ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

INGENIERO ELECTROMECÁNICO

CONTROL DE UN CUADRICÓPTERO PARA NAVEGACIÓN EN INTERIORES USANDO UN SENSOR DE FLUJO ÓPTICO

Autor: Néstor González García
Directores: Juan Luis Zamora Macho
José Porras Galán

Madrid

Julio de 2016

UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID

INSTITUTO DE INVESTIGACIÓN
TECNOLÓGICA

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESINAS O MEMORIAS DE BACHILLERATO

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

El autor D. Néstor González García, como estudiante de la UNIVERSIDAD PONTIFICIA COMILLAS (ICAI, E.T.S.I. de Ingeniera Industrial)

DECLARA que es el titular de los derechos de propiedad intelectual de la obra: *"Control de un cuadricóptero para navegación en interiores usando un sensor de flujo óptico"*, 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.

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, 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 y 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 y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; 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, 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, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
e) Asignar por defecto a estos trabajos una licencia *Creative Commons*.
f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente).*

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

a) Que la Universidad identifique claramente su nombre como autor de la misma
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.
d) 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.
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:

- 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 exclusive 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.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 20 de Julio de 2016

ACEPTA

Fdo……

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

"CONTROL DE UN CUADRICÓPTERO PARA NAVEGACIÓN EN INTERIORES USANDO UN SENSOR DE FLUJO ÓPTICO"

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2016 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

PROYECTO REALIZADO POR EL ALUMNO

Néstor González García

Fdo.:  Fecha: 16 / 07 / 2016

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Juan Luis Zamora Macho

Fdo.:  Fecha: 16 / 07 / 2016

José Porras Galán

Fdo.:  Fecha: 16 / 07 / 2016

Vº Bº DEL COORDINADOR DE PROYECTOS

Álvaro Sánchez Miralles

Fdo.: Fecha: / /

UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID

INSTITUTO DE INVESTIGACIÓN
TECNOLÓGICA

INSTITUTO DE INVESTIGACIÓN TECNOLÓGICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

INGENIENERÍA ELECTROMECÁNICA

PROYECTO FIN DE GRADO

RESUMEN

"Control de un cuadricóptero para navegación en interiores usando un sensor de flujo óptico"

Autor:

Néstor González García

Directores:

Juan Luis Zamora Macho
José Porras Galán

Madrid
Junio de 2016

UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID

INSTITUTO DE INVESTIGACIÓN
TECNOLÓGICA

CONTROL DE UN CUADRICÓPTERO PARA NAVEGACIÓN EN INTERIORES USANDO UN SENSOR DE FLUJO ÓPTICO

Autor: González García, Néstor
Estudiante de Ingeniera Electromecánica

Directores: Zamora Macho, Juan Luis; Porras Galán, José
Entidad Colaboradora: I.C.A.I. – Universidad Pontificia Comillas

Madrid, España

RESUMEN DEL PROYECTO

***Abstracto* — En este proyecto se persigue diseñar un control para un cuadricóptero que consiga estabilizarlo en vuelo estacionario. Este proyecto se enmarca en una línea de trabajo cuyo objetivo final es que el cuadricóptero sea capaz de posicionarse en un recinto cerrado y navegar de manera autónoma. En este sentido se han hecho grandes avances, tanto en el ámbito del control de estabilización como en el del sensor, que permiten un avance exitoso hacia el objetivo final.**

Palabras Clave — Cuadricóptero, UAV, dron, PX4FLOW, Flujo Óptico, Sensor, Control, Matlab, Simulink, Estabilización, Navegación.

I. INTRODUCCIÓN

Como la aplicación de los cuadricópteros tiene un enorme potencial, con proyectos en esta línea de trabajo se espera dar a la Universidad Pontificia de Comillas experiencia en este ámbito, con doble finalidad: Por un lado, la formación de los estudiantes en el ámbito académico. Y por el otro lado, facilitar el desarrollo de nuevas líneas de investigación sobre UAVs. El sistema al completo se implementará en el entorno Matlab/Simulink, con el propósito de utilizar unas herramientas potentes adaptadas a los estudiantes. Este proyecto pretende ser la continuación de otros proyectos de cursos anteriores [1]. Lo que se pretende es revisar los resultados de dicho proyecto, adaptarlos a nuestra nueva tarjeta de control e intentar ajustar parámetros para conseguir vuelo autónomo en interiores.

Los objetivos del proyecto son los siguientes:

- ✓ Conseguir adaptar el modelo de cuadricóptero para el controlador OPENPILOT gracias a Matlab y Simulink.
- ✓ Adaptación del modelo ya existente al entorno de simulación que se ha empleado.
- ✓ Diseño e implantación del control de vuelo y estabilización del cuadricóptero.
- ✓ Diseño del control de navegación para distancia constante al suelo.

Estos objetivos se han cumplido satisfactoriamente exceptuando el último, que solo se podido completar parcialmente. Adicionalmente se han realizado otras tareas para avanzar en la implantación del control de orientación en recinto cerrado y del vuelo autónomo:

- ✓ Instalación y calibración del sensor de flujo óptico para medir la distancia al suelo.
- ✓ Diseño de una interfaz en Simulink para interactuar con las medidas del sensor.
- ✓ Creación de un **modelo conjunto** para prueba, simulación y vuelo, **para cualquier tipo de aparato multirrotor**.
- ✓ Diseño de un diagrama de bloques para modificar los parámetros del control en tiempo real desde el PC, cuando estamos volando el dron.
- ✓ Diseño de un bloque lector de pulsos por PPM (Demodulador), en caso de tener limitados los canales de la emisora.
- ✓ Ajuste de las aceleraciones lineales y angulares procedentes de la IMU.
- ✓ Diseño de los bloques de transmisión de información por UART entre el dron y el PC.

El cuadricóptero empleado se muestra en la siguiente figura [Figura 1]. Cada una de sus hélices propulsoras se ve accionada por un motor trifásico *brushless*. La velocidad de giro de los motores se regula con Controladores Electrónicos de Velocidad (ESC), encargados de transformar la corriente continua de la batería en corriente alterna trifásica necesaria para accionar lo motores, además de generar la señal PWM.

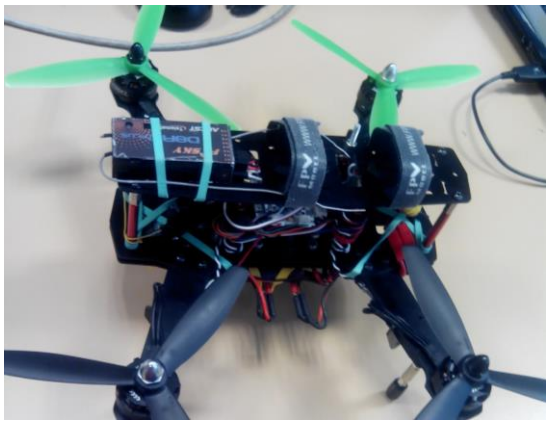


Figura 1. Elementos del cuadricóptero.

Dicho cuadricóptero consta de una estructura rígida fabricada en fibra de carbono (*ZMR250*) de HobbyKing, motores EMAX MT2204 II 2300KV/CW, ESCs *DYS BLHeli Mini 20A*, Receptor RC *FrSKY D8R-II PLUS,* Módulo Bluetooth HC-05 y Sensor de flujo óptico *PX4FLOW* , todo ello alimentado por una única batería LiPo de 3 celdas (11.1V tensión nominal). También se ha utilizado la emisora RC *TURNIGY 9XR* para enviar las referencias del control y activar las transiciones de la máquina de estados.

La tarjeta utilizada para implantar el control es la **OpenPilot *Revolution***. Dicha tarjeta es perfecta para el vuelo de vehículos aéreos, ya que utiliza una unidad de coma flotante (FPU) para el procesamiento preciso a baja latencia por medio de algoritmos de estimación avanzados. Esta cualidad única, junto con su carácter *OpenSource* y su alto índice de personalización, la sitúa bastante por encima del resto de sus competidoras, como APM o Pixhawk.

Además se han diseñado dos cableados: uno para monitorizar la tensión de la batería conectando el sensor de Potencia (PWR) a la placa, y otro para alimentar el sensor de flujo óptico. Por último, se ha construido una cubierta protectora, gracias a un diseño en *SolidEdge* que posteriormente se ha fabricado en material ABS en una impresora 3D, para colocar encima del sensor y evitar que alguna de sus frágiles partes sufra daños durante el aterrizaje o con alguna maniobra inestable.

II. METODOLOGÍA

Para la consecución de los objetivos se ha empezado revisando el modelo descrito en [2] y [3]. Una vez completado el modelo, se estudiado el entorno de Matlab para OpenPilot, se han sustituido los *Driver Blocks* de APM por bloques de *Waijung*, y se han calibrado los distintos elementos de la IMU: acelerómetros y giróscopos. A continuación se ha remodelado tanto el entorno de simulación como el control de estabilización, haciendo uso de una emisora para enviar parámetros en vuelo real. Por último se ha instalado y calibrado el sensor de flujo óptico, y se ha diseñado una máquina de estados completa para introducir el vuelo autónomo.

A. Adaptación del modelo

A partir del modelo descrito en [2] se ha diseñado un modelo dinámico en espacio de estado no lineal y multivariable. Las entradas son las señales de control de la emisora y la tensión de la batería, y las salidas son las medidas de la IMU (los giróscopos miden aceleraciones angulares, y los acelerómetros aceleraciones lineales) y los ángulos de Euler correspondientes a la orientación del cuadricóptero (*yaw*, *pitch* y *roll*), que sirven para computar los actuadores (señales PWM enviadas a los ESCs). El vector de estados consta de 16 variables de estado: las 3 componentes de la velocidad lineal en el sistema inercial, los ángulos de Euler, las velocidades angulares en los ejes propios del cuadricóptero, las 3 componentes de la posición de su centro de masas en el sistema inercial y las 4 velocidades angulares de los motores.

El cálculo de las fuerzas que afectan al cuadricóptero, y el modelo dinámico exacto utilizado en este proyecto se explicarán con más detalle en la memoria descriptiva.

B. Implantación en OpenPilot

Para descargar el control directamente sobre la tarjeta se ha utilizado un conjunto de bloques de Simulink diseñados por un grupo de programadores Tailandeses, llamado ***Waijung Blockset***. Dicho software se encarga de generar de forma fácil y automática el código en C# para Matlab y el entorno de Simulink, pensado específicamente para el chipset STM32 presente en las tarjetas de OpenPilot. Tanto el conjunto del *Waijung Blockset* para el STM32F4 como los *Driver Blocks* de años anteriores han sido completamente rediseñados con nuevas características y mejoras para este proyecto.

Para asegurar un correcto funcionamiento de los bloques que proporciona *Waijung* es necesario especificar una serie de parámetros: el micro exacto que estamos utilizando, compilador, velocidad el reloj, etc. Todo ello se define en un tipo de bloque especial que recibe el nombre de "***Target Setup***". Esto también es necesario para habilitar cualquier tipo de comunicación donde se seleccionen las características básicas de funcionamiento, velocidad de transmisión, períodos de muestreo internos, timers, módulo SPI, UART, I2C…

C. Entorno de simulación

Junto al modelo a implantar se ha incluido un entorno de simulación adaptado para poder probar distintos controles. Dicho entorno posee una interfaz gráfica en la que se pueden seleccionar las distintas opciones:

- ✓ Configurar señales de entrada desde una emisora virtual para observar la respuesta del control a determinadas acciones (Por ejemplo, escalones en los mandos de thrust, pitch, roll o yaw).
- ✓ Añadir ruido en las medidas de giróscopos, acelerómetros y sensores, añadir un offset en la medida de los giróscopos y elegir su magnitud.
- ✓ Elegir si los ángulos de Euler que se realimentan en el lazo de control son los reales (calculados por el modelo pero no medibles en la planta real) o los estimados mediante un filtro a elegir (Kalman, Complementario no lineal, DDF…).
- ✓ Elegir las referencias en los 3 ángulos de Euler, si sólo actúa el control de estabilización, o las referencias de altura y distancia a la pared, si actúa el control completo.

D. Control de estabilidad

Para mantener el cuadricóptero en vuelo estacionario con control manual por RC se ha diseñado un control de linealización mediante realimentación basado en un **control PID modificado** [4]. Incorpora todas las ventajas de un control PID (acción proporcional, Integral y diferencial) añadiendo una realimentación de los ángulos de Euler estimados y de las velocidades angulares, multiplicados por sus respectivas ganancias y añadiendo términos no lineales [5]. Con ello se consigue calcular los mandos intermedios para los ángulos, y seguidamente se hace la conversión adecuada en función del empuje para generar las señales de control requeridas por los ESC (PWM).

Es necesario añadir una realimentación ya que el control PID solamente se puede aplicar a una **planta lineal**, mientras que las ecuaciones que definen el comportamiento dinámico del cuadricóptero contienen términos donde algunas de las variables de estado se multiplican entre sí, creando términos no lineales. Esto se soluciona, como ya se ha explicado anteriormente, con la creación de variables intermedias.

E. Estimador de estados

El estimador de estados se encarga de proporcionar al control de estabilización una serie de variables estimadas, ya que estas son difíciles o imposibles de medir por causas ajenas al control (por ejemplo presencia de ruido). El estimador a utilizar, el **Filtro Complementario No Lineal**, combina las medidas de la IMU y añade un filtro paso alto a la medida de los giróscopos y uno paso bajo a la de los acelerómetros para eliminar el error de ambos valores.

Este filtro se hace especialmente útil en la estimación de variables de estado de la Unidad de Medición Inercial (IMU) ya que se obtiene la medida de aceleración lineal y angular libre de errores, previo filtro paso alto en los giróscopos y paso bajo en acelerómetros.

F. Monitorización y ajuste de parámetros en vuelo

Para poder conocer el valor de las distintas variables durante los ensayos se ha utilizado un módulo Bluetooth para poder recibir en el PC algunos valores del dron (orientación y rpm de los motores, por ejemplo) sin que exista interferencia con la emisora RC. Los valores se reciben en un archivo Simulink preparado para la representación gráfica de los mismos.

Después de diseñar e implantar el control de estabilización es necesario realizar un pequeño ajuste de los parámetros con el cuadricóptero funcionando. Se tiene de proyectos anteriores un Simulink que permite modificar los parámetros del control en tiempo real. Pero por motivos que se explican en el documento de la memoria, hemos ocupado todos los canales de la emisora para el control manual del aparato y no quedan canales libres para ajuste de parámetros en vuelo.

Para ello se ha añadido al archivo de Simulink de lectura de datos una serie bloques que permiten variar el valor de ciertos parámetros.

Consta de unos **diales** circulares de escala definida por el usuario con gran utilidad para afinar las ganancias del control PID, cambiar la ponderación de acelerómetro y giróscopo en el estimador de estados, etc. Dichos parámetros se van a transmitir al cuadricóptero a través de la propia conexión Bluetooth de lectura. Este método de ajuste del control ha resultado ser un éxito, por lo que será muy útil para futuros proyectos.

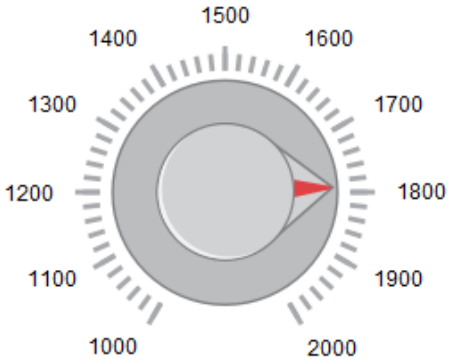


Figura 2. Ejemplo de dial para ajuste de valores.

G. Máquina de estados

Para coordinar los distintos controles y las referencias que actúan en cada uno se ha diseñado una máquina de estados en Simulink con la herramienta *Stateflow*. Como entradas recibe una señal de tiempo, el valor del *throttle* de la emisora, la tensión de la batería y una señal para permitir el **cambio de estado** (armar y desarmar motores). Esta señal, en un principio un interruptor de la emisora, se ha terminado cambiando por una señal (CH12) combinación de los canales *roll* y *pitch* por limitación de canales de la emisora.

Los estados que se han diseñado son los siguientes: ***Inicio***, estado inicial con los motores desarmados; ***Calibración***, donde se eliminan posibles *offsets* en los giróscopos y acelerómetros; ***Armado***, donde se espera comando del operario para armar motores; ***Vuelo***, donde sigue las trayectorias indicadas por el usuario (vuelo manual) o por el control de navegación (vuelo autónomo); ***Baja_Batería***, en el que se alerta de un nivel de carga límite de las celdas de la batería LiPo y se procede a descender; y ***Fin***, en el que los motores se detienen completamente.

Los estados de ***Baja_Batería*** y de ***Fin*** sirven como estados de fallo, siendo más conveniente una total desactivación de los motores frente a un descenso progresivo (por motivos de seguridad).

H. Sensor de Flujo Óptico

El sensor de flujo óptico constituye la piedra angular de este proyecto, ya que es un elemento necesario para el funcionamiento del control de navegación. A diferencia de los sensores infrarrojos de otros proyectos el sensor **PX4FLOW** [6] combina una serie de elementos: **cámara óptica**, capaz de detectar el flujo acumulado y la velocidad en los ejes horizontales, **sonar** integrado para una precisa medida de la distancia al suelo, y giróscopo, que aunque ya tenemos incluido en la IMU, puede resultar muy útil si combinamos las medidas de ambos para reducir el error.

El inconveniente de usar la medida de altura de este sensor es que por **debajo de 30 cm** no se obtiene una medida valida, nos salimos del rango útil de funcionamiento. Por esta razón, desde la máquina de estados es necesario aplicar un mando inicial que levante el cuadricóptero, y una vez que se supere la distancia mínima puede entrar en funcionamiento el control de altura.

Figura 3. Imagen del sensor PX4FLOW.

I. Control de navegación

Dicho control funciona de manera muy parecida al Control de Estabilización, solo que en lugar de las señales de la emisora utiliza las medidas de los sensores (sensor de flujo óptico en nuestro caso, solo tenemos uno). Esto nos permite diseñar un control que mantiene altura constante, dato que se puede definir en el código a cargar o mandar a través de la UART desde el PC.

Se ha logrado implementar de manera correcta la medida de la distancia de los sensores infrarrojos, aunque no se ha conseguido introducir la medida del sensor de flujo óptico. Sin embargo, lo más difícil ya está hecho: la instalación y calibración del sensor ha sido correcta, y el sensor ahora cuenta con sus propios bloques de lectura en Simulink. Solo queda usar dichos avances en un futuro proyecto de vuelo autónomo, pudiendo instalar numerosos sensores, diseñar controles de altura, seguimiento de rutas, etc.

III. RESULTADOS

El principal logro del proyecto ha sido la estabilización del cuadricóptero. A pesar de tener que implantar el control desde cero en un entorno de Simulink totalmente nuevo, se ha conseguido un control que mantiene la nave suspendida en el aire y que sigue las referencias de ángulos y de empuje que se indican desde la emisora, permitiendo su vuelo manual. A esto hay que añadir otros logros: los excelentes resultados de la medida de flujo acumulado, velocidad angular en los 3 ejes y distancia al suelo con el sensor de flujo óptico. La cámara se encuentra lista para ser implantada en cualquier multicóptero del proyecto conjunto. También la creación de un Demodulador para poder transmitir los pulsos de la emisora por PPM, etc.

Partiendo del diseño por asignación de polos en lazo cerrado y con el sistema de modificación de ganancias en simulación, se han afinado los parámetros del control para mejorar la estabilidad. Los parámetros definitivos se muestran en la siguiente tabla:

	Roll	Pitch	Yaw
ω_n	8	8	1.5
ζ	1	1	1
K	64	64	2.25
Ti	0	0	0
Td	0.25	0.25	1.3333
b	1	1	1

Tabla 1. Parámetros del control de estabilización

En la siguiente figura se muestra una representación las velocidades en los 3 ejes durante una simulación de despegue. Se puede observar que el cuadricóptero se mantiene estable, con unas velocidades horizontales oscilantes para compensar, y la velocidad vertical en torno a 0.

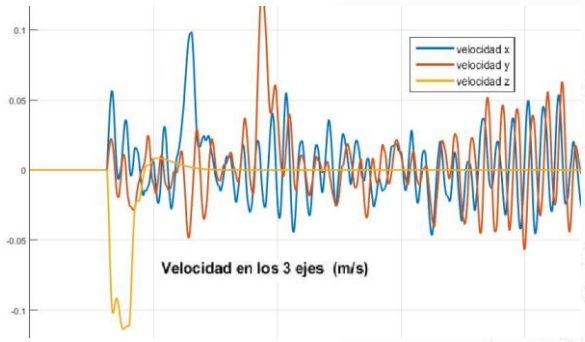


Figura 4. Velocidades ante un escalón en el empuje.

IV. CONCLUSIONES

La conclusión fundamental es que se ha alcanzado casi la totalidad de los principales objetivos del proyecto: la implementación en OpenPilot, la estabilización del cuadricóptero en vuelo estacionario y el correcto funcionamiento del sensor. Además se ha diseñado un entorno de simulación completo e intuitivo y se han realizado otras muchas tareas, detalladas al principio de este documento, que han permitido un gran avance hacia la navegación autónoma de multirrotores.

En cuanto a las mejoras de cara a futuros proyectos que continúen esta línea de trabajo se proponen las siguientes:

- ✓ Estimación de altura combinando las medidas del sensor de flujo óptico y los sensores infrarrojos.
- ✓ Desarrollo de un control de avance y giro basado en la actuación sobre la referencia de cabeceo/alabeo.
- ✓ Desarrollo de un sistema de telemetría utilizando la emisora para poder enviar canales adicionales (CH5, 6 y 7), o el uso de una tarjeta que soporte el uso de más de 4 canales.
- ✓ Uso de control predictivo de cara a programar el dron para realizar complejas tareas en modo automático.

V. REFERENCIAS

[1] J. Martínez Olondo, «Control de un cuadricóptero para vuelos autónomos en interiores,» Universidad Pontificia de Comillas, 2015.

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

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

[4] H. Bolandi, M. Rezaei, R. Mohsenipour, H. Nemati y S. Smailzadeh, «Attitude Control of a Quadrotor with Optimized PID Controller,» *Intelligent Control and Automation*, vol. 4, nº 3, pp. 335-342, 2013.

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

[6] D. Honegger, L. Meier, P. Tanskanen y M. Pollefeys, «An Open Source and Open Hardware Embedded Metric Optical Flow for indoor and outdoor applications,» *(ICRA) IEEE International Conference.*

UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID

INSTITUTO DE INVESTIGACIÓN
TECNOLÓGICA

CONTROL OF A QUADRICOPTER FOR NAVIGATION IN INTERIOR USING AN OPTICAL FLOW SENSOR

Author: González García, Néstor
Electromechanical engineering student

Directors: Zamora Macho, Juan Luis; Porras Galán, José
Collaborator Entity: I.C.A.I. – Universidad Pontificia Comillas

Madrid, Spain

PROJECT ABSTRACT

***Abstract* — this project aims to design a control for a quadcopter that will stabilize it during stable hovering. This project is part of a line of work whose ultimate goal is to make the copter able to position itself in an enclosed room and navigate autonomously. In this regard great progress has been made, both in the area of stabilization control and also sensor measurement, allowing for a successful advancement towards the ultimate goal.**

Keywords — Quadcopter, UAV, drone, PX4FLOW, Optical Flow, Sensor, Control, Matlab, Simulink, Stabilization, Navigation.

I. INTRODUCTION

As the application of the quadcopters has an enormous potential, with projects in this line of work the Universidad Pontificia Comillas is expected to receive experience in this particular field, with a dual purpose in mind: on one hand, training the students on an academic level. And on the other hand, facilitate the development of new lines of research on UAVs. The complete system will be implemented in the Matlab/Simulink environment, with the aim of using powerful tools adapted to students. This project aims to be the continuation of other projects of previous years [1]. It´s aim is to review the results of the project, adapting them to our new control card and to try to adjust parameters in order to achieve autonomous flight indoor.

The objectives of the project are the following:

- ✓ Adapt the quadcopter model to new controller OPENPILOT, thanks to Matlab and Simulink.
- ✓ Adaptation of the existing model to simulation environment that has been used.
- ✓ Design and implementation of the flight and stabilization control of the quadcopter.
- ✓ Design of a navigation control for maintaining constant height.

These objectives have been met satisfactorily except the last one, which only is partially complete. Additionally other tasks have been made to make progress in the implementation of the orientation at closed environments and autonomous flight control:

- ✓ Installation and calibration of optical flow sensor to measure the distance to the ground.
- ✓ A UI design in Simulink to interact with the sensor measures.
- ✓ Creation of a **joint model** for testing, simulation and flight, **for any type of multirrotor device**.
- ✓ Design of a block diagram to modify the control parameters in real time from the PC, mid-flight.
- ✓ Design of a reader block that detects PPM pulses (demodulator), in the case of limited channels on the transmitter.
- ✓ Adjustment of the linear and angular accelerations measurements from the IMU.
- ✓ The block design for transmission of information between the drone and the PC by UART.

The quadcopter used in this thesis is shown in the following figure [Figure 1]. Each of its propulsion propellers is powered by a *brushless*, three-phased electrical motor. The rotation speed of the motor is regulated by an Electronic Speed Controller (ESC), responsible for transforming DC battery power into three-phased alternating current required to power the engines, in addition to generating the PWM signal.

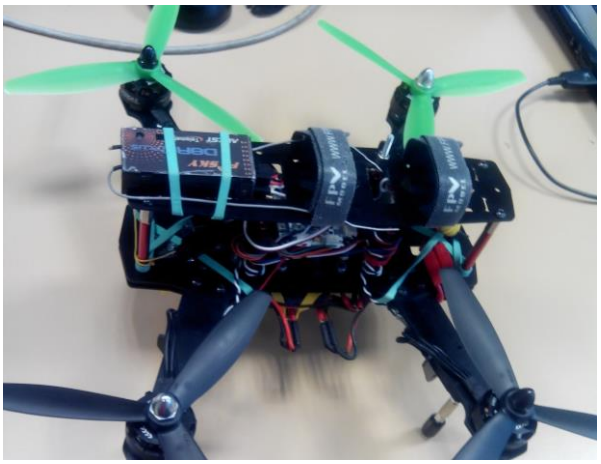


Figure 1. Elements of the quadcopter.

This quadcopter consists of a rigid structure made of carbon fiber (*ZMR250*) from HobbyKing, engines EMAX MT2204 II 2300KV/CW, *DYS BLHeli Mini*20A/ESCs, receiver RC *FrSKY D8R-II PLUS*, HC-05 Bluetooth module and optical flow *PX4FLOW* , all powered by a single LiPo battery of 3 cells (11.1V voltage rating). The *TURNIGY 9XR* RC transmitter has also been used to send the control references and activate the state-machine transitions.

The card used to implement control is the **OpenPilot *Revolution***. This card is perfect for aerial vehicles flight, since it uses a floating-point (FPU) drive for accurate processing at low latency through advanced estimation algorithms. This unique attribute, along with its *open source* nature and its high rate of personalization, leaves it situated quite above the rest of its competitors, such as APM or Pixhawk.

Also two cables have been designed: one to monitor the battery voltage by connecting sensor power (PWR) to the plate, and the other to power the optical flow sensor. Finally, a protective cover for the sensor has been built, thanks to a design in *SolidEdge* which subsequently was manufactured in ABS material on a 3D printer, and placed on top of the sensor to prevent fragile parts from being damaged during landing or some other unstable maneuver.

II. METHODOLOGY

In order to achieve the desired objectives, the project was started by reviewing the model described in [2] and [3]. Once completed the model, the Matlab environment for OpenPilot has been studied, the *Driver Blocks* of APM by *Waijung* blocks have been replaced, and the various elements of the IMU have been calibrated: accelerometers and gyroscopes. Additionally, both the simulation environment and stabilization control have been remodeled, making use of a transmitter to send parameters in real flight. Finally, the optical flow sensor has been installed and calibrated, and a complete state machine has been designed to introduce autonomous flight.

A. Adaptation of the model

From the model described in [2] we have designed a dynamic model in non-linear and multi-variable state space. Radio-control signals and the battery voltage are the inputs, and the outputs are the IMU measurements (the gyroscopes measure angular accelerations, and accelerometers linear accelerations) and Euler angles corresponding to the orientation of the quadcopter (yaw, pitch and roll), which are used to compute the actuators (PWM signals sent to the ESCs). The State vector consists of 16 state variables: the 3 components of the linear velocity in the inertial system, Euler angles, the angular velocities of the quadcopter shafts, the 3 components of the position of its center of mass in the inertial system and 4 angular speeds of the motors.

The calculation of the forces that affect the copter, and the exact dynamic model used in this project are explained in more detail in the project paper.

B. Implantation in OpenPilot

To download the control directly on the card a set of blocks in Simulink (designed by a group of Thai programmers) called ***Waijung Blockset*** have been used. Such software is responsible for easily and automatically generate code in C# for Matlab and the surroundings of Simulink, designed specifically for the STM32 chipset present in OpenPilot cards. Both the set of *Waijung Blockset* for the STM32F4 and the Driver Blocks from previous years have been completely redesigned with new features and improvements for this project.

To ensure a proper function of the blocks provided by Waijung it is necessary to specify a number of parameters: the exact microcontroller that we are using, its compiler, clock speed, etc. This is defined in a special type of block that receives the name of "*Target Setup*". This is also necessary to enable any type of communication where you select the basic characteristics of performance, output speed, internal sampling periods, timers, SPI, UART, I2C module...

C. Simulation Environment

Next to the model to be implemented a simulation environment adapted to try different controls has been included. This environment has a graphical interface in which you can select the various options:

- ✓ Configure input signals from a virtual station to observe the response of the control to certain actions (for example, steps in thrust, pitch, roll, or yaw control).
- ✓ Add noise in measurements of gyroscopes, accelerometers and sensors, also add an offset to the extent of the gyroscopes and choose their size.
- ✓ Choose if the Euler angles that is fed into the control loop are the real ones (calculated by the model but not measurable in the actual plant) or estimates through a filter to choose (Kalman, non-linear complementary, DDF...).
- ✓ Choose references in each of the 3 Euler angles, if the stabilization control is active, or references of height and distance to the wall, if the full control is active.

D. Attitude Control

To keep the copter hovering with manual control through RC a control has been designed using feedback linearization based on a **modified PID control** [4]. It includes all the advantages of a normal PID control (proportional action, Integral and differential) adding a feedback loop of the estimates of the angular velocities and Euler angles, multiplied by their respective gains and adding non-linear terms [5]. This is achieved to calculate the mid-controls for angles, and subsequently the conversion to generate the control signals required by the ESC (PWM) is made, according to the degree of the throttle command.

It is necessary to add feedback since the PID control can only be applied to a linear plant, while the equations that define the dynamic behavior of the quadcopter contain terms where some of the state-variables are multiplied together, creating non-linear terms. This is solved, as already explained above, with the creation of intermediate variables.

E. State Estimator

The State Estimator is responsible for providing the stabilization control with a series of estimated variables, since these are difficult or impossible to measure for reasons beyond our control (e.g. presence of noise). The estimator to be used, the Non-linear complementary filter, combines the IMU measurements and adds a high-pass filter to those of the gyroscopes and low-pass one to the accelerometers, to eliminate the error of both values.

This filter becomes especially useful in the estimation of the output of the Inertial Measurement Unit (IMU) state variables since you get error-free linear and angular acceleration measurement, prior low-pass filter higher in the gyroscopes values and low-pass in the accelerometers.

F. Monitoring and in-flight parameter adjusting

In order to measure the value of the different variables during test, a Bluetooth module has been used to receive some values of the drone (orientation and rpm of the engine, for example) on the PC without any interference with the RC transmitter. The values are received in a Simulink file prepared for the graphical representation of them.

After designing and implementing the stabilization control is it necessary to make a small adjustment of the parameters with the copter running. In previous projects a Simulink that allows to modify the control parameters in real time was used. But for reasons that are explained in the paper document, we have used all available channels on the transmitter for manual control of the device, and there are no free channels for in-flight adjustment of parameters.

A series blocks that allow to vary the value of certain parameters have been added to the Simulink serial-read data file.

It consists of a series of user-defined **circular dials** useful for tuning PID control gains, change the weighting of accelerometer and gyroscope in the State Estimator, etc. These parameters will be transmitted to the copter through the same Bluetooth connection used for reading. This method of adjusting the control has proved to be a success, so it will be very useful for future projects.

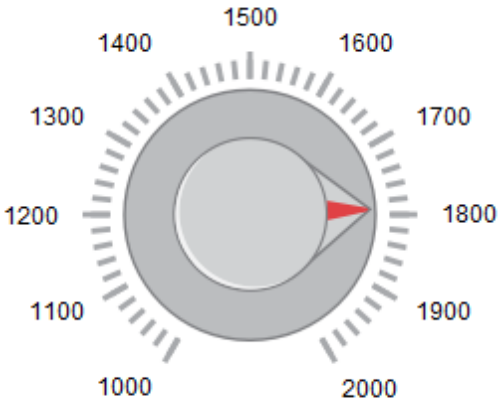


Figure 2. Example of dial for value adjustment.

G. State Machine

To coordinate the various controls and references that act in each one of them a state machine has been designed using Simulink *Stateflow* tool. It receives a time signal, the value of the throttle of the transmitter, the battery voltage and a signal to allow **change of the present state** (assemble and disassemble engines). This signal, initially a switch of the radio station, has been changed to a combination of the RC *roll* and *pitch* channels (CH12) by limitation of radio channels.

States that are designed are as follows: ***Inicio*** (start), initial state with unarmed engines; ***Calibración*** (Calibration), which removed possible offsets in the gyroscopes and accelerometers; ***Armado*** (Assembly), where the operator command is expected to assemble engines; ***Vuelo*** (Flight), where it follows the paths specified by the user (manual flight) or by the navigation control (autonomous flight); ***Baja_Batería*** (Low_batery), which alerts of minimum charge level of the cells of the LiPo battery and proceeds to descend; and ***Fin*** (End), where engines are completely stopped.

Baja_Batería states and **Fin** serve as fault states, being more convenient total deactivation of motors against a progressive decrease (for security reasons).

H. Optical flow sensor

The Optical flow sensor is the core of this project, since it is a necessary element for the operation of the navigation control. The PX4FLOW [6] differs from infrared sensors used in past projects in that it combines a number of elements: **optical camera**, capable of detecting the accumulated flow and speed on horizontal shafts, integrated **sonar** for a precise measurement of the ground distance, and a gyroscope, that although we have already included in the IMU, can be very useful if we combine both measures to reduce the error.

The drawback of using this sensor for height measurement is that **below 30 cm** it cannot get a valid measure, as it's out of the useful operating range. For this reason, it is necessary to apply an initial command that lift the copter from the state machine, and once the minimum distance is exceeded can the height control be operational.

Figure 3. PX4FLOW sensor view.

I. Navigation Control

This control works very similarly to the stabilization Control, only that instead of the signals of the radio station it uses sensor measurements (flow optical in our case, only have one). This allows us to design a control that keeps constant height, which can be defined in the code to upload or sent from the PC via UART.

The measure of the distance of infrared sensors has been implemented correctly, but the measurement of optical flow sensor has not been introduced. However, the hardest part is already done: the installation and calibration of the sensor has been correct, and the sensor now has its own reading blocks in Simulink. Said advances can be used in a future project of autonomous flight, installing numerous sensors, designing height controls, route-tracking, etc.

III. RESULTS

The main achievement of the project has been the stabilization of the quadcopter. Despite having to implement the control from scratch in a completely new Simulink environment, we have created a control which maintains the ship suspended in the air and following the angles and thrust references that are indicated from the RC station, allowing its manual flight. To this we add other achievements: the excellent results of the measurement of accumulated flow, angular velocity in all 3 axes and distance to the ground with the optical flow sensor. The camera is ready to be implemented in any multicopter of the joint project. Also the creation of a demodulator to transmit pulses of radio station by PPM, etc.

Based on design by pole assignment in closed loop and with modification of earnings in simulation system, will have tuned the control parameters to improve stability. The final parameters are shown in the following table:

	Roll	Pitch	Yaw
ω_n	8	8	1.5
ζ	1	1	1
K	64	64	2.25
Ti	0	0	0
Td	0.25	0.25	1.3333
b	1	1	1

Table 1. Stabilization control parameters.

The following figure shows a representation speeds on the 3 axes during a take-off simulation. You can see that the copter is stable, with a swing to compensate for horizontal speeds, and the vertical speed is around 0.

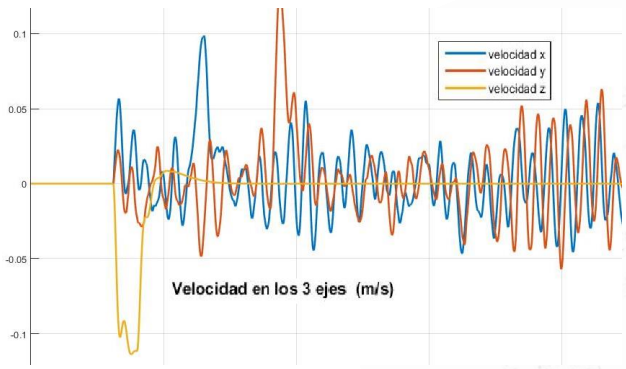


Figure 4. Step reaction in linear velocities.

IV. CONCLUSIONS

The key conclusion is that almost all of the main objectives of the project has been reached: the implementation in OpenPilot, the stabilization of the copter hovering and the proper functioning of the sensor. In addition a complete and intuitive simulation environment is designed and performed many other tasks, detailed at the beginning of this document, which have enabled a breakthrough towards the autonomous navigation of multirrotores.

Regarding improvements with a view to future projects that aim to continue this line of work are proposed as follows:

- ✓ Height estimation by combining optical flow sensor and infrared sensors measures.
- ✓ Development of a roll and pitch control based on the performance on the respective reference.
- ✓ Development of a telemetry system using radio to send additional channels (CH5, 6 and 7), or the use of a card that supports more than 4 channels.
- ✓ Use of predictive control in order to program the drone to perform complex tasks in automatic mode.

V. REFERENCES

[1] J. Martínez Olondo, «Control de un cuadricóptero para vuelos autónomos en interiores,» Universidad Pontificia de Comillas, 2015.

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

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

[4] H. Bolandi, M. Rezaei, R. Mohsenipour, H. Nemati y S. Smailzadeh, «Attitude Control of a Quadrotor with Optimized PID Controller,» *Intelligent Control and Automation*, vol. 4, nº 3, pp. 335-342, 2013.

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

[6] D. Honegger, L. Meier, P. Tanskanen y M. Pollefeys, «An Open Source and Open Hardware Embedded Metric Optical Flow for indoor and outdoor applications,» *(ICRA) IEEE International Conference.*

UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID

INSTITUTO DE INVESTIGACIÓN
TECNOLÓGICA

UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID

INSTITUTO DE INVESTIGACIÓN TECNOLÓGICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

INGENIENERÍA ELECTROMECÁNICA

PROYECTO FIN DE GRADO

CONTROL DE UN CUADRICÓPTERO PARA NAVEGACIÓN EN INTERIORES USANDO UN SENSOR DE FLUJO ÓPTICO

Autor:

Néstor González García

Directores:

Juan Luis Zamora Macho
José Porras Galán

Madrid
Junio de 2016

UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID

INSTITUTO DE INVESTIGACIÓN
TECNOLÓGICA

UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID
INSTITUTO DE INVESTIGACIÓN TECNOLÓGICA

Agradecimientos

A Juan Luis, por su gran esfuerzo y dedicación.

A José y Antonio del Taller, por su inestimable ayuda.

A Antonio y Jorge, por todo el trabajo conjunto y por nunca rendirse.

A Javi, por ser algo más.

A Germán, Álvaro, Felipe, Ramón y Rita, por estar siempre a mi lado.

A mi familia por ese apoyo constante que tanto ayuda.

Y en general para todos aquellos que en definitiva habéis tenido algo que ver con el proyecto. No lo habría conseguido sin vosotros.

"Los imposibles de hoy serán posibles mañana"

Konstantin Tsiolkovsky

UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID

INSTITUTO DE INVESTIGACIÓN
TECNOLÓGICA

INSTITUTO DE INVESTIGACIÓN TECNOLÓGICA

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)

INGENIENERÍA ELECTROMECÁNICA

PROYECTO FIN DE GRADO

MEMORIA

"Control de un cuadricóptero para navegación en interiores usando un sensor de flujo óptico"

Autor:

Néstor González García

Directores:

Juan Luis Zamora Macho
José Porras Galán

Madrid
Junio de 2016

ÍNDICE

Capítulo 1

Introducción

Todos estamos siendo testigos del notable crecimiento del uso de drones en los últimos tiempos, también conocidos como UAV o plataformas de vuelo aéreo no tripuladas (por sus siglas en inglés). Aunque pueden parecer un invento bastante novedoso, lo cierto es que la mecánica de los multicópteros se lleva desarrollando desde inicios del siglo pasado. Pero no ha sido hasta estas últimas décadas que los avances en la electrónica integrada han permitido la miniaturización y compactación de los sistemas de vuelo y control del aparato, consiguiendo un vehículo ligero, potente y robusto.

Sus numerosas posibilidades comprenden desde el entretenimiento a la investigación, desde el reconocimiento y el rescate a la filmación aérea. Y viendo que con cada día que pasa se le encuentran nuevas aplicaciones, no debemos subestimar la adaptabilidad de estos dispositivos para tareas que surjan en un futuro no muy lejano.

1.1 Vehículos y sistemas aéreos no tripulados

En este punto es necesario diferenciar entre dos conceptos que, aunque pueden parecer parecidos a simple vista, tienen una diferencia fundamental:

UAV (*Unmaned Aircraft Vehicle*) consiste en una aeronave que no está tripulada, pero que es controlada de forma remota por un operario en tierra. Es por ello que aunque puede contener ciertos ajustes o reguladores para mejorar su estabilidad en vuelo, no posee un sistema de control de vuelo autónomo y necesita una supervisión constante por parte del piloto.

Cuando al conjunto de un UAV se le incluye un sistema para conseguir un control de vuelo autónomo, es decir, que de forma totalmente aislada el dron es capaz de despegar, estabilizarse a sí mismo, volar y aterrizar, se trata de un modelo totalmente distinto de dron denominado **UAS (*Unmaned Aircraft System*).**

En este proyecto, debido a la complejidad del vuelo autónomo, se va a desarrollar primero el conjunto de scripts y diagramas de Simulink para poder volar el cuadricóptero de forma manual, para más adelante implementar el sensor con el fin de esbozar el vuelo autónomo.

1.2 Motivación y objetivos

El objetivo principal de este proyecto es utilizar el sensor de flujo óptico para lograr que el cuadricóptero se coloque en una posición estable a una distancia fija del suelo, sin intervención del operario, ampliable a conseguir que el dron sea capaz de mantenerse en esa posición estable bajo efecto de ligeras perturbaciones externas (que constituye el preámbulo para el desarrollo de proyectos de vuelo autónomo).

Aunque existen multitud de variantes de multicópteros, dependiendo del número de hélices que presenten (3 para el tricóptero, 4 el cuadricóptero, 6 el hexacóptero…) se va a utilizar el cuadricóptero, de 4 hélices, debido a la simplicidad del control y del montaje de hardware. Añadir más hélices no asegura una mayor estabilidad y maniobrabilidad. De hecho, en algunos casos supone empeorar el control de estabilidad.

El sistema de control se iba a implantar en una tarjeta *PIXHAWK FMU*, pero tras unas semanas con problemas de compatibilidad con el programa de Matlab y la falta de soporte del software de la placa (desactualizado desde 2013) se optó por cambiar de controladora. En su lugar se escogió la placa *OpenPilot REVOLUTION* [1], basada en un sistema *Open Source*. Este cambio no supone un gran impacto significativo en el proyecto, ya que *OpenPilot* también está diseñado para interactuar con Matlab y posee bloques y librerías para la creación del control. Sin embargo, en años anteriores no se han desarrollado en Matlab diagramas de bloques para el control de la aeronave en OpenPilot, sino para *ArduPilot Mega* (APM). Por lo tanto, aunque este proyecto pretende ser la continuación de proyectos anteriores de la universidad, es necesario volver a diseñar, modelar y revisar los resultados obtenidos en proyectos de años pasados.

Como ya hemos visto previamente, los drones son uno de los campos tecnológicos en auge en la actualidad por su versatilidad, gran número de aplicaciones y su sencillez constructiva. Aun así, aún queda mucho por mejorar en cuanto a los métodos de control de los cuadricópteros se refiere. Constituyen un campo de investigación perfecto para el ingeniero industrial de ICAI, sea cual sea su especialidad, por la gran cantidad de tecnologías que están involucradas en su construcción y/o manejo:

- ✓ Mecánica -> Modelado y Montaje del aparato. Cálculo de las matrices de rotación y de inercia del aparato.
- ✓ Electrónica Digital -> Sensores (Cámara Óptica) y electrónica de control.
- ✓ Electrónica de Control –> Diagrama de bloques en SIMULINK, además de la programación en MATLAB.
- ✓ Electrónica de Potencia -> ESC (“Electronic Speed Controls” - Alimentación de los motores) y baterías.
- ✓ Comunicaciones Industriales -> Dispositivo de radiocontrol del dron y Bluetooth.
- ✓ Máquinas Eléctricas -> Motores “brushless” (sin escobillas para cambio de polaridad).

El control de los cuadricópteros aún está en desarrollo pero posee un tremendo potencial. Mediante el desarrollo y la investigación de este proyecto se pretende ampliar la experiencia de la Universidad Pontificia Comillas en el ámbito de la electrónica de control de aeronaves, así como proporcionar a futuros estudiantes una base para su formación académica.

También cabe destacar que este proyecto se propone como una continuación sobre los ya existentes proyectos de navegación de cuadricópteros de años anteriores. Aunque se lograron pulir algunos problemas con el diseño de las aeronaves o la implantación de controles en SIMULINK, no se alcanzaron los objetivos de estabilidad y vuelo no guiado. Es por ello que se tratará de alcanzar estos objetivos, y dar soporte a los futuros proyectos que se desarrollen en el campo de los cuadricópteros.

En este proyecto se persigue la realización de los siguientes objetivos:

- ✓ Conseguir adaptar el modelo de cuadricóptero para el controlador OPENPILOT gracias a Matlab y Simulink, creando un entorno de simulación para probar el funcionamiento de forma virtual.
- ✓ Diseño del control de vuelo basándonos en la simulación anterior
- ✓ Implementación en el cuadricóptero y pruebas con vuelo dirigido.
- ✓ Posicionamiento autónomo del dron a distancia fija del suelo.

Además de estos objetivos principales, existe otro objetivo secundario que se intentará alcanzar en la medida de lo posible:

- ✓ Control de estabilidad frente a perturbaciones.

1.3 Metodología y recursos

Para alcanzar estos objetivos, se van a enumerar las siguientes tareas y se recogerán en un cronograma [Figura 1.1]:

- ✓ Montaje del cuadricóptero
- ✓ Caracterización y prueba de servomotores
- ✓ Configuración del modelo de Simulink
- ✓ Diseño del control de estabilidad
  - Unidad de Medición Inercial (IMU)
  - Modulación por Ancho de Pulso (PWM)
  - Modulación por Posición de Pulso (PPM)
  - Control de Actitud inercial (Attitude Control)
- ✓ Diseño de la máquina de estados e implementación en Matlab
- ✓ Prueba sobre vuelo en simulación
- ✓ Ajuste de parámetros de control en vuelo simulado
- ✓ Comunicación sensor - placa de control
  - Análisis Trama I2C
- ✓ Ensayo del sensor (cámara óptica)
- ✓ Prueba sobre vuelo real estacionario
- ✓ Ajuste de parámetros de control en vuelo real
- ✓ Prueba sobre vuelo real autónomo
- ✓ Diseño de control frente a perturbaciones
- ✓ Ajustes finales de robustez

Tareas	Enero				Febrero				Marzo				Abril				Mayo				Junio			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
Anexo B																								
Montaje																								
Prueba Motores																								
Modelo Simulink																								
Diseño y ajustes de la IMU																								
Modulación del Pulso																								
Attitude Control																								
Máquina de estados																								
Pruebas de simulación																								
Ajuste vuelo simulado																								
Prueba vuelo real																								
Ajuste vuelo real																								
Trama I2C																								
Ensayos sensores																								
Prueba vuelo autónomo																								
Control de perturbaciones																								
Ajustes finales																								
Memoria																								

Figura 1.1. Cronograma del Proyecto.

Durante el desarrollo del proyecto se han empleado los siguientes recursos:

- ✓ Cuadricóptero *ZMR250 (H250) Carbon Fiber Mini FPV* de Hobby King, que incluye: placa de distribución de potencia, cableado, amortiguadores, etc.
- ✓ Tarjeta de control *OpenPilot Revolution*
- ✓ Motores Brushless EMAX MT2204 II 2300KV/CW
- ✓ ESCs *DYS BLHeli Mini 20A*
- ✓ Módulo Bluetooth HC-05
- ✓ Receptor de radiocontrol *FrSKY D8R-II PLUS*
- ✓ Emisora de Radiocontrol *TURNIGY 9XR*
- ✓ Sensor de flujo óptico *PX4FLOW*
- ✓ Batería LiPo *TURNIGY* para el cuadricóptero y la emisora (1A, 11.1 V, 3S)
- ✓ Cargador-balanceador de tensión *iMAX B6AC*
- ✓ Matlab 2015a y Simulink
- ✓ Ordenador del Laboratorio de Control

Capítulo 2

Estado de la cuestión

2.1 Breve Historia

Aunque los cuadricópteros estén empezando a ponerse de moda en la actualidad, el desarrollo de vehículos de hélice con capacidad de despegue y aterrizaje vertical comenzó en la década de 1920. El ingeniero norteamericano *George de Bothezat* fue el primero en hacer volar un aparato cuadrimotor a una altura máxima de 5 metros del suelo, pero ante varios problemas en el desarrollo se canceló el proyecto. Un par de años más tarde, en 1922, el europeo *Étienne Œhmichen* construyo un modelo parecido que logro un vuelo estacionario de 5 minutos. Su siguiente versión del modelo se elevó 10 metros en un vuelo de 7 minutos de duración total.

Cabe destacar que ambos aparatos eran vehículos tripulados, de un peso y dimensiones considerables, y con mandos manuales hidráulicos, con lo que el manejo del aparato para conseguir la estabilidad era sumamente difícil. Esto provocó que los proyectos de investigación de vehículos multirrotores (la gran mayoría de fines militares) se viesen congelados, y más tarde cancelados. Y es que las ecuaciones de comportamiento de la aerodinámica datan de hace mucho más tiempo (en torno a 1890). Por poner un ejemplo, los controles PID ya se utilizaban en la dirección automática de embarcaciones en torno a la década de 1910, aunque harían falta más años hasta lograr que se perfeccionasen del todo. ¿Entonces, por qué no se han desarrollado los cuadricópteros hasta nuestros días?

La respuesta a esa pregunta es muy sencilla: porque la tecnología no había avanzado lo suficiente. De la misma manera que algunos diseños del famoso Leonardo da Vinci no pudieron ponerse en práctica por lo avanzado de su desarrollo en aquella época, lo mismo ocurrió con los cuadricópteros. No fue ya hasta casi la década de 1970 que se empezaron a perfeccionar algunos proyectos que habían quedado relegados al olvido.

En Noviembre de 1963 se fabricaron dos prototipos del prototipo experimental **Curtiss-Wright X-19**, que constaba de un fuselaje de avión al que se le habían montado 4 motores *Avco Lycoming T55-L-5* de turbo-eje. Se había diseñado como una aeronave de transporte VTOL que no necesitase de pista de despegue o aterrizaje, y con mayor velocidad que un helicóptero convencional. Sin embargo un accidente en las pruebas del primer prototipo provocó que el programa fuera suspendido.

Figura 2.1. Curtiss-Wright X-19.

La empresa Bell por su parte desarrolló el **X-22**, que partía de una idea similar al ejemplo anterior. Solo que ahora que las hélices del avión irían metidas en unos tubos para favorecer la estabilidad del aparato. Demostró bien las posibilidades de los aviones VTOL. Pero no alcanzó las especificaciones de velocidad requeridas por la Marina de EEUU y no se construyeron más modelos.

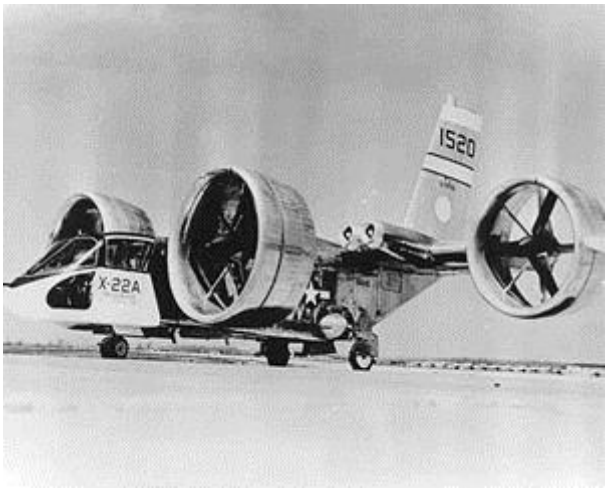


Figura 2.2. X-22.

No obstante estos proyectos no resultaron en fracaso, porque gran parte de la experiencia obtenida en el desarrollo, prueba y control de estos aparatos acabo aplicándose al **XV-15** concebido como el primer aparato bi-rotor de gran éxito, perfeccionado más adelante en el **XV-22 Ospre**y. La maravilla de este invento es combinar la facilidad del despegue y aterrizaje de un helicóptero con la velocidad y el gran radio de acción y velocidad de un avión. Es por ello que esta aeronave cumple perfectamente el papel de nave de asalto.

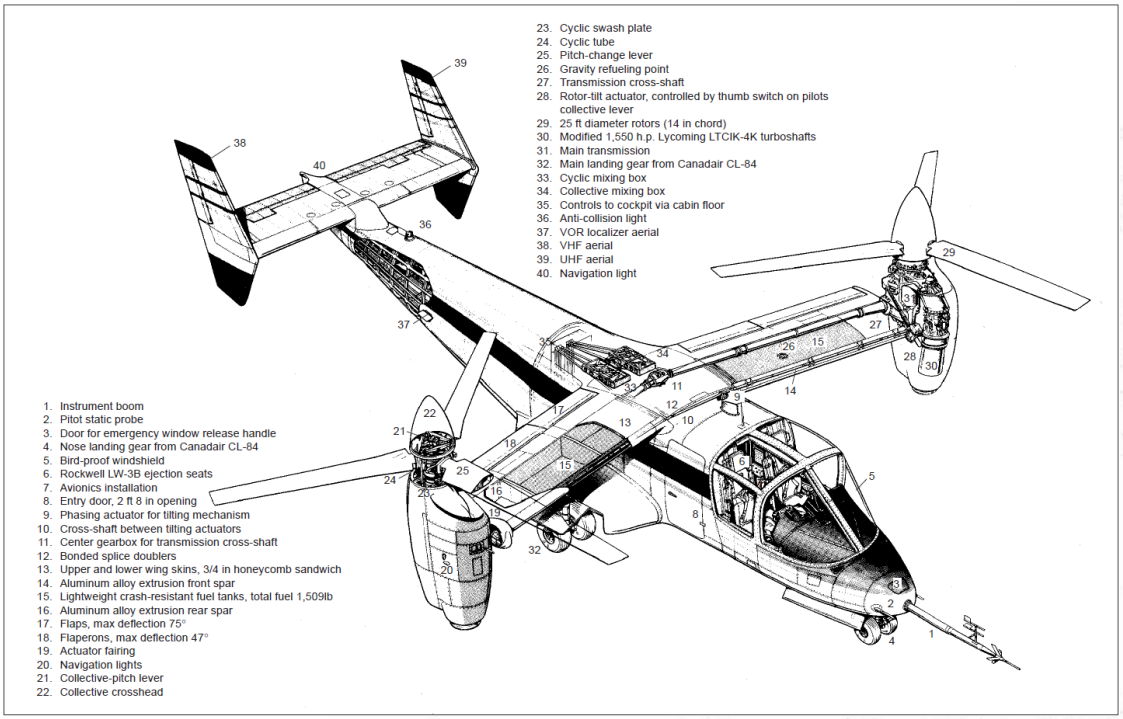


Figura 2.3. XV-15.

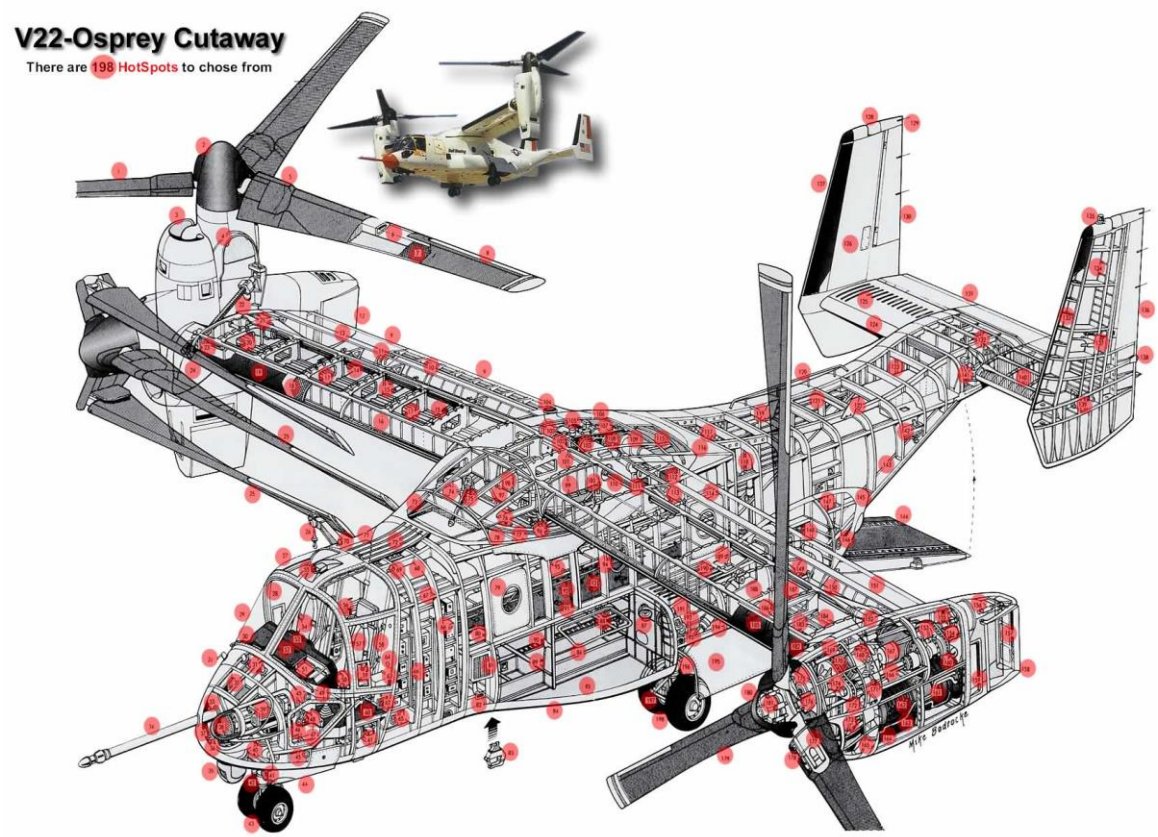


Figura 2.4. V-22 Osprey.

Sin embargo dicen que a la tercera va la vencida, y por fin gracias al avance de la tecnología de la electrónica integrada, los microprocesadores (tal y como predijo Moore, las capacidades e doblan cada 2 años y el tamaño se reduce a la mitad), las baterías, y tantos otros inventos modernos hemos conseguido incluir todo el aparato en un pequeño dispositivo no tripulado (autónomo o por control remoto) cuya estabilidad es mucho más fácil de conseguir, tiene una gran velocidad en vuelo, permite alcanzar lugares que nosotros mismos no podríamos, tiene una autonomía bastante aceptable, y un montón de etcéteras.

2.2 Modelado del Cuadricóptero

2.2.1 Sistemas de Ejes y Movimientos Básicos

El complejo movimiento de un cuadricóptero por el aire se puede definir gracias a su sistema de ejes y a sus movimientos básicos. El cuadricóptero posee un total de 6 grados de libertad: 3 por los desplazamientos en los ejes "x, y, z", y otros 3 debido a los giros respecto a dichos ejes. Dichos grados de libertad condicionan los movimientos básicos que veremos más adelante: dos grados para permitir el desplazamiento horizontal (*roll* y *pitch*), uno para la orientación del cuadricóptero (*yaw*) y tres para el movimiento en los 3 ejes:

✓ ***Yaw*, o movimiento de guiñada:**

El cuadricóptero debe volar siempre en una posición horizontal para permanecer estable. Es por ello que el único giro estable lo realiza respecto a su eje vertical (z) llamado guiñada, o *yaw*. Esto permite al cuadricóptero orientarse en la dirección correcta. Esto es posible aumentando la velocidad de giro de dos motores opuestos, frente a los restantes.

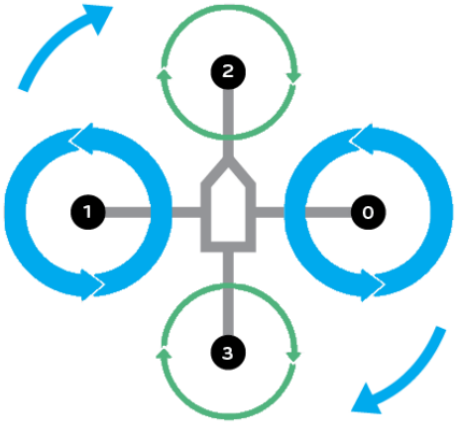


Figura 2.5. Movimiento de guiñada.

✓ ***Pitch*, o movimiento de cabeceo:**

El movimiento de cabeceo, o *pitch*, permite al cuadricóptero inclinarse hacia delante o hacia atrás. Esto permite a la aeronave avanzar y retroceder, de manera idéntica a un helicóptero. Para ello se aumenta la velocidad de giro de los motores traseros (para avanzar) o delanteros (retroceder).

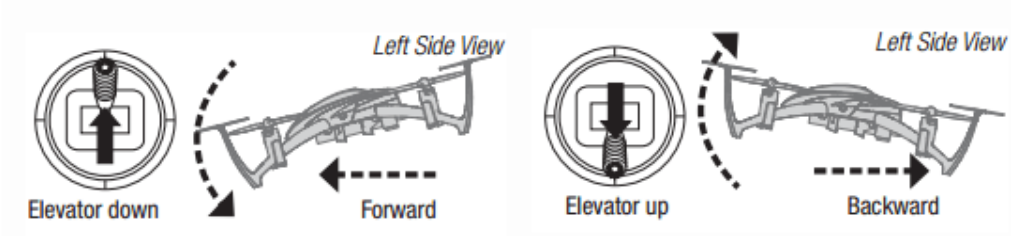


Figura 2.6. Movimiento de Cabeceo.

✓ ***Roll*, o movimiento de alabeo:**

De forma muy parecida al movimiento anterior, el alabeo (o *roll*) inclina el cuadricóptero gracias al aumento de potencia en dos motores consecutivos. Solo que en este caso se trata de los motores del lado izquierdo (para desplazarse hacia la derecha) o del lado derecho (hacia la izquierda).

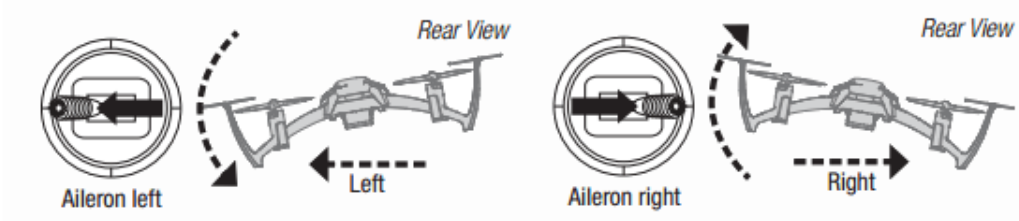


Figura 2.7. Movimiento de Alabeo.

Las ecuaciones de Euler son el método más común para trabajar con el comportamiento de un sólido rígido cualquiera. Su ventaja reside en que son bastante cómodas para su uso en cuadricópteros, ya que posicionan el objeto en función de los ángulos de cabeceo, alabeo y guiñada, ya explicados anteriormente.

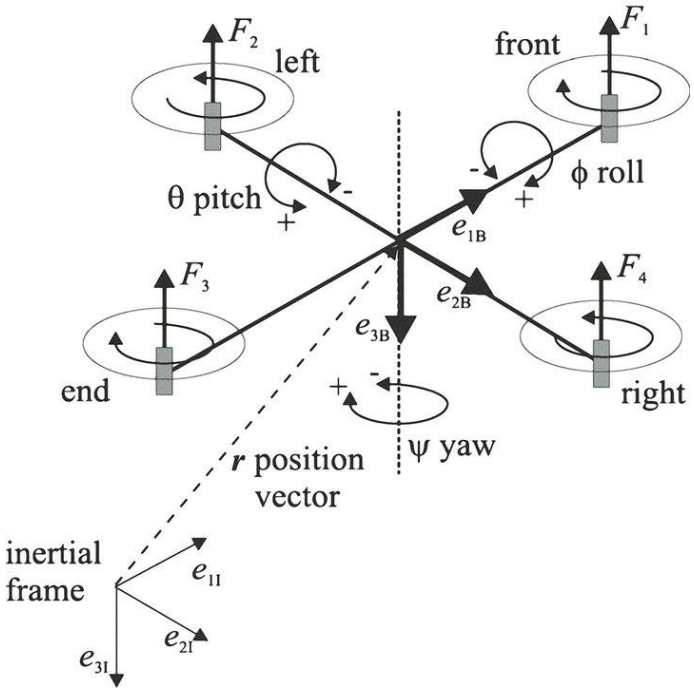


Figura 2.8. Sistemas de ejes y movimientos del cuadricóptero.

De esta manera, se consiguen expresar las coordenadas de los vectores del sistema inercial $\{e_{1I}, e_{2I}, e_{3I}\}$ en una base solidaria al cuerpo $\{e_{1B}, e_{2B}, e_{3B}\}$, utilizando las matrices de giro de los giros *Roll*, *Pitch* y *Yaw*, representadas en las siguientes ecuaciones [Ecuaciones 2.1, 2.2. y 2.3 respectivamente].

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{bmatrix} \cdot \begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix} \tag{2.1}$$

$$\begin{bmatrix} X_1 \\ Y_1 \\ Z_1 \end{bmatrix} = \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} \cdot \begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix} \tag{2.2}$$

$$\begin{bmatrix} X_2 \\ Y_2 \\ Z_2 \end{bmatrix} = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} X_3 \\ Y_3 \\ Z_3 \end{bmatrix} = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{2.3}$$

2.2.2 Matriz de Euler

Al componer los tres giros, aparece una matriz final de cambio de base [Figura 2.6], resultado del producto de las tres matrices de giro anteriores, que transforma directamente un vector de la base solidaria a la base fija. Esta matriz, llamada Matriz de Rotación, se expresa en función de los ángulos de Euler y resulta:

$$\boldsymbol{R} = \begin{bmatrix} cos\theta \cdot cos\psi & sin\phi \cdot sin\theta \cdot cos\psi - cos\phi \cdot sin\psi & cos\phi \cdot sin\theta \cdot cos\psi + sin\phi \cdot sin\psi \\ cos\theta \cdot sin\psi & sin\phi \cdot sin\theta \cdot sin\psi + cos\phi \cdot cos\psi & cos\phi \cdot sin\theta \cdot sin\psi - sin\theta \cdot cos\psi \\ -sin\theta & sin\phi \cdot cos\theta & cos\phi \cdot cos\theta \end{bmatrix} \quad (2.4)$$

Como matriz de giro, la Matriz de Rotación posee unas propiedades que la distinguen del resto: Su principal característica es que es ortogonal, lo cual se traduce en que el producto escalar de sus vectores fila o columna con ellos mismos son 1, y con el resto el producto es nulo. Por ello la matriz inversa coincide con la traspuesta, de manera que se puede realizar el cambio de base sólo con usar la Matriz de Rotación traspuesta $\boldsymbol{R^t}$ **(**del sistema fijo al solidario, o sea, un **cambio de base recíproco**).

2.2.3 Cálculo de fuerzas y momentos

Aquí se incluyen en una tabla todos los efectos de fuerzas y momentos sobre la aeronave, según el proceso descrito en el siguiente artículo, utilizado el año pasado en proyectos de vuelo de cuadricópteros de forma satisfactoria [2]. Es necesario establecer también unas condiciones de partida:

- ✓ Estructura rígida y simétrica. Tensor de inercia se supone diagonal, con $I_{XX} = I_{YY}$.
- ✓ Centro de gravedad situado en el centro de la estructura (origen de coordenadas).
- ✓ Palas rígidas, al girar no varían su ángulo de ataque.
- ✓ La fuerza vertical o empuje (*thrust*) y momento de arrastre (*drag*), derivados del giro de los propulsores, se suponen proporcionales al cuadrado de la velocidad angular.

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

Efectos Aerodinámicos

Las fuerzas de sustentación del cuadricóptero no son más que las cuatro fuerzas verticales que ejercen los motores, y que ejercen la fuerza necesaria para levantar la aeronave. Estas fuerzas se suponen proporcionales al cuadrado de las velocidades de giro de los motores, por lo que se define como **factor de empuje b** (*thrust factor*), a la constante de proporcionalidad. De la misma manera se define el **factor de arrastre d** (*drag factor*), como la constante de proporcionalidad entre los cuadrados de las velocidades angulares de los propulsores y el momento de resistencia aerodinámica que generan. Las cuatro fuerzas por tanto se calculan según la siguiente ecuación:

$$F_i = b \cdot \omega_i^2 \tag{2.5}$$

Pares de Inercia

El sentido de giro de los propulsores es según se muestra en siguiente imagen, con los motores 1 y 3 girando en sentido horario (CW) y los otros dos en sentido anti horario (CCW), enfrentados dos a dos. La configuración para definir los ejes del cuadricóptero es una configuración en x.

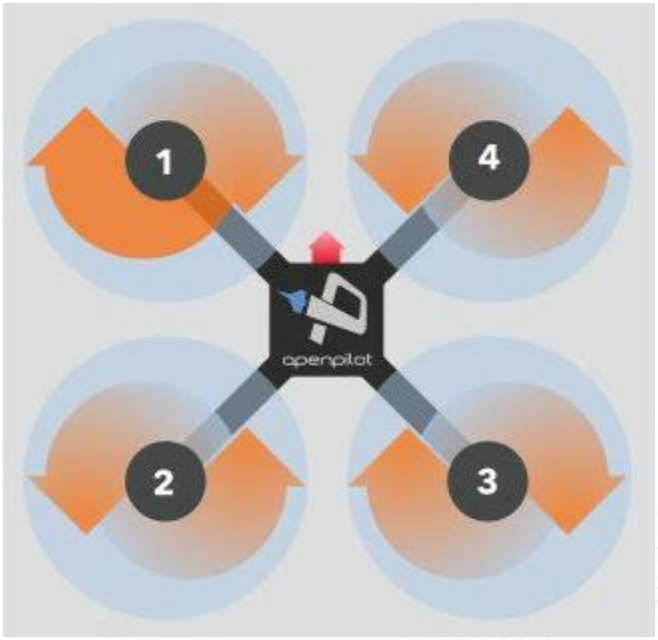


Figura 2.9. Imagen del sentido de giro de los propulsores.

Para simplificar la expresión de las fuerzas y momentos, se definen cuatro variables llamadas mandos, relacionadas con la fuerzas de empuje [Ecuación 2.6]: El primer mando u_1 es el empuje vertical total. El segundo es la diferencia entre los dos empujes que generan momento con respecto al eje x. El tercero es análogo, pero con respecto al eje y. Por último, el cuarto es el momento total generado por las palas en el eje z, que es el mismo que sufren éstas por el efecto aerodinámico (pero aplicado en sentido contrario debido a la ley de acción y reacción).

$$\begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} b(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) \\ b * L_{roll}(\omega_1^2 + \omega_2^2 - \omega_3^2 - \omega_4^2) \\ b * L_{pitch}(\omega_1^2 + \omega_4^2 - \omega_1^2 - \omega_3^2) \\ d(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) \end{bmatrix} \tag{2.6}$$

Efecto Gravitatorio

La fuerza neta se define de la siguiente manera en la Ecuación (2.7):

$$\vec{F} = mg \cdot \vec{e}_{3I} - b(\omega_1^2 + \omega_2^2 + \omega_3^2 + \omega_4^2) \cdot \vec{e}_{3B} \tag{2.7}$$

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

$$\begin{bmatrix} F_X \\ F_Y \\ F_Z \end{bmatrix} = \begin{bmatrix} -(sin\theta \cdot cos\phi \cdot cos\psi + sin\psi \cdot sin\phi) \cdot u_1 \\ -(sin\theta \cdot cos\phi \cdot sin\psi + cos\psi \cdot sin\phi) \cdot u_1 \\ mg - cos\theta \cdot cos\phi \cdot u_1 \end{bmatrix} \tag{2.8}$$

Los pares generados por el efecto aerodinámico de las palas, referidos a los ejes solidarios al cuerpo, quedan definidos como:

$$\begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} = \begin{bmatrix} L_{roll} \cdot u_2 \\ L_{pitch} \cdot u_3 \\ u_4 \end{bmatrix} \tag{2.9}$$

Donde L_{roll} y L_{pitch} son la longitud desde cada motor hasta los ejes Y (para el roll) y X (para el pitch) del cuadricóptero. La razón de que existan dos brazos distintas es debido a la propia asimetría de la estructura: los motores no se encuentran a la misma distancia del eje x de la aeronave que del eje y. Dichos pares están referidos a los ejes solidarios al cuerpo.

Efectos Giroscópicos

Es necesario ahora tener en cuenta los efectos giroscópicos. Se ven reflejados varios efectos: Primero, debido a la aparición simultánea de velocidades angulares en dos ejes distintos. Y segundo, por 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 efectos giroscópicos). Las ecuaciones que reflejan estos momentos son, respectivamente:

$$\begin{bmatrix} \tau'_x \\ \tau'_y \\ \tau'_z \end{bmatrix} = \begin{bmatrix} (I_{yy} - I_{zz})\dot{\theta}\dot{\psi} \\ (I_{zz} - I_{xx})\dot{\phi}\dot{\psi} \\ (I_{xx} - I_{yy})\dot{\phi}\dot{\theta} \end{bmatrix} \tag{2.10}$$

$$\begin{bmatrix} \tau''_x \\ \tau''_y \end{bmatrix} = \begin{bmatrix} I_m\, \dot{\theta}(\omega_1 - \omega_2 + \omega_3 - \omega_4) \\ -I_m\, \dot{\phi}(\omega_1 - \omega_2 + \omega_3 - \omega_4) \end{bmatrix} \tag{2.11}$$

Tercero, se encuentran los pares de inercia al acelerar los motores. Se pueden despreciar si la dinámica de los motores es suficientemente rápida, y en caso de que no solamente afectarían al eje Z. En cuanto a la fricción aerodinámica de avance del cuadricóptero, simplemente es considerada una perturbación.

Fricción

Además de las fuerzas verticales y del propio peso del cuadricóptero, aparecen otras horizontales debido al efecto aerodinámico del giro de las palas. Sin embargo, estas fuerzas se consideran despreciables en comparación con el empuje vertical, y por lo tanto no se tienen en cuenta.

2.2.4 Ecuaciones de movimiento

Se han realizado multitud de estudios distintos sobre el tema de la aerodinámica y el cálculo de fuerzas y momentos en un cuadricóptero [3]. En ellos se tienen en cuenta varios efectos, como la variación del ángulo de las hélices cuando éstas se desplazan horizontalmente [4]. Sin embargo, como este proyecto se encuentra orientado a vuelo en interiores, únicamente se recogen las ecuaciones simplificadas a partir de los momentos y fuerzas descritos anteriormente. Partiendo de las fuerzas en los tres ejes de la [Ecuación 2.8], se calculan las aceleraciones en los ejes fijos dividiendo entre la masa.

$$\begin{bmatrix}\ddot{X}\\ \ddot{Y}\\ \ddot{Z}\end{bmatrix} = \begin{bmatrix}-(sin\theta \cdot cos\phi \cdot cos\psi + sin\psi \cdot sin\phi) \cdot \dfrac{u_1}{m}\\ -(sin\theta \cdot cos\phi \cdot sin\psi + cos\psi \cdot sin\phi) \cdot \dfrac{u_1}{m}\\ g - cos\theta \cdot cos\phi \cdot \dfrac{u_1}{m}\end{bmatrix} \quad (2.12)$$

Por otro lado, de las ecuaciones de momentos se deduce la expresión de los momentos netos de alabeo, cabeceo y guiñada, a partir de los cuales se deducen las expresiones de las aceleraciones angulares:

$$\begin{bmatrix}I_{xx}\ddot{\phi}\\ I_{yy}\ddot{\theta}\\ I_{zz}\ddot{\psi}\end{bmatrix} = \begin{bmatrix}l\,u_2\\ l\,u_3\\ u_4\end{bmatrix} + \begin{bmatrix}(I_{yy}-I_{zz})\dot{\theta}\dot{\psi}\\ (I_{zz}-I_{xx})\dot{\phi}\dot{\psi}\\ (I_{xx}-I_{yy})\dot{\phi}\dot{\theta}\end{bmatrix} + I_m\begin{bmatrix}-\dot{\theta}(\omega_1-\omega_2+\omega_3-\omega_4)\\ \dot{\phi}(\omega_1-\omega_2+\omega_3-\omega_4)\\ 0\end{bmatrix} + \begin{bmatrix}0\\ 0\\ I_m(\dot{\omega}_1-\dot{\omega}_2+\dot{\omega}_3-\dot{\omega}_4)\end{bmatrix} \quad (2.13)$$

$$\begin{bmatrix}\ddot{\phi}\\ \ddot{\theta}\\ \ddot{\psi}\end{bmatrix} = \begin{bmatrix}\dfrac{l}{I_{xx}}u_2 + I_1\dot{\theta}\dot{\psi} - \dfrac{I_m}{I_{xx}}\dot{\theta}(\omega_1-\omega_2+\omega_3-\omega_4)\\ \dfrac{l}{I_{yy}}u_3 + I_2\dot{\phi}\dot{\psi} + \dfrac{I_m}{I_{yy}}\dot{\phi}(\omega_1-\omega_2+\omega_3-\omega_4)\\ \dfrac{1}{I_{zz}}u_4 + I_3\dot{\phi}\dot{\theta} + \dfrac{I_m}{I_{zz}}(\dot{\omega}_1-\dot{\omega}_2+\dot{\omega}_3-\dot{\omega}_4)\end{bmatrix} \quad (2.14)$$

Por último, en la última expresión se utilizan los momentos de inercia I_1, I_2 e I_3 para simplificar la expresión. La relación de momentos de inercia en la base del cuerpo y los nuevos momentos es la que viene descrita a continuación:

$$\begin{bmatrix}I_1\\ I_2\\ I_3\end{bmatrix} = \begin{bmatrix}\dfrac{I_{yy}-I_{zz}}{I_{xx}}\\ \dfrac{I_{zz}-I_{xx}}{I_{yy}}\\ \dfrac{I_{xx}-I_{yy}}{I_{zz}}\end{bmatrix} \quad (2.15)$$

2.2.5 Dinámica de los motores "Brushless"

Los motores empleados se (explicarán en detalle en la sección "*3.1.2. Motores Brushless"*) tienen la particularidad de ser motores trifásicos, pero alimentados con corriente continua, por lo que presentan desde fuera el esquema equivalente de un motor DC. La dinámica que rige su comportamiento se considera la misma que la de un motor DC:

Existen dos valores constantes bastante importantes. Por un lado **Kt** que relaciona par con corriente, y por otro lado **Ke** que relaciona la tensión del motor con su velocidad de giro angular. Estas dos constantes se pueden suponer idénticas, por lo que a partir de ahora se hablará de una única constante **K**.

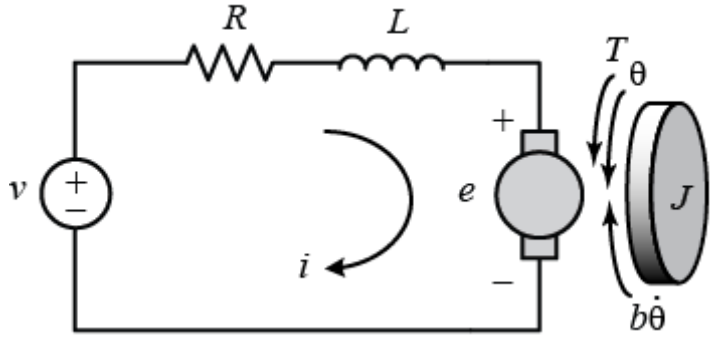


Figura 2.10. Esquema equivalente de un motor DC

Las ecuaciones diferenciales que rigen el funcionamiento de un motor DC son:

$$v = R\,i + L\,\frac{di}{dt} + e = R\,i + L\,\frac{di}{dt} + K\,\omega \tag{3.4}$$

$$i = \frac{T}{K} = \frac{1}{K}\,(I_m\,\frac{d\omega}{dt} + T_R) \tag{3.5}$$

Al resolver el sistema de ecuaciones diferenciales nos aparece la ecuación diferencial que relaciona la tensión aplicada al motor (v) y la velocidad angular de salida (ω). Como se han utilizado segundas derivadas (aceleraciones), la ecuación resultante es de segundo orden. Pero si se supone que la inductancia es lo suficientemente pequeña, debido al tamaño de los motores, la ecuación queda simplificada de la siguiente manera:

$$v = \frac{R}{K}\,(I_m\,\frac{d\omega}{dt} + T_R) + K\,\omega \tag{3.6}$$

En el caso de que el motor esté unido a una hélice, aparece un contrapar debido al arrastre. Por lo que la ecuación, despreciando el rozamiento interno en el motor, queda de la siguiente manera:

$$v = \frac{R}{K}\,(I_m\,\frac{d\omega}{dt} + d\omega^2) + K\,\omega \tag{3.7}$$

2.2.6 Efecto suelo y efecto techo

El **efecto suelo** consiste en el aumento del empuje vertical que sufre una aeronave cuando está cerca del suelo. Esto se debe a que el flujo de aire descendente que generan los propulsores rebota contra el suelo si está suficientemente cerca, creando una zona de latas presiones e impulsando la nave hacia arriba. De manera parecida, si la aeronave se encuentra situada cerca del techo se produce el análogamente el **efecto techo**. El flujo de aire producido por las palas crea ahora una zona de bajas presiones entre la aeronave y el techo, por lo que el empuje también aumenta de forma considerable. Ambos efectos influyen de manera significativa en el control de una aeronave cuando volamos en un recinto cerrado [5].

En lo que se refiere a este proyecto, se deberán considerar estos efectos durante el despegue y el aterrizaje (efecto suelo) y durante el modo de vuelo (efecto techo) de la siguiente manera:

Al despegar, el efecto suelo supone una ayuda inicial que disminuye la potencia requerida a los motores, aunque por otro lado puede provocar un despegue demasiado brusco y una pérdida de la estabilidad necesaria para el vuelo.. Por este motivo, se tratará como una perturbación que el control deberá ser capaz de vencer. Al aterrizar, el efecto suelo suaviza un acercamiento brusco al suelo, por lo que se tendrá en cuenta de manera empírica para diseñar el aterrizaje.

En cuanto al efecto techo, existe el riesgo de que el cuadricóptero choque contra él si se aproxima demasiado. A menor distancia del techo, mayor será el empuje que sufra el aparato hacia arriba, lo que a su vez hará que se acerque cada vez más. No contamos en este proyecto con sensores orientados a controlar la distancia entre el aparato y el techo. Sin embargo, sabiendo la altura total de la habitación y haciendo uso de la medida de distancia al suelo gracias al sensor de flujo óptico, podemos establecer un valor límite para la altura que puede alcanzar el cuadricóptero.

2.2.7 Modelo dinámico del Cuadricóptero

Este proyecto se ha desarrollado tomando como base de partida proyectos de años anteriores. Varios elementos, entre ellos las ecuaciones del modelo matemático y algunos parámetros del cuadricóptero, se han basado en estudios previos [6]. Otros parámetros fueron calculados a partir del banco de ensayos de motores y de una serie de ensayos experimentales que se llevaron a cabo con las hélices y los motores. Estos datos se recogen en la siguiente tabla:

PARÁMETROS MECÁNICOS Y ESTRUCTURALES		**PARÁMETROS DÍNAMICOS DE LOS MOTORES**	
Masa (Kg)	0.5427	**Kv (rpm/v)**	2300
Brazo_Roll (m)	0.1025	**Rm (ohm)**	0.405
Brazo_Pitch (m)	0.0826	**Inercia (Kg*m^2)**	$104e^{-8}$
I_{XX} **(Kg*m^2)**	$7.83e^{-4}$	**Diámetro (m)**	0.127
I_{YY} **(Kg*m^2)**	$7.85e^{-4}$	***b* (Ns/rad)**	$4.77360e^{-7}$
I_{ZZ} **(Kg*m^2)**	$1.4e^{-3}$	***d* (Ns/rad)**	$4.774e^{-9}$

Tabla 2.2. Parámetros físicos de la estructura del cuadricóptero y de los rotores

De ese estudio, también se concluye que los efectos aerodinámicos sobre la estructura son despreciables en el modelo, por lo que no se tienen en cuenta. Además de ésta, se hicieron otras simplificaciones, como suponer el tensor de inercia diagonal debido a la simetría que presenta la aeronave. Esto facilita el cálculo y permite simplificar las ecuaciones. Debido a esta simetría también se comprueba que los momentos de inercia en los ejes horizontales son prácticamente iguales.

Capítulo 3

Hardware y software

3.1 Elementos del cuadricóptero

Aquí se incluye los distintos elementos presentes en el cuadricóptero. La mayoría se han adquirido juntos en un pack al comprar la estructura *ZMR250 (H250) Carbon Fiber Mini FPV* en '*HobbyKing.com'*, y el resto se pueden obtener en cualquier tienda especializada de juguetes de radiocontrol o tiendas electrónicas.

3.1.1 Estructura

La estructura del dron no es nada más que el soporte para el resto de elementos del cuadricóptero. Está construida en fibra de carbono, material lo suficientemente rígido para soportar las fuerzas y aceleraciones del aparato sin deformarse apenas. Al contrario que otros cuadricópteros, no se trata de una estructura simétrica: por ello no tiene la misma maniobrabilidad inclinándose hacia adelante que hacia los lados. Por ello posee una disposición en 'X', ya que no existe configuración en '+' asimétrica. No consta de circuito de potencia integrado, por lo que será necesario añadir una placa de distribución de potencia más adelante. Además, como la altura de las patas no es suficiente para poder colocar el sensor en la parte inferior, se ha optado por colocar unas patas más largas como extensión de las que vienen por defecto.

Figura 3.1. Estructura ZMR250 del cuadricóptero.

Cabe destacar también que la fibra de carbono puede conducir electricidad de forma aleatoria, así que es conveniente proteger las conexiones de potencia y asegurarse de que la placa de control, potencia, sensor, etcétera se encuentran aislados (usando tornillos y arandelas de plástico) para evitar que una descarga estropee algún componente valioso del cuadricóptero.

3.1.2 Motores “Brushless”

Como la alimentación del dron es por baterías, y estas proporcionan corriente continua (DC), lo más lógico sería usar motores DC para la propulsión del cuadricóptero. Dichos motores funcionan generando un campo magnético giratorio que fuerza al rotor a girar, gracias a una computación mecánica de la corriente, las escobillas y las bobinas de los electroimanes. Pero estos motores presentan varios inconvenientes, entre ellos el alto desgaste producido por la fricción escobilla-delga, cuyo mantenimiento no es trivial. Y además suelen ser bastante aparatosos en contraste con la alternativa: los ampliamente extendidos motores *“brushless”*, ya que carecen de escobillas (*“brush”* en inglés).

Los motores Brushless no son nada menos que pequeños motores trifásicos síncronos, que funcionan gracias al campo magnético giratorio producido por el paso de la corriente por las bobinas del estator. Al no tener apenas rozamiento, no producen apenas calor y pérdidas por fricción. Y por lo tanto carecen de mantenimiento, alarga la vida útil del producto, evita tener que cambiar los motores constantemente, etc. Es por ello que en este proyecto se van a utilizar dichos motores.

Figura 3.2. Motores EMAX-MT2204-II-2300KV.

Pero el hecho de usar dichos motores presenta un solo inconveniente: es necesario un componente para transformar la corriente y tensión de la batería (DC) en corriente y tensión admisibles por los motores trifásicos (AC), además de poder regular dicha tensión para controlar la velocidad de giro de los motores. Es aquí donde entran en juego unos útiles circuitos miniaturizados llamados ESCs, que explicaremos en el siguiente punto.

3.1.3 Controlador electrónico de velocidad (ESC)

Un ESC, de sus siglas en inglés *Electronic Speed Control*, consta de un circuito inversor que permite generar corriente trifásica (AC) a partir de corriente continua (DC). También cuenta con una entrada de señal PWM (*Pulse Width Modulation*) que permite controlar la frecuencia de giro de los motores con un ancho de pulso variable, normalmente entre 1000 y 2000 µs. Así, con una señal de entrada de ancho 1ms, el motor estará completamente parado; y a 2ms, el motor estará a máximas revoluciones. Esto es equivalente a la señal de control de un motor DC normal.

Pero además el uso de estos ESC presenta ventajas adicionales: Por un lado se trata de ESC programables, es decir, podemos ajustar la diferencia de ancho de pulso para el rango de funcionamiento del motor. Esto es útil si queremos que el motor se encienda, digamos, en 1200µs, y se apague en 1800 (por limitaciones de la emisora, o porque queremos ajustar cierto ancho de pulso a girar en un sentido y parte al sentido contrario por ejemplo). Por otro lado, estos ESC cuentan con un limitador de corriente BEC (*Battery Eliminator Circuit*), para evitar daños a los motores en caso de picos en la alimentación, así como un conjunto de leds y sonidos que nos permite saber el estado de armado de los motores.

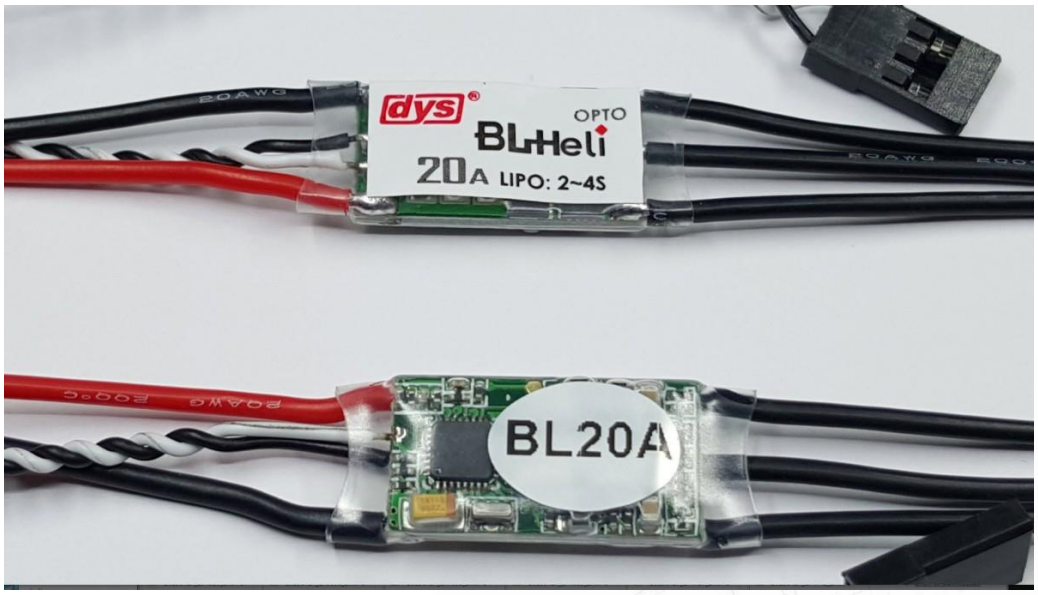


Figura 3.3. Detalle de los ESCs DYS BL-Heli 20A.

Por último los ESC son capaces parar el motor de forma casi instantánea gracias al fenómeno de **freno dinámico** o **freno reostático**, gracias a que se puede emplear como un generador para disminuir la velocidad de forma brusca. La energía restante se disipa en forma de calor, por lo que no es recomendable hacer uso continuo de esta función bajo la posibilidad de calentamiento excesivo del circuito de control. Dicha funcionalidad es especialmente útil en situaciones de peligro o emergencia donde se requiera una parada total de los motores, como puede ser por ejemplo el hecho de que el dron se descontrole y se precipite hacia una persona. En caso de un motor normal, la inercia de las hélices podría ocasionar alguna lesión en el individuo en cuestión, pero con este método se minimiza cualquier herida cortante ocasionada por el giro de las palas.

3.1.4 Placa de Distribución y Sensor de Potencia

La Placa de Distribución de Potencia, abreviada PDB (*Power Distribution Board*), supone el enlace directo entre la fuente de alimentación primaria (en nuestro caso, baterías) y el resto de componentes del cuadricóptero. Esto es necesario porque en caso de sobrecarga o cortocircuito, la placa contiene fusibles y elementos electrónicos que impiden que otros elementos de la aeronave queden inservibles. Pero a costa de la salud de la propia placa, como por desgracia se ha comprobado reiteradamente a lo largo del desarrollo del proyecto.

Pero se ha elegido una PDB que contiene un elemento bastante importante para el control de vuelo autónomo: un sensor de potencia (PWR) que permite a la placa de control monitorizar en todo momento los niveles de tensión y amperaje que la batería proporciona a los ESCs. Si se detecta algo anormal, como que por ejemplo el nivel de tensión de la batería esta próximo al nivel mínimo, podemos hacer que el dron (en vuelo autónomo) entre en estado de aterrizaje para descender suavemente. De lo contrario, tarde o temprano uno de los motores recibiría menos potencia, perdería par de giro, y la aeronave al completo se precipitaría de forma bastante brusca contra el suelo.

En nuestro caso, la PDB cuenta con las siguientes características técnicas:

- ✓ PDB/5.3V 3A UBECs/V and A Sensors all in one
- ✓ 60A capable
- ✓ 4s capable
- ✓ Standardized sizing 35mm (30.5mm)
- ✓ HK Pilot and APM/Pixhawk compatible
- ✓ UBEC Current: 3A outputs
- ✓ Power input: 2 x contact points for Voltage/current monitor
- ✓ Power output: 8 x contact points

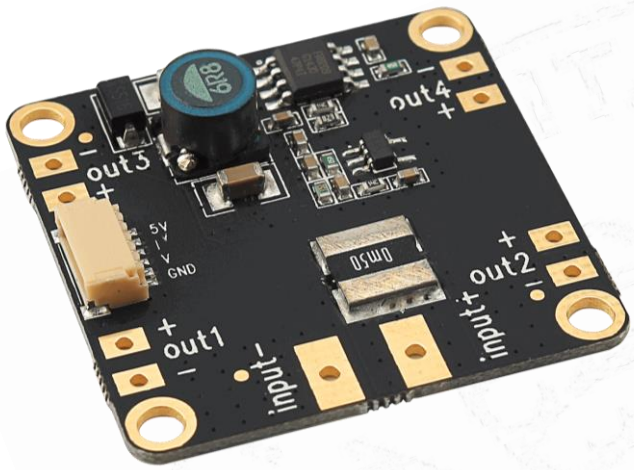


Figura 3.4. Placa de Distribución de Potencia Mini UBEC PDB 4S.

3.1.5 Alimentación

Baterías LiPo

Las baterías constituyen la fuente de alimentación general para toda la electrónica a bordo del cuadricóptero. Se descarta el uso de cualquier fuente externa dado el peligro que conlleva volar un aparato con hélices y que arrastre un cable (aparte de que la zona de vuelo sería bastante limitada). En principio serán necesarias dos baterías: una para alimentar la aeronave y otra para alimentar la emisora de radiofrecuencia. Esta última podría alimentarse desde la red por cable, pero queremos que el conjunto se pueda transportar y volar independientemente del lugar

Figura 3.5. Batería LiPo para alimentar el cuadricóptero.

Para este proyecto se han elegido baterías **LiPo** [7] frente a otras opciones, como pueden ser níquel e hidruro metálico (NiMH), de níquel y cadmio (NiCd), de ion litio (Li-ion), y de litio y ferrofosfato (LiFe). Entre las razones cabe destacar su gran densidad energética (para una determinada masa son capaces de almacenar mayor carga energética), la alta y rápida capacidad de descarga (necesario para maniobras bruscas y, y su largo ciclo de vida (permite numerosas cargas y descargas).

Estas baterías están compuestas de varias celdas en serie de un compuesto metálico, Polímero de iones de litio (LiPo), en nuestro caso serán suficientes 3 celdas. Cada celda tiene su propia tensión nominal y tensión máxima independientemente del resto, ya que la descarga de tensión durante la alimentación no es uniforme para todas ellas. Es por ello que en la recarga de la batería es necesario también hacer un balance de tensión entre las celdas de vez en cuando (ya que existe un umbral límite de tensión mínima por celda, bajo la cual puede quedar inservible, arder o incluso explotar). Por ello es muy importante tener en cuenta que **bajo ningún concepto deben las celdas alcanzar una tensión menor de 3V**. También debe recordarse que es necesario ajustar las baterías a una tensión especial en caso de que vayan a ser almacenadas durante una temporada, ya que en caso contrario puede producirse una evaporación de los electrolitos y la consecuente expansión (inflado) de la carcasa, afectando a la fiabilidad y vida útil de la batería.

Cargador de baterías

El modelo *IMAX* permite la recarga energética de multitud de baterías distintas, de compuestos distintos, o con diverso número de celdas. Contiene multitud de programas para cada batería: *Charge, Discharge, Store, Balance*…por lo que además de cargar la batería, puede funcionar como balanceador o estabilizador. De esta manera permite regular la tensión de cada celda por separado, y así evitar problemas como la sobrecarga, la carga desigual o la descarga de alguna celda. Como además contiene un adaptador de corriente alterna (AC) en su interior, se elimina la necesidad de un transformador y adaptador externo, y puede enchufarse directamente a la red. Por todo ello el cargador *IMAX B6AC* supone el perfecto dispositivo para mantener las baterías cargadas y en un nivel saludable.

Figura 3.6. Cargador-balanceador de baterías IMAX B6AC

3.1.6 Tarjeta de Control

La tarjeta de control es el cerebro operativo del dron. Comunica todos y cada uno de los elementos electrónicos, recibe y manda información, procesa, ejecuta comandos…en definitiva el componente central de la aeronave. Es muy importante elegir una tarjeta lo suficientemente potente para asegurar un periodo de muestreo estable, o de lo contrario se producirán varios errores de lectura de sensores y será bastante más difícil conseguir que el cuadricóptero vuele. Existen multitud de controladoras pensadas especialmente para implantarse en drones, como pueden ser *ArduPilotMega* (APM, basado en Arduino), *OpenPilot* (OP, basado en código *OpenSource*), *PIXHAWK* (PX4, basado en *POSIX*), etc.

Como el proyecto se encuentra centrado respecto al sensor de flujo óptico PX4-FLOW (explicado más adelante), se decidió en un principio optar por emplear la tarjeta PX4 FMU [8] para el control de vuelo. Dicha tarjeta contiene multitud de puertos para telemetría, GPS, Switch de seguridad (failsafe), altavoz, LED RGB programable…en definitiva, una tarjeta bastante completa, robusta, y con multitud de posibilidades diferentes.

Sin embargo, como se descubrió más adelante, la interacción de la PIXHAWK con MATLAB no es del todo tan buena como se deseaba. Y es que aunque los desarrolladores han proporcionado algunas librerías, bloques y scripts para implementar controles con Simulink, dichos bloques son poco o nada personalizables, lo cual supone un conflicto con la idea de hacer nuestro propio control en MATLAB. Por si fuera poco, el Software para PC publicado lleva desactualizado desde el año 2013, por lo que es incompatible con las nuevas y recientes versiones de MATLAB. Al intentar instalarlo, requiere de una actualización manual desde la consola de Windows por GitHub, para descargar ciertos repositorios, que se queda colgada. Esto supone que los bloques de Matlab no son reconocidos como tal, y no funciona cargar el programa.

Figura 3.7. Tarjeta de control Pixhawk PX4 FMU.

Esto no significa que la PIXHAWK sea una mala elección de tarjeta de control, pues cargando el software oficial de la página web el sistema funciona a las mil maravillas. Pero como ese no es el objetivo de este proyecto, tras varias semanas intentando acceder a la programación de la tarjeta de manera infructuosa se decidió que era más conveniente cambiar de tarjeta de control por una que contuviese un código más abierto y sencillo de interactuar. De las dos posibilidades restantes, APM y OpenPilot, se decidió escoger la última ya que existen ya numerosos proyectos de años anteriores basados en el sistema de Arduino, y de esta manera ampliar el conocimiento existente de control de cuadricópteros a otras plataformas.

Esta placa de control 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 [9]. 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.

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 **STM32F405RGT6,** contiene un ARM Cortex-M4 como núcleo a 210MIPS y funciones de saturación DSP *digital signal processor* [10]. 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 SIDO. Todos estos se pueden configurar para acceder desde diferentes pines habilitados para tales efectos.

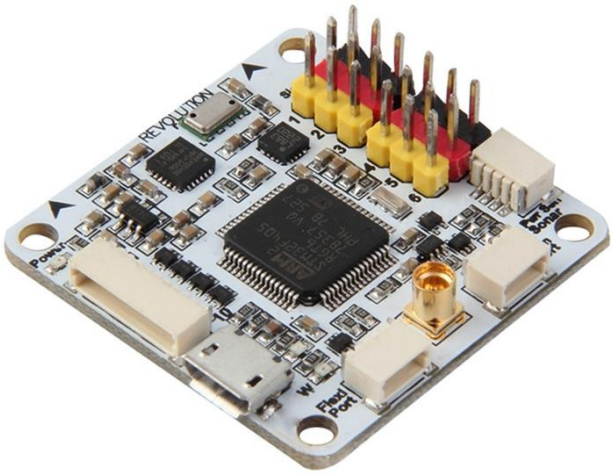


Figura 3.8. Tarjeta de control OpenPilot Revolution.

CARACTERÍSTICAS	ESPECIFICACIONES
Puerto FLEXI-IO de entrada/salida.	Voltaje de entrada: 5 ~ 8.4V
Puerto MAIN (telemetría) USART serie w/, con velocidad de transmisión ajustable.	
Puerto FlexiPort para funciones de telemetría.	Enlace de telemetría: 433 MHz
La salida de RF de 433 MHz.	
Power Sensor (PWR) /Sonar Port.	Dimensiones: 36 x 36 mm
STM32F4 CPU de 32 bits.	
Procesador de paquetes digitales ARM32.	Conector de antena: SMA
USB2.0, I2C, SPI, USART, CAN y SIDO.	
14 Timers de varios canales.	Conectores de alimentación: JST SH-4-pin
Giróscopo de 3 ejes. Acelerómetro de 3 ejes. Magnetómetro de 3 ejes.	
Sensor de presión barométrica.	Peso: 9 g
PWM / PPM	

La tarjeta de control *OpenPilot Revolution* es bastante más pequeña que las otras tarjetas, y no tiene puertos especializados en funciones específicas como era el caso de la PIXHAWK. Cada puerto programable (*FLEXI, MAIN, FLEXI-IO y SERVO/ESC*) puede ser utilizado para un fin distinto, como puede ser recibir datos de sensores, comunicar información al receptor de radio, alimentarse de la batería, etc. De todas maneras determinados puertos están recomendados para ciertas funciones: **SERVO/ESC** para conectar los cables de señal para los motores, **FLEXI-IO** para la lectura de los canales del receptor de radio, etc. El resto de puertos tienen una función especial no programable: **SWD** se utiliza para cargar los programas a funcionar, *PowerSensor* (**PWR**) para alimentar y realizar mediciones de la calidad de la batería, y **RF** para incorporar una antena de radiofrecuencia (deshabilitado en este proyecto).

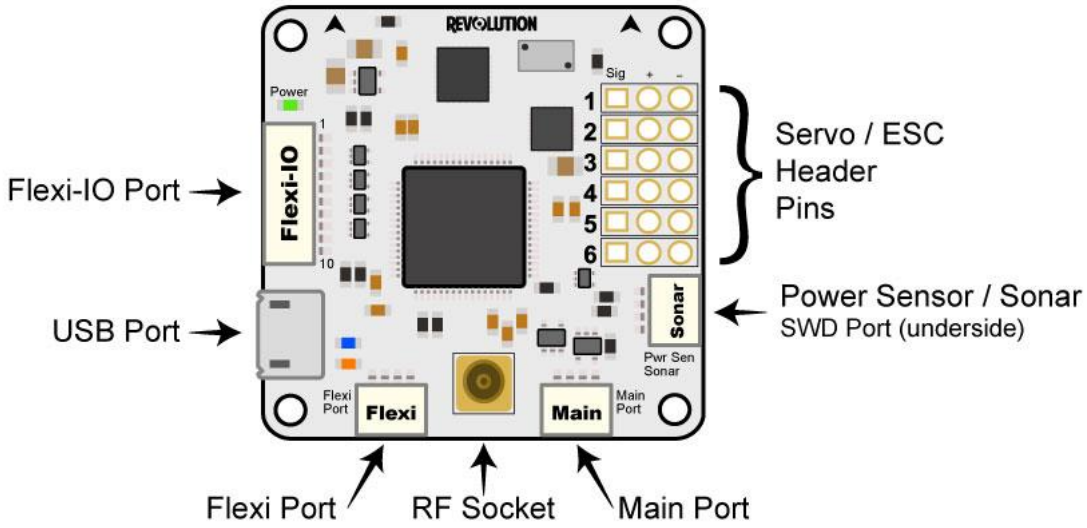


Figura 3.9. Detalle de los puertos de la tarjeta OpenPilot Revolution.

En nuestro caso, vamos a hacer un montaje un poco especial. Debido a que existen determinadas restricciones en la placa con respecto a los *timers* a utilizar, y también porque no queremos que haya conflicto entre ninguno de ellos, hemos programado algunos puertos de manera distinta a la habitual. Los ESC nº 1 y 2 han sido conectados en el puerto SERVO 3 y 4, y los dos restantes en los puertos 5 y 6 del FLEXI-I/O. Los 4 canales de la emisora han sido repartidos de manera equivalente: canales 1 y 2 en los puertos 7 y 8 del FLEXI-I/O, los dos restantes en los puertos 5 y 6 del SERVO. Esto es debido a que los bloques de lectura de PWM y los bloques de escritura de servo no deben compartir *timer*, y en el caso de que conectásemos los ESC a SERVO y los canales al FLEXI-I/O, se produce dicho conflicto. Esto se explicará con más detalle en los apartados *"3.2 Configuración de Puertos de la Placa de Control" y "3.3 Configuración de la Emisora*".

Por otro lado, conectaremos el sensor de flujo óptico en el **FlexiPort**, Las radios Bluetooth para lectura de parámetros en el **MainPort**, el programador USB en el puerto **SWD** y la alimentación de la placa en el **PWR**. El zócalo para la antena RF esta deshabilitado, y no es necesario conectar un sonar adicional pues el sensor ya lleva uno incorporado. A continuación se presenta un diagrama con la vista previa del conexionado [Figura 3.9].

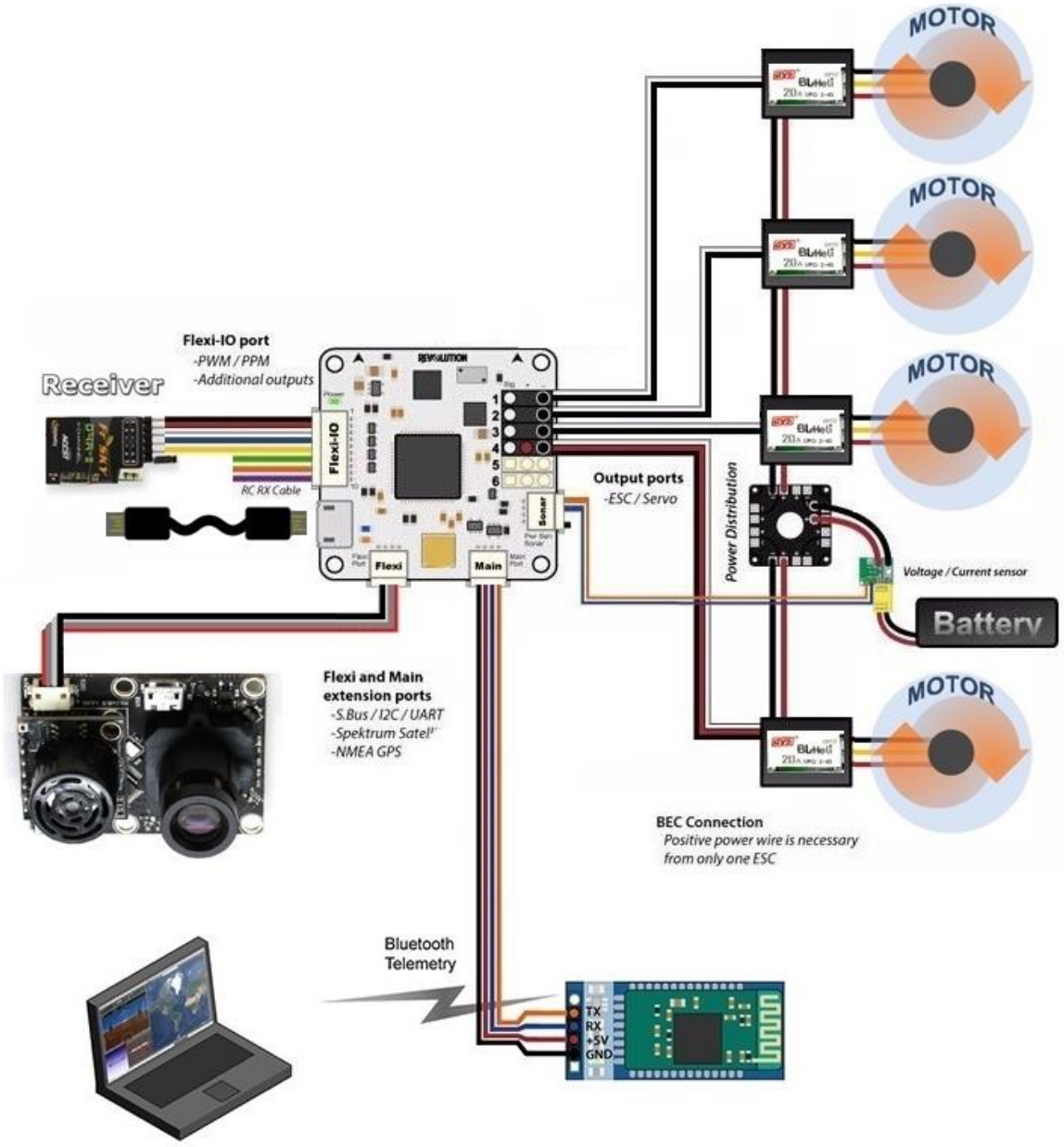


Figura 3.10. Diagrama de conexionado de elementos a la placa de control.

3.1.7 Unidad de medición inercial (IMU)

La IMU, ya comentada anteriormente, consta de diversos elementos microelectrónicos para la medición del estado del cuadricóptero. Se trata del modelo *MPU-9250* de la marca *INVENSENSE*, con un total de 9 ejes (Gyro + Accel + Compass) en solo 3x3x1 mm. En este caso, nos permite conocer en todo momento la orientación angular y la aceleración a la que se ve sometida el dron. Utilizaremos tanto el giróscopo como el acelerómetro integrados en la placa, el magnetómetro no será necesario ya que para la medida de la distancia usaremos el sensor de flujo óptico.

- ***Acelerómetro***

El funcionamiento de los acelerómetros digitales ST-MEMS (*Micro Electro Mechanical Systems*) está basado en **sensores piezoeléctricos**, capaces de medir fuerzas en un solo eje y ser ajenos a las perturbaciones en cualquier dirección ortogonal a este. Esto es posible gracias a pequeños sensores acoplados a un material especial (ferroeléctricos, polímeros polares y determinados cristales como el cuarzo). Al verse sometido a tensiones mecánicas durante la aceleración, la masa del material adquiere polarización eléctrica, y aparece por lo tanto una diferencia de potencial en su superficie (proporcional a la aceleración).

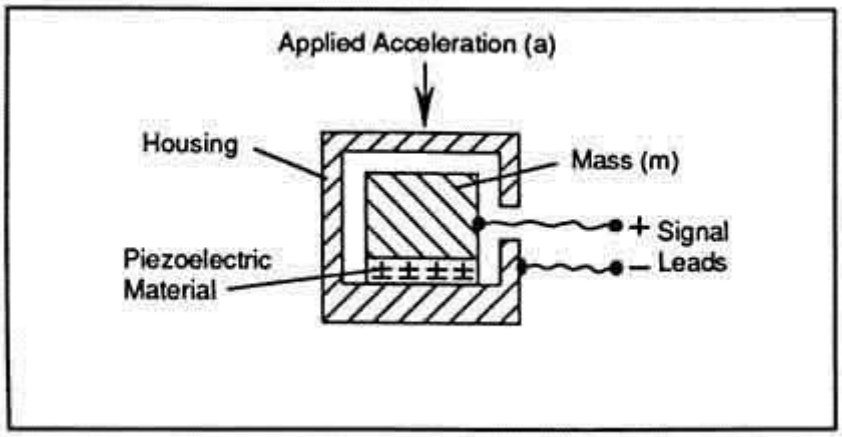


Figura 3.11. Esquema de funcionamiento de un sensor piezoeléctrico.

- ***Giróscopo***

El giróscopo contenido en la unidad de medida inercial es capaz de medir las velocidades angulares del cuerpo con bastante precisión. Se trata de un **Giróscopo de Estructura Vibrante o CVG** (*Coriolis Vibratory Gyroscope*). Su funcionamiento se basa en el efecto Coriolis [11]: un objeto que se encuentra vibrando tiende a seguir vibrando en el mismo plano, aun si su soporte gira. El efecto Coriolis hace que este objeto ejerza una fuerza sobre su soporte, y midiendo dicha fuerza uno es capaz de determinar la fuerza de rotación. Forzamos que el objeto vibre solo en una dirección (x, y, z) , y colocando 3 de estos sensores de forma ortogonal somos capaces de medir el giro en cualquier dirección del espacio.

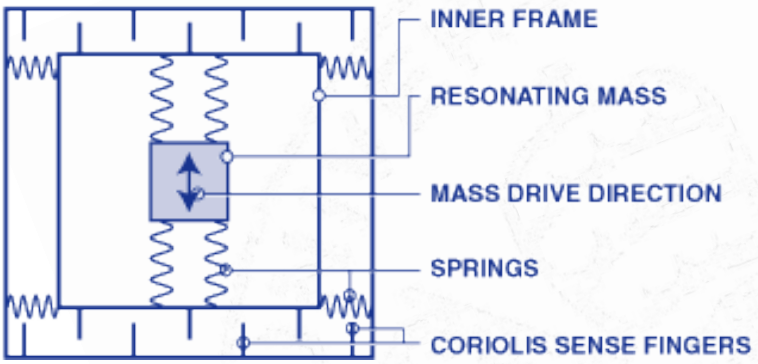


Figura 3.12. Elementos del Sensor de Efecto Coriolis

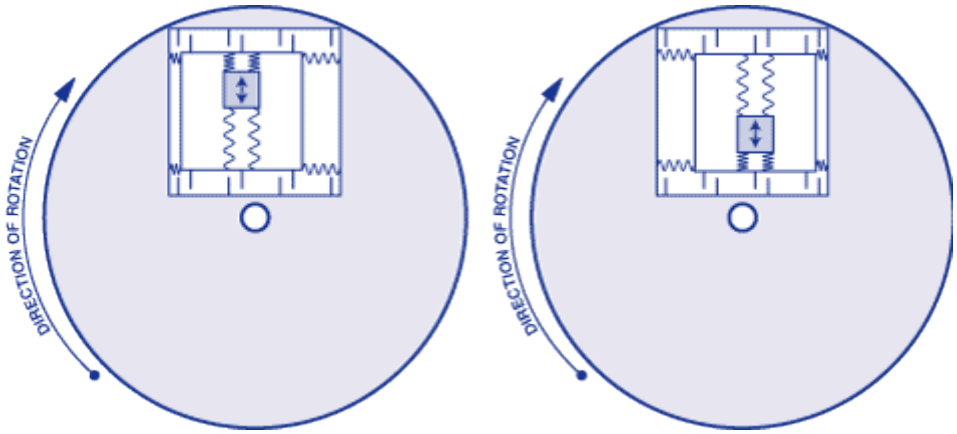


Figura 3.13. Esquema de funcionamiento de un giróscopo CVG.

3.1.8 Sistemas de comunicaciones

El objetivo principal de este proyecto es conseguir un sistema de control para vuelo autónomo de la aeronave. Sin embargo, antes de poder conseguir que el aparato vuele solo, es necesario ser capaz de volar con cierto control manual (para poder determinar y ajustar parámetros para el vuelo). Además no es mala idea conservar un enlace para poder controlar de forma remota al cuadricóptero, en caso de que el control de vuelo autónomo falle.

Para ello contaremos con dos sistemas de comunicación distintos: dron-operario y dron-PC, para el control de vuelo manual y para el vuelo autónomo y lectura de datos respectivamente. La primera de ellas se ha logrado gracias a una emisora de radiocontrol y un receptor colocado en el aparato. Por otro lado conectaremos el cuadricóptero al PC a través de Bluetooth en vez de radio, para evitar que interfieran con la frecuencia de la emisora.

- ***Emisora de Radiocontrol***

Los cuatro canales básicos (o sticks) de toda emisora son: Throttle para controlar el empuje vertical, Aileron para el alabeo, Rudder para la guiñada y Elevator para el cabeceo. A estos cuatro canales básicos se les puede añadir todo tipo de señales analógicas y digitales, como pueden ser interruptores, ajuste de parámetros con ruedas variables, gatillo de apagado de emergencia (failsafe). La emisora a utilizar, la TURNIGY 9XR, permite el uso de hasta 9 canales distintos para enviar señales al cuadricóptero.

- ***Receptor de Radiocontrol***

Pero no vale solo con la emisora, también es necesario colocar un receptor RC en el aparato para la recepción de señales. Dicho receptor debe tener suficientes salidas para cada canal, o una única salida que soporte PPM para sacar todos los pulsos uno detrás de otro por el mismo cable. En nuestro caso usaremos el receptor *FrSKY d8R-II PLUS*, que aunque no tiene opción de PPM, permite salida de hasta 8 canales. En un principio se le acopló un convertidor PPM [12], pero más adelante fue descartado por restricciones de la placa de control (se explicara más adelante junto con el método para enlazar con la emisora).

Figura 3.14. [Izquierda] Emisora RC TURNIGY 9XR

Figura 3.15. [Arriba] Receptor RC FrSKY d8R-II PLUS.

- ***Radios 3DR***

Antes de cambiar a la placa OpenPilot Revolution, se empleaban un par de radios 3DR para lectura de datos y conexión inalámbrica de la PIXHAWK con el PC. Estas radios estas especialmente ajustadas a una frecuencia distinta a la de la emisora de radiocontrol, para evitar que haya conflicto de señales. No obstante el rendimiento de estas radios es bastante inferior al de los dispositivos Bluetooth, y no presentan grandes ventajas sobre los mismos (rango parecido, tamaño un poco más grande, etc.). También podían entrar en conflicto con otras radios 3DR en la cercanía, y sincronizarse con ellas aleatoriamente, lo cual suponía un verdadero quebradero de cabeza para realizar ensayos. Es por ello que más adelante se descartaron a favor del Bluetooth.

Figura 3.16. Emisoras 3DR para enlace con Simulink

- ***Radios Bluetooth HC-05***

En el lugar de las emisoras 3DR se han colocado los cómodos dispositivos Bluetooth HC-05, que ocupan un muy reducido espacio, y tienen un ratio de transmisión bastante elevado (57600 baudios) con una calidad más que aceptable. El único defecto encontrado es que, al igual que las radios 3DR, también puede enlazarse con otras radios bluetooth similares que estén en las cercanías. Por ello no es recomendable tener más de 2 dispositivos (maestro y esclavo) funcionando al mismo tiempo.

Figura 3.17. Radios Bluetooth HC-05

3.1.9 Cámara de Flujo Óptico (PX4FLOW)

El sensor de flujo óptico constituye la piedra angular de este proyecto. Sin él, no se podría plantear la posibilidad de vuelo autónomo, pues las medidas de la IMU y el barómetro no bastan para conocer la posición exacta de la aeronave en un recinto cerrado, solo podemos conocer o estimar la orientación de la misma. Para este fin se va a colocar el sensor de flujo óptico PX4FLOW en la parte inferior del dron, orientado hacia abajo, para que proporcione distintas medidas para el control de vuelo: flujo acumulado, altura desde el suelo, velocidad relativa…

El sensor se puede dividir en dos partes fácilmente diferenciables por su forma. En primer lugar: la **Cámara Óptica**, claramente apreciable gracias a la forma característica de la lente. Es la parte más importante, pues realiza la mayoría de medidas del sensor. Hay que tener especial cuidado con su manipulación ya que es muy delicada, no se debe golpear pues se corre el peligro de desalinear la lente, romper el cristal o los delicados sensores fotoeléctricos del interior. Debe cubrirse la lente con una tapa o cubierta siempre que no se esté utilizando. Y en segundo lugar, el **Sonar**, cuyo desempeño no es nada desdeñable, pues nos proporciona la medida más útil: la distancia al suelo. Esta medida es de una precisión enorme, con un rango fiable entre los 0.3 y los 4 metros. Por debajo de los 0,3 metros el sensor no es capaz de estimar la distancia al suelo debido a la resonancia que producen las ondas y el periodo de muestreo del sensor. Es por ello que será necesario arrancar y despegar el cuadricóptero en modo manual y situarlo a una altura superior a los 0,3 metros para que el control autónomo surta efecto.

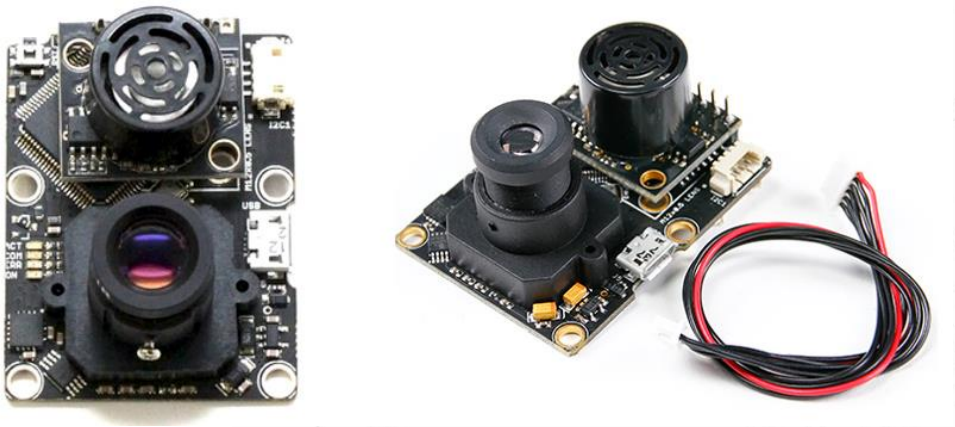


Figura 3.18. Detalle de la PX4FLOW.

El sensor cuenta con un total de 4 puertos de comunicación: un **puerto M**icroUSB para la descarga y actualización del Firmware (también puede transmitir los datos al PC pero no se puede emplear para transmitir a la placa de control), un **puerto I2C** que permite sacar los datos por tramas binarias (trama de ultimo valor de flujo *"i2c_frame"* y trama de flujo acumulado *"i2c_integral_frame"*) y dos **puertos USARTX**, para la transmisión de datos usando paquetes de datos de MAVLink (Micro Air Vehicle Link, protocolo de transmisión de datos muy extendido en cuadricópteros, helicópteros y demás aeronaves teledirigidas). De las dos USART disponibles, la USART3 está pensada para transmitir datos a la tarjeta de control, y la USART2 para enlazar junto con otros sensores de flujo óptico (si los hubiera) y así crear un bus de datos.

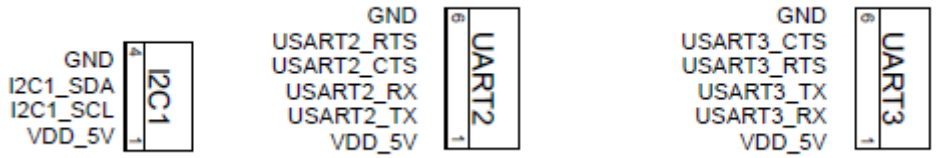


Figura 3.19. Detalle de los puertos del sensor PX4FLOW.

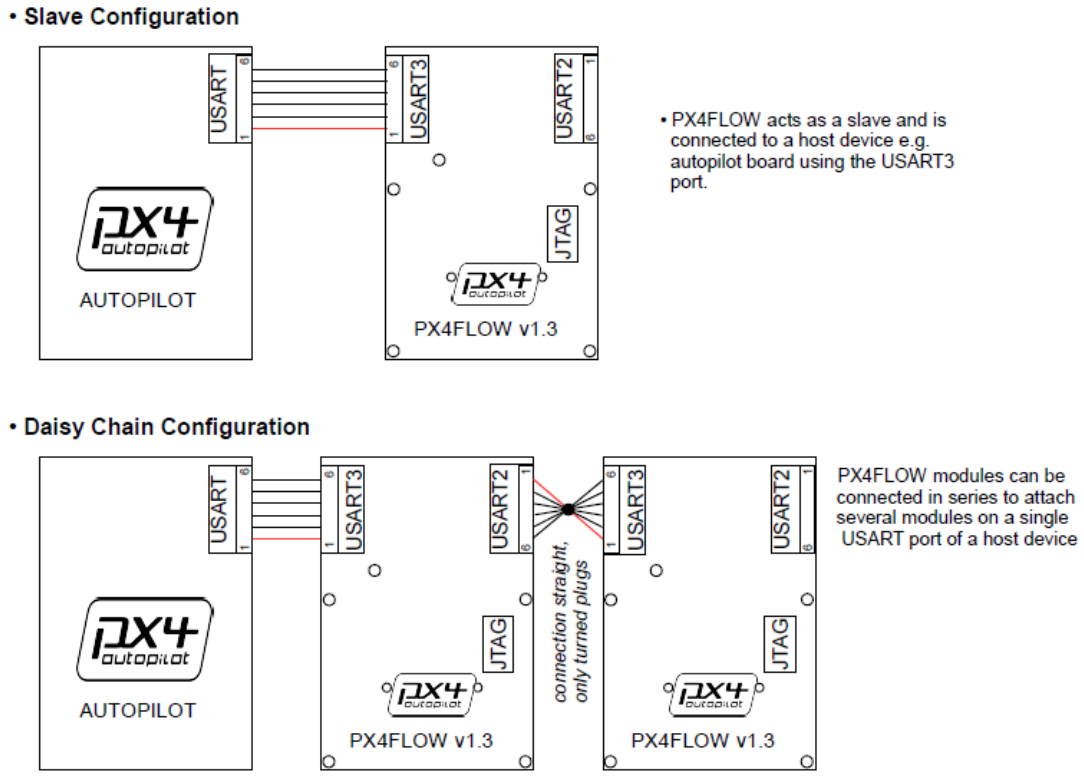


Figura 3.20. Detalle de la conexión en Bus USART para PX4FLOW.

En caso de que hubiésemos mantenido la PIXHAWK, la comunicación Sensor-Tarjeta de Control ya está implantada en el Firmware de la PX4 por lo que simplemente con conectar ambas partes podríamos empezar a leer datos del sensor. Pero al utilizar una controladora diferente (OpenPilot) que no está diseñada especialmente para ser compatible con el sensor, es bastante más difícil conseguir una lectura de los datos en el PC. Los pasos que se han seguido para conseguir una lectura correcta se desarrollaran más adelante, en el apartado "*5.3. Instalación y calibración de la PX4-FLOW*".

3.2 Configuración de Puertos de la Placa de Control

Al estar trabajando con un controlador totalmente distinto a los de años anteriores, se ha hecho un gran esfuerzo en revisar y reconstruir todos los diagramas de Matlab para adaptarlos al nuevo formato. Esto supone empezar de cero, mapear los puertos disponibles, analizar los times, librerías, pines, etc. Y luego básicamente prueba y error hasta dar con la configuración óptima.

3.2.1 IMU

Siguiendo las instrucciones del manual de la placa OP Revo, y consultando en el *schematic* los pines asociados [*Figura 3.21*] se consiguió la relación de pines y *timers* para el correcto funcionamiento de la IMU por conexión SPI [*Tabla 3.1*]. La frecuencia establecida es de 1MHz. Y como más tarde averiguamos (aunque no viene explicado en ningún apartado del manual) la IMU hace uso del **Timer 4**, provocando conflictos con otros componentes que hagan uso del mismo *timer*.

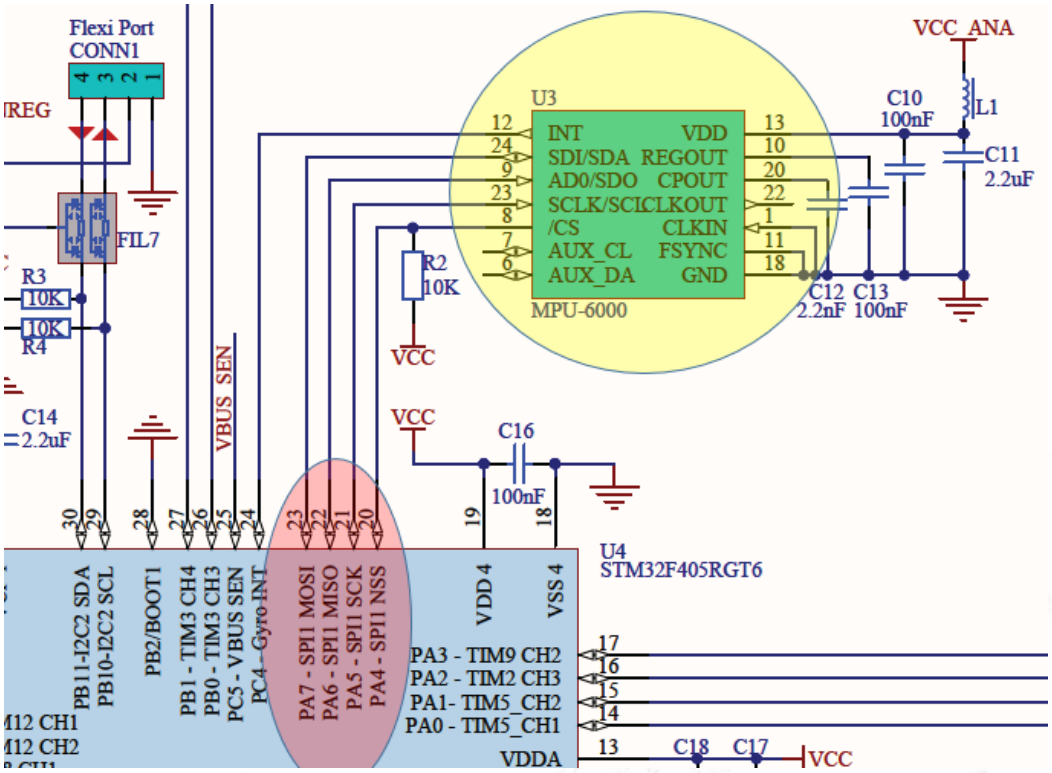


Figura 3.21. Imagen del schematic del OP Revo. En color amarillo la IMU. En color rojo los pines de la placa asociados a la IMU.

Pin	Puerto
A4	NSS
A5	SCK
A6	MISO
A7	MOSI

Tabla 3.1. Conexionado de la IMU

3.2.2 UART (Bluetooth)

Al igual que con la IMU, se han localizado los pines correspondientes a la UART (verificando que tenemos cómodo acceso a los mismo para una fácil conexión y desconexión). De acuerdo con el modelo de conector Bluetooth, se ha establecido un Baud rate de 57600 bps, valor óptimo de velocidad de transmisión para nuestra configuración *master-slave*.

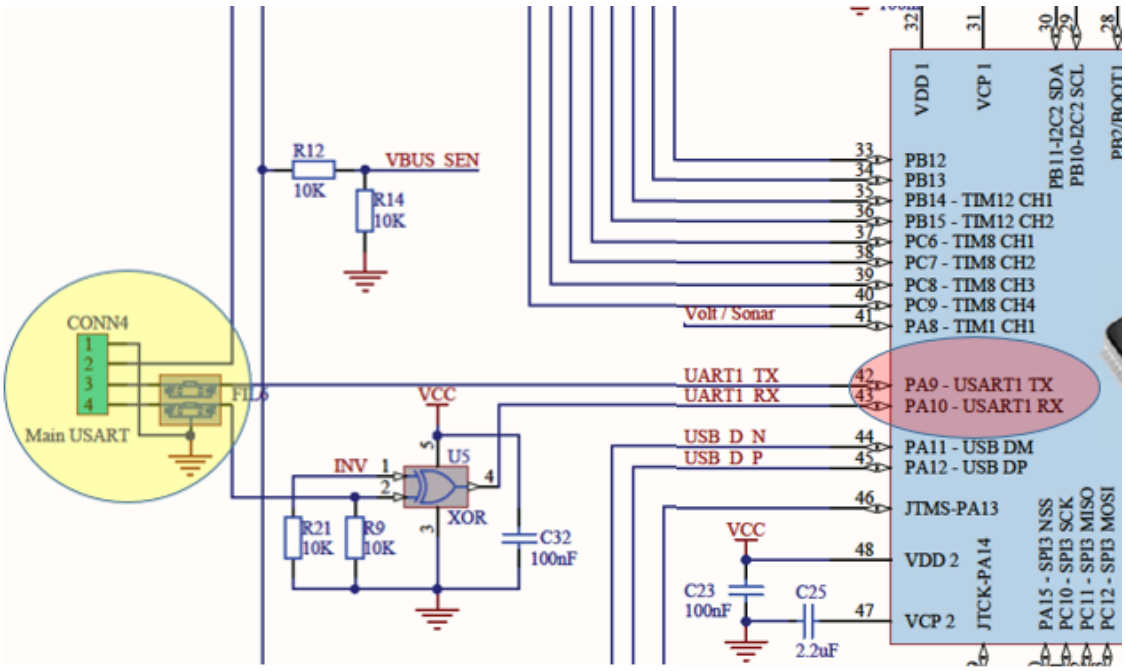


Figura 3.22. Imagen del schematic del OP Revo. En color amarillo el puerto UART (Main). En color rojo los pines de la placa asociados a la UART.

Pin	Puerto
A9	TX
A10	RX

Tabla 3.2. Conexionado de la UART (Puerto Main)

3.2.3 ESCs

En un principio se habían conectado los ESC en los puertos SERVO3, 4, 5 y 6, por comodidad. Pero por motivos que se explicarán más adelante, esta opción se tuvo que descartar por motivo de conflicto de timers con otros componentes. Por ello se han colocado dos (ESC 1 y 2) en puertos SERVO5 y 6 (**Timer 9**), y los restantes (ESC3 y 4) en los puertos 5 y 6 del FLEXI-I/O (**Timer12**). A efectos prácticos este cambio de pines no supone problema alguno, porque aunque los conectores del FLEXI-I/O no pueden alimentar los ESC, ya los estamos alimentando desde la PDB. Nuestra conexión al ESC solo es de señal PWM.

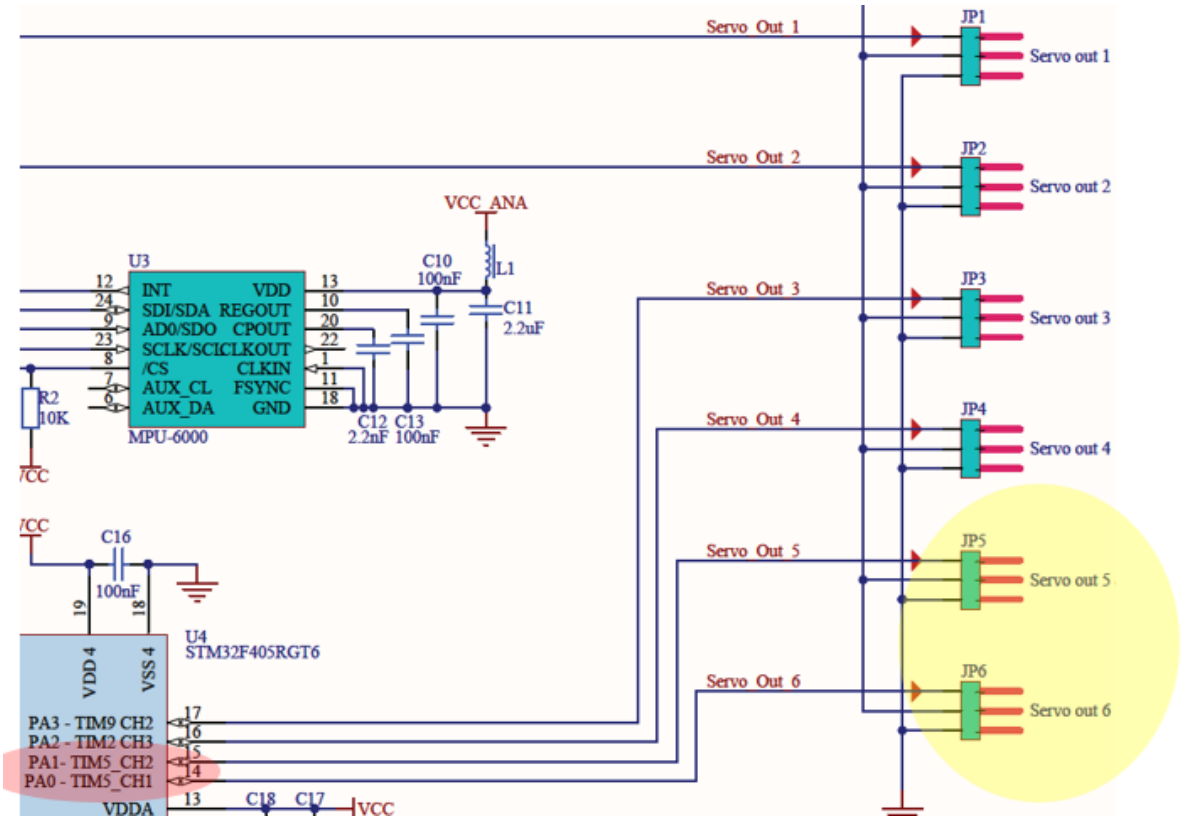


Figura 3.23. Imagen del schematic del OP Revo. En color amarillo los puertos ESC (1 y 2). En color rojo los pines de la placa asociados dichos ESC.

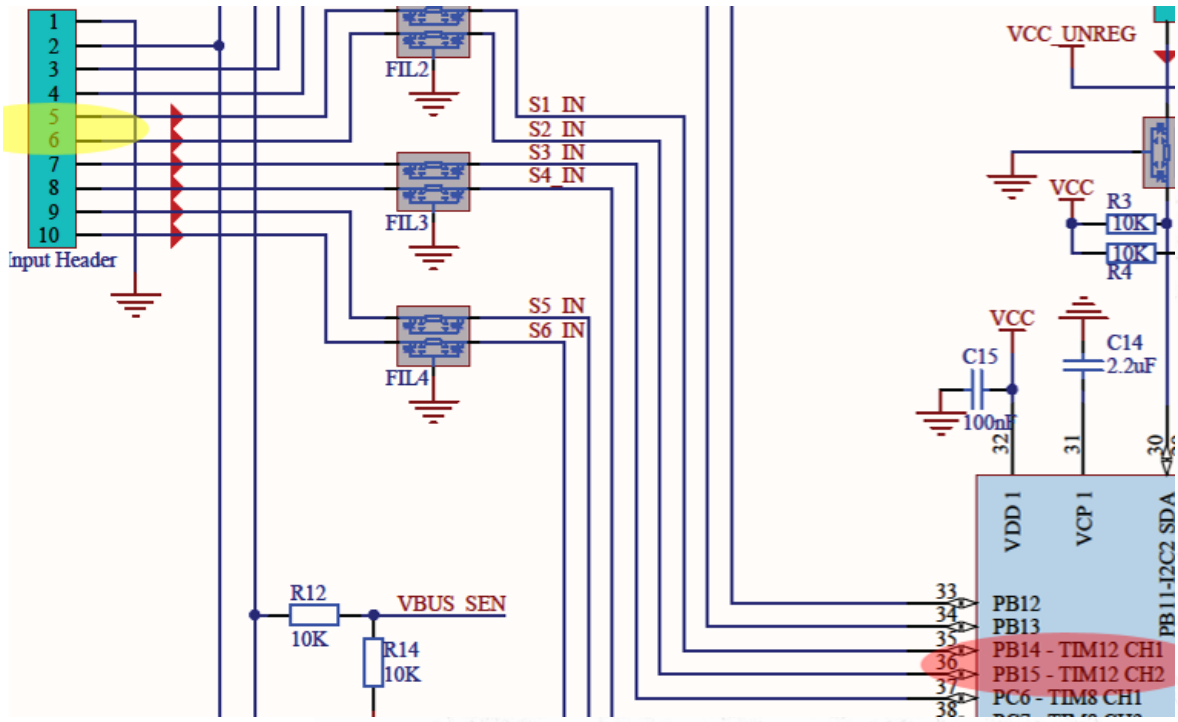


Figura 3.24. Imagen del schematic del OP Revo. En color amarillo los puertos ESC (3 y 4). En color rojo los pines de la placa asociados dichos ESC.

Pin	Puerto	Timer
A2	SERVO OUT 5	Timer 9
A3	SERVO OUT 6	
B14	FLEXI-I/O 5	Timer 12
B15	FLEXI-I/O 6	

Tabla 3.3. Conexionado de los ESC (Puerto SERVO y FLEXI-I/O)

3.2.4 PWR

La conexión al *PowerSensor* de la PDB nos permite alimentar la placa, además de monitorizar el amperaje y la tensión de la batería en todo momento. Aunque el conector de la PDB es de 6 pines y el de la OP Revo es de 4, ambos se pueden conectar. Simplemente, el conector de la PDB tiene duplicados los pines de GND y VCC, por lo que basta con conectar los 4 puertos centrales del PWR y dejar al aire los de los extremos (o usarlos para alimentar otro componente).

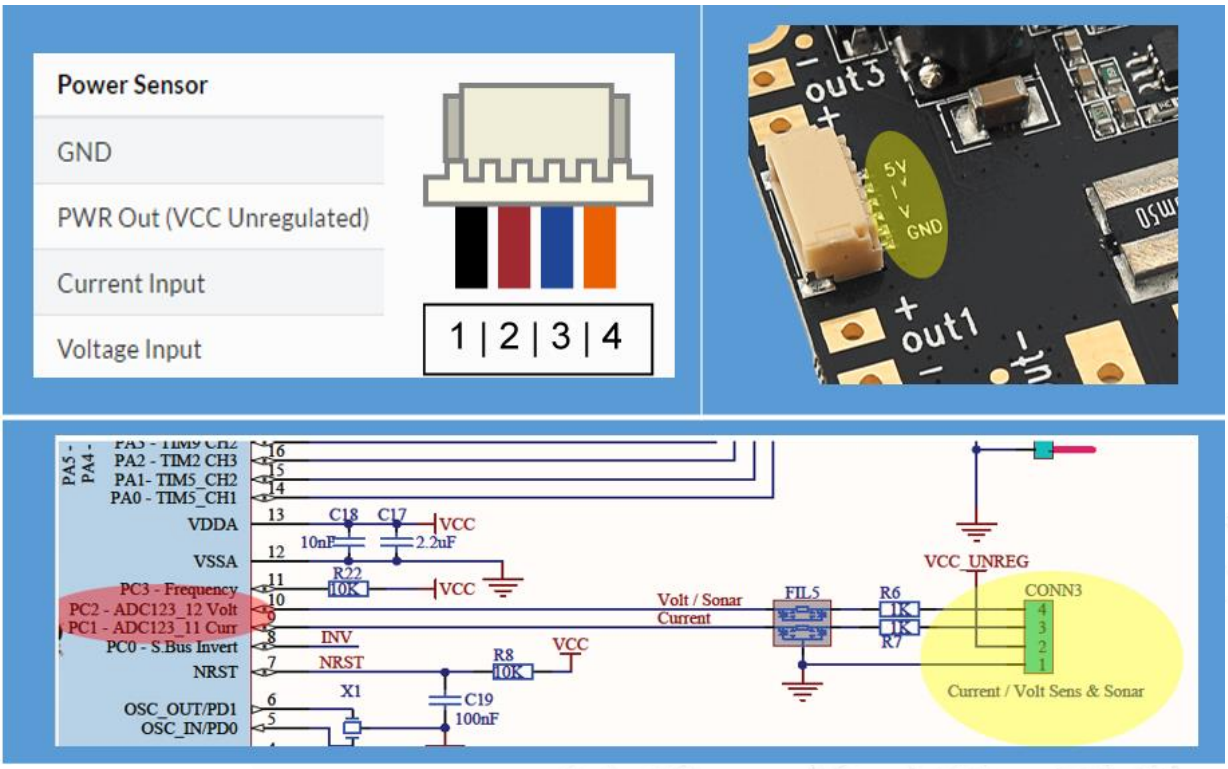


Figura 3.25. [Arriba, Izquierda] Pinout del puerto PWR del OP-Revo.
[Arriba, Derecha]En amarillo, marcados los pines del sensor de la PDB.
[Abajo] En amarillo, marcado el puerto PWR del OP-Revo. En rojo, los pines asociados.

Pin	Puerto
C2	Volt Sens
C1	Current Sens

Tabla 3.4. Conexionado del PowerSensor (PWR)

3.2.5 PWM (Emisora)

Una vez detectado el problema con el conflicto de timers, pasamos a tener solo 4 entradas: los 4 canales básicos de la emisora (*Thrust, Aileron, Rudder y Elevator*). Esto vuelve innecesario el PPM y simplemente podemos conectar las señales de entradas a 4 pines, con cuidad de comprobar que no existe conflicto con los *timers*.

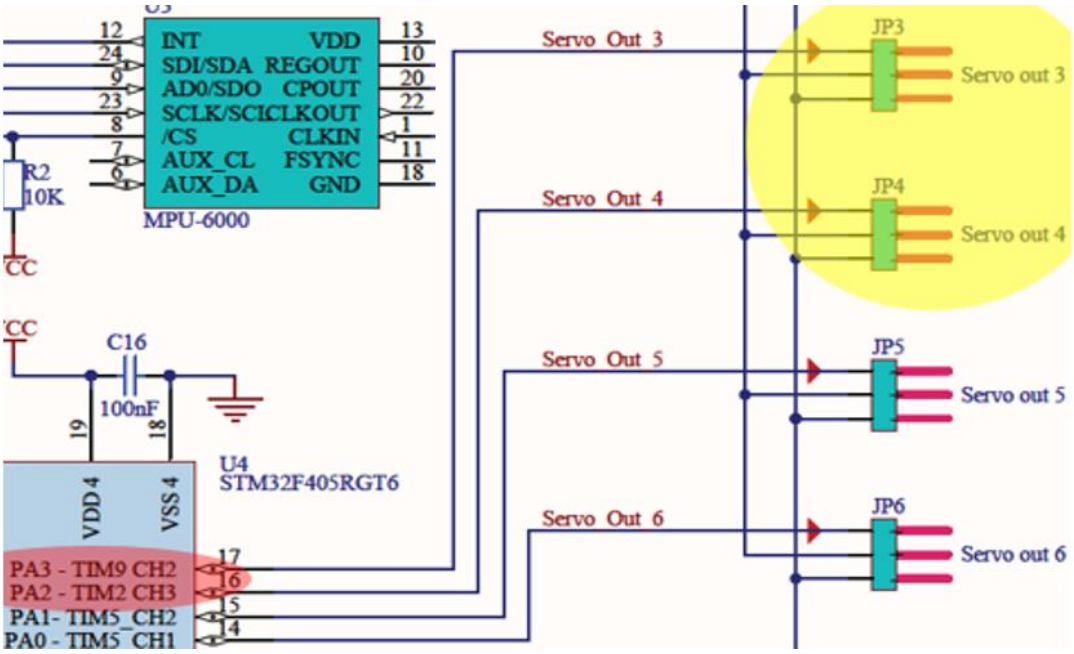


Figura 3.26. Imagen del schematic del OP Revo. En color amarillo los puertos PWM (CH 3 y 4). En color rojo los pines de la placa asociados dichos canales de la emisora.

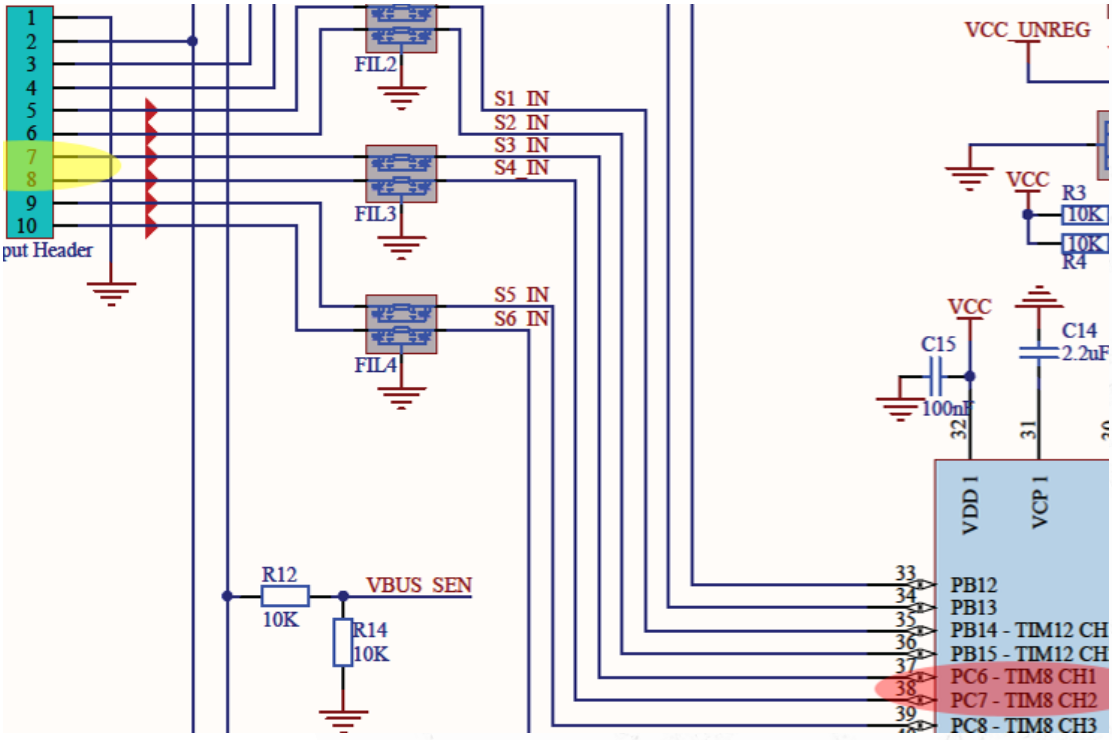


Figura 3.27. Imagen del schematic del OP Revo. En color amarillo los puertos PWM (CH 1 y 2). En color rojo los pines de la placa asociados dichos canales de la emisora.

Pin	Puerto	Canal	Timer
C7	FLEXI-I/O 7	1	Timer 3
C6	FLEXI-I/O 8	2	Timer 8
A0	SERVO OUT 3	3	Timer 2
A1	SERVO OUT 4	4	Timer 5

Tabla 3.5. Conexionado de los canales de la emisora (Puerto SERVO y FLEXI-I/O)

3.2.6 PX4FLOW

El cableado del sensor PX4FLOW va a ser expuesto en el punto "*5.3.2. Alimentación y Cableado*", junto con una imagen de la distribución de pines y puertos en la OP Revo (Consultar Figura 5.5.). Aquí solo vamos a presentar un pequeño resumen en forma de tabla [*Tabla 3.6.*]:

Pin	Puerto
B11	SDA
B10	SCL

Tabla 3.6. Conexionado del sensor PX4FLOW

3.3 Configuración de la emisora

En nuestro caso, la emisora TURNIGY 9XR tiene la posibilidad de combinar diversos mandos, tanto digitales (*switches*) como analógicos (diales o *pots*), hasta un máximo de 8 canales. A lo largo del desarrollo del proyecto se han realizado dos configuraciones de emisora distintas: Por **Modulación de Posición de Pulso** (PPM) de 8 canales a 1 canal, y por **Lectura por canales** (PWM) con 4 canales distintos:

- ***Modulación de Posición de Pulso***

En el estándar usado para transmisión en RC, las señales tienen un pulso comprendido entre los 1 y 2 ms (mínimo y máximo de la señal) para cada canal. Si nuestro controlador tiene menos puertos que canales tiene la emisora, y queremos mandar todos los canales, es necesario un protocolo que agrupe todos los canales en una sola señal que el micro sea capaz de leer (o varias, pero para simplificar las cosas se suelen agrupar todos los pulsos en la misma señal). Esto se consigue gracias a un dispositivo llamado Modulador de Posición de Pulso, aunque cada vez más receptores de RC incorporan este elemento integrado.

En este protocolo los pulsos se colocan uno a continuación de otro, cada 2 ms, sumando un total entre 8 y 16ms (para 8 canales que es el máximo). Se completa la trama con otro pulso, de una duración determinada para cada receptor, que actúa como ***Sync***: permite luego al controlador reconocer cuando acaba una trama y empieza la siguiente. Esto se aprecia fácilmente en la siguiente figura [*Figura 3.28*]:

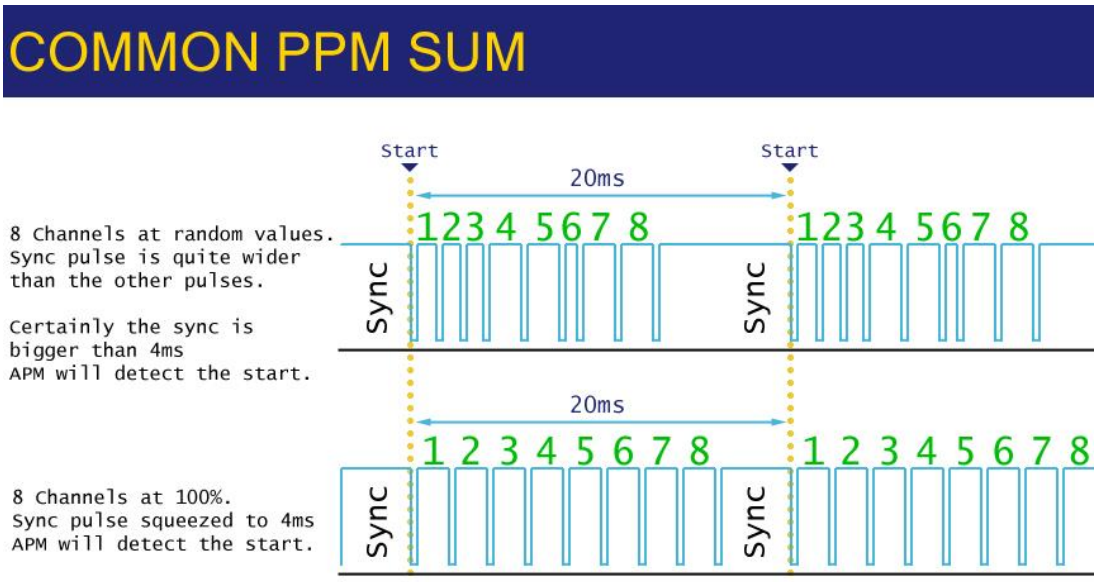


Figura 3.28. Imagen explicando el funcionamiento del PPM.

Sin embargo, nos topamos con un ligero inconveniente: para nuestro receptor, *el FrSky d8R-II plus*, los desarrolladores han establecido el ancho de la trama total en 18ms. Esto significa que el *sync* para el caso limite (todas las señales al 100%) tiene un ancho de 2ms, por lo que el controlador verá imposible detectar el inicio y fin de la trama PPM [*Figura 3.29*]:

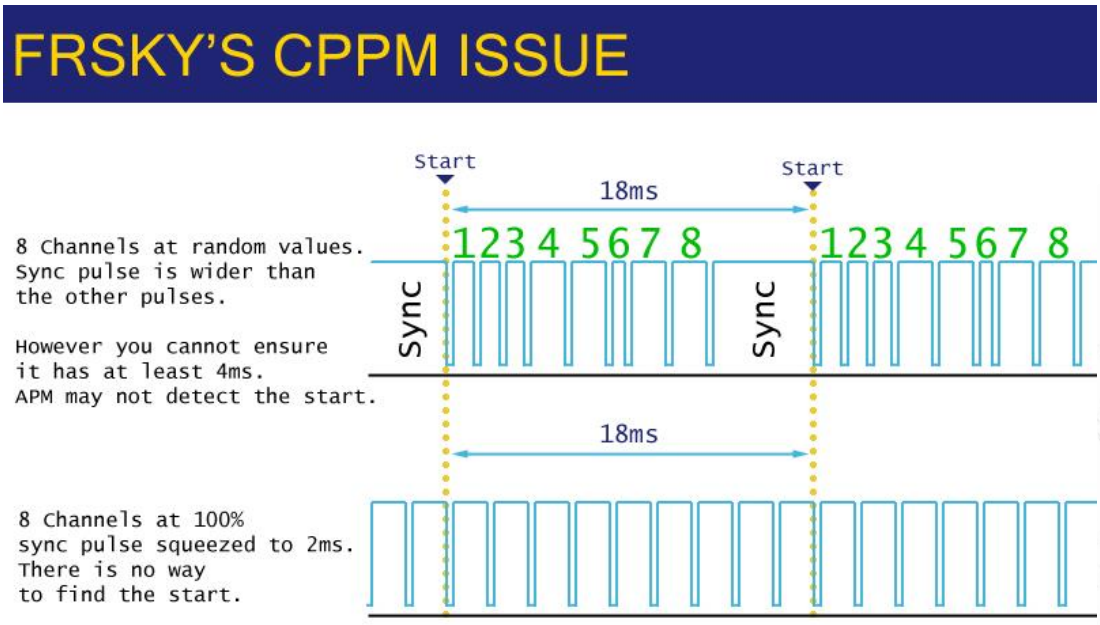


Figura 3.29. Problema encontrado para detectar las tramas en nuestro receptor RC.

Este problema se solucionó forzando a que una de las señales (normalmente el canal 7 u el 8) tuviese un ancho de 1ms, de modo que nunca se llegase al caso donde el ***sync*** se confunde con los canales. Una idea sencilla y práctica a la vez, poro que necesita que se sacrifique un canal para que el resto funcionen. La otra opción consiste en *flashear* (descargar e instalar) un Firmware modificado para alterar el ancho de la trama por defecto (pasarlo de 18 a 20ms). Pero nos arriesgamos a dejar el dispositivo estropeado si algo sale mal en el proceso de actualización, así que optamos por no tocar nada de la configuración interna del dispositivo.

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.

- ***Lectura por canales***

Debido a que no se usará el módulo PPM del receptor, los problemas explicados en el apartado anterior, se pasará de 8 canales previstos a realizar la lectura directa de los canales de la emisora. Como efecto negativo, al controlador solo le quedan disponibles 4 timers para la lectura de pulsos, por lo que solo podremos leer 4 canales. Esto obliga a rehacer la máquina de estados, de la cual se hablará en el **Capítulo 6:** ***"6.3 Máquinas de Estados"***. 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.

En la siguiente figura, sacada del manual de usuario, se muestran todos los diferentes mandos indicados con sus respectivas etiquetas.

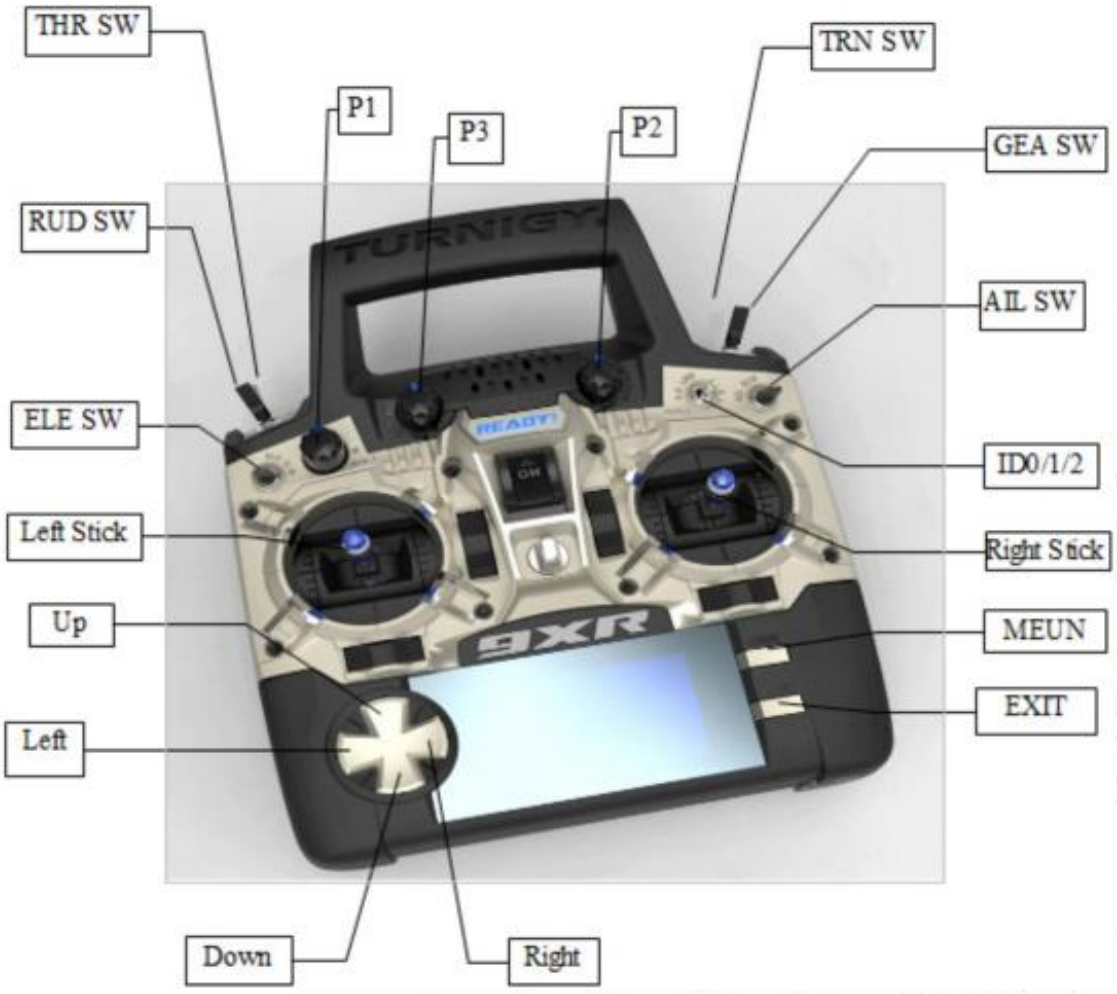


Figura 3.30. Mandos de la emisora.

Los mandos se pueden clasificar en ***sticks***, que varían su valor de manera analógica al desplazarlos linealmente; ***pots***, equivalentes a los sticks pero con forma circular y que varían su valor al girarlos, y ***switches***, que tienen dos o tres posiciones que funcionan como señales digitales. Por continuidad con el uso tradicional de estas emisoras, para que no haya cambios a la hora de manejar el cuadricóptero en vuelo manual, se han reservado los 2 sticks para transmitir las referencias de alabeo, cabeceo, empuje y guiñada. Estas referencias se envían íntegramente por los 4 primeros canales, en ese orden. Y no se han configurado más canales a falta de timers para la lectura en el controlador.

Capítulo 4

Control del Cuadricóptero

4.1 Métodos de control

De las numerosas técnicas de control diseñadas para sistemas de aeronaves, es necesario conseguir una que sirva para un sistema de planta altamente inestable como lo es un cuadricóptero. Estas estrategias van desde el control proporcional más sencillo, hasta el complejo control predictivo. Pero no por usar un control más complicado aseguramos una mayor estabilidad, pues a mayor complejidad mayor es también el tiempo de muestreo necesario. Por lo que estaríamos sacrificando tiempo de reacción frente a las perturbaciones, y en casos extremos esto es suficiente para volver inestable el control. A continuación se expondrán los controles más comúnmente usados en este tipo de situaciones:

4.1.1 PID

En un control PID simplemente aplicamos una acción Proporcional-Integral-Diferencial al error de seguimiento, es decir, a la resta de la referencia menos la medida (o la estimación). Cada parte se encarga de corregir una serie de factores:

- ✓ La acción **Proporcional** se encarga de compensar el error de seguimiento. A mayor error, más intensidad tiene dicha acción.

- ✓ La acción **Integral** se encarga de ir acumulando el error para compensarlo completamente, aunque a expensas de la velocidad del control (es más lento que el resto, aunque más preciso).

- ✓ La acción **Diferencial** o **Derivativa** que se encarga de generar una tendencia rápida hacia el valor final, generando un mando proporcional a la derivada del error, aunque esta vez a costa de amplificar también el ruido de la señal del error (alta frecuencia). Por ello es conveniente acompañarlo de un filtro paso bajo.

Se trata de un control bastante simple, que si bien no siempre asegura estabilidad frente a perturbaciones, ha resultado funcionar de manera satisfactoria en el **Control de Estabilidad** (*Attitude Control*) [13]. Para la simulación se ha linealizado el modelo, con el punto de operación correspondiente al de vuelo estacionario, aunque de nuevo sigue sin tener mucha robustez frente a grandes perturbaciones.

4.1.2 LQR

El control LQR (**Linear-Quadratic Regulator**) se trata de un sistema basado en ajustar el punto de operación en base a unas ponderaciones predefinidas por el usuario. Dicho ajuste se hace variando los valores de la matriz de realimentación de estados, con el objetivo de minimizar las desviaciones respecto a las especificaciones. Implementado inicialmente a velocidades bajas por Hoffman [14], fue optimizado más adelante por Cowling [15] que consiguió lograr seguimiento de trayectorias. Más tarde Bouabdallah [16] lo aplicó para la implementación de sistemas de estados independientes, y así calcular la matriz a partir del estado y no del punto de operación. Un gran avance en autonomía y robustez, no restringe las trayectorias a vuelos cuasiestacionarios.

4.1.3 Realimentación de Estado

Como hemos visto anteriormente, las ecuaciones que rigen el comportamiento dinámico de un cuadricóptero contienen términos donde algunas de las variables de estado se multiplican entre sí, creando términos no lineales. Para el diseño de controles en plantas no lineales es necesario la creación de variables intermedias, que tengan un comportamiento lineal respecto a la salida (actuadores). Esto supone una conversión del mando inicial al mando intermedio, donde se acumulan todos los términos no lineales basados en el modelo previo. Dicho sistemas recibe el nombre de **Linealización por Realimentación de Estado** o *Feedback Linealization.*

Para la implementación del control se ha utilizado el siguiente vector de estados, con resultados satisfactorios:

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

Así conseguimos eliminar la relación no lineal presente entre las velocidades angulares derivadas y los ángulos de Euler. El control resultante es equivalente a un PD, ya que realimenta tanto las velocidades angulares como los propios ángulos de Euler. El control por Realimentación de Estado fue empleado como modelo de control de estabilización en proyectos del año pasado, aunque en este proyecto se ha descartado en favor del Control PID, que ha terminado arrojando resultados más favorables.

4.1.4 Controles Alternativos

Además de los clásicos métodos de control ya vistos, se han explorado otros controles mucho más potentes y robustos, aunque debido a su complejidad se han terminado descartando. Métodos tales como los basados en Control Borroso (*Fuzzy Controller*) [17] y Lógica Difusa (*Fuzzy Logic*) [18]. Y no olvidarnos del potente Control Predictivo [19], especialmente interesante para aplicaciones que presenten la necesaria potencia de cálculo (utiliza el método de mínimos cuadrados para optimizar el error entre la referencia y la estimación).

4.2 Estimadores de estado

El estimador o estimadores de estado, presentes en el controlador de una aeronave, juegan un papel importante para asegurar la estabilidad del control. Y es que durante todo el proceso de control es necesario realizar mediciones de algunas de las variables a controlar, como por ejemplo los ángulos de Euler en nuestro caso. Sin embargo, debido a una serie de circunstancias ajenas a nosotros (vibración en los motores, naturaleza de los sensores, etc.), dichas variables se presentan imposibles de medir o presentan un gran índice de ruido, volviéndolas inservibles si queremos logra estabilidad en el control. Para evitar estos problemas, se recurre a los estimadores de estado.

Un estimador de estado se encarga, como su propio nombre indica, de estimar el valor real de una variable, en lugar de tomar una medida. Los tres tipos de estimadores de estado se exponen a continuación:

- ✓ El **Estimador de Lazo Abierto** únicamente tiene en cuenta los mandos, y se basa en un modelo de la planta (de ahí su nombre).

- ✓ El **Observador de Orden Reducido** emplea algunas de las salidas y mandos para estimar mejor el estado del sistema.

- ✓ Cuando se emplean todas y cada una de las salidas y los mandos, estamos en el caso de un **Observador de Orden Completo**.

En concreto en el caso de los cuadricópteros es de vital importancia estimar los ángulos de Euler, medida que no podemos obtener únicamente a partir de las velocidades angulares y aceleraciones lineales proporcionadas por el giróscopo y el acelerómetro (respectivamente) Los estimadores anteriores no son suficientes, y se emplean en su lugar otros que permiten filtrar y combinar las medidas anteriores. Se trata del **Filtro de Kalman** y el **Filtro Complementario**. Concretamente en este proyecto se va a emplear el **Filtro Complementario No Lineal**.

4.2.1 Filtro de Kalman

El filtro de Kalman consiste en un observador en tiempo discreto recursivo, o lo que es lo mismo, un estimador de estado en tiempo real. Su principal diferencia respecto a otros estimadores es que, mientras estos tienen una matriz de ganancias K utilizada para estimar las variables de estado no medibles, el Filtro de Kalman hace uso de una **matriz de covarianzas P** para generar automáticamente la matriz K óptima. Y dicho proceso funciona de la siguiente manera:

I. Primero existe una fase de predicción de resultados. En dicha fase se generar una estimación "a priori" de la matriz P, basándose en parámetros como el modelo y el estado anterior.

II. A continuación una fase de corrección de resultados, donde se calculan los valores de la matriz K a partir de la ya estimada matriz P, a partir del error entre la medida y el modelo.

III. Por último, se actualizan tanto la matriz de covarianzas (para volver a ser usada en la siguiente iteración) como la estimación del estado (añadiendo el nuevo error multiplicado por la ganancia).

Queda destacar el hecho de que dicho sistema solo es aplicable a sistemas lineales. Debido a la presencia de elementos no lineales en la dinámica de los cuadricópteros, el Filtro de Kalman en principio no sería válido como estimador de estados. Pero para ello se emplea el **Filtro de Kalman Extendido**, que añade una linealización al modelo mediante aproximación de Taylor en cada iteración [20].

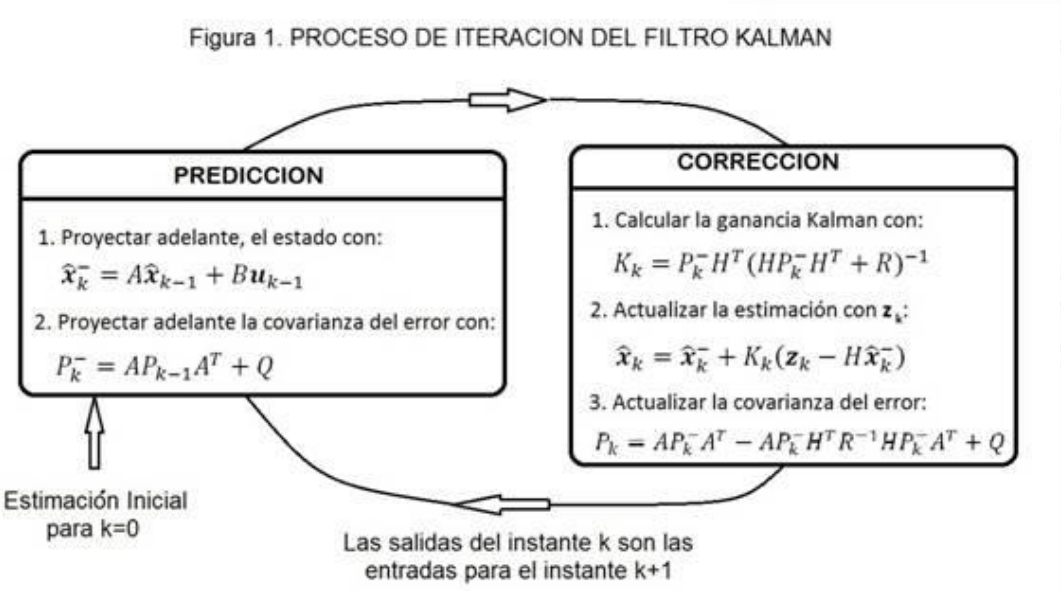


Figura 4.1. Esquema del funcionamiento del Filtro de Kalman.

4.2.2 Filtro complementario

En vez de recurrir a estimadores tan complejos, uno podría aproximar los ángulos de Euler a partir de las velocidades angulares, mediante uno o varios integradores. Y por otro lado a partir de las velocidades en ejes solidarios al cuerpo estimar las velocidades angulares en función de dichos ángulos. El problema radica en el pequeño error de medición que introducen los giróscopos (normalmente debido a la temperatura), que si bien es un valor bastante pequeño normalmente, integrar constantemente supone acumular un error cada vez más grande en la estimación.

De manera análoga, se puede conocer la orientación del cuadricóptero a partir de las aceleraciones lineales por la proyección de la gravedad. Pero esto tiene gran el inconveniente de que no se tiene en cuenta el hecho de que al estimar los ángulos de esta manera, las propias aceleraciones del cuadricóptero alteran el valor real de las proyecciones de la gravedad. Por lo tanto este método tampoco es válido.

Para evitar estos problemas, el **Filtro Complementario** [21] combina las medidas de tanto el giróscopo como del acelerómetro, eliminando aquellas componentes que nos perturban la medida. Para ello se coloca un filtro paso alto para eliminar el error de medida de los giróscopos (se supone no varía a temperatura constante) y un filtro paso alto para los acelerómetros (para eliminar todos los valores menos la gravedad, que es un valor fijo).

Este filtro se hace especialmente útil en la estimación de variables de estado de la Unidad de Medición Inercial (IMU) [22] ya que se obtiene la medida de aceleración lineal y angular libre de errores, previo filtro paso alto en los giróscopos y paso bajo en acelerómetros.

4.3 Diseño del Control

A continuación se van a exponer los distintos pasos llevado a cabo para el diseño de las diferentes partes involucradas en el control del cuadricóptero. Los elementos se han ido diseñando de forma secuencial, siguiendo el sentido marcado por el flujo de la información dentro del control: Primero el módulo de calibrado de la IMU, pasando al Estimador de Estados, luego al Control de Estabilidad y por último el Control de Navegación.

4.3.1 Calibrado de la IMU

Para conseguir obtener medidas fiables de los sensores de la IMU, sabiendo el error que introducen giróscopos y acelerómetros, se ha diseñado un sistema de bloques que consiste de las siguientes partes: **Filtro Paso Alto** para los giróscopos y **Filtro Paso Bajo** para los acelerómetros. Esto permite reducir de manera significativa el offset de ambas medidas, eliminándolo casi por completo al finalizar el periodo de calibración de 20 segundos.

El bloque de calibrado de la IMU funciona de la siguiente manera:

I. Primero se obtienen las medidas de giróscopos (GX, GY, GZ) y acelerómetros (AZ, AY, AZ) de la IMU. Estas señales se agrupan en dos grupos: GYRO por un lado, ACCEL por el otro.

II. Segundo, se obtienen las señales que inician y desactivan la calibración CAL_ENABLE y CAL_STOP. Estas señales son generadas desde la máquina de estados: el *Enable* cuando el usuario da el comando de entrar en el estado de calibración, y el *Stop* cuando pasan 20 segundos de calibración.

La imagen de la estructura interna del bloque de calibración se puede ver a continuación:

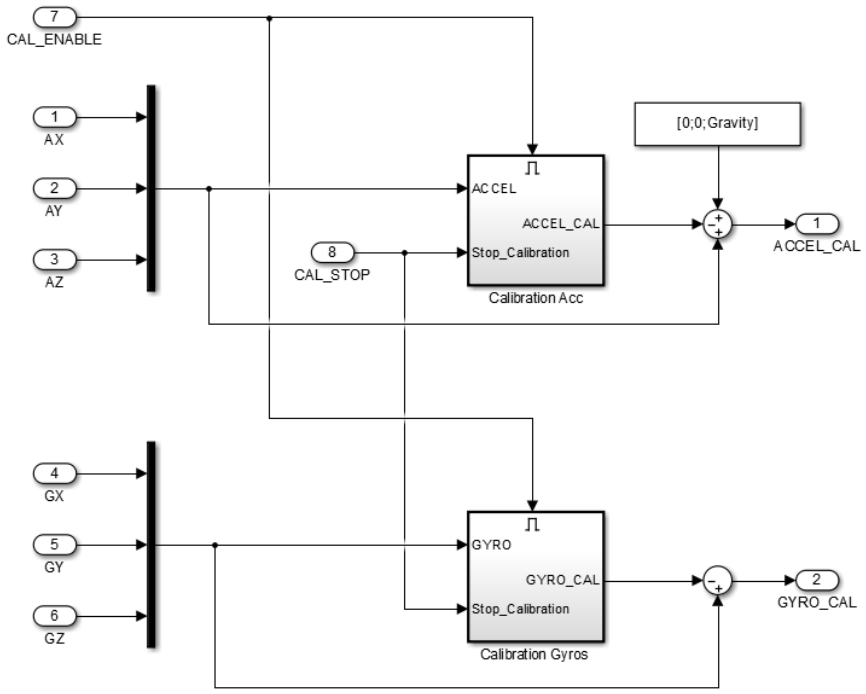


Figura 4.2. Bloque de calibrado de la IMU.

Se debe tener especial cuidado de comprobar que las aceleraciones lineales y angulares de la IMU son consecuentes con la orientación de la aeronave. En nuestro caso debe haber algún problema con algunas medidas internas de la IMU, ya que nos sale un valor correcto de aceleración pero en sentido contrario al esperado. Esto puede ser debido a algún fallo en la lectura de los valores de los registros de la IMU, que los sensores estén colocados al revés, etc. Para corregir dicho error basta con colocar ganancias de valor "-1" en aquellos valores que lo necesiten, tras realizar un ensayo de giróscopos y acelerómetros (esta corrección se aplica antes del calibrado).

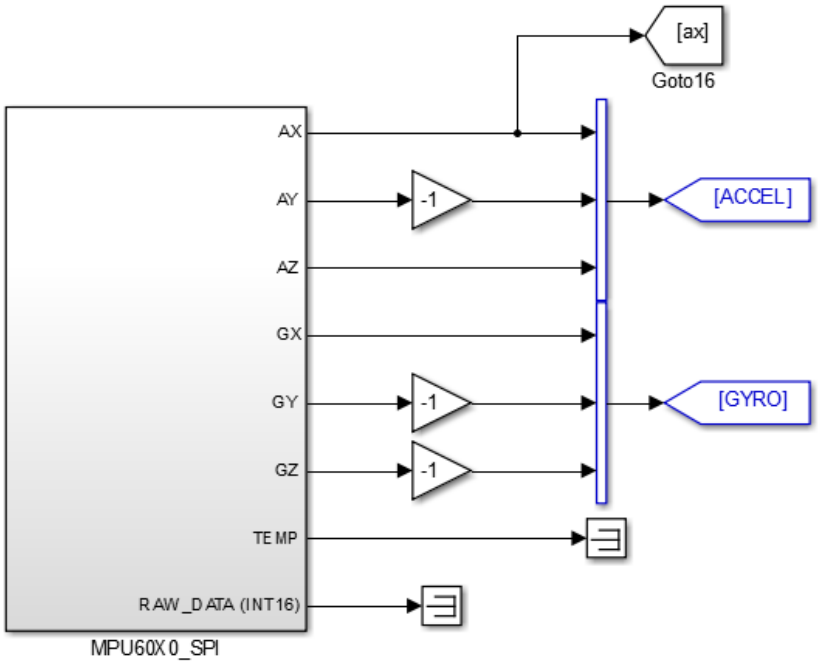


Figura 4.3. Corrección de las medidas de la IMU.

4.3.2 Estimación por Filtro Complementario No Lineal

Como ya se ha explicado anteriormente, el Filtro Complementario permite obtener los ángulos de Euler estimados. Por ello recibe las medidas de la IMU ya calibradas, y mediante la combinación lineal de las mismas se normalizan y se consigue estimar los ángulos reales.

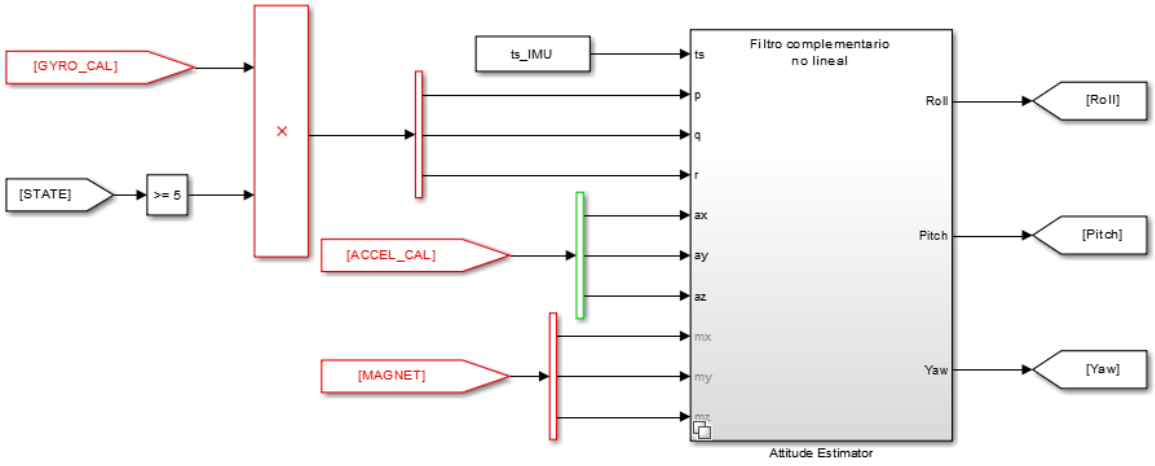


Figura 4.4. Imagen del Estimador de Estados.

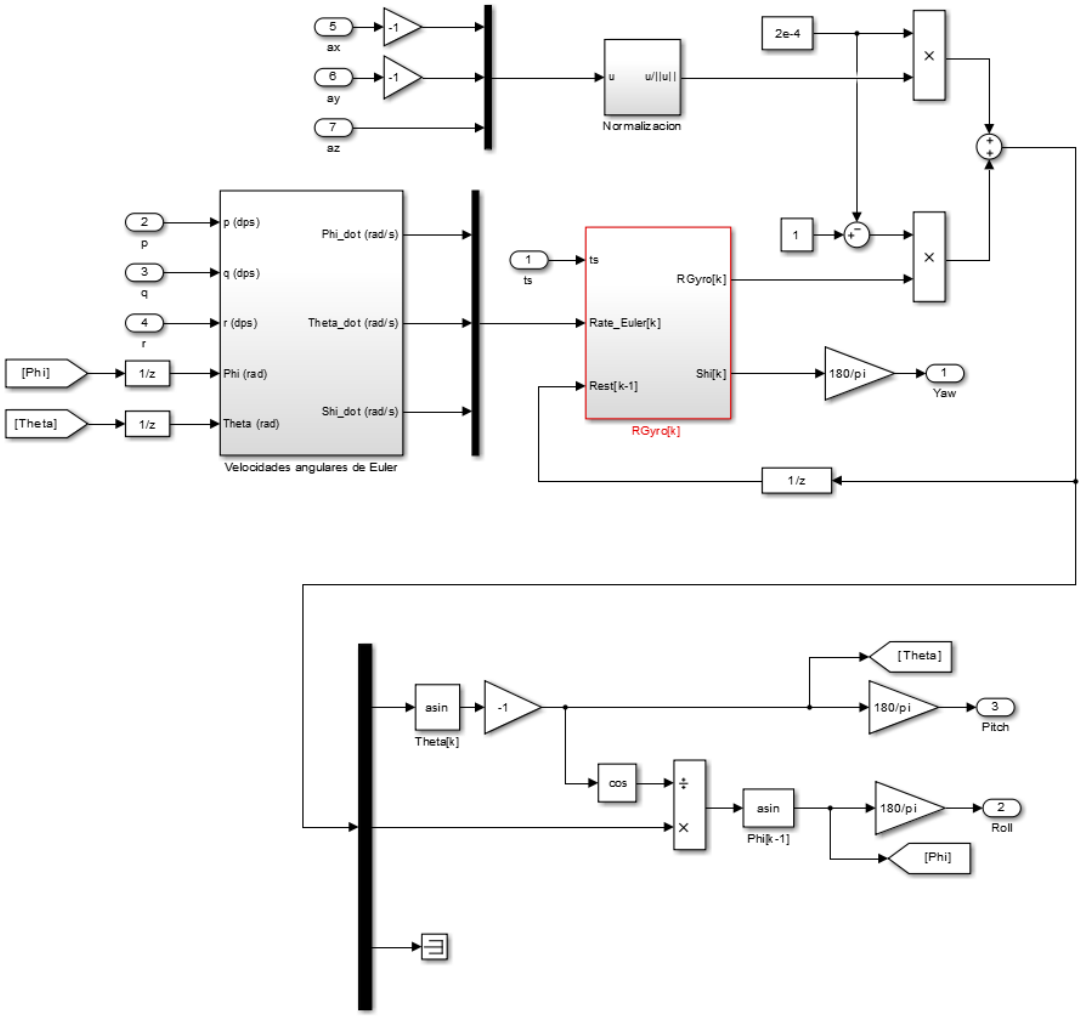


Figura 4.5. Estructura interna del Filtro Complementario No Lineal.

4.3.3 Control de Estabilización

El Control de Estabilización utilizado en este proyecto consiste en un **control PID modificado** [23]. En su interior se realimentan tanto los ángulos como las velocidades angulares en los 3 ejes, multiplicando por sus respectivas ganancias y añadiendo términos no lineales. Las derivadas de los ángulos de Euler no llegan a coincidir con las velocidades angulares registradas por los giróscopos, pero la diferencia es ínfima para pequeñas inclinaciones en tanto alabeo como 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

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

Esta definición es una versión reducida de la mostrada en la Ecuación (4.1). Las ecuaciones de estado son las mostradas en (2.14) despreciando los términos giroscópicos debido a la rotación de las palas, y los contrapares durante su aceleración, gracias a que los momentos de inercia de los motores son suficientemente pequeños (cuando el cuadricóptero está en su fase de vuelo).

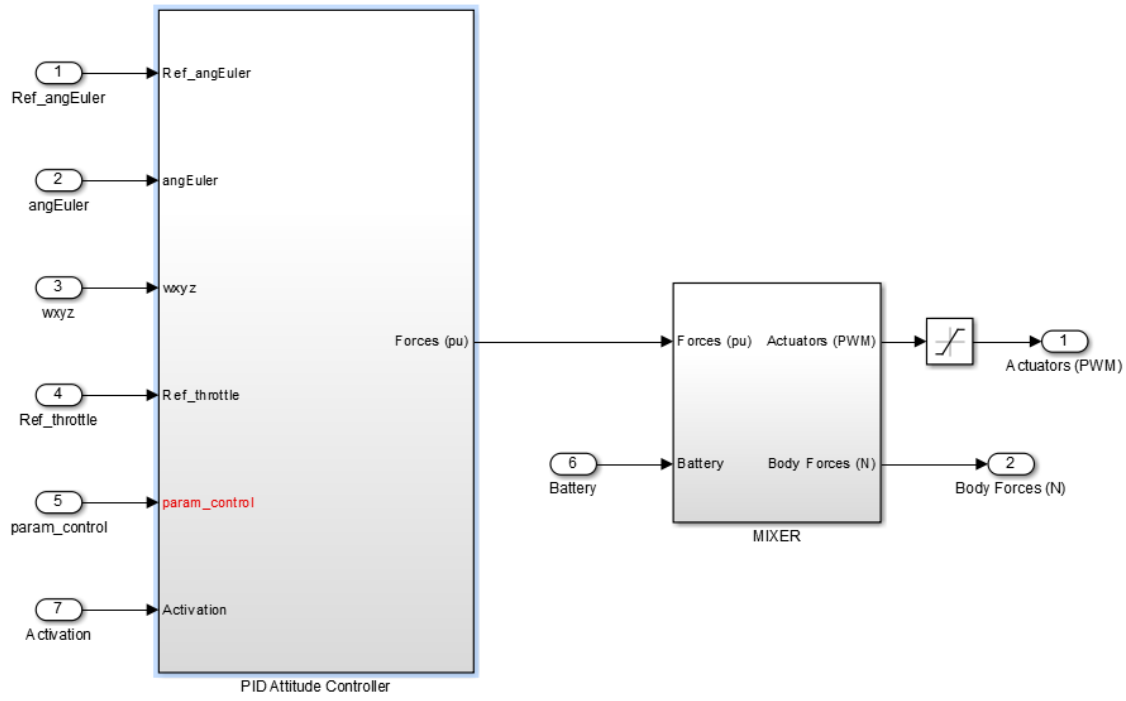


Figura 4.6. Esquema del Control de Estabilidad.

4.3.4 Control de navegación

Una vez realizado el control de estabilidad para realizar el vuelo manual, el siguiente paso a seguir es el modelado e implantación de la cámara de flujo óptico en un control de vuelo autónomo, es decir, la creación de un Control PID de Navegación. Dicho control funciona de manera muy parecida al Control de Estabilización, solo que en lugar de las señales de la emisora utiliza las medidas de los sensores (sensor de flujo óptico en nuestro caso, solo tenemos uno). Esto nos permite diseñar un control que mantiene altura constante, dato que se puede definir en el código a cargar o mandar a través de la UART desde el PC (Consultar apartado *6.5. Ajuste del control en tiempo real*).

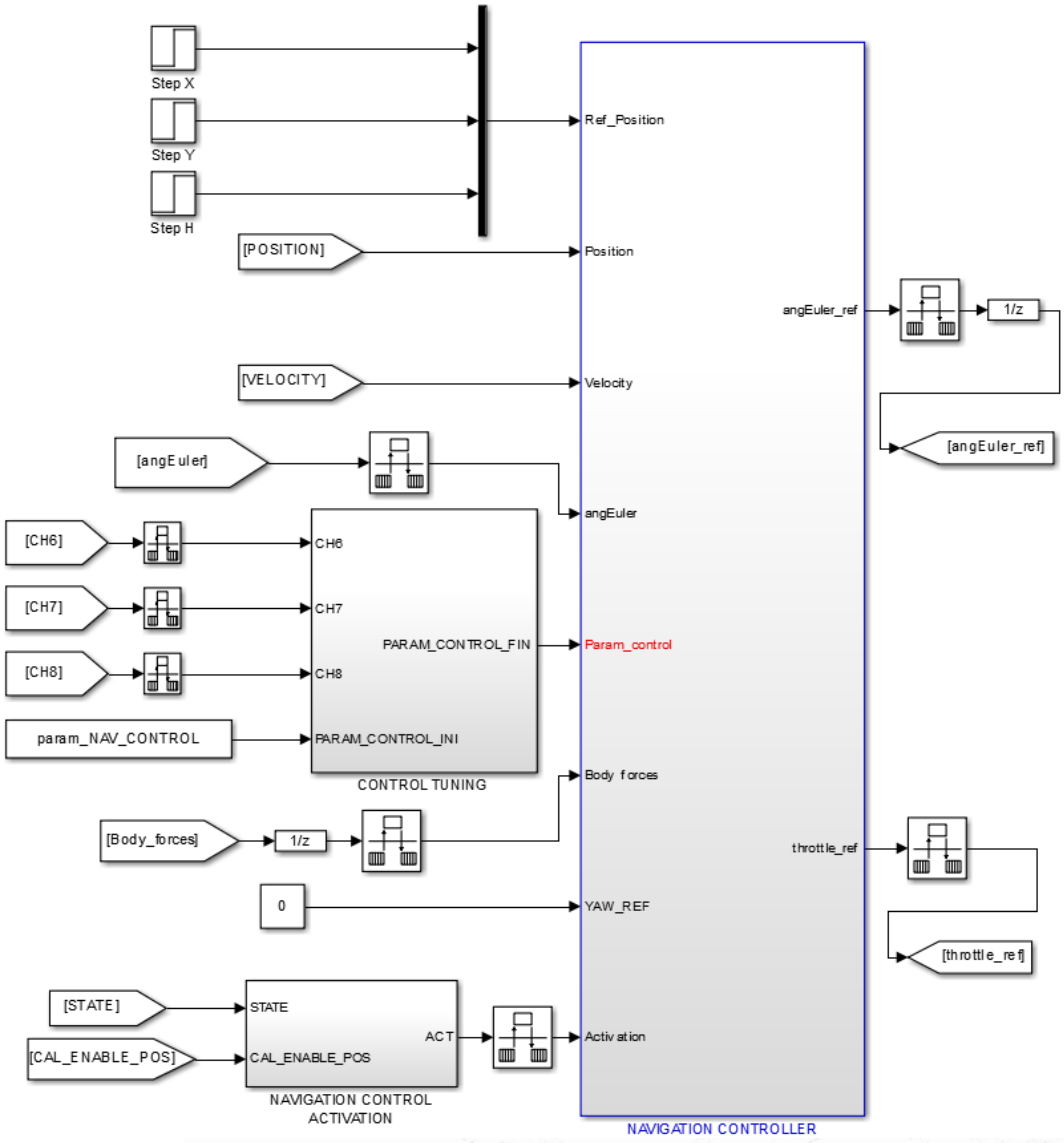


Figura 4.7. Imagen del Control de Navegación con Infrarrojos.

Sin embargo, aunque se ha logrado la creación de un control de navegación basado en sensores infrarrojos del año pasado, por limitación de tiempo se ha hecho imposible introducir la medida del sensor de flujo óptico. No obstante, el funcionamiento del sensor en el entorno de Matlab es correcto y se ha diseñado toda la interfaz de dicho sensor en Simulink de manera satisfactoria (consultar apartado *6.6. Implementación del sensor en Simulink)*, por lo que se deja abierta la posibilidad de continuar este trabajo en proyectos posteriores para diseñar controles de altura, seguimiento de rutas, vuelo en recintos limitados por marcas, etc.

Capítulo 5

Cámara de Flujo Óptico

5.1 Introducción a la teoría del Flujo Óptico

El flujo óptico es el patrón del movimiento aparente de los objetos, superficies y bordes en una escena causado por el movimiento relativo entre un observador (un ojo o una cámara) y la escena. El concepto de flujo óptico se estudió por primera vez en la década de 1940 y, finalmente, fue publicado por el psicólogo estadounidense James J. Gibson como parte de su teoría de la *affordance* [24]. Las aplicaciones del flujo óptico tales como la detección de movimiento, la segmentación de objetos, el tiempo hasta la colisión, la codificación del movimiento compensado y la medición de la disparidad estereoscópica utilizan este movimiento de las superficies y bordes de los objetos.

Las principales ventajas de los algoritmos de visión artificial basados en flujo óptico son la consistencia, fiabilidad y objetividad de los controles, la capacidad de funcionamiento en entornos difíciles, el control a altas velocidades, el manejo de aeronaves de tamaño reducido, el control de alta precisión, y el gran volumen de datos generados en el proceso [25]. Sin embargo los dispositivos de flujo óptico son muy dependientes de dos principales cosas: el grado de iluminación del entorno, ya que a baja luminosidad la cámara funciona peor; y la rugosidad del terreno, ya que para un funcionamiento correcto la cámara debe ser capaz de detectar puntos significativos, algo imposible en suelo liso, plano y sin bordes, picos o aristas [26].

A continuación se van a revisar los algoritmos más comunes para el cálculo del flujo óptico, extraídos de tesis de la aplicación del flujo óptico a la navegación con cuadricópteros [27] de la Universidad de Bolonia:

5.1.1 Método de Lucas-Kanade

El método de Lucas- Kanade [28] fue desarrollado por Bruce D. Lucas y Takeo Kanade en 1981. Lucas y Kanade fueron capaces de estimar el flujo óptico suponiendo que los grupos de los píxeles adyacentes de la imagen tienen la misma velocidad de movimiento, y el uso de la ecuación el flujo óptico para todos los píxeles transformó el problema mediante un sistema de ecuaciones diferenciales para una sistema lineal, por el criterio de mínimos cuadrados. Al combinar la información de varios píxeles cercanos, el método Lucas-Kanade a menudo puede resolver la ambigüedad inherente de la ecuación de flujo óptico. También es menos sensible al ruido de imagen que los métodos de punto se refieren. Por otra parte, ya que es un método puramente local, no puede proporcionar información de flujo en el interior de las regiones uniformes de la imagen.

Este concepto sentó la base del futuro método **Kanade-Lucas-Tomasi** (KLT) de rastreo de características, para paliar del gran coste de las técnicas tradicionales de registro de imágenes.

5.1.2 Pirámide de Lucas-Kanade

En 2000 Jean-Yves Bourget propuso una aplicación modificada del método Lucas-Kanade, que se basa en el concepto de **Pirámide Gaussiana** [29]. En la práctica, la estimación del flujo óptico es calculada inicialmente en el nivel superior de la pirámide, es decir, en la resolución más baja, después de lo cual se propaga al siguiente nivel donde se convierte en la primera estimación para el cálculo de flujo óptico. En cada nivel se calcula el flujo óptico con el método de Lucas-Kanade, sin embargo con un tiempo de cálculo inferior al haber inicializado el valor estimado en el nivel superior. La restricción de tener un flujo óptico igual para grupos de píxeles adyacentes también se ve cumplido en todos los niveles.

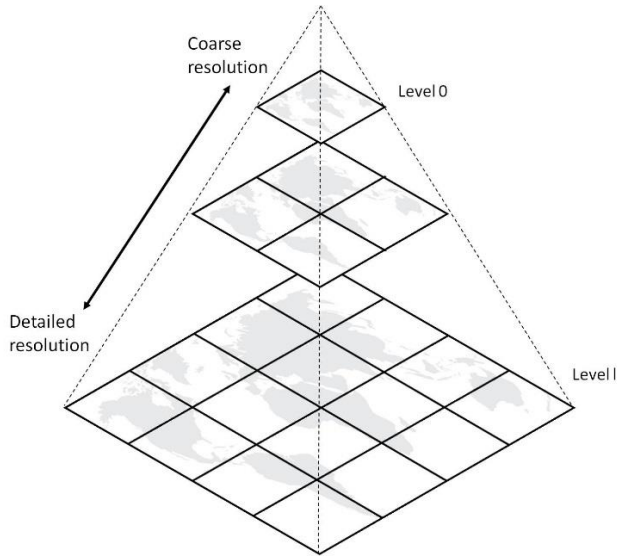


Figura 5.1. Ilustración de la Pirámide Gaussiana.

5.1.3 Métodos "*Block-Matching*"

Los métodos de ***Block-Matching*** o **Coincidencia de Bloques** [30] tienen por objetivo estimar la disparidad de un punto P en la primera imagen mediante la comparación de una pequeña región alrededor del zócalo *p*, que será la ventana de referencia, con una serie de otras regiones, del mismo tamaño de la ventana de referencia, extraída de una segunda imagen. El valor para la función de correspondencia se obtiene mediante la comparación de conjuntos de la primera imagen con una ventana deslizante en la segunda imagen. Dicha ventana deslizante se mueve en la segunda imagen a lo largo de la línea *epipolar* de *p*, y por cada aumento se genera una nueva trama (*frame*) de datos para su posterior análisis e interpretación.

5.1.4 Técnicas de Correlación

La correlación [31] es una técnica que genera un mapa de densa disparidad, es decir, la disparidad se encuentra en cada punto de la imagen de referencia. Sin embargo, debido a que procesa todos los píxeles la correlación es relativamente lenta, tanto que no se recomienda para adaptarla a sistemas en tiempo real RTOS. Mediante la comparación de la algebraicamente los valores de los píxeles presentes en la ventana y la trama, se compara la ventana de referencia con las diversas tramas capturadas. En la siguiente imagen se puede ver un ejemplo de cómo se comparan dos imágenes para el cálculo de la disparidad entre las mismas, y se observa como la segunda en la segunda imagen la cámara se ha desplazado hacia la derecha.

Figura 5.2. Ejemplo del cálculo de la disparidad entre dos imágenes, obtenido en [27].

5.1.5 Ejemplos de aplicaciones del Flujo Óptico

El flujo óptico se ha implantado con éxito en multitud de campos diferentes. Por ejemplo, la tecnología óptica se encuentra presente en un objeto bastante cotidiano como puede ser el ratón óptico, que presenta una precisión claramente superior a los ya en declive ratones mecánicos. También se ha aplicado al ámbito audiovisual, en aquellos largometrajes que requieren una representación completa en 3D de un personaje: el movimiento corporal, los gestos y sobre todo la captura de expresiones faciales, que se hace gracias a unos diminutos dispositivos lumínicos colocados por encima del cuerpo del actor que la cámara detecta con facilidad.

Pero la aplicación más extendida del flujo óptico es la estimación de la velocidad de un objeto a partir de imágenes, muy útil para sistemas de seguimiento y rastreo para la detección de objetos. Es por ello bastante común el empleo del flujo óptico en radares en vías de circulación para una certera medida de la velocidad de los vehículos. En la siguiente imagen, proporcionada por Matlab [32], se aprecia la correcta detección de los vehículos circulando por la carretera:

Figura 5.3. Detección de objetos móviles en una serie de tramas usando flujo óptico.

5.2 Características del sensor PX4FLOW

Las especificaciones de la cámara son las siguientes:

- ✓ 168 MHz Cortex M4F CPU (128 + 64 KB RAM)
- ✓ Sensor de imagen CMOS MT9V034, resolución de 752x480 píxeles
- ✓ L3GD20 3D Gyro integrado, 2000°/s y 780 Hz de tasa de refresco, modo de alta precisión por defecto a 500°/s
- ✓ Sonar HRLV-EZ4
- ✓ Lente M12 de 16mm (incluye un filtro de IR)
- ✓ Sensibilidad a la luz de 24×24 μm super-píxeles
- ✓ Tamaño de 45.5 mm X 35 mm
- ✓ Potencia necesaria: 115mA / 5V
- ✓ Frecuencia de Muestreo: 400Hz
- ✓ Procesamiento de Flujo Óptico a imágenes dicretizadas 4x4
- ✓ Conexión serie MicroUSB de hasta 921600 baudios

La cámara también cuenta con una serie de *status-LEDs* para indicar al usuario el estado del sensor y de la comunicación: **LED Verde** (sensor encendido), **LED Rojo** (error), **LED Ámbar** (sensor transmitiendo datos) y **LED Azul** (sensor procesando datos). Además también tiene un Switch en un lateral que permite hacer un RESET del funcionamiento del sensor, en caso de que haya acumulado error en las medidas, o simplemente queramos empezar de nuevo a leer tramas.

Desarrollado por la universidad suiza **ETH Zúrich** © [33], se puede adquirir por internet en casi cualquier tienda de electrónica y robótica online, y al tratarse de *Open Source* y *Open Hardware* se puede reprogramar de forma totalmente libre para realizar cualquier tipo de visión por ordenador a nivel bajo de forma eficiente.

5.3 Instalación y Calibración de la PX4-FLOW

En este apartado se van a listar todos los procesos involucrados en la puesta en marcha del sensor, en orden temporal, para que sirva de guía a futuros usuarios que deseen instalar el sensor de flujo óptico PX4FLOW en un dron con OpenPilot o controladores similares.

5.3.1 Actualización del Firmware y Ajuste desde QGroundControl

En primer lugar antes de instalar el sensor en el cuadricóptero, debemos asegurarnos de que cuenta con la última versión estable de Software (evitaremos instalar eso si cualquier versión Beta para evitar inutilizar el sensor). Los pasos para realizar una actualización del Firmware los podemos encontrar en el siguiente enlace en la página oficial de PIXHAWK, por lo que no es necesario ponerlos aquí de nuevo [34].

También es necesario ajustar la calidad de la cámara y realizar una serie de ensayos para asegurarnos que la lente está enfocando correctamente. Para ello conectaremos el sensor al PC por USB y encenderemos el programa QGroundControl. Este programa nos es muy útil, pues podemos realizar la actualización del Firmware con él, verificar que la información del sensor es la correcta, incluso acceder a la visualización de la cámara del sensor.

Para todo ello, una vez detectada la cámara, establecemos la conexión (permite hasta 921600 baudios para máxima calidad) y entramos en la pestaña "*Tool Widgets" → "Video Downlink"* (para la versión 2.0.1 de QGC o posteriores). Una vez allí apuntamos la cámara a un objeto a una distancia recomendada de hasta 3 metros, distancia optima entre 1 y 1,5 metros, y ponemos un valor de 1 en el parámetro "VIDEO_ONLY" (importante si queremos calidad aceptable en el video) [Figura 5.4].

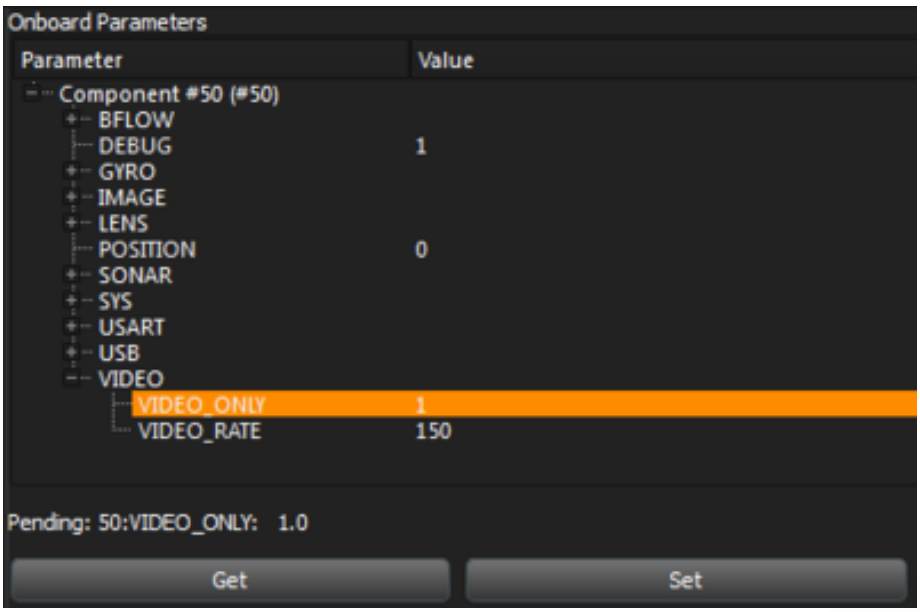


Figura 5.4. Detalle del parámetro a cambiar en QGroundControl.

Para más información respecto a estos pasos a seguir consultar el siguiente enlace en la página oficial de PIXHAWK: https://pixhawk.org/modules/px4flow#image_quality_and_output

5.3.2 Alimentación y Cableado

Cuando en un primer momento conectamos el sensor PX4FLOW a la tarjeta de control PIXHAWK se observó que el sensor funciona de forma correcta. Así que fue todo una sorpresa cuando más adelante se procedió a conectarlo por I2C a la nueva placa OP Revo y ni siquiera se encendían los status-LEDS. Tras un análisis exhaustivo de la configuración de los puertos, se seguía sin encontrar el fallo, así que se propuso crear un conector I2C desde cero que permitiese monitorizar las señales enviadas entre sensor y placa. Para ello se empalmaron cables en sendos conectores para cada componente, se conectaron a una *ProtoBoard* y se empleó un osciloscopio para analizar las señales. Se adjuntan imágenes de los planos esquemáticos de ambos componentes [Figura 5.5 y Figura 5.6].

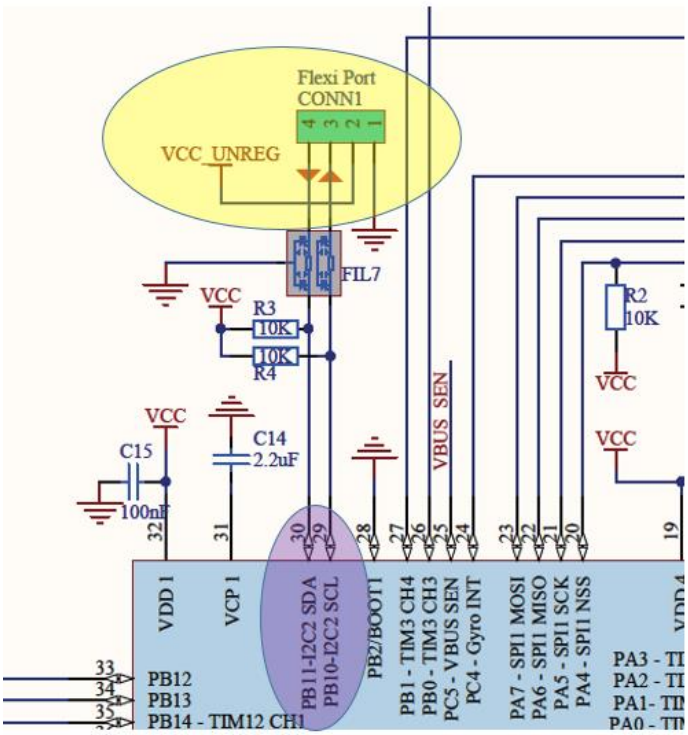


Figura 5.5. Detalle del FlexiPort de OP Revo. Marcado en amarillo, la distribución de los cables de señal, tierra y tensión. En morado, los dos pines de señal SDA y SCL.

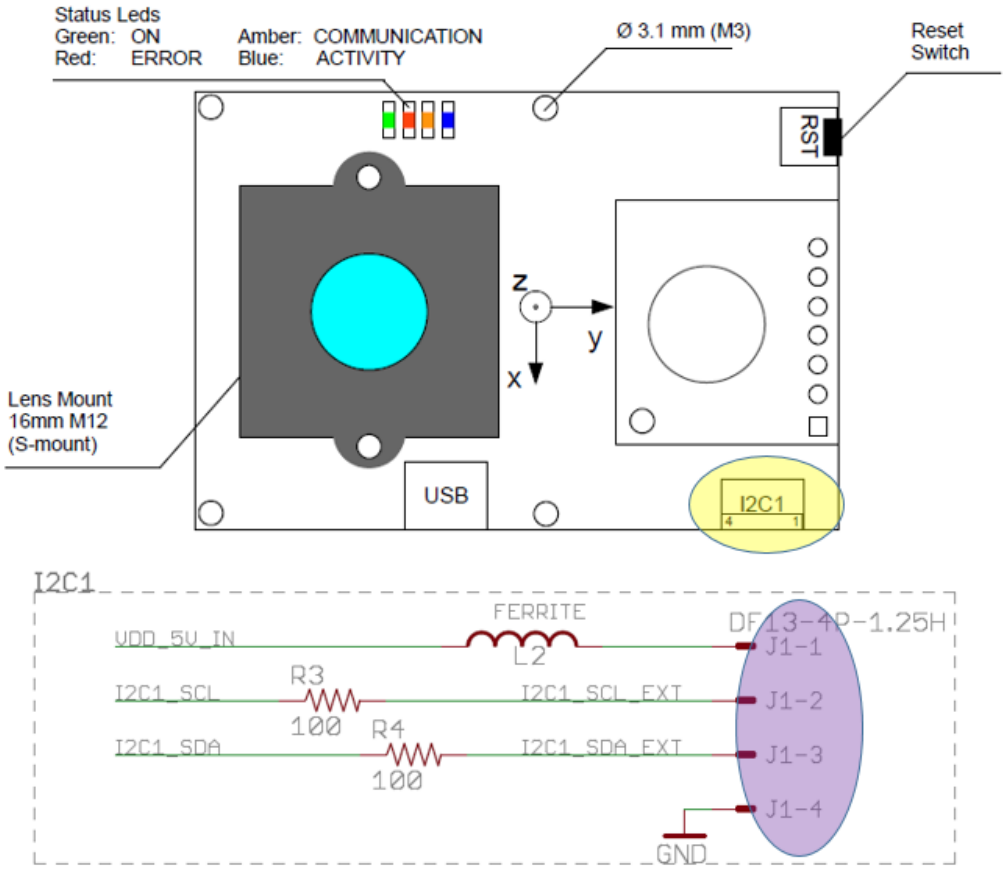


Figura 5.6.Detalle del Puerto I2C del PX4FLOW. Marcado en amarillo, la posición y orientación del mismo. En morado, la distribución de los cables de señal, tierra y tensión.

Al analizar con un osciloscopio no se apreciaba que el sensor emitiese ninguna señal, ni siquiera se encendía aun recibiendo corriente y tensión. ¿Entonces donde estaba el problema? No fue hasta más tarde, cuando al conectar el sensor de nuevo a la PIXHAWK y midiendo con un polímetro se hizo evidente el error: se medían correctamente las señales SCL (clock) y SDA (Data), pero las señales de VDD (5V) y GND (ground) aparecen cambiadas, al contrario de lo que indica el schematic y el manual. Si hubiesen estado en orden GND-SDA-SCL-VDD se habría considerado un fallo de la orientación del conector (dado la vuelta), pero en este caso la distribución era totalmente distinta. El cambio se ilustra en la siguiente figura [Figura 5.7]:

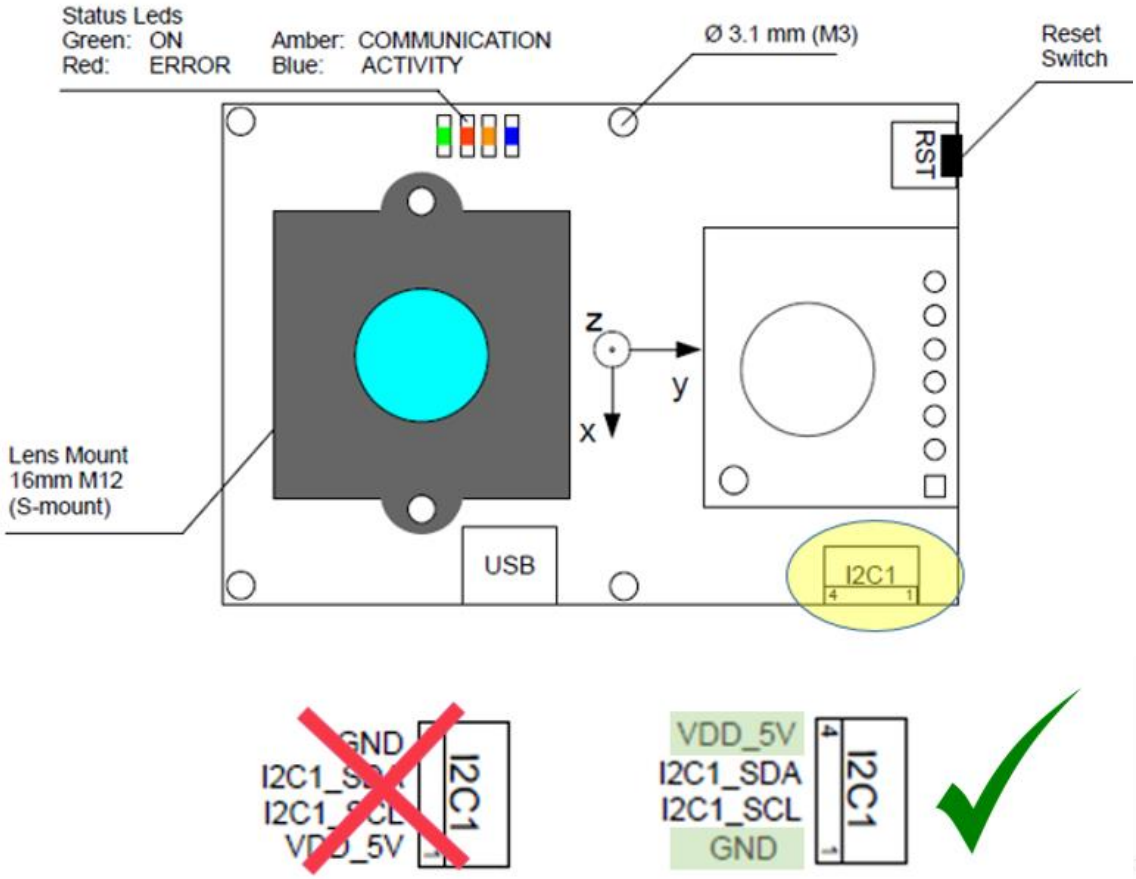


Figura 5.7. Detalle del Puerto I2C del PX4FLOW corregido. A la izquierda aparece la configuración de puertos errónea del schematic, a la derecha la correcta.

Pero este no era el único obstáculo para conseguir que funcionas el sensor. Una vez conseguida la correcta distribución de pines del sensor, Nos topamos con otro problema diferente. La placa OP Revo no consigue suministrar al sensor la alimentación necesaria para alimentar el sensor (5V y 115 mA) y para mantener la tensión de la línea I2C en un valor adecuado para la lectura de datos (3.3V). Este inconveniente se solucionó alimentando el sensor desde un regulador de tensión externo a la placa de control. Y para mantener tensión aceptable en la línea, se procedió a realizar un *Pull-Up* en las dos líneas de datos, empalmando resistencias de 470mΩ entre la línea y VDD a 5V. El cable quedaría ahora de la siguiente manera [Figura 5.8]:

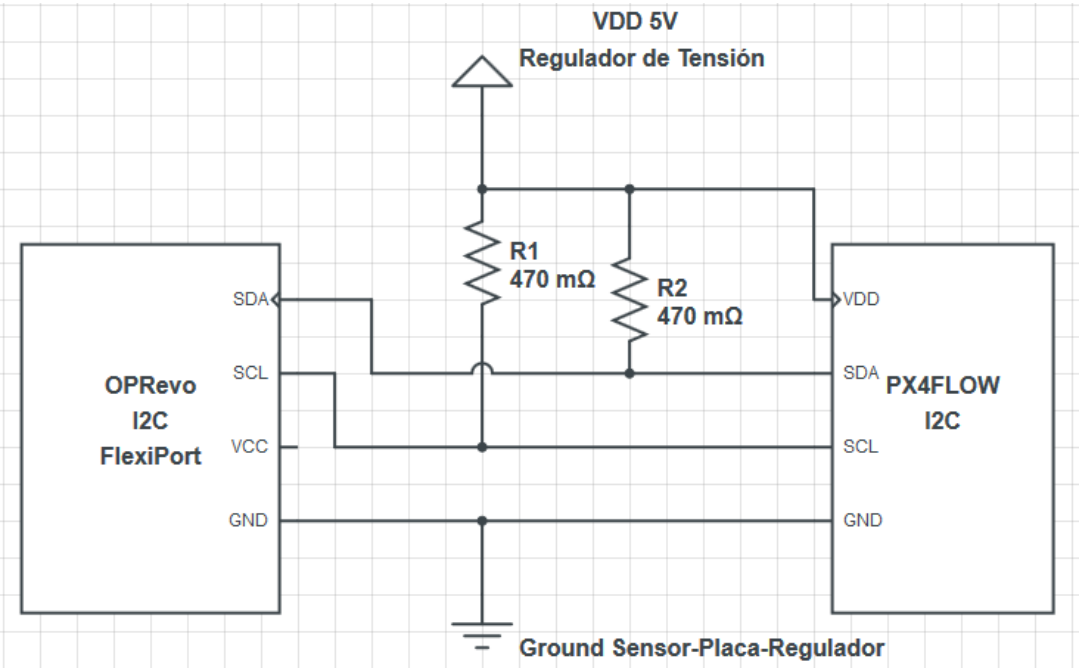


Figura 5.8. Detalle del cableado para alimentar de forma correcta el PX4FLOW

5.3.3 Análisis de la trama I2C

Una vez alimentado correctamente el sensor, es la hora de configurar la trama I2C desde Simulink para que ambos componentes trabajen con el mismo protocolo de información. Como la PIXHAWK se comunica de forma automática con el sensor al conectarlos, no se incluye en la documentación información apenas de la trama de I2C necesaria para pedir al sensor que envíe datos. Pero con la ayuda de nuestro querido osciloscopio podemos analizar los bits transmitidos por el puerto de datos (SDA) contando los ciclos del reloj (SCL). El resultado de las lecturas de la transmisión PIXHAWK-PX4FLOW se muestra a continuación [Figura 5.9]:

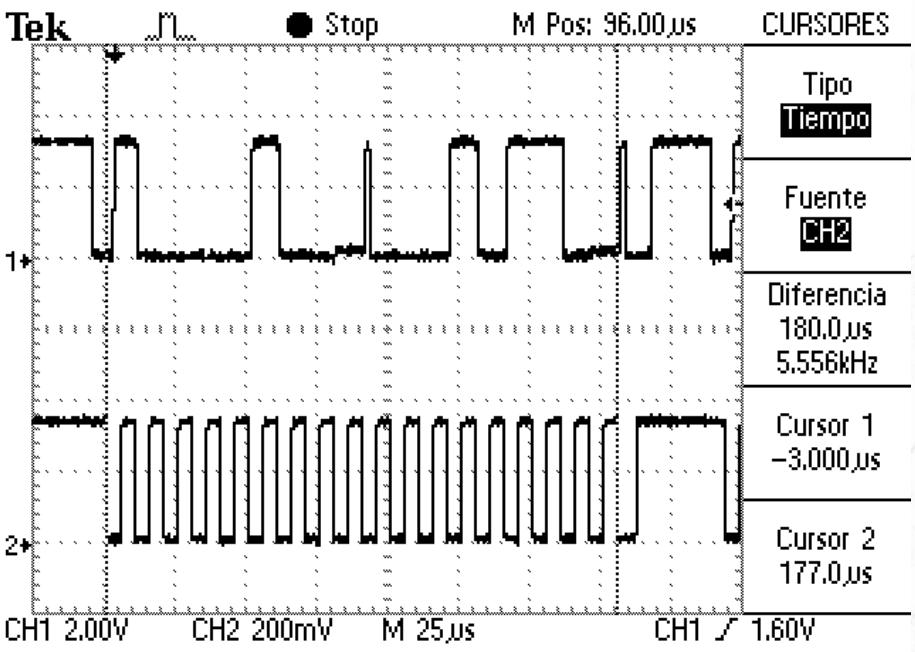


Figura 5.9. Trama I2C transmitida por la Pixhawk.

Si contamos los ciclos de reloj en un periodo determinado de tiempo, podemos averiguar la frecuencia de la conexión I2C necesaria. En nuestro caso, contamos 18 ticks del reloj en un periodo de 180µs, lo cual nos da un pulso cada 10µs → obtenemos una frecuencia de 100KHz. Ahora procedemos a analizar los bits enviados en la transmisión tomando como referencia el reloj: cada pulso de reloj equivale a un bit, siendo el pulso la transición de nivel bajo a nivel alto de la señal de clock [Figura 5.10]. La trama i2C agrupa los bits de 4 en 4, asigna 3 bits a la dirección y el último al comando en cuestión (0-write y 1-read) [Tabla 5.1]:

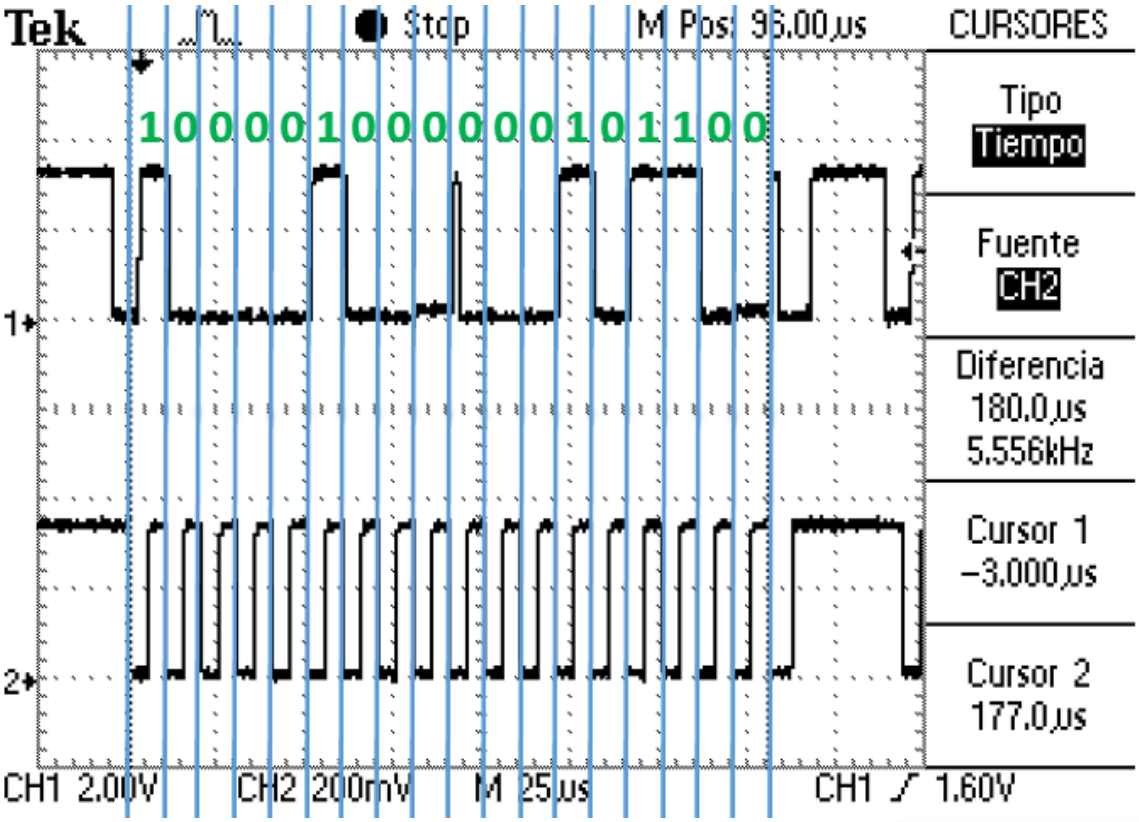


Figura 5.10. Separación de los bits de la trama I2C

	ANÁLISIS TRAMA I2C PX4FLOW																		
Nº bit	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
Grupo	1				2				3				4				5		
Lectura	1	0	0	0	0	1	0	0	0	0	0	0	1	0	1	1	0	0	0
análisis	4			w	2			w	0			w	check				No interesan		

Tabla 5.1. Análisis de los bits de la trama I2C

De esta manera hemos descubierto el inicio de la trama de lectura del sensor para la trama normal de I2C: dirección hexadecimal "0x42", dato "0x0", comando *write* (0). De esta manera, en nuestro diagrama de Simulink lo único que nos queda por hacer es configurar el bloque de lectura/escritura de I2C para que escriba un 0 en la dirección 0x42, y leer la respuesta del sensor. Dependiendo del dato que escribamos, el sensor nos devuelve una trama distinta. Escribiendo un "0x0" obtenemos la trama **i2c_frame** y escribiendo un "0x16", la trama **i2c_integral_frame.** Sabemos por la documentación que la trama proporcionada por el sensor es de 22 bytes (**I2C frame**) y 26 bytes (**I2C integral frame**), así que ya podemos ordenar los bytes a la salida del bloque para agruparlos en las distintas medidas del sensor, listadas a continuación [*Tabla 5.2* y *Tabla 5.3*]:

Register Address	**Register Content**	**Units**
0x00 (0)	Framecounter lower byte	#frames
0x01 (1)	Framecounter upper byte	
0x02 (2)	latest Flow*10 in x direction lower byte	pixels
0x03 (3)	latest Flow*10 in x direction upper byte	
0x04 (4)	latest Flow*10 in y direction lower byte	pixels
0x05 (5)	latest Flow*10 in y direction upper byte	
0x06 (6)	x velocity*1000 lower byte	meters/sec
0x07 (7)	x velocity*1000 upper byte	
0x08 (8)	y velocity*1000 lower byte	meters /sec
0x09 (9)	y velocity*1000 upper byte	
0x0A (10)	Optical flow quality lower byte	(0:bad, 255 max)
0x0B (11)	Optical flow quality upper byte	
0x0C (12)	latest gyro x rate lower byte	rad/sec
0x0D (13)	latest gyro x rate upper byte	
0x0E (14)	latest gyro y rate lower byte	rad/sec
0x0F (15)	latest gyro y rate upper byte	
0x10 (16)	latest gyro z rate lower byte	rad/sec
0x11 (17)	latest gyro z rate upper byte	
0x12 (18)	gyro range, [0 .. 7] = [50 .. 2000]	deg/sec
0x13 (19)	Sonar Timestamp	miliseconds
0x14 (20)	Grounddistance lower byte	meters
0x15 (21)	Grounddistance upper byte	

Tabla 5.2. Agrupación de Bytes de la Trama "i2c_frame"

Register Address	**Register Content**	**Units**
0x16 (22)	Framecounter since last I2C readout lower byte	#frames
0x17 (23)	Framecounter since last I2C readout upper byte	
0x18 (24)	Accumulated flow around x axis since last I2C readout lower byte	rad*10000
0x19 (25)	Accumulated flow around x axis since last I2C readout upper byte	
0x1A (26)	Accumulated flow around y axis since last I2C readout lower byte	rad*10000
0x1B (27)	Accumulated flow around y axis since last I2C readout upper byte	
0x1C (28)	Accumulated gyro x rates since last I2C readout lower byte	rad*10000
0x1D (29)	Accumulated gyro x rates since last I2C readout upper byte	
0x1E (30)	Accumulated gyro x rates since last I2C readout lower byte	rad*10000
0x1F (31)	Accumulated gyro x rates since last I2C readout upper byte	
0x20 (32)	Accumulated gyro x rates since last I2C readout lower byte	rad*10000
0x21 (33)	Accumulated gyro x rates since last I2C readout upper byte	
0x22 (34)	Accumulation timespan in microseconds since last I2C readout byte 0	microseconds
0x23 (35)	Accumulation timespan in microseconds since last I2C readout byte 1	
0x24 (36)	Accumulation timespan in microseconds since last I2C readout byte 2	
0x25 (37)	Accumulation timespan in microseconds since last I2C readout byte 3	
0x26 (38)	Time since last sonar update byte 0	microseconds
0x27 (39)	Time since last sonar update byte 1	
0x28 (40)	Time since last sonar update byte 2	
0x29 (41)	Time since last sonar update byte 3	
0x2A (42)	Grounddistance in meters*1000 lower byte	meters
0x2B (43)	Grounddistance in meters*1000 upper byte	
0x2C (44)	Temperature in Degree Celsius*100 lower byte	degcelsius*100
0x2D (45)	Temperature in Degree Celsius*100 upper byte	
0x2E (46)	Averaged quality of accumulated flow values	(0:bad, 255 max)

Tabla 5.3. Agrupación de Bytes de la Trama "i2c_integral_frame"

5.3.4 Instalación de la cubierta protectora

Sabiendo lo importante que es el sensor para el proyecto, y sabiendo también que el cuadricóptero va a recibir unos cuantos golpes durante las pruebas de vuelo, nos podemos plantear dos opciones distintas: o volamos sin el sensor en las primeras pruebas, estimando los parámetros, y más tarde lo añadimos; o por otro lado buscamos una manera de proteger el sensor frente a los posibles impactos que pueda recibir durante las fases de vuelo y aterrizaje. Y se mire como se mire, la opción más sensata sigue siendo la segunda. Por ello se pensó en diseñar una cubierta protectora con un programa de CAD (Ej: Solidedge) para luego imprimirla con una impresora 3D, y así asegurarnos que el delicado sensor no sufre daño alguno.

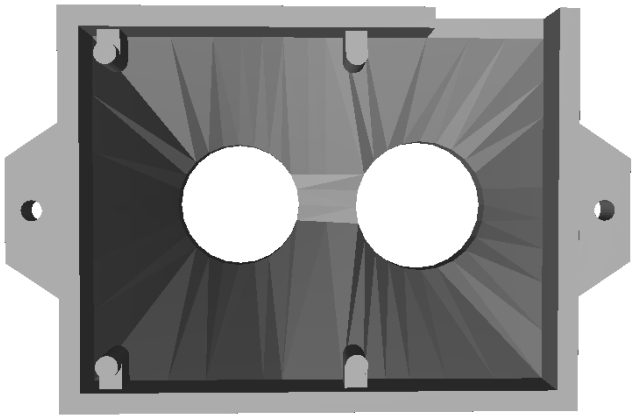


Figura 5.11. Vista interior de la cubierta del sensor.

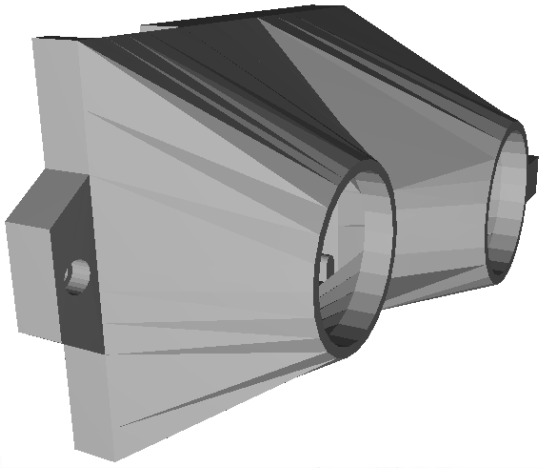


Figura 5.12. Vista lateral de la cubierta del sensor.

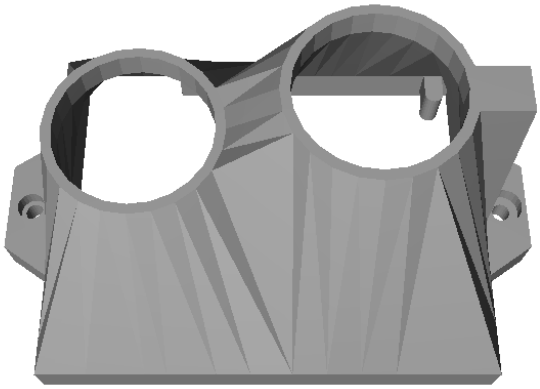


Figura 5.13. Vista superior de la cubierta del sensor.

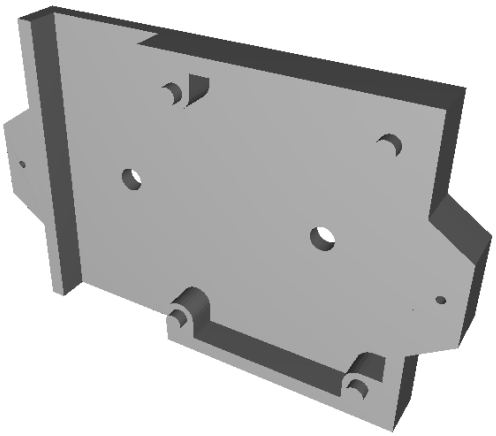


Figura 5.14. Vista superior de la base del sensor.

La cubierta está basada en un modelo ya existente colgado en internet: el diseño **Alpha1** publicado por **AmperCZ** en Thingiverse [35], bajo licencia de Creative Commons - Attribution - Non-Commercial © [36]. Como la licencia exige, el diseño se ha utilizado para un fin no lucrativo, dando crédito de la autoría al creador. Como también se expone en la licencia, se nos permite hacer cambios en el diseño, siempre que dichos cambios se indiquen a continuación: se ha modificado el diámetro de los agujeros de la pieza de la base para que concuerde con los de la parte superior, se ha realizado una abertura en un lateral de la cubierta para poder conectar el cable USB, se han eliminado salientes de la parte interna de la cubierta para que el sensor se pueda introducir de forma más sencilla, y por último se ha rebajado la cara interna de la cubierta en un lateral para evitar que roce contra los pines de conexionado entre la placa y el sonar. Estos cambios se marcan en la siguiente figura [*Figura 5.15*]:

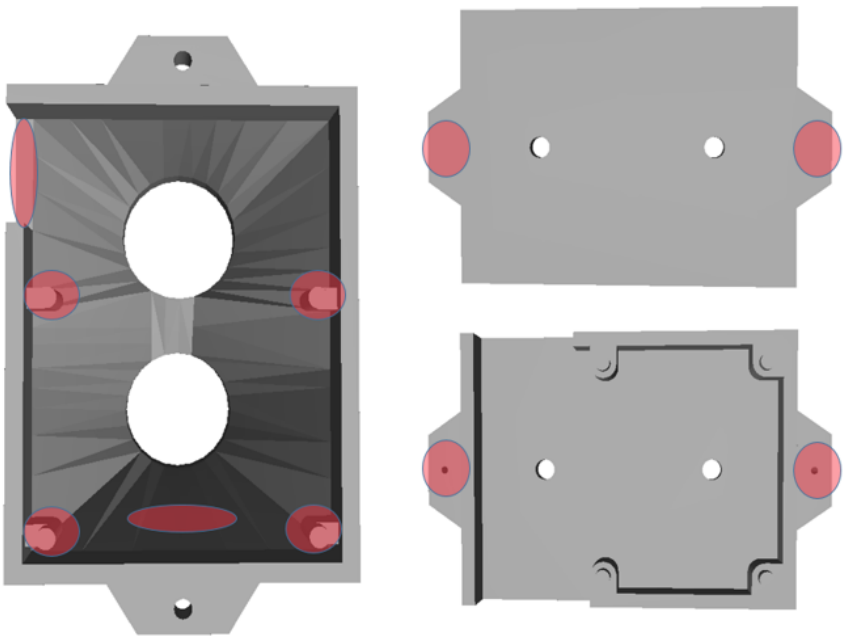


Figura 5.15. Vista de las piezas de la cubierta [Izquierda] y la base [Derecha] con los cambios marcados en color rojo.

Capítulo 6

Implantación

6.1 Software: Matlab y Waijung

Para hacer la programación del control y descargarlo con éxito es necesario un software específico de cada tarjeta. En este proyecto se ha utilizado Matlab y Simulink para los bloques genéricos del control, pero es preciso diseñar los bloques específicos para que la tarjeta sea reconocida como tal y responda correctamente en este entorno. Para ello necesitaremos software o librerías de terceros que exploraremos con más detalle a continuación:

6.1.1 Pixhawk PSP

Dado que en un principio se contempló utilizar la tarjeta de control PX4FMU, era necesario por lo tanto instalar el ***Pilot Support Package*** (PSP) para exportar datos de la PX4 a Simulink. Este paquete incluía el software necesario para reconocer la tarjeta, librerías, y bloques prediseñados para el control de las distintas funciones que ofrece la placa. Sin embargo los bloques son poco o nada personalizables, y se echa de menos una documentación detallada de dichos puertos, timers, creación de bloques para interactuar con la placa, etc. Por ello y por otras razones que se expusieron anteriormente, se decidió abandonar la tarjeta PIXHAWK en favor de la OpenPilot Revolution, mucho más potente y abierta que la anterior.

6.1.2 Arduino Driver Blocks

En el anterior proyecto se diseñaron una serie de *Driver Blocks* para la tarjeta HKPilot Mega [37]. 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 salidas (outputs) según convenga, tanto para recoger medidas del controlador como para actualizar registros y salidas. Hay que tener en cuenta que no tienen efecto alguno durante las simulaciones.

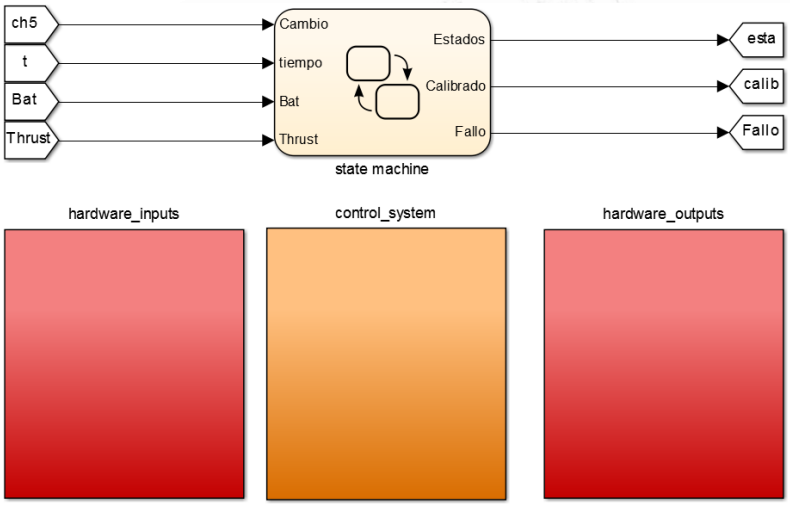


Figura 6.1. Ejemplo de Control con APM Driver Blocks.

Para este proyecto se ha encontrado una dificultad importante. Al cambiar la tarjeta de control a la nueva OpenPilot Revolution ninguno de los *Driver Blocks* ya preparados de años anteriores funciona. Esto significa que o bien empezamos a programar bloques desde cero, o bien buscamos un software para interactuar con las distintas funciones de OpenPilot. Para ello se ha tenido que recurrir a un paquete de software tailandés llamado **Waijung,** cuyos bloques y librerías se explicarán en el siguiente apartado.

6.1.3 Waijung Blockset

Para descargar el control directamente sobre la tarjeta se ha utilizado un conjunto de bloques de Simulink diseñados por un grupo de programadores Tailandeses, llamado ***Waijung Blockset*** [38]. Dicho software se encarga de generar de forma fácil y automática el código en C+ para Matlab y el entorno de Simulink, pensado específicamente para el chipset presente en las tarjetas de OpenPilot. Actualmente Waijung ha sido diseñado específicamente para apoyar a la familia de microcontroladores **STM32F4**, que pertenece a STMicroelectronics. Por lo tanto el conjunto del Blockset de Waijung como los *DriverBlocks* para el STM32F4 han sido completamente rediseñados con nuevas características y mejoras.

Para asegurar un correcto funcionamiento de los bloques que proporciona *Waijung* es necesario especificar una serie de parámetros: el micro exacto que estamos utilizando, compilador, velocidad el reloj, etc. Todo ello se define en un tipo de bloque especial que recibe el nombre de "***Target Setup***". Esto también es necesario para habilitar cualquier tipo de comunicación donde se seleccionen las características básicas de funcionamiento, velocidad de transmisión, períodos de muestreo internos, timers, módulo SPI, UART, I2C…

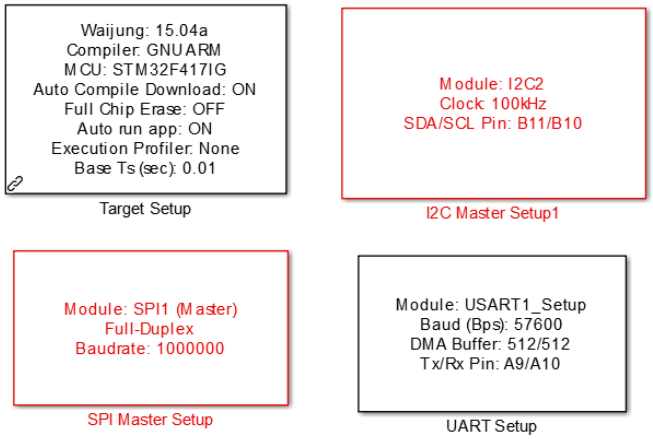


Figura 6.2. Imagen del "Target Setup" y los diversos bloques de configuración utilizados

6.2 Entorno de Trabajo Conjunto

Durante el desarrollo del proyecto, todos los avances que se han ido obteniendo en el proyecto se han integrado con los de otros compañeros trabajando en otros proyectos de control de aeronaves de múltiples rotores. Se ha creado por lo tanto un **Entorno de Trabajo Conjunto** que permite la **simulación, prueba y vuelo de cualquier tipo de aeronave**, que se puede seleccionar desde el archivo de configuración, y modificar multitud de parámetros adicionales (tipo de estimador de estados, tipo de filtros, peso de las medidas de la IMU, tipo de sensores, etc.). A continuación se muestra una imagen esquemática de dicho entorno:

Figura 6.3. Esquema del Entorno de Trabajo Conjunto.

Una gran implementación que ha hecho posible la creación de dicho entorno es la utilidad llamada "***Variant Manager***". Esta sencilla herramienta permite desarrollar dentro de un mismo bloque o subsistema diferentes bloques, los cuales se pueden seleccionar desde cualquier script de Matlab, y de los cuales solamente uno de ellos entra en funcionamiento a la vez (son mutuamente excluyentes). Con ello se ha podido establecer un fichero maestro único para todos los proyectos de aeronaves. Simplemente señalando que se trata de un "cuadricóptero" se habilita el modelo adecuado, lo mismo para "tricóptero" o "helicóptero coaxial"

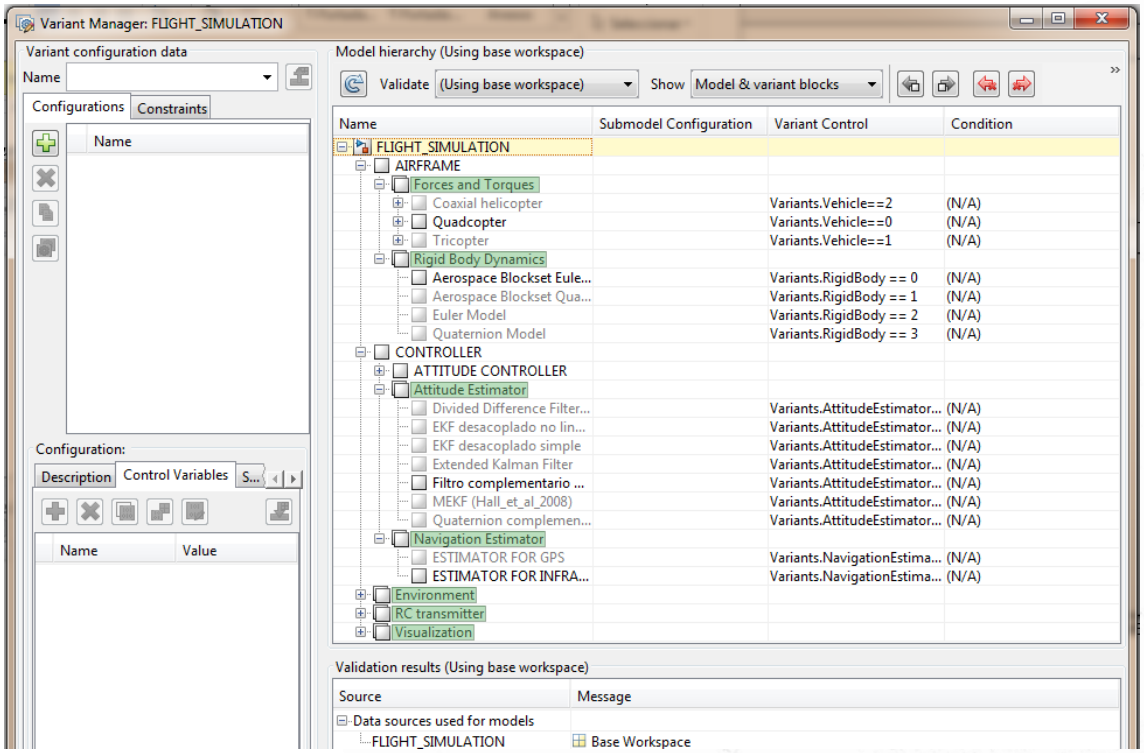


Figura 6.4. Esquema del Variant Manager de Simulink.

6.3 Máquinas de estados

A la hora de controlar el cuadricóptero se ha de trabajar con múltiples variables y cada una debe comenzar a actuar en un determinado momento en concreto. Debido a ello el control es bastante complejo. Por otro lado el funcionamiento que se espera de él no puede ser el mismo en todos los instantes, pues sigue el proceso a seguir ha de ser secuencial. Primero tiene que arrancar para poder despegar, seguidamente debe volar y finalmente aterrizar. Para implementar dicho control sobre el cuadricóptero se han tenido que diseñar varias máquinas de estados con la herramienta *Stateflow* de Simulink. 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 las siguientes han sido la solución para una recepción de tan sólo 4 canales. Todas ellas están orientadas para vuelo manual.

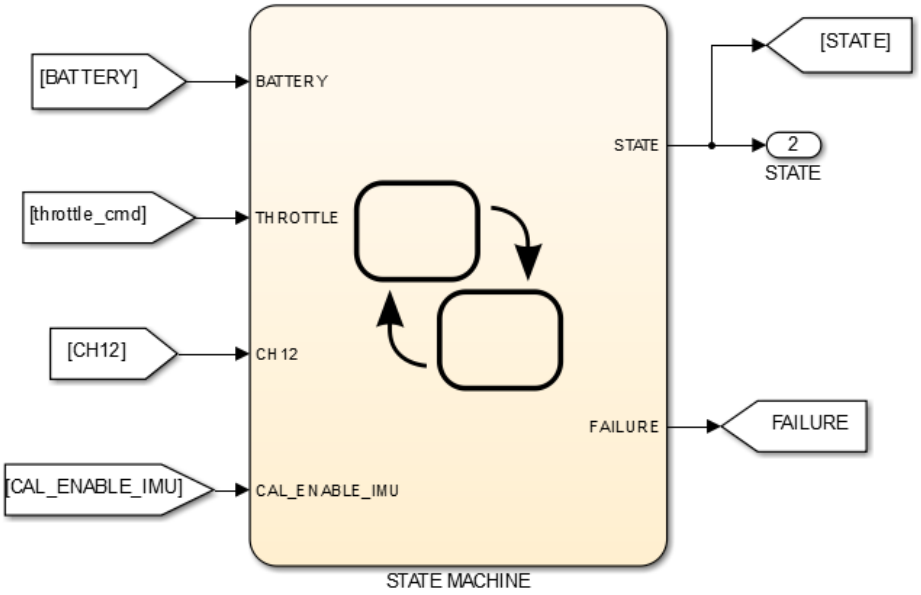


Figura 6.5. Vista externa de la Máquina de Estados.

6.3.1 Máquina de estados de 8 canales.

Esta Máquina de Estados, empleada en años anteriores con éxito [6], contenía la variable "Cambio" que provenía del Ch5 de la emisora. Era una medida bastante cómoda, pues cambiando los interruptores se podía pasar de un estado a otro con total comodidad. Se han añadido una serie de ajustes para mejorar su implementación: el tiempo de calibración se cambió por 20 (anteriormente estaba en 10 segundos) para una mayor precisión a la hora de estimar los ángulos de Euler. Aunque en la Máquina de Estados realmente solo se han usado 5 canales para las transiciones se podrían incluir hasta los 8 de la emisora (de ahí el nombre), aunque es recomendable reservar alguno canal para ajuste de parámetros en vuelo.

El funcionamiento de dicha máquina de estados es el siguiente: Inicialmente se arman los motores y se espera una señal de activación para comenzar a calibrar los giróscopos (*Calibración*). Pasados 20 segundos, termina la calibración (*Armado*) y el cuadricóptero responde con el control desactivado únicamente a la referencia de empuje (señal "*cambio*"). Una vez dentro del estado de vuelo (*Vuelo*) se activa el control, que responde a las referencias enviadas con los sticks de la emisora de alabeo, cabeceo, empuje y guiñada. Existen dos estados adicionales de error que desactivan directamente los motores, en caso de parada por el usuario (*Fin*) o en caso de batería baja (*Baja_Bateria*). A continuación se muestra el diagrama de estados:

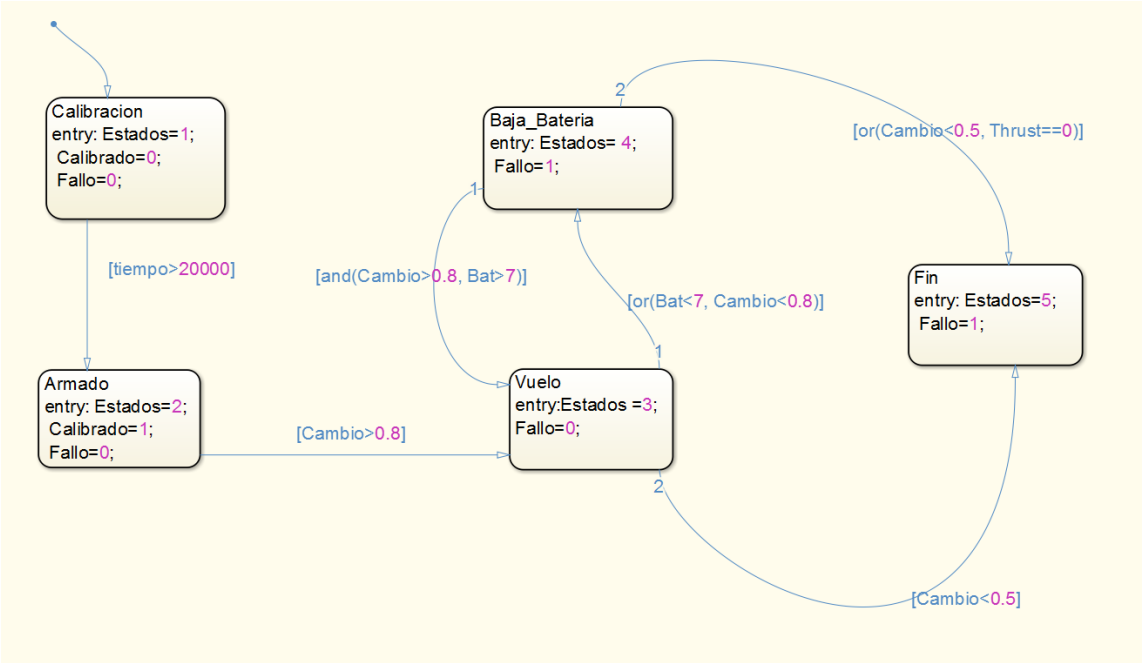


Figura 6.6. Diagrama de estados con 8 canales.

6.3.2 Máquina de estados de 4 canales, con entrada de tiempo.

A continuación se muestra el esquema de la nueva Máquina de Estados, tras ajustarla para trabajar solamente con 4 canales.

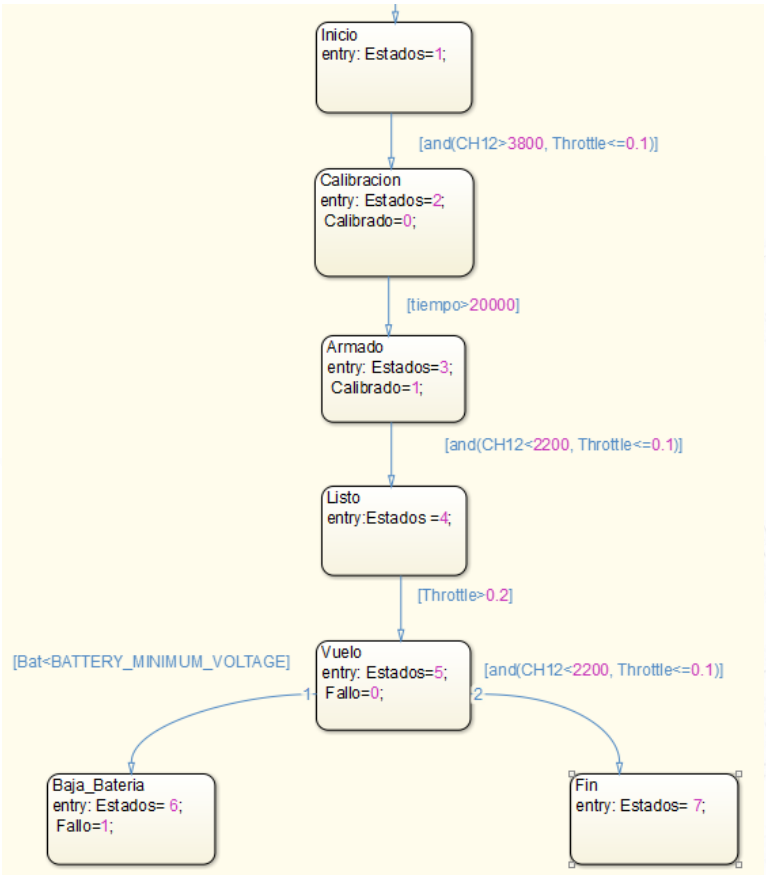


Figura 6.7.Diagrama de estados con 4 canales.

Esta Máquina de Estados se ha diseñado para solventar los problemas ocasionados por la recepción de solamente 4 canales, con el objetivo de controlar la activación del control de la forma más parecida posible a la anterior. El mayor cambio reside en la creación de una señal de activación similar a la señal "Cambio" manejada desde el Ch5 de la emisora, pero 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 creó una variable "CH12" que viene a ser una combinación de los pulsos de los canales 1 y 2 (cuando se colocan ambos canales al mínimo, se activa el Ch12). Los valores del *Throttle* junto con dicha señal permiten hacer la transición correcta entre estados.

6.3.3 Máquina de estados de 4 canales, sin entrada de tiempo.

Para optimizar aún más el funcionamiento de la máquina de estados, y evitar algún conflicto de timer con alguno de los sensores se procedió a modificar la Máquina de Estados para cambiar la señal de tiempo por una señal de habilitación (**CAL_ENABLE_IMU**): de esta manera la cuenta del tiempo entre estados no se hace dentro de la propia máquina de estados, sino que se crea un bloque contador fuera ("CALIBRATION TIMERS") y la maquina únicamente recibe el comando de pasar al estado siguiente.

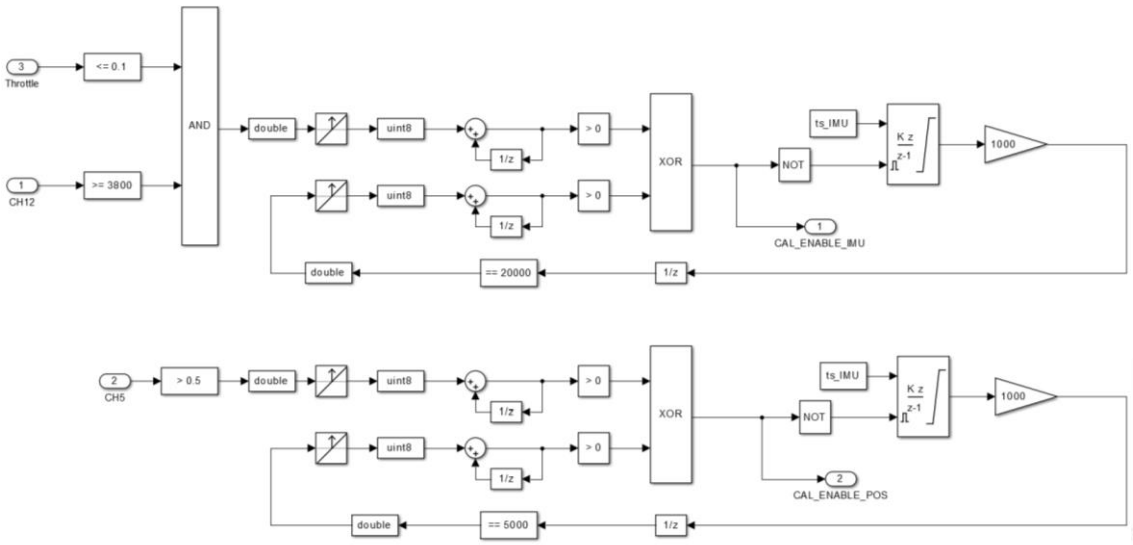


Figura 6.8. Esquema del bloque contador para la Máquina de Estados

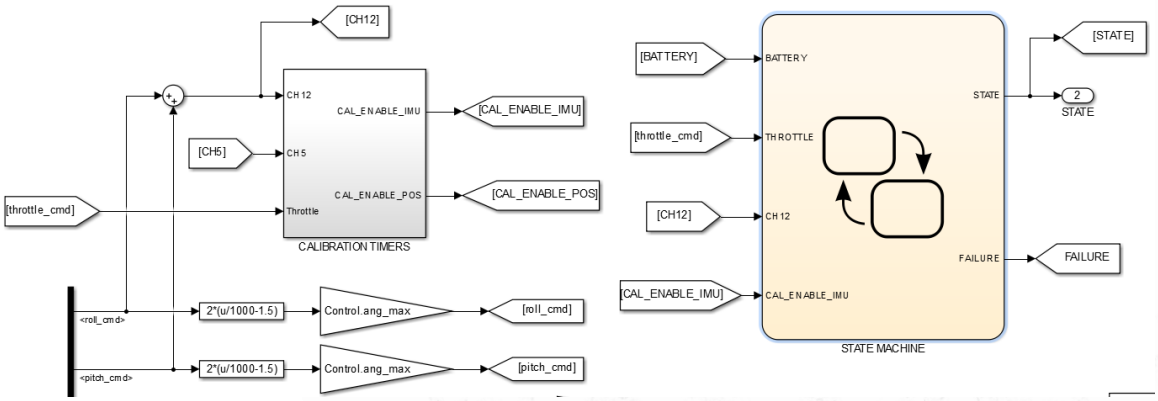


Figura 6.9. Esquema de la relación entre el bloque contador y la Máquina de Estados.

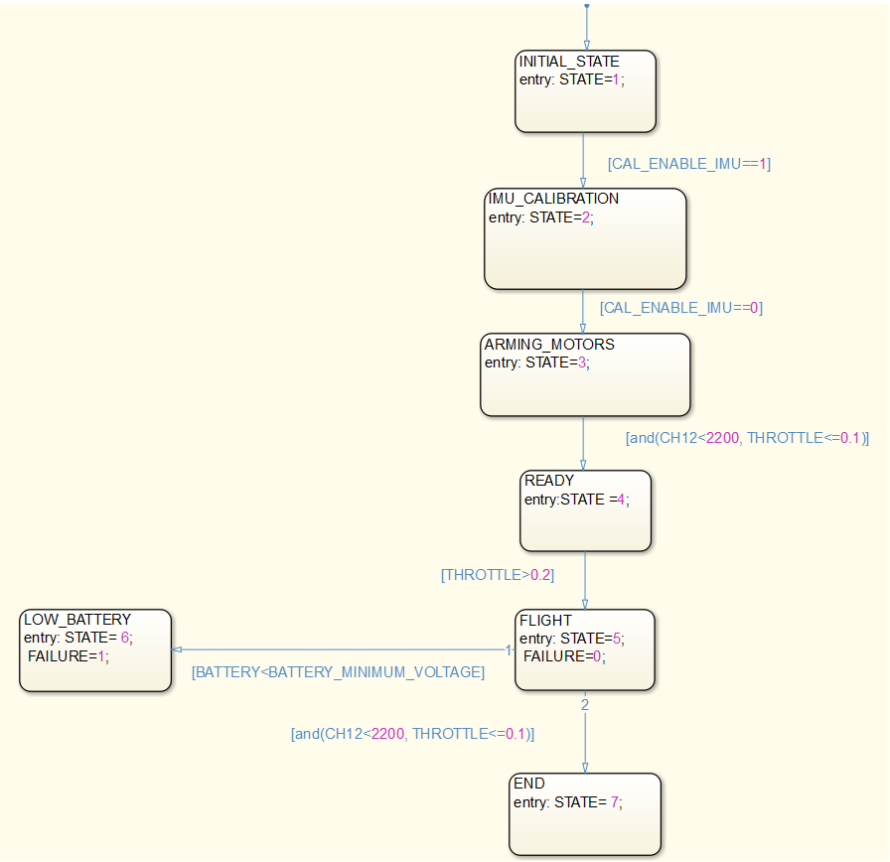


Figura 6.10.Diagrama de estados con 4 canales modificada.

6.4 Ajuste del control en tiempo real

Después de diseñar e implantar el control de estabilización es necesario realizar un pequeño ajuste de los parámetros con el cuadricóptero funcionando. Se tiene de proyectos anteriores un Simulink que permite modificar los parámetros del control en tiempo real. Pero por motivos que se explican en el documento de la memoria, hemos ocupado todos los canales de la emisora para el control manual del aparato y no quedan canales libres para ajuste de parámetros en vuelo.

Para ello se ha añadido al archivo de Simulink de lectura de datos, llamado "**SERIAL_OPREVO_PC**", una serie bloques que permiten variar el valor de ciertos parámetros. Consta de unos diales circulares de escala definida por el usuario con gran utilidad para afinar las ganancias del control PID, cambiar la ponderación de acelerómetro y giróscopo en el estimador de estados, etc. Dichos parámetros se van a transmitir al cuadricóptero a través de la propia conexión Bluetooth de lectura. Este método de ajuste del control ha resultado ser un éxito, por lo que será muy útil para futuros proyectos.

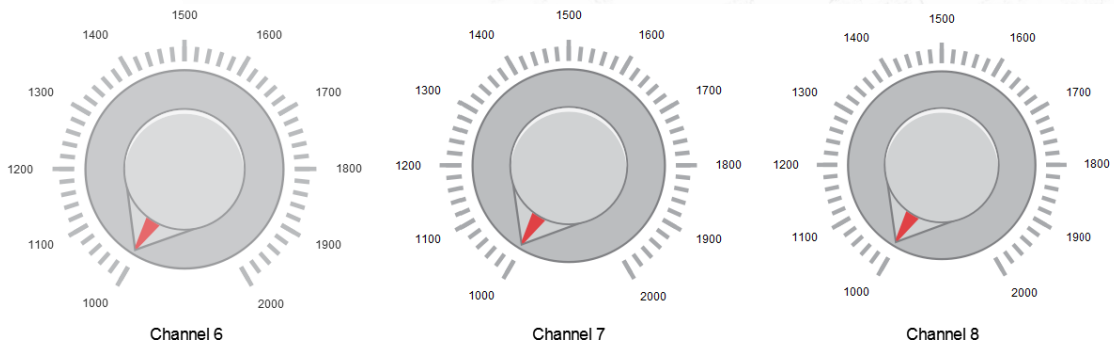


Figura 6.11. Diales utilizados en el ajuste de parámetros en vuelo.

6.5 Implementación del sensor en Simulink

Una vez que hemos configurado el sensor para que funcione de manera correcta, es la hora de crear una interfaz en Simulink para que el control interactúe con las medidas del sensor. Para ello se vuelve a hacer uso del ***Waijung Blockset*** para OpenPilot Revolution, y más concretamente de los bloques de comunicación **I2C**. A continuación se muestra la vista exterior del bloque de lectura del sensor. El interior del bloque de lectura se desarrollará con más detalle en el Anexo O.

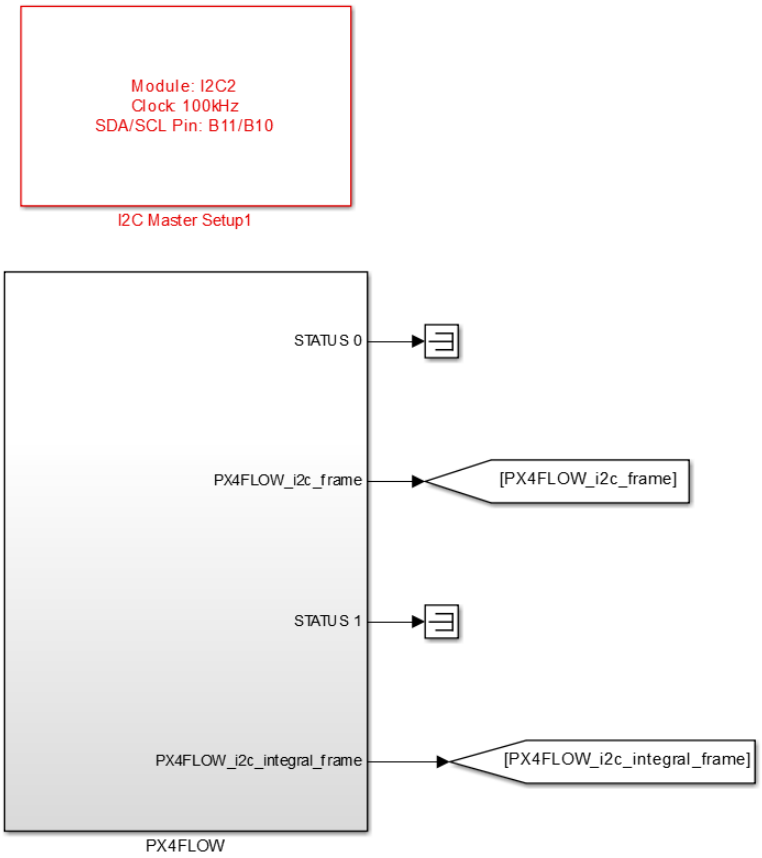


Figura 6.12. Esquema del bloque de lectura I2C del sensor

6.6 Entorno de Simulación

El objetivo principal por el cual se decide crear un entorno de simulación es tener que evitar hacer modificaciones al control diseñado a la hora de implantarlo sobre el hardware real. Se intenta que sea lo más parecido posible al fichero de implantación maestro "OPENPILOT_REVO.slx", por ello el bloque de "CONTROLLER" es exactamente el mismo tanto para simular como para volcar en la placa. A partir de ahí se han hecho modificaciones para incluir el modelo dinámico del cuadricóptero así como el entorno de monitorización, para poder medir todas las variables importantes y así poder afinar un buen control. También contiene una herramienta de generación de pulsos para emular la emisora, ya que en simulación lo controlamos todo desde el PC [*Figura 6.14*].

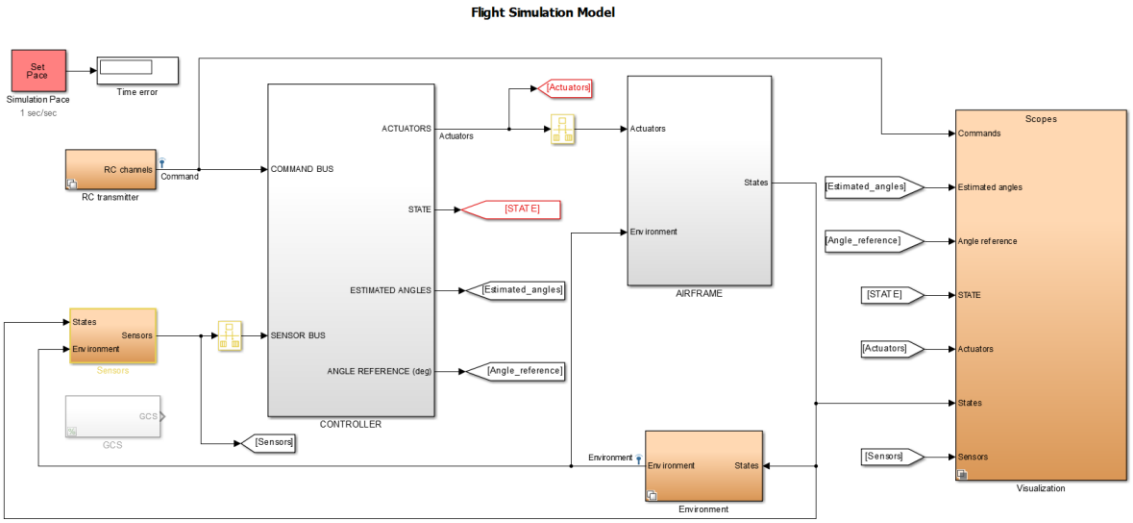


Figura 6.13. Vista general del Entorno de Simulación.

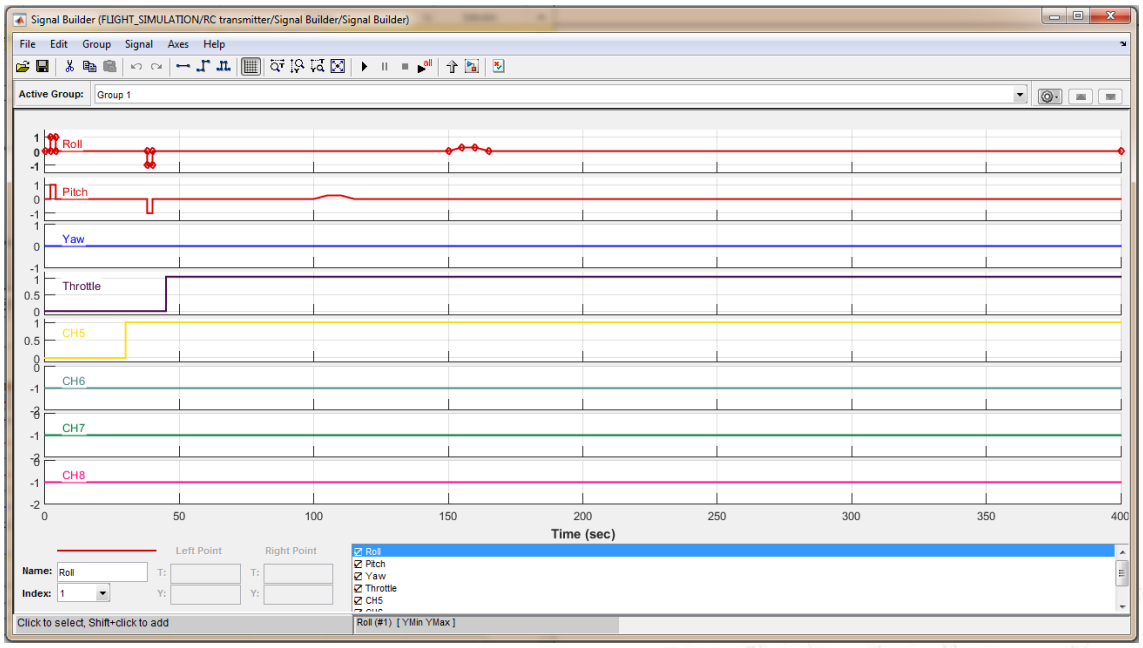


Figura 6.14. Vista del Generador de señales del simulador.

6.7 Ensayos en simulación

Una vez que el control se hubo implantado y que los parámetros se ajustaron, se realizaron ensayos de vuelo estacionario para comprobar con medidas el seguimiento de las referencias. Uno de los principales problemas durante los ensayos son los errores en la estimación de los ángulos de Euler. Un error en una de las variables realimentadas supone que el cuadricóptero no siga correctamente la referencia. El inconveniente que se deriva de esto durante el vuelo estacionario es que pequeñas inclinaciones el los ejes "x" e "y" suponen desplazamientos horizontales del cuadricóptero, a velocidades relativamente elevadas para interiores. Por esta razón es necesario realizar ligeras correcciones con la emisora, a fin de garantizar que la nave se mantenga en todo momento paralela al suelo y de esta manera mantenga fija su posición. Además, por el hecho de realizarse estos ensayos en un recinto cerrado con mobiliario, también es necesario en ocasiones corregir la posición del cuadricóptero para evitar que colisiones.

A continuación se muestran los resultados de simulación para escalones en las referencias de *roll*, *pitch* y *throttle*, aplicadas en tiempos distintos para evitar que se superpongan los resultados. En la simulación se han realimentado los ángulos estimados y además se han incluido ruidos en las medidas de giróscopos y acelerómetros, además de un pequeño offset para los giróscopos. El periodo de muestreo utilizado en el control es de 10 ms.

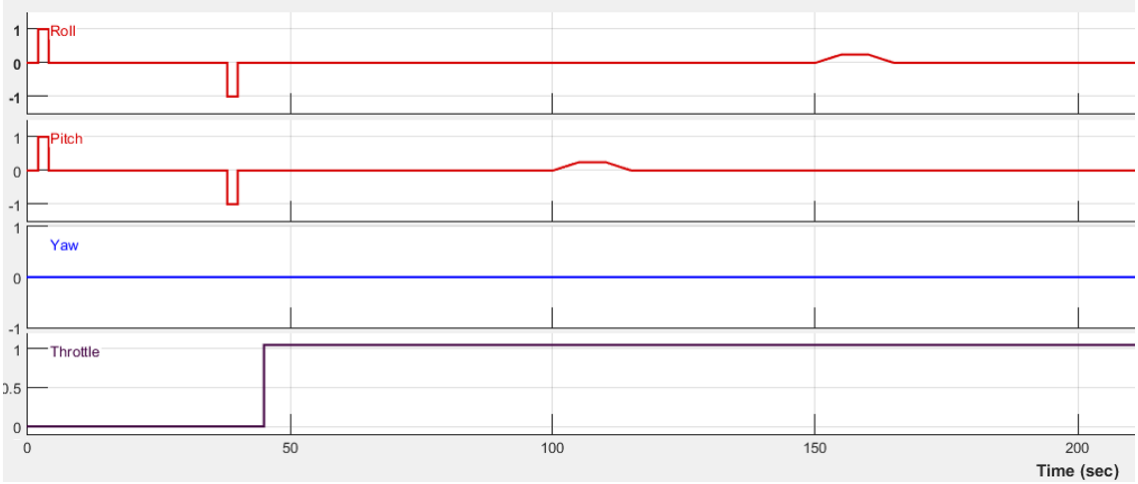


Figura 6.15. Configuración de los mandos para el ensayo de simulación.

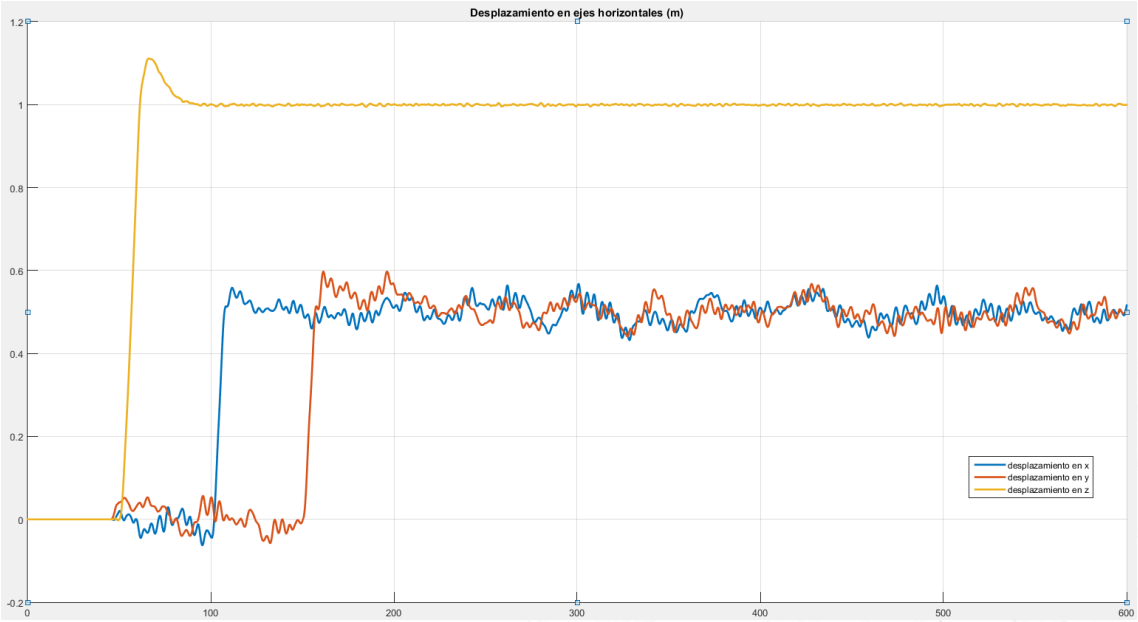


Figura 6.16. Desplazamiento en los 3 ejes en simulación.

Ahora vamos a analizar la respuesta que nos dan los ángulos de Euler a estos 3 escalones, puesto que es necesario comprobar que dichos ángulos varían de la forma adecuada para producir los desplazamientos de la simulación. A continuación se presentan las gráficas de los ángulos de roll y pitch, con el siguiente código de colores: en color azul la referencia de ángulo, en color rojo el ángulo real, y en color amarillo el ángulo estimado.

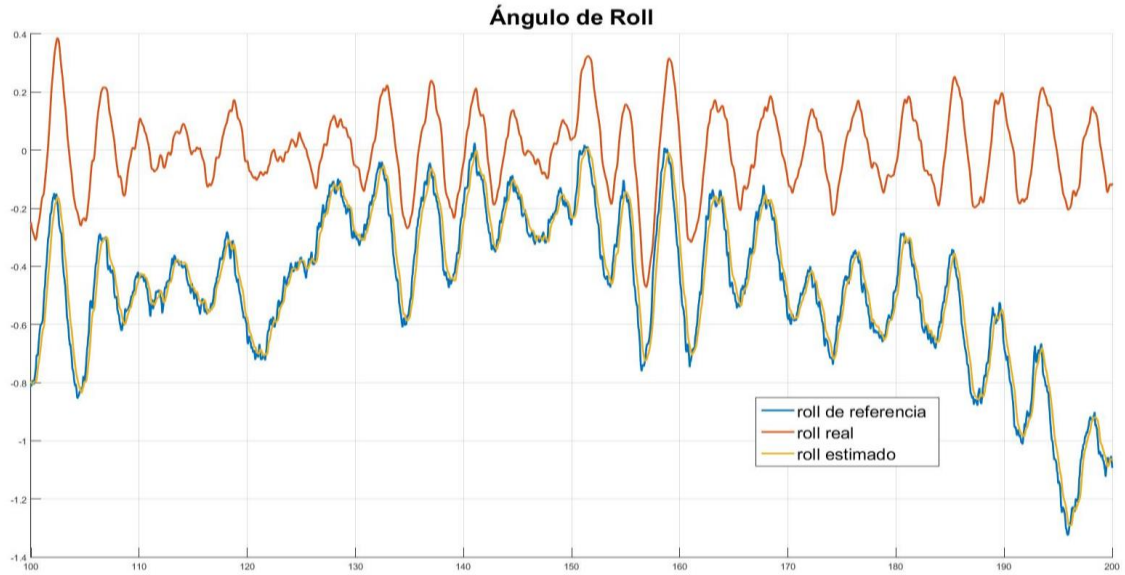


Figura 6.17. Representación del Roll en la simulación.

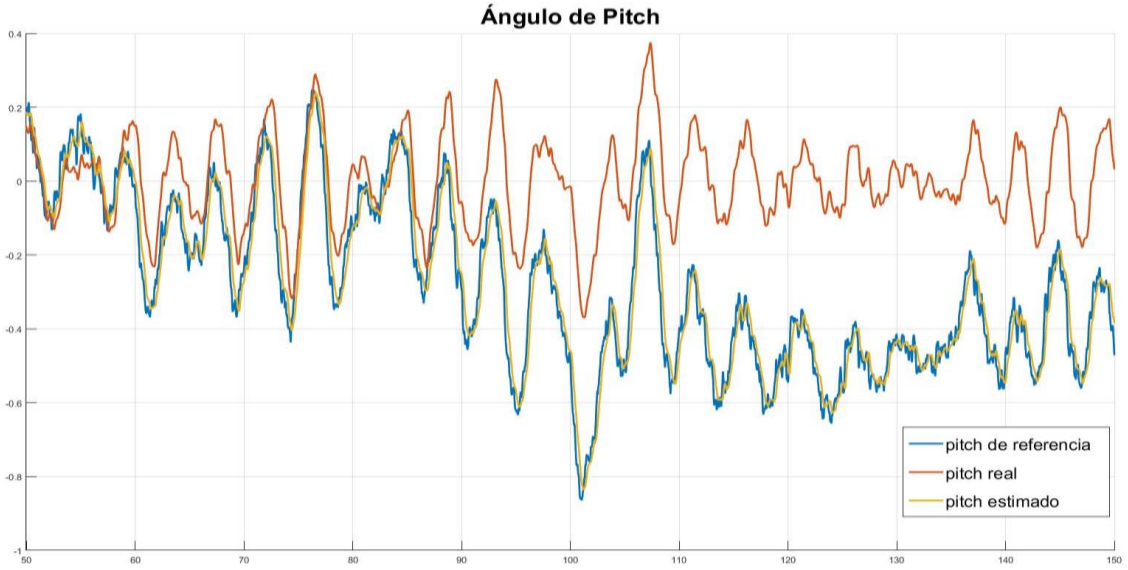


Figura 6.18. Representación del Pitch en la simulación.

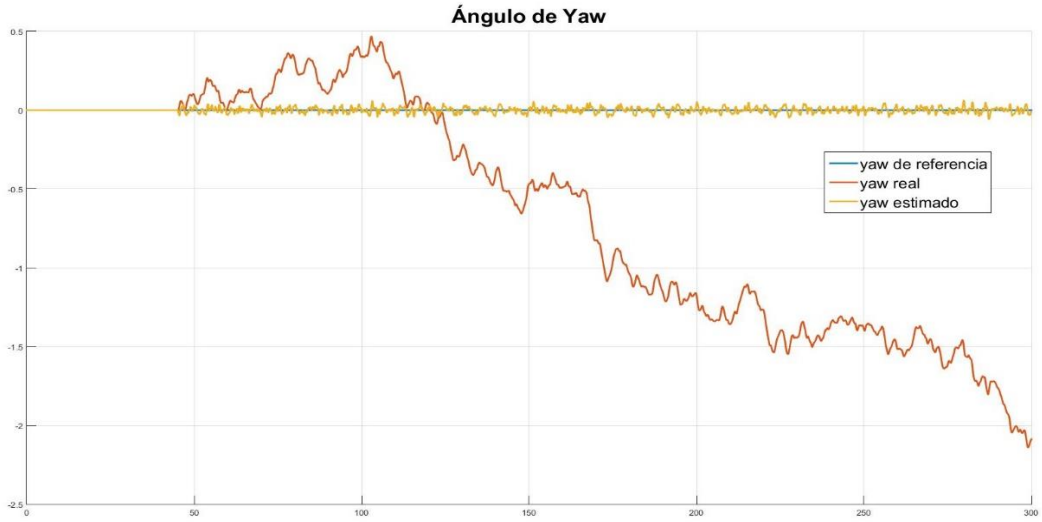


Figura 6.19. Representación del Yaw en la simulación.

Se puede observar que en general sigue la referencia con relativa precisión en el caso de los ángulos de *Roll* y de *Pitch*. El escalón del *Pitch* se aplica en t=100, y el de *Roll* en t=150, y se aprecia la variación de ambos ángulos con respecto al valor real. El valor estimado de los dos ángulos sigue de manera bastante precisa a la referencia, que es justo lo que queremos comprobar. Además se comprueba su estabilidad, ya que el ángulo tiende a 0 cuando la referencia se anula. Las variaciones del ángulo de *Yaw* no deben sorprendernos, ya que no es un ángulo que hayamos manejado para hacer ninguno de los desplazamientos, no es un ángulo controlado por lo tanto. Y nos ha dado una desviación de 2º, lo cual es un valor aceptable de error y posible de corregir en vuelo manual. Las desviaciones máximas del resto de ángulos no son nunca superiores a 1º.

Capítulo 7

Conclusiones y futuras mejoras

La finalidad de este proyecto era avanzar hacia la implantación de un control de orientación en recintos cerrados para poder realizar vuelos autónomos con un cuadricóptero. En este sentido se han realizado grandes avances, empezando por lograr la estabilización del cuadricóptero en vuelo manual, y consiguiendo un buen funcionamiento del sensor de flujo óptico. En este capítulo se expone un resumen del proyecto, así como una lista de posibles mejoras de cara a futuros proyectos que continúen esta línea de trabajo.

7.1 Resumen del trabajo realizado

A lo largo del proyecto se han llevado a cabo diversas tareas que permiten situar más cerca la culminación del objetivo final: el vuelo autónomo en interiores. A continuación se detallan las más relevantes, que a su vez pueden servir como una base sólida de partida de cara a futuros proyectos:

- ✓ Diseño del modelo del cuadricóptero basado en [6].
- ✓ Diseño de un entorno de configuración apropiado para el trabajo con la tarjeta de control OpenPilot Revolution haciendo uso de los bloques de Waijung
- ✓ Creación de un nuevo Entorno de Trabajo Conjunto de implementación y simulación único para todos los proyectos de drones, con la ayuda de la utilidad de "*Variant Manager*" de Simulink.
- ✓ Diseño de un fichero maestro único para unificar todos los proyectos de drones
- ✓ Diseño y montaje de un circuito de alimentación del sensor.
- ✓ Diseño del bloque de lectura del sensor de flujo óptico PX4FLOW, tanto para la trama I2C normal como la trama I2C integral.
- ✓ Mejora de la estructura empleada como cubierta del sensor.
- ✓ Diseño de un sistema de calibración de giróscopos y acelerómetros de manera que se elimine la componente continua de las medidas.
- ✓ Corrección del signo de las medidas de giróscopos y acelerómetros de la IMU.
- ✓ Mejora del estimador de ángulos mediante el uso del Filtro Complementario No Lineal.
- ✓ Diseño de un diagrama de bloques para modificar los parámetros del control en tiempo real desde el ordenador.
- ✓ Desarrollo de un sistema de telemetría inalámbrica mediante bluetooth para una rápida monitorización con el ordenador.
- ✓ Diseño de un bloque lector (Demodulador) de pulsos por PPM.
- ✓ Diseño de una nueva máquina de estados para solventar las dificultades encontradas de lectura única de 4 canales

7.2 Resumen de resultados obtenidos

En cuanto al grado de cumplimiento de los objetivos se puede afirmar que los objetivos principales del proyecto (adaptación del modelo y diseño del entorno de simulación, integración del sensor e implantación del control de vuelo manual) se han cumplido plenamente. Además, también se ha avanzado en la consecución del objetivo final de vuelo autónomo con el sensor óptico. Como resultados del proyecto se han conseguido los siguientes:

- ✓ Modelo no lineal completo del *Airframe* del cuadricóptero en Simulink, que incluye todas sus especificaciones y parámetros.
- ✓ Entorno Conjunto de Simulación completo de Simulink, con multitud de diferentes controles y aeronaves a elegir. Posee también una interfaz gráfica donde visualizar todas aquellas variables importantes para la mejora del control junto con un programa de navegación prediseñado.
- ✓ Entorno Conjunto de Simulink donde se recoge cada uno los elementos que forman parte del Control de la aeronave: la calibración de la IMU (giróscopos y acelerómetros), el Filtro Complementario No Lineal (Estimador de Estados) y por supuesto el Control de Estabilidad (permite mantener vuelo estacionario y seguir las referencias de ángulos enviadas desde la emisora). Todo gestionado por la máquina de estados para su buen funcionamiento, con las precauciones de seguridad pertinentes.
- ✓ Fichero de lectura de datos en Simulink de las principales variables del cuadricóptero, y desde donde modificar los parámetros del control, en pleno vuelo.
- ✓ Fichero del Sensor PX4FLOW en Simulink, con los bloques de escritura de registros y de lectura I2C, ordenados e integrados en un Bloque único de lectura de datos del sensor.

7.3 Mejoras Planeadas

De cara a los sucesivos proyectos que continúen en la línea de trabajo de éste, se han concretado una serie de mejoras para facilitar la consecución del objetivo final de vuelo autónomo en interiores con el sensor óptico.

- ✓ Estimación de altura combinando las medidas del sensor de flujo óptico y los sensores infrarrojos. También se pueden incluir sensores de ultrasonidos para aumentar el campo de visión y poder detectar el suelo o la pared a mayores distancias (de cara a vuelo exterior).
- ✓ Desarrollo de un control de avance y giro basado en la actuación sobre la referencia de cabeceo/alabeo, una vez que se quiera programar seguimiento de rutas en vuelo autónomo.
- ✓ Desarrollo de un sistema de telemetría para buscar una solución al problema de la limitación de canales debido a los Timers y sus conflictos de sincronización, para poder obtener mediante el demodulador la lectura directa de los 8 canales por PPM.
- ✓ Uso de control predictivo de cara a programar el dron para realizar complejas tareas en modo automático.
- ✓ Aumento de la velocidad de transmisión del módulo de transmisión por Bluetooth mediante un modelo más nuevo que soporte mayores velocidades, reprogramando y adaptando el protocolo maestro-esclavo.
- ✓ Debido a la dificultad que supone controlar de forma correcta el dron en modo manual, no estaría mal facilitar al usuario tutoriales y manuales de la emisora RC para aumentar rápidamente la pericia del usuario al manejar el aparato.

Capítulo 8

Referencias

[1] «OpenPilot Wiki,» [En línea]. Available: http://opwiki.readthedocs.io/en/latest/user_manual/revo/revo.html.

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

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

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

[5] 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.

[6] J. Martínez Olondo, «Control de un cuadricóptero para vuelos autónomos en interiores,» Universidad Pontificia de Comillas, 2015.

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

[8] L. Meier, P. Tanskanen, L. Heng, G. Hee Lee, F. Fraundorfer y M. Pollefeys, «PIXHAWK: A Micro Aerial Vehicle Design for Autonomous Flight using Onboard Computer Vision,» 2012.

[9] «OP Revo,» 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.

[10] E. García-Castrillón Triquell, «Optimización de funciones de DSP para procesador con instrucciones de aritmética completa,» Universidad Complutense de Madrid, 2008.

[11] «Analog,» [En línea]. Available: http://www.analog.com/library/analogDialogue/archives/37-03/gyro.html.

[12] «DIY Drones FrSKY PPM Issue,» [En línea]. Available: http://diydrones.com/profiles/blogs/why-frsky-cppm-signal-is-so-disappointing.

[13] 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.

[14] 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.

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

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

[17] 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.

[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] 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.

[20] 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.

[21] D. Gaydou, J. Redolfi y J. Henze, «Filtro complementario para estimación de actitud aplicado al controlador embebido de un cuatrirrotor,» Universidad Tecnológica Nacional, 2011.

[22] 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.

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

[24] J. J. Gibson, «The Perception of the Visual World,» Houghton Mifflin, 1950.

[25] G. Di Marco, «Optical flow general concepts and implemented algorithms,» 2013.

[26] H. Nagel, «On the estimation of optical flow: relations between different approaches and some results,» *Artificial Intelligence,* nº 33, pp. 299-324, 1987.

[27] G. Di Marco, L. Marconi y R. Naldi, «Integration of optical flow based vision algorithms for the control of an unmanned aircraft,» 2014.

[28] B. D. Lucas y T. Kanade, «An Iterative Image Registration Technique with an Application to Stereo Vision,» *Proceedings of Imaging Understanding Workshop,* pp. 21-130, 1981.

[29] J.-Y. Bouguet, «Pyramidal Implementation of the Affine Lucas Kanade Feature Tracker».

[30] B. Kitt, B. Ranft y H. Lategahn, «Block-Matching based Optical Flow Estimation with Reduced Search Space based on Geometric Constraints,» de *13th International IEEE Annual Conference on Intelligent Transportation Systems*, Madeira Island, 2010.

[31] J. N. Pan, Y. Q. Shi y C. Q. Shu, «Correlation-feedback technique in optical flow determination,» IEEE, 1998.

[32] Matlab, «Optical flow for motion estimation in video,» [En línea]. Available: http://es.mathworks.com/discovery/optical-flow.html.

[33] D. Honegger, L. Meier, P. Tanskanen y M. Pollefeys, «An Open Source and Open Hardware Embedded Metric Optical Flow for indoor and outdoor applications,» *(ICRA) IEEE International Conference.*

[34] «PX4FLOW Overview,» [En línea]. Available: http://ardupilot.org/copter/docs/common-px4flow-overview.html.

[35] AmperCZ, «PX4FLOW Cover,» Thingiverse, [En línea]. Available: http://www.thingiverse.com/thing:547631.

[36] C. Commons, «Attribution-NonCommercial License,» [En línea]. Available: https://creativecommons.org/licenses/by-nc/3.0/.

[37] «HKPilot Mega,» HobbyKing, [En línea]. Available: http://www.hobbyking.com/hobbyking/store/__56052__HKPilot_Mega_2_7_Flight_Controller_USB_GYRO_ACC_MAG_BARO.html..

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

Anexo A

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

```
%%%%%%%%%%%%%%%%%%%%%%%%
% MASTER CONFIG SCRIPT %
%%%%%%%%%%%%%%%%%%%%%%%%

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=10e-3;
% Sampling time for UART
ts_UART=20e-3;
% Sampling time for ATTTITUDE CONTROL
ts_ATT_CONTROL=10e-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')
% PX4FLOW
run('..\HARDWARE_COMPONENTS\config_PX4FLOW')
% LIDAR LITE
run('..\HARDWARE_COMPONENTS\config_LIDAR_LITE')

%% 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;


%% SENSOR DYNAMICS PARAMETERS
SensorParameters
```

```
%% 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=0.995;

% Calibration filter for POSITION
alfa_CAL_POS=0.95;
```

```
%% 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 ...
   49  170  44  83  46   0 ]);

return
```

Anexo B

En este anexo se recoge el código incluido en el script **ModelParam_QuadNestor.m**. En él se encuentran recogidas todas las especificaciones físicas de la aeronave de este proyecto que se usarán tanto en el simulador como en el control de vuelo.

```
%%%%%%%%%%%%%%%%%%%%
% MODEL PARAMETERS %
%%%%%%%%%%%%%%%%%%%%

% 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 = 0;

%%%%% Airframe %%%%%%
% Frame: 1: + / 2: x
UAV = struct('Frame',2);
% Mass (kg)
UAV.Mass = 0.5427;
% Arm Length for roll and pitch torques (m)
UAV.RollArmLength=0.2050/2;
UAV.PitchArmLength_Front=0.1652/2;
UAV.PitchArmLength_Rear=0.1652/2;
% 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 = 2300;
% 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 = 4*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.ThrustData = [0  37 82 140 200 250 303 370 413 450
450.1]/1000*Gravity;
% Motor.ThrustData = [0  17 37  49  65  74  93 114 137 155 172
]/1000*Gravity;
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 el script **config_PX4FLOW.m**. En él se encuentran los parámetros necesarios para un correcto funcionamiento del sensor.

```
%%%%%%%%%%%%%%%%%%%%%%
% PX4FLOW PARAMETERS %
%%%%%%%%%%%%%%%%%%%%%%

% Sampling time
ts_PX4FLOW = 10e-3;
% Ininitialization value
INIT = hex2dec('0');


%%%%%%%%%%%%%%%%%%%%%%
% PX4FLOW Registers %
%%%%%%%%%%%%%%%%%%%%%%

%define PX4FLOW_ADDRESS 0x42
PX4FLOW_ADDRESS = 2*hex2dec('42'); % 0x42 desplazando 1 bit a la
izq;
%define PX4FLOW_TIMEOUT 10
PX4FLOW_TIMEOUT = 10;    % 10ms
%define PX4FLOW_DEBUG true
PX4FLOW_DEBUG = boolean(1);    %true;
```

Anexo D

En este anexo se recoge el código incluido en **PIDAttitudeControl.m,** la *Matlab Function* encargada del Control de Estabilización utilizado en este proyecto. Se puede observar cómo se habilitarán ciertos componentes según se ha elegido un tipo u otro de aeronave (cuadricóptero, helicóptero o tricóptero). Aparecen incluidas las ecuaciones de diseño del control.

```
%%%%%%%%%%%%%%%%%%%%
% ATTITUDE 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 E

En este anexo se recoge el código incluido en **PIDNavigationControl.m**, la Matlab Function encargada del Control de Navegación utilizado en este proyecto. Al igual que en el control de estabilidad, también se habilitarán ciertos componentes según se ha elegido un tipo u otro de aeronave (cuadricóptero, helicóptero o tricóptero).

```
%%%%%%%%%%%%%%%%%%%%%%
% NAVIGATION CONTROL %
%%%%%%%%%%%%%%%%%%%%%%

function Navigation = PIDNavigationControl(ts_NAV_CONTROL)

% Periodo de control (s)
Navigation  =  struct('tc',ts_NAV_CONTROL);
% Natural frequency (rad/s)
Navigation.NaturalFrequency = 0.25;
% Damping
Navigation.Damping = 1;
% Natural frequency (rad/s)
Navigation.NaturalFrequencyHeight = 0.25;
% Damping
Navigation.DampingHeight = 1;


%% Flight Controller PID
%_____________________________________________________________________
_____
%     CONTROL PID DE NAVEGACION
%_____________________________________________________________________
_____
% Vector de estado (sistema de referencia inercial)
% X = [x y z vx vy vz]'
% U = [Phi_ref  Theta_ref  Shi_ref  throttle_ref]'

% Body forces
% Force in X axis: u1
% Force in Y axis: u2
% Thrust or force in Z axis: u3
% throttle_ref = -u3/(m*g)

% Acceleration in earth reference frame
% dvxyz_e = [0 ; 0 ; g] + DCM_be_*[u1 ; u2 ; u3]/m = [ux ; uy ; uz]
% DCM_be =
%  [ 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)                          ];
```

```
% Linearized DCM_be =
%  [ cos(Shi)            Phi_ref*Theta_ref*cos(Shi)-sin(Shi)
Theta_ref*cos(Shi)+Phi_ref*sin(Shi)
%    sin(Shi)            Phi_ref*Theta_ref*sin(Shi)+cos(Shi)
Theta_ref*sin(Shi)-Phi_ref*cos(Shi)
%   -Theta_ref           Phi_ref
1                           ];

%%%%%%%%% X axis equation %%%%%%%%%%
% ux = cos(Shi)*u1/m + (cos(Shi)*Phi_ref*Theta_ref-sin(Shi))*u2/m +
...
%    + (Theta_ref*cos(Shi)+Phi_ref*sin(Shi))*u3/m
% ux/g - cos(Shi)*u1/m/g + sin(Shi)*u2/m/g = ...
%   = [cos(Shi)*u2/m/g  -sin(Shi) -cos(Shi)]* ...
%     *[Phi_ref*Theta_ref  Phi_ref*throttle_ref
Theta_ref*throttle_ref]'
%%%%%%%%% Y axis equation %%%%%%%%%%
% uy = sin(Shi)*u1/m + (sin(Shi)*Phi_ref*Theta_ref+cos(Shi))*u2/m +
...
%    + (Theta_ref*sin(Shi)-Phi_ref*cos(Shi))*u3/m
% uy/g - sin(Shi)*u1/m/g - cos(Shi)*u2/m/g = ...
%   = [sin(Shi)*u2/m/g  cos(Shi) -sin(Shi)]* ...
%     *[Phi_ref*Theta_ref  Phi_ref*throttle_ref
Theta_ref*throttle_ref]'
%%%%%%%%% Z axis equation %%%%%%%%%%
% uz = -Theta_ref*u1/m + Phi_ref*u2/m + u3/m + g
% uz/g = -Theta_ref*u1/m/g + Phi_ref*u2/m/g - throttle_ref + 1
% uz/g - 1 = [u2/m/g  -u1/m/g  -1]*[Phi_ref Theta_ref throttle_ref]'
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%% Notation %%%%%%%%%
% x1 = Phi_ref ; x2 = Theta_ref ; x3 = throttle_ref
% b1 = ux/g - cos(Shi)*u1/m/g + sin(Shi)*u2/m/g
% b2 = uy/g - sin(Shi)*u1/m/g - cos(Shi)*u2/m/g
% b3 = uz/g - 1
% matA = [aij] = [ cos(Shi)*u2/m/g  -sin(Shi) -cos(Shi)
%                  sin(Shi)*u2/m/g   cos(Shi) -sin(Shi)
%                       u2/m/g       -u1/m/g     -1     ]
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%% Equation system %%%%%
% b1 = a11*x1*x2 + a12*x1*x3 + a13*x2*x3
% b2 = a21*x1*x2 + a22*x1*x3 + a23*x2*x3
% b3 = a31*x1    + a32*x2    + a33*x3

%%%%% Theta_ref computation %%%%%
%  b1*cos(Shi) + b2*sin(Shi) = u2/m/g*x1*x2 - x2*x3 = x2*(u2/m/g*x1 -
x3) =
%                             = x2*(b3 + u1/m/g*x2)
% u1/m/g*x2^2 + b3*x2 - (b1*cos(Shi) + b2*sin(Shi)) = 0 => x2 (2
solutions)
% b1*cos(Shi) + b2*sin(Shi) = ux/g*cos(Shi) + uy/g*sin(Shi) - u1/m/g
% u1/m/g*x2^2 + (uz/g - 1)*x2 - ux/g*cos(Shi) - uy/g*sin(Shi) + u1/m/g
= 0
% x2 = Theta_ref =
%   = ((1-uz/g) + sqrt((1-uz/g)^2 - 4*u1/m/g*(-ux/g*cos(Shi)-
uy/g*sin(Shi)+u1/m/g)))/(2*u1/m/g)
% if u1 = 0  => x2 = Theta_ref = (-ux/g*cos(Shi) - uy/g*sin(Shi))/(1-
uz/g)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
%%%%% Phi_ref computation %%%%%
% -b1*sin(Shi) + b2*cos(Shi) = x1*x3
% -b1*sin(Shi) + b2*cos(Shi) = -ux/g*sin(Shi) + uy/g*cos(Shi) - u2/m/g
% x1*x3 = -ux/g*sin(Shi) + uy/g*cos(Shi) - u2/m/g
% uz/g - 1 = u2/m/g*x1 - u1/m/g*x2 - x3
% (uz/g - 1 + u1/m/g*x2)*x1 = u2/m/g*x1^2 + ux/g*sin(Shi) -
uy/g*cos(Shi) + u2/m/g
% u2/m/g*x1^2 + (1 - uz/g - u1/m/g*x2)*x1 + ux/g*sin(Shi) -
uy/g*cos(Shi) + u2/m/g = 0
% if u2 = 0  => x1 = Phi_ref = (-ux/g*sin(Shi) + uy/g*cos(Shi))/(1 -
uz/g - u1/m/g*x2)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%% throttle_ref computation %%%%%
% x3 = throttle_ref = 1 - uz/g + u2/m/g*Phi_ref - u1/m/g*Theta_ref
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Control PID de posicion
% Control PID
% C(s) = P + I/s + D*s
% F(s) = (D*s^2 + P*s + I) / (s^3 + D*s^2 + P*s + I)


%%%%%%%     X and Y axis
wn = Navigation.NaturalFrequency;
seta = Navigation.Damping;
p = -5*wn;

% Polinomio denominador lazo cerrado
% (s^2 + 2*seta*wn + 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_x = a1;
I_x = a0;
D_x = a2;
Navigation.K_x = P_x;
Navigation.invTi_x = I_x/P_x;
Navigation.Td_x = D_x/P_x;
Navigation.b_x = 1;
Navigation.N_x = 3;
P_y = a1;
I_y = a0;
D_y = a2;
Navigation.K_y = P_y;
Navigation.invTi_y = I_y/P_y;
Navigation.Td_y = D_y/P_y;
Navigation.b_y = 1;
Navigation.N_y = 3;
```

```
%%%%%%%   Z axis
wn = Navigation.NaturalFrequencyHeight;
seta = Navigation.DampingHeight;
p = -5*wn;
% Polinomio denominador lazo cerrado
% (s^2 + 2*seta*wn + 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_z = a1;
I_z = a0;
D_z = a2;
Navigation.K_z = P_z;
Navigation.invTi_z = I_z/P_z;
Navigation.Td_z = D_z/P_z;
Navigation.b_z = 1;
Navigation.N_z = 3;

return
```

Anexo F

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');
```

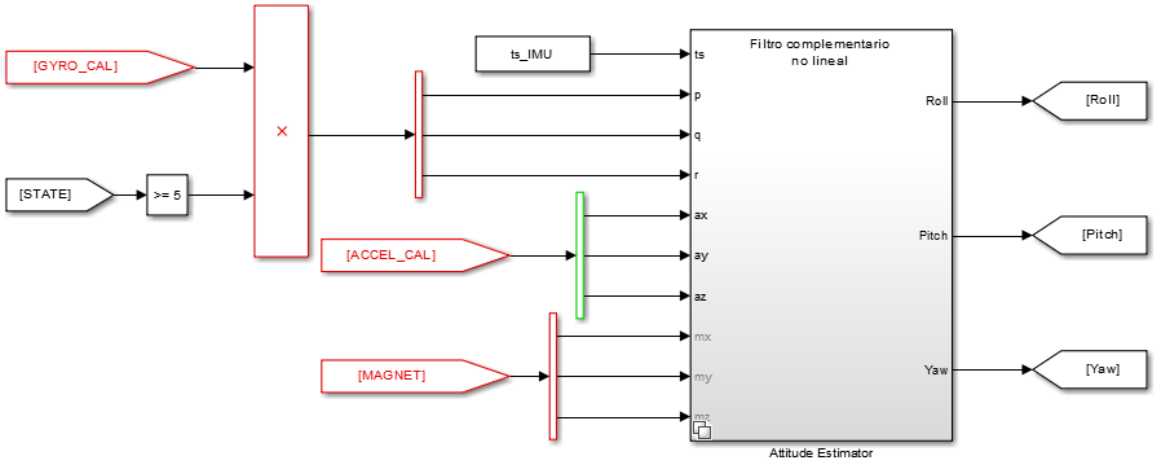
```
MPU60X0_I2C_SLV0_ADDR = hex2dec('25');
MPU60X0_I2C_SLV0_ADDR_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV0_REG = hex2dec('26');
MPU60X0_I2C_SLV0_REG_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV0_CTRL = hex2dec('27');
MPU60X0_I2C_SLV0_CTRL_BYTE = bin2dec('00000000');
MPU60X0_I2C_SLV1_ADDR = hex2dec('28');
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'
```

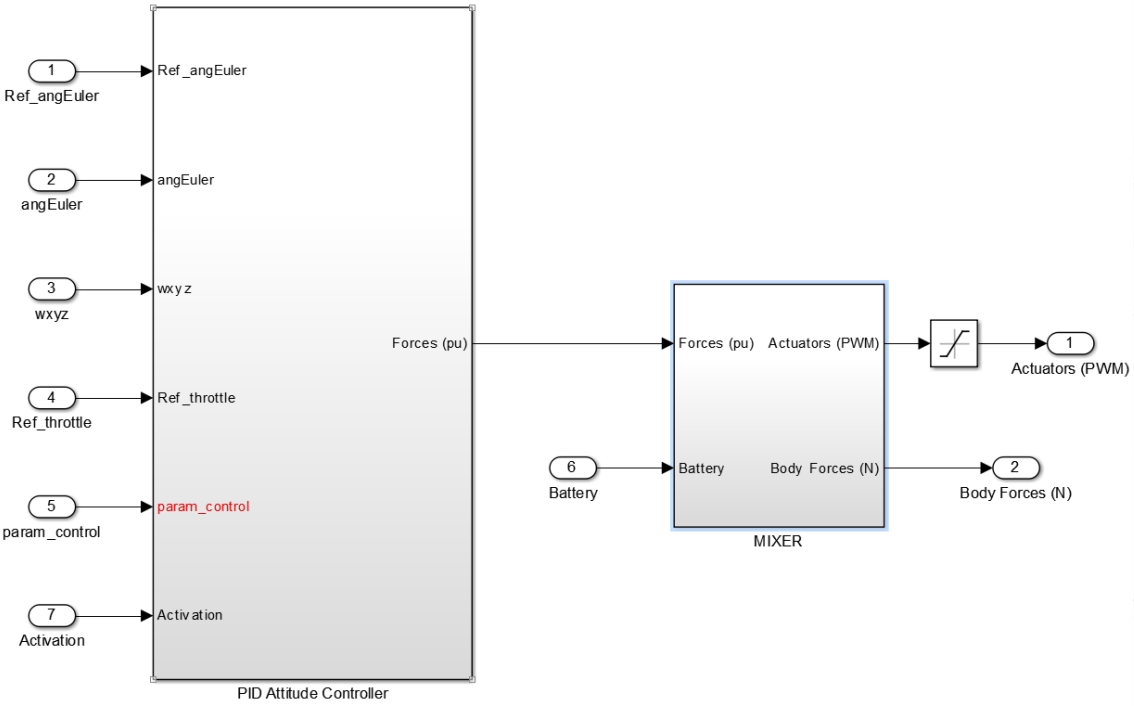
Anexo G

Aquí se detallan los diagramas de Simulink del interior del bloque de control general llamado "**CONTROLLER.slx**", en concreto los bloques más importantes.

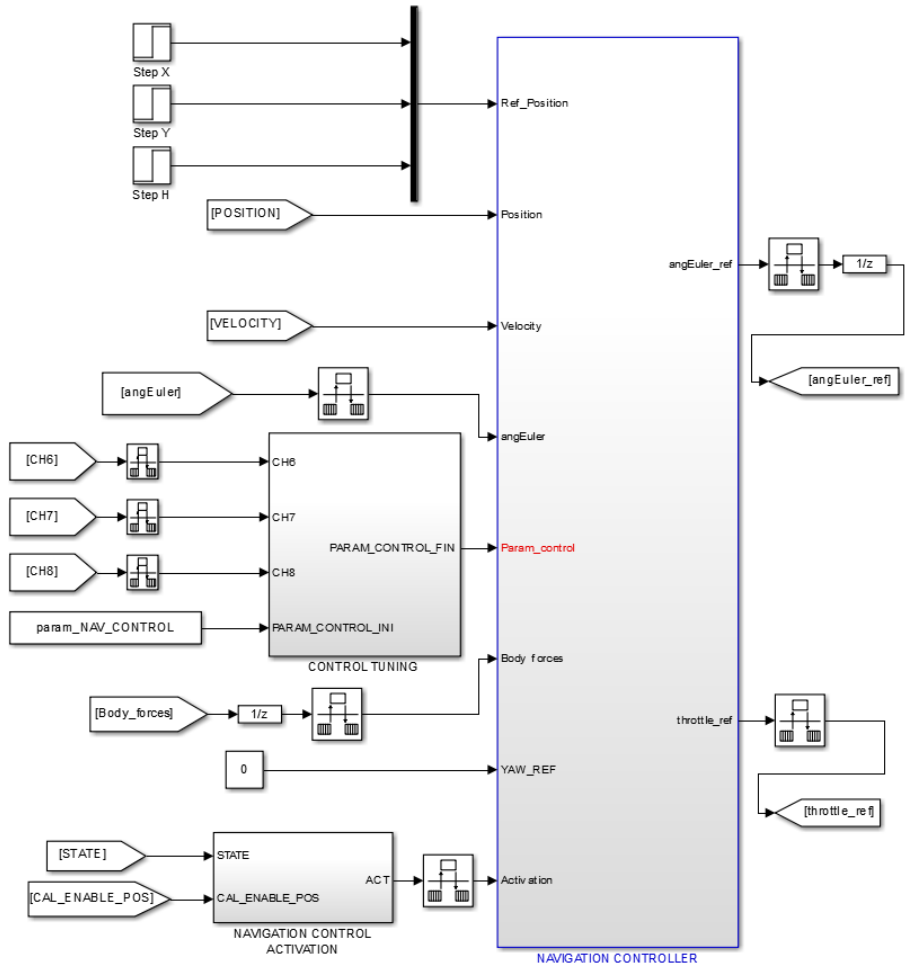
- ***Filtro Complementario No Lineal:***



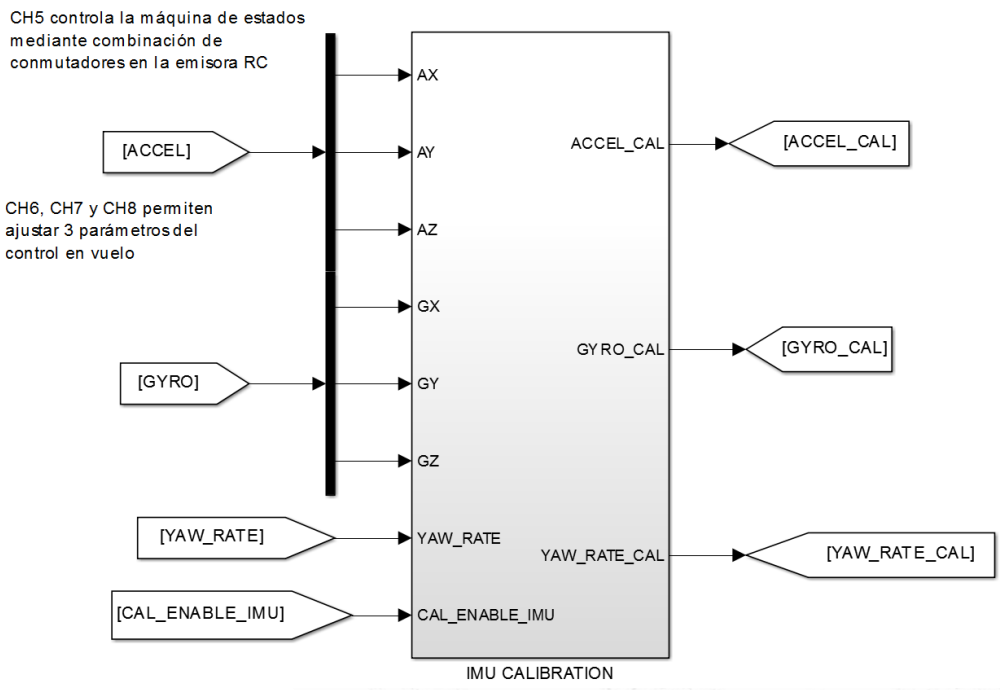
- ***Control de Estabilidad:***



➢ ***Control de Navegación:***



➢ ***Calibracion de la IMU:***



Anexo H

Aquí se detallan los diagramas de Simulink del interior del bloque que contiene el **Control de Estabilidad** empleado para conseguir que el cuadricóptero pueda volar con control manual.

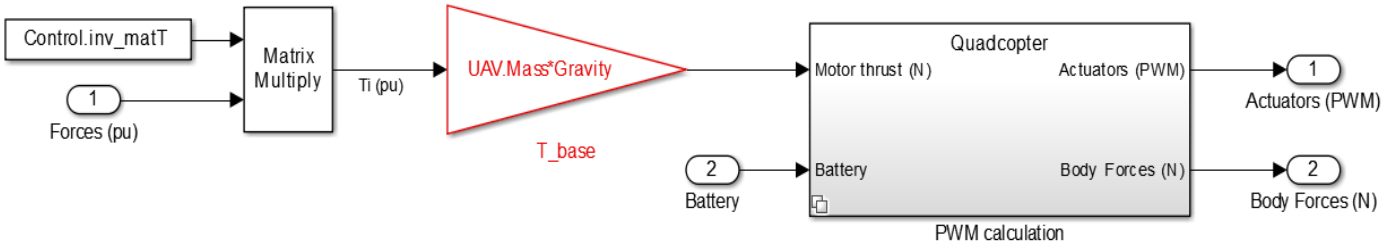


Figura 8.1. Bloque "Mixer".

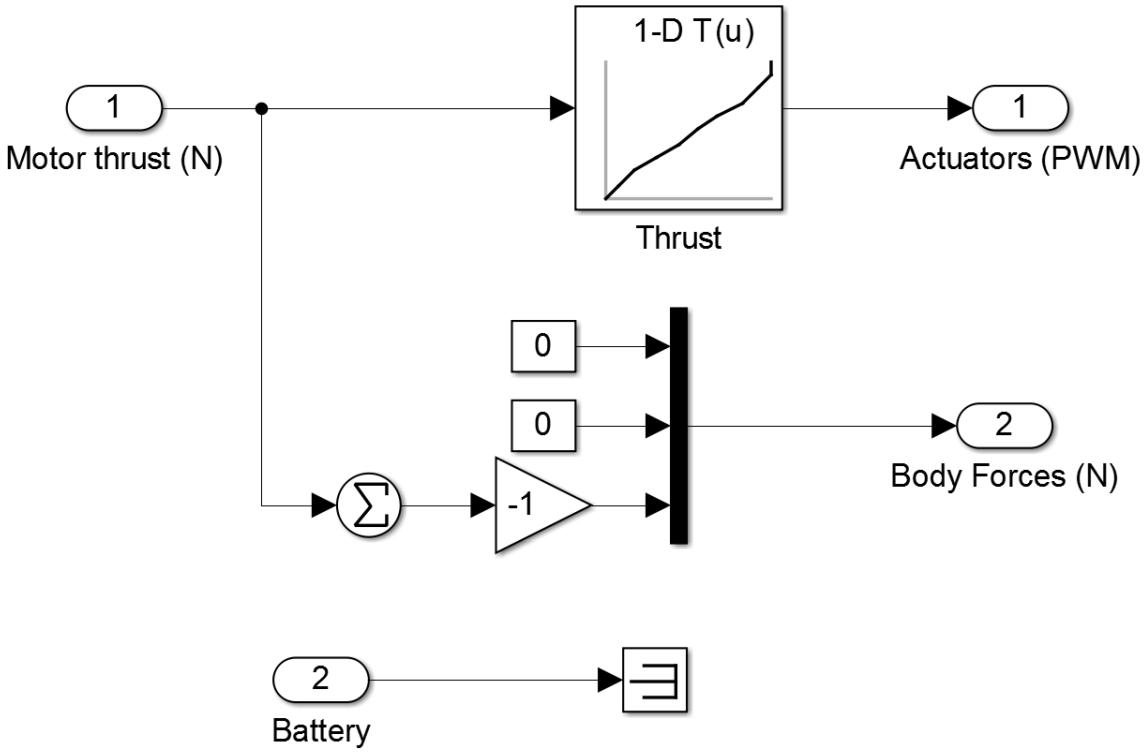
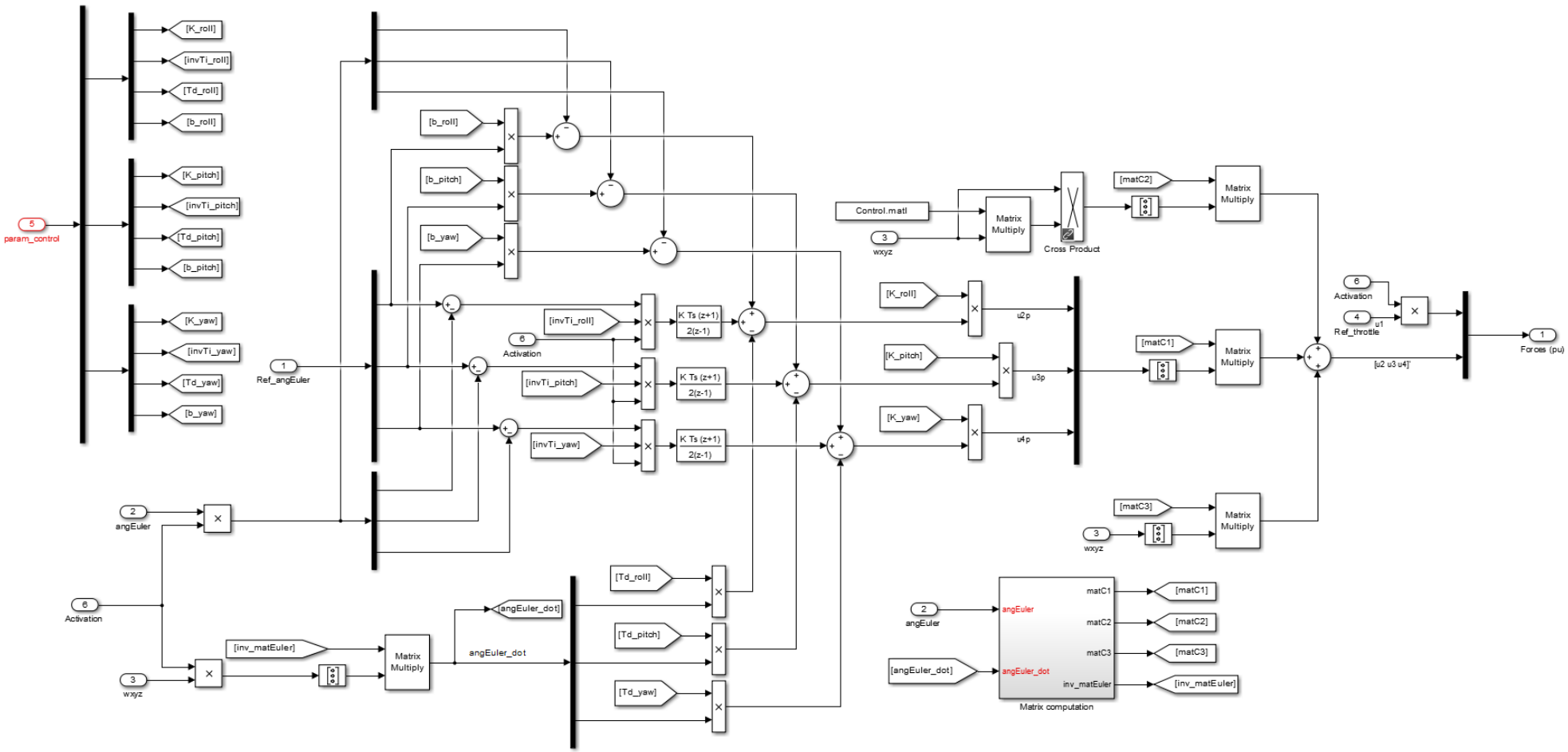
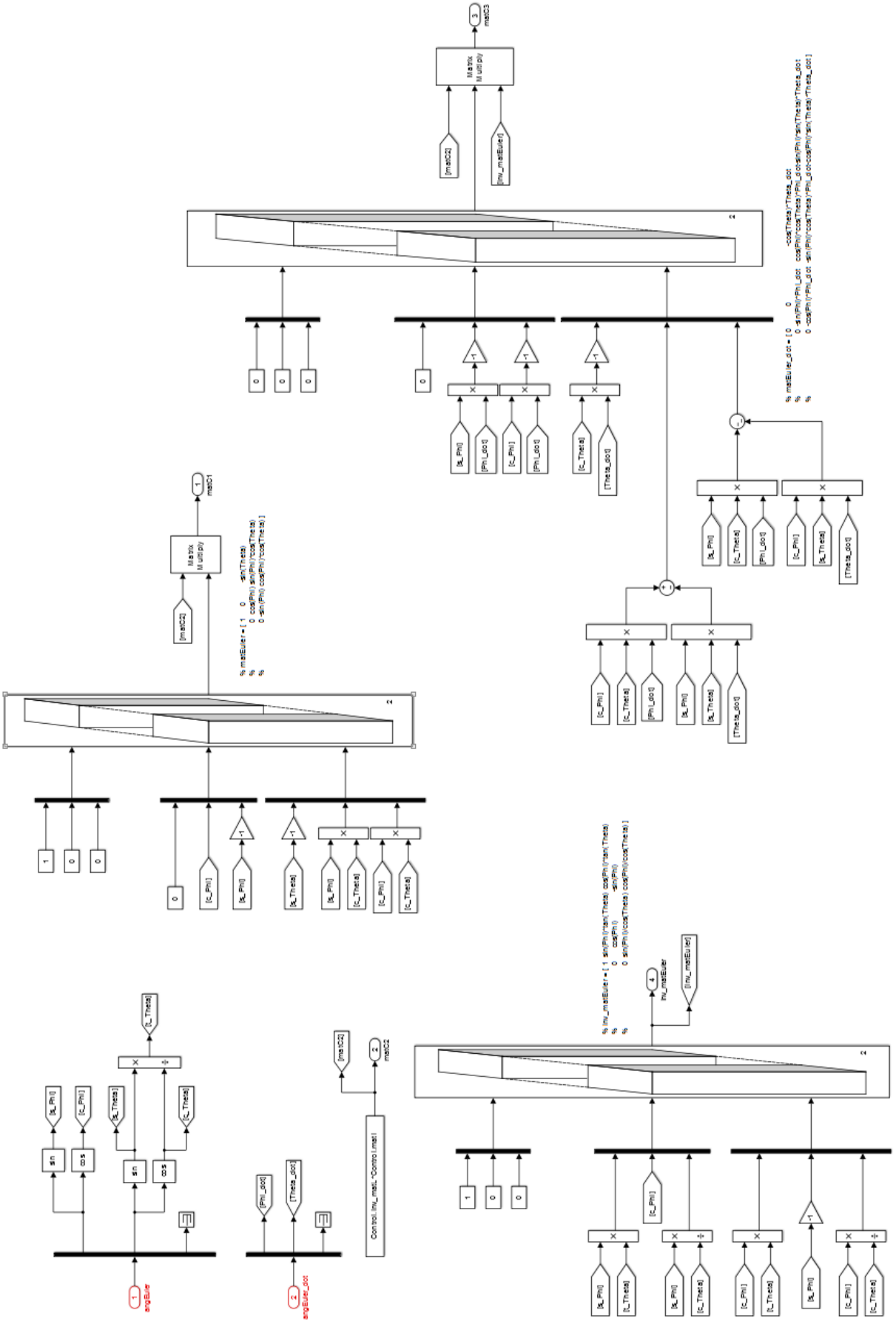


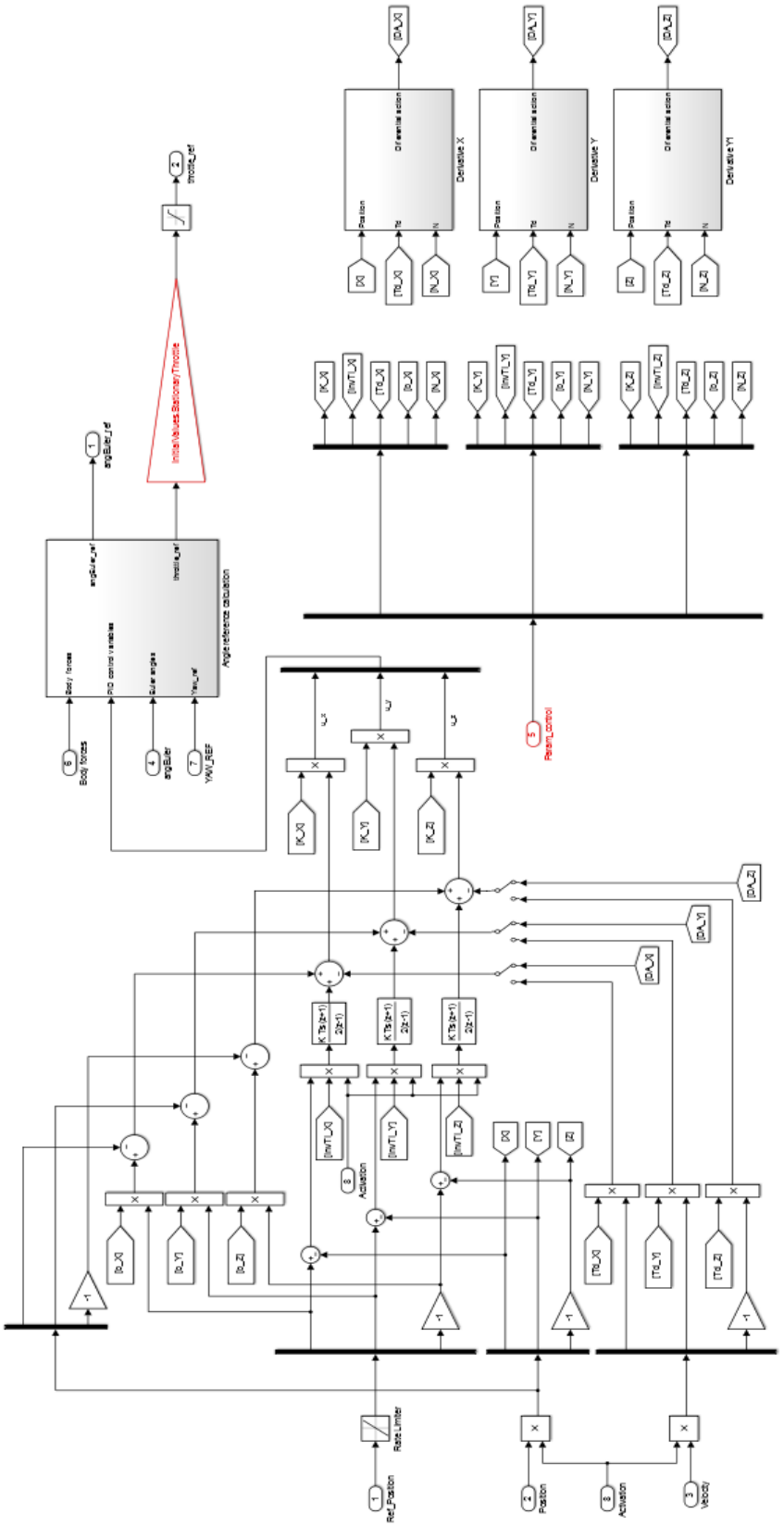
Figura 8.2. Bloque "PWM calculation".


5
param_control
[K_roll]
[invTi_roll]
[Td_roll]
[b_roll]
[K_pitch]
[invTi_pitch]
[Td_pitch]
[b_pitch]
[K_yaw]
[invTi_yaw]
[Td_yaw]
[b_yaw]
1
Ref_angEuler
6
Activation
2
angEuler
3
wxyz
[inv_matEuler]
Matrix Multiply
[angEuler_dot]
angEuler_dot
K Ts (z+1) 2(z-1)
Control.matl
Cross Product
[matC2]
[matC1]
[matC3]
u2p
u3p
u4p
4
Ref_throttle
u1
[u2 u3 u4]'
Forces (pu)
matC1
matC2
matC3
inv_matEuler
Matrix computation



Anexo I

En este anexo se muestra el diagrama de Simulink del interior del bloque que contiene el **Control de Navegación** empleado para conseguir que el cuadricóptero pueda volar de forma autónoma.



Anexo J

En este anexo se detallan la estructura interna del **Filtro Complementario No Lineal** que se ha utilizado para la estimación de ángulos por parte de la tarjeta de control del tricóptero.

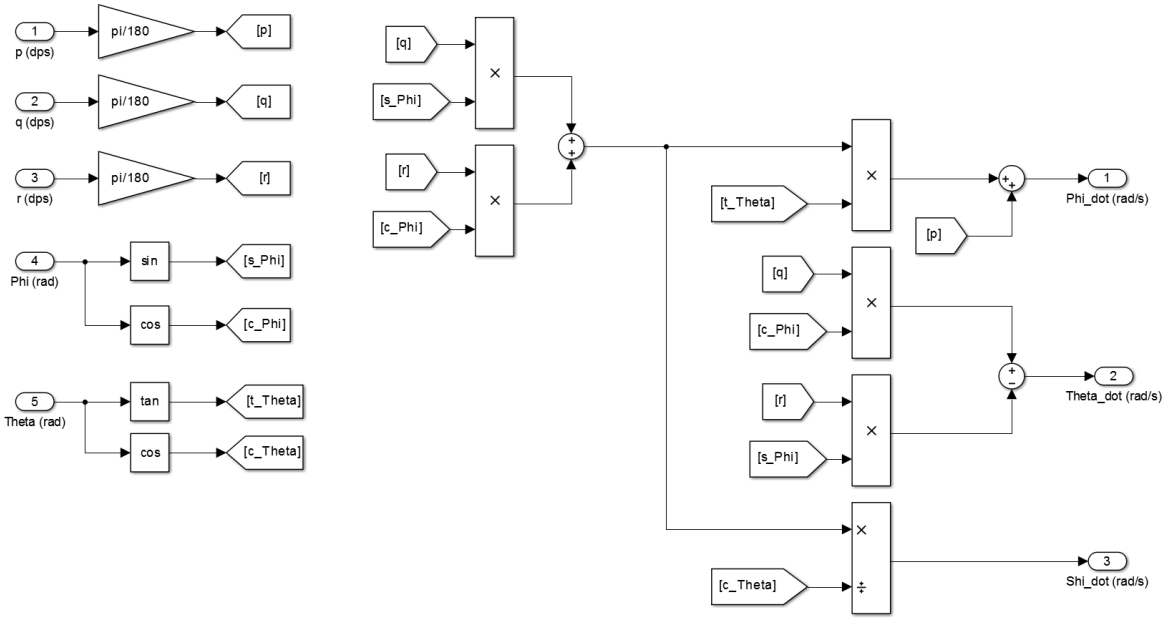


Figura 8.3.Bloque "Velocidades angulares de Euler".

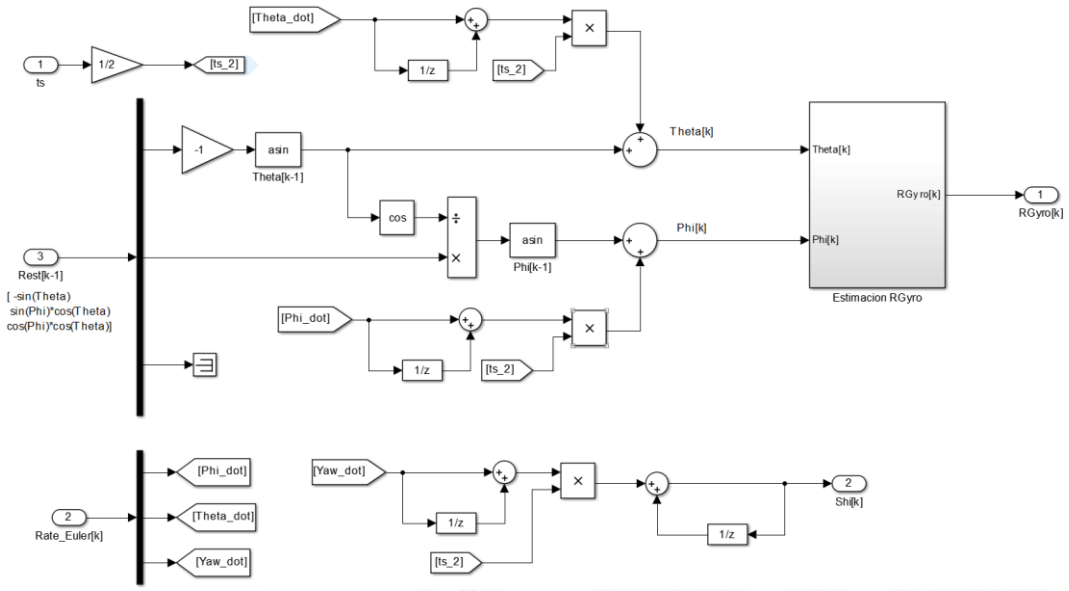


Figura 8.4. Bloque "RGyro[k]".

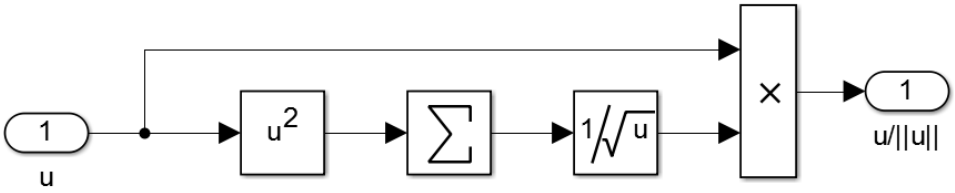


Figura 8.5. Bloque "Normalizacion"

Anexo K

En este anexo se detallan la estructura interna del Bloque de **Calibración de la Unidad de Medición Inercial** (IMU), en especial el conexionado de los filtros de calibración de los giróscopos y acelerómetros.

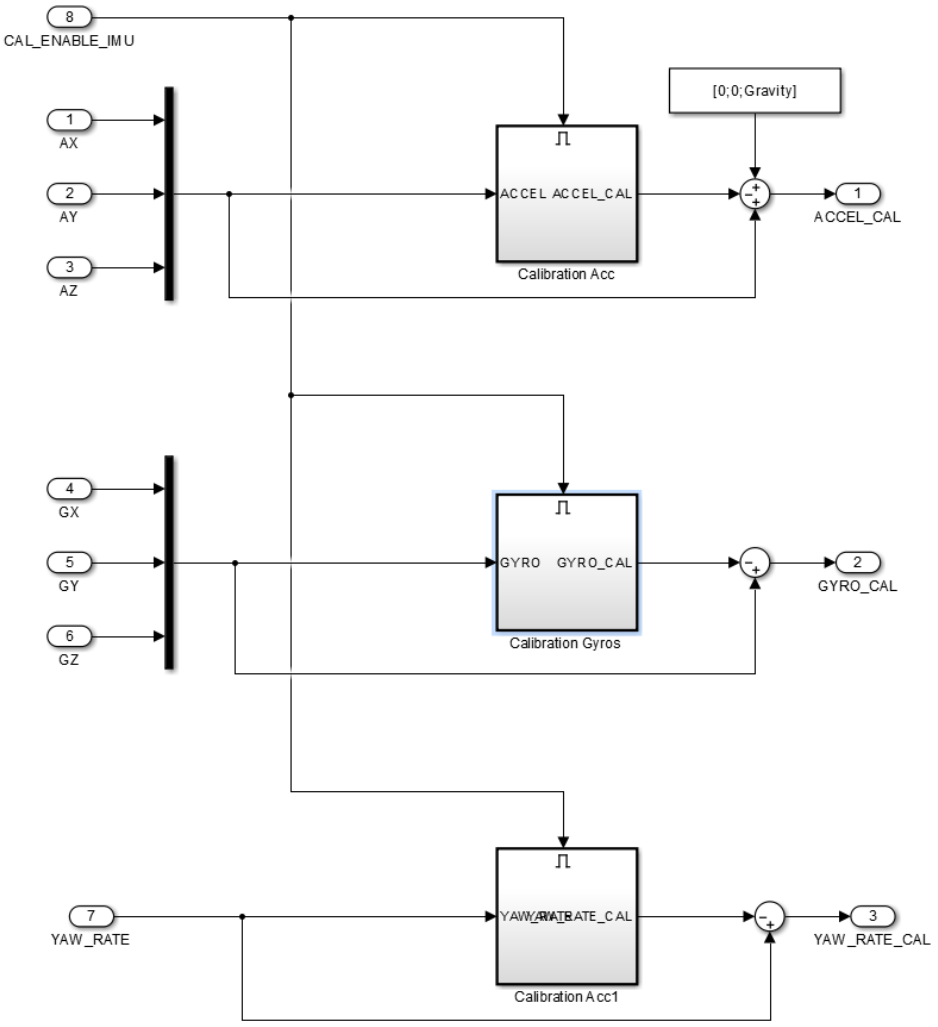


Figura 8.6. Estructura interna de la IMU.

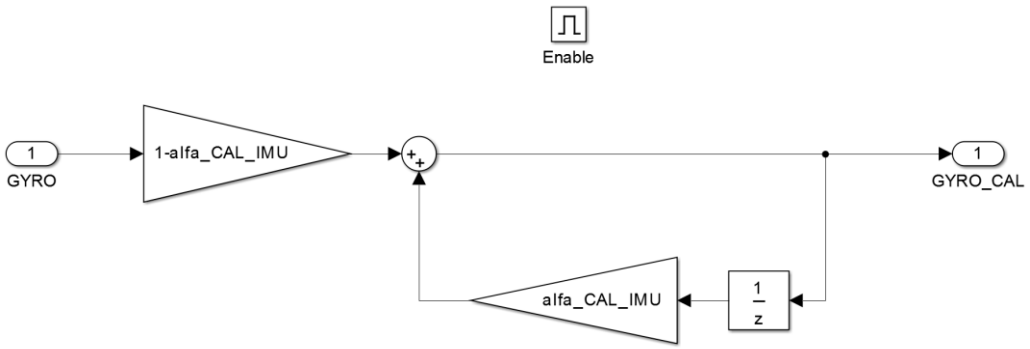
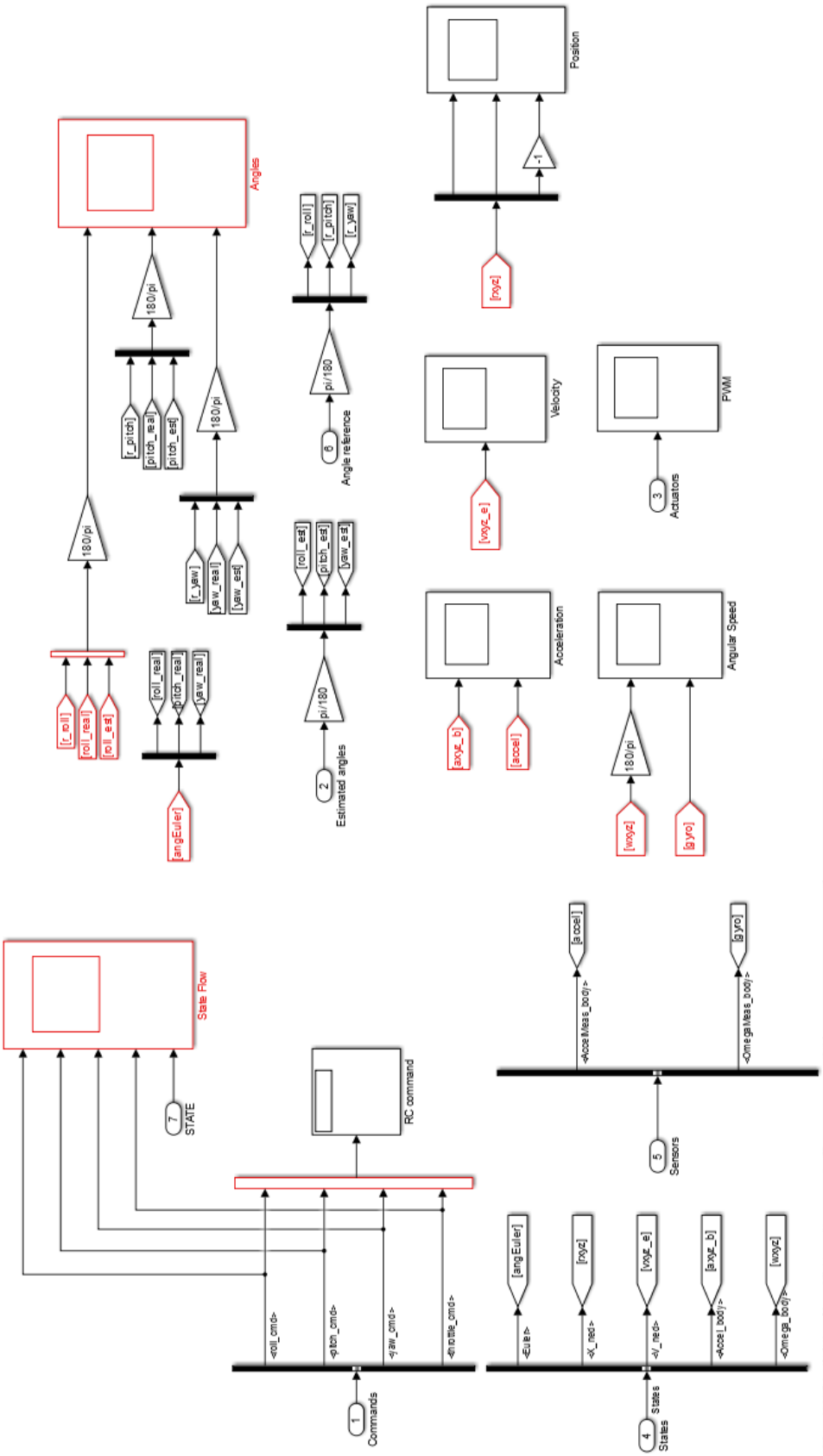


Figura 8.7. Estructura del filtro de la IMU para Gyro y Accel.

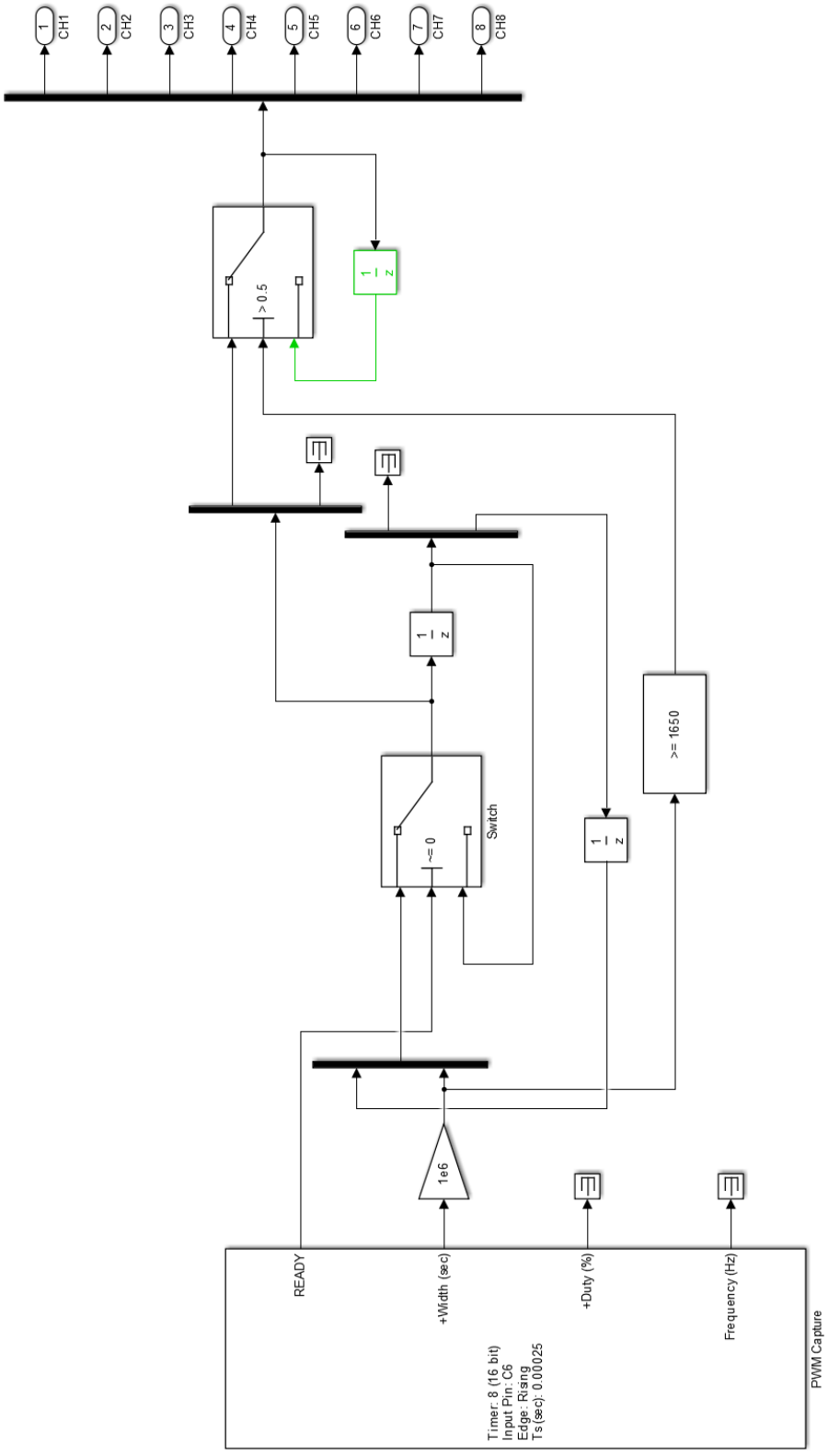
Anexo L

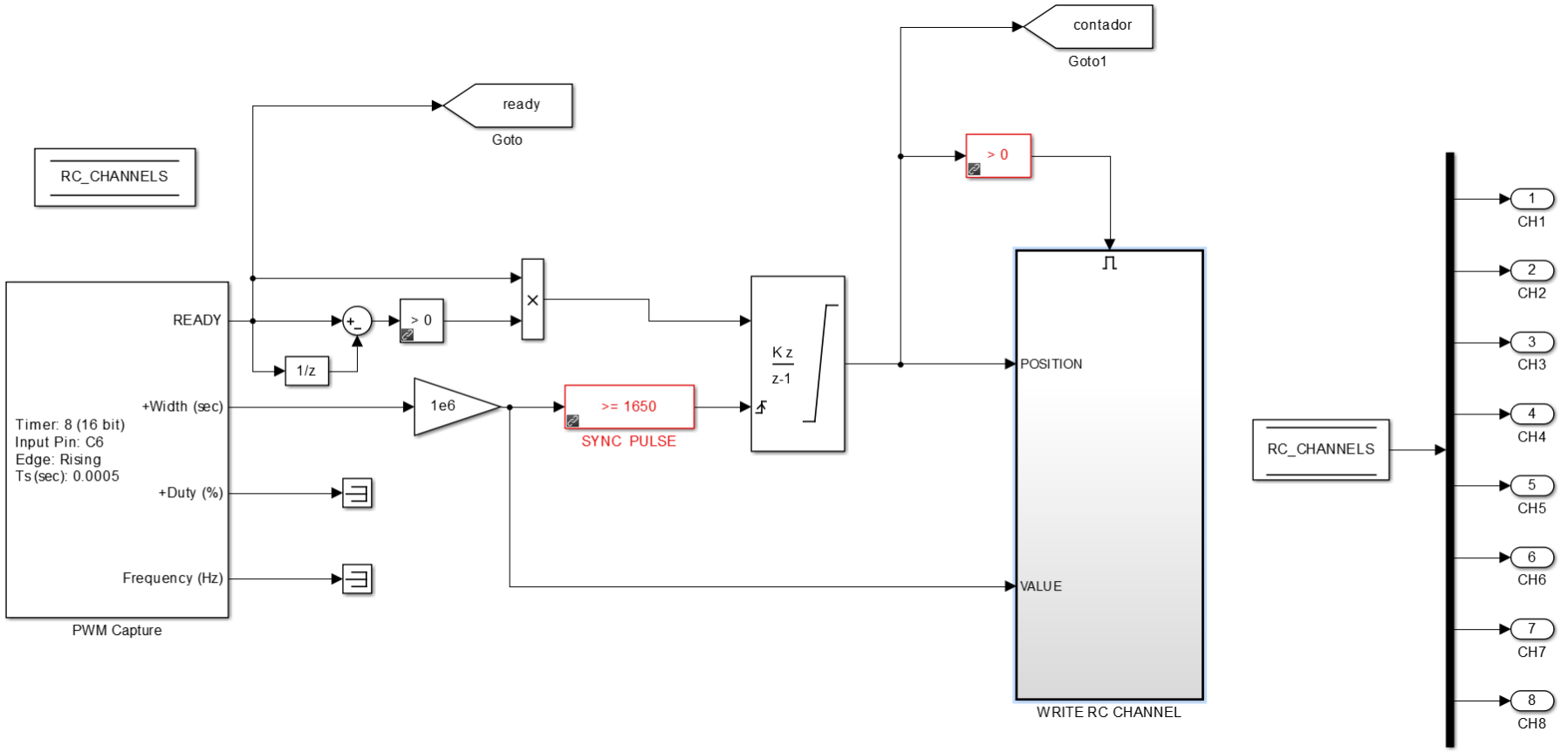
En este anexo se presenta el entorno de Simulación, en concreto el bloque de monitorización.



Anexo M

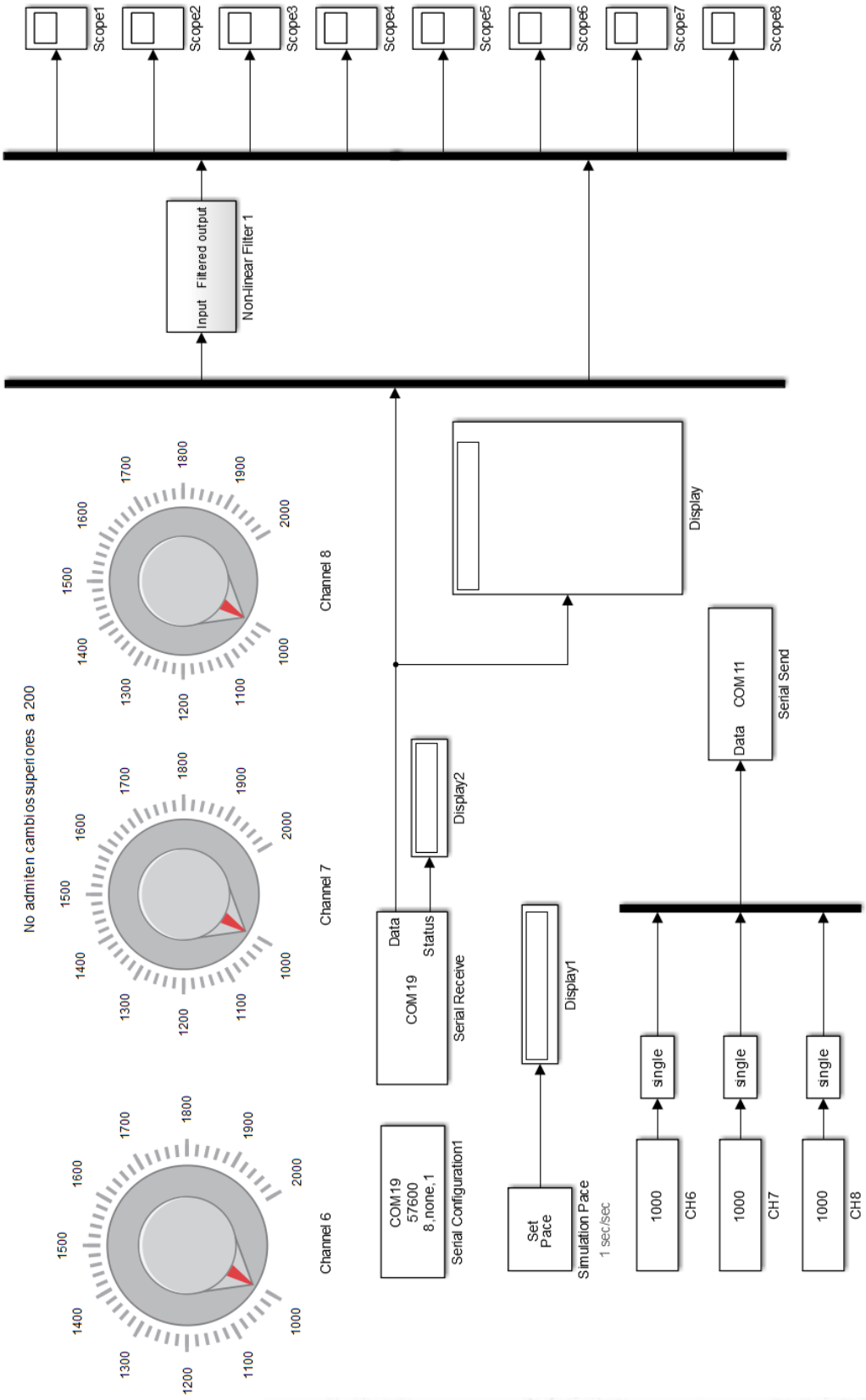
En este anexo se detallan la estructura interna del **Bloque Demodulador** creado específicamente para la recepción de los 8 canales de la emisora por transmisión PPM en un único puerto. A continuación se presentan dos versiones distintas funcionales, la ultima la más reciente.




RC_CHANNELS
ready
Goto
contador
Goto1
> 0
READY
+Width (sec)
Timer: 8 (16 bit)
Input Pin: C6
Edge: Rising
Ts (sec): 0.0005
+Duty (%)
Frequency (Hz)
PWM Capture
1/z
> 0
1e6
>= 1650
SYNC PULSE
K z
z-1
POSITION
VALUE
WRITE RC CHANNEL
RC_CHANNELS
1
CH1
2
CH2
3
CH3
4
CH4
5
CH5
6
CH6
7
CH7
8
CH8

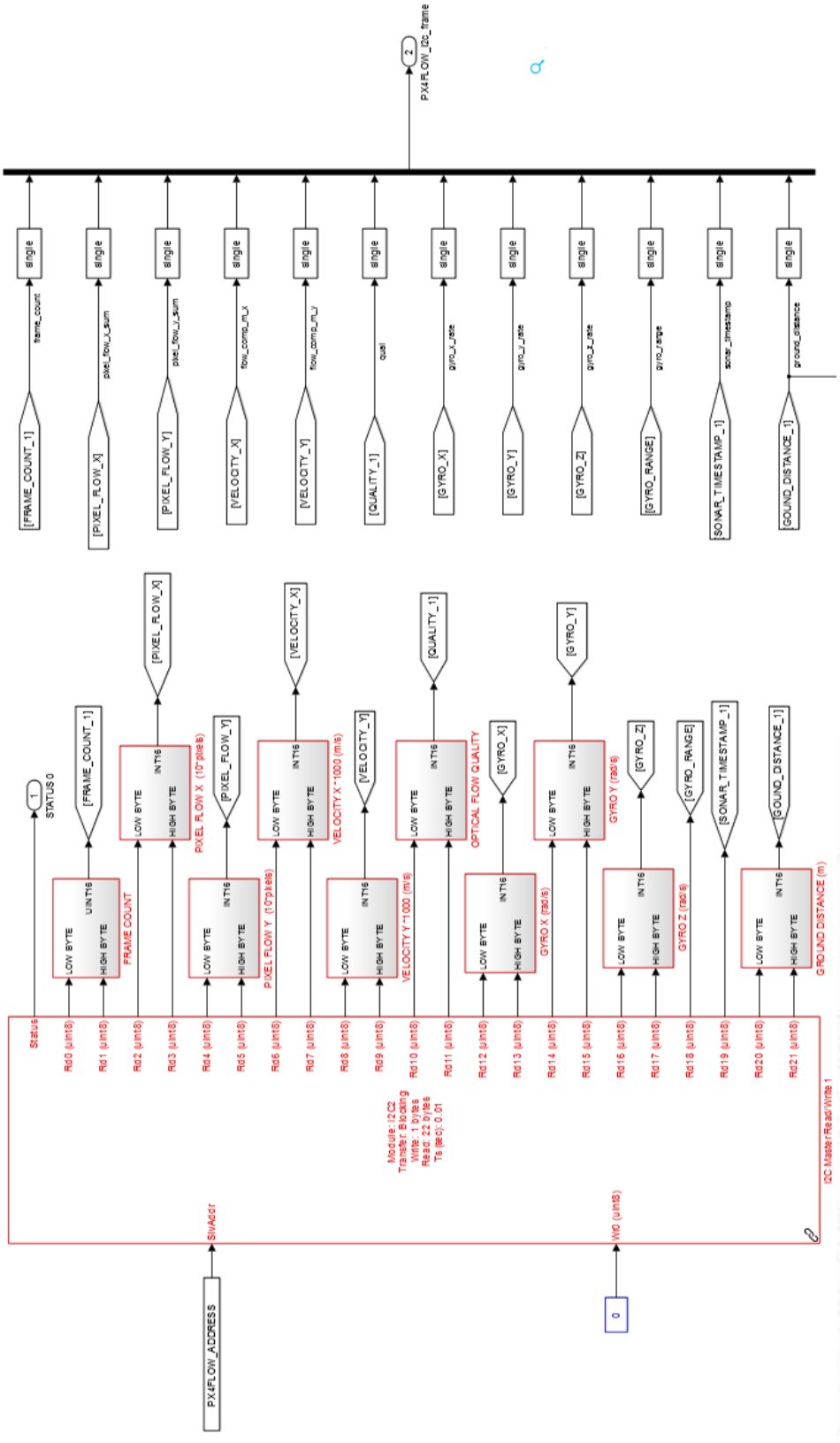
Anexo N

En este anexo se muestra el diagrama de Simulink del archivo **SERIAL_OPREVO_PC** que permite la lectura de parámetros en el PC así como el ajuste de datos en vuelo.



Anexo O

En este anexo se muestra el diagrama de Simulink del archivo de **lectura del sensor PX4FLOW**.



UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID

INSTITUTO DE INVESTIGACIÓN TECNOLÓGICA

PROYECTO FIN DE GRADO

PRESUPUESTO

"Control de un cuadricóptero para navegación en interiores usando un sensor de flujo óptico"

Autor:

Néstor González García

Directores:

Juan Luis Zamora Macho
José Porras Galán

Firma del Autor:

Firma de los directores:

Madrid
Junio de 2016

UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID

INSTITUTO DE INVESTIGACIÓN
TECNOLÓGICA

ÍNDICE

UNIVERSIDAD PONTIFICIA
ICAI ICADE
COMILLAS
MADRID

INSTITUTO DE INVESTIGACIÓN
TECNOLÓGICA

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

Componente	Cantidad
Estructura ZMR250 (H250) Carbon Fiber Mini FPV	1
Tarjeta OpenPilot Revolution	1
Motor EMAX MT2204 II 2300KV/CW	4
ESC DYS BLHeli Mini 20A	4
Sensor PX4FLOW	1
Módulo Bluetooth HC-05	2
Receptor FrSKY D8R-II PLUS	1
Emisora TURNIGY 9XR	1
Hélices Tripala 7x3.5	8
Resistencia 470mΩ	2
Regulador Turnigy 5A SBEC	1
Placa de Potencia Mini UBEC PDB 4S	1
Batería LiPo (1A, 11.1 V, 3S)	2

1.2 Herramientas y Software

Componente	Cantidad	Horas de Proyecto	Horas de uso al año
Ordenador	1	450	1000
Cargador de Baterías iMAX B6AC	1	50	100
Polímetro Digital PROMAX FP-2b	1	5	50
Herramientas: Destornillador, Alicates…	1	10	50
Cables Mini-MicroUSB	2	5	20
Matlab/Simulink	1	350	500
Microsoft Office	1	50	200

1.3 Mano de Obra

Actividad	Horas
Montaje del Cuadricóptero	10
Desarrollo del Entorno Conjunto de Simulación	60
Desarrollo del control	120
Implantacion del Sensor PX4FLOW	40
Ensayos y Depuración	200
Redacción de la documentación	80

Capítulo 2

Costes Unitarios

En esta sección se detallan los costes o precios de cada uno de los elementos previamente analizados, por unidad.

2.1 Componentes Principales

Componente	Precio (€/Ud)
Estructura ZMR250 (H250) Carbon Fiber Mini FPV	32,00
Tarjeta OpenPilot Revolution	60,00
Motor EMAX MT2204 II 2300KV/CW	15,90
ESC DYS BLHeli Mini 20A	9,47
Sensor PX4FLOW	94,99
Módulo Bluetooth HC-05	4,90
Receptor FrSKY D8R-II PLUS	45,51
Emisora TURNIGY 9XR	54,59
Hélices Tripala 7x3.5	1,90
Resistencia 470mΩ	1,00
Regulador Turnigy 5A SBEC	19,50
Placa de Potencia Mini UBEC PDB 4S	5,59
Batería LiPo (1A, 11.1 V, 3S)	15,00

2.2 Herramientas y Software

Componente	Precio (€/Ud)
Ordenador	800,00
Cargador de Baterías iMAX B6AC	40,00
Polímetro Digital PROMAX FP-2b	78,00
Herramientas: Destornillador, Alicates…	50,00
Cables Mini-MicroUSB	3,92
Matlab/Simulink	200,00
Microsoft Office	90,00

2.3 Mano de Obra

Actividad	Precio (€/hora)
Montaje del Cuadricóptero	20
Desarrollo del Entorno Conjunto de Simulación	40
Desarrollo del control	40
Implantación del Sensor PX4FLOW	20
Ensayos y Depuración	50
Redacción de la documentación	45

Capítulo 3

Sumas Parciales

A continuación se calcula el coste total de cada uno de los tipos de recursos empleados.

3.1 Componentes Principales

Componente	Cantidad	Coste (€/Ud)	Coste Total (€)
Estructura ZMR250 (H250) Carbon Fiber Mini FPV	1	32,00	32,00
Tarjeta OpenPilot Revolution	1	60,00	60,00
Motor EMAX MT2204 II 2300KV/CW	4	15,90	63,60
ESC DYS BLHeli Mini 20A	4	9,47	37,88
Sensor PX4FLOW	1	94,99	94,99
Módulo Bluetooth HC-05	2	4,90	9,80
Receptor FrSKY D8R-II PLUS	1	45,51	45,51
Emisora TURNIGY 9XR	1	54,59	54,59
Hélices Tripala 7x3.5	8	1,90	15,20
Resistencia 470mΩ	2	1,00	2,00
Regulador Turnigy 5A SBEC	1	19,50	19,50
Placa de Potencia Mini UBEC PDB 4S	1	5,59	5,59
Batería LiPo (1A, 11.1 V, 3S)	2	15,00	30
TOTAL			**470,66**

3.2 Herramientas y Software

Para realizar el cálculo del coste de los bienes de equipo, herramientas y software dedicados al proyecto se tiene en cuenta la proporción de horas destinadas al proyecto frente a las horas totales de utilización de ese recurso. Para ello, se consideran las horas de uso anuales y la amortización anual del precio. La Ecuación (1) recoge la fórmula que rige el cálculo:

$$C_{T=}C_U \cdot n \cdot \frac{t_p}{t_a} \cdot a \qquad (1)$$

Donde C_T es el coste total, C_U el coste unitario, n la cantidad, t_p el tiempo dedicado al proyecto, t_a el tiempo anual de uso, y a la amortización, en tanto por uno.

Componente	Cantidad	Horas de Proyecto	Horas de uso al año	Amortización Anual	Coste (€/Ud)	Coste Total (€)
Ordenador	1	450	1000	25%	800,00	90
Cargador de Baterías iMAX B6AC	1	50	100	25%	40,00	5
Polímetro Digital PROMAX FP-2b	1	5	50	25%	78,00	1,95
Herramientas: Destornillador, Alicates…	1	10	50	25%	50,00	2,5
Cables Mini-MicroUSB	2	5	20	25%	3,92	0,49
Matlab/Simulink	1	350	500	25%	200,00	35
Microsoft Office	1	50	200	25%	90,00	5,625
TOTAL						**140,57**

3.3 Mano de Obra

Actividad	Horas	Coste (€/hora)	Coste Total (€)
Montaje del Cuadricóptero	10	20	400
Desarrollo del Entorno Conjunto de Simulación	60	40	1600
Desarrollo del control	120	40	2000
Implantacion del Sensor PX4FLOW	40	20	400
Ensayos y Depuración	200	50	3000
Redacción de la documentación	80	45	2250
TOTAL			**9650,00**

Capítulo 4

Presupuesto Total

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

Actividad	Horas	Porcentaje
Componentes	470,66	5%
Herramientas y Software	140,57	1%
Mano de Obra	9650,00	94%
TOTAL	**10261,23 €**	**100%**