MASTER_MODE_SELECT));
   I2C_Send7bitAddress (I2Cx, slaveAddr, I2C_Direction_Receiver);

   While (!I2C_CheckEvent (I2Cx,
I2C_EVENT_MASTER_RECEIVER_MODE_SELECTED));

   while (NumByteToRead)
   {
      if (NumByteToRead == 1)
      {
         I2C_AcknowledgeConfig (I2Cx, DISABLE);
         I2C_GenerateSTOP (I2Cx, ENABLE);
      }
      if (I2C_CheckEvent (I2Cx, I2C_EVENT_MASTER_BYTE_RECEIVED))
      {
         *pBuffer = I2C_ReceiveData(I2Cx);
         pBuffer++;
         NumByteToRead--;
      }
    }
    I2C_AcknowledgeConfig (I2Cx, ENABLE);
}
```

La función `void MPU6050_Init(void);` sirve para inicializar el módulo gy-86, que mediante la llamada de la función `MPU6050_I2C_Init();` configuran los puertos y pines (tal y como se muestra en el apartado 6.1.2). También se realizan una serie de configuraciones como activar el giroscopio o cambiar la escala de trabajo a través de la función `MPU5060_I2C_ByteWriteC` que acabamos de analizar.

```
void MPU6050_Init(void)
{
 MPU6050_I2C_Init (); // Habilita la configuración PINES, PUERTOS,
I2C1...

// Modificar el registro MPU_PWR_MGMT_1 a 00000000 para activar el
gyroscopio
 MPU5060_I2C_ByteWriteC (I2C1, MPU6050_ADDRESS, &buf1,
MPU_PWR_MGMT_1);
// Modificar el registro MPU6050_GYRO_CONFIG a 00001111 para que
trabaje a 500 g/s, escala máxima
 MPU5060_I2C_ByteWriteC(I2C1, MPU6050_ADDRESS, &buf2,
MPU6050_GYRO_CONFIG);
}
```

Es importante mencionar que para poder comunicarse con un sensor en concreto debemos conocer su dirección. La dirección es única para cada tipo de sensor, para el sensor MPU6050 esa dirección es `0x68` expresada en hexadecimal. También debemos conocer los registros o direcciones donde se almacena la información o respuestas del giroscopio, el registro de configuración de escala, etc.

```
#define MPU_PWR_MGMT_1              0x6B
#define MPU6050_CONFIG              0x1A
#define MPU6050_GYRO_CONFIG         0x1B

#define MPU_GYRO_XOUT_H             0x43
#define MPU_GYRO_XOUT_L             0x44
#define MPU_GYRO_YOUT_H             0x45
#define MPU_GYRO_YOUT_L             0x46
#define MPU_GYRO_ZOUT_H             0x47
#define MPU_GYRO_ZOUT_L             0x48
```

Finalmente se han implementado tres funciones para extraer la información del giroscopio para cada uno de los ejes. Para el caso de la información de la velocidad angular del eje X el código se muestra a continuación. Se leen 2 registros `MPU_GYRO_XOUT_H` y `MPU_GYRO_XOUT_L`, siendo el primero con los bits más significativos y el segundo con los menos significativos, el resultado de ambas se consigue mediante un desplazamiento de bits del primero y una suma lógica OR se consigue el valor deseado.

```
int16_t MP6050_ReadX(void)
{
   uint8_t msb, lsb;
   MPU6050_I2C_BufferRead (I2C1, MPU6050_ADDRESS, &msb,
MPU_GYRO_XOUT_H, 1);
   MPU6050_I2C_BufferRead (I2C1, MPU6050_ADDRESS, &lsb,
MPU_GYRO_XOUT_L, 1);
   return (msb << 8) | lsb;
}
```

6.3. Diseño del controlador

Para el control del cuadricóptero se ha empleado un controlador PID independiente para cada eje: *pitch*, *roll* y *yaw*. Se ha utilizado un controlador PID debido a la sencillez de su algoritmo y debido al hecho de que es posible aplicarlo de forma general a la mayoría de sistemas de control. Si el control se realiza correctamente, éste puede aceptar sin ningún problema, falta de exactitud y cambios bruscos en los parámetros del modelo.

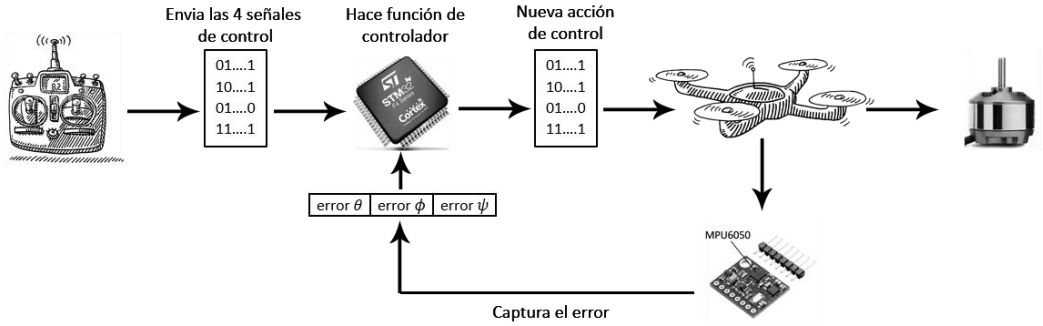


Figura 50. *Esquema del diseño del controlador.*

En la figura 50 se muestra el funcionamiento del controlador. A través del mando RC enviamos 4 señales de control. Tres de ellas indican la cantidad de grados por segundo que el cuadricóptero debe girar (velocidad angular de viraje, ver figura 51), causando los movimientos *pitch*, *roll* y *yaw*. La otra se corresponde con la aceleración (*throttle*) de los motores con la que variaremos la altura. Éstas señales son recibida por el receptor RC y transmitidas al controlador que es encargará de interpretarlas. Al mismo tiempo, el sensor envía al controlador información de los grados por segundo que detecta que el cuadricóptero está girando sobre cada uno de los ejes.

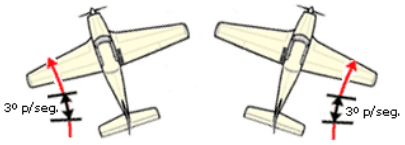


Figura 51. *Velocidad angular o ratio de viraje*

La diferencia entre lo que mide el sensor y lo que ordenamos con el mando es el error. El controlador se encargará de calcular la señal de control necesaria para que el error sea nulo y el cuadricóptero tenga la velocidad angular deseada. Para ello, se ha creado una función que calcula la acción de cada uno de los PIDs. Para el caso del control sobre el eje X, el código es el que vemos a continuación, para los otros dos PIDs es muy similar.

```
void PID_action(void){
      /* ##### Cálculo PID eje X - ROLL   #######*/
      /* Calculamos el error */
      roll_error = roll_input - rc_ch1_roll;
      /* Calculamos la acción del proporcional */
      roll_P = roll_error * KP_roll;
      /* Calculamos la acción del integral */
      roll_I = roll_I + roll_error * KI_roll;
      /* La acumulación de la integral incrementa de manera muy
rápida, nos debemos asegurar de que no sobrepase un límite. */
      if(roll_I>PID_max_roll)roll_I = PID_max_roll;
      else if(roll_I<(PID_max_roll*(-1)))roll_I = PID_max_roll*(-1);
      /* Calculamos la acción del derivatiov */
      roll_D = (roll_error - roll_error_last) * KD_roll;
      /* Obtenemos la acción del PID_roll */
      PID_roll = roll_P + roll_I + roll_D;
      /* Nos aseguramos de que no sobrepase un límite */
      if(PID_roll>PID_max_roll)PID_roll = PID_max_roll;
      else if(PID_roll<(PID_max_roll*(-1)))PID_roll =
PID_max_roll*(-1);
      /* Almacenamos el error actual, que utilizaremos en el
siguiente cálculo */
      roll_error_last = roll_error;

}
```

6.4. Medidas de seguridad

Las medidas de seguridad son el conjunto de instrucciones que se ocupan de evitar un mal funcionamiento y prevenir un estado no deseado. En primer lugar, es importante calibrar el sensor. Es decir, tomar varias muestras haciendo un sumatorio. El valor final se divide entre el número de muestras obteniendo así, el valor medio.

```
/* Empieza la calibración */
   for (cal_int = 0; cal_int < 2000; cal_int ++){
      //Sumatorio de muestras para cada uno de los ejes.
      gyro_axis_cal[1] = gyro_axis_cal[1] + MP6050_ReadX ();
      gyro_axis_cal[2] = gyro_axis_cal[2] + MP6050_ReadY ();
      gyro_axis_cal[3] = gyro_axis_cal[3] + MP6050_ReadZ ();
      Delayms (3); //Esperamos 3ms por cada muestra.
   }
//Una vez obtenidas las 2000 muestras dividimos por 2000 para
obtener el valor medio.
  gyro_axis_cal[1] = gyro_axis_cal[1]/2000;
  gyro_axis_cal[2] = gyro_axis_cal[2]/2000;
  gyro_axis_cal[3] = gyro_axis_cal[3]/2000;
```

En segundo lugar, una serie de comprobaciones antes de inicializar el vuelo. Se comprueba que la posición del "*stick*" de aceleración, está en posición de estado de reposo y además el "*switch D*" del mando debe estar desactivado (posición hacia arriba).

```
if(Time5<=1500 || Time5>5000 || Time3<1050){
   // Motores parados
   TIM_SetCompare3(TIM1, 1000);
   TIM_SetCompare1(TIM1, 1000);
   TIM_SetCompare2(TIM1, 1000);
   TIM_SetCompare4(TIM1, 1000);
   // Condiciones iniciales
   PID_roll = 0;
   roll_error_last = 0;
   PID_pitch = 0;
   pitch_error_last = 0;
   PID_yaw = 0;
   yaw_error_last = 0;

   led_red_on();
   led_blue_off();

   stado=0;
}
```

De modo que, cuando está en posición de reposo, paramos los motores, reestablecemos el error anterior a cero, así como la acción de control de cada PID y desactivamos el bucle de control. En cambio, si el "*switch D*" está activado (posición hacia abajo) y el "*stick*" de aceleración es superior a 1050 activamos el bucle de control. El bucle de control está activo siempre y cuando "estado" sea igual a uno.

```
if(Time3>1050 && Time3<2005 && Time5>1500){
   estado = 1;

   led_blue_on();
   led_red_off();
}
```

Por otra parte, se han limitado algunos parámetros, por ejemplo, en los tres controladores PID si el error es muy grande, la parte integral crece de manera muy rápida y esto podría ocasionar la desestabilización del sistema. Se ha implementado una condición en la que si la acción de la parte Integral es superior a un valor predefinido que se limite a ese valor. Del mismo modo se ha limitado en general la acción de las tres componentes del PID.

```
if(roll_I>PID_max_roll)roll_I = PID_max_roll;
else if(roll_I<(PID_max_roll*(-1)))roll_I = PID_max_roll*(-1);

if(PID_roll>PID_max_roll)PID_roll = PID_max_roll;
else if(PID_roll<(PID_max_roll*(-1)))PID_roll =
```

Algo similar se realiza para los parámetros extraídos del receptor RC, se deja una zona muerta para obtener unos resultados mejores ya que los datos obtenidos en estado de reposo a veces no son muy estables y presentan pequeñas variaciones cuando se supone que no deben variar. De modo que, hasta que no varíen por encima o por debajo de valor especificado, permanecerán con valor de reposo. En este mismo trozo de código, además, convertimos las unidades o el pulso de la señal PWM a grados por segundo. De modo que, si dividimos entre 3.075, podríamos hacer girar al cuadricóptero con una velocidad angular máxima de 160 grados por segundo

```
if(Time2 > 1508 && Time2 < 2000){
   // (500 - 8) / 3.075 = 160 grados/segundo MAXIMO!
   rc_ch1_roll = (Time2 - 1508)/3.075;
}
else if(Time2 > 1000 &&Time2 < 1492){
   rc_ch1_roll = (Time2 - 1492)/3.075;
}
else if(Time2 >= 1492 && Time2 <= 1508){
   rc_ch1_roll = 0;
}
```

Finalmente se ha implementado un filtro complementario para suavizar la respuesta del sensor dando un 70 % de importancia al valor anterior y un 30 al valor actual. Dividimos entre 65.5 (ver datasheet) para transformar la información del sensor a grados por segundo.

```
pitch_input = (pitch_input * 0.70) + ((dataY/65.5) * 0.30);
roll_input  = (roll_input * 0.70)  + ((dataX/65.5) * 0.30);
yaw_input   = (yaw_input * 0.70)   + ((dataZ/65.5) * 0.30);
```

6.5. Bucle principal

El bucle principal es la parte de código que se ejecuta repetidamente siempre que el controlador permanezca encendido. Esta parte incluye todas las medidas de seguridad analizadas en el apartado 6.4 y además contiene el bucle de control.

6.5.1. Bucle de control

El bucle de control es la parte de código que se ejecuta repetidamente hasta que la condición asignada a dicho bucle de control deje de cumplirse. En este caso siempre que "estado" sea igual a uno. En el bucle de control se calcula la acción de control de cada uno de los PIDs.

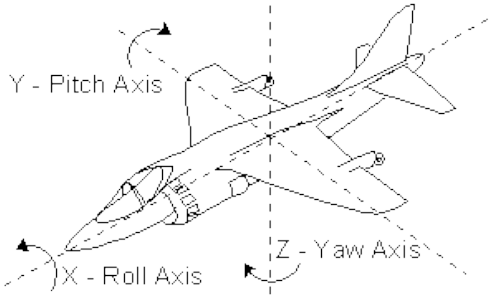


Figura 52. *Eje de referencia*

El eje de referencia es el que aparece en la figura 52 y la posición de los motores se muestra en la figura 53. En base a estos dos criterios se calcula la acción de control que se debe aplicar a cada uno de los cuatro rotores.

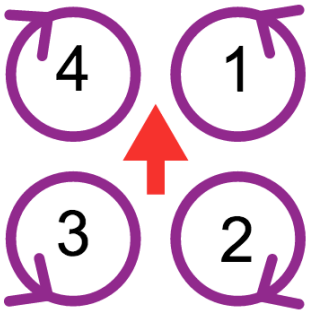


Figura 53. *Posición de los motores*

Si los motores no giran en ese sentido se debe intercambiar uno de los cables del motor con el ESC. También hay que prestar atención a las hélices, ya que hay una que es para giro horario y otra para anti-horario, de acuerdo con el giro del rotor se debe colocar una u otra para conseguir que la fuerza de empuje vertical sea hacia arriba. A continuación, se muestra el código del bucle de control.

```
if(estado==1){
   /* Se reliza el calculo de la acción de control */
   PID_action();

   if (throttle>1800)throttle=1800; // dejamos un rango de 200
   para la acción de control sobre los motores

   /* Acción de control */
   ESC1 = throttle - PID_pitch + PID_roll - PID_yaw;
   ESC2 = throttle + PID_pitch + PID_roll + PID_yaw;
   ESC3 = throttle + PID_pitch - PID_roll - PID_yaw;
   ESC4 = throttle - PID_pitch - PID_roll + PID_yaw;

   if(ESC1 > 2000)ESC1=2000;
   else if (ESC1 < 1000)ESC1=1000;
   if(ESC2 > 2000)ESC2=2000;
   else if (ESC2 < 1000)ESC2=1000;
   if(ESC3 > 2000)ESC3=2000;
   else if (ESC3 < 1000)ESC3=1000;
   if(ESC4 > 2000)ESC4=2000;
   else if (ESC4 < 1000)ESC4=1000;

   TIM_SetCompare4(TIM1, ESC1);
   TIM_SetCompare1(TIM1, ESC2);
   TIM_SetCompare2(TIM1, ESC3);
   TIM_SetCompare3(TIM1, ESC4);
}
```

Es muy importante conocer el periodo en el que se realiza el bucle de control, conocido como el *refresh rate*. En nuestro caso se ha establecido a 400 Hz.

```
//TM_Time tiempo actual y time tiempo al inicializar el bucle prin.
while((TM_Time - time) < 2.5);
```

CAPÍTULO 7. PRUEBAS Y CALIBRACIONES

"El error es el precio que pagamos por el progreso."
Alfred North Whitehead (1861 –1947)

Las primeras pruebas no fueron muy buenas, el cuadricóptero no permanecía estable y se notaban sobre-oscilaciones, signos de que los valores de las ganancias de los PIDs no eran para nada los más adecuados. Tras realizar varias pruebas modificando los parámetros la cosa fue mejorando, aunque por alguna razón el cuadricóptero en estado de vuelo tiende a desplazarse hacia delante. El motivo más probable es debido a que la parte delantera pesase un poquito más que la trasera. Pero se supone que el regulador debe compensar este peso haciendo girar los motores delanteros un poquito más que los traseros, de modo que se ha aumentado la velocidad de los motores delanteros un 5% para analizar los resultados y ver si seguía esa tendencia de ir hacia delante. Parece que la cosa mejoró un poco y es entonces cuando se hizo evidente que las ganancias del controlador pitch deben ser mayores y no iguales a las de roll como se hizo en la gran mayoría de las pruebas. Esto es debido a la propia estructura, hay una mayor distribución de peso sobre al eje x en comparación con el eje y. Finalmente se consiguió el comportamiento deseado, pero aun así, si el cuadricóptero no se coloca en posición totalmente horizontal, sino que, con una inclinación al iniciar el vuelo, el cuadricóptero vuela en dirección perpendicular al plano de esa inclinación y para estabilizar a una posición fija habría que corregir ese ángulo manualmente con el mando RC. En el apartado mejora (7.2) veremos cómo solucionar este problema.

7.1. PID tuning

En este apartado explicaremos el procedimiento seguido para conseguir los valores de ganancias más óptimos o que mejor se adaptan a nuestro sistema. El método es totalmente experimental así que se basa en, realizar una sucesión de pruebas hasta conseguir el comportamiento deseado. En primer lugar, partiendo de que todas las ganancias son nulas, fijamos las ganancias del controlador *yaw*, de las cuales*, la* proporcional a 4 y la derivativa a 0.2. Mantendremos esta configuración a lo largo de todo el procedimiento.

En segundo lugar, procedemos a configurar los reguladores que controlan el sistema en el plano horizontal, es decir, el controlador *pitch* y *roll*. Para ello, primero buscaremos el valor más óptimo de la ganancia derivativa, para ambos controladores inicialmente los parámetros serán iguales, de modo que, inicialmente fijaremos la parte derivativa para ambos a un valor de 3. Con este valor comprobaremos, girando con la mano el cuadricóptero, si los motores tratan de compensar el movimiento causado. Si los motores lo compensan, quiere decir que el controlador funciona correctamente. Sabiendo que funciona correctamente, incrementaremos la ganancia derivativa en escalones de 3, de modo que llegará un momento en el que la resistencia de los motores hace que vibren. Esto quiere decir que el sistema está sobre-oscilando. En nuestro caso, esto ocurre cuando la ganancia derivativa es igual a 27, para determinar el valor final, decrementos ese valor un 30% obteniendo que la ganancia derivativa está entre 18 y 19.

En tercer lugar, procederemos a estimar el valor de la ganancia proporcional, con un valor adecuado sería suficiente para mantener un vuelo más o menos estable. El procedimiento es similar al que hemos empleado para estimar el valor de la parte derivativa. Se basa en buscar el valor que hace que el sistema comience a sobre-oscilar. Partiremos con un valor de ganancia de 0,2 e iremos incrementando en 0,2. Finalmente se obtiene que el valor de ganancia, que hace que el sistema sobre-oscile es 2. De ahí que, el valor final sería reducir éste un 50% obteniendo que la ganancia proporciona es 1.

En cuarto lugar, estimaremos el valor de la ganancia integrar, el procedimiento es el mismo que en los dos anteriores. Debemos tener mucho cuidado con la parte integral ya que, al ejercer una acción en base al sumatorio del error a lo largo del tiempo, si la ganancia es alta, ese error puede crecer muy rápidamente y, por tanto, en general el valor de la ganancia debería ser pequeño o el sistema se volverá inestable.

Por último, hay que realizar un par de ajustes más, como modificar los parámetros del controlador yaw en base a cómo de bruscos se desean los giros sobre el eje Z y modificar los parámetros del pitch aumentando las ganancias con respecto a las del controlador roll debido a la distribución asimétrica del peso como ya mencionamos antes.

7.2 Mejoras

Tras una serie de vuelos desfavorables, se ha decidido mejorar el control, para ello se han incorporado otros sensores con el único objetivo de mejorar aún más el comportamiento del cuadricóptero.

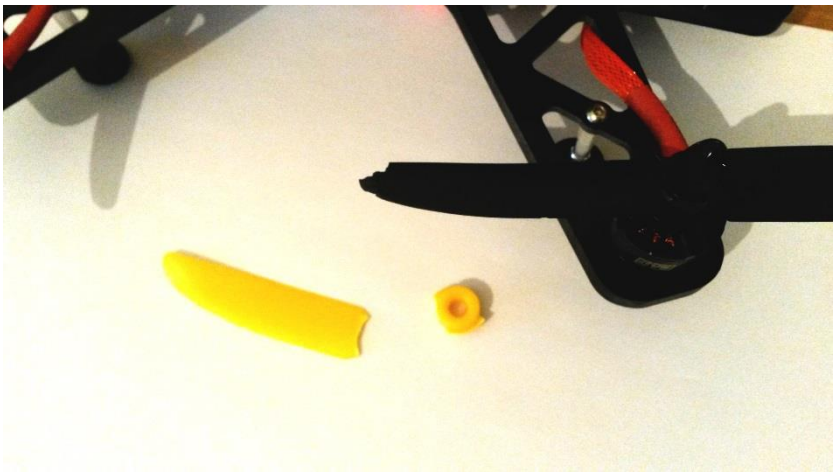


Figura 54. *Hélice rota, tras realizar una de las prueba de estabilidad.*

Con el control previamente establecido se ha conseguido mantener el vuelo estable en cuanto a orientación. Sin embargo, no disponemos de control de altura de modo que, es muy difícil y aún más para novatos, mantener el cuadricóptero a la altura deseada, puesto que debemos moderar la aceleración de los rotores manualmente. Además, si el cuadricóptero no se coloca totalmente horizontal no puede permanecer en la misma posición, esto es debido al hecho de que el control es en base a velocidades angulares (variaciones de ángulo) y no posición angular. El ángulo que tenga al inicializar el vuelo será su ángulo de referencia durante todo el vuelo.

7.2.1 Control por posición angular

La primera mejora consiste en diseñar otros dos controladores pero que estos controlen la posición angular. El objetivo es que inicialmente al despegar se utilice este tipo de control y al alcanzar una altura cambiar al control por velocidades angulares. Con esto conseguiremos solucionar el problema de mantener el cuadricóptero totalmente horizontal, aunque al despegar no esté posicionado totalmente horizontal por irregularidades del suelo.

Para diseñar el controlador necesitaremos obtener los ángulos, que digamos, es la parte difícil. Para ello disponemos de dos sensores el giroscopio y el acelerómetro, el primero mide velocidades angulares y el segundo aceleraciones lineales. Con el acelerómetro podemos obtener fácilmente los ángulos, pero primero debemos conocer su funcionamiento. Debido a la interacción gravitatoria existe una fuerza que tira hacia el centro del planeta, y por consiguiente suponiendo que el sensor está totalmente horizontal y en equilibrio el sensor detectará en el eje Z esa fuerza. Para un mejor entendimiento supongamos que el acelerómetro es una caja y dentro hay una bolita, cuando la bolita toca uno de los lados es detectado por el sensor.

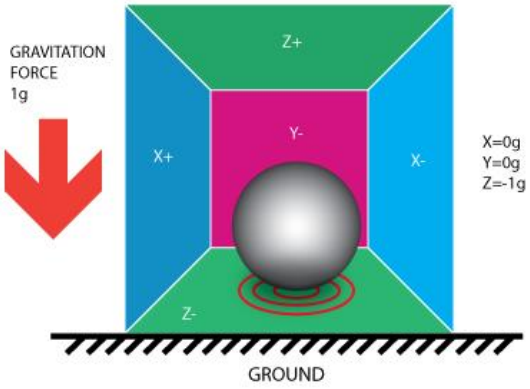


Figura 55. *Lectura del acelerómetro, en estado de equilibrio y totalmente horizontal.*

Ahora bien, si giramos el sensor 45 grados sobre el eje Y, el sensor detectará que sobre el eje X también existe una fuerza.

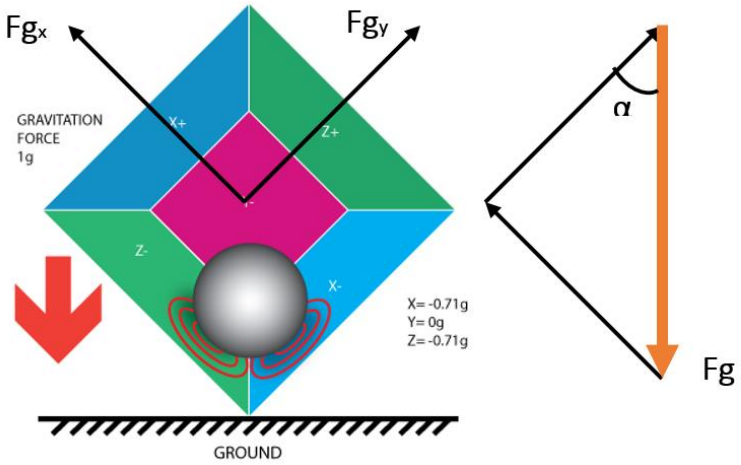


Figura 56. *Lectura del acelerómetro, en estado de equilibrio y girado 45 grados.*

Supongamos que no conocemos ese ángulo de 45 grados, y disponemos de los datos del sensor. En el eje Z detecta 0.71 y en el eje X detecta 0.71. Si sacamos el módulo del vector de aceleración lineal utilizando el teorema de Pitágoras:

$$Fg = \sqrt{0.71^2 + 0.71^2} = 1g \rightarrow 9.8\, m/s^2$$

de ahí que ya podemos calcular el ángulo:

$$\alpha = \arccos\left(\frac{0.707}{1}\right) = 45°$$

Ahora bien, para medir los dos ángulos de rotación de pitch y roll obtenemos que:

$$Acc_{xyz} = \sqrt{acc_x{}^2 + acc_y{}^2 + acc_z{}^2}$$

Siendo: acc_x, acc_yy acc_z la fuerza detectada por el sensor en los ejes X,Y,Z respectivamente. De modo que, los ángulos pitch y roll son:

$$pitch = \arccos(\frac{acc_y}{Acc_{xyz}})$$

$$roll = \arccos(\frac{acc_y}{Acc_{xyz}})$$

Los ángulos obtenidos con el acelerómetro son muy precisos y buenos siempre y cuando el sensor este en reposos (sólo con la aceleración lineal causada por la gravedad), si aparecen nuevas aceleraciones lineales ya sean debidas al movimiento del cuadricóptero o debido a vibraciones, los valores o ángulos son erróneos.

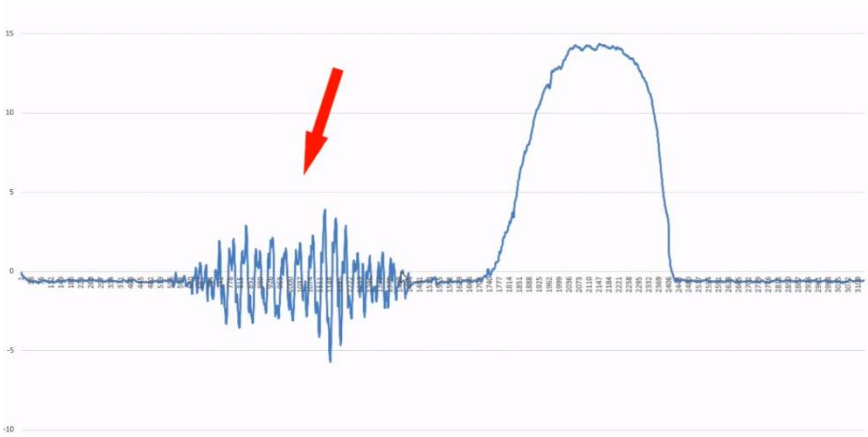


Figura 57. *Lectura del acelerómetro, izquierda con vibraciones derecha medición de ángulos sin vibraciones.*

Filtrando la señal aplicando el promedio todo ese ruido no deseado desaparecer, pero el ángulo medido se retrasa, es decir estaríamos midiendo otro ángulo lo que vuelve este sensor totalmente inútil. Sin embargo, veremos que podría resultarnos útil a la hora de medir los ángulos con el giroscopio y también para el control de altura.

Para medir la posición con el giroscopio simplemente tenemos que multiplicar la velocidad angular actual por el tiempo transcurrido entre la primera lectura del sensor y la segunda y sumar el ángulo anterior (inicialmente cero).

Veamos el siguiente ejemplo:

$$pitch_{gyro-1} = 0\,°/s \qquad pitch_{gyro} = 80\,°/s \qquad t = 4\,ms$$

$$angle_{pitch_{gyro}} = angle_{pitch_{gyro}} + pitch_{gyro} \times t$$

$$angle_{pitch_{gyro}} = 0 + 80 \times 0.004 = 0{,}32°$$

En lenguaje C:

```
    // 1 °/s = 65.5 // refresh rate 400 Hz = 0.0025
//0.000061538 = (1 / 65.5) * (1 / 400 Hz)
angle_pitch_gyro = angle_pitch_gyro + (Acc_Y * 0.000061538);
angle_roll_gyro = angle_roll_gyro + (Acc_X * 0.000061538);
```

Ahora bien, esto es válido solo si estamos girando sobre el eje X o solo el eje Y, si bien estamos girando sobre ambos ejes tendríamos que tener en cuenta las rotaciones sobre el eje Z para obtener el ángulo real. Es decir, si giramos sobre el eje X 30 grados, el sensor detecta perfectamente ese ángulo, pero, estando en esta posición si giramos sobre el eje Z, estaríamos cambiando el ángulo del eje X y también del eje Y, pero con el código anterior la respuesta no sería válida. De modo que, teniendo en cuenta las rotaciones sobre el eje Z se obtiene:

```
//0.000001074 = 0.000061538 * (3.142(PI) / 180degr)
angle_pitch_gyro = (angle_pitch_gyro + angle_roll_gyro * sin(Acc_Z *
0.000001074));
angle_roll_gyro = (angle_roll_gyro - angle_pitch_gyro * sin(Acc_Z *
0.000001074));
```

Teniendo en cuenta que con el tiempo estando en una misma posición el giroscopio acumula error, de modo que, si esta en reposos y marca 0 grados al cabo de unos segundos podría estar marcando 2 grados, por ello necesitamos compensar ese error y aquí es donde interviene el acelerómetro.

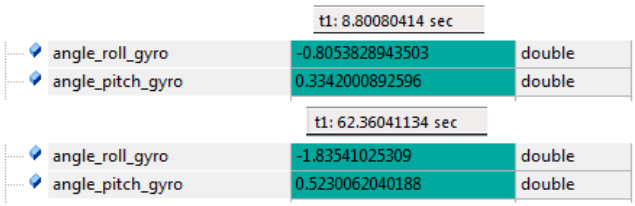


Figura 58. *Acumulación de error al medir ángulos con el giroscopio.*

De modo que, si el ángulo medido con el acelerómetro es igual a cero (siendo este más preciso en ausencia de movimientos de traslación) tendiéramos que el ángulo medido con el giroscopio es igual al mismo por 0.9996 siendo una multiplicación suficiente a lo largo del tiempo como para compensar la acumulación de error.

```
angle_roll_gyro = (angle_roll_gyro*0.9996 + angle_roll_acc *0.0004);
angle_pitch_gyro = (angle_pitch_gyro*0.9996 +angle_pitch_acc*0.0004);
```

Finalmente implementamos el controlador PID, que es muy parecido al que se realizó para el control de velocidades angulares, el único dato que varía es que el error es la diferencia entre el ángulo deseado y el ángulo medido. La acción de control total aplicada a cada uno de los rotores se mantiene igual que para el control por velocidades angulares, sólo que con los nuevos parámetros obtenidos a partir del control por posición angular.

```
   /* Calculamos el error */
   roll_error = roll_input_gyro - (rc_ch1_roll/10);
   /* Calculamos la acción del proporcional */
   roll_P = roll_error * KP_roll_M2;
   /* Calculamos la acción del integral */
   roll_I = roll_I + (roll_error * KI_roll_M2);
   /* La acumulación del integral incrementa de manera muy rapida,
nos debemos asegurar de que no sobrepase un límite. */
   if(roll_I>PID_max_roll)roll_I = PID_max_roll;
   else if(roll_I<(PID_max_roll*(-1)))roll_I = PID_max_roll*(-1);
   /* Calculamos la acción del derivatiov */
   roll_D = (roll_error - roll_error_last) * KD_roll_M2;
   /* Obtenemos la acción del PID_roll */
   PID_roll = roll_P + roll_I + roll_D;
   /* Nos aseguramos de que no sobrepase un límite */
   if(PID_roll>PID_max_roll)PID_roll = PID_max_roll;
   else if(PID_roll<(PID_max_roll*(-1)))PID_roll = PID_mx_roll*(-1);
/* Almaenamos el error atual, que utilizaremos en el siguiente
cálculo */
   roll_error_last = roll_error;
```

7.2.2 Control de altura

La segunda mejora consiste en diseñar un regulador que sea capaz de mantener el cuadricóptero a la altura deseada, para ello, precisamos de un sensor que pueda medir alturas. Sin embargo, al no disponer de ninguno, se ha decidido utilizar el acelerómetro. De modo que, suponiendo que cuando la aeronave está en una altura fija el módulo de la aceleración lineal medida con el acelerómetro, sólo debería corresponderse con la aceleración de la gravedad. El control consiste en mantener la velocidad lineal constante a cero de modo que si crece o decrece ejecutar la acción de control necesaria para volver al estado inicial.

En primer lugar, debemos obtener la velocidad lineal a partir de la aceleración lineal. Para ello, debemos calcular el módulo de la aceleración lineal y conocido el tiempo transcurrido entre la toma de mediciones obtenemos el módulo de la velocidad lineal.

```
Acc_total =sqrt((pow((Acc_X),2))+(pow((Acc_Y),2))+(pow((Acc_Z),2)));

//Dejamos un margen de error //4026 equivale a 1g lo que es 9.8 m/s2
   if(Acc_total<4050 && acc_total>3900){
      Acc_total = 4026;
      vel = 0;
   }
   //0.000973671 = (1 / 4026) * 9.8 * (1 / 400 Hz)
   vel = vel + ((acc_total-4026) * 0.000973671);
```

El controlador PID se muestra a continuación, nótese que el error es la velocidad lineal, por lo tanto, siempre que la velocidad lineal sea distinta de cero habrá error y por tanto acción de control.

```
void PID_Z(void){
   // El error de velocidad
   z_error = vel;
   // Accioón proporcional
   z_P = yaw_error * KP_Z;
   // Accioón integral
   z_I = yaw_I + (yaw_error * KI_Z);
   // Acción de control
   PID_z = z_P + z_I;
   // Limitamos la acción de control
   if(PID_z>PID_max_z)PID_z = PID_max_z;
   else if(PID_z<(PID_max_z*(-1)))PID_z = PID_max_z*(-1);
   // Error anterior es igual al error actual
    z_error_last = z_error;
}
```

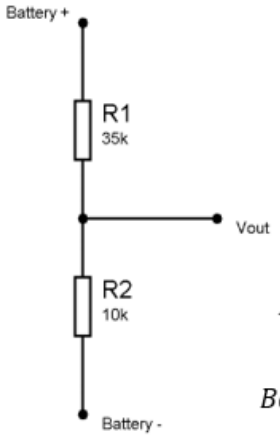
Finalmente debemos aplicar la acción de control a cada uno de los rotores. Teniendo en cuenta el sistema de referencias escogido, cuando el cuadricóptero se eleva la velocidad lineal es positiva y si baja negativa. Se obtiene que:

```
   /* Acción de control */
   ESC1 = (throttle-PID_z) - PID_pitch + PID_roll - PID_yaw;
   ESC2 = (throttle-PID_z) + PID_pitch + PID_roll + PID_yaw;
   ESC3 = (throttle-PID_z) + PID_pitch - PID_roll - PID_yaw;
   ESC4 = (throttle-PID_z) - PID_pitch - PID_roll + PID_yaw;
```

Es importante señalizar que la acción de PID para el control de altura sólo se aplica cuando está en posición horizontal, es decir cuando no se reciben señales de giro a partir del mando RC y además cuando la aceleración de los rotores ordenada con el *stick* del mando es superior a 1450.

```
   /*control de altura */
   if(throttle>=1450 && rc_ch1_roll == 0 && rc_ch2_pitch == 0){
      PID_Z();
   }
   else{
      PID_z = 0;
   }
```

Por otro lado, también resulta interesante considerar el estado de la batería puesto que cuando empieza a descargarse, a lo largo del tiempo la fuerza de empuje disminuye. Para consultar el estado de la batería se ha implementado un divisor de tensión.



$$Vout = \frac{Battery}{R1 + R2} \times R2$$

$$Battery_{full} \rightarrow Vout_{max} = \frac{12.6}{(10 + 35) \times 10E - 3} \times 10E - 3 = 2.8\,V$$

$$Battery_{empty} \rightarrow Vout_{min} = \frac{9.6}{(10 + 35) \times 10E - 3} \times 10E - 3 = 2.13\,V$$

Figura 59. *Divisor de tensión para medir estado de la batería.*

Se ha configurado el ADC, de modo que, si detecta 3 V equivale a lectura digital de 4095 (12 bit ADC) y cuando detecta 0 V la lectura es 0. Entonces para 2.8 V sería 3822. La sensibilidad es 1260/4095 (mV/V). De ahí que el voltaje de la batería se puede calcular de la siguiente forma:

```
    // Voltaje de la batería en mV
    // 274 compensación = 4095 - 3822
    ADC_ConvertedValue = ADC_Read();
    battery_voltage = (ADC_ConvertedValue + 274) * 0.30769;
```

De modo que, la acción de control aplicada a los rotores incremente con la caída de tensión.

```
if (battery_voltage < 1230 && battery_voltage > 920){
    ESC1 = ESC1 + ESC1 * ((1230 - battery_voltage)/(float)3500);
    ESC2 = ESC2 + ESC2 * ((1230 - battery_voltage)/(float)3500);
    ESC3 = ESC3 + ESC3 * ((1230 - battery_voltage)/(float)3500);
    ESC4 = ESC4 + ESC4 * ((1230 - battery_voltage)/(float)3500);
}
```

CONCLUSIÓN Y TRABAJOS FUTUROS

"Un científico debe tomarse la libertad de plantear cualquier cuestión, de dudar de cualquier afirmación, de corregir errores"
Julius Robert Oppenheimer (1904-1967)

Se ha diseñado y construido un cuadricóptero capaz de posicionarse a una altura deseada y mantener la estabilidad siempre y cuando no se den ordenes de cambio de orientación. Sin embargo, este proyecto aún está en versión de prueba puesto que hay una serie de aspectos que se deben mejorar y en concreto la seguridad del código implementado. Se deben implementar más medidas de seguridad siendo un ejemplo claro el código utilizado para extraer información de los sensores mediante la comunicación I2C. Si analizamos el código veremos que existen muchos bucles *while* que en caso de que no se cumpla alguna de las condiciones el sistema permanece en un bucle infinito ocasionando la pérdida total de control con la aeronave.

Es importante mencionar que en la simulación (modelo teórico) los parámetros de entrada de los tres PIDs (pitch, roll y yaw), por una parte son, las tres posiciones angulares *phi (ϕ), theta (θ), psi (ψ)* introducidas manualmente y por otra los tres valor estimado de la posición angular del cuadricóptero obtenido a partir de las ecuaciones de la dinámica del cuadricóptero. También aparece la velocidad angular que se ha utilizado para calcular la acción de la parte derivativa, que si nos fijamos en el diagrama de bloques (figura 31), lo hace para no tener que derivar la posición angular, aunque de este modo no se tiene en cuenta el error actual de la velocidad angular de forma directa.

Por otro lado, en el sistema real (modelo empírico) el control no se realiza de la misma forma. Los parámetros de entrada no son posiciones angulares, sino que, velocidades angulares, puesto que el sensor también mide velocidades angulares. De modo que, el error de los controladores PIDs es error de velocidad angular. El motivo es que, de esta forma se simplifica el problema y el control depende totalmente del sensor. De la otra forma, para calcular la posición angular tendríamos que estimar esa posición angular y para ello no podríamos utilizar solo el giroscopio, sino que también necesitaríamos otros sensores como el acelerómetro o el magnetómetro. En conclusión, la parte de simulación nos ha servido más bien para comprender el funcionamiento y no tanto para comparar con el sistema real.

Después de realizar varias pruebas de vuelo y analizar el comportamiento del cuadricóptero se aprecia la necesidad de un regulador de altura ya que resulta casi imposible mantener una altura estable. Sin embargo, para ello, necesitaríamos un sensor que nos aporte esa información puesto que el giroscopio solo mide velocidades angulares, el más adecuado sin lugar a duda es el barómetro. Sin embargo, al no disponer de alguno, finalmente se ha decidido utilizar el acelerómetro y diseñar un algoritmo de control de altura en base a los datos que nos ofrece ese sensor.

Finalmente, con el control por posiciones angulares se han solucionado el problema presente en el control por velocidades angulares que implicaba la necesidad de despegar el cuadricóptero a partir de una posición totalmente horizontal y con el control de altura se consigue más o menos permanecer en una posición, aunque digamos que se puede mejorar bastante el algoritmo de control, pero para ello sería imprescindible incorporar también el barómetro.

En cuanto a los trabajos futuros, en primer lugar, sería obligatorio optimizar y garantizar la seguridad del código, establecer una serie de pautas que impidan la pérdida de control con la aeronave, tales como, condiciones de retorno en caso de permanecer en un bucle infinito. En segundo lugar, resultaría interesante limitar el uso de la CPU a la hora de extraer información de los sensores y como alternativa utilizar el acceso directo a la memoria DMA. Una transferencia DMA consiste principalmente en copiar un bloque de memoria de un dispositivo a otro. En lugar de que la CPU inicie la transferencia, esta se lleva a cabo por el controlador DMA. Un ejemplo típico es mover un bloque de memoria desde una memoria externa a una interna más rápida. Tal operación no ocupa al procesador y, por ende, éste puede efectuar otras tareas. Las transferencias DMA son esenciales para aumentar el rendimiento de aplicaciones que requieran muchos recursos.

Por último, para profundizar aún más en las posibilidades y aplicaciones de los cuadricóptero podríamos diseñar un control avanzado, por ejemplo, un control de trayectorias con el cual podríamos convertir la aeronave en un dispositivo totalmente autónomo. Para ello necesitaríamos incorporar aún más sensores, siendo imprescindible la utilización de un módulo GPS y una brújula digital.

BIBLIOGRAFÍA

¿Que es un ESC opto? (September de 2014). Obtenido de http://www.multicopters.es/: http://www.multicopters.es/foro/vbulletin/archive/index.php/t-3966.html

avayan. (29 de August de 2009). *Understanding PWM.* Obtenido de http://ebldc.com/?p=48

Comunication, K. C. (2008). *Control of quadcopter.* Obtenido de kth: https://www.nada.kth.se/utbildning/grukth/exjobb/rapportlistor/2008/rapporter08/sikiric_vedran_08027.pdf

dramida. (8 de September de 2012). *Attitude hold improvment solution.* Obtenido de multiwi: http://www.multiwii.com/forum/viewtopic.php?f=8&t=2371&start=40

ESC Electronic Speed Controller. (30 de May de 2013). Obtenido de http://www.algunascosas.com/esc-electronic-speed-controller

FERNÁNDEZ, L. S. (Juny de 2014). *MODELADO Y CONTROL DE UN.* Obtenido de ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI): http://www.iit.comillas.edu/pfc/resumenes/538ddc6870df6.pdf

Gallardo, E. M. (May de 2015). *Modelado matemático y control inteligente de un cuadricóptero.* Obtenido de https://www.researchgate.net/publication/280938919_Modelado_matematico_y_control_inteligente_de_un_cuadricoptero_Mathematical_modelling_and_control_of_a_quadcopter#pf23

LHelge. (10 de April de 2012). *Pitch and roll estimating Kalman filter for stabilizing quadrocopters.* Obtenido de http://www.linushelgesson.se/2012/04/pitch-and-roll-estimating-kalman-filter-for-stabilizing-quadrocopters/

mandamealgo. (Juny de 2012). *¿Qué es un ESC? y el BEC ?* Obtenido de http://www.multicopters.es/: http://www.multicopters.es/foro/vbulletin/showthread.php?399-Variadores-%BFQu%E9-es-un-ESC-y-el-BEC-(Explicaci%F3n)

MEM Senior Design Team 37. (2014). *Quadcopter Dynamic Modeling and Simulation Using MATLAB and Simulink.* Obtenido de github.com/dch33/Quad-Sim: https://www.youtube.com/watch?v=kIgwZFGlgio

Montes, D. F. (2013). *SISTEMA DE CONTROL PARA LA ESTABILIDAD Y ORIENTACIÓN DE UN HELICÓPTERO QUADROTOR.* Obtenido de eia: http://repository.eia.edu.co/bitstream/11190/734/1/MECA0125.pdf

Roberto. (17 de May de 2014). *¿Qué es un cuadricoptero?* Obtenido de http://cuadricoptero.org/que-es-un-cuadricoptero/

robot-electronics. (s.f.). *Using the I2C Bus.* Obtenido de http://www.robot-electronics.co.uk/i2c-tutorial

Shahryiar, S. (1 de November de 2015). *STM32 Timers.* Obtenido de http://embedded-lab.com/blog/stm32-timers/

starlino. (29 de December de 2019). *A guide to using IMU.* Obtenido de starlino: http://www.starlino.com/imu_guide.html

totalphase. (s.f.). *7-bit, 8-bit, and 10-bit I2C Slave Addressing.* Obtenido de http://www.totalphase.com/support/articles/200349176-7-bit-8-bit-and-10-bit-I2C-Slave-Addressing

Tr4nsduc7or. (15 de Octubre de 2014). *Tutorial de Arduino y MPU-6050.* Obtenido de robologs: http://robologs.net/2014/10/15/tutorial-de-arduino-y-mpu-6050/

Vehicles, O. S. (s.f.). *Build your own quadcopter.* Obtenido de tech-insider: http://tech-insider.org/diy-drones/research/acrobat/1209.pdf

visualgdb. (4 de February de 2014). *Controlling STM32 Hardware Timers with Interrupts.* Obtenido de http://visualgdb.com/tutorials/arm/stm32/timers/

vueloartificial. (s.f.). *La electrónica de vuelo para una aeronave no tripulada.* Obtenido de http://vueloartificial.com/: http://vueloartificial.com/introduccion/primeros-pasos/la-electronica-de-vuelo/

Wikipedia. (s.f.). *Batería eléctrica.* Obtenido de https://es.wikipedia.org/wiki/Bater%C3%ADa_el%C3%A9ctrica#Capacidad_de_carga

Директ, Я. (2014). *I2C в STM32 на примере компаса HMC5883L и stm32f4 discovery.* Obtenido de http://blablacode.ru/mikrokontrollery/501

ANEXOS

I. MPU6050 datasheet

Características del giroscopio: se muestran las escalas escogidas

6 Electrical Characteristics

6.1 Gyroscope Specifications

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
GYROSCOPE SENSITIVITY						
Full-Scale Range	FS_SEL=0		±250		°/s	
	FS_SEL=1		±500		°/s	
	FS_SEL=2		±1000		°/s	
	FS_SEL=3		±2000		°/s	
Gyroscope ADC Word Length			16		bits	
Sensitivity Scale Factor	FS_SEL=0		131		LSB/(°/s)	
	FS_SEL=1		65.5		LSB/(°/s)	
	FS_SEL=2		32.8		LSB/(°/s)	
	FS_SEL=3		16.4		LSB/(°/s)	
Sensitivity Scale Factor Tolerance	25°C	-3		+3	%	
Sensitivity Scale Factor Variation Over Temperature			±2		%	
Nonlinearity	Best fit straight line; 25°C		0.2		%	
Cross-Axis Sensitivity			±2		%	
GYROSCOPE ZERO-RATE OUTPUT (ZRO)						
Initial ZRO Tolerance	25°C		±20		°/s	
ZRO Variation Over Temperature	-40°C to +85°C		±20		°/s	
Power-Supply Sensitivity (1-10Hz)	Sine wave, 100mVpp; VDD=2.5V		0.2		°/s	
Power-Supply Sensitivity (10 - 250Hz)	Sine wave, 100mVpp; VDD=2.5V		0.2		°/s	
Power-Supply Sensitivity (250Hz - 100kHz)	Sine wave, 100mVpp; VDD=2.5V		4		°/s	
Linear Acceleration Sensitivity	Static		0.1		°/s/*g*	
SELF-TEST RESPONSE						
Relative	Change from factory trim	-14		14	%	1
GYROSCOPE NOISE PERFORMANCE	**FS_SEL=0**					
Total RMS Noise	DLPFCFG=2 (100Hz)		0.05		°/s-rms	
Low-frequency RMS noise	Bandwidth 1Hz to10Hz		0.033		°/s-rms	
Rate Noise Spectral Density	At 10Hz		0.005		°/s/ √ Hz	
GYROSCOPE MECHANICAL FREQUENCIES						
X-Axis		30	33	36	kHz	
Y-Axis		27	30	33	kHz	
Z-Axis		24	27	30	kHz	
LOW PASS FILTER RESPONSE						
	Programmable Range	5		256	Hz	
OUTPUT DATA RATE						
	Programmable	4		8,000	Hz	
GYROSCOPE START-UP TIME	**DLPFCFG=0**					
ZRO Settling (from power-on)	to ±1°/s of Final		30		ms	

1. Please refer to the following document for further information on Self-Test: *MPU-6000/MPU-6050 Register Map and Descriptions*

Características del acelerómetro: se muestran las escalas escogidas

InvenSense	MPU-6000/MPU-6050 Product Specification	Document Number: PS-MPU-6000A-00 Revision: 3.4 Release Date: 08/19/2013

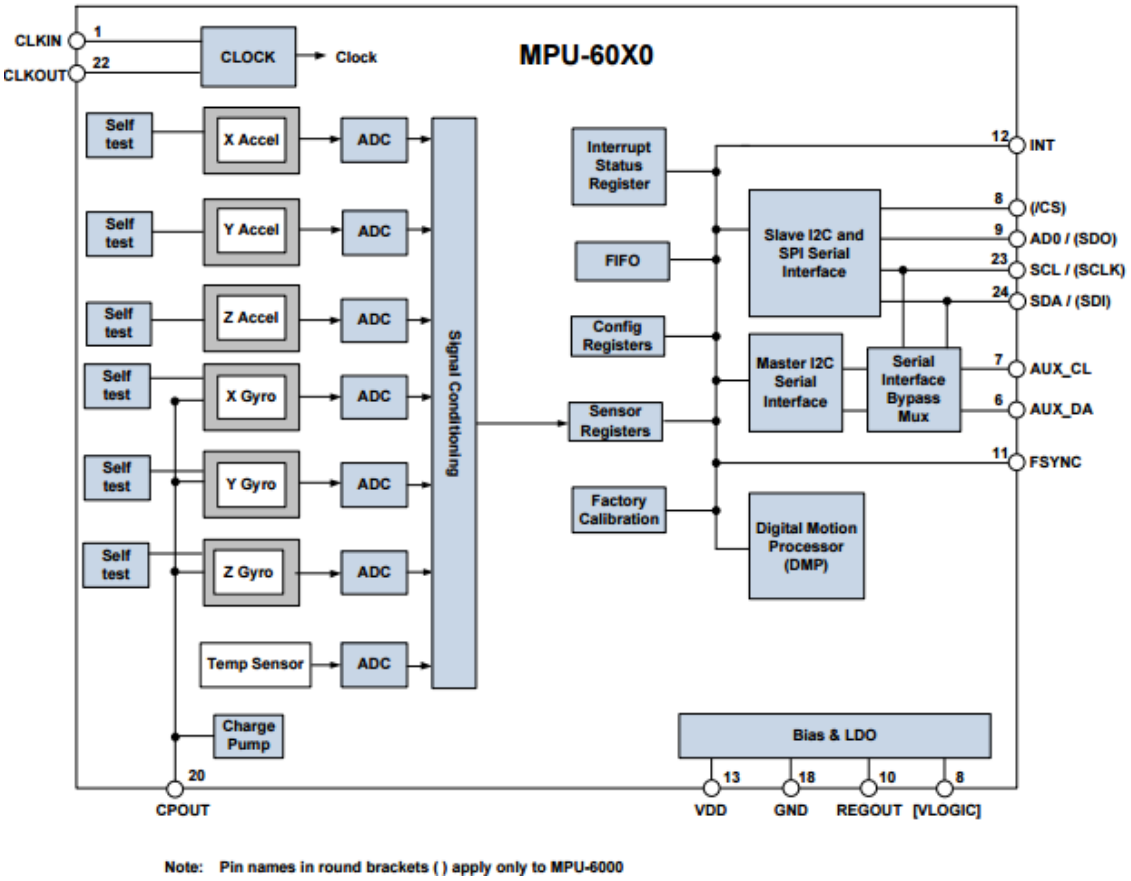
6.2 Accelerometer Specifications

VDD = 2.375V-3.46V, VLOGIC (MPU-6050 only) = 1.8V±5% or VDD, T_A = 25°C

PARAMETER	CONDITIONS	MIN	TYP	MAX	UNITS	NOTES
ACCELEROMETER SENSITIVITY						
Full-Scale Range	AFS_SEL=0		±2		*g*	
	AFS_SEL=1		±4		*g*	
	AFS_SEL=2		±8		*g*	
	AFS_SEL=3		±16		*g*	
ADC Word Length	Output in two's complement format		16		bits	
Sensitivity Scale Factor	AFS_SEL=0		16,384		LSB/*g*	
	AFS_SEL=1		8,192		LSB/*g*	
	AFS_SEL=2		4,096		LSB/*g*	
	AFS_SEL=3		2,048		LSB/*g*	
Initial Calibration Tolerance			±3		%	
Sensitivity Change vs. Temperature	AFS_SEL=0, -40°C to +85°C		±0.02		%/°C	
Nonlinearity	Best Fit Straight Line		0.5		%	
Cross-Axis Sensitivity			±2		%	
ZERO-G OUTPUT						
Initial Calibration Tolerance	X and Y axes		±50		mg	1
	Z axis		±80		mg	
Zero-G Level Change vs. Temperature	X and Y axes, 0°C to +70°C		±35			
	Z axis, 0°C to +70°C		±60		mg	
SELF TEST RESPONSE						
Relative	Change from factory trim	-14		14	%	2
NOISE PERFORMANCE						
Power Spectral Density	@10Hz, AFS_SEL=0 & ODR=1kHz		400		µg/√Hz	
LOW PASS FILTER RESPONSE						
	Programmable Range	5		260	Hz	
OUTPUT DATA RATE						
	Programmable Range	4		1,000	Hz	
INTELLIGENCE FUNCTION INCREMENT			32		mg/LSB	

1. Typical zero-g initial calibration tolerance value after MSL3 preconditioning
2. Please refer to the following document for further information on Self-Test: *MPU-6000/MPU-6050 Register Map and Descriptions*

Diagrama del MPU6050, se puede observar que para cada eje hay un ADC, esto nos permite obtener información para cada uno de los ejes al mismo tiempo



Note: Pin names in round brackets ( ) apply only to MPU-6000

PRESUPUESTO

A continuación, se adjunta el presupuesto que incluye todos los gastos que suponen la realización del presente proyecto.

1. PRESUPUESTO COMPONENTES DEL CUADRICOPTERO

COMPONENTES DEL CUADRICÓPTERO					
Ref	**Ud**	**Descripción**	**Cantidad**	**Precio**	**Total**
01.01	**ud**	**Estructura Quanum Falcon Billet**	**1**	**29,99 €**	**29,99 €**
01.02	**ud**	**Placa STM32F4 Discovery**	**1**	**20,00 €**	**20,00 €**
01.03	**ud**	**Módulo GY- 86**	**1**	**10,91 €**	**10,91 €**
01.04	**ud**	**Motor DYS BE1806 2300 KV**	**4**	**8,44 €**	**33,76 €**
01.05	**ud**	**ESC DYS BL20A**	**4**	**9,60 €**	**38,40 €**
01.06	**ud**	**Mando y receptor RC Flysky FS-i6**	**1**	**44,95 €**	**44,95 €**
01.07	**ud**	**Helie Gemfan 5040 (2 pair)**	**2**	**1,17 €**	**2,34 €**
01.08	**ud**	**Conector XT30**	**1**	**1,70 €**	**1,70 €**
01.09	**ud**	**Helie Gemfan 5040 (2 pair)**	**2**	**1,17 €**	**2,34 €**
01.10	**ud**	**Cables**	**30**	**0,10 €**	**3,00 €**
01.11	**ud**	**Batería MultiStar 1400 mAh**	**1**	**16,00 €**	**16,00 €**
01.12	**ud**	**Otros accesorios**	**1**	**10,00 €**	**10,00 €**
01.13	**ud**	**Soporte para la batería**	**1**	**29,99 €**	**29,99 €**
01.14	**ud**	**Soporte para el microcontrolador**	**1**	**20,00 €**	**20,00 €**
01.15	**ud**	**Soporte para el receptor**	**1**	**10,91 €**	**10,91 €**
01.16	**ud**	**Soporte para el sensor**	**1**	**7,35 €**	**7,35 €**
		TOTAL PRESUPUESTO COMPONENTES DEL CUADRICÓPTERO			**281,64 €**

2. PRESUPUESTO MANO DE OBRA

MANO DE OBRA				
Ref	**Ud**	**Descripción**	**Precio**	**Total**
B0005.0001	10 h	Diseño piezas	5,00 €	50,00 €
B0005.0002	45 h	Programación del microcontrolador	5,00 €	225,00 €
B0005.0003	8 h	Montaje	5,00 €	40,00 €
		Subtotal mano de obra		**275,00 €**
		MEDIOS AUXILARES		
%0500	5,00%	Medios auxiliares	275,00 €	13,75 €
		Subtotal medios auxiliares		**13,75 €**
		COSTE PARTIDA		**288,75 €**

3. PRESUPUESTO TOTAL

1.00	COMPONENTES DEL CUADRICOPTERO	281,64 €
2.00	MANO DE OBRA	288,75 €
	TOTAL EJECUCIÓN MATERIAL	**570,39 €**

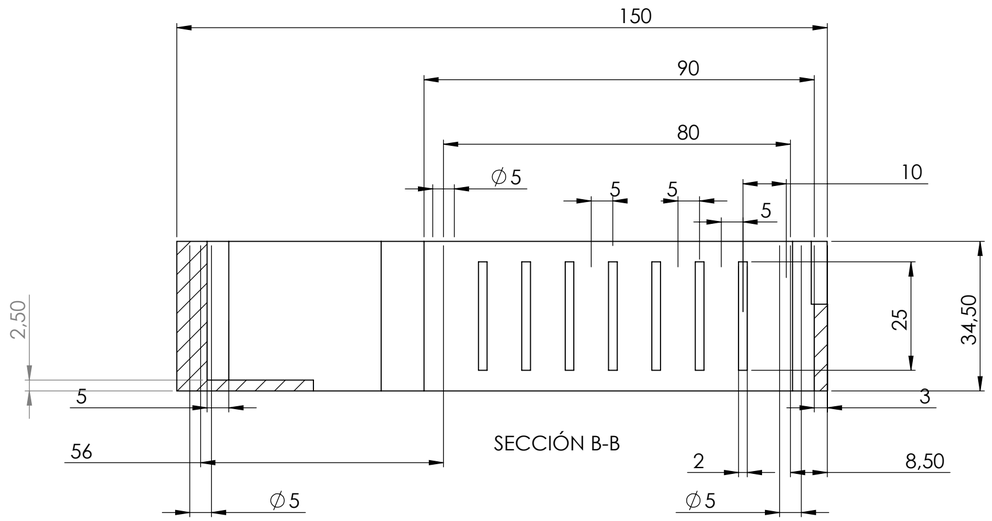
13,00 % Gastos generales	74,15 €
6,00 % Beneficio industrial	34,22 €
TOTAL SUMA G.G. Y B.I.	**108,37 €**

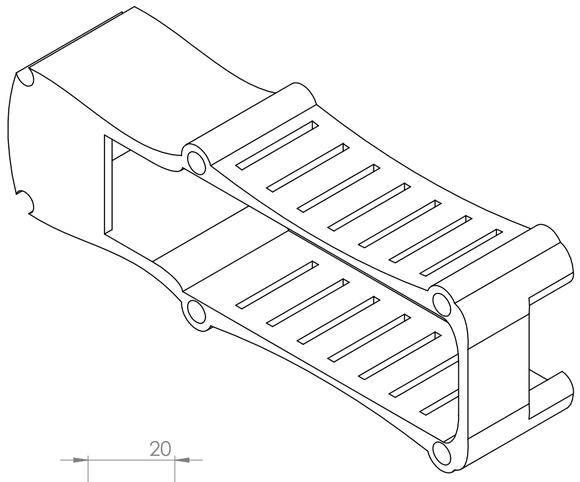
21,00 % I.V.A.	**142,54 €**

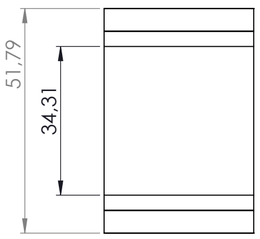
TOTAL PRESUPUESTO CONTRATA	**821,30 €**

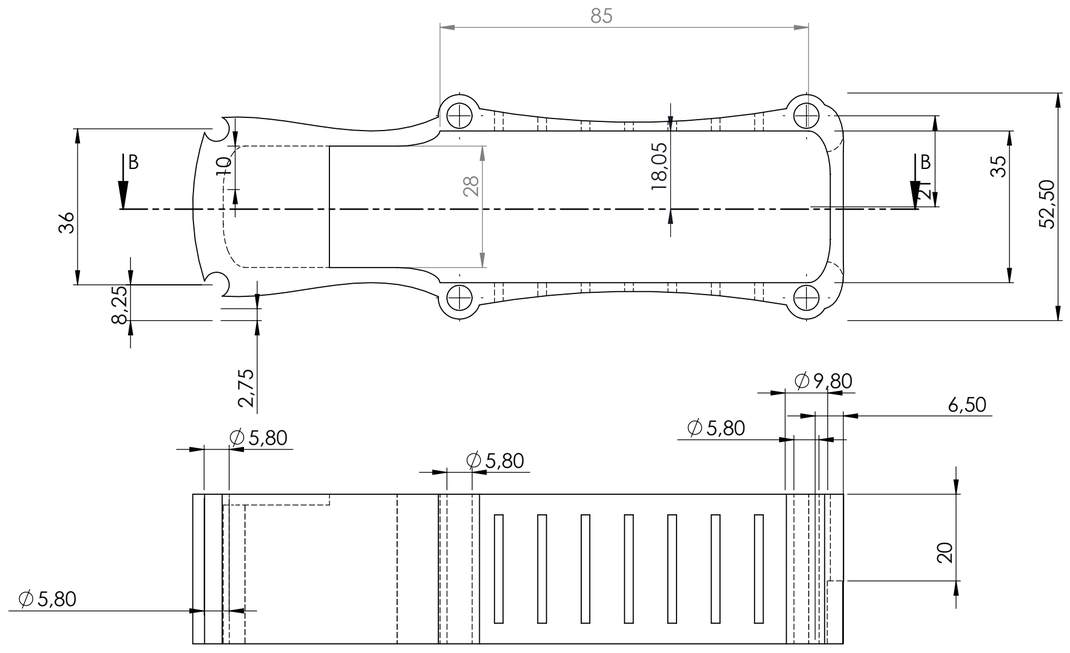
TOTAL PRESUPUESTO GENERAL	**821,30 €**

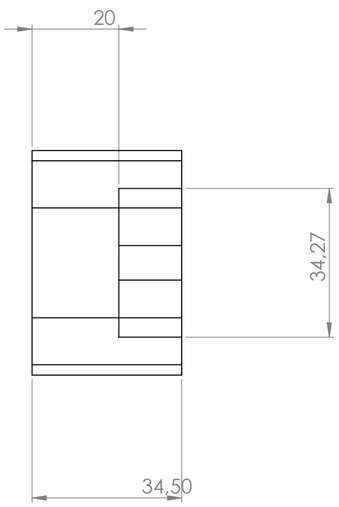
PLANOS



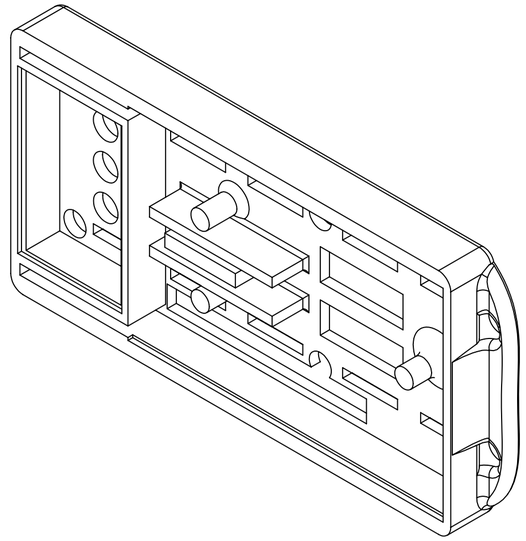


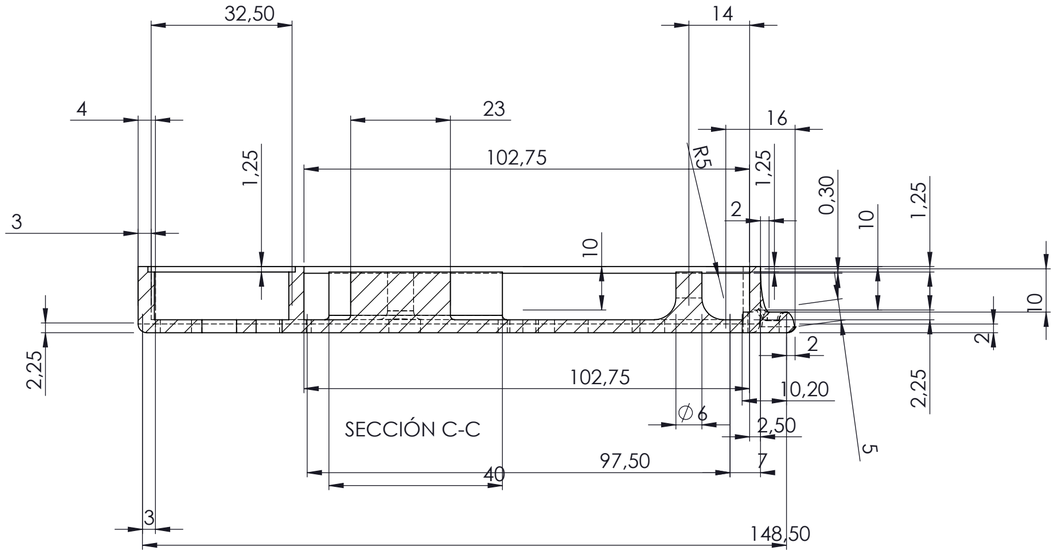


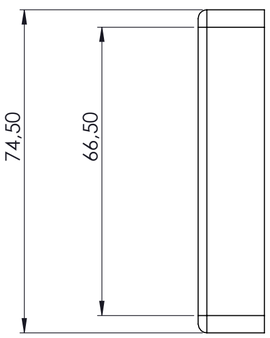


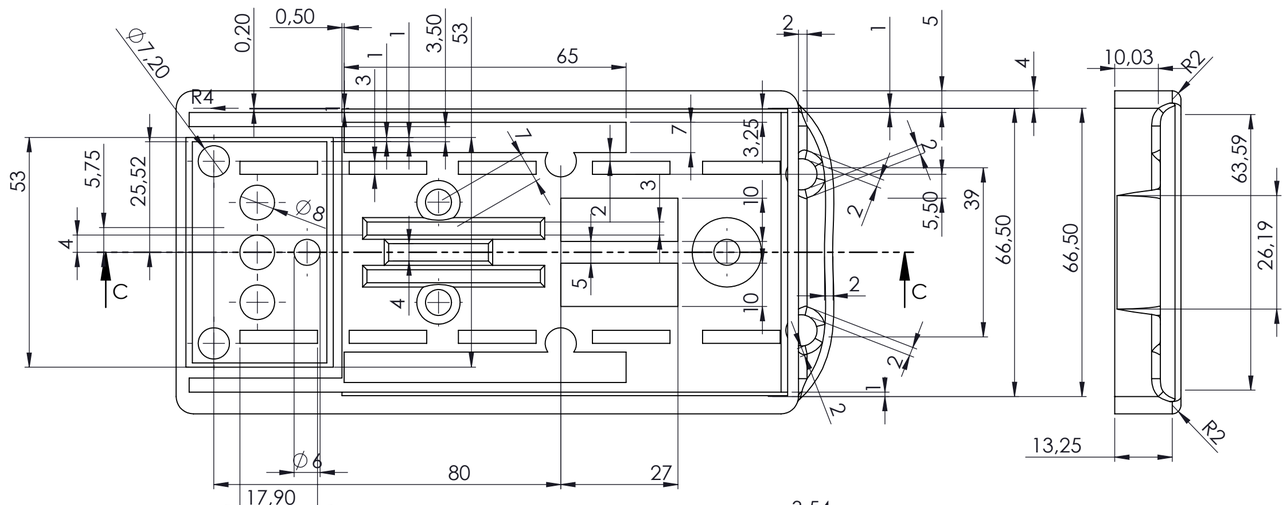


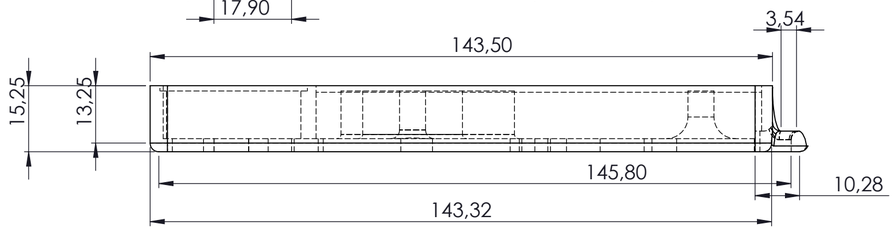
	NOMBRE	FECHA	DISEÑO E IMPLEMENTACIÓN DE UN CUADRICÓPTERO BASADO EN EL MICROCONTROLADR STM32F4
PROYECTISTA	EMILOV GORANOV, ALEKS	02/06/2016	
ESCALA 1:1	**SOPORTE PARA LA BATERÍA**		PLANO Nº 1



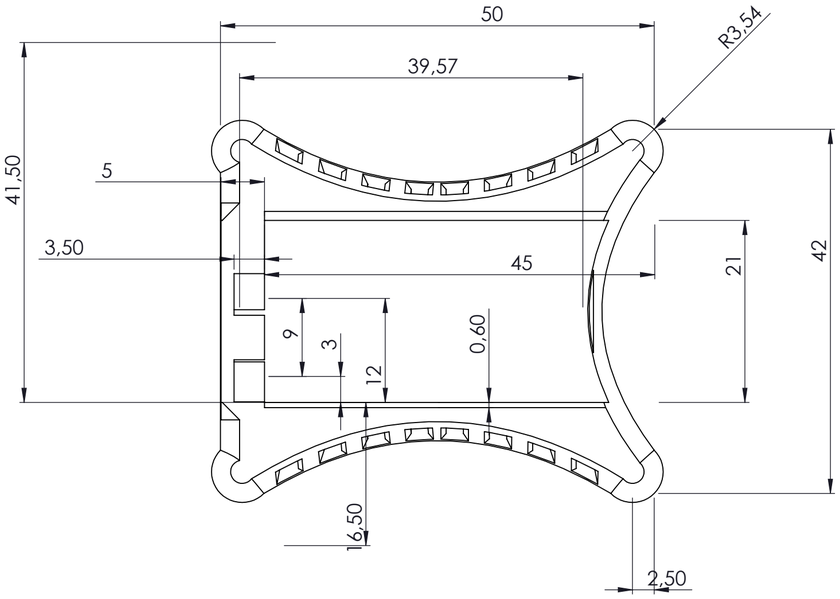


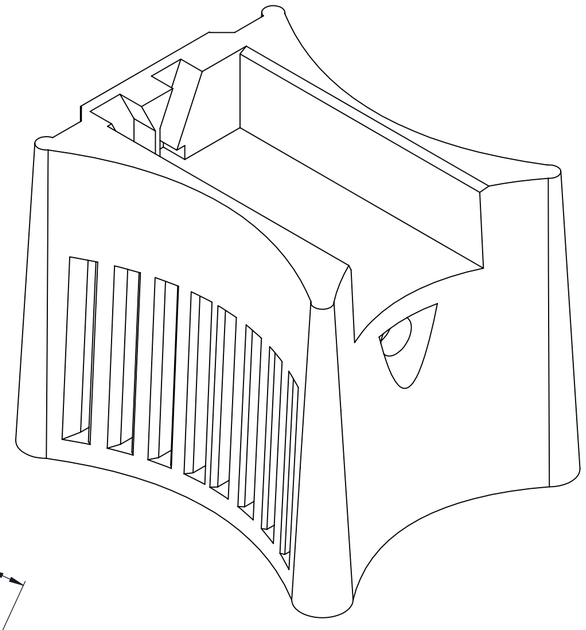


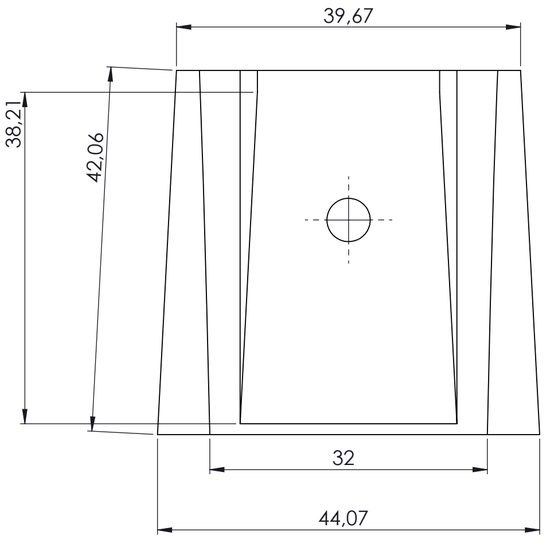


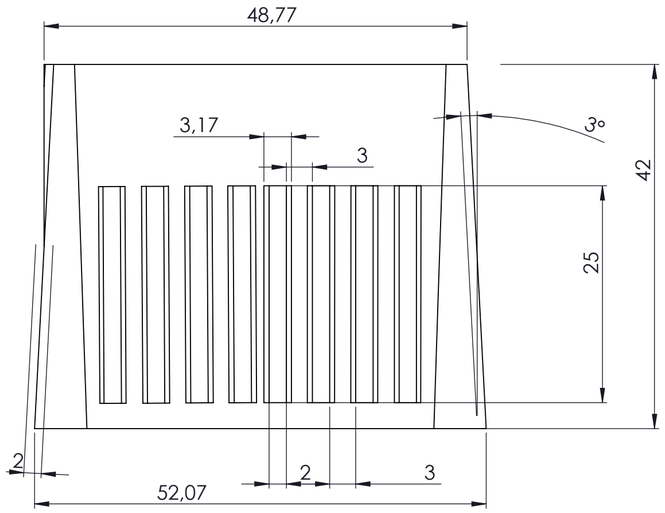


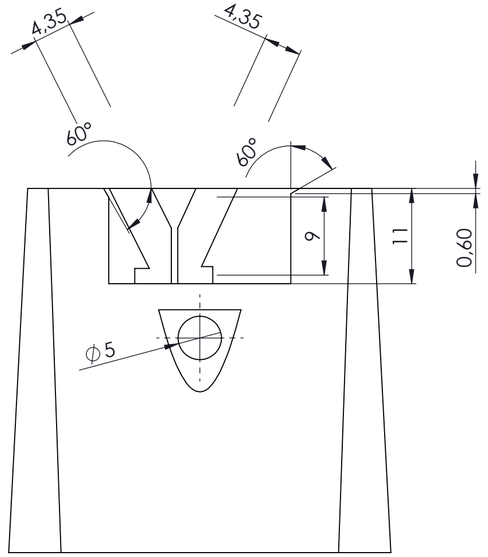
	NOMBRE	FECHA	DISEÑO E IMPLEMENTACIÓN DE UN CUADRICÓPTERO BASADO EN EL MICROCONTROLADR STM32F4
PROYECTISTA	EMILOV GORANOV, ALEKS	02/06/2016	
ESCALA 1:1	**SOPORTE PARA EL MICROCONTROLADOR**		PLANO Nº 2

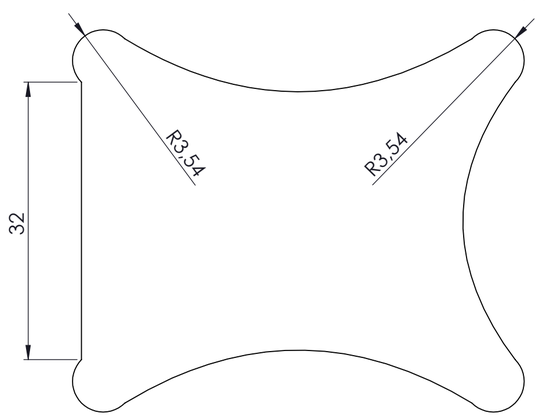




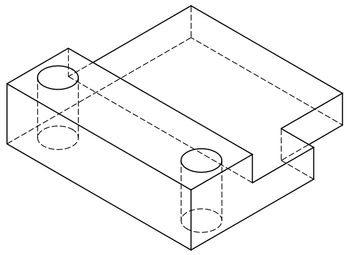


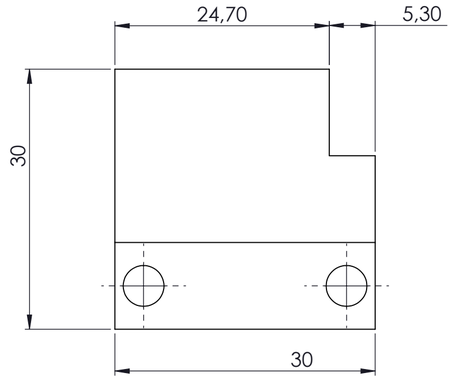


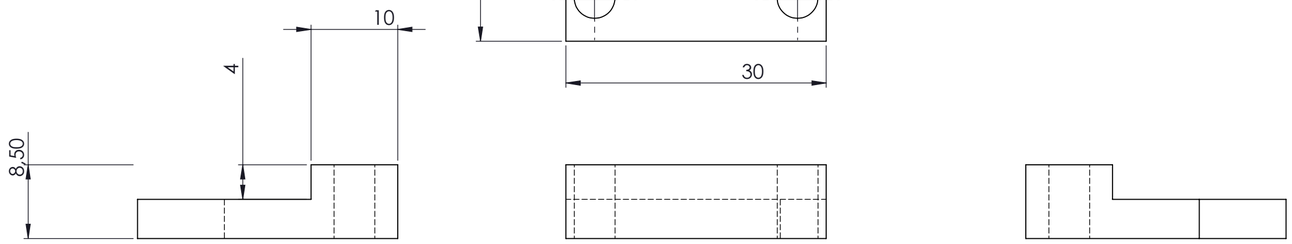


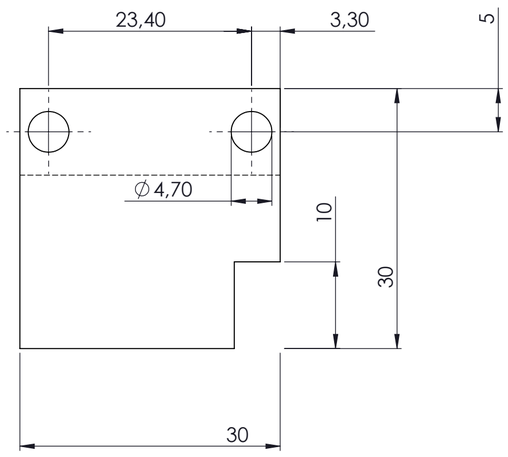


	NOMBRE	FECHA	DISEÑO E IMPLEMENTACIÓN DE UN CUADRICÓPTERO BASADO EN EL MICROCONTROLADR STM32F4
PROYECTISTA	EMILOV GORANOV, ALEKS	02/06/2016	
ESCALA 2:1	**SOPORTE PARA EL RECEPTOR Y EL MEDIDOR DE TENSIÓN**		PLANO Nº 3









	NOMBRE	FECHA	DISEÑO E IMPLEMENTACIÓN DE UN CUADRICÓPTERO BASADO EN EL MICROCONTROLADR STM32F4
PROYECTISTA	EMILOV GORANOV, ALEKS	02/06/2016	
ESCALA 2:1	**SOPORTE PARA EL SENSOR**		PLANO Nº 4