UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

PROYECTO FIN DE GRADO

CONTROL DE UN TRICÓPTERO PARA VUELO EN INTERIORES

Autor:
Jorge Buil García

Directores:
Juan Luis Zamora Macho
José Porras Galá

Madrid
Julio de 2016

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN ACCESO *ABIERTO* (RESTRINGIDO) DE DOCUMENTACIÓN

1º. Declaración de la autoría y acreditación de la misma.

El autor D. Jorge Buil García, como estudiante de la UNIVERSIDAD PONTIFICIA COMILLAS (COMILLAS), **DECLARA**

que es el titular de los derechos de propiedad intelectual, objeto de la presente cesión, en relación con la obra ‘‘Control de un tricóptero para vuelo en interiores” (Proyecto Fin de Grado)[1], que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual como titular único o cotitular de la obra.

En caso de ser cotitular, el autor (firmante) declara asimismo que cuenta con el consentimiento de los restantes titulares para hacer la presente cesión. En caso de previa cesión a terceros de derechos de explotación de la obra, el autor declara que tiene la oportuna autorización de dichos titulares de derechos a los fines de esta cesión o bien que retiene la facultad de ceder estos derechos en la forma prevista en la presente cesión y así lo acredita.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad y hacer posible su utilización de *forma libre y gratuita* (*con las limitaciones que más adelante se detallan*) por todos los usuarios del repositorio y del portal e-ciencia, el autor **CEDE** a la Universidad Pontificia Comillas de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución, de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra (a) del apartado siguiente.

3º. Condiciones de la cesión.

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia, el repositorio institucional podrá:

(a) Transformarla para adaptarla a cualquier tecnología susceptible de incorporarla a internet; realizar adaptaciones para hacer posible la utilización de la obra en formatos electrónicos, así como incorporar metadatos para realizar el registro de la obra e incorporar “marcas de agua” o cualquier otro sistema de seguridad o de protección.

(b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica,

[1] Especificar si es una tesis doctoral, proyecto fin de carrera, proyecto fin de Máster o cualquier otro trabajo que deba ser objeto de evaluación académica.

incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato…

(c) Comunicarla y ponerla a disposición del público a través de un archivo abierto institucional, accesible de modo libre y gratuito a través de internet[2].

(d) Distribuir copias electrónicas de la obra a los usuarios en un soporte digital[3]

4º. Derechos del autor.

El autor, en tanto que titular de una obra que cede con carácter no exclusivo a la Universidad por medio de su registro en el Repositorio Institucional tiene derecho a:

a) A que la Universidad identifique claramente su nombre como el autor o propietario de los derechos del documento.

b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.

c) Solicitar la retirada de la obra del repositorio por causa justificada. A tal fin deberá ponerse en contacto con el vicerrector/a de investigación (curiarte@rec.upcomillas.es).

d) Autorizar expresamente a COMILLAS para, en su caso, realizar los trámites necesarios para la obtención del ISBN.

e) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.

b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.

c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.

---

[2] En el supuesto de que el autor opte por el acceso restringido, este apartado quedaría redactado en los siguientes términos:
(c) Comunicarla y ponerla a disposición del público a través de un archivo institucional, accesible de modo restringido, en los términos previstos en el Reglamento del Repositorio Institucional

[3] En el supuesto de que el autor opte por el acceso restringido, este apartado quedaría eliminado.

d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

a) Deberes del repositorio Institucional:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.

- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusiva del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.

- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.

b) Derechos que se reserva el Repositorio institucional respecto de las obras en él registradas:

- retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 8 de Julio de 2016

ACEPTA

Fdo……

Proyecto realizado por el alumno:

Jorge Buil García

Fdo: Fecha: / /

Autorizada la entrega del proyecto cuya información no es de carácter confidencial

EL DIRECTOR DEL PROYECTO

Juan Luis Zamora Macho

Fdo: Fecha: / /

José Porras Galán

Fdo: Fecha: / /

Vº Bº del Coordinador de Proyectos

Álvaro Sánchez Miralles

Fdo: Fecha: / /

UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

PROYECTO FIN DE GRADO

CONTROL DE UN TRICÓPTERO PARA VUELO EN INTERIORES

Autor:
Jorge Buil García

Directores:
Juan Luis Zamora Macho
José Porras Galá

Madrid
Julio de 2016

Agradecimientos

A Juan Luis por su dedicación y esfuerzo

A Antonio y Néstor por las frustraciones superadas y el apoyo conjunto

A Javi, Germán y Álvaro por el día a día

A mi familia por su preocupación y ayudarme a perseverar

UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

PROYECTO FIN DE GRADO

MEMORIA

CONTROL DE UN TRICÓPTERO PARA VUELO EN INTERIORES

Autor:
Jorge Buil García

Directores:
Juan Luis Zamora Macho
José Porras Galá

Madrid
Julio de 2016

Índice

Capítulo 1

Introducción

La electrónica es la rama de la ingeniería que más ha evolucionado en los últimos treinta años. Gracias al desarrollo de la tecnología y al avance en los microprocesadores, junto a la reducción de su tamaño, es ahora posible controlar sistemas que antes podían resultarnos imposibles. Los sistemas de Control, junto a la Regulación Automática, han sido fundamentales para la profundización en medios de transporte aéreos. Los vehículos aéreos no tripulados (UAVs), conocidos comúnmente como *drones*, son uno de los últimos ejemplos de las grandes posibilidades que esta revolución tecnológica. Eran necesarias una alta estabilidad y precisión para llevar a cabo estos sistemas y por ello no se han podido desarrollar hasta que no han existido los medios electrónicos necesarios.

Encontramos muchos tipos de UAVs, nuestro objetivo de estudio son los denominados multi-rotor. Los más potentes son los multicópteros, vehículos dotados de varias hélices controladas de manera independiente que permiten desplazarlo tridimensionalmente y definir con precisión su posición en el espacio. Cada motor aporta un grado de libertad, por lo que normalmente se suele trabajar con cuatro (cuadricópteros), seis (hexacópteros) e incluso ocho (octocópteros) debido a que los cuerpos tridimensionales tienen seis grados de libertad.

En este proyecto, fuera de la normalidad, se va a utilizar un tricóptero. Cómo tiene tres motores (y por tanto tres grados de libertad) necesitamos agregarle otro para poder así controlarlo de manera eficiente en el espacio. Para ello se utilizará un servomotor en el motor trasero que otorgará un ángulo de inclinación α, consiguiendo así los cuatro grados de libertad. Se utilizará un tricóptero debido a la simplicidad de control frente a las alternativas con más hélices, también debido a que no supone una peor estabilidad. Se usará un Control adecuado para despegar verticalmente y mantenerse en el aire, lo llamaremos vuelo estacionario; seguir trayectorias concretas, que será tanto para vuelo manual como autónomo y finalmente aterrizar verticalmente. El s*oftware* de control será desarrollado en el entorno de Matlab/Simulink con la herramienta OpenPilot.

Está pensado para vuelos en interiores y utilizaremos dicha herramienta para poder aplicar posteriormente los resultados de este proyecto a la docencia en la Universidad.

1.1 Motivación del proyecto y sus objetivos

Este es un campo que aún está desarrollándose, el control de los multicópteros tiene un enorme potencial y precisan de una tecnología muy puntera. Con este proyecto se espera

aportar a la Universidad Pontificia de Comillas una nueva experiencia en este campo que puede llegar a ser muy importante de cara a futuras investigaciones y con aplicación práctica muy clara. Permite aplicar los conocimientos de diversas materias como máquinas eléctricas (motores *Brushless*), comunicaciones (interfaz de comunicación entre el ordenador y la nave), sistemas electrónicos (sensores y actuadores), electrónica de potencia (conexión a los motores y su tarjeta correspondiente) y mecánica (modelado)

La idea también es recuperar los resultados de proyectos de otros años con cuadricópteros, que, aunque no sean tricópteros ambos multicópteros comparten gran parte de similitudes a la hora de modelar y simular distintas técnicas de Control.

Por todas estas virtudes considero que es un proyecto muy adecuado para mi propio desarrollo académico, además, al haber elegido un multicóptero diferente al usual utilizado abro una nueva línea de investigación y realizamos, así, una contribución al avance en esta área para la universidad.

En este proyecto se persiguen los siguientes objetivos principales:

1. Adaptar el actual entorno de trabajo para el controlador OpenPilot, al no existir un *blockset* de Simulink específico para aeronaves, tendremos que desarrollar los *Driver blocks* con los que vamos a trabajar utilizando Matlab creando así un entorno de simulación apropiado.
2. Adaptación y desarrollo del modelo dinámico de un tricóptero para simulación.
3. Diseño del sistema de control de vuelo del tricóptero con la creación necesaria de un Mixeador para adaptar las salidas del control a los actuadores correspondientes.
4. Implantación del control y navegación guiada mediante una emisora de radiofrecuencia.
5. En simulación, posicionamiento autónomo del dron a distancia fija del suelo y control de estabilidad frente a pequeñas perturbaciones.

1.2 Metodología y recursos

Para alcanzar estos objetivos, se definen las siguientes tareas, que se recogen en el cronograma de la Figura 1.1:

- Instalación y configuración de la herramienta OpenPilot para Matlab/Simulink en los ordenadores de trabajo, y familiarización con ella.
- Revisión del modelo mecánico actual y adaptarlo para un tricóptero en Simulink e integración en un sistema de control para construir un simulador.
- Revisión de la bibliografía sobre avances en los UAVs y estudio del control predictivo-adaptativo.
- Diseño de un entorno de simulación en Matlab y Simulink.

- Diseño de una máquina de estados e implementación en Matlab/Simulink.
- Diseño del control de estabilización mediante *feedback linealization* para conseguir un control robusto.
- Diseño del Mixeador para transformar las fuerzas provenientes del control en los mandos actuadores del tricóptero.
- Ajuste de los parámetros del control para conseguir un control robusto en simulación.
- Ensayos en Tierra y ajustes de Parámetros para perfeccionar el tiempo de respuesta y la robustez del control en la planta física.
- Ensayos de vuelo manual una vez implantado todo el control con monitorización desde PC por transmisión Bluetooth.
- Ajuste final del control para conseguir un control robusto en vuelo manual.

Tarea	Enero				Febrero				Marzo				Abril				Mayo			
	Sm 1	Sm 2	Sm 3	Sm 4	Sm 1	Sm 2	Sm 3	Sm 4	Sm 1	Sm 2	Sm 3	Sm 4	Sm 1	Sm 2	Sm 3	Sm 4	Sm 1	Sm 2	Sm 3	Sm 4
Anexo B																				
Instalación y configuración de OpenPilot																				
Modelado y posibles ensayos																				
Estudio de distintos controles																				
Diseño de entorno de simulación																				
Diseño de la máquina de estados																				
Diseño del control																				
Diseño del Mixeador para el Tricóptero																				
Ajuste control para simulación																				
Ensayos en Tierra y ajuste de Parámetros																				
Ensayo en vuelo manual																				
Ajustes finales																				
Redacción de la memoria																				

Figura 1.1. Cronograma del Proyecto.

Para el desarrollo de este proyecto serán necesarios los siguientes elementos:

- Tricóptero de Hobby King, con motores *brushless*, servo y ESC.
- Emisora de Radio Control y receptor de Turnigy.
- Tarjeta de control OpenPilot Revolution.
- Placa distribuidora de potencia con regulador de tensión y sensor para medición de intensidad y tensión.
- Ordenador del laboratorio de proyectos con el programa Matlab 2015b y Simulink, en el que instalar la herramienta para OpenPilot
- Batería LiPo de tres celdas para el tricóptero y LiFe para la emisora y un cargador-balanceador.

- Conectores USB, MicroUSB, programador de ST Link y cables.

Capítulo 2

Descripción de tricóptero

2.1 Elementos del Tricóptero

Se expondrán a continuación los diferentes elementos presentes en un tricóptero de pequeño tamaño como el que se utiliza en este proyecto. La mayoría de los componentes han sido obtenidos en una tienda especializada para montar directamente el aparato. El resto han sido buscados de tal manera que encajara con lo ya tenido.

2.1.1 Estructura

La estructura del tricóptero es el armazón sobre el que se colocan todos los demás elementos. Debe ser lo suficientemente rígida y consistente como para no deformarse durante el vuelo. La forma básica de ésta ha de ser en forma de "Y" con igual longitud entre motores. Desde la junta donde salen los brazos hacia los motores delanteros hasta una distancia mínima de seguridad con el motor trasero, debe tener espacio suficiente para colocar la tarjeta de control y todos los demás periféricos necesarios para la utilización del control durante el vuelo. Puesto que este proyecto no se centra en la construcción y modelado de éste, se decidió por comprar la estructura. Se eligió una bastante básica que tenía como ventaja la adaptación para el "servo" del motor trasero ya preparada.

Figura 2.1 Estructura del Tricóptero

Figura 2.2 Adaptación del Servo

2.1.2 Motores

Dado que la alimentación se hará con una batería que proporciona corriente continua, es necesario utilizar motores DC para hacer que las hélices giren. Los motores de continua utilizan escobillas a la hora de realizar una conmutación mecánica de la corriente y generar el par mediante unos imanes permanentes. Su funcionamiento se basa en mantener fijo el campo magnético del rotor.

Como principal inconveniente a estos motores se encuentra que su coste de mantenimiento es alto ya que las escobillas se deterioran con facilidad debido al roce continuo con las delgas. Por ello encontramos como alternativa los motores *brushless* que no tienen escobillas. En realidad, estos motores son trifásicos síncronos, en su funcionamiento se encuentran bobinas e imanes permanentes. En el estator las bobinas se alimentan con corriente alterna trifásica y se genera un campo magnético giratorio. A su vez, el campo en el rotor generado por los imanes permanentes tiende a alinearse con el del estator, produciendo el giro de éste.

Los motores *brushless* presentan la gran ventaja de que en ausencia de las escobillas se consiguen evitar las pérdidas por rozamiento. Por otro lado, esto facilita el mantenimiento y se consigue alargar su vida útil. El inconveniente es que tenemos que pasar de una alimentación de una batería en corriente continua a una trifásica. Para modificar su velocidad de giro será necesario variar la frecuencia de la corriente alterna con la que se alimenta el estator, tarea llevada a cabo con los ESCs

Especificaciones

- KV (RPM / V): **2150**
- Células Lipo: **2 ~ 3s**
- Potencia máxima: **120w**
- Amps max: **10A**
- Dimensiones (Dia.xL): **27 x 17 mm**
- Eje del motor: **3,17 mm (eje de la hélice de 5 mm)**
- Eje de la hélice: **5mm**
- Peso: **30 g**

Figura 2.3 Multistar 2150KV Motor The "Baby Beast"

2.1.3 ServoMotor

Pieza fundamental en el dron tricóptero de este proyecto. Es así porque gracias a éste se permite tener el cuarto grado de libertad necesario para poder controlar un cuerpo tridimensional situado en el espacio. Su función es principalmente girar un ángulo α uno de los tres motores descritos en el apartado anterior. Su alimentación es puramente de continua y se usará el regulador de uno de los ESCs, la señal que hará que gire unos grados determinados provendrá directamente de la tarjeta de control.

Especificaciones

- Modelo: **TGY-9018MG**
- Voltaje de funcionamiento: **4.8V / 6.0V**
- Corriente de funcionamiento: **200mA / 250mA**
- Velocidad de funcionamiento: **0.12sec.60º / 0.10sec.60º**
- Puesto par: **2.3kg.cm /2.5kg.cm**
- Tamaño: **23X12.1X25.8mm**

- Peso: **13 g**

Figura 2.4 Turnigy TGY-MG 9018MG Servo

2.1.4 Controlador electrónico de velocidad

Cuando se habla de Controlador electrónico de velocidad o ESC (*Electronic Speed Control),* se suele referir a un pequeño circuito inversor que genera corriente alterna trifásica a partir de una tensión continua y una señal de entrada para controlar su frecuencia. La alimentación de los ESC va directamente desde la batería y la señal de control que reciben estos es un PWM con un t_{on} que va entre 1ms y 2ms, con el que se ajusta la frecuencia de la alimentación de los motores.

La ventaja de estos controladores es que permiten trabajar de manera sencilla con estos motores, de tal manera que el esquema equivalente visto desde la batería es el de uno de corriente continua normal en el que la señal de control regula la tensión de alimentación. Siguiendo este razonamiento, si esta señal es de 1000µs los motores estarán parados, mientras que si es de 2000µs estos girarán a velocidad máxima.

Puesto que nuestra estructura era la básica, ha sido necesario obtener aparte los 3 ESC que se han utilizado, se han colocado sobre ésta de manera que no afecten a la estabilidad del tricóptero.

Especificaciones

- Consumo de corriente: **12A continuo**
- Rango de voltaje: **2-4s LiPo**
- BEC: **5.0V 0.5A Linear**
- Frecuencia de entrada: **1 KHz**
- Peso: **10 g**

Figura 2.5 Afro ESC 12Amp BEC UltraLite

2.1.5 Baterías

Es necesario utilizar una batería para alimentar tanto los motores como la tarjeta de control que se ha empleado, por supuesto además de todos los circuitos ya conectados soportados en la estructura. El descarte del uso de una fuente externa es claro debido a los inconvenientes que se presentarían por temas de cables. La tarjeta *OpenPilot* necesita de una alimentación de 5 V, aunque gracias al regulador de tensión incorporado en la placa de distribución que se usará, es posible alimentar la tarjeta sin miedo de sobrecargarla. Por otro lado, para el uso y manejo de la emisora también será necesaria otra batería para la obtención así de mayor independencia.

En el mercado se pueden encontrar varios tipos de baterías que pueden resultar útiles para proyectos como el que se lleva a cabo. Entre las más comunes encontramos las que son de níquel o hidruro metálico (NiMH), de ion litio (Li-ion), de litio y ferro fosfato (LiFe), de níquel y cadmio (NiCd) y de polímero de litio (LiPo).

Todas estas están formadas por distintas celdas en serie por lo que existen múltiples combinaciones. Cada celda tiene una tensión máxima y una nominal debido a que ésta no se mantiene constante durante el proceso de descarga. Se ha de tener especial cuidado y precaución pues algunas de ellas tienen una tensión mínima por debajo de la cual ésta queda inservible e incluso puede arder o explotar.

A continuación, se analizan en más detalle las dos principales opciones a tener en cuenta para este proyecto, que serían las LiPo y las LiFe puesto que las Li-ion quedan descartadas al ser las predecesoras de las LiPo y las otras dos tienen el inconveniente de proporcionar una tensión demasiado baja para lo que necesitan los motores.

Baterías LiPo

Las baterías de polímero de litio, en comparación con las demás baterías del mercado, tienen una gran densidad energética. Esto significa que la relación entre su masa y la energía que pueden almacenar es muy elevada, muy poca masa proporciona gran potencia energética [1]. Por otro lado, también tienen una alta capacidad de descarga lo cual resulta muy beneficioso puesto que la potencia de los motores para levantar y permitir el vuelo del tricóptero es elevada. Su ciclo de vida es largo permitiendo así cientos de cargas y descargas. Son las más recomendadas para aplicaciones en proyectos de drones.

Figura 2.6 Batería LiPo de tres celdas

No se puede olvidar que también tienen ciertos inconvenientes, suelen presentar fallos cuando trabajan sometidas a temperaturas altas o bajas, motivo por el cual para el vuelo en interiores del proyecto por sus condiciones de vuelo no supone ningún problema. Tienen una tensión mínima bajo la cual ésta queda inservible, aunque no llegue arder o a explotar. Hay que tener cuidado en recargarlas en modo especial si se pretende dejarlas almacenadas durante periodos largos.

La tensión nominal de cada celda es de 3.7V y su máxima de 4.2V, para este proyecto se usará una batería de tres celdas. No es preocupante el aumento de peso de una de dos celdas frente a una de tres debido a la potencia de los motores que se utilizan.

Baterías LiFe

Las baterías de litio y ferro fosfato son muy parecidas a las LiPo, aunque tienen ciertas diferencias bastante importantes. La principal ventaja frente a las otras es que no arden si se descargan en exceso y tienen una duración más larga. Por otro lado, su densidad energética es mejor por lo que son más pesadas que las LiPo. Su tensión nominal es de sólo 3.3V llegando a un máximo de 3.6V por lo que serían necesarias sí o sí tres celdas para el tricóptero.

Por estos motivos lo aconsejable es el uso de las baterías LiPo para el dron ya que lo que se obtiene mediante la LiFe en autonomía se termina perdiendo por peso al demandar más potencia a los motores. Eso sí, en cuanto a la emisora, sí es más conveniente usar una LiFe debido a que el peso no es tan importante y por autonomía proporciona mayor comodidad.

Cargador de baterías

Para cargar correctamente las baterías será necesario utilizar un cargador específico para cada una. Por eso lo más cómodo es usar uno con distintos modos que sirva para cargar cualquiera que se tenga, independientemente del tipo. Existen cargadores que además presentan el modo *Balance*, que sirve para estabilizar, para regular la tensión de cada celda de manera que todas estén cargadas por igual y evitar así problemas si alguna se sobrecarga o se queda por debajo del límite.
Para este proyecto se empleará un cargador IMAX que cumple con las especificaciones dichas anteriormente.

Figura 2.7 Cargador-balanceador de baterías

2.1.6 Placa de Distribución y Sensor

Para poder transmitir la suficiente potencia desde la batería a los ESCs y a la tarjeta de control es necesario un *Power Module*. El que se ha elegido para este proyecto lleva incorporado un regulador de tensión, el cual es necesario para evitar sobrecalentar la placa, y también un sensor de tensión e intensidad. En la Figura (2.8) se puede apreciar un esclarecedor esquema de sus conexiones.

Especificaciones

- Tensión de entrada máxima **18V**

- Corriente máxima **90ª**
- Salida del Regulador **5.3V a 2.25A máx** (medidas por defecto 5V ADC)

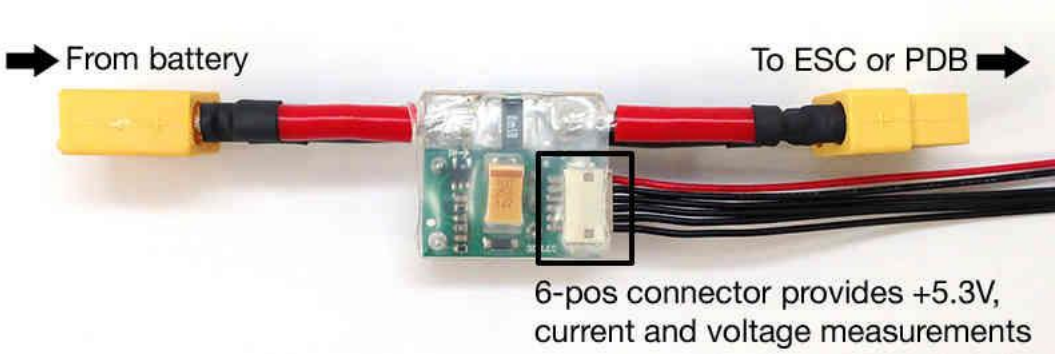


Figura 2.8 HKPilot Power Module

2.1.7 Tarjeta de Control

El objetivo del proyecto es diseñar un control para que el tricóptero vuele. Dicho control se debe descargar en una tarjeta de control para que se ejecute desde ahí de manera externa. El periodo de muestreo debe ser lo suficientemente pequeño para lograr la estabilidad necesaria y eso repercute directamente en la potencia de la tarjeta. En años anteriores se usó ArduPilot Mega que usaba la simplicidad del manejo de Arduino con una estructura especialmente adaptada y diseñada para UAVs.

Para este proyecto se usará OpenPilot Revolution [2] cuya principal ventaja es el precio (1/4 frente al ArduPilot Mega). A diferencia de la usada en otros años, no hay entorno de trabajo ya preparado en Matlab Simulink por lo que el proyecto consistirá también en desarrollar lo necesario para poder volcar todo en nuestra tarjeta de control.

La placa consta de una estructura pensada preparada para unas determinadas conexiones, consta por defecto de las entradas y salidas necesarias para conectar los ESC, además de puertos prediseñados (*Main Port, Flexi Port*) para comunicación I2C o SPI. Encontramos el llamado *Power Sensor* que es por el cual alimentaremos la placa desde el regulador de la tarjeta de distribución y finalmente 8 pines en el *Flexi-I/O Port* para aquellas que veamos necesarias. Por supuesto incorpora asimismo una IMU con giróscopo y acelerómetro para monitorizar el estado del tricóptero y conexión microUSB para la alimentación de la tarjeta en modo descarga. La descarga y programación se hará mediante el programador ST-Link con las conexiones adecuadas

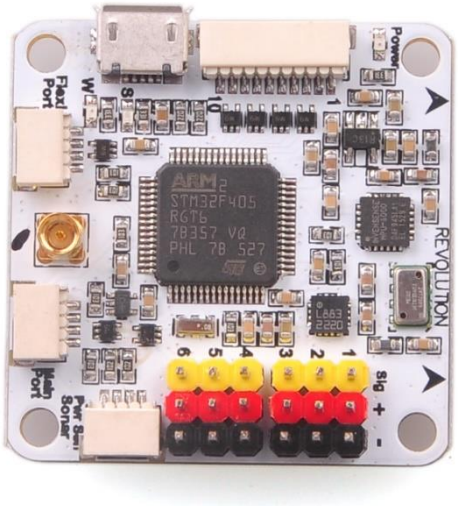


Figura 2.9 Tarjeta de control OpenPilot Revolution

2.1.8 Unidad de medición inercial (IMU)

La IMU es la unidad de medición inercial (*Intertial Measurement Unit)* y consta de varios elementos microelectromecánicos de medición para conocer en todo momento el estado, en este estado, del tricóptero. En este proyecto se utilizarán las medidas de la IMU para conocer la orientación del cuerpo. La IMU de la tarjeta OpenPilot Revolution consta sólo de giróscopos y acelerómetros, principal diferencia con la del HKPilot Mega, que tiene estos y además incorpora un magnetómetro para conocer la posición.

Giróscopo

Un giróscopo es un instrumento capaz de medir las velocidades angulares de un cuerpo. Debido al desarrollo de los sistemas microelectromecánicos (MEMS), es posible integrarlos dentro de pequeños chips. Su funcionamiento se basa en el efecto Coriolis. Al someterse a una rotación, gracias a unas pequeñas galgas que generan una tensión proporcional a la elongación, el giróscopo mide la deformación de una masa inercial interna [2]. Es posible conocer la velocidad angular del giro en función de esta tensión. La IMU consta de tres giróscopos, uno para cada eje, de modo que la medida resultante es la velocidad angular total expresada en el sistema solidario, los ejes del cuerpo

Acelerómetros

Los acelerómetros tienen un funcionamiento basado en sensores piezoeléctricos. Son capaces de medir las aceleraciones en un eje, siendo a su vez insensibles a las aceleraciones

ortogonales a éste. Su principio de funcionamiento se basa en las elongaciones que se generan cuando una masa inercial es sometida a una fuerza [3]. Estas elongaciones se traducen en unas diferencias de potencial que permiten calcular la aceleración. Al igual que ocurre con los giróscopos, la IMU cuenta con 3 acelerómetros orientados en los 3 ejes para medir la aceleración total en el sistema del cuerpo.

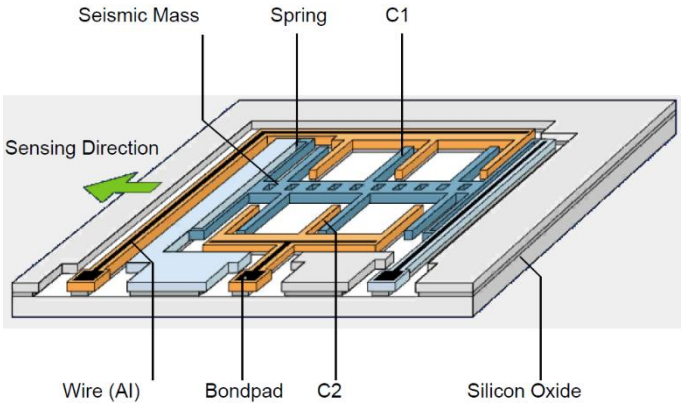


Figura 2.10 Acelerómetro

2.1.9 Sistemas de comunicaciones

El diseño del control, así como el programa que permitirá el vuelo en interiores del tricóptero, se realiza en el ordenador con el programa Matlab y su extensión de Simulink. Sin embargo, es necesario descargar el código generado en la tarjeta de control que va conectada al sistema de emisora, al de transmisión por Bluetooth y a los motores. A su vez, para dar las órdenes de vuelo, tanto en la fase de vuelo manual como para indicar parada de emergencia, hace falta enviar unas referencias desde la emisora.

Para la exitosa realización de estas tareas, es necesario utilizar sistemas de comunicaciones. Primero entre el ordenador y el micro, que se hará a través del puerto USB para alimentarlo y del programador para cargarle el programa. A partir de sistema de transmisión por Bluetooth enviaremos y recibiremos datos en pleno vuelo. Como el micro no lo soporta directamente será necesario añadir otro receptor.
Se utilizará el protocolo maestro-esclavo, para programar los módulos Bluetooth HC-05 (modelo escogido para este proyecto) se ha utilizado un arduino [4]. Han sido configurados a transmitir a una velocidad de 57600 baudios.

En cuanto a la comunicación con la emisora, desde donde se enviarán las referencias del control, el sistema debe ser necesariamente inalámbrico. Debido a que el sistema más extendido y adaptado al vuelo de drones es por radiocontrol (RC) se usará este. Las emisoras RC para aeronaves tienen cuatro canales analógicos para dar las referencias de alabeo, cabeceo, guiñada y empuje. La que se usará en este proyecto tiene más canales, tanto digitales

como analógicos que permiten combinar las señales de varios de ellos. Se usará la 9XR de Turnigy por tener la facilidad de emitir hasta en 9 canales y por ya tener experiencia previa en proyectos de otros años. A las emisoras hay que añadirles un transmisor que se coloca en la parte posterior y además enlazarlo con un receptor, que va colocado en la estructura del tricóptero y conectado por pines a la tarjeta de control

PPM

PPM o también conocido como *Pulse-Position Modulation,* es uno de los métodos de codificación más usados en radiocontrol. Actualmente ya se empieza a reemplazar por PCM por su mayor comodidad, pero no todos los receptores están preparados para esta codificación y al ser eficiente y bastante confiable, es el que hemos intentado usar en este proyecto.

El estándar utilizado en R/C es que el ancho de pulso varíe entre 1 ms y 2 ms, siendo el primero el mínimo y el segundo el máximo. Este pulso ha de repetirse en un periodo de 18 ms para el receptor utilizado.

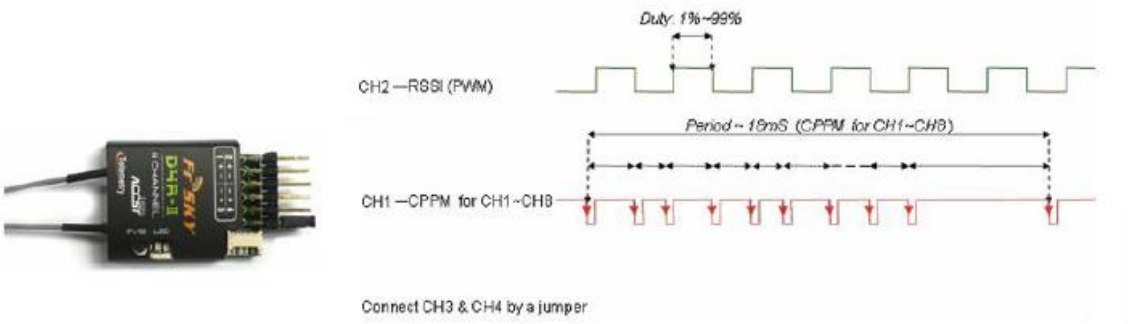


Figura 2.11 Señal de Codificación por Posición y ancho de Pulso para 8 canales

Como se puede apreciar en la imagen, mediante el PPM se era capaz de recibir los 8 canales de la emisora todos recogidos en un período de 18 ms [5]. Esto planteaba un problema si el ancho de pulso era máximo en todos los canales pues no había forma de diferenciar cuando comenzaba o finalizaba la trama pues se obtenían 9 señales de 2 ms. Para ello se ideó en Simulink un *Demodulador* que permitía justo eso, obtener de manera fiable las 8 señales que se recibían de la emisora. El conflicto por el cual se tuvo que desechar el uso del PPM en el proyecto fue debido a que el Demodulador usaba un *timer* de la placa de control que entraba en conflicto con el de la IMU y con el de generación de la señal PWM para los motores, haciendo que los canales no se recibieran correctamente y hubiera una des-sincronización dentro del propio *Demodulador*, haciendo imposible el uso de éstas.

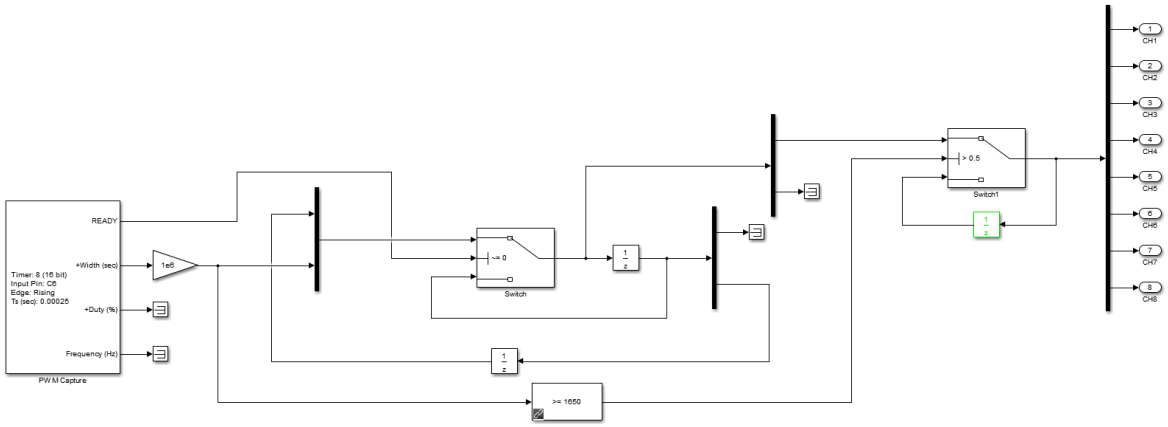


Figura 2.12 Demodulador para PPM en 8 canales

Lectura por Canales

Debido a los problemas explicados en el apartado anterior con el módulo del PPM se ha tenido que hacer la lectura directa de los canales de la emisora. Como efecto negativo, el receptor que se ha utilizado es de cuatro canales lo cual ha obligado a rehacer la máquina de estados, de la cual se hablará en el [5.2]. Además, con esta medida se perdió la posibilidad de modificar parámetros del control en vuelo mediante los canales 6-8. Como alternativa para no perder tan clara ventaja se ha implementado en el fichero Serial RW PC unos diales que transmitirán por Bluetooth estas variaciones.

2.2 Instalación y conexiones de la tarjeta OpenPilot Revolution

Este probablemente sea uno de los puntos más importantes del proyecto. Al estar trabajando con una nueva placa, OpenPilot Revolution, ha sido necesario partir desde cero a la hora de revisar y construir todos los diagramas en Matlab, dentro del entorno de Simulink. Para ello se ha trabajado con la librería Waijung, que contenía los bloques de configuración básicos de nuestra tarjeta.

Para saber qué había que conectar y en qué puertos se ha utilizado el *Revolution Schematic* [6]. Las conexiones que se han llevado a cabo han sido las siguientes:

2.2.1 IMU

Siguiendo la información del *Schematic* [7] se buscó qué pines de la tarjeta de control correspondían a una conexión de tipo SPI y se decidió conectar al A4 el *NSS* al A5 el *SCK* al A6 el *MISO* y al A7 el *MOSI* con un *Baud Rate* de 1 MHz

2.2.2 UART

Con el mismo procedimiento se buscaron los pines correspondientes vigilando que se tuviese acceso físico a ellos, debido a que la transmisión de la UART sería por Bluetooth. Por esto se ha tenido que hacer un cable que llevara los pines del módulo emisor/receptor a los elegidos en la tarjeta de control. El Pin A9 para la transmisión (*TX*) y el A10 para la recepción (*RX*). El Baud Rate [7] es de 57600 bps debido a la configuración maestro-esclavo de la transmisión Bluetooth configurada.

La transmisión Bluetooth, así como sus conexiones a los pines se harán mediante el *Main Port* de la tarjeta de control.

Figura 2.13 Módulo BlueTooth HC-05

2.2.3 ESC/Servo

En las conexiones de los ESC y el servo se ha tenido el mismo problema por el que se tuvo que abandonar el uso del PPM, los *timers*. A medida que se avanzaba con el proyecto se descubría una de las principales limitaciones que más tiempo ha llevado solventar, el uso de mismos *timers* para lectura o escritura de pulsos hacía que estos se desconfiguraran y quedaran inservibles. Debido a eso para los ESC 1 y 2 se decidió usar el Timer 9, cuyos pines son A2 y A3 (situados en los puertos pensados para ESC/Servos de la placa OpenPilot) mientras que para el ESC 3 y el Servo se utilizó finalmente el Timer 12, con pines B14 y B15 (correspondientes al *Flexi I/O port* de 8 pines). A efectos prácticos esto no supone ningún problema porque la alimentación de los ESC se hace mediante la placa distribuidora y la de la placa de control mediante el *Power Sensor*, por otro lado, el Servo será alimentado mediante tensión y tierra de uno de los ESC puesto que se descubrió que la tarjeta no proporcionaba intensidad suficiente para ser capaz de moverlo.

2.2.4 Power Sensor

Esta es la conexión que se encarga de alimentar el OpenPilot Revolution, desde el regulador de tensión del sensor de la placa de distribución vienen cuatro cables, dos de ellos son tierra y tensión por los que va la alimentación y los otros son las medidas analógicas de la tensión y corriente que se utilizarán para la monitorización en vuelo de la batería.

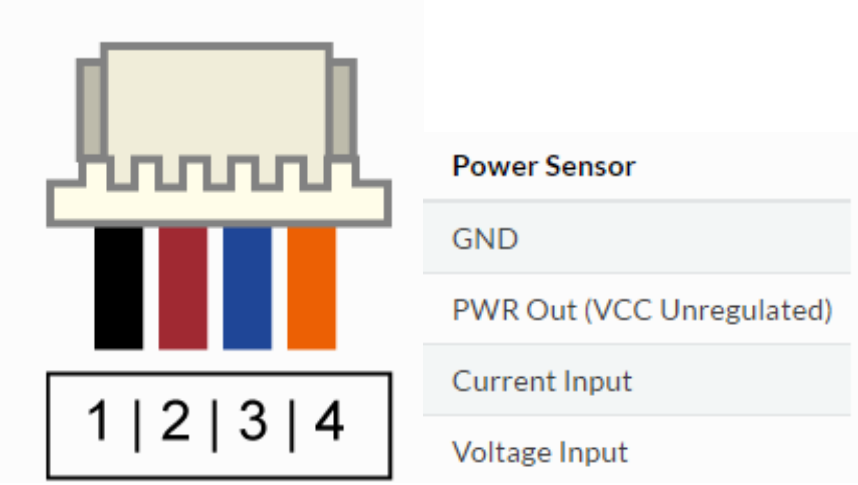


Figura 2.14 Conexión del Power Sensor

2.2.5 Canales de la Emisora

Aquí se tomó la solución al uso del PPM, de nuevo se encontró una nueva limitación que llevó gran parte del tiempo en solventar. Para la recepción de pulsos *(PWM Capture)* no se podía usar pines con el mismo *Timer*, ni entre ellos ni con los generadores de pulsos (*PWM Basic).* Se ha tenido que ir buscando por un lado pines accesibles, que se pudieran conectar al receptor de RC, y por el otro que no coincidiera su timer. Finalmente se encontró una combinación para poder llevar esto a cabo para los 4 canales de lectura de la emisora. El Ch1 en el pin C7 (Timer 3) y el Ch2 en el pin C6 (Timer 8), ambos del *Flexi I/O Port.* Y los otros dos, el Ch3 en el pin A0 (Timer 2) y el Ch4 en el pin A1 (Timer 5), correspondientes de la parte originalmente pensada para la señal de ESC/Servo.

2.3 Configuración de la emisora

La emisora a utilizar en este proyecto, como ya se ha dicho anteriormente, es la 9XR de Turnigy, junto con el receptor FrSky D4R ofrece la posibilidad de combinar tanto mandos digitales como analógicos con ponderaciones a configurar en 8 canales. Debido a que no se

usará el módulo PPM del receptor se pasará de los 8 canales previstos a los 4 analógicos. En la figura 2.15 se muestra en la imagen, sacada del manual de usuario [8], los diferentes mandos indicados.

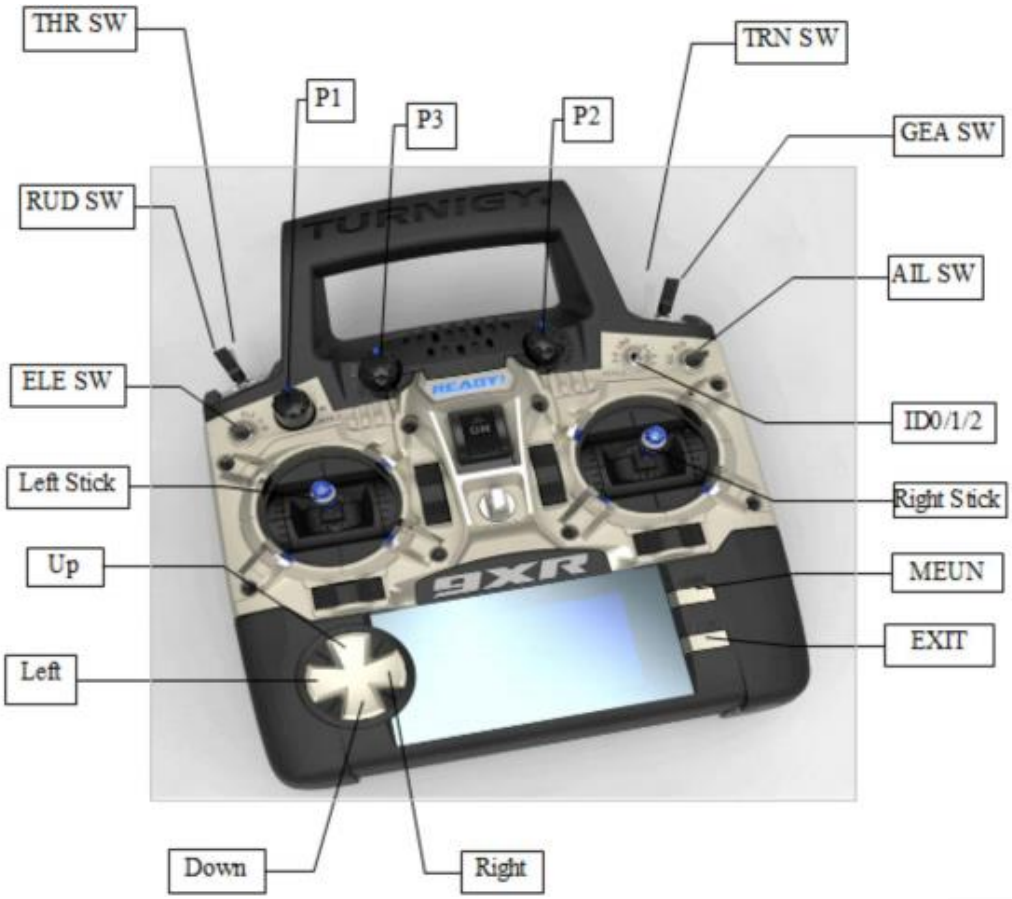


Figura 2.15 Mandos de la emisora.

Como se puede apreciar, se puede apreciar por un lado los *sticks* que varían su valor de manera analógica al desplazarse linealmente. Los *Pots* que son equivalente a los *sticks* pero con forma circular y varían al girarse. Y por último los *switches* que tienen dos o tres posiciones dependiendo de estos y que funcionan como señales digitales. Los dos *sticks* se han usado para transmitir las cuatro señales principales para el vuelo manual, que se envían en su totalidad por los 4 primeros canales y son las referencias de alabeo, cabeceo, empuje y guiñada.

Por motivos de precaución se ha definido el nivel de alarma de la batería en 9V de manera que, cuando la batería LiFe tenga una tensión inferior a ésta, la emisora avise con una señal acústica. También, para ahorrar batería, se ha activado la alarma si está más de 5 minutos encendida sin utilizarse. Por otro lado, por motivos de seguridad, si al encenderse no están todos los *switches* en posición original y la señal de empuje (*Throttle)* no está a cero, no comenzará a emitir.

Respecto a la relación emisión-recepción cabe comentar que no basta con que sean

ambos dispositivos compatibles, sino que es necesario enlazarlos. La transmisión por RC (radiofrecuencia) se realiza en la banda de los 2.4 GHz y debido al protocolo que siguen, para que ambos dispositivos estén sintonizados en la misma frecuencia es necesario hacer el enlace. Una vez realizado con éxito el piloto del receptor conectado a nuestra tarjeta de control dejará de parpadear intermitentemente y se mantendrá fijo, ahora la emisora ya está preparada para enviar las señales adecuadas al receptor.

Capítulo 3

Modelado

3.1 Modelado de la planta

En los últimos años se ha podido simplificar la compleja mecánica asociada a costa de aumentar la precisión y la eficacia de su control. En esta línea, los cuadricópteros, modelo del que partiremos para poder establecer nuestro tricóptero, han destacado por su especial sencillez: constan tan solo de cuatro motores que mueven una hélice cada uno. Nuestro multicóptero tiene tres hélices que se disponen en forma de aspa y se controlan con facilidad mediante un PWM mientras que el servo que controla el ángulo α también se hará con otro módulo de ancho variable. Al tener cuatro actuadores que funcionan de manera independiente, conseguimos cuatro grados de libertad.
La estructura y las ecuaciones que rigen los movimientos de estos son muy parecidas independientemente del modelo concreto. A continuación, se presentan algunos modelos matemáticos relevantes y los elementos principales de estudio de estas ecuaciones para este proyecto.

3.1.1 Ejes y matriz de cambio de base de Euler

El modelado del tricóptero UAV consiste de dos marcos de referencias. El marco de referencia del cuerpo *{C},* que es solidario al tricóptero, representado por las variables *{x, y, z}* mientras que el de referencia de la tierra *{T}*, sistema de ejes fijos también llamado inercial, lo está por *{X, Y, Z}*. Los cuatro grados de libertad permiten mover el tricóptero en sus tres ejes y girar sobre su eje *z*, que es el de guiñada o *yaw* en inglés. Los otros dos restantes corresponden a los giros en torno al eje *x* (alabeo, *roll* en inglés) y entorno al eje *y* (cabeceo, *pitch* en inglés).

Hay que tener en consideración que este sistema de ejes conlleva una anomalía respecto a otros sistemas utilizados. Tal y como se puede observar en la Figura 3.1 el eje *Z* resulta estar orientado hacia abajo. Hecho que afectará al signo de los vectores de las ecuaciones mecánicas.

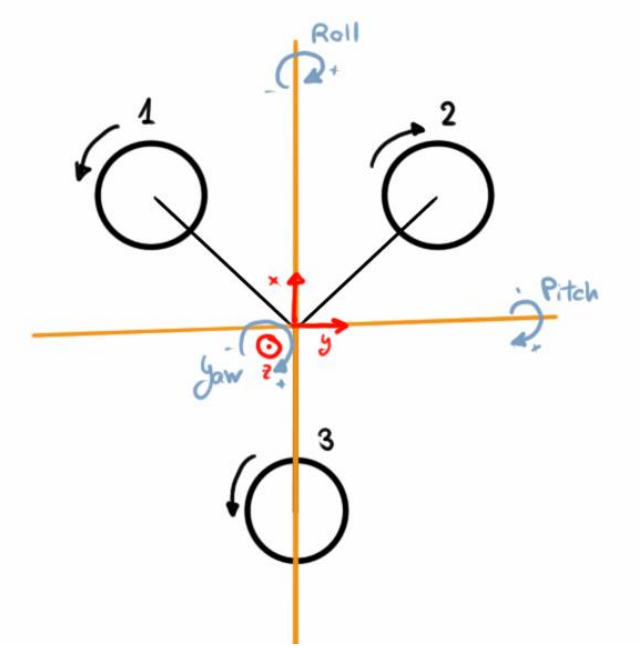


Figura 3.1 Sistema de ejes y movimientos del tricóptero

Lo más común a la hora de trabajar las ecuaciones mecánicas que rigen el comportamiento de un sólido rígido es utilizar las ecuaciones de Euler. Su mayor ventaja es la descripción de la orientación del cuerpo en función de los ángulos de alabeo, cabeceo y guiñada (ϕ, ϑ, ψ), también llamados de navegación o de Tait Bryan [9]. En su momento se propuso realizar el modelado del multicóptero mediante el uso de cuaterniones, que son elementos matemáticos cuya principal ventaja es que permiten definir completamente la orientación de un cuerpo y salvar situaciones en otrora críticas como un ángulo de 90º en alabeo o cabeceo. Se descartó debido a su complejidad y su composición anti-intuitiva.

Para expresar las coordenadas de un vector en un sistema fijo o inercial *{e11, e21, e31}* en la base solidaria al cuerpo *{e1B, e2B, e3B}* se han de utilizar matrices de giro. Los giros se van componiendo uno sobre otro así hasta sucederse los tres posibles. También se realizan en orden consecuente, primero el de alabeo sobre el eje *X*. A continuación, se hará el giro de cabeceo sobre el eje *Y1*, pues no es el eje *Y* original sino el resultante del primero giro de alabeo. Por último, se aplica el giro de guiñada sobre el eje *Z2*, que es el resultante de girar el eje *Z* primero el ángulo de alabeo y a continuación el de cabeceo. Se pueden observar los giros a continuación en las siguientes ecuaciones.

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix} \tag{3.1}$$

$$\begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix} = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} \cdot \begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix} \tag{3.2}$$

$$\begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix} = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_3 \\ Y_3 \\ Z_3 \end{bmatrix} = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \quad (3.3)$$

Como resultado de la composición de los tres giros sucesivos aparece una matriz de cambio de base [Ecuación 3.4], que transforma directamente un vector de la base solidaria a la base fija. Esta matriz recibe el nombre de Matriz de Rotación y se expresa en función de los ángulos de Euler de la siguiente manera:

$$\boldsymbol{R} = \begin{bmatrix} cos\theta \cdot cos\psi & sin\phi \cdot sin\theta \cdot cos\psi - cos\phi \cdot sin\psi & cos\phi \cdot sin\theta \cdot cos\psi + sin\phi \cdot sin\psi \\ cos\theta \cdot sin\psi & sin\phi \cdot sin\theta \cdot sin\psi + cos\phi \cdot cos\psi & cos\phi \cdot sin\theta \cdot sin\psi - sin\theta \cdot cos\psi \\ -sin\theta & sin\phi \cdot cos\theta & cos\phi \cdot cos\theta \end{bmatrix} \quad (3.4)$$

Como toda matriz de giro, la Matriz de Rotación tiene una serie de propiedades que la hacen especial. Su principal característica es que es ortogonal, lo cual significa que el producto escalar de sus vectores fila o columna con ellos mismos son 1 y con los demás son 0. Gracias a esta propiedad, la matriz inversa coincide con la traspuesta de manera que, para realizar el cambio de base, de una manera recíproca, es decir, del sistema fijo al solidario, sólo bastaría con usar la Matriz de Rotación traspuesta $\boldsymbol{R}^t$.

3.1.2 Fuerzas y momentos que aparecen

Para el cálculo de los momentos y de las fuerzas se seguirá el proceso descrito en [10], por ser uno de los más rigurosos y cuya experiencia con él, en proyectos del año pasado, fue satisfactoria. En el tricóptero, para el modelado de éste, harán falta hacer algunas modificaciones.

El primer paso es identificar todos los elementos con sus respectivos efectos que intervienen en la dinámica del tricóptero. Estos se recogen en la Tabla 3.1:

Efecto	*Fuente*	*Formulación*
Efectos aerodinámicos	– Rotación de las palas	$C\omega_m^2$
Pares de inercia	– Cambio en la velocidad angular de los rotores	$I_m\dot{\omega}_m$

	– Cambio en el ángulo de ataque del Servo	cos(α)
Efecto gravitatorio	– Posición del centro de masa	
Efectos giroscópicos	– Cambio en la orientación del cuerpo rígido	$I\dot{\theta}\dot{\psi}$
	– Cambio en la orientación del plano de fuerzas	$I_m\Omega_r\dot{\theta},\dot{\phi},$
Fricción	– Movimiento del cuadricóptero	$C\dot{\theta},\dot{\phi},\dot{\psi}$

Tabla 3.1 Efectos que intervienen en la dinámica de un tricóptero

Una vez que se tiene en cuenta que participa y cómo lo hace, es importante, en segundo lugar, hacer una serie de suposiciones de partida. aunque impida obtener resultados totalmente fiables, den una muy buena aproximación de lo que se busca:

- Estructura rígida.
- Estructura simétrica, por lo que el tensor de inercia se supone diagonal con *Ixx = Iyy*
- Centro de gravedad situado en el centro físico de la estructura y origen de coordenadas.
- Las palas rígidas de los motores 1 y 2 (los que no llevan el servo) no varían su ángulo de ataque al girar
- La fuerza vertical de empuje, *thrust*, y el momento de arrastre, *drag*, derivados del giro de los motores se supondrán proporcionales al cuadrado de su velocidad angular asociada.

Ahora, teniéndose en cuenta todo el conjunto, se detallarán uno por uno cada uno de los efectos expuestos en la Tabla 3.1.

Efectos Aerodinámicos

Las fuerzas de sustentación del tricóptero son las tres fuerzas verticales que ejercen los motores [11]. Estas fuerzas se suponen proporcionales al cuadrado de las velocidades de giro de estos, por lo que se definirá como factor de empuje *b* a la constante de proporcionalidad. De la misma manera ocurre con el factor de arrastre (*drag factor) d*, que será en este caso la constante de proporcionalidad entre los cuadrados de las velocidades angulares de los propulsores y el momento de resistencia aerodinámica que generan.

Las fuerzas se calculan por tanto mediante la Ecuación (3.5)

$$F_i = b \cdot \omega_i^2 \tag{3.5}$$

El sentido de giro de los propulsores es según se muestra en la siguiente figura, con dos de ellos girando en sentido anti horario (CCW), los motores 1 y 3, y el restante, el motor 2, girando en sentido horario (CW). Por la estructura en *"Y"* del tricóptero, habrá tres brazos a la hora de tener en cuenta los pares de las fuerzas, uno perteneciente al alabeo o *roll* y dos para el cabeceo o *pitch*. En la figura 2.4 se pueden observar tanto el sentido de giro de los motores como sus brazos correspondientes.

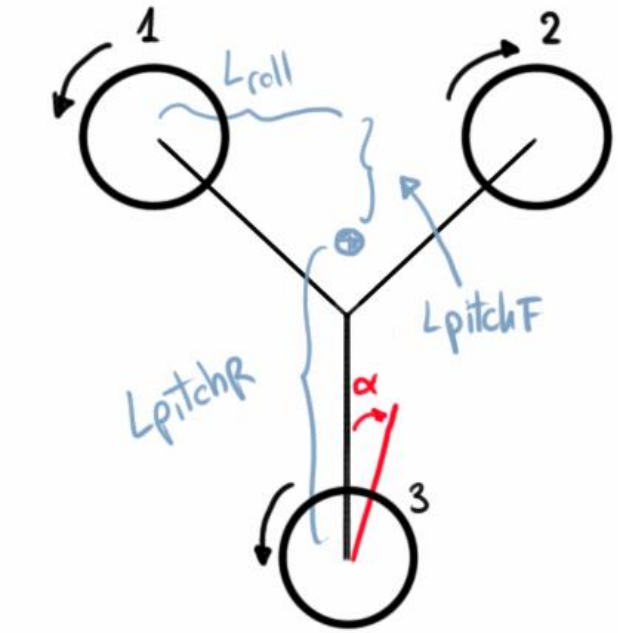


Figura 3.2 Representación de motores y ángulo de ataque del Servo

Se puede observar cómo se ha llamado al brazo de alabeo L_{roll}. Por otro lado, como se tienen dos brazos de cabeceo, se ha diferenciado entre el frontal L_{pitchF} y el trasero L_{pitchR}. No se puede despreciar, por efecto del servo, el brazo de altura L_{height} que intervendrá en el alabeo con la proyección horizontal del ángulo de ataque α del servo.

Además del propio peso del tricóptero, así como de estas fuerzas, aparecen otras horizontales debido al efecto aerodinámico del giro de las palas. Estas fuerzas se pueden considerar despreciables en comparación con el empuje vertical pero no podemos pasar por alto las proyecciones de la fuerza de empuje del motor 3 debido al servo. Ésta ya no es despreciable.

Se definen cuatro variables llamadas *mandos*, relacionadas con la fuerza de empuje para simplificar la expresión de los momentos y fuerzas:

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} b(\omega_1^2 + \omega_2^2 + \omega_3^2\cos(\alpha)) \\ b * L_{roll}(\omega_1^2 - \omega_2^2 + \omega_3^2\sin(\alpha) * \frac{\mathrm{L_{height}}}{\mathrm{L_{roll}}}) \\ b * L_{pitchF}(\omega_1^2 - \omega_2^2 - \omega_3^2\cos(\alpha) * \frac{\mathrm{L_{pitchR}}}{L_{pitchF}} - \frac{d}{b * L_{pitchF}} * \omega_3^2\sin(\alpha)) \\ d(\omega_1^2 - \omega_2^2 + \omega_3^2\cos(\alpha) - \frac{b * L_{pitchR}}{d} * \omega_3^2\sin(\alpha)) \end{bmatrix} \quad (3.6)$$

El primer mando *u1* es el empuje vertical total, *Throttle*. El segundo corresponde a los pares que aplican efecto de alabeo, *roll,* es decir, sobre el eje *x.* El tercero es aquel que se aplica sobre el eje *y,* aquellos pares que provocan efecto de cabeceo, *pitch*. Por último, el cuarto es el momento total generado por las palas en el eje *z*, que es el mismo que éstas sufren por el efecto aerodinámico, pero debido a la ley de acción y reacción será aplicado en sentido contrario.

La fuerza neta se define de la siguiente manera en la Ecuación (3.7):

$$\vec{F} = mg \cdot \vec{e}_{3I} - b(\omega_1^2 + \omega_2^2 + \omega_3^2\cos(\alpha)) \cdot \vec{e}_{3B} \quad (3.7)$$

Para expresar la fuerza neta anterior en la base fija se hace uso de la Matriz de Rotación obteniendo:

$$\begin{bmatrix} F_X \\ F_Y \\ F_Z \end{bmatrix} = \begin{bmatrix} -(sin\theta \cdot cos\phi \cdot cos\psi + sin\psi \cdot sin\phi) \cdot u_1 \\ -(sin\theta \cdot cos\phi \cdot sin\psi + cos\psi \cdot sin\phi) \cdot u_1 \\ mg - cos\theta \cdot cos\phi \cdot u_1 \end{bmatrix} \quad (3.8)$$

Los pares generados por el efecto aerodinámico de las palas quedan definidos como:

$$\begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} L_{roll} \cdot u_2 \\ L_{pitchF} \cdot u_3 \\ u_4 \end{bmatrix} \quad (3.9)$$

Estos están referidos a los ejes solidarios al cuerpo.

Una vez esto, es necesario ahora tener en cuenta los efectos giroscópicos. Se ven reflejados dos efectos, por un lado, debido a la aparición simultánea de velocidades angulares en dos ejes distintos, y por el otro debido a la aparición de una velocidad angular en un eje simultáneo al giro de los rotores. Este se manifiesta en forma de momento de fuerza, pero solamente en los ejes *x* e *y* ya que los ejes de giro de los motores son paralelos al eje *z* y por tanto no generan efecto giroscópico. Las ecuaciones que reflejan estos momentos son, respectivamente:

$$\begin{bmatrix} \tau'_x \\ \tau'_y \\ \tau'_z \end{bmatrix} = \begin{bmatrix} (I_{yy} - I_{zz})\dot{\theta}\dot{\psi} \\ (I_{zz} - I_{xx})\dot{\phi}\dot{\psi} \\ (I_{xx} - I_{yy})\dot{\phi}\dot{\theta} \end{bmatrix} \tag{3.10}$$

$$\begin{bmatrix} \tau''_x \\ \tau''_y \end{bmatrix} = \begin{bmatrix} -I_m\,\dot{\theta}(\omega_1 - \omega_2 + \omega_3\cos(\alpha)) \\ I_m\,\dot{\phi}(\omega_1 - \omega_2 + \omega_3\cos(\alpha)) \end{bmatrix} \tag{3.11}$$

Por último, se encuentran los pares de inercia al acelerar los motores, que se pueden despreciar despreciables si la dinámica de los motores es suficientemente rápida y afectarían únicamente en el eje *z*. Esto con el servo no es del todo cierto, pero ya se ha tenido el efecto de éstas a la hora de generar los mandos {*u1, u2, u3, u4}.*

3.1.3 Ecuaciones mecánicas del movimiento

En los diferentes estudios que se han realizado acerca de la aerodinámica y el cálculo de fuerzas y momentos en multicópteros se tienen en cuenta varios efectos [12]. Entre estos la variación del ángulo de las hélices cuando éstas se desplazan horizontalmente o el ángulo provocado por el ataque del viento. Sin embargo, al tratarse de un tricóptero para vuelo en interiores, se usarán las ecuaciones simplificadas a partir de las fuerzas y momentos descritas previamente en los apartados anteriores.

Se calculan las aceleraciones en los ejes fijos dividiendo entre la masa. Para ello partimos de las fuerzas sobre los tres ejes de la Ecuación (3.8)

$$\begin{bmatrix} \ddot{X} \\ \ddot{Y} \\ \ddot{Z} \end{bmatrix} = \begin{bmatrix} -(sin\theta \cdot cos\phi \cdot cos\psi + sin\psi \cdot sin\phi) \cdot \dfrac{u_1}{m} \\ -(sin\theta \cdot cos\phi \cdot sin\psi + cos\psi \cdot sin\phi) \cdot \dfrac{u_1}{m} \\ g - cos\theta \cdot cos\phi \cdot \dfrac{u_1}{m} \end{bmatrix} \tag{3.12}$$

Se deducen las expresiones de las aceleraciones angulares a partir de la expresión de los momentos netos de alabeo, cabeceo y guiñada. La cual se extrae de las ecuaciones de los momentos.

$$\begin{bmatrix} I_{xx}\ddot{\phi} \\ I_{yy}\ddot{\theta} \\ I_{zz}\ddot{\psi} \end{bmatrix} = \begin{bmatrix} l\,u_2 \\ l\,u_3 \\ u_4 \end{bmatrix} + \begin{bmatrix} (I_{yy}-I_{zz})\dot{\theta}\dot{\psi} \\ (I_{zz}-I_{xx})\dot{\phi}\dot{\psi} \\ (I_{xx}-I_{yy})\dot{\phi}\dot{\theta} \end{bmatrix} + I_m \begin{bmatrix} -\dot{\theta}(\omega_1-\omega_2+\omega_3-\omega_4) \\ \dot{\phi}(\omega_1-\omega_2+\omega_3-\omega_4) \\ 0 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ I_m(\dot{\omega}_1-\dot{\omega}_2+\dot{\omega}_3-\dot{\omega}_4) \end{bmatrix} \quad (3.13)$$

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{l}{I_{xx}}u_2 + I_1\dot{\theta}\dot{\psi} - \frac{I_m}{I_{xx}}\dot{\theta}(\omega_1-\omega_2+\omega_3-\omega_4) \\ \frac{l}{I_{yy}}u_3 + I_2\dot{\phi}\dot{\psi} + \frac{I_m}{I_{yy}}\dot{\phi}(\omega_1-\omega_2+\omega_3-\omega_4) \\ \frac{1}{I_{zz}}u_4 + I_3\dot{\phi}\dot{\theta} + \frac{I_m}{I_{zz}}(\dot{\omega}_1-\dot{\omega}_2+\dot{\omega}_3-\dot{\omega}_4) \end{bmatrix} \quad (3.14)$$

En esta última expresión se utilizan los momentos de inercia I_1, I_2 e I_3 para simplificar la expresión. Estos se definen como:

$$\begin{bmatrix} I_1 \\ I_2 \\ I_3 \end{bmatrix} = \begin{bmatrix} \frac{I_{yy}-I_{zz}}{I_{xx}} \\ \frac{I_{zz}-I_{xx}}{I_{yy}} \\ \frac{I_{xx}-I_{yy}}{I_{zz}} \end{bmatrix} \quad (3.15)$$

Por otro lado, si lo que se busca es la derivada de los ángulos de Euler, obtenida a partir de las velocidades angulares en los ejes propios del tricóptero, se usará la siguiente expresión, despreciando la aceleración de los motores.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & tan\theta\cdot sin\phi & tan\theta\cdot cos\phi \\ 0 & cos\phi & -sin\phi \\ 0 & \frac{sin\phi}{cos\theta} & \frac{cos\phi}{cos\theta} \end{bmatrix} \cdot \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \quad (3.16)$$

3.1.4 La Dinámica en los tres motores y el servo

Las señales de entrada al sistema son las velocidades angulares de cada motor y el ángulo de inclinación del servo α [12]. Estas señales son controladas con los tres ESCs y el servo mediante una señal digital PWM proveniente de la tarjeta de control. La salida de los motores son las velocidades de rotación y la salida del servo es el ángulo de inclinación. El motor y el servo han de ser modelados para averiguar cómo variará el modelo físico debido a la inclusión de estos elementos.

Motores

Como se explicó anteriormente, en la sección 2.1.2, los motores que se han empleado son

los llamados motores *brushless*. Estos aun a pesar de ser trifásicos, se alimentan con corriente continua. Debido a esto, visto desde un nivel superior, se puede suponer que tienen un esquema equivalente al de un motor DC. Por esto mismo, la dinámica que rige su comportamiento es también la misma que para un motor DC. Se tiene una K_e que relaciona la tensión en bornes del motor con su velocidad angular de giro y una constante K_t que relaciona el par con la corriente. Expresadas en las unidades adecuadas se pueden suponer iguales ambas constantes, hablando en este caso de una única constante K.

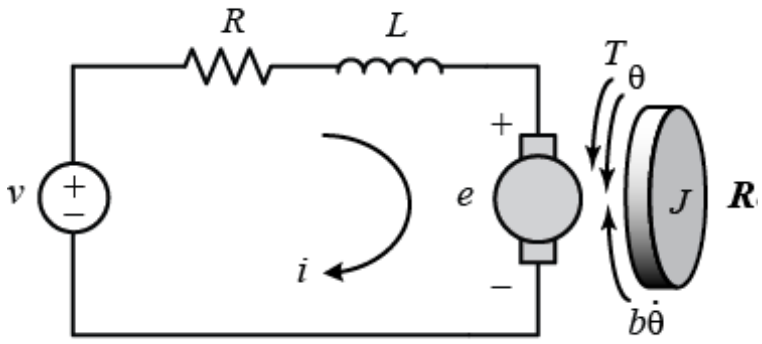


Figura 3.3 Esquema equivalente de un motor DC

Las ecuaciones diferenciales que rigen el funcionamiento de estos motores son:

$$v = R\,i + L\,\frac{di}{dt} + e = R\,i + L\,\frac{di}{dt} + K\,\omega \tag{3.17}$$

$$i = \frac{T}{K} = \frac{1}{K}\,(I_m\,\frac{d\omega}{dt} + T_R) \tag{3.18}$$

La ecuación diferencial que relaciona la tensión aplicada al motor *(v)* y la velocidad angular de salida (ω*)* es de segundo orden. Todo esto porque se trata de un sistema de ecuaciones diferenciales donde aparecen las derivadas de dos variables. Mediante la suposición de que la inductancia es suficientemente pequeña debido al tamaño de los motores simplificamos la ecuación de la siguiente manera:

$$v = \frac{R}{K}\,(I_m\,\frac{d\omega}{dt} + T_R) + K\,\omega \tag{3.19}$$

Servo

Es también un motor de corriente continua [13] donde el ancho de pulso que le entra se convierte en un ángulo de giro de hasta 180º (tiene un tope mecánico que impide girar más).

Con las ecuaciones diferenciales de las expresiones Ecuación (3.16) y Ecuación (3.17) se obtiene la función de transferencia entre la tensión de entrada y el ángulo de salida.

$$\alpha_{servo} = \frac{k_2}{s(1 + sk_1)} \tag{3.20}$$

Como se muestra en la Ecuación (3.19), hay dos polos. El servo controla el ángulo del motor y por lo tanto puede ser descrito mediante esta función de transferencia. La velocidad es controlada por el motor DC y no por el ángulo del motor del servo.

A efectos prácticos para controlar un servo hay que enviar una serie de pulsos. La caracterización corresponde a una recta por la que 1ms= 0 grados y 2ms= máx. grados (un poco menos de 180º). Entre estos valores da un ángulo de salida proporcional. Generalmente se considera que en 1.5ms se está en el "centro". El factor limitante es el tope del potenciómetro que tiene en su interior y los límites mecánicos construidos en el servo. Cuando se escucha el sonido de un zumbido normalmente indica que es está forzando el tope.

Para este proyecto se caracterizará el servo de la siguiente manera

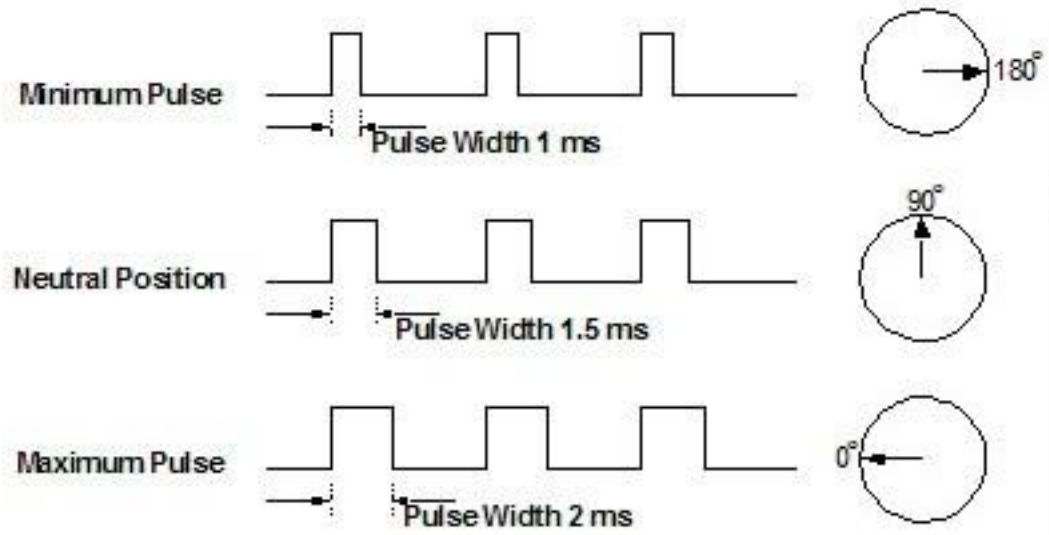


Figura 3.4 Relación ancho de pulso con ángulos correspondientes en un Servo

Capítulo 4

Diseño del sistema de control

4.1 Métodos de control

Para todo tipo de sistemas existen numerosas técnicas de control ya diseñadas. Van desde las más sencillas como los controles proporcionales ("P"), hasta el conocido y complejo "Control Predictivo". Se necesita el uso de estrategias suficientemente robustas para controles implantados en drones para compensar la alta inestabilidad de la planta con el contrapunto que el uso de controles demasiado complejos conlleva la necesidad de utilizar periodos de muestreo altos. Algo que puede provocar que la aeronave sea inestable. Se procederá, a continuación, a explicar brevemente los controles más comunes.

4.1.1 PID

Se trata de un tipo de control que actúa en el mando aplicando una respuesta Proporcional, Integral y Derivativo del error. La acción proporcional responde con mayor intensidad cuanto más grande es el error de seguimiento. Por otro lado, la parte integral es la que se encarga de ir acumulando el error para compensarlo totalmente, incluso compensa el error ante perturbaciones constantes, todo esto mediante una referencia constante. La desventaja de éste es que supone tener una respuesta más lenta. Por último, tenemos la acción derivativa, también llamada diferencial que se encarga de generar un mando proporcional a la derivada del error, lo que provoca también una tendencia más rápida. Lo negativo de esto es que amplifica en el mando, a su vez, el ruido de los sensores por lo que es conveniente acompañarlo de un filtro paso bajo.

Aunque no ha demostrado ser suficientemente robusto ante perturbaciones ha resultado funcionar de manera adecuada. Es el control que finalmente se ha terminado usando en el proyecto tanto a la hora de la simulación como de vuelo manual. En la simulación se ha usado un punto de operación para corresponder al vuelo estacionario [14]

4.1.2 LQR

El LQR (Linear-Quadratic Regulator) es un sistema de control basado en la minimización del coste. Teniendo en cuenta unas ponderaciones definidas por el usuario ajusta el punto de

operación para obtener la mejor respuesta. El ajuste se hace dada la matriz de realimentación de estados para minimizar las desviaciones. El que implementó este control fue *Hoffmann* [10], aunque primero lo hizo a velocidades bajas. Más adelante *Cowling* consiguió un método para optimizar estas ponderaciones y lograr así seguir trayectorias [15].

Sin embargo, quien implementó el sistema con estados independientes [10] fue *Bouabdallah*, de manera que la matriz se calculaba para el estado y no para un único punto de operación, como si se tratara de vuelo estacionario. Esto, al no restringir las trayectorias a vuelos prácticamente estacionarios, supuso mayor autonomía y robustez.

4.1.3 Control por linealización mediante realimentación

Tanto las ecuaciones que describen el comportamiento dinámico de un cuadricóptero como el de un tricóptero no son lineales sino que tienen términos donde unas variables dependen entre ellas, como las de estado. Una técnica para diseñar controles para este tipo de plantas es mediante la creación de variables intermedias que tengan una relación lineal con la salida. De esta forma se hace una conversión al mando intermedio desde el mando inicial y en dicha conversión se acumulan todos los términos no lineales basándose en el modelo previo. También es llamada *feedback linearization* [16].

Este control fue implementado con éxito utilizando como vector de variables de estado

$$X^T = [\dot{x}\ \dot{y}\ \dot{z}\ \phi\ \theta\ \psi\ \dot{\phi}\ \dot{\theta}\ \dot{\psi}] \tag{4.1}$$

Para poder así eliminar las relaciones no lineales entre los ángulos de Euler y las derivadas de las velocidades angulares. Simplificando, el control resultante era equivalente a un PD porque realimenta tanto las velocidades angulares como los propios ángulos. Este control fue el que se tomó como modelo el año pasado para la estabilización de los drones.

4.1.4 Otros posibles controles

Se han explorado, además de estos controles que son los más utilizados, otros más potentes y robustos como los basados en redes neuronales [17]. Métodos basados en lógica difusa (*fuzzy logic*) y el control borroso (*fuzzy controller*) [18]. No podemos olvidar mencionar el control predictivo [9] del cual estamos especialmente interesados en sus investigaciones

4.2 Estimadores de estado

Para el proceso del control es necesario hacer continuas mediciones de las variables que se desean controlar, en este proyecto serán de vital importancia los ángulos de Euler. Ahora,

para evitar los problemas de que, haciendo continuas mediciones de las variables de estados, éstas presenten ruido debido a los sensores, a las vibraciones de los motores o sencillamente a su naturaleza y lleven a que el control sea inestable, utilizamos los estimadores de estado.

Debe ser capaz de estimar el valor real de la medida a partir de una lectura de todas las salidas. Los tres estimadores más comunes para controles por realimentación de estado van en función de las variables que utilicen en su proceso de estimación. Cuando sólo se emplean algunas de las salidas y no todas ellas, se habla de un observador de orden reducido. Por otro lado, el llamado observador de orden completo combina los mandos con todas las salidas de la planta para estimar mejor las variables del sistema. Finalmente, el tercero, el estimador de lazo abierto que recibe ese nombre porque se basa en un modelo de la planta y tiene en cuenta únicamente los mandos.

Es necesario utilizarlos para estimar los ángulos de Euler ya que no es posible medirlos directamente, sólo se pueden las velocidades angulares gracias a los giróscopos y las aceleraciones lineales gracias a los acelerómetros. Aunque se ha hablado del de Lazo abierto, del de Orden reducido y del de Orden Completo, en concreto, para los multicópteros los estimadores más comunes son unos que filtran las medidas dichas anteriormente y las combinan. Estos son conocidos como filtro complementario y filtro de Kalman. En este proyecto en concreto se utilizará el filtro complementario no lineal.

4.2.1 Filtro de Kalman

El filtro de Kalman puede funcionar en tiempo real ya que se trata de un observador en tiempo discreto recursivo. Se distingue de otros estimadores en que estos tienen una matriz de ganancias K, que es la que utiliza para estimar las variables de estado que no se pueden medir, mientras que Kalman es a partir de la matriz de covarianzas P que genera automáticamente cada vez una matriz K óptima.

Consta de dos fases principales: la primera, también llamada de predicción, calcula una estimación a priori del estado y de la matriz P basándose en el modelo y en los valores que tenían en el instante anterior. En la segunda, que es la de corrección, calcula las ganancias donde debe actuar la matriz K a partir de P y utiliza las medidas para obtener el error entre medida y modelo. Finalmente actualiza la estimación *a priori* del estado añadiendo el error multiplicado por la ganancia actualizando, también, la matriz de covarianzas que será posteriormente utilizada en la siguiente iteración.

Hay que tener en cuenta la dificultad existente al ser el Filtro Kalman sólo válido para sistemas lineales, por ello, para los multicópteros se utilizará el Filtro Kalman Extendido [19] que consiste en linealizar en cada iteración mediante la aproximación de Taylor.

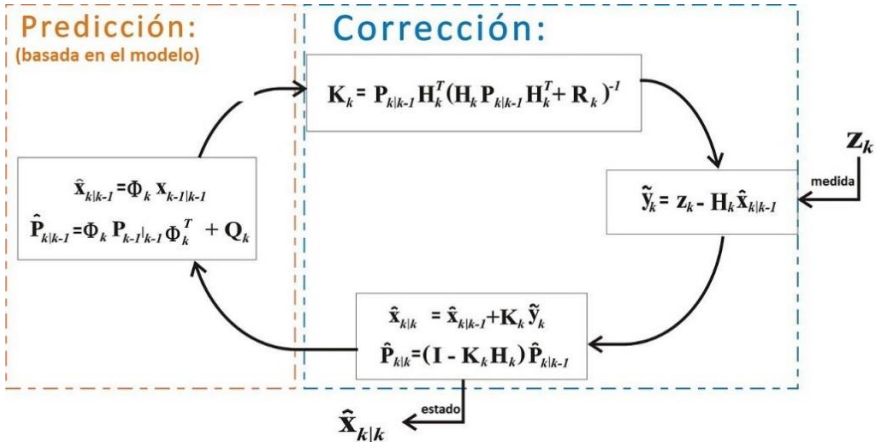


Figura 4.1. Esquema de funcionamiento del filtro de Kalman.

4.2.2 Filtro Complementario

El valor de los ángulos de Euler se puede estimar a partir de las velocidades angulares por integración. La expresión de éstas expresadas en ángulos de Euler es posible deducirse a partir de las velocidades en ejes solidarios [20] y una vez conocidas se integran para obtener estos ángulos. El principal problema es debido al error de medida que suelen tener los giróscopos, que es resultante fundamentalmente de la temperatura, e integrar este error supone acumular uno cada vez mayor en la estimación.

Sin embargo, a partir de las aceleraciones lineales, expresadas en ejes del cuerpo, se puede conocer la orientación del tricóptero por la proyección de la gravedad. El principal inconveniente de estimar los ángulos mediante esta forma es que las aceleraciones propias del cuadricóptero alteran el valor real de dichas proyecciones de la gravedad.

Con este filtro se combinan las medidas de ambos, tanto de los giróscopos como de los acelerómetros eliminando así primero las componentes que nos desvían de las mediciones esperadas. Por ello primero hay una etapa de calibración en la cual además de establecer un nuevo punto de referencia, se pasan las medidas de los giróscopos por un filtro paso alto, que elimina el error de medida que se supondrá constante a igual temperatura, y las de los acelerómetros por un filtro paso bajo, para quedarse así con el valor fijo que es la gravedad.

Éste es muy útil para estimar las variables de estado que provienen de la IMU [21] porque se unifican ambas medidas utilizando previamente un filtro paso alto para las medidas de los giróscopos y un filtro paso bajo para los acelerómetros, de tal forma que no se tienen en cuenta las componentes que provocan los errores.

4.3 Control del Tricóptero

La estrategia de control sobre el tricóptero se ha estructurado de manera sistemática. Primero se diseñó el módulo por el cual se calibraban las medidas de la IMU. Se ha seguido con el Filtro Complementario No Lineal que nos proporciona los ángulos de Euler (*roll, pitch y yaw).*

Éstos ángulos (convertidos a grados), junto con las medidas normalizadas recibidas por la emisora entran directamente al control de estabilización, que para este proyecto se utilizará un PID modificado. Una vez obtenidas las fuerzas del control se pasa a otro bloque (aún dentro del Control de Estabilización) que lleva como nombre "Mixeador" el cual se encarga de generar los mandos que serán las salidas para generar el *PWM* que saldrán a los ESC y al Servo.

4.3.1 Calibración de la IMU

Se ha diseñado un método de calibración de giróscopos y acelerómetros para eliminar la componente continua de las medidas. Que actúa como un filtro paso alto para los giróscopos y filtro paso bajo para los acelerómetros. Este sistema permite reducir de manera exponencial los offsets de las medidas, de manera que al cabo de 20 segundos son prácticamente 0.

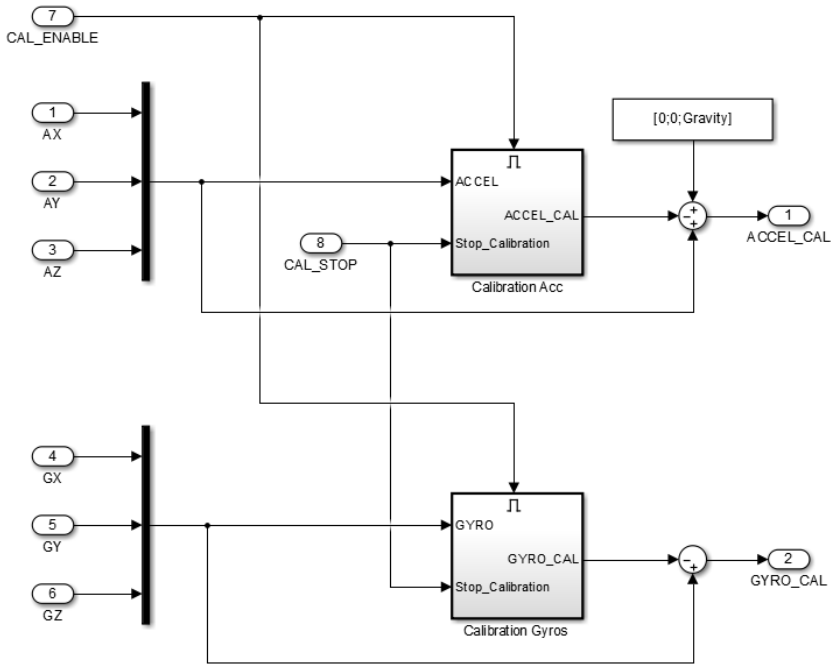


Figura 4.2. Esquema de calibración de la IMU.

4.3.2 Estimación por Filtro Complementario No Lineal

A este bloque entran las medidas, ya calibradas, de los giróscopos y acelerómetros. Como

ya se ha explicado anteriormente en el Apartado 4.2.2, gracias el filtro complementario no lineal se obtienen los ángulos de Euler estimados. Por medio de combinaciones de las medidas obtenidas gracias a la IMU se normalizan y se van estimando los ángulos reales. En este punto se ha tenido en consideración que había medidas recibidas tanto de los acelerómetros como de los giróscopos que estaban invertidas y provocaban que los ángulos resultantes fueran negativos cuando debían salir positivos.

4.3.3 Control de Estabilización

El control de estabilización empleado es un control PID modificado [22]. Se realimentan los 3 ángulos y las 3 velocidades angulares multiplicados por sus respectivas ganancias, a los que se añaden los términos no lineales. Aunque las velocidades angulares medidas por los giróscopos no son las derivadas de los ángulos de Euler, se pueden aproximar para inclinaciones pequeñas en alabeo y cabeceo.

La linealización para la realimentación se hace definiendo como vector de estado

$$X^T = [\phi\ \theta\ \psi\ \dot{\phi}\ \dot{\theta}\ \dot{\psi}] \tag{4.2}$$

Y como vector de mandos

$$u^T = [u1\ u2\ u3\ u4] \tag{4.3}$$

Dónde estos son los que se han explicado previamente en la Ecuación (3.6)

La estrategia que se va a utilizar es la propuesta por el método *Feedback Linearizing* [16] [22]. El principal objetivo es controlar las fuerzas de los mandos, para ello se procederá a crear unos mandos virtuales de tal manera que la planta a tratar sea la siguiente:

$$Planta_{objetivo} = \frac{1}{s^2} \tag{4.4}$$

Se presenta el problema de que las únicas medidas de las que se disponen, las de los giróscopos, no corresponden con los ángulos de Euler. Por ello para comenzar a diseñar los mandos virtuales se necesita antes hacer la conversión.

$$\begin{bmatrix}\omega_x\\ \omega_y\\ \omega_z\end{bmatrix} = \begin{bmatrix}1 & tan\theta \cdot sin\phi & tan\theta \cdot cos\phi\\ 0 & cos\phi & -sin\phi\\ 0 & \frac{sin\phi}{cos\theta} & \frac{cos\phi}{cos\theta}\end{bmatrix}^{-1} \cdot \begin{bmatrix}\dot{\phi}\\ \dot{\theta}\\ \dot{\psi}\end{bmatrix} \tag{4.5}$$

Donde ω_{xyz} son las velocidades angulares de los giróscopos ya en términos del sistema solidario al cuerpo pues la matriz inversa que multiplica ofrece la relación de lo que miden los

giróscopos con los respectivos ángulos de Euler. Ésta expresión ya se utilizó para calcular los propios ángulos de Euler en la Ecuación (3.16).

Como el objetivo son mandos (fuerzas), las velocidades angulares que se han obtenido previamente se derivan para conseguir así las aceleraciones angulares de los ángulos de Euler. De esta manera se cumple nuestro principal objetivo, que es controlar el vector de estados y su derivada.

$$\frac{dX}{dt} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \\ \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} \tag{4.6}$$

Una vez obtenidas construimos nuestros mandos virtuales que son las fuerzas que nos permitirán, una vez aplicado el control diseñado, realizar el camino inverso para sonsacar los verdaderos mandos. El cómo se obtienen las aceleraciones angulares en ángulos de Euler, así como los productos necesarios para la creación de dichos mandos virtuales vienen explícitamente detallados en el código del Anexo C.

Con nuestra planta reducida a algo tan sencillo como una doble integral, y unos mandos perfectamente diseñados, se procede a diseñar el control que se desea aplicar siendo tan sólo necesario el diseño de ω_n , ξ y p , la frecuencia natural, el factor de amortiguamiento y el polo respectivamente. El control PID, con una planta simple como la que ha resultado, viene detallado también en el Anexo C.

Tras obtener las variables del control se procede a realizar el camino inverso, que es la fundamentalización del *feedback linealization*. Ahora se despejan los mandos reales de los virtuales ya elegidos. Para ello se usarán matrices auxiliares que en este caso se han llamado *matC1, matC2 y matC3* que en ellas están las relaciones necesarias entre matrices de inercia y aceleraciones del cuerpo.

De manera que los mandos reales quedan de la siguiente forma:

$$u = matC1 \cdot u_{virtual} \cdot \left(\omega_{xyz}\, x\, \left(matI \cdot \omega_{xyz}\right)\right) + matC3 \cdot \omega_{xyz} \tag{4.7}$$

Donde $matI$ es la simplificación de los momentos de inercia usada en la Ecuación (3.15) Ahora ya se tienen la fuerza de sustentación y las fuerzas que generan los pares de alabeo, cabeceo y guiñada. Ya se tienen todos los elementos para dar paso al siguiente punto, el mixeador, que dependiendo del dron en cuestión se usará un modelo u otro. Para este proyecto se ha usado el del tricóptero.

4.3.4 Mixeador

A este bloque le llegan las fuerzas calculadas en el control de estabilidad. La idea del Mixeador es realizar el camino inverso que se explicó en el Apartado 3.1. Para ello se acude a los *mandos* que se formaron relacionados con la fuerza de empuje que simplificaba la expresión de los momentos y fuerzas. La Ecuación (2.6) proporciona lo necesario para generar la matriz de transformación *matT* junto con el vector de variables escogido (en pu) para obtener las relaciones. Se puede observar en la siguiente expresión

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = matT * \begin{bmatrix} T1_{pu} \\ T2_{pu} \\ T3_{pu}\cos(\alpha) \\ T3_{pu}\sin(\alpha) \end{bmatrix} \tag{4.4}$$

Donde el vector de mandos en el que nos basamos es:

$$\begin{aligned} u_1 &= T1_{pu} + T2_{pu} + T3_{pu}\cos(\alpha) \\ u_2 &= T1_{pu} - T2_{pu} + \frac{L_{height}}{L_{roll}} T3_{pu}\sin(\alpha) \\ u_3 &= T1_{pu} + T2_{pu} - \frac{L_{pitchR}}{L_{pitchF}} T3_{pu}\cos(\alpha) - \frac{1}{ratio_{TDT} * L_{pitchF}} T3_{pu}\sin(\alpha) \\ u_4 &= T1_{pu} - T2_{pu} + T3_{pu}\cos(\alpha) - \mathrm{ratio}_{\mathrm{TDT}} * L_{pitchR} * T3_{pu}\sin(\alpha) \end{aligned} \tag{4.5}$$

La constante $\mathrm{ratio}_{\mathrm{TDT}}$ que aparece ahí es la relación entre el *thrust* y el *drag* (*b* y *d*) obtenida a partir de la recta de especificación expuesta en Apartado 3.1.5

Una vez que se tiene claro los que se quiere conseguir, se construye la *matT* expresando estos mandos en función de nuestro vector de variables quedando de la siguiente forma:

$$matT = \begin{bmatrix} 1 & 1 & 1 & 0 \\ 1 & -1 & 0 & \frac{L_{height}}{L_{roll}} \\ 1 & 1 & -\frac{L_{pitchR}}{L\_pitchF} & -\frac{1}{ratio_{TDT} * L_{pitchF}} \\ 1 & -1 & 1 & -ratio_{TDT} * L_{pitchR} \end{bmatrix} \tag{4.6}$$

De esta manera, multiplicando el vector de mandos por la inversa de *matT* se calcula nuestro vector de variables $[T1, T2, T3cos(\alpha), T3sin(\alpha)]$'. Ahora se pasa a la etapa final del Mixeador donde el objetivo es generar la señal de los actuadores que irán a los 3 ESCs y al servo. Para ello hay que conseguir en primer lugar $[T1, T2, T3, \alpha]$para posteriormente

convertir todos en los pulsos necesarios *PWM*. En la siguiente figura se puede observar cómo se han de combinar para llegar hasta la señal de los actuadores. Cabe destacar la conversión mediante la caracterización de los rotores y del servo también explicada en el Apartado 3.1.4.

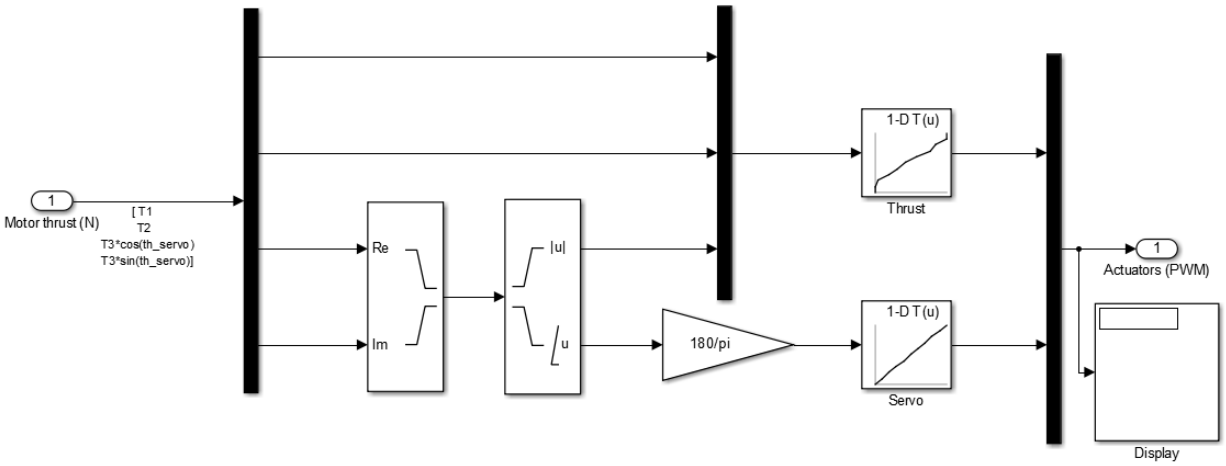


Figura 4.3 Paso final del Mixeador para obtener los actuadores

Capítulo 5

Implantación

5.1 Software y Simulink

Para hacer la programación del control y descargarlo con éxito en la tarjeta es necesario usar un software específico. En este proyecto se utilizará para ello Matlab y Simulink pero es preciso diseñar en este entorno los bloques específicos para que la tarjeta sea reconocida como tal y responda correctamente.

5.1.1 Driver blocks

En el anterior proyecto se diseñaron una serie de driver blocks para la tarjeta HKPilot Mega [23] . Los *Driver blocks* son funciones programables en Matlab y Simulink que permiten descargar un código desde el entorno al controlador. Pueden ser tanto entradas (*inputs)* como saludas (*outputs)* según convengan para recoger medidas del controlador o para actualizar registros o saludas. Hay que tener en consideración que estos no afectan durante las simulaciones.

Para este proyecto se ha encontrado una dificultad importante. Al cambiar a la tarjeta de control OpenPilot Revolution ninguno de los *Driver blocks* ya buscados y preparados de años anteriores ha servido. Para ello se ha tenido que recurrir a uno bloques llamados Waijung que se explicarán a continuación junto con las especificaciones de la nueva controladora.

5.1.2 OpenPilot Revolution

Esta tarjeta de control nace con la idea de aprovechar el poder y la imaginación de la comunidad de código abierto. OpenPilot ha creado una nueva generación de potentes controladores de vuelo a bajo coste. Proporciona una base sólida para proyectos comerciales y de investigación por igual [24].

Utiliza como microcontrolador el STM32F4 que utiliza de hardware la unidad de coma flotante o FPU (*Floating Point Unit)*, que presenta un enorme avance para controladoras de drones en aficionados. La FPU permite el procesamiento preciso de baja latencia de las mediciones utilizando avanzados algoritmos de estimación.

Además de tener la unidad de coma flotante integrada, el chip STM32F4171G, contiene un ARM Cortex-M4 como núcleo a 210MIPS y funciones de saturación DSP [25] *digital signal processor.* La CPU tiene una gran gama de módulos de hardware integrados que permiten la programación y funcionan independientemente, lo que hace que ésta se sobrecargue poco. Estos incluyen 14 *Timers* independientes para múltiples canales, 3 ADC síncronos de muestreo que sirven hasta para 24 canales, 2 DAC, y un controlador de memoria matricial con DMA 16-stream.

Los módulos de comunicación incluyen USB 2.0, 3x I2C, SPI x3, 4x USART, 2x CAN y la SIDO. Todos estos se pueden configurar para acceder desde diferentes pines habilitados para tales efectos.

La actual placa de control que se va a utilizar en este proyecto se desarrolló originalmente con el concepto de conseguir una controladora de multicópteros potente a un coste reducido respecto a sus otras alternativas. La solución resultante contiene como microcontrolador el potente STM32. Consta de dos núcleos principales, por un lado, el que sería el micro y sus conectores y por otro el AHRS (*Attitude and heading reference system*) que donde se encuentran los sensores de los giróscopos y acelerómetros. Las conexiones de estos se realizan mediante conexión SPI.

Características

- Desarrollo de código abierto.
- Flexi-IO port de entrada/salida.
- Mainport (telemetría) USART serie w/ velocidad de transmisión ajustable.
- FlexiPort para funciones de telemetría.
- La salida de RF de 433 MHz.
- Power Sensor/Sonar Port.
- STM32F4 CPU de 32 bits.
- Procesador de paquetes digitales impulsado ARM32.
- Módulos de comunicación USB2.0, I2C, SPI, USART, CAN y Sido.
- 14 Timers de varios canales.
- Giróscopo de 3 ejes.
- Acelerómetro de 3 ejes.
- Magnetómetro de 3 ejes.
- Sensor de presión barométrica.
- PWM / PPM.

Especificaciones

- Voltaje de entrada: **5 ~ 8.4V**
- Enlace de telemetría: **433 MHz**
- Conectores de alimentación: **JST SH-4-pin**
- Conector de antena: **SMA**
- Dimensiones: **36 x 36 mm**
- Peso: **9 g**

5.1.3 Waijung

"*Waijung*" o "*ไวจัง*", que significa *"so fast"* es el "*Blockset*" de Simulink que se usa para generar de manera fácil y automática el código en C para el programa Matlab y el entorno de trabajo de Simulink para diferentes tipos de controladores como el STM32 de la tarjeta de control OpenPilot [26].

Actualmente Waijung ha sido diseñado específicamente para apoyar a la familia de microcontroladores STM32F4, que pertenece a STMicroelectronics.
Tanto el conjunto del Blockset de Waijung como los DriverBlocks para el STM32F4 han sido completamente rediseñados con nuevas características y mejoras.

Para poder realizar una buena comunicación con nuestra placa es necesario establecer el llamado "Target Setup" y seleccionar el micro exacto que tenemos. Así como para habilitar cualquier tipo de comunicación se ha de implantar en nuestro entorno de Simulink el BlockSet de configuración pertinente en el que se seleccionan las características básicas de funcionamiento, velocidad de transmisión, períodos de muestreo internos, timers, módulo SPI, UART, I2C…

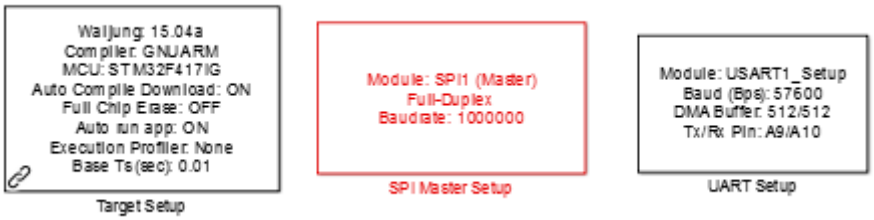


Figura 5.1 Bloques de Configuración del Microcontrolador, del módulo SPI y UART

5.2 Máquina de estados

A la hora de controlar un tricóptero se ha de trabajar con múltiples variables que tienen cada una que comenzar a actuar en un determinado momento en concreto debido a ello el control es algo complejo. No se puede olvidar que tampoco el funcionamiento que se espera de él es el mismo en todos los instantes pues sigue un proceso que ha de ser secuencial si se

quiere asegurar una buena experiencia. Primero tiene que arrancar para poder despegar, seguido de volar y finalmente aterrizar. Para implementar el control sobre el tricóptero se han tenido que diseñar dos máquinas de estados, la primera fue una adaptación de la realizada el año anterior con algunas mejoras pero que se tuvo que desechar al tener que suprimir el uso del PPM y la segunda ha sido la solución para una recepción de tan sólo 4 canales. Ambas están orientadas para vuelo manual.

5.2.1 Máquina de estados con 4 canales

La Máquina de estados que se ha tenido que diseñar, para solventar los problemas ocasionados por la única recepción de 4 canales, ha sido con el objetivo de controlar la activación del control. Inicialmente, según se conecta la batería, se arman los motores y se espera una señal de activación. En la máquina de estados antigua dicha señal, que recibía por nombre "Cambio" era manejada desde el Ch5 de la emisora, puesto que ya no se tiene acceso a dicho canal por los inconvenientes presentados en la configuración de nuestra tarjeta OpenPilot se tuvo que recurrir a otra solución para poder tener una transición segura entre estados. Para ello se ideó crear una variable "CH12" que viene a ser la suma de los pulsos de los canales 1 y 2. Ésta, junto con el valor del *Throttle* permitirán hacer la transición correcta entre estados.

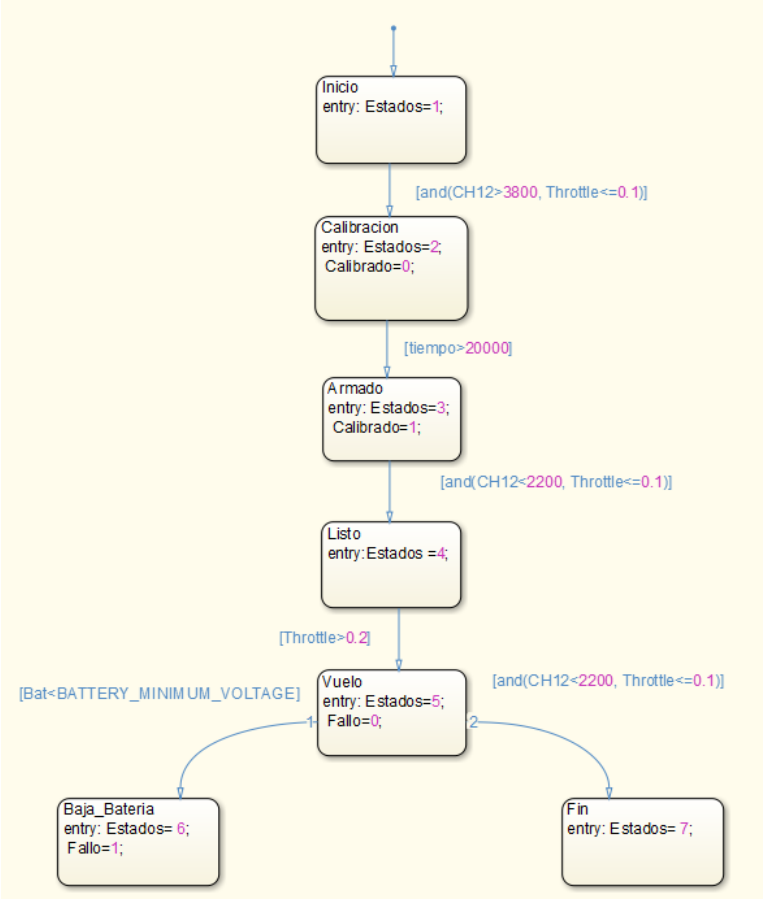


Figura 5.2 Máquina de Estados con 4 canales

Como se puede observar, solamente se entra en el estado de Calibración mediante la combinación de los dos *sticks, Throttle* completamente a cero (por motivos de seguridad) y los canales 1 y 2 al máximo. De esta manera comienza el periodo de calibración de los sensores internos que durará 20 segundos, se comprobó que un tiempo menor producía fallos es la estimación de los ángulos. Una vez concluido este estado pasaríamos al estado previo al vuelo llamado "Listo", de nuevo mediante una combinación del CH12 y el *Throttle*. Éste último siempre abajo marcando el cero para evitar problemas de puesta en marcha y la otra señal ahora, en vez de antes que era llevarlos a su valor máximo, en su valor mínimo. Una vez cumplida dicha condición pasamos al estado de "Vuelo" donde ya se puede controlar el tricóptero de manera manual con la emisión de las señales de alabeo, cabeceo, empuje y guiñada de la emisora. Hay dos formas de salir de este estado, la primera es por baja batería.

Desde el *Power Sensor* se recibe el valor de la batería y si ésta llega a un valor crítico que pueda poner en peligro el vuelo del tricóptero entonces la señal de mínimo voltaje activa la transición a este modo que hace que el dron aterrice. Por otro lado, también se puede poner fin al vuelo de manera manual teniendo el *Throttle* al mínimo y usando la combinación que sirvió previamente para entrar en el mismo estado de "Vuelo".

5.2.2 Máquina de estados antigua

Esta máquina de estados, como se puede ver en la imagen a continuación, contenía la variable "Cambio" que provenía del Ch5 de la emisora. Era una medida muy cómoda pues permutando los *switches* se podía pasar de un estado a otro con total comodidad. Respecto a la utilizada el año anterior se habían añadido una serie de ajustes para mejorar su implementación. El tiempo de calibración que por entonces estaba en 10 segundos se cambió por 20 para una mayor precisión a la hora de estimar los ángulos de Euler. Por otro lado, se añadió un estado extra en el cual, cuando en pleno vuelo se cambiaba el Ch5 a una determinada posición con la idea de finalizar el vuelo, el dron en vez de entrar en modo "Fallo =1", entraba en modo "Aterrizaje" de manera que la potencia del *Throttle* se iba disminuyendo poco a poco hasta que finalmente aterrizaba y pasado un tiempo, si no recibía ninguna señal de la emisora, entraba en el estado "Fin".

5.3 Ajuste del control en tiempo real

Una vez comprobado su funcionamiento en simulación, diseñado ya el control, se decidió a volcarlo sobre la tarjeta de control para ensayarlo sobre el tricóptero. Fue necesario, sin embargo, modificar alguno de los parámetros para tratar de lograr un vuelo lo suficientemente estable para que respondiera a los mandos de la manera adecuada.

Esto supuso el inconveniente de la pérdida de tiempo innecesaria al tener que estar cambiando parámetros para afinar el control en el ordenador seguido de la descarga consecutiva del programa en la placa. Para ello el año pasado se pensó en un sistema de ajuste de los principales parámetros del control en tiempo real por medio de los canales 6,7 y 8 de la emisora. En un principio esto era lo que se iba a hacer este año hasta que surgió la dificultad de que sólo se podían leer los 4 primeros canales de ésta y hubo que idear otra opción. Para ello en el fichero de simulink "*SERIAL_RW_PC*", en el **Error! Reference source not found.**, se establecieron unos diales que enviaban por UART vía Bluetooth al OpenPilot Revolution los parámetros a modificar de estos canales. Se recibían en la tarjeta de control sin problema y permitían así el ajuste de parámetros en vuelo desde el ordenador.

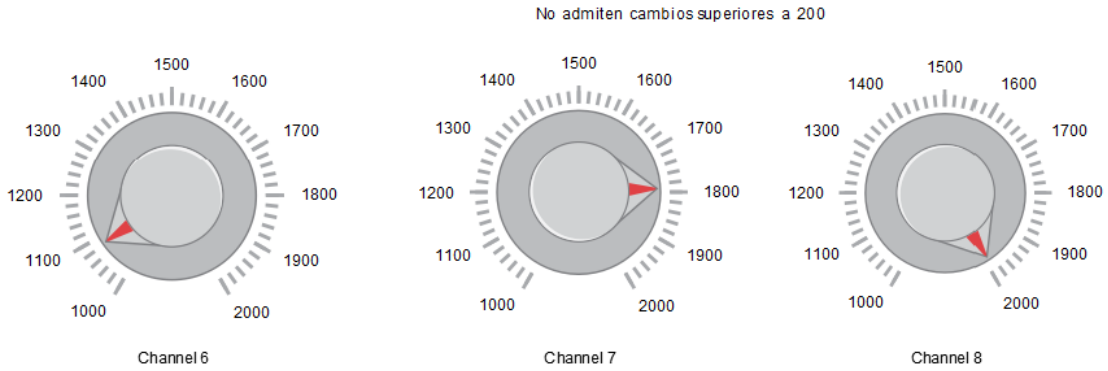


Figura 5.3 Diales usados para la modificación de parámetros

Capítulo 6

Resultados Experimentales

6.1 Implantación del Modelo del Tricóptero

Se ha adaptado el modelo dinámico del cuadricóptero de proyectos anteriores. Éste estaba basado en el que se utilizó en Anexo F.Se trata de una estructura que parte de una serie de variables de estado cuyas derivadas se calculan según las ecuaciones que hemos visto en los Capítulos 3 y 4. A continuación se puede observar un esquema realizado en Simulink:

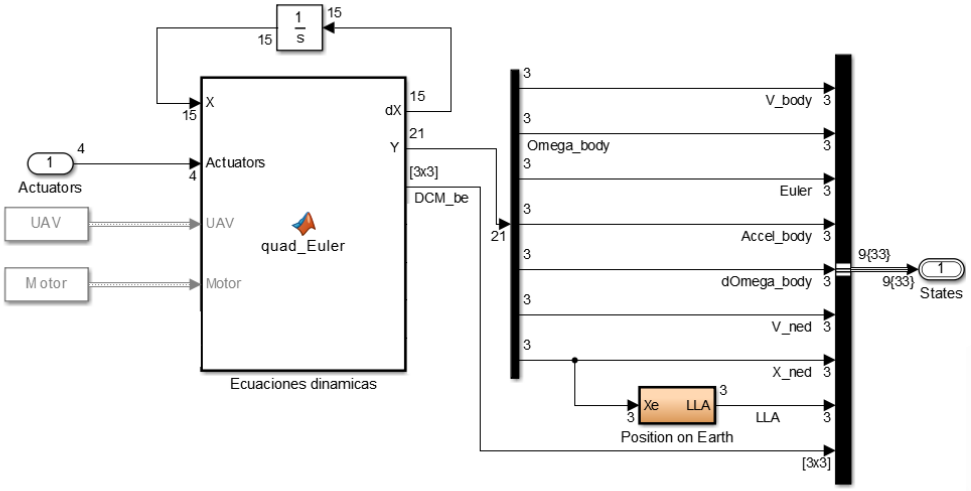


Figura 6.1 Esquema del Modelo Dinámico de un Tricóptero

El vector de derivadas se integra y se realimenta para que las variables de estado sean entradas. Como entradas se encuentran, las cuatro señales de control que llegan, 3 a los ESCs y 1 al Servo; y los parámetros físicos como los momentos de inercia o la masa. Las salidas del modelo son, por un lado, el vector de derivadas y por otro, todas las variables que se pueden medir o que pueden servir para dar información acerca del funcionamiento del control que se quiera comprobar. Entre otras destacar los ángulos de Euler reales, las velocidades lineales o las aceleraciones del cuerpo.

El vector de estado que se realimenta tiene 15 variables de estado:

$$X = \left[\, \theta, \phi, \psi, v_x, v_y, v_z, \omega_x, \omega_y, \omega_z, r_x, r_y, r_z, \omega_1, \omega_2, \omega_3 \right]^T \tag{6.1}$$

En orden: Primero los ángulos de Euler seguidos de las 3 componentes de la velocidad lineal en el sistema inercial, las velocidades angulares en los ejes propios del tricóptero, las 3 componentes de la posición de su centro de masas en el sistema inercial y las 3 velocidades angulares de los motores.

Del control provienen los mandos que son las 4 señales, las que van a los ESCs y a los Servos. Todas vienen en la franja entre 1000 y 2000 µs, que es el tiempo en el que todos los componentes funcionan para el PWM. Los dos vectores que entran *UAV* y *Motor* contienen los valores definidos en el fichero "ModelParam_Tricopter" del Anexo B, que se unifican en un *bus* de datos para entrar en el bloque del modelo. Entre otros, en el UAV se destaca la masa, los brazos del tricóptero, los momentos principales de inercia. En el *bus* de Motor se pueden encontrar el Kv de los motores, su inercia, la curva de caracterización, la tensión, el ángulo de rotación del servo, su recta característica…

Por último, el vector de salidas contiene las velocidades lineales, las medidas de los giróscopos y acelerómetros, la posición respecto el sistema inercial, los ángulos de Euler reales y las derivadas de las velocidades angulares del cuerpo.

El bloque ha sido creado mediante *Matlab Function* que permite implementar en *Simulink* mediante ecuaciones un modelo matemático. Este permite definir las entradas y las salidas realizando dentro las operaciones necesarias según las ecuaciones dispuestas. Las condiciones iniciales están dispuestas en el integrador de la realimentación de las variables de estado. Esto permitiría estudiar diferentes puntos de comportamiento según diferentes salidas.

En el 0 se encuentra recogido el código completo donde se calculan las derivadas de las variables de estado y las salidas en función de las ecuaciones de movimiento y las de los motores. Se han despreciado algunos efectos como fuerzas de rozamiento aerodinámico sobre la estructura o la aceleración de las hélices.

6.2 Entorno de Simulación

Se ha decidido crear un entorno de simulación para así tener que evitar hacer modificaciones al control diseñado a la hora de implantarlo sobre el hardware real. Se ha tratado de conseguir que todo sea lo más parecido al fichero maestro "*OPENPITLO_REVO.slx*", por ello el mismo bloque de "*CONTROLLER*" se usará tanto para simular como para volcar en la placa. A partir de ahí se han hecho modificaciones para incluir tanto el modelo dinámico explicado previamente, como la generación de pulsos para emular la emisora, así como el entorno de monitorización, lugar donde se muestran todas las variables importantes para poder afinar un buen control. Hay otro donde además se definen las condiciones de la simulación.

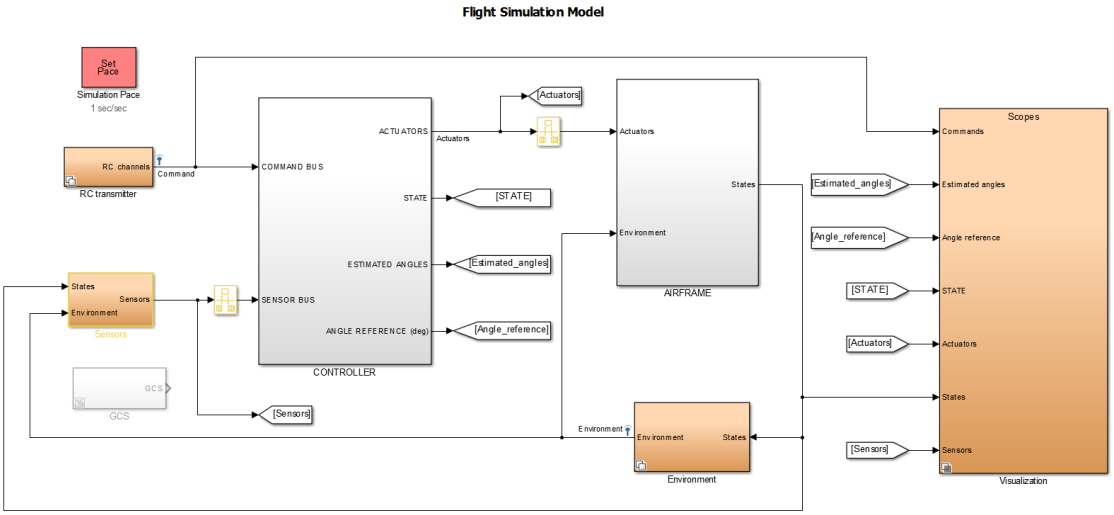


Figura 6.2 Imagen general del Entorno de Simulación

El principal recurso que ha servido como nexo para los diferentes elementos del entorno de simulación ha sido mediante la utilidad de *signal routing,* en concreto los bloques *from* y *Go to*, también conocidos como etiquetas. Para mantener coherencia se han utilizado como "variables locales" para así trabajar de forma más modular y poder usar bloques comunes tanto en simulación como en implementación. Otro medio muy utilizado han sido los *buses* que han permitido guardar estructuras de datos para no generar vectores de parámetros difíciles de trabajar con.

Una gran implementación que se ha hecho en este proyecto ha sido el uso de la utilidad llamada "*Variant Manager*". Esta permite desarrollar dentro de un mismo bloque o subsistema diferentes bloques que sólo van a entrar en funcionamiento mediante una selección que se puede realizar mediante cualquier *script* de Matlab. Con ella se ha podido establecer un fichero maestro único para todos los proyectos de drones independientemente de éstos en cuestión. Simplemente señalando que se trata de un "tricóptero" se habilitan tanto el modelo como el mixeador adecuado, lo mismo para "cuadricóptero" o "helicóptero coaxial"

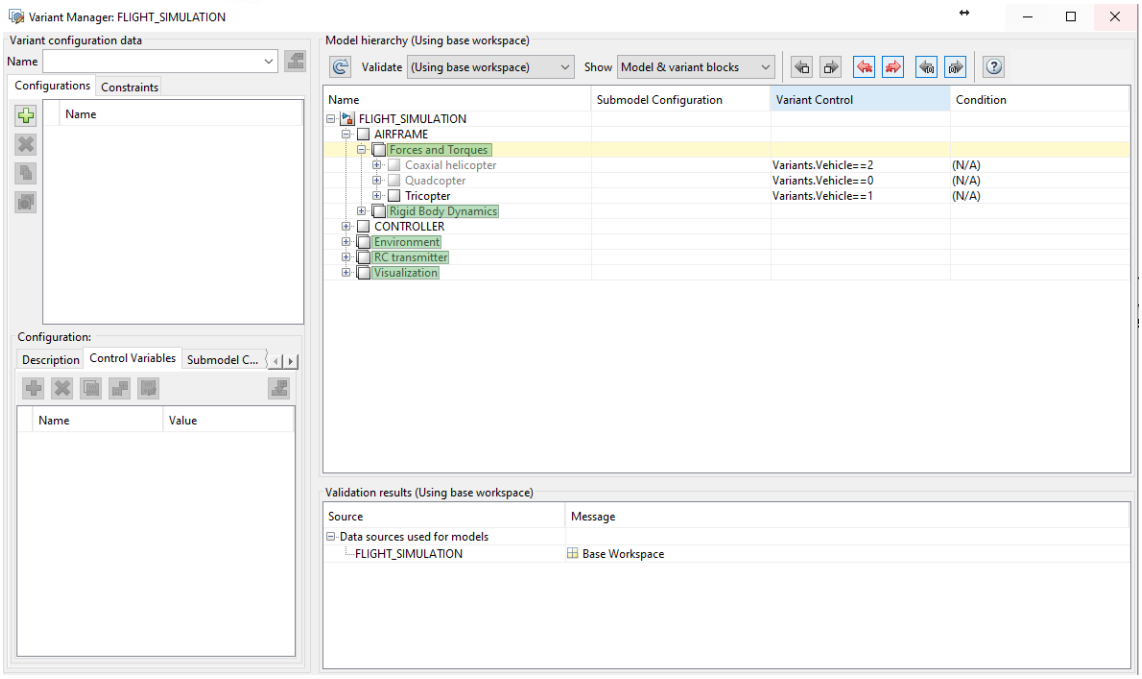


Figura 6.3 Variant Manager de Simulink

6.3 Ensayo de Simulación

Para los ensayos de simulación se ha utilizado un diseño por el cual se introducía en el entorno de simulación una referencia. Un sistema de geoposicionamiento en el que el tricóptero en este caso tratará de alcanzar. Primero calcula la altura y según va actuando el modelo dinámico va aplicando diferentes mandos como reacción para conseguir que haga lo previsto. De esta manera se puede observar cómo se comportaría la aeronave respecto a condiciones típicas de vuelo. A continuación, se muestran imágenes de los buenos resultados obtenidos con la monitorización de los ángulos de Euler, la salida de los *PWM* y la posición.

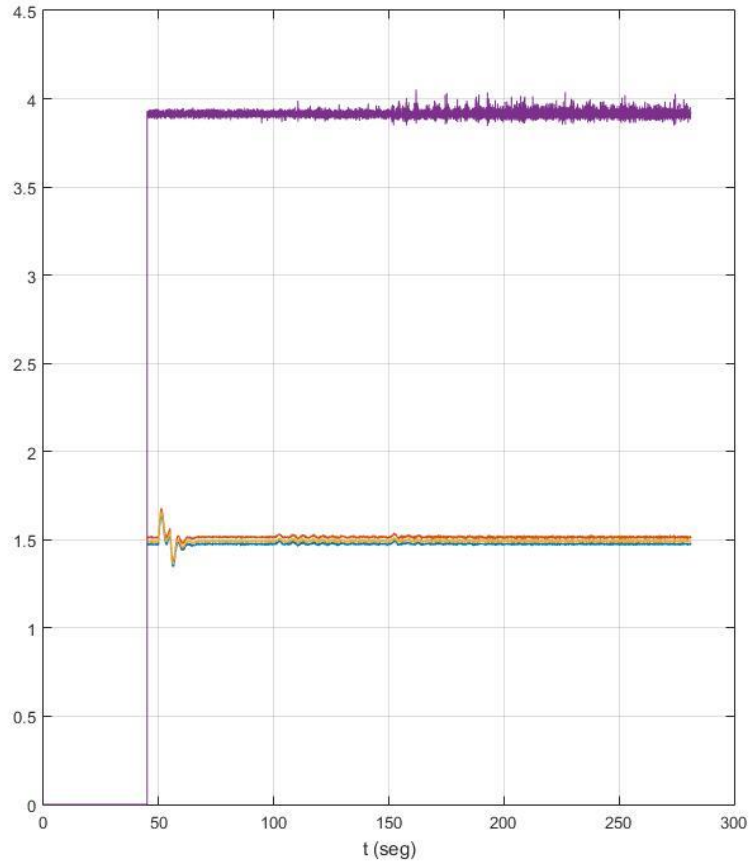


Figura 6.4 Actuadores en Simulación

Como se puede observar, el *PWM* del servo está a un poco más de 1520µs que sería equivalente a unos 4º, esto es necesario para poder compensar los pares y así el tricóptero ascienda. Puesto que el resto de motores reciben la misma señal del actuador.

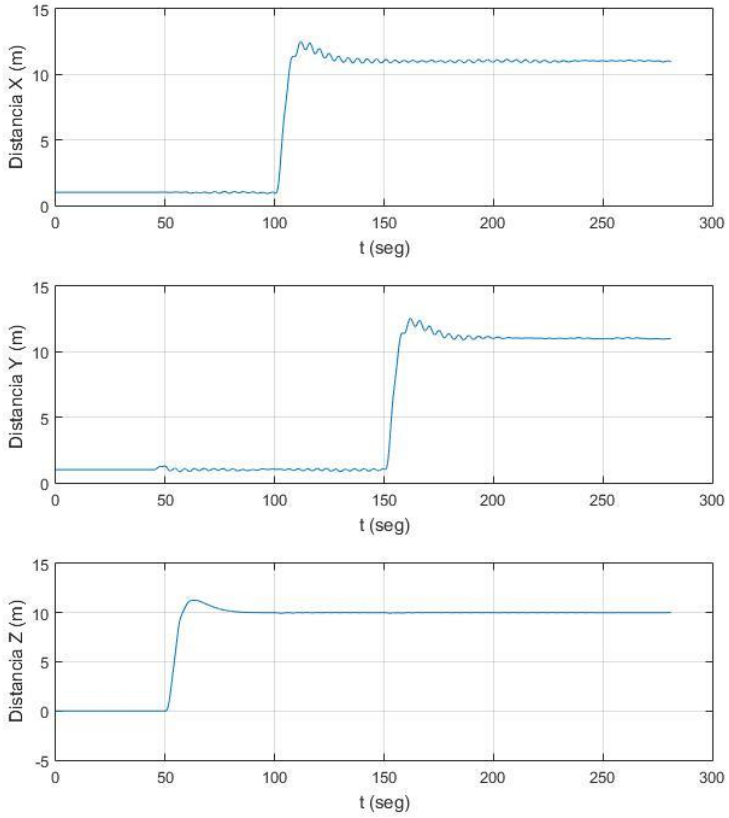


Figura 6.5 Posiciones en el Simulador

En esta gráfica, que representa la posición de cada uno de los ejes, puede corroborar el buen funcionamiento de la simulación. La simulación estaba preparada para mantener al tricóptero a una altura fija de 10m, seguido de un desplazamiento en X de otros 10m finalmente en Y otros 10M y que a partir de ahí se mantuviera. Se puede apreciar la principal dificultad que tiene el vuelo con el tricóptero que es al inicio, cuando se da *throttle*, que se observa como aparece una fuerza que le hace desplazarse en Y, provocando un desplazamiento inicial.

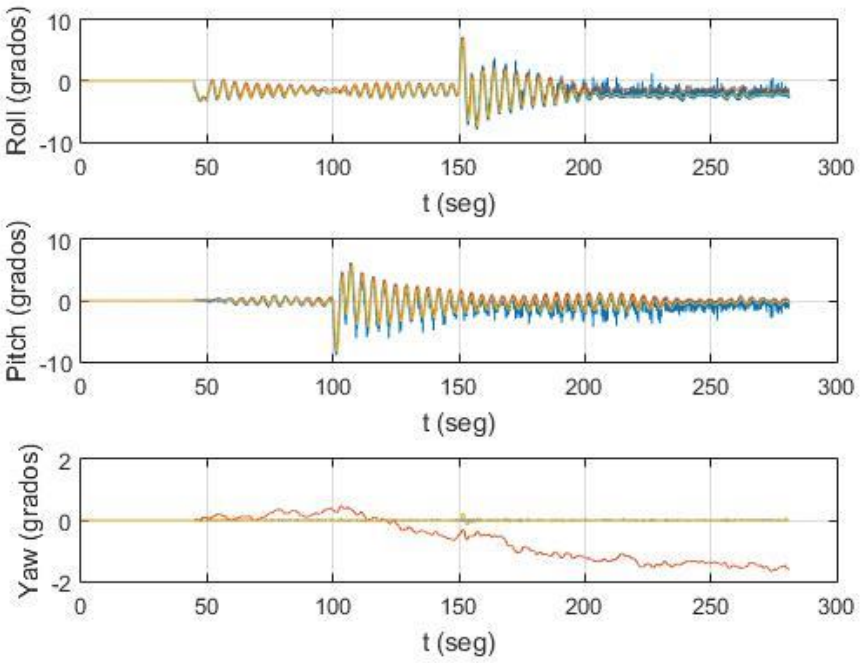


Figura 6.6 Ángulos de Simulación

Aquí aparecen representados los ángulos de cada uno de los ejes de Euler. Cada color representa uno de los ángulos, el ángulo de referencia, el real y el estimado en ese orden.

- Amarillo es la referencia
- Azul es el real
- Rojo el estimado

Capítulo 7

Conclusiones y futuros avances

La finalidad de este proyecto era introducir el uso de drones tricópteros como alternativas a la hora de trabajar con aeronaves, para ello la idea original era la implantación de un control que permitiera su manejo en recintos cerrados mediante vuelos manuales.

En ese sentido los avances realizados han sido notorios, empezando por lograr desarrollar un modelo dinámico de la aeronave junto con el mixeador que convertía las salidas del control a sólo tres motores y un servo. También se ha conseguido desarrollar un control por el que el tricóptero responde satisfactoriamente. Se presenta en este capítulo un resumen de lo que ha sido el proyecto en sí, así como una serie de mejoras que se pueden tener en cuenta de cara a futuros proyectos relacionados.

7.1 Resumen de tareas realizadas

Durante la realización del proyecto se han llevado a cabo diversas tareas que han sido necesarias para poder llegar al objetivo final, el vuelo manual del tricóptero. Como pueden servir de base sólida de partida de cara a futuros proyectos ahora a continuación se detallan aquellas que han sido más relevantes:

- Diseño del modelo de un tricóptero basado en [27].
- Diseño del Mixeador que permitiera convertir las salidas del control a los tres ESCs y el servo
- Creación de un nuevo entorno de simulación único para todos los proyectos de drones, con la ayuda de la utilidad de "*Variant Manager*" de Simulink.
- Diseño de un sistema de calibración de giróscopos y acelerómetros de manera que se elimine la componente continua de las medidas.
- Mejora del estimador de ángulos mediante el uso del Filtro Complementario No Lineal.
- Diseño de una nueva máquina de estados para solventar las dificultades encontradas de lectura única de 4 canales
- Diseño de un diagrama de bloques para modificar los parámetros del control en tiempo real desde el ordenador.
- Diseño de un fichero maestro único para unificar todos los proyectos de drones
- Diseño de un entorno de configuración apropiado para el trabajo con la tarjeta de control OpenPilot Revolution haciendo uso de los bloques de Waijung
- Mejora del estimador de ángulos a uno con mayor precisión y rapidez a partir de un filtro complementario no lineal.
- Desarrollo de un sistema de telemetría inalámbrica mediante bluetooth para una rápida monitorización con el ordenador.

7.2 Resumen de resultados

Con respecto al grado de cumplimiento de los objetivos se puede confirmar que los objetivos principales de este proyecto, adaptación del actual modelo del cuadricóptero al correspondiente tricóptero, diseño del mixeador, así como el diseño del control para simulación e implantación) se han cumplido plenamente. Además, se ha avanzado en la consecución del objetivo final de navegación guiada mediante una emisora de radiofrecuencia. Como resultados del proyecto se pueden enunciar los siguientes:

- Modelo completo en Simulink del *Airframe* de un tricóptero que incluye todas sus especificaciones.
- Mixeador para la conversión de las fuerzas del control en los tres ESCs y el servo.

- Entorno de Simulink donde se recogen todos los elementos que forman parte del control, la calibración de la IMU (giróscopos y acelerómetros), el Filtro Complementario No Lineal y por supuesto el Control de Estabilidad. Todos ellos supeditados a la máquina de estados para su buena calibración y su propio funcionamiento con las precauciones de seguridad pertinentes.
- Fichero de monitorización en Simulink de las principales variables del tricóptero en vuelo y desde donde modificar los parámetros del control.
- Con los tres anteriores se ha conseguido un entorno de simulación completo en Simulink con interfaz gráfica donde visualizar todas aquellas variables importantes para la mejora del control junto con un programa de navegación prediseñado.
- Control de estabilización implantado sobre el tricóptero en el que se observa cómo reacciona correctamente ante los distintos ángulos estimados.

7.3 Futuras mejoras

Para facilitar la consecución del objetivo último de vuelo manual mediante radiofrecuencia se han concretado una serie de mejoras de cara a los sucesivos proyectos que continúen en esta línea de trabajo.

- Encontrar una solución al problema encontrado con el módulo PPM debido a los *Timers* y sus conflictos de sincronización para poder obtener mediante el demodulador la lectura directa de los 8 canales.
- Aumentar la velocidad de transmisión de la UART y del módulo de transmisión por Bluetooth, para ello habrá que reprogramar y adaptar el protocolo maestro-esclavo utilizado.
- Profundizar en el manejo de la emisora de radiofrecuencia para un aumento de la pericia del usuario a la hora de manejar el vuelo de un dron
- Mejora de un control de estabilización más afinado respecto a la corrección de guiñada por parte del Servo trasero para que responda con mayor velocidad y precisión.

7.4 Pruebas en Tierra

Se han realizado múltiples ensayos con el tricóptero, sin hélices por motivos de seguridad, para probar múltiples cosas y su funcionamiento de cara a la puesta en punto para las pruebas en vuelo.

Telemetría

Para empezar que el módulo de transmisión de telemetría por Bluetooth recibía los datos de la tarjeta de control. Esta innovación respecto al año pasado ha resultado ser un completo éxito, aunque mejorable. Por culpa de la velocidad de transmisión 57600 de *baud rate* en los *scopes* se apreciaban picos sin aparente explicación alternativa por lo que habría de tratar de aumentar ésta velocidad. También se experimentó que cuando se estaba trabajando con la otra tarjeta OpenPilot de vez en cuando se cruzaban las señales y se recibía lo que el otro dron enviaba. Por otro lado, el envío de datos para la modificación de los parámetros de control en vuelo ha resultado poder hacerse correctamente.

Emisora

Una vez solventada la dificultad presentada por el PPM y habiéndose hecho la lectura directa de los cuatro canales, se probó si la señal que se recibía de la emisora estaba en el intervalo entre los 1000µs y los 2000µs. Los resultados fueron que realmente el margen de variación iba entre prácticamente los 900ms hasta un poco más allá de los 2000ms, estos datos han sido importantes a la hora de desarrollar correctamente la máquina de estados.

Control

Probablemente uno de los puntos más importantes antes de las pruebas en vuelo. Para este ensayo ha sido necesario cambiar los pesos en la medida de los acelerómetros y giróscopos. Para pruebas en tierra el peso ha de recaer principalmente en los giróscopos puesto que son los que actúan prioritariamente.

Una vez realizadas las preparaciones pertinentes se dispuso a probar el control de estabilidad diseñado. Monitoreando los ángulos estimados se comprobó que los elementos del tricóptero, sus rotores y Servo, reaccionaban correctamente dependiendo de los ángulos que se recibían por UART.

7.5 Pruebas en Vuelo

Una vez realizadas las pruebas en tierra con éxito y habiendo obtenido resultados satisfactorios en la simulación se procedió a realizar las pruebas de vuelo manual. Este punto ha resultado ser el más difícil y no se ha llegado a conseguir en su totalidad.
En este proyecto podemos distinguir entre dos principales fases:

Primera Fase

Sólo se consiguió alzar el vuelo con el tricóptero gracias a unas cuerdas debido a que la combinación de la potencia de los motores junto con las hélices hacía que la fuerza de empuje fuera desmesurada y por seguridad hubo que sujetarlo. Viendo que esto para vuelo en interiores era completamente inviable se procedió a un cambio de hélices y para ello fue necesario tener que recalcular la caracterización de la fuerza de empuje y el momento de inercia necesario.

Figura 7.1 Hélices tripala

Segunda Fase

Con las nuevas hélices y todo recalculado para éstas, se dispuso a probar el vuelo manual. La principal complicación detectada ha sido que el control de guiñada no respondía con suficiente rapidez y hacía que no se pudiera manejar correctamente el tricóptero cuando alzaba el vuelo.

También se ha observado que el volar drones como tricópteros no es tan intuitivo como los cuadricópteros y es necesario cierta pericia y experiencia al respecto pues hay que corregir con la emisora ciertas perturbaciones normales que se experimentan en la fase de vuelo.

Figura 7.2 Tricóptero con las hélices bipala

Capítulo 8

Referencias

[1] «Battery University,» [En línea]. Available: http://batteryuniversity.com/learn/article/lithium_based_batteries.

[2] A. R. L. B. R. C. L. C. J. P. E. Rocon, «Un nuevo sensor para medida del movimiento tembloroso basado en giroscopios,» de *XXIV Jornadas de Automática de León*, 2003.

[3] «Sensor Medidor de Aceleración,» [En línea]. Available: http://bibing.us.es/proyectos/abreproy/11638/fichero/Capitulo+4.pdf.

[4] B. W. Evans, Arduino Notebook: A Beginner's Reference Written, 2007.

[5] I. M. f. FRSky_D4R_II, FRSky.

[6] R. Schematics, OpenPilot Revolution.

[7] J. V. a. R. Yerraballi, «Chapter 11: Serial Interfacing,» de *Embedded Systems - Shape The World*.

[8] T. 9. U. Manual.

[9] P. J-C. Samin, Symbolic Modeling of Multibody Systems.

[10] A. N. R. S. S. Bouabdallah, «PID vs LQ control techniques applied to an indoor micro quadrotor,» Swiss Federal Institute of Technology, 2004.

[11] «Aerodynamics in Racing Multirotors,» [En línea]. Available: http://flitetest.com/articles/aerodynamics-in-racing-multirotors.

[12] K.-J. Barsk, Model Predictive Control of a Tricopter, 2012.

[13] «Elementos de Electrónica Servomotor,» [En línea]. Available: http://www.info-ab.uclm.es/labelec/solar/electronica/elementos/servomotor.htm.

[14] M. R. R. M. H. N. S. S. H. Bolandi, «Atitude Control of a Quadcopter with Optimized PID Controller,» *Intelligent Control and Automation,* vol. 4, nº 3, pp. 335-342, 2013.

[15] J. C. Adams, An Interpolation Approach to Optimal Trajectory Planning for Helicopter, Naval Postgraduate School, June 2012.

[16] H. Voos, «Nonlinear Control of a Quadrotor Micro-UAV using Feedback-Linearization,» de *2009 IEEE International Conference on Mechatronics*, 2009.

[17] H.-S. K. K.-F. H. R. Hercus, «Control of an unmanned aerial vehicle using a neuronal network,» *2013 IEEE Symposium on Computational Intelligence, Cognitive Algorithms, Mind, and Brain (CCMB),* pp. 73-79, 2013.

[18] L. M. P. C. I. M.-B. M. A. Olivares-Mendez, «Cross-Entropy optimization for scaling factors of a fuzzy controller: a see-and-avoid approach for unmanned aerial systems,» *Journal of Intelligent & Robotic Systems,* vol. 69, nº 1-4, pp. 189-205, 2013.

[19] A. M. S. L. Alessandro Benini, «An IMU/UWB/Vision-based Extended Kalman Filter for Mini-UAV Localization in Indoor Environment using 802.15.4a Wireless Sensor Network,» *Journal of Intelligent & Robotic Systems,* vol. 70, pp. 461-476, Abril 2013.

[20] J. R. A. H. David Gaydou, «Filtro Complementario para estimación de actitud aplicado al controlador embebido de un cuatrirrotor,» Córdoba, Universidad Tecnología Nacional.

[21] P. C. ,. R. M. ,. J. K. T. H. M. Euston, «A complementary filter for attitude estimation of a fixed-wing UAV,» de *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst*, 2008.

[22] D. E. S. Michael A. Henson, Feedback Linearizing Control, Louisiana: Louisiana State University.

[23] «Copter,» [En línea]. Available: http://copter.ardupilot.com/wiki/common-autopilots/common-apm25-and-26-overview/#powering_the_apm_2526_board.

[24] «Hobbyking,» [En línea]. Available: http://www.hobbyking.com/hobbyking/store/__89563__OpenPilot_CC3D_Revolution_Revo_32bit_F4_Based_Flight_Controller_w_Integrated_433Mhz_OPLink.html.

[25] E. G.-C. Triquell, Optimización de funciones de DSP para procesador con instrucciones de arimética completa, Universidad Complutense de Madrid, Facultad de informática, 2008.

[26] «Waijung,» [En línea]. Available: http://waijung.aimagin.com/index.htm?command_lines.htm.

[27] K.-L. S. H. T. Jie-Tong Zou, «The modeling and implementation of tri-rotor flying robot,» January 2012.

[28] P. C. R. M. Paul Pounds, «Modelling and control of a quad-rotor robot,» Australian National University, 2006.

[29] L. Sevilla, «Modelado y control de un cuadricóptero,» Universidad Pontificia de Comillas, 2014.

[30] C. A. P. C. R. A. Bemporad, «Hierarchical and hybrid model predictive control of quadcopter air vehicles,» vol. 3. Parte 1, pp. 14-19, 2009.

[31] G. E. Moore, «Cramming more components onto integrated circuits,» *Electronics,* vol. 38, nº 8, 19 Abril 1965.

[32] D. M. A. K. B. K. V. K. Caitlin Powers, Influence of Aerodynamics and Proximity Effects in Quadrotor Flight, Springer, 2013.

[33] L. F. School, «http://www.langleyflyingschool.com/,» [En línea]. Available: http://www.langleyflyingschool.com/Pages/Aerodynamics%20and%20Theory%20of%20Flight.html#AERODYNAMICS%20AND%20THEORY%20OF%20FLIGHT.

[34] W. Johnson, Helicopter Theory, New York: Dover Publications, Inc., 1980.

[35] G. M. C. J. T. Rajnarayan D. G. Waslander S. L. Dostal D. Jang J. S. Hoffmann, «The stanford testbed of autonomous rotorcraft for multi agent control,» de *Digital Avionics Systems Conference*, 2004.

[36] J. F. W. A. K. C. Ian D. Cowling, «Optimal trajectory planning and LQR,» de *Proc. UKACC Int. Conf. Control*, 2006.

[37] F.-D. X. H. R. A. H. M. Robert F. Hartley, «Development and Flight Testing of a Model Based Autopilot Library for a Low Cost Unmanned Aerial System,» de *AIAA Guidance, Navigation, and Control (GNC) Conference*, 2013.

[38] D. M. Botella, Control de un Cuadricóptero para vuelos autónomos en exteriores, Universidad Pontificia de Comillas, Junio 2015.

[39] J. M. Olondo, Control de un Cuadricóptero para vuelos autónomos en interiores, Universidad Pontificia de Comillas, Junio 2015.

Anexo A

En este anexo se recoge el código del fichero maestro ***Config_SIMULATOR.m***. Este archivo contiene lo necesario para inicializar todos los *scripts* que llevan a la puesta en punto de tanto la implantación por medio de la placa OpenPilot como del propio simulador.

```
clc
clear
format short e

% Gravity (m/s^2)
Gravity=9.81;

%% Sampling times
% Sampling time for RC transmitter
ts_PPM=20e-3;
% Sampling time for IMU and STATE ESTIMATION
ts_IMU=5e-3;
% Sampling time for UART
ts_UART=20e-3;
% Sampling time for ATTTITUDE CONTROL
ts_ATT_CONTROL=5e-3;
% Sampling time for NAVIGATION CONTROL
ts_NAV_CONTROL=20e-3;

%% SIMULATION PARAMETERS
% Simulation final time
tfin=600;

%% HARDWARE COMPONENTS
% IMU
run('..\HARDWARE_COMPONENTS\config_MPU60X0')
% COMPASS
run('..\HARDWARE_COMPONENTS\config_HMC5883L')



%% Model parameters
% ModelParam_QuadAntonio
% ModelParam_QuadNestor
ModelParam_Tricopter

%% Bus definitions
BusDefinitionUAV;
% BusDefinitionMotor;
BusDefinitionCommand;
BusDefinitionSensors;
BusDefinitionEnvironment
BusDefinitionStates;
```

```
%%  Variants Conditions
% Add enum structure for the Variants

Simulink.defineIntEnumType('Variants',{'Command','Vehicle','Environmen
t','Actuator',...

'Visualization','RigidBody','AttitudeEstimator','NavigationEstimator'}
,[0;0;0;0;0;0;0;0]);
% Command: 0-Signal Builder / 1-Joystick / 2-Data
Variants.Command = 0;
% Enviorement: 0-Constant / 1-Variable
Variants.Environment = 0;
% Visualization: 0-Scopes / 1-WorkSpace / 2-Flightgear / 3-MAVLink
Variants.Visualization = 3;
% Actuators: 0-BLDC / 1-Instantaneous / 2-First order
% 0 - BLDC dynamics: motor model and rotor gyroscopic torques (slow
simulation)
% 1 - Instantaneous actuator without rotor gyroscopic torques (fast
simulation)
% 2 - First order actuator without rotor gyroscopic torques (slow
simulation)
Variants.Actuator = 1;
% Rigid Body Dynamics : 0-Aerospace Blockset Euler / 1-Aerospace
Blockset Quaternions / 2-Euler / 3-Quaternion
Variants.RigidBody = 0;
% Vehicle: 0-Quadcopter / 1-Tricopter / 2-Coaxial helicopter / 3-
Hybrid RVJET
% Variants.Vehicle = Defined in ModelParam_* file
% Attitude estimator
% 0 - Divided Difference Filter NO FUNCIONA
% 1 - EKF desacoplado no lineal OK
% 2 - EKF desacoplado lineal  OK
% 3 - EKF (Extended Kalman Filter) OK
% 4 - Filtro complementario no lineal  OK GOLD
% 5 - Multiplicative extended Kalman Filter (MEKF) OK SILVER
% 6 - Quaternion complementary Filter OK
Variants.AttitudeEstimator = 4;
% Navigation Estimator: 0-GPS / 1-Infrared
Variants.NavigationEstimator = 1;



%% Initial values
% Initial date
InitialValues = struct('Date', [2015 1 1 0 0 0]);
%  Madrid GPS coordinates
InitialValues.PosLLA = [40.4167754 -3.7037901999999576 646.4];
% NED frame coordinates
InitialValues.PosNED = [1 1 0];
% Body velocity
InitialValues.VelBody = [0 0 0];
% NED Velocity
```

```
InitialValues.VelNED = [0 0 0];
% Euler angles
InitialValues.Euler = [0 0 0];
% Quaternions
InitialValues.Quaternion = [1 0 0 0];
% Body Angular Rates
InitialValues.AngRates = [0 0 0];
% Earth Reference frame
InitialValues.Greenwich = 0;
% Throttle para compensar el peso
InitialValues.StationaryThrottle =
interp1(Motor.ThrustData,Motor.PWMData_pu,UAV.Mass*Gravity/4);

%% Attitude Control Design
Control=PIDAttitudeControl(UAV,Motor,Variants,ts_ATT_CONTROL,Gravity);
param_ATT_CONTROL = [ Control.K_roll  Control.invTi_roll
Control.Td_roll  Control.b_roll  ...
    Control.K_pitch Control.invTi_pitch Control.Td_pitch
Control.b_pitch ...
    Control.K_yaw   Control.invTi_yaw   Control.Td_yaw
Control.b_yaw]';
% Maximun angle for RC channel (rad)
Control.ang_max = 15*pi/180;
% Maximun yaw rate for RC channel (rad/s)
Control.yaw_rate_max = 35*pi/180;

%% Navigation Control Design
Navigation=PIDNavigationControl(ts_NAV_CONTROL);
param_NAV_CONTROL = [ Navigation.K_x  Navigation.invTi_x
Navigation.Td_x  Navigation.b_x  Navigation.N_x  ...
    Navigation.K_y  Navigation.invTi_y  Navigation.Td_y
Navigation.b_y  Navigation.N_y  ...
    Navigation.K_z  Navigation.invTi_z  Navigation.Td_z
Navigation.b_z  Navigation.N_z]';

%% Other parameters
% Calibration filter for IMU
alfa_CAL_IMU=0.9999;
% Calibration filter for POSITION
alfa_CAL_POS=0.99;



%% SENSOR DYNAMICS PARAMETERS
SensorParameters

%% MAVLink
% Magic or seed byte for version checking
MAVLINK_MESSAGE_CRCS =uint8([...
   50  124 137   0 237 217 104 119   0   0 ...
    0   89   0   0   0   0   0   0   0   0 ...
```

```
  214   159 220 168   24   23 170 144   67 115 ...
   39   246 185 104 237 244 222 212    9 254 ...
  230    28  28 132 221 232  11 153   41  39 ...
  214   223 141  33  15   3 100  24 239 238 ...
   30   200 183   0 130   0 148  21   0  52 ...
  124     0   0   0  20   0 152 143   0   0 ...
    0     0   0   0   0   0   0   0   0 231 ...
  183    63  54   0   0   0   0   0   0   0 ...
  175   102 158 208  56   0   0   0   0   0 ...
    0     0   0   0   0   0   0   0   0   0 ...
    0     0   0   0   0   0   0   0   0   0 ...
    0     0   0   0   0   0   0   0   0   0 ...
    0     0   0   0   0   0   0   0   0   0 ...
    0     0   0   0   0   0   0   0   0   0 ...
    0     0   0   0   0   0   0   0   0   0 ...
    0     0   0   0   0   0   0   0   0   0 ...
    0     0   0   0   0   0   0   0   0   0 ...
    0     0   0   0   0   0   0   0   0   0 ...
    0     0   0   0   0   0   0   0   0   0 ...
    0     0   0   0   0   0   0   0   0   0 ...
    0     0   0   0   0   0   0   0   0   0 ...
    0     0   0   0   0   0   0   0   0   0 ...
    0     0   0   0   0   0   0   0   0   0 ...
   49   170  44  83  46   0 ]);

return
```

Anexo B

En este anexo se recoge el código que hay incluido en el *script* ***ModelParam_Tricopter.m***. En él se encuentran recogidas todas las especificaciones físicas del tricóptero de este proyecto que se usarán tanto en el simulador como en el control.

```
%%%%% Airframe %%%%%%
% BATTERY
BATTERY_CELLS=3;
BATTERY_NOMINAL_VOLTAGE=3.7*BATTERY_CELLS;
BATTERY_MAXIMUM_VOLTAGE=4.2*BATTERY_CELLS;
BATTERY_MINIMUM_VOLTAGE=3.5*BATTERY_CELLS;

% VEHICLE
% Vehicle: 0-Quadcopter / 1-Tricopter / 2-Coaxial helicopter / 3-
Hybrid RVJET
Variants.Vehicle = 1;

%%%%% Airframe %%%%%%
% Mass (kg)
UAV = struct('Frame',0);
% Mass (kg)
UAV.Mass = 0.4568;
% Arm Length for roll and pitch torques (m)
UAV.RollArmLength=0.13;
UAV.PitchArmLength_Front=0.071166;
UAV.PitchArmLength_Rear=0.1423;
UAV.HeightArmLength=0.05;

% Inertia matrix (kg.m^2)
UAV.Inertia = diag([7.83e-4  7.85e-4  1.4e-3 ]);
% Quaternion normalization gain
UAV.QuatGain = 1;

%%%% Brushless DC motor + propeller %%%%%
% Electric  resistance (ohm)
Motor = struct('Resistance',0.405);
% KV (rpm/V)
Motor.KV = 2150;
% Inertia motor (kg.m^2)
Motor.Inertia = 104e-8;
% Motor propeller diameter (m)
Motor.PropellerDiameter = 5*2.54/100;
% Drag Calculation
Motor.DragCoeff = [.1 .1 .3];
Motor.DragSection = 3*pi*(Motor.PropellerDiameter/2)^2;
% Thrust - Drag Torque Ratio
Motor.ThrustDragTorqueRatio = 100;
% Thrust and drag torque lookup table
```

```
Motor.ThrustData = [0 0.1 21  98 160 218 297 399 437 515
515.1]/1000*Gravity;
Motor.ServoAngleData = linspace(-90,90,11);
Motor.PWMData = [1000 1100 1200 1300 1400 1500 1600 1700 1800 1900
2000];
Motor.PWMData_pu = (Motor.PWMData-
Motor.PWMData(1))/(Motor.PWMData(end)-Motor.PWMData(1));
Motor.DragTorqueData = Motor.ThrustData/Motor.ThrustDragTorqueRatio;
Motor.AngularRotationData =
BATTERY_NOMINAL_VOLTAGE*Motor.KV*pi/30*Motor.PWMData_pu;
Motor.VoltageData = BATTERY_NOMINAL_VOLTAGE*Motor.PWMData_pu;

% Motor Time Constant
% Km/Rm/(Jm*s+Dm)/(1+Km^2/Rm/(Jm*s+Dm))
% Km/(Jm*Rm*s+Rm*Dm+Km^2) = Km/(Rm*Dm+Km^2)/(Jm*Rm/(Rm*Dm+Km^2)*s+1)
Motor.Gain = 30/Motor.KV/pi/(Motor.Resistance*1e-
9+(30/Motor.KV/pi)^2);
Motor.TimeConstant =
Motor.Inertia*Motor.Resistance/(Motor.Resistance*1e-
9+(30/Motor.KV/pi)^2);

return
```

Anexo C

En este anexo se recoge el código incluido en la *Matlab function* del modelo de Simulink del **Control de Estabilización** utilizado en este proyecto. En éste se puede observar cómo según se ha elegido el tipo de dron (cuadricóptero o tricóptero) se habilitará un mixeador u otro. También aparecen las ecuaciones de diseño del control.

```
function Control =
PIDAttitudeControl(UAV,Motor,Variants,ts_ATT_CONTROL,Gravity)

% Sampling time (s)
Control  =  struct('tc',ts_ATT_CONTROL);
% Natural frequency (rad/s)
Control.NaturalFrequency = 8;
Control.NaturalFrequencyYaw = 1.5;
% Damping
Control.Damping = 1;
Control.DampingYaw = 1;

%_____________________________________________________________________
_____
%     ATTITUDE PID CONTROL DESIGN
%_____________________________________________________________________
_____
% State vector (body reference frame)
% X = [Phi Theta Shi Phi_dot Theta_dot Shi_dot]'
% U = [u1 u2 u3 u4]'
% Thrust_base: T_base = m*g
% Control variables: quad + configuration
% u1=(T1+T2+T3+T4)/T_base;          % Thrust (-)
% u2=(T2-T4)/T_base;                % Roll (+)
% u3=(T1-T3)/T_base;                % Pitch (+)
% u4=(T1-T2+T3-T4)/T_base;          % Yaw (+)
% Control variables: quad x configuration
% u1=(T1+T2+T3+T4)/T_base;          % Thrust (-)
% u2=(T1+T2-T3-T4)/T_base;          % Roll (+)
% u3=(T1-T2-T3+T4)/T_base;          % Pitch (+)
% u4=(T1-T2+T3-T4)/T_base;          % Yaw (+)
% Control variables: tricopter configuration
% u1=(T1+T2+T3*cos(th_s))/T_base;
% Thrust (-)
% u2=(T1-T2+T3*sin(th_s)*L_height/L_roll)/T_base;
% Roll (+)
% u3=(T1+T2-T3*cos(th_s)*L_pitch_r/L_pitch_f-
d/b*T3*sin(th_s)/L_pitch_f)/T_base;    % Pitch (+)
% u4=(T1-T2+T3*cos(th_servo)-b/d*L_pitch_r*T3*sin(th_s))/T_base;
% Yaw (+)

% Modelo en angulos de Euler
```

```
% wxyz_dot without gyroscopic effects in the motors
% wxyz_dot = inv_matI*(matL*u - wxyz x (matI*wxyz))
Control.matI = UAV.Inertia;
m = UAV.Mass;
g = Gravity;
T_base = m*g;
L_roll = UAV.RollArmLength;
L_pitch_f = UAV.PitchArmLength_Front;
L_pitch_r = UAV.PitchArmLength_Rear;
ratio_TDT = Motor.ThrustDragTorqueRatio;
matL = diag([T_base*L_roll T_base*L_pitch_f  T_base/ratio_TDT]);

% wxyz = matEuler*angEuler_dot
% matEuler = [ 1     0         -sin(Theta)
%              0  cos(Phi) sin(Phi)*cos(Theta)
%              0 -sin(Phi) cos(Phi)*cos(Theta) ]
% wxyz_dot = matEuler_dot*ang_Euler_dot + matEuler*ang_Euler_dot2
% matEuler_dot = [ 0          0                           -
cos(Theta)*Theta_dot
%                  0 -sin(Phi)*Phi_dot   cos(Phi)*cos(Theta)*Phi_dot-
sin(Phi)*sin(Theta)*Theta_dot
%                  0 -cos(Phi)*Phi_dot  -sin(Phi)*cos(Theta)*Phi_dot-
cos(Phi)*sin(Theta)*Theta_dot ]
% ang_Euler_dot2 = inv_matEuler*(wxyz_dot -
matEuler_dot*ang_Euler_dot) = u_p
% inv_matEuler = [ 1  sin(Phi)*tan(Theta) cos(Phi)*tan(Theta)
%                  0      cos(Phi)              -sin(Phi)
%                  0  sin(Phi)/cos(Theta) cos(Phi)/cos(Theta) ]
% u_p = inv_matEuler*(inv_matI*(matL*u - wxyz x (matI*wxyz)) -
matEuler_dot*ang_Euler_dot)
% u_p = inv_matEuler*(inv_matI*(matL*u - wxyz x (matI*wxyz)) -
matEuler_dot*inv_matEuler*wxyz)
% Control variable computation
% u = matC1*u_p + matC2*(wxyz x (matI*wxyz)) + matC3*wxyz
% Matrices for control variable computation
% matC1 = inv(inv_matEuler*inv_matI*matL) = inv_matL*matI*mat_Euler
% matC2 = inv_matL*matI
% matC3 = inv_matL*matI*matEuler_dot*inv_matEuler
Control.inv_matL = inv(matL);

% Control PD
% C(s) = K*(1 + Td*s) = P + D*s
% P = K      =>   K = P
% D = K*Td   =>   Td = D/P
% F(s) = (D*s + P) / (s^2 + D*s + P)
% Control PID
% C(s) = K*(1 + 1/Ti/s + Td*s) = P + I/s + D*s
% P = K      =>   K = P
% I = K/Ti   =>   Ti = P/I
% D = K*Td   =>   Td = D/P
% F(s) = (D*s^2 + P*s + I) / (s^3 + D*s^2 + P*s + I)
```

```
wn = Control.NaturalFrequency;
seta = Control.Damping;
p = -0*wn;
% Polinomio denominador lazo cerrado PD
% (s^2 + 2*seta*wn*s + wn^2)
% Polinomio denominador lazo cerrado PID
% (s^2 + 2*seta*wn*s + wn^2)*(s-p)
% s^3 + (2*seta*wn - p)*s^2 + (wn^2 - 2*seta*wn*p)*s - p*wn^2
% % Coeficientes
a2 = 2*seta*wn - p;
a1 = wn^2 - 2*seta*wn*p;
a0 = -p*wn^2;
% % Parametros del control
P_roll = a1;
I_roll = a0;
D_roll = a2;
P_pitch = a1;
I_pitch = a0;
D_pitch = a2;
Control.K_roll = P_roll;
Control.invTi_roll = I_roll/P_roll;
Control.Td_roll = D_roll/P_roll;
Control.b_roll = 1;
Control.K_pitch = P_pitch;
Control.invTi_pitch = I_pitch/P_pitch;
Control.Td_pitch = D_pitch/P_pitch;
Control.b_pitch = 1;
```

```
wn=Control.NaturalFrequencyYaw;
seta=Control.DampingYaw;
p=-0*wn;
a2 = 2*seta*wn - p;
a1 = wn^2 - 2*seta*wn*p;
a0 = -p*wn^2;
P_yaw = a1;
I_yaw = a0;
D_yaw = a2;
Control.K_yaw = P_yaw;
Control.invTi_yaw = I_yaw/P_yaw;
Control.Td_yaw = D_yaw/P_yaw;
Control.b_yaw = 1;
```

```
switch Variants.Vehicle

    case 0  % Quadcopter

        switch UAV.Frame

            case 1 % quad + configuration
                % Control variables: quad + configuration
                % Ti_pu = Ti/T_base  i = 1,2,3,4
                % [u1 u2 u3 u4]' = matT*[T1_pu T2_pu T3_pu T4_pu]'
                % matT=[1 1 1 1 ; 0 1 0 -1 ; 1 0 -1 0 ; 1 -1 1 -1];
                Control.inv_matT = [
                    0.2500         0    0.5000    0.2500
                    0.2500    0.5000         0   -0.2500
                    0.2500         0   -0.5000    0.2500
                    0.2500   -0.5000         0   -0.2500 ];

            case 2 % quad x configuration
                % Control variables: quad x configuration
                % Ti_pu = Ti/T_base  i = 1,2,3,4
                % [u1 u2 u3 u4]' = matT*[T1_pu T2_pu T3_pu T4_pu]'
                % matT=[1 1 1 1 ; 1 1 -1 -1 ; 1 -1 -1 1 ; 1 -1 1 -1];
                Control.inv_matT = [
                    0.2500    0.2500    0.2500    0.2500
                    0.2500    0.2500   -0.2500   -0.2500
                    0.2500   -0.2500   -0.2500    0.2500
                    0.2500   -0.2500    0.2500   -0.2500 ];

        end

    case 1 % Tricopter
        % Control variables: tricopter configuration
        % Ti_pu = Ti/T_base  i = 1,2,3,4
        % [u1 u2 u3 u4]' = matT*[T1_pu T2_pu T3_pu*cos(th_servo)
T3_pu*sin(th_servo)]'
        L_height=UAV.HeightArmLength;
        Control.inv_matT = inv([
            1     1               1                          0
            1    -1               0                 L_height/L_roll
            1     1     -L_pitch_r/L_pitch_f     -
1/ratio_TDT/L_pitch_f
            1    -1               1              -ratio_TDT*L_pitch_r
]);

end

return
```

Anexo D

A continuación, se detallan los diagramas que conforman el **Filtro Complementario No Lineal** que se ha utilizado para la estimación de ángulos por parte de la tarjeta de control del tricóptero. Así como los bloques de **Calibración de la IMU**.

Filtro Complementario No Lineal

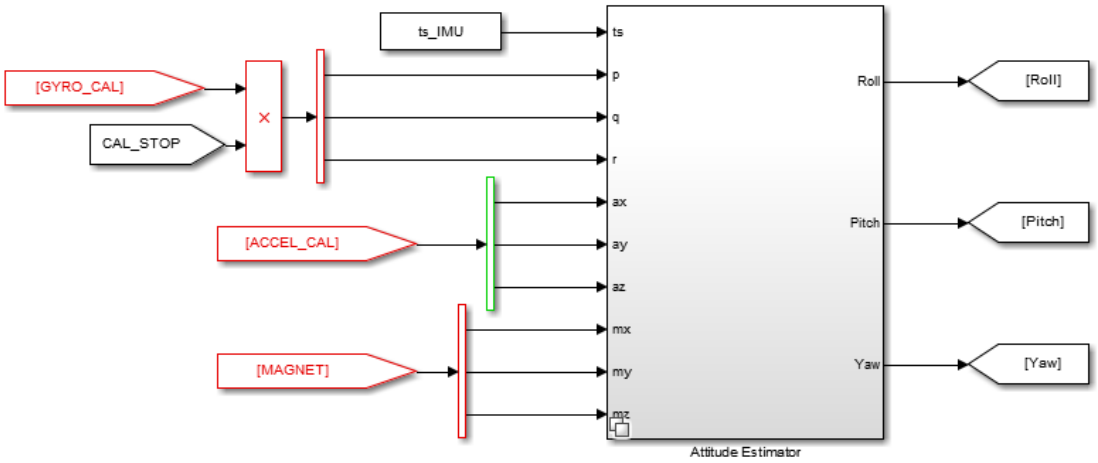
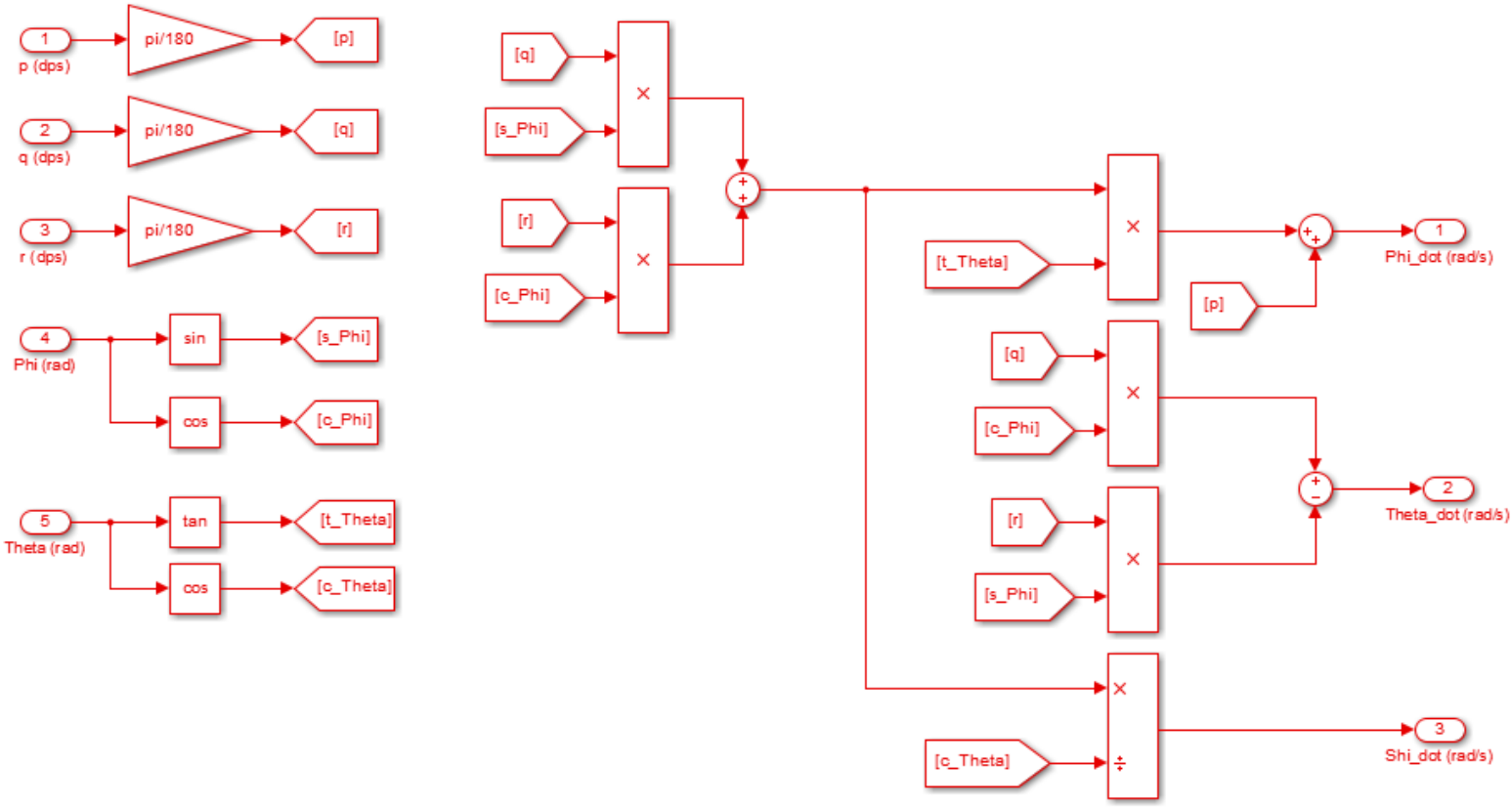
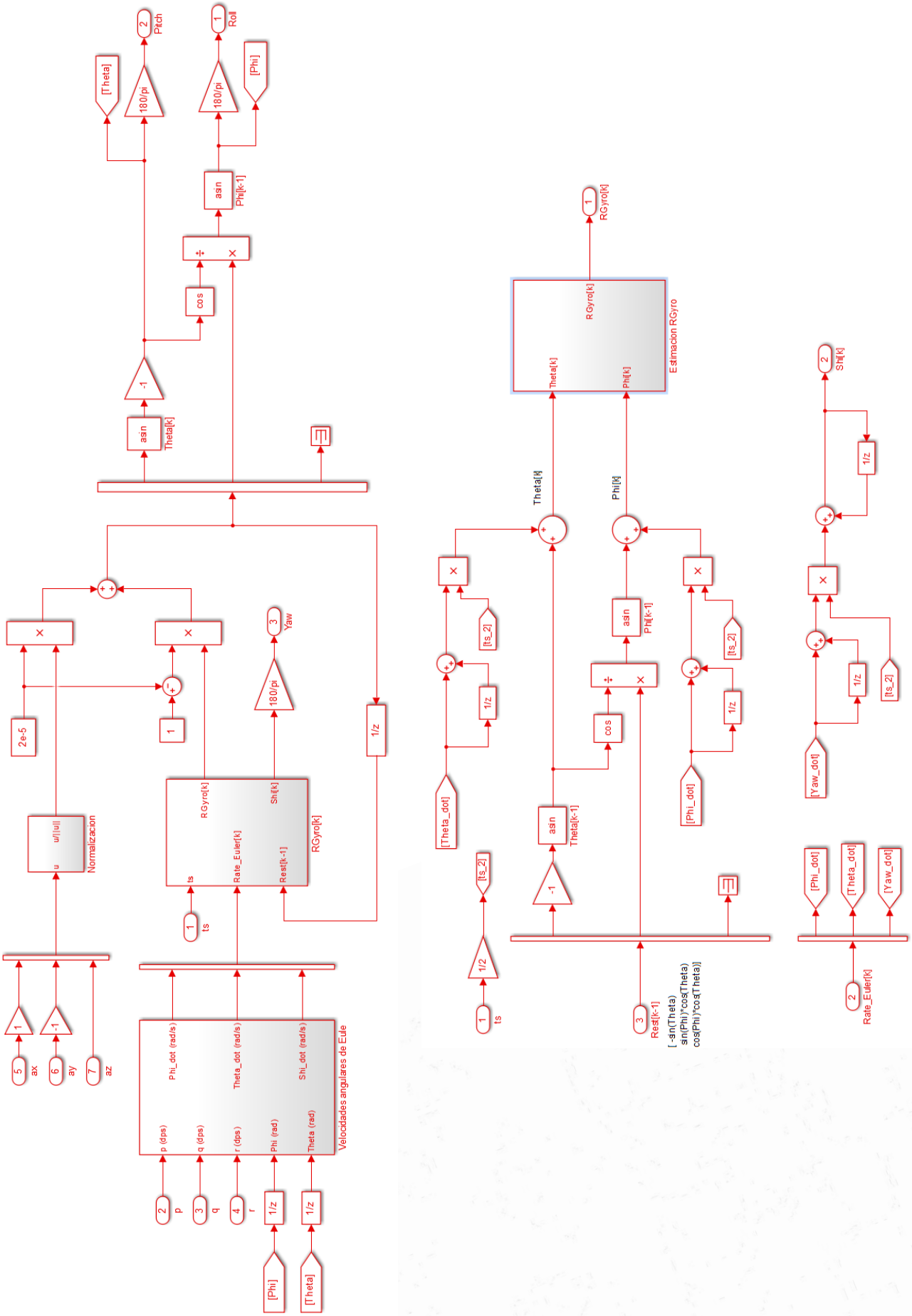


Figura D.1 Nivel alto del Filtro Complementario No Lineal




Pitch
Roll
[Theta]
180/pi
[Phi]
asin
Phi[k-1]
cos
-1
Theta[k]
Yaw
2e-5
1
Normalizacion
u/||u||
RGyro[k]
Sh[k]
Rate_Euler[k]
Rest[k-1]
ts
1/z
Velocidades angulares de Euler
Phi_dot (rad/s)
Theta_dot (rad/s)
Shi_dot (rad/s)
p (dps)
q (dps)
r (dps)
Phi (rad)
Theta (rad)
ax
ay
az
p
q
r
Estimacion RGyro
Theta[k]
Phi[k]
[ts_2]
1/2
[Theta_dot]
[Phi_dot]
[Yaw_dot]
Rest[k-1]
[-sin(Theta)
sin(Phi)*cos(Theta)
cos(Phi)*cos(Theta)]

Calibración de la IMU

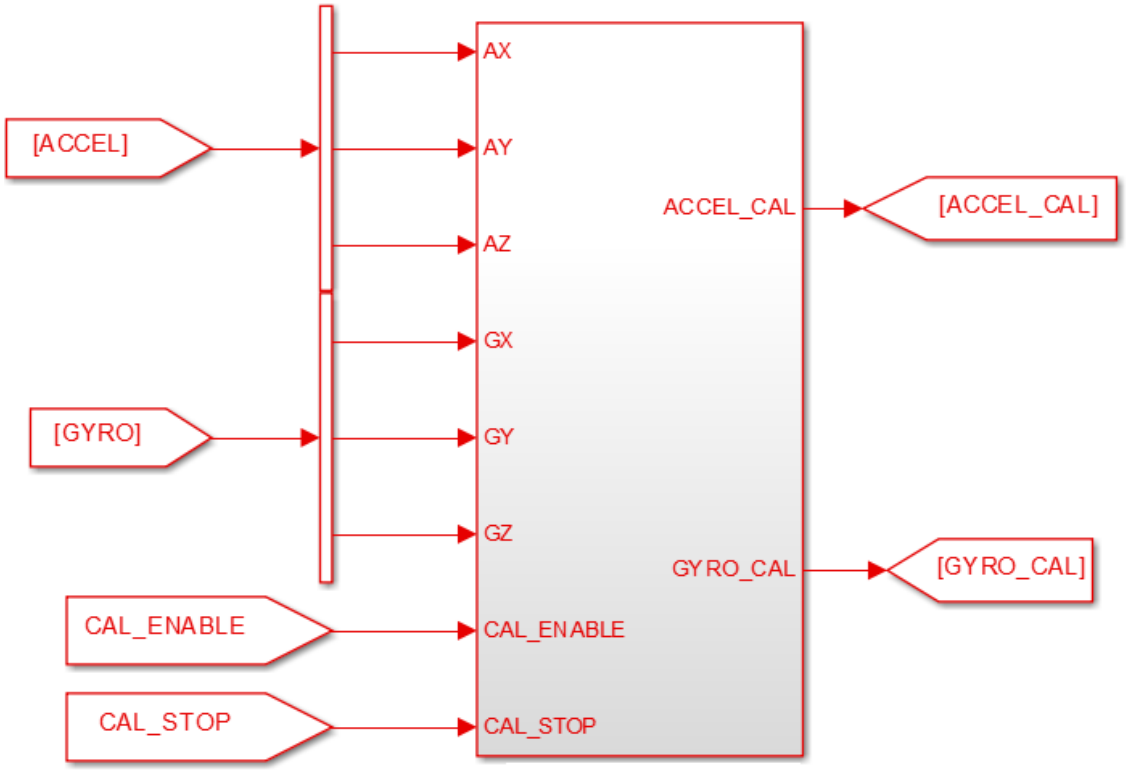
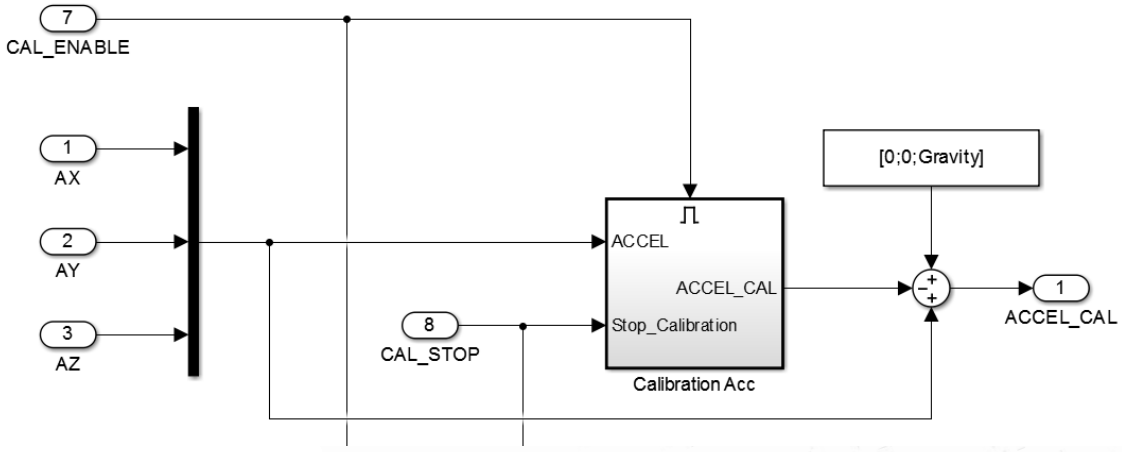
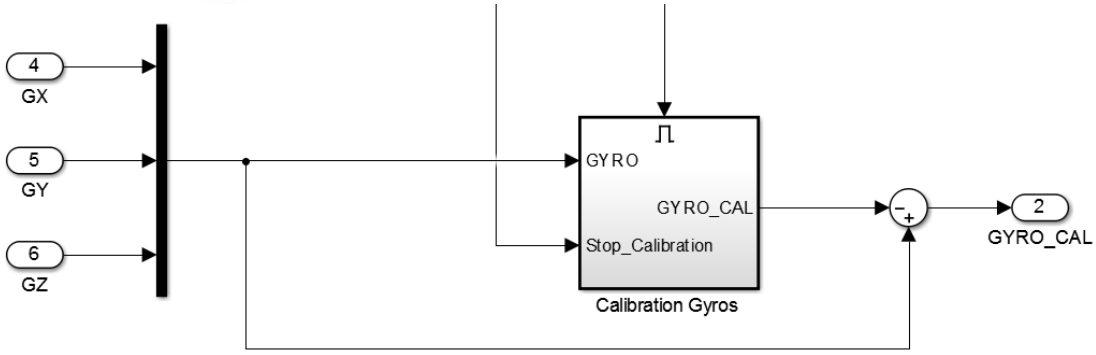
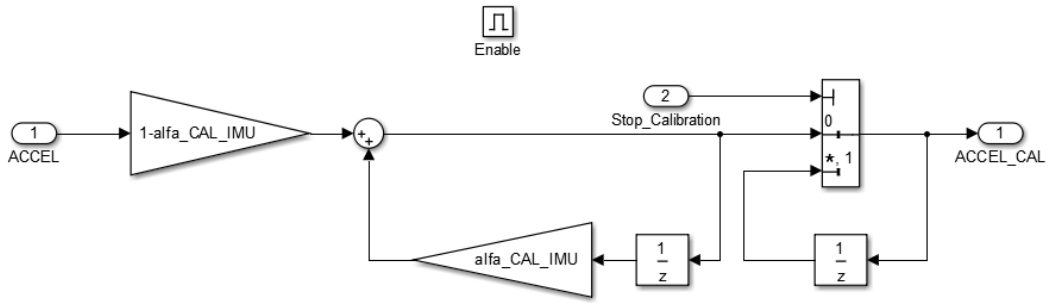


Figura D.2 Calibración de la IMU

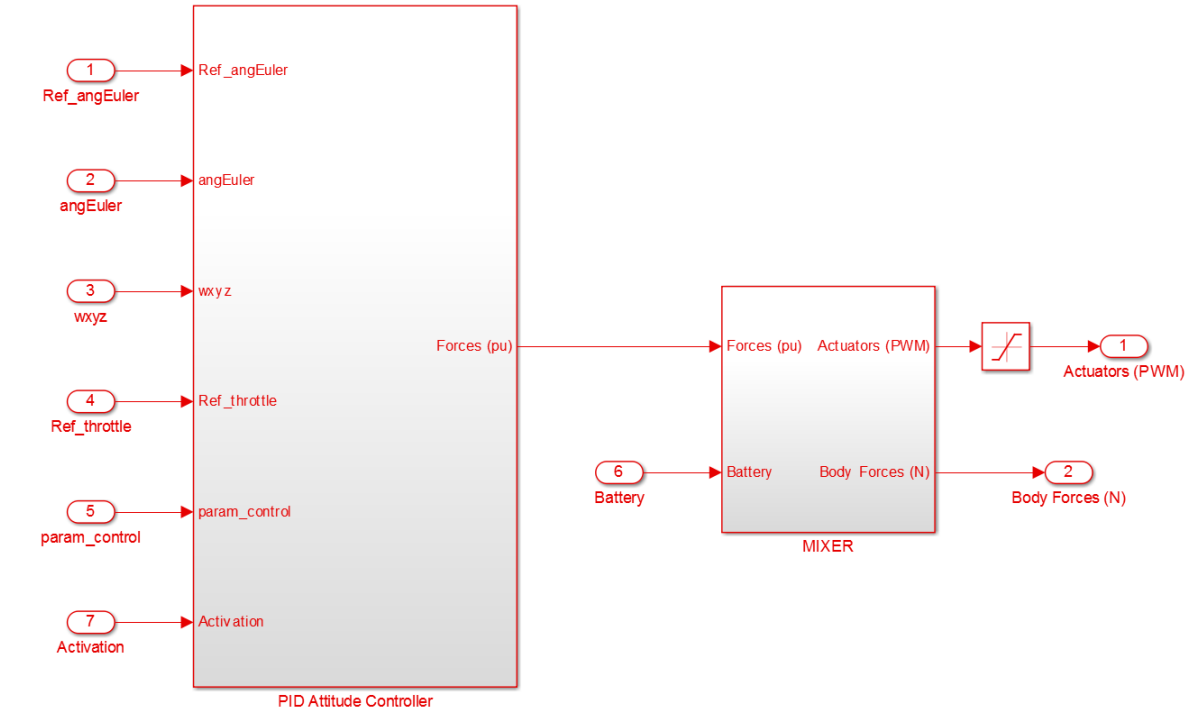


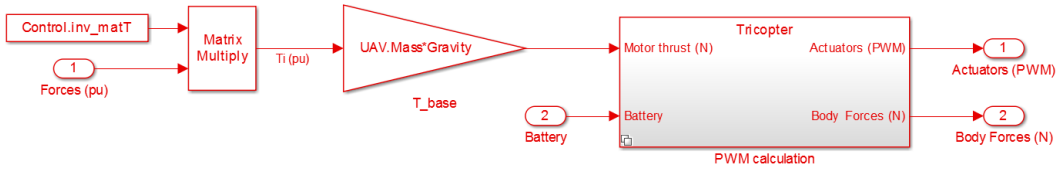

4
GX
5
GY
6
GZ
GYRO
GYRO_CAL
Stop_Calibration
Calibration Gyros
2
GYRO_CAL

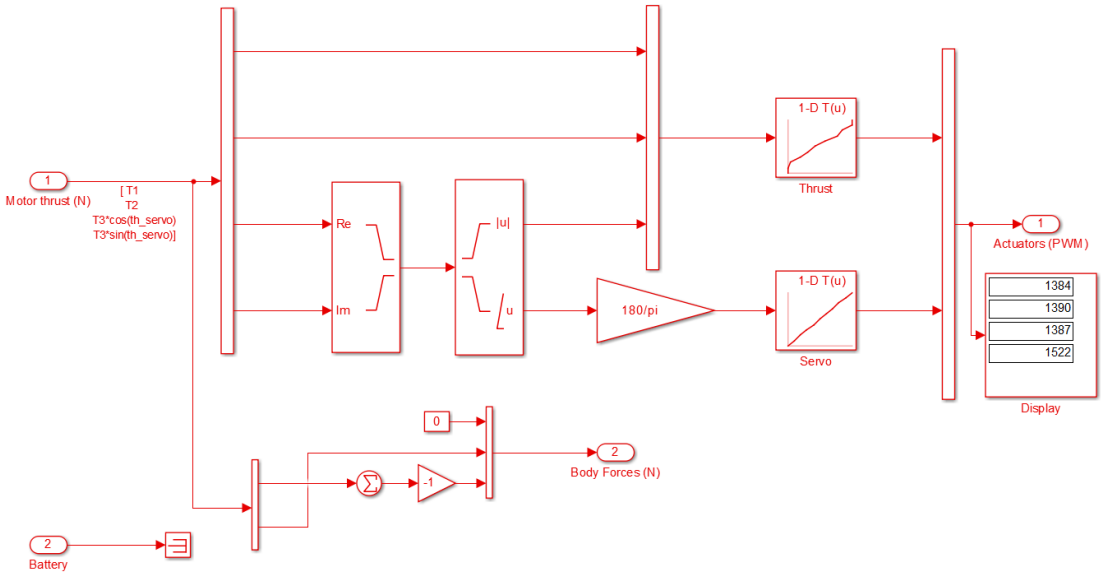

Enable
1
ACCEL
1-alfa_CAL_IMU
2
Stop_Calibration
0
*, 1
1
ACCEL_CAL
alfa_CAL_IMU
1
z
1
z

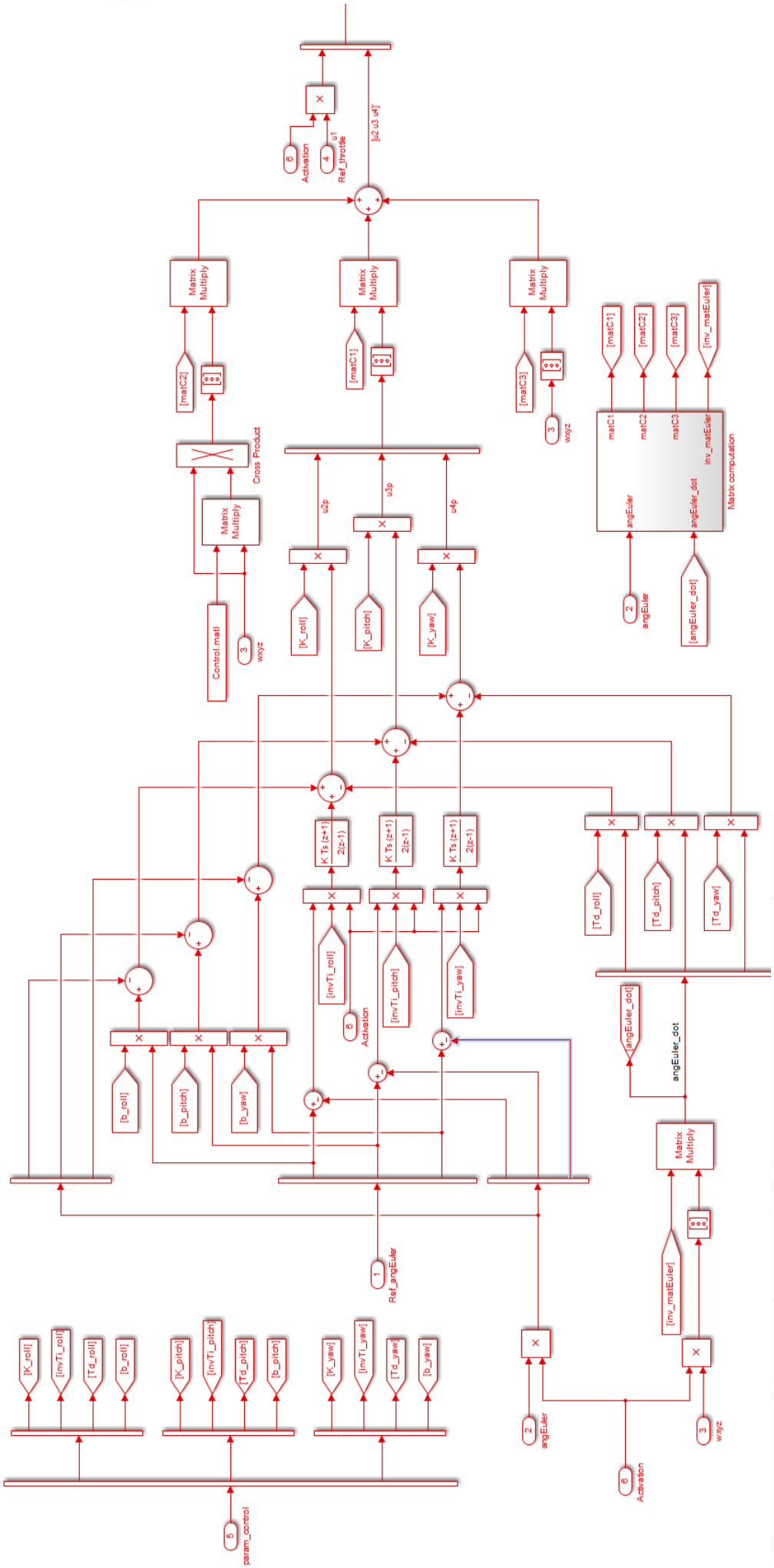
Anexo E

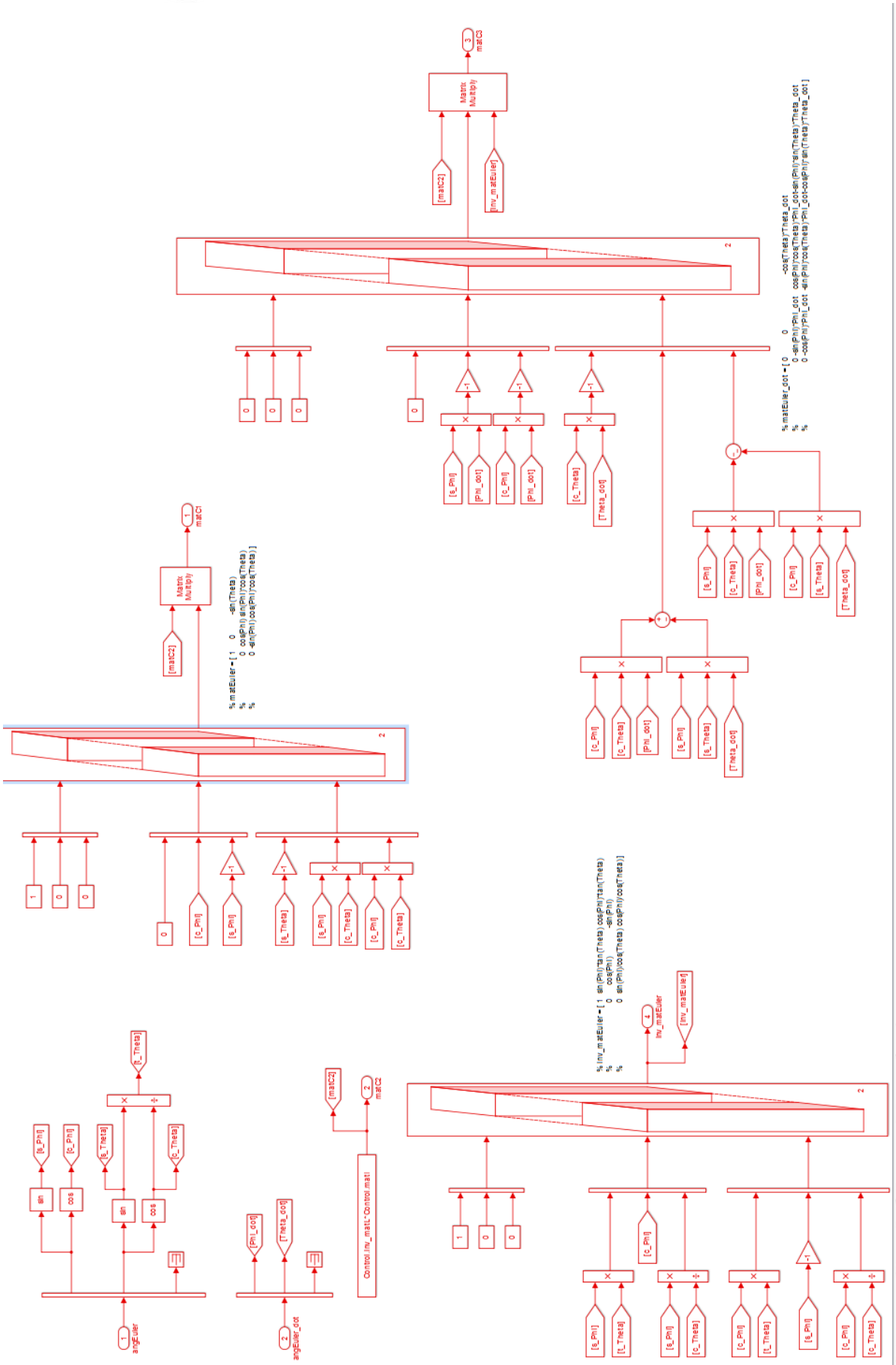
Aquí se muestran los diagramas de simulink del bloque ***Controller.slx*** que contiene tanto el control como el mixeador





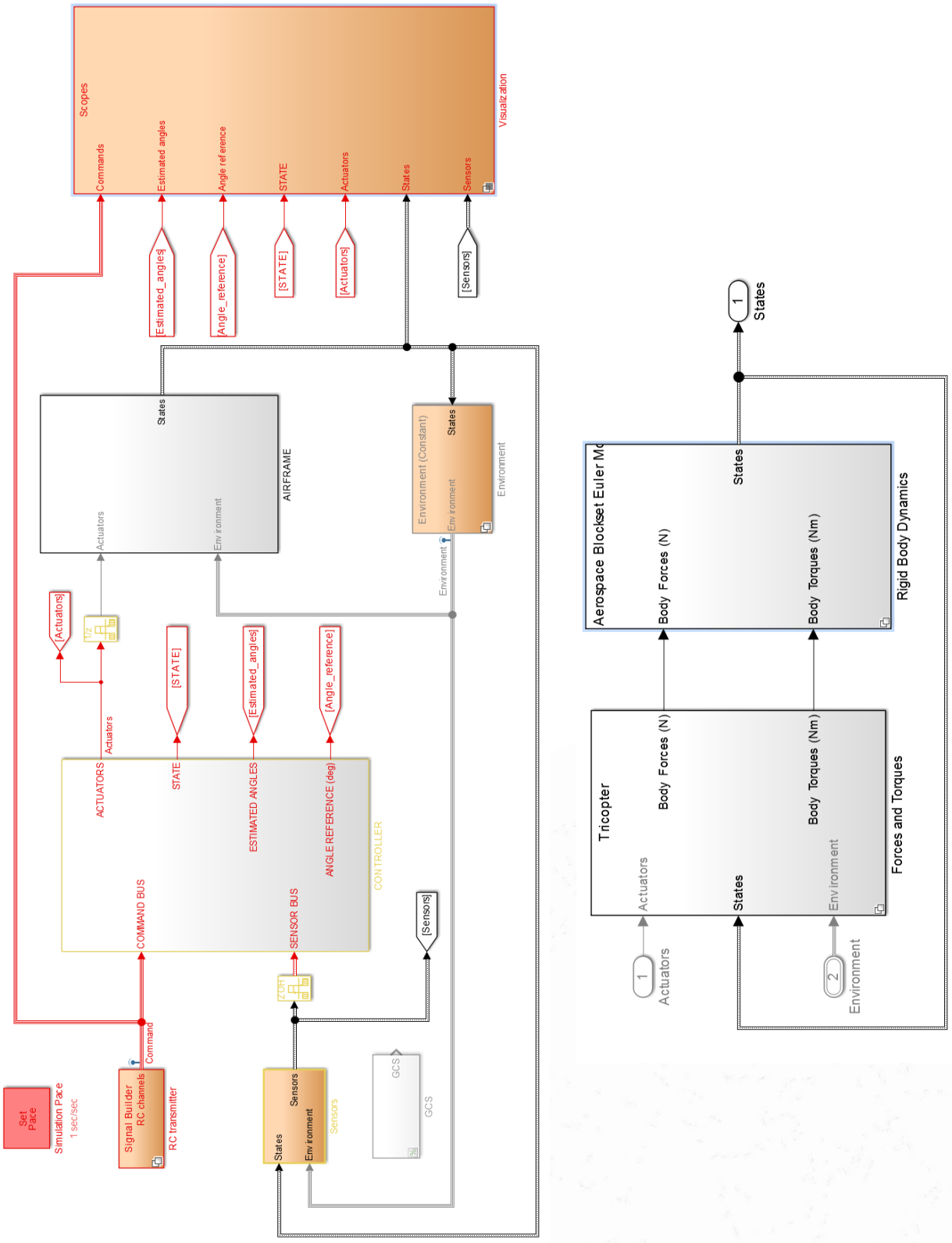


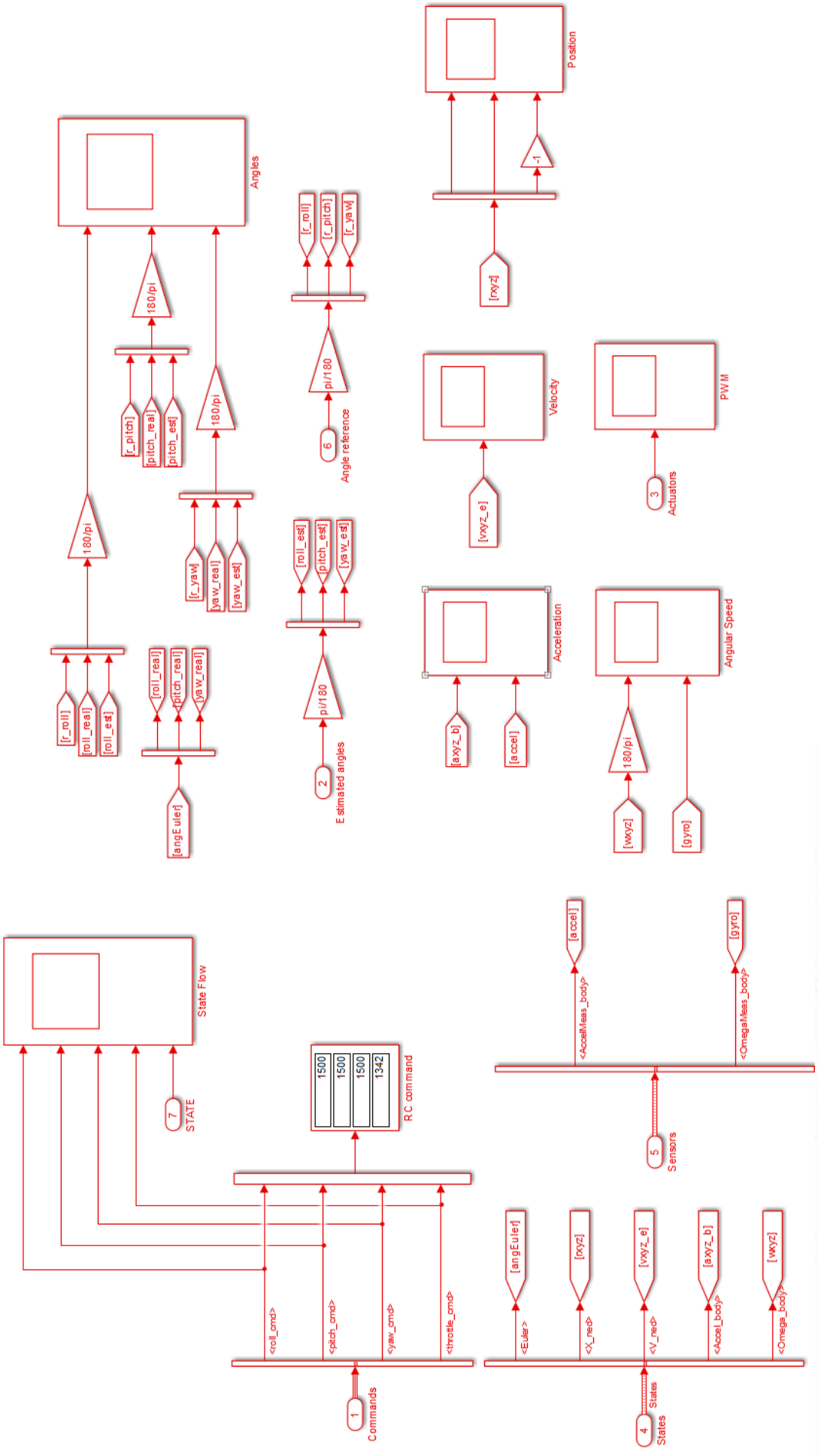

Activation
Ref_throttle
u1
[u2 u3 u4]'
Matrix Multiply
[matC2]
[matC1]
[matC3]
wxyz
Cross Product
Control matl
u2p
u3p
u4p
[K_roll]
[K_pitch]
[K_yaw]
[matC1]
[matC2]
[matC3]
[inv_matEuler]
matC1
matC2
matC3
inv_matEuler
angEuler
angEuler_dot
Matrix computation
angEuler
[angEuler_dot]
K_Ts (z+1)
2(z-1)
[Td_roll]
[Td_pitch]
[Td_yaw]
[invTi_roll]
[invTi_pitch]
[invTi_yaw]
Activation
[b_roll]
[b_pitch]
[b_yaw]
angEuler_dot
Ref_angEuler
param_control
[inv_matEuler]
angEuler
Activation
wxyz



Anexo F

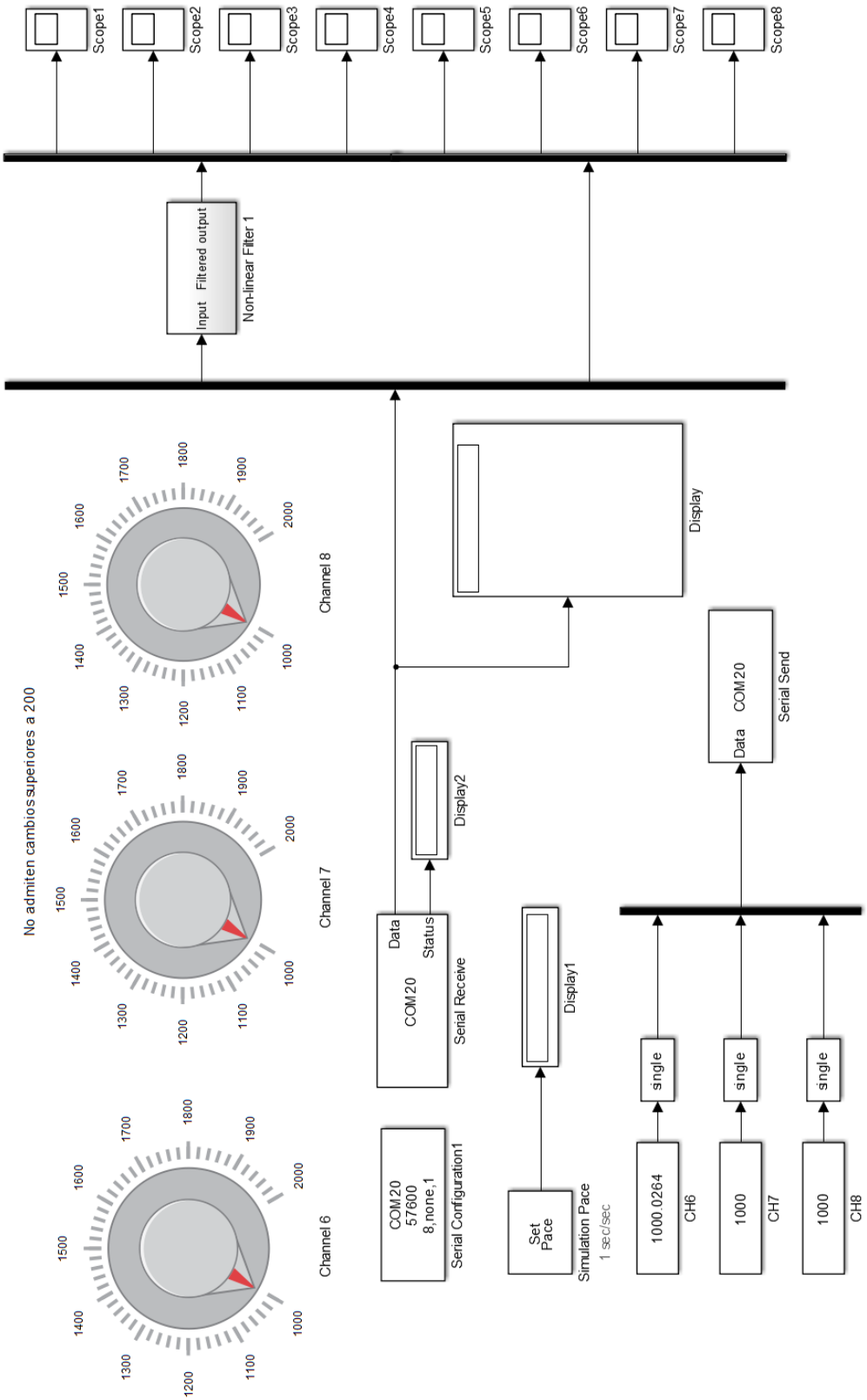
En este anexo se presenta el entorno de Simulación, tanto la parte del Airframe como la monitorización.




Position
-1
[rxyz]
Angles
180/pi
[r_pitch]
[pitch_real]
[pitch_est]
180/pi
[r_roll]
[roll_real]
[roll_est]
[r_yaw]
[yaw_real]
[yaw_est]
[angEuler]
[roll_real]
[pitch_real]
[yaw_real]
[r_roll]
[r_pitch]
[r_yaw]
pi/180
6
Angle reference
[roll_est]
[pitch_est]
[yaw_est]
pi/180
2
Estimated angles
Velocity
[vxyz_e]
PWM
3
Actuators
Acceleration
[axyz_b]
[accel]
Angular Speed
180/pi
[wxyz]
[gyro]
State Flow
7
STATE
1500
1500
1500
1342
RC command
[accel]
[gyro]
<AccelMeas_body>
<OmegaMeas_body>
5
Sensors
<roll_cmd>
<pitch_cmd>
<yaw_cmd>
<throttle_cmd>
1
Commands
[angEuler]
[rxyz]
[vxyz_e]
[axyz_b]
[wxyz]
<Euler>
<X_ned>
<V_ned>
<Accel_body>
<Omega_body>
4
States
States

Anexo G

En este anexo se observa el diagrama de simulink perteneciente a ***SERIAL_R/W_PC*** para la monitorización de datos, así como el envío a la tarjeta controladora



Anexo H

En este anexo se muestra el modelo dinámico que inicialmente hubo que elaborar para poder comenzar a realizar pruebas en simulación. Contiene todas las ecuaciones preparadas para el tricóptero.

```
function [dX,Y,DCM_be,Ugen,Wgen,Eulergen,dWgen] =
quad_Euler(X,Actuators,UAV,Motor,Battery)

%% Parametros
% Masa del quadcopter
m=UAV.Mass;
% Distancia entre el eje de cada motor y el CDG
L_roll=UAV.L_roll;
L_pitch_1=UAV.L_pitch_1;
L_pitch_2=UAV.L_pitch_2;

% Thrust factor (factor de empuje)
b=Motor.ThrustFactor;
% Drag Factor (factor de arrastre)
d=Motor.DragFactor;
% Momentos de inercia
Ix=UAV.Inertia(1,1);
Iy=UAV.Inertia(2,2);
Iz=UAV.Inertia(3,3);
I1=(Iy-Iz)/Ix;
I2=(Iz-Ix)/Iy;
I3=(Ix-Iy)/Iz;
% Resistencia del motor en ohm
Rm=Motor.Resistance;
% Constante del motor (V.s o N.m/A)
Km=1/2/pi/Motor.KV*60;
% Momento de inercia de los motores
Im=Motor.Inertia;
% Gravedad
g=9.81;

%% Sistemas de referencia
% Sistema de referencia inercial
% X,e11 - Norte
% Y,e21 - Este
% Z,e31 - Hacia abajo
% Sistema de referencia del cuerpo con origen en el CDG
% Xm,e1B - Hacia adelante (brazo del motor 1)
% Ym,e2B - Hacia la derecha, cuando se mira hacia adelante (brazo del
motor 4)
% Zm,e3B - Hacia abajo

%% Entradas
% PWM de los motores y tension Vbat de la bateria
% Tensiones de los motores
Actuators(1:3) = Actuators(1:3).*(Actuators(1:3)>=1000 &
Actuators(1:3)
<=2000)+1000*(Actuators(1:3)<1000)+2000*(Actuators(1:3)>2000);
if Actuators(4) == 0
    Actuators(4) = 1500;
end
```

```
um=(Actuators(1:3)*0.001-1)*Battery;

th_servo=(2*Actuators(4)*0.001-3)*pi/2; %recta con ordenada 1000-90
2000+90

%% Variables de estado
% Angulos de Euler (SRC)
angEuler=X(1:3);
% Velocidades traslacion (SRI)
vxyz_e=X(4:6);
% Velocidades angulares de rotacion (SRC)
wxyz=X(7:9);
% Posicion centro de masas (SRI)
rxyz=X(10:12);
% Velocidades angulares de los motores
wm=X(13:15);

%% Motores
% FCEM
em=Km*wm;
% Corrientes
im=(um-em)/Rm;
% Pares motores
Tm=Km*im;
% Ecuacion mecanica
dwm=1/Im*(Tm-d*wm.^2);

%% Fuerzas y pares sobre el cuerpo
% Motores 1 (delantero) y 3 (trasero): sentido contrario a la agujas
del reloj
% Motores 2 (izquierdo) y 4 (derecho): sentido de las agujas del reloj
w1=wm(1); w2=wm(2) ; w3=wm(3); %w4=wm(4);
%Fuerza ascendente eje Z del cuerpo (-)
        u1=b*(w1^2+w3^2+w2^2*cos(th_servo));
        u2=L_roll*b*(w1^2-w3^2); % Par neto de roll (+)
        u3=L_pitch_1*b*(w1^2+w3^2)-L_pitch_2*b*w2^2*cos(th_servo)-
d*w2^2*sin(th_servo); % Par neto de pitch (+)
        u4=d*(w1^2-w3^2+w2^2*cos(th_servo))-
b*w2^2*sin(th_servo)*L_pitch_2; % Par de yaw (-)
Ugen=[u1;u2;u3;u4];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
% Ecuaciones de estado
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
% Angulos de Euler en rad: angEuler=[Phi Theta Shi]
% Roll / Pitch / Yaw  |  Alabeo / Cabeceo / Guiñada
Phi=angEuler(1);
Theta=angEuler(2);
Shi=angEuler(3);
% Velocidad angular en el sistema de referencia del cuerpo en rad/s:
wxyz=[wx wy wz]'
wx=wxyz(1);
wy=wxyz(2);
wz=wxyz(3);
Wgen=[wx;wy;wz];
% La matriz de rotacion (SR cuerpo => SR inercial) es:
matR = ...
 [ cos(Theta)*cos(Shi) sin(Phi)*sin(Theta)*cos(Shi)-cos(Phi)*sin(Shi)
cos(Phi)*sin(Theta)*cos(Shi)+sin(Phi)*sin(Shi)
```

```
   sin(Shi)*cos(Theta)  sin(Phi)*sin(Theta)*sin(Shi)+cos(Phi)*cos(Shi)
cos(Phi)*sin(Theta)*sin(Shi)-sin(Phi)*cos(Shi)
  -sin(Theta)           sin(Phi)*cos(Theta)
cos(Phi)*cos(Theta)                              ];


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Velocidades angulares de Euler
dPhi = wx + tan(Theta)*sin(Phi)*wy + tan(Theta)*cos(Phi)*wz;
dTheta = cos(Phi)*wy - sin(Phi)*wz;
dShi = sin(Phi)/cos(Theta)*wy + cos(Phi)/cos(Theta)*wz;
% % Velocidades angulares de Euler (version simplificada)
% dPhi = wx;
% dTheta = wy;
% dShi = wz;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dinamica de rotacion en el SRC (incluye efecto giroscopico)
dwx= I1*wy*wz + 1/Ix*u2 - Im/Ix*wy*(w1-w3+w2*cos(th_servo)); %+w4;
dwy= I2*wx*wz + 1/Iy*u3 + Im/Iy*wx*(w1-w3+w2*cos(th_servo));
dwz= I3*wx*wy + 1/Iz*u4;
dwxyz = [dwx dwy dwz]';
dWgen=[dwxyz];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Dinamica de traslacion (SRI - angulos de Euler)
dvx = -(cos(Phi)*sin(Theta)*cos(Shi)+sin(Phi)*sin(Shi))*u1/m;
dvy = -(cos(Phi)*sin(Theta)*sin(Shi)-sin(Phi)*cos(Shi))*u1/m;
if rxyz(3)<0
    aux_g = g;
else
    aux_g = u1/m*((u1/m)<g)+g*((u1/m)>=g);
end
dvz = aux_g - cos(Phi)*cos(Theta)*u1/m;
dvxyz_e = [dvx dvy dvz]';
vxyz_b = matR'*vxyz_e;

% Vector de derivadas de las variables de estado
dX=[dPhi ; dTheta ; dShi  ; dvxyz_e ; dwxyz ; vxyz_e ; dwm];
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Ecuaciones de salida
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Medidas de los sensores:
% - Giroscopos: [wx wy wz]'
% - Acelerometros: proyeccion de g en el sistema de referencia del
cuerpo + aceleraciones)

% Aceleracion del cuerpo
accel_b = 1/m*[0 0 -u1]' + matR'*[0 0 aux_g]' - cross(wxyz,vxyz_b);

% Quaternion
% q0 = cos(Phi/2)*cos(Theta/2)*cos(Shi/2) +
sin(Phi/2)*sin(Theta/2)*sin(Shi/2);
```

```
% q1 = sin(Phi/2)*cos(Theta/2)*cos(Shi/2) -
cos(Phi/2)*sin(Theta/2)*sin(Shi/2);
% q2 = cos(Phi/2)*sin(Theta/2)*cos(Shi/2) +
sin(Phi/2)*cos(Theta/2)*sin(Shi/2);
% q3 = cos(Phi/2)*cos(Theta/2)*sin(Shi/2) -
sin(Phi/2)*sin(Theta/2)*cos(Shi/2);
% quat = [q0 q1 q2 q3]';

 % Vector de salida
 Y = [vxyz_b ;  wxyz ; angEuler ; accel_b ; dwxyz ; vxyz_e ; rxyz];
 Eulergen=[angEuler];
 DCM_be = matR;
 return
```

Anexo I

En este anexo se recoge el código de configuración del microcontrolador *MPU60X0* que pone a punto todos los registros y los *bytes* también para la lectura de la IMU.

```
% Mask for reading
MPU60X0_READ_FLAG = bin2dec('10000000');

% ACCELEROMETER SENSITIVITY SCALE FACTOR
% MPU60X0_AFS_SEL='00':  +/- 2g
% MPU60X0_AFS_SEL='01':  +/- 4g
% MPU60X0_AFS_SEL='10':  +/- 8g
% MPU60X0_ AFS_SEL='11':  +/- 16g
MPU60X0_AFS_SEL='00';
switch(bin2dec(MPU60X0_AFS_SEL))
    case 0
            MPU60X0_ACCEL_SSF=16384;
    case 1
            MPU60X0_ACCEL_SSF=8192;
    case 2
           MPU60X0_ ACCEL_SSF=4096;
    case 3
            MPU60X0_ACCEL_SSF=2048;
end

%  GYROSCOPE SENSITIVITY SCALE FACTOR
% MPU60X0_FS_SEL='00':  +/- 250 dps
% MPU60X0_FS_SEL='01':  +/- 500 dps
% MPU60X0_FS_SEL='10':  +/- 1000 dps
% MPU60X0_FS_SEL='11':  +/- 2000 dps
MPU60X0_FS_SEL='00';
switch(bin2dec(MPU60X0_FS_SEL))
    case 0
            MPU60X0_GYRO_SSF=131;
    case 1
            MPU60X0_GYRO_SSF=65.5;
    case 2
            MPU60X0_GYRO_SSF=32.8;
    case 3
           MPU60X0_GYRO_SSF=16.4;
end

% DIGITAL LOW PASS FILTER (DLPF)
% Bits [2:0] in CONFIG
MPU60X0_DLPF_CFG='100';
% The accelerometer and gyroscope are filtered according to the value
% shown in the table below (bits [2:0]).
% Accelerometer (Fs = 1kHz)
% 000: BW=260Hz  (delay=0ms)
```

```
% 001: BW=184Hz   (delay=2ms)
% 010: BW=94Hz    (delay=3ms)
% 011: BW=44Hz    (delay=4.9ms)
% 100: BW=21Hz    (delay=8.5ms)
% 101: BW=10Hz    (delay=13.8ms)
% 110: BW=5Hz      (delay=19ms)
% Gyroscope
% 000: BW=256Hz   (delay=0.98ms)  Fs=8kHz
% 001: BW=188Hz   (delay=1.9ms)   Fs=1kHz
% 010: BW=96Hz    (delay=2.8ms)   Fs=1kHz
% 011: BW=42Hz    (delay=4.8ms)   Fs=1kHz
% 100: BW=20Hz    (delay=8.3ms)   Fs=1kHz
% 101: BW=10Hz    (delay=13.4ms)  Fs=1kHz
% 110: BW=5Hz     (delay=18.6ms)

%  TEMPERATURE SENSITIVITY SCALE FACTOR
MPU60X0_TEMP_SSF=340;
% TEMPERATURE OFFSET (DEGREES)
MPU60X0_TEMP_OFFSET=35;

% CLOCK SOURCE
%MPU60X0_CLKSEL: Bits [2:0] in MPU60X0_PWR_MGMT_1
MPU60X0_CLKSEL='001';
% '000': Internal 8MHz oscillator
% '001': PLL with X axis gyroscope reference
% '010': PLL with Y axis gyroscope reference
% '011': PLL with Z axis gyroscope reference
% '100': PLL with external 32.768kHz
% '101': PLL with external 19.2MHz
% '110': Reserved
% '111': Stops the clock and keeps the timing generator in reset

%%%%%%%%%%%%%%%%%%
% MPU-60X0 Registers
%%%%%%%%%%%%%%%%%%
MPU60X0_ADDRESS = bin2dec('11010000');
MPU60X0_ADDRESS_WRITE = bin2dec('11010001');
MPU60X0_ADDRESS_READ = bin2dec('11010000');
MPU60X0_SELF_TEST_X = hex2dec('0D');
MPU60X0_SELF_TEST_X_BYTE = bin2dec('00000000');
MPU60X0_SELF_TEST_Y = hex2dec('0E');
MPU60X0_SELF_TEST_Y_BYTE = bin2dec('00000000');
MPU60X0_SELF_TEST_Z = hex2dec('0F');
MPU60X0_SELF_TEST_Z_BYTE = bin2dec('00000000');
MPU60X0_SELF_TEST_A = hex2dec('10');
MPU60X0_SELF_TEST_A_BYTE = bin2dec('00000000');
MPU60X0_SMPLRT_DIV = hex2dec('19');
% SAMPLE RATE DIVIDER
% This register specifies the divider from the gyroscope output rate
used
% to generate the Sample Rate for the MPU-60X0.
```

```
% The Sample Rate is generated by dividing the gyroscope output rate
% by SMPLRT_DIV: Sample Rate = Gyroscope Output Rate / (1 +
MPU60X0_SMPLRT_DIV)
MPU60X0_SMPLRT_DIV_BYTE = bin2dec('00000000');
MPU60X0_CONFIG = hex2dec('1A');
MPU60X0_CONFIG_BYTE = bin2dec(['00000' MPU60X0_DLPF_CFG]);
MPU60X0_GYRO_CONFIG = hex2dec('1B');
MPU60X0_GYRO_CONFIG_BYTE = bin2dec(['000' MPU60X0_FS_SEL '000']);
MPU60X0_ACCEL_CONFIG = hex2dec('1C');
MPU60X0_ACCEL_CONFIG_BYTE = bin2dec(['000' MPU60X0_AFS_SEL '000']);
MPU60X0_MOT_THR = hex2dec('1F');
MPU60X0_MOT_THR_BYTE = bin2dec('00000000');
MPU60X0_FIFO_EN = hex2dec('23');
MPU60X0_FIFO_EN_BYTE = bin2dec('00000000');
MPU60X0_I2C_MST_CTRL = hex2dec('24');
MPU60X0_I2C_MST_CTRL_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV0_ADDR = hex2dec('25');
MPU60X0_I2C_SLV0_ADDR_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV0_REG = hex2dec('26');
MPU60X0_I2C_SLV0_REG_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV0_CTRL = hex2dec('27');
MPU60X0_I2C_SLV0_CTRL_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV1_ADDR = hex2dec('28');
MPU60X0_I2C_SLV1_ADDR_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV1_REG = hex2dec('29');
MPU60X0_I2C_SLV1_REG_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV1_CTRL = hex2dec('2A');
MPU60X0_I2C_SLV1_CTRL_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV2_ADDR = hex2dec('2B');
MPU60X0_I2C_SLV2_ADDR_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV2_REG = hex2dec('2C');
MPU60X0_I2C_SLV2_REG_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV2_CTRL = hex2dec('2D');
MPU60X0_I2C_SLV2_CTRL_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV3_ADDR = hex2dec('2E');
MPU60X0_I2C_SLV3_ADDR_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV3_REG = hex2dec('2F');
MPU60X0_I2C_SLV3_REG_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV3_CTRL = hex2dec('30');
MPU60X0_I2C_SLV3_CTRL_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV4_ADDR = hex2dec('31');
MPU60X0_I2C_SLV4_ADDR_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV4_REG = hex2dec('32');
MPU60X0_I2C_SLV4_REG_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV4_DO = hex2dec('33');
MPU60X0_I2C_SLV4_DO_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV4_CTRL = hex2dec('34');
MPU60X0_I2C_SLV4_CTRL_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV4_DI = hex2dec('35');
MPU60X0_I2C_MST_STATUS = hex2dec('36');
MPU60X0_INT_PIN_CFG = hex2dec('37');
```

```
MPU60X0_INT_PIN_CFG_BYTE = bin2dec('00000000');
MPU60X0_INT_ENABLE = hex2dec('38');
MPU60X0_INT_ENABLE_BYTE = bin2dec('00000000');
MPU60X0_INT_STATUS = hex2dec('3A');
MPU60X0_ACCEL_XOUT_H = hex2dec('3B');
MPU60X0_ACCEL_XOUT_L = hex2dec('3C');
MPU60X0_ACCEL_YOUT_H = hex2dec('3D');
MPU60X0_ACCEL_YOUT_L = hex2dec('3E');
MPU60X0_ACCEL_ZOUT_H = hex2dec('3F');
MPU60X0_ACCEL_ZOUT_L = hex2dec('40');
MPU60X0_TEMP_OUT_H = hex2dec('41');
MPU60X0_TEMP_OUT_L = hex2dec('42');
MPU60X0_GYRO_XOUT_H = hex2dec('43');
MPU60X0_GYRO_XOUT_L = hex2dec('44');
MPU60X0_GYRO_YOUT_H = hex2dec('45');
MPU60X0_GYRO_YOUT_L = hex2dec('46');
MPU60X0_GYRO_ZOUT_H = hex2dec('47');
MPU60X0_GYRO_ZOUT_L = hex2dec('48');
MPU60X0_EXT_SENS_DATA_00 = hex2dec('49');
MPU60X0_EXT_SENS_DATA_01 = hex2dec('4A');
MPU60X0_EXT_SENS_DATA_02 = hex2dec('4B');
MPU60X0_EXT_SENS_DATA_03 = hex2dec('4C');
MPU60X0_EXT_SENS_DATA_04 = hex2dec('4D');
MPU60X0_EXT_SENS_DATA_05 = hex2dec('4E');
MPU60X0_EXT_SENS_DATA_06 = hex2dec('4F');
MPU60X0_EXT_SENS_DATA_07 = hex2dec('50');
MPU60X0_EXT_SENS_DATA_08 = hex2dec('51');
MPU60X0_EXT_SENS_DATA_09 = hex2dec('52');
MPU60X0_EXT_SENS_DATA_10 = hex2dec('53');
MPU60X0_EXT_SENS_DATA_11 = hex2dec('54');
MPU60X0_EXT_SENS_DATA_12 = hex2dec('55');
MPU60X0_EXT_SENS_DATA_13 = hex2dec('56');
MPU60X0_EXT_SENS_DATA_14 = hex2dec('57');
MPU60X0_EXT_SENS_DATA_15 = hex2dec('58');
MPU60X0_EXT_SENS_DATA_16 = hex2dec('59');
MPU60X0_EXT_SENS_DATA_17 = hex2dec('5A');
MPU60X0_EXT_SENS_DATA_18 = hex2dec('5B');
MPU60X0_EXT_SENS_DATA_19 = hex2dec('5C');
MPU60X0_EXT_SENS_DATA_20 = hex2dec('5D');
MPU60X0_EXT_SENS_DATA_21 = hex2dec('5E');
MPU60X0_EXT_SENS_DATA_22 = hex2dec('5F');
MPU60X0_EXT_SENS_DATA_23 = hex2dec('60');
MPU60X0_I2C_SLV0_DO = hex2dec('63');
MPU60X0_I2C_SLV0_DO_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV1_DO = hex2dec('64');
MPU60X0_I2C_SLV1_DO_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV2_DO = hex2dec('65');
MPU60X0_I2C_SLV2_DO_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV3_DO = hex2dec('66');
MPU60X0_I2C_SLV3_DO_BYTE = bin2dec('00000000');
MPU60X0_I2C_MST_DELAY_CTRL = hex2dec('67');
```

```
MPU60X0_I2C_MST_DELAY_CTRL_BYTE = bin2dec('00000000');
MPU60X0_SIGNAL_PATH_RESET = hex2dec('68');
MPU60X0_SIGNAL_PATH_RESET_BYTE = bin2dec('00000000');
MPU60X0_MOT_DETECT_CTRL = hex2dec('69');
MPU60X0_MOT_DETECT_CTRL_BYTE = bin2dec('00000000');
MPU60X0_USER_CTRL = hex2dec('6A');
MPU60X0_USER_CTRL_BYTE = bin2dec('00010000');
MPU60X0_PWR_MGMT_1 = hex2dec('6B');
MPU60X0_PWR_MGMT_1_BYTE = bin2dec(['00000' MPU60X0_CLKSEL]);
MPU60X0_PWR_MGMT_2 = hex2dec('6C');
MPU60X0_PWR_MGMT_2_BYTE = bin2dec('00000000');
MPU60X0_FIFO_COUNTH = hex2dec('72');
MPU60X0_FIFO_COUNTH_BYTE = bin2dec('00000000');
MPU60X0_FIFO_COUNTL = hex2dec('73');
MPU60X0_FIFO_COUNTL_BYTE = bin2dec('00000000');
MPU60X0_FIFO_R_W = hex2dec('74');
MPU60X0_FIFO_R_W_BYTE = bin2dec('00000000');
MPU60X0_WHO_AM_I = hex2dec('75');
MPU60X0_WHO_AM_I_BYTE=hex2dec('68');
```

UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA

PROYECTO FIN DE GRADO

PRESUPUESTO

CONTROL DE UN TRICÓPTERO PARA VUELO EN INTERIORES

Autor:
Jorge Buil García

Directores:
Juan Luis Zamora Macho
José Porras Galá

Madrid
Julio de 2016

Índice

Capítulo 1

Recursos Empleados

En este capítulo se calculará la cantidad de recursos que se han dedicado al proyecto, tanto materiales como equipo, software o humanos

1.1 Componentes Principales

Componentes	Cantidad
Quanun Trifecta Mini Foldable Tricopter Frame	1
Multistar 2150KV Motor The “Baby Beast”	3
TGY-9018 MG Servo	1
Afro ESC 12Amp	3
Batería LIPO 11.1V	1
Hélice (NPI)	7
OpenPilot Revolution	1
Emisora	1
Receptor	1
Transmisor Bluetooth	1
Receptor Bluetooth	1

1.2 Herramientas y Software

Componentes	Cantidad	Horas de proyecto	Horas de uso al año
Ordenador	1	400	1000
Cargador de Baterías	3	20	40
Cable USB-microUSB	1	5	200
Polímetro	3	5	250
Destornillador, alicates y otras	1	10	1000
Matlab/Simulink	7	300	1500
Microsoft Office	1	50	1000

1.3 Mano de Obra

Actividad	Horas
Montaje del Tricóptero	5
Desarrollo del entorno de simulación	50
Desarrollo del control	100
Ensayos y Depuración	200
Redacción de documentación	60

Capítulo 2

Costes Unitarios

En esta sección se detallan los costes y precios de cada uno de los elementos previamente analizados con los que se han llevado a cabo este proyecto.

2.1 Componentes

Componentes	Precio (€/Ud)
Quanun Trifecta Mini Foldable Tricopter Frame	27.59
Multistar 2150KV Motor The “Baby Beast”	13.79
TGY-9018 MG Servo	5.15
Afro ESC 12Amp	9.19
Batería LIPO 11.1V	15.00
Hélice (NPI)	2.00
OpenPilot Revolution	44.23
Emisora	85.00
Receptor	20.00
Transmisor Bluetooth	8.60
Receptor Bluetooth	8.60

2.2 Herramientas y Software

Componentes	Precio (€/Ud)
Ordenador	800.00
Cargador de Baterías	40.00
Cable USB-microUSB	1.99
Polímetro	40.00
Destornillador, alicates y otras	65.00
Matlab/Simulink	200
Microsoft Office	89.00

2.3 Mano de Obra

Actividad	Precio (€/hora)
Montaje del Tricóptero	20.00
Desarrollo del entorno de simulación	40.00
Desarrollo del control	40.00
Ensayos y Depuración	55.00
Redacción de documentación	45.00

Capítulo 3

Sumas Parciales

3.1 Componentes Principales

En este capítulo se calcula el coste total de cada uno de los recursos empleados a partir de las mediciones y los precios unitarios.

Componentes	Cantidad	Coste (€/Ud)	Coste Total (€)
Quanun Trifecta Mini Foldable Tricopter Frame	1	27.59	27.59
Multistar 2150KV Motor The “Baby Beast”	3	13.79	41.37
TGY-9018 MG Servo	1	5.15	5.15
Afro ESC 12Amp	3	9.19	27.57
Batería LIPO 11.1V	1	15.00	15.00
Hélice (NPI)	7	2.00	14.00
OpenPilot Revolution	1	44.23	44.23
Emisora	1	85.00	85.00
Receptor	1	20.00	20.00
Transmisor Bluetooth	1	8.60	8.60
Receptor Bluetooth	1	8.60	8.60
		Total	**297.11 €**

3.2 Herramientas y Software

Para calcular el coste de los bienes de equipo, herramientas y software dedicados a este proyecto, se ha supuesto un tiempo de amortización típico de 4 años y se ha repartido de forma proporcional en función del número de horas empleadas

Componentes	Cantidad	Horas de Proyecto	Horas de uso al año	Precio (€/Ud)	Amortización anual	Coste (€)
Ordenador	1	400	1000	800.00	25%	80.00
Cargador de Baterías	3	20	40	40.00	25%	15.00
Cable USB-microUSB	1	5	200	1.99	25%	0.012
Polímetro	3	5	250	40.00	25%	0.60
Destornillador, alicates y otras	1	10	1000	65.00	25%	0.16
Matlab/Simulink	1	300	500	200	25%	30.00
Microsoft Office	1	100	1000	89.00	25%	2.25
					Total	**128.02 €**

3.3 Mano de Obra

Actividad	Horas	Precio (€/h)	Precio total (€)
Montaje del Tricóptero	20.00	20.00	400
Desarrollo del entorno de simulación	40.00	40.00	1600
Desarrollo del control	40.00	40.00	1600

Ensayos y Depuración	55.00	55.00	3025
Redacción de documentación	45.00	45.00	2025
		Total	**8650 €**

Capítulo 4

Presupuesto General

Sumando todas las contribuciones, se concluye que el coste del proyecto asciende a:

Concepto	Coste (€)
Componentes Principales	297.11
Herramientas y Software	128.02
Mano de Obra	8650.00
Total	**9075.13 €**