Proyecto Fin de Carrera
Grado en Ingeniería de las Tecnologías Industriales

Revisión y Modificación del Firmware de Libre Acceso ArduCopter para su uso en el Proyecto AirWhale

Autor: Alejandro Romero Galán
Tutor: Daniel Limón Marruedo

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2015

Proyecto Fin de Carrera
Ingeniería de Telecomunicación

Revisión y Modificación del Firmware de Libre Acceso ArduCopter para su uso en el Proyecto AirWhale

Autor:
Alejandro Romero Galán

Tutor:
Daniel Limón Marruedo
Profesor titular

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2015

Proyecto Fin de Carrera: Revisión y Modificación del Firmware de Libre Acceso ArduCopter para su uso en el Proyecto AirWhale

Autor: Alejandro Romero Galán

Tutor: Daniel Limón Marruedo

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2015

El Secretario del Tribunal

A todo aquel que creyó en mí

Agradecimientos

A riesgo de hastiar al lector del presente trabajo, no debo olvidar a ninguno de aquellos que hicieron posible que este proyecto saliese adelante.

En primer lugar, dar las gracias a mi padre y a mi madre, que siempre han estado dispuestos a asentir con la cabeza aún cuando lo que merecía era un 'no' por respuesta. A ellos les dedico y les debo todo lo que he hecho de provecho en mi vida, y lo que haga en la siguiente.

Agradezco profundamente a mis hermanos por aguantar preguntas sin sentido (el mayor), una ardua convivencia (el pequeño) y un carácter difícil que no merece perdón ni atención (ambos).

Quiero mencionar individualmente a cada uno de los compañeros que, junto a mi persona, componen la asociación universitaria EsiTech y han trabajado activamente en el proyecto AirWhale: José Luis Holgado Álvarez, Elena Manga Caballero, Juan Carlos Mancebo Sánchez, Javier Eduardo Mitjavila Samayoa, Inmaculada Gómez Vázquez, Miguel Ángel Rodrigo Lisbona, Juan Carlos Martín Rodríguez y Ana Caballos Torroba. Ha sido un honor trabajar y reír a vuestro lado; no podría haber completado este escrito sin vuestra ayuda.

A mi tutor, Dani, le debo todo lo que he hecho en casi un año de ideas, intenciones, caminos sin salida y hechos; y la pasión por lo que hacía, que he ido adquiriendo conforme trabajaba en este proyecto. La dedicación que ha empleado en temas, a priori, desconocidos para él, ha sido un gran referente en el camino del autoaprendizaje.

De forma general, porque si tuviese que mencionar a cada uno de ellos este trabajo sería interminable, gracias a todos aquellos compañeros de universidad que en algún momento me hablaron, me consolaron, me animaron, me hicieron reir, me ilusionaron o, simplemente, me consideraron una persona digna de entablar amistad con ellos. Nunca os olvidaré.

Debo nombrar a mis compañeros de piso en Sevilla, a Carlos, Javi y Manolete, por suponer aquella válvula de escape en los momentos en los que necesitaba desconectar de tantos números, códigos y dificultades.

Gracias, Ester, por aguantarme. Por ser el motor que ha movido todo esto.

Por último, no pienso olvidarme de agradecer a todos aquellos españoles que, pasando verdaderos problemas para llegar a fin de mes y poder darles a sus hijos una vida digna y plena de derechos, han seguido colaborando económicamente con mi formación académica (y mi salud), incluso cuando esa élite corrupta e indigna, que se hacen llamar políticos, no ha hecho más que colocarles más peso en su malograda espalda. Espero, algún día, devolverles cada uno de los euros que han invertido en mí con mis futuros trabajos.

Resumen

Palabras clave: UAV, multirotor, dirigible, AirWhale, modelo dinámico, autopiloto, ArduPilot, firmware, prototipo

El **proyecto AirWhale** consiste en el desarrollo de una aeronave híbrido entre dirigible y multirotor. El objetivo principal de dicho proyecto reside en la urgente necesidad de aumentar la autonomía de los quadrotors sin abandonar la agilidad y maniobrabilidad que los caracteriza. La intención, claramente innovadora, ha llevado a un grupo de alumnos reunirse y trabajar juntos por la construcción completa de dicha aeronave, formalizando un equipo de desarrollo amparado por la asociación universitaria **EsiTech** y la **Universidad de Sevilla**. Todos los trabajos personales de los alumnos que componen el equipo forman un Trabajo Fin de Grado extenso y detallado, en la línea del trabajo general del proyecto AirWhale.

Para implementar las distintas leyes de control que gobiernan el movimiento del AirWhale, son imprescindibles el diseño de un modelo dinámico asoaciado a la aeronave, para la experimentación en un primer nivel de precisión y para la sintonización de los distintos controladores, y la selección de una plataforma que se encargue de la gestión eficaz de sensores que recopilen información sobre el entorno que rodea a la aeronave y actuadores que lleven a cabo una serie de acciones con la intención de cumplir ciertos propósitos de forma autónoma, ya sean relacionados con el movimiento de la nave o con la obtención de datos en forma de imagen u otro formato. En primer lugar, se expone el procedimiento por el cual se obtiene **el modelo dinámico** realizando simplificaciones coherentes con uno de los objetivos de este trabajo: la construcción de un prototipo del AirWhale. En segundo lugar, se ha visto necesario el uso de la plataforma **ArduPilot**, destacada por su capacidad para la gestión simultánea de distintos periféricos y sensores, la popularidad con la que cuenta en el mundo de la aeronavegación y la disponibilidad de distintos y variados firmwares según la configuración de la nave a controlar.

Sin embargo, debido al evidente carácter innovador que posee la aeronave AirWhale, no existe un firmware específico que se ajuste a la misma, a nivel de modelo dinámico. En este sentido, hay que mencionar la dificultad añadida de la inexistencia de documentación oficial y detallada sobre el funcionamiento de los distintos firmwares, ya que estos han nacido de la mente colectiva de una comunidad de miles de desarrolladores, y, aunque son trabajos realmente eficientes, hace falta una revisión del código para conocer qué se debe modificar y adoptar para el propósito de este escrito: obtener un firmware adecuado para gobernar la aeronave AirWhale.

Finalmente, para comprobar la validez de dicho firmware modificado, se llevarán a cabo dos experimentaciones. La primera consiste en el uso de un sistema '**hardware-in-the-loop**' formado por Ardupilot y el programa de simulación **Matlab**; la información detallada sobre la configuración, comunicación entre hardware y software, e interfaz programada en Matlab del sistema HIL se podrá consultar íntegramente en el trabajo correspondiente a la alumna **Elena Manga Caballero**. Los resultados no se mostrarán en este escrito.

La segunda experimentación, más ambiciosa, conlleva la construcción de un prototipo con una dinámica aproximada al AirWhale que muestran los trabajos de **Juan Carlos Mancebo Sánchez, Javier Eduardo Mitjavila Samayoa e Inmaculada Gómez Vázquez**. El diseño en **CATIA**, disponible en el trabajo de **José Luis Holgado Álvarez**, y el proceso de construcción, gestionado y llevado a cabo personalmente de forma conjunta con el mencionado alumno, se han concebido siempre intentando distanciarse lo menos posible del AirWhale original y minimizando la complejidad para hacer posible la obtención del prototipo final.

Abstract

The AirWhale project consists of the development of a hybrid aircraft between airship and multirotor. The main aim of the project lies in the urgent necessity of increasing the autonomy of quadrotors without abandoning the agility and manoeuvrability that characterizes them. The clearly innovative intention has been carried out by a group of students that have met and worked together in the complete building of the mentioned aircraft, creating a development team inside the college association EsiTech and with the collaboration of the University of Seville. All the individual works of the students that are part of the team constitute a wide and detailed End of Grade work. Each of the individual EOG works inside the AirWhale project will be mentioned throughout the document as a reference for this one.

The selection of a platform in charged of efficiently managing the sensors which gather the information about the environment surrounding the aircraft, and also controlling the actuators that carry out several actions with the intention of working autonomously in the movement of the craft or in the obtaining of data has been necessary. This has been required for implementing the different laws of control that govern the movement of the AirWhale. Thus, it has been necessary to use the ArduPilot platform, outstanding for its capacity to simultaneously managing different peripherals and sensors, the popularity in the world of air navigation and the availability of several firmwares that depend on the settings of the craft to be controlled.

However, due to the evidently innovative nature of the AirWhale aircraft, there is not a specific firmware for it, at the level of a dynamic model. In this sense, it is necessary to mention the added difficulty of the lack of official and detailed information about the functioning of the different firmwares. As the only information comes from the collective cooperation and work of a community of thousands of developers, it is necessary to review the code to know what is required to be modified to fulfil the aim of this project and obtain an appropriated firmware to control the AirWhale aircraft.

Finally, to check the validity of the modified firmware, two experiments will be carried out. The first one consists of the use of a “hardware-in-the-loop” system, constituted by Ardupilot, the software of the platform Mission Planner and the simulation program Matlab. The detailed information about settings, communication between hardware and software and interface of the HIL system programmed in Matlab could be checked completely in the EOG work of Elena Manga Caballero. Results won’t be shown.

The second experiment is more ambitious, as it involves the building of a prototype with a similar dynamic to the AirWhale. This is shown by the EOG works of Juan Carlos Mancebo Sánchez, Javier Eduardo Mitjavila Samayoa e Inmaculada Gómez Vázquez. The design in CATIA, which is available in the work of José Luis Holgado Álvarez, and the process of the building of the aircraft have been conceived always trying to be as similar as possible to the original AirWhale and minimizing the complexity to make it possible to obtain the final prototype.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

A^*	Conjugado
c.t.p.	En casi todos los puntos
c.q.d.	Como queríamos demostrar
■	Como queríamos demostrar
e.o.c.	En cualquier otro caso
E	número e
IRe	Parte real
IIm	Parte imaginaria
sen	Función seno
Tg	Función tangente
arctg	Función arco tangente
sen	Función seno
$\sin^x y$	Función seno de *x* elevado a *y*
$\cos^x y$	Función coseno de *x* elevado a *y*
Sa	Función sampling
Sgn	Función signo
Rect	Función rectángulo
Sinc	Función sinc

∂y ∂x	Derivada parcial de y respecto
$x_\circ$	Notación de grado, x grados.
Pr(A)	Probabilidad del suceso A
SNR	Signal-to-noise ratio
MSE	Minimum square error
:	Tal que
<	Menor o igual
>	Mayor o igual
\	Backslash
⇔	Si y sólo si

1 INTRODUCCIÓN

"Si he logrado ver más lejos, ha sido porque he subido a hombros de gigantes"

Isaac Newton

El concepto de mejorar una carencia de una tecnología considerablemente en auge no es nuevo. Es algo inherente al ingeniero o al científico: intentar ir más allá, partiendo desde donde lo dejaron sus semejantes. Aportar algo al conocimiento tecnológico actual desde un punto de vista práctico. En este sentido, nació el proyecto AirWhale.

1.1 El Proyecto AirWhale

Hace aproximadamente un año, al final del curso académico 2013/2014, un grupo de alumnos, entre los que me incluyo, de distintos grados de ingeniería, impartidos en la Escuela Superior Técnica de Ingenieros de la Universidad de Sevilla, se embarcó en la difícil tarea de idear una tecnología que supliese la falta de una gran autonomía que caracteriza al actualmente famoso multirotor. Bien es conocido este problema inherente a estas aeronaves, provocado, principalmente, por el peso considerable de los componentes electrónicos necesarios para su funcionamiento (motores, variadores, baterías, sensores, autopiloto…) y la estructura en sí, y por la necesaria actuación ininterrumpida de los rotores para poder mantener al multirotor en el aire, de manera que, aún reduciendo el peso, la duración de las baterías decrece de manera drástica.

Ante este problema, se adaptó la idea básica concebida por el alumno José Luis Holgado Álvarez de usar la sustentación del helio para eliminar parte del peso que los rotores debían vencer y reducir la actuación de los mismos de forma continuada. Así, surgió el AirWhale. Un híbrido entre multirotor y dirigible. La idea es sencilla: se mantiene la agilidad, la velocidad y la capacidad de respuesta que aportan los distintos rotores con los que contaba el multirotor, y se añade autonomía y estabilidad acoplando a la estructura un volumen de helio confinado para sustentar gran parte del peso de la aeronave.

Figura 1.1. Diseño conceptual actualizado de la aeronave AirWhale [1]

El diseño del AirWhale, así como los materiales a usar, electrónica empleada, leyes de control que lo gobiernan y verificación de resultados relacionados con estos aspectos, han sido los temas generales sobre los que ha girado el trabajo del equipo de desarrollo, compuesto por los siguientes alumnos, divididos en las tres grandes ramas de la ingeniería que engloban el proyecto:

- Equipo Mecánico-Aeronáutico:
  - Juan Carlos Mancebo Sánchez
  - Javier Eduardo Mitjavila Samayoa
  - Inmaculada Gómez Vázquez

- Equipo Electrónico:
  - Juan Carlos Martín Rodríguez
  - Ana Caballos Torroba
  - Miguel Ángel Rodrigo Lisbona

- Equipo Automático:
  - José Luis Holgado Álvarez
  - Elena Manga Caballero
  - Alejandro Romero Galán

Todos estos aspectos se han ido perfilando a lo largo de un año de trabajo en el que todos hemos aprendido a trabajar en equipo y a planificar de manera profesional y eficiente los objetivos propios de un equipo de desarrollo. Un ejercicio de ingeniería concurrente que, opinando personalmente, contiene una lección de cara al mundo laboral tan imprescindible como todo aquel conocimiento que hemos podido aprender a lo largo de estos años de formación.

1.1.1 Diseño conceptual

El concepto AirWhale ha evolucionado rápidamente a lo largo de su vida. En sus comienzos, cuando el equipo carecía de una opinión aeronáutica, se pensaba en la implementación de un volumen esférico de helio a la estructura clásica de un quadrotor. Es decir, se partía de la idea de un globo esférico relleno de helio y rodeado por un cinturón rígido conectado al mismo. A este cinturón se anexionarían los cuatro rotores mediante un par de barras que soportarían la mayor parte de las cargas mecánicas.

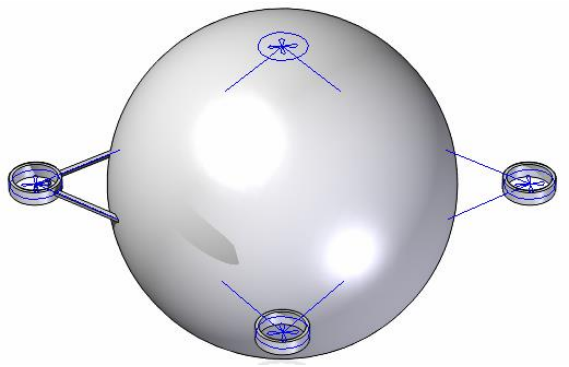

Figura 1.2. Diseño inicial del AirWhale [1]

El problema de este diseño residía en la ineficiencia aerodinámica del mismo. El globo y su forma esférica suponían unas fuerzas de resistencia aerodinámica elevadas y muy problemáticas en movimientos de avance y retroceso. Estas conclusiones se tomaron gracias a la inclusión en el equipo de la ingeniera aeronáutica Inmaculada Gómez Vázquez, y los consejos del profesor Sergio Esteban Roncero. A partir de aquí, el concepto AirWhale cambió radicalmente.

Debido a las nuevas ideas aportadas por la alumna de la rama aeronáutica, que se pueden consultar en su trabajo [2], y después de pasar por varios diseños intermedios, que se pueden repasar en el trabajo de mi compañero Juan Carlos Mancebo Sánchez [1], se llegó al diseño mostrado en la Figura 1.1.

En él se pueden observar los siguientes elementos:

- Dos rotores de eje vertical, situados en el extremo de cada ala, que se encargarán del movimiento en el eje Z y del par en X o alabeo.
- Dos rotores de eje horizontal, situados en la parte trasera de la aeronave, que se encargarán del movimiento en el eje X y del par en Z o guiñada.
- Dos superficies de control situadas en los estabilizadores verticales (timones de dirección) que permiten controlar el par en Z o guiñada en caso de estar la aeronave en movimiento.
- Dos superficies de control situadas en los estabilizadores horizontales (timones de profundidad) que permiten controlar el par en X o alabeo y, de manera exclusiva, el par en Y o cabeceo, en caso de estar el AirWhale en movimiento.
- Dos alas que ayudan en la sustentación y la aerodinámica, y que soportan los rotores de eje vertical.
- Un volumen de helio distribuido en tres lóbulos confinado por una lona de material adecuado.

Este diseño definitivo concreta la electrónica a usar para el funcionamiento óptimo del AirWhale.

1.1.2 Especificaciones y objetivos del Proyecto AirWhale

El principal motivo por el que se decide adoptar el diseño mostrado en la Figura 1.1. lo marcan las especificaciones y los distintos objetivos que se desea que cumpla el AirWhale. Como ya se ha comentado, en un principio, el Proyecto AirWhale nació con el objetivo de mejorar las deficiencias de dos aeronaves ya existentes: el dirigible y el multirotor. Al unir las tecnologías de una y otra, resta marcar unas metas a la nueva aeronave resultante, las cuales surgieron de la mente colectiva del equipo de desarrollo. Se enumeran a continuación:

- La utilidad principal para el AirWhale fue seleccionada entre varias ideas surgidas en las reuniones: fotografiar la fauna de distintos medios naturales o la cobertura de eventos deportivos y de ocio, así como la vigilancia de territorios comprometidos (fronteras, tráfico o eventos donde sea imprescindible la seguridad); búsqueda y rescate de personas desaparecidas o en situación delicada o inaccesible por los equipos de rescate, mediciones relacionadas con el clima o desastres naturales, vigilancia de bosques

para dar la alarma de forma casi simultánea al inicio del incendio, inspección de estructuras allá donde sea difícil para un ser humano o, incluso, actuar como un repetidor de señal en zonas críticas. Aunque la mayoría son realizables, teniendo en cuenta la máxima de reducir el peso lo máximo posible, se optó por la cobertura de eventos deportivos o de ocio, ya que conlleva la implementación de una cámara, de poco peso; no requiere la conexión de otros sensores que puedan aumentar la complejidad de procesado y el peso, y da opción al movimiento de la aeronave en ausencia de viento.

- El objetivo real del proyecto, de cara a una posible construcción y vuelo de un prototipo, es un vuelo en interiores, de tal manera que pueda despegar, volar un tiempo determinado, mostrando que puede realizar los movimientos principales que los rotores instalados permiten, y aterrizar.
- Las especificaciones que marcan el diseño final del AirWhale incumben a la velocidad de crucero, altura máxima que puede alcanzar, autonomía, dimensiones y peso, las cuales se detallan a continuación:
    - Velocidad de crucero: teniendo en cuenta el propósito de la aeronave de cubrir eventos deportivos y de ocio de manera nítida, se limitó a unos 5 m/s.
    - Altura máxima que puede alcanzar: se decidió de manera conjunta que fuese unos 20 m.
    - Autonomía: debido a la acción del helio y de las superficies sustentadoras, y el reducido peso que se supone deben vencer los rotores verticales, se impuso como mínimo una autonomía de una hora.
    - Dimensiones: se decidió de manera conjunta que la aeronave midiese 1 m de alto y 4 m de largo, como máximo.
    - Peso: se decidió de manera conjunta que no superase los 500 gramos.

Se podrá consultar en los trabajos del equipo mecánico-aeronáutico [1] [2] [3] los valores finales que toman algunos de estos aspectos de la aeronave AirWhale.

1.2 La asociación universitaria EsiTech

De forma paralela al desarrollo y evolución del Proyecto AirWhale, al equipo de desarrollo ya presentado se le aconsejó, de parte del profesor Daniel Limón Marruedo, formalizar a ojos de la Universidad de Sevilla dicho proyecto y dicho equipo de alumnos. Las razones eran, principalmente, dos:

- Los materiales necesarios para la construcción de un prototipo conllevan costes elevados, por lo que era conveniente la colaboración económica de la Universidad de Sevilla.
- El trabajo del equipo de desarrollo necesitaba de un espacio propio para poder llevar a cabo el desarrollo del Proyecto AirWhale de manera adecuada, recurso que se le ofrecen a las asociaciones universitarias.

Así nació la asociación universitaría EsiTech, formada por los integrantes ya mencionados del equipo de desarrollo y organizada según la Figura 1.3.

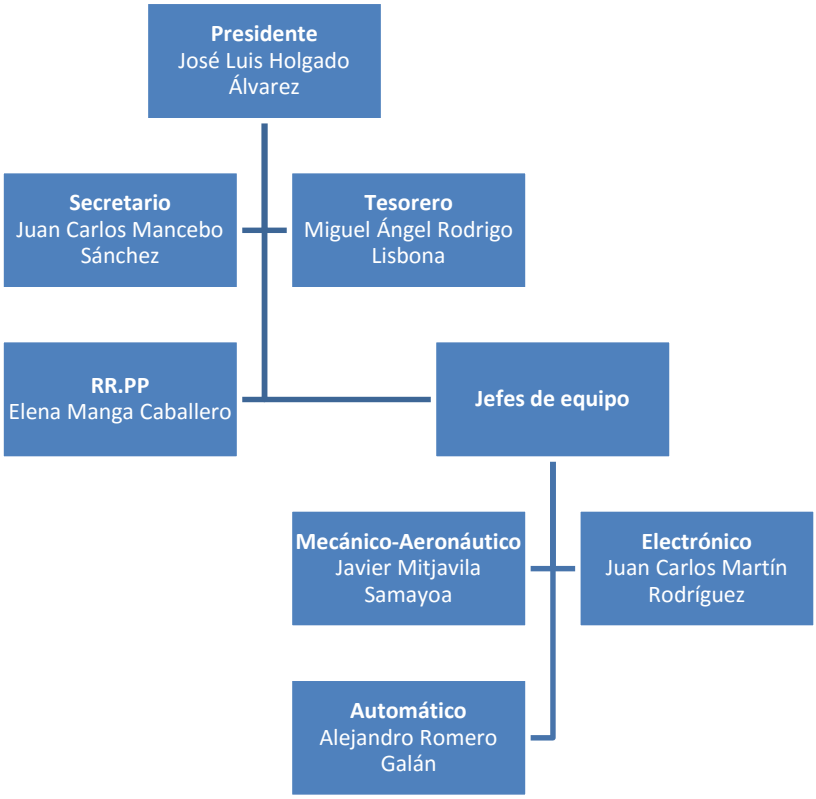

Figura 1.3. Esquema de la organización de la asociación EsiTech (a extinguir)

Con un fondo económico disponible y la habilitación de un espacio de trabajo adecuado, la asociación, además de continuar desarrollando el Proyecto AirWhale con el ánimo de lograr hacer volar el prototipo final, está ideando nuevos proyectos a desarrollar el próximo curso por alumnos ya pertenecientes a la misma y/u otros que quieran sumarse para aumentar sus conocimientos en las distintas ramas de la ingeniería, y, así, poder escoger un Trabajo de Fin de Grado que, además de ser apasionante y adecuado a sus gustos, le dé la oportunidad de aprender a trabajar en equipo con otros compañeros especializados en distintas ramas.

El logo de la asociación se muestra en la Figura 1.4.

Figura 1.4. Logo de la asociación universitaria EsiTech

1.2.1 Participación en el concurso Fly Your Ideas

Los componentes de la asociación decidieron participar en el concurso *Fly Your Ideas* 2015 que organiza la empresa aeronáutica Airbus cada año. Debido a que el concurso se basa en aeronaves de pasajeros, se tuvo que presentar al concurso una versión modificada de la idea del AirWhale con la que se estaba trabajando, aumentando su tamaño considerablemente y añadiéndole cabinas para pasajeros y pilotos. También hubo que idear varios conceptos relacionados con el despegue y aterrizaje de la aeronave y la inclusión de paneles fotovoltaicos flexibles adheridos a la lona del AirWhale para darle mayor autonomía de vuelo, entre otras ideas.

Al concurso se presentaron, debido a que los equipos no pueden superar los cinco integrantes, Inmaculada

Gómez, José Luis Holgado, Juan Carlos Mancebo, Juan Carlos Martín y Javier Mitjavila. Después de superar la primera fase, donde participaron alrededor de 500 equipos, el equipo AirWhale se quedó a las puertas de la fase final, estando entre los 100 primeros equipos.

Personalmente, aunque no pude formar parte del equipo que participó en el concurso, colaboré activamente en la redacción de documentos a presentar en el concurso, en el diseño y creación de ilustraciones y carteles relacionados con el AirWhale y aportación de ideas que aparecen en el vídeo de presentación del equipo EsiTech correspondiente a la segunda fase del concurso.

Figura 1.5. Cartel promocional del AirWhale

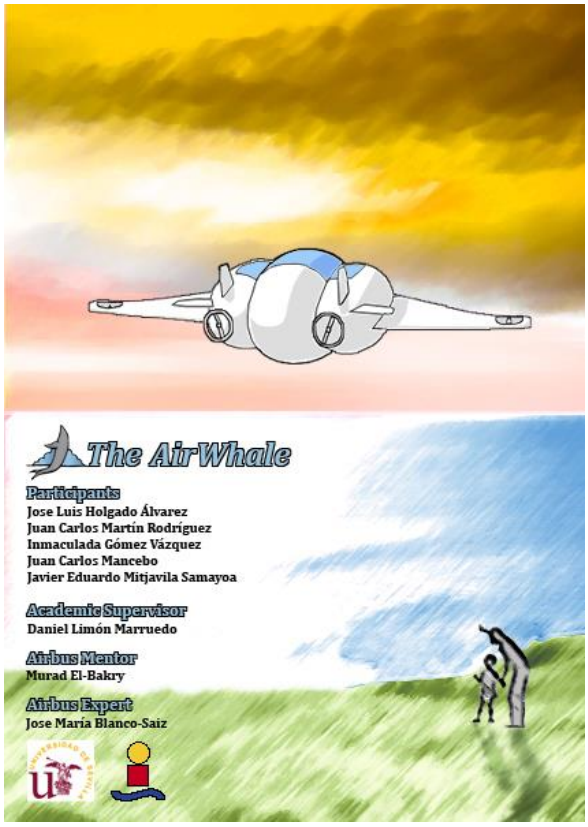

Figura 1.6. Cartel de presentación del AirWhale en el concurso *Fly Your Ideas*

2 EQUIPO AUTOMÁTICO: ORGANIZACIÓN E INTRODUCCIÓN A MI TRABAJO PERSONAL

"Llegar juntos es el principio. Mantenerse juntos es el progreso. Trabajar juntos es el éxito"

Henry Ford

El trabajo que concierne a este escrito, como se pudo leer en el resumen, consiste en una revisión detallada del firmware de libre acceso *ArduCopter*, perteneciente a un repositorio público de varios archivos de código y firmwares realizado por multitud de usuarios apasionados por la robótica, bajo el nombre de *DIYDrones*, y en su modificación para poder usarlo adecuadamente en la aeronave AirWhale. Este estudio resulta ser la parte central del trabajo conjunto del equipo Automático de la asociación universitaria EsiTech, compuesto por Elena Manga Caballero, José Luis Holgado Álvarez y Alejandro Romero Galán, el que escribe estas palabras. Cabe detallar la relación que guardan los trabajos individuales de cada uno de los alumnos entre ellos para conocer los límites del presente escrito y los datos pertenecientes a los demás proyectos que se necesitan para darlo por concluido.

2.1 Introducción al trabajo del equipo Automático

El cometido del equipo Automático dentro del Proyecto AirWhale comprende, en líneas generales, el cumplimiento de las siguientes misiones:

- Definición de un modelo dinámico simulable en Matlab/Simulink. Esta tarea se ha llevado a cabo entre mi compañero José Luis Holgado y yo, de forma conjunta, basándose en el trabajo del alumno Javier Mitjavila. Se detallará en el Capítulo 3.
- Definición de leyes de control adecuadas para el funcionamiento óptimo y deseado del AirWhale, y la experimentación de su implementación sobre el modelo dinámico diseñado para su validación. Este trabajo pertenece íntegramente al alumno José Luis Holgado Álvarez y se puede revisar en su Trabajo Fin de Grado [4].
- Selección de plataforma autopiloto, sensores, actuadores y electrónica necesaria para poder cumplir los objetivos propuestos. Este trabajo se ha llevado a cabo conjuntamente entre mi compañero Juan Carlos Martín Rodríguez, del equipo Electrónico, y yo. Se detallará en el Capítulo 4.
- Revisión del firmware ArduCopter y su adaptación al proyecto AirWhale, lo cual comprende la programación de las leyes de control, el estudio de las librerías ya diseñadas para el firmware ArduCopter, y otros detalles. Este trabajo, desarrollado íntegramente en este escrito, se detalla en el Capítulo 5.
- Verificación del firmware modificado mediante un sistema Hardware-in-the-loop desarrollado completamente por la alumna Elena Manga Caballero, aportando una interfaz en Matlab completa y necesaria [5]. La planificación y los resultados de los experimentos de validación no se han podido incluir en este trabajo debido a los distintos ritmos de trabajo que se ha llevado entre el alumno asociado a este escrito y la citada alumna, pero sí podrán verse en su trabajo.
- Construcción de una primera versión de un prototipo del AirWhale y validación del trabajo anterior

mediante el vuelo del mismo, observando los resultados y su desviación con los obtenidos de forma teórica. Esta tarea corresponde al alumno José Luis Holgado y a mí, y puede encontrarse redactado en el Capítulo 6.

Ante esta información, es clara la relación existente entre el trabajo de cada uno de los miembros del equipo Automático, la cual se muestra en el esquema de la Figura 2.1.

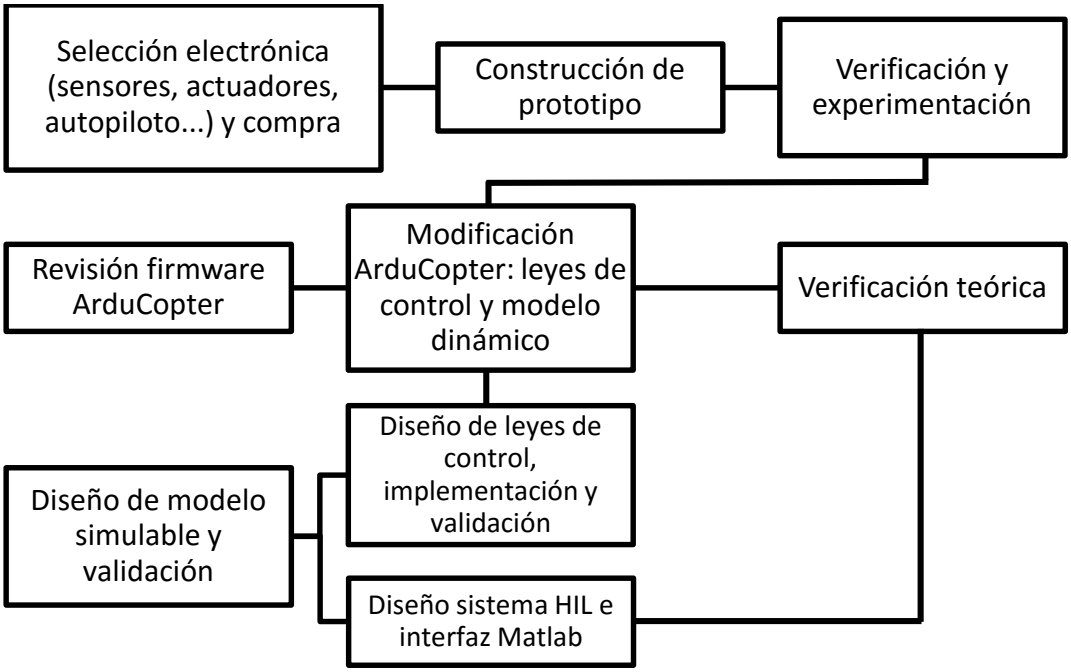

Figura 2.1. Esquema de trabajo del equipo Automático

Las relaciones clave se dan una vez se hayan diseñado las leyes de control y el modelo dinámico, ya que ello supone completar los recursos necesarios para la completa modificación del firmware ya existente y el diseño del sistema hardware-in-the-loop en su fase correspondiente al software Matlab/Simulink. Por otro lado, el firmware modificado es necesario para la verificación teórica obtenido del sistema HIL, y viceversa, y para poder volar el prototipo correctamente. Aún existiendo estas relaciones, cada uno de los integrantes del equipo ha podido trabajar individualmente dentro de unos límites amplios.

2.2 Introducción a la plataforma ArduPilot

2.2.1 Estado del arte

La necesidad de una plataforma autopiloto es clara. Independientemente de la aeronave que se esté tratando, el uso de un autopiloto es totalmente imprescindible para la implementación de las leyes de control y ecuaciones necesarias para el movimiento autónomo de la misma. Actualmente, existen varias plataformas a disposición del interesado para su uso en el control de aeronaves, cada una con características propias que las hace interesantes en uno u otro sentido. A continuación se recogen algunas de las más populares, por sus grandes prestaciones y relevancia por su pertenencia a proyectos más ambiciosos que incluyen desarrollos de firmwares y otros componentes electrónicos, paralelos al Proyecto ArduPilot.

2.2.1.1 PX4 PIXHAWK

La plataforma PX4 PIXHAWK puede resultar una de las alternativas más interesantes al ArduPilot. Pertenece al Proyecto PX4, que también goza de una comunidad de desarrolladores que han llevado a cabo un firmware independiente, de un programa que permite la visualización de datos y la calibración de los sensores como es QGroundControl (análogo a Mission Planner) y, además, permite la implementación de firmwares desarrollados por el Proyecto ArduPilot, aumentando considerablemente la versatilidad de su placa insignia [6]. Las características de la plataforma se muestran a continuación:

Figura 2.2. Plataforma PIXHAWK

- Procesador de 32 bits STM32F427 Cortex M4 Core con FPU (unidad de punto flotante), de 168 MHz, 256 kB de memoria RAM, 2 Mb de memoria FLASH. También dispone de un coprocesador de 32 bits STM32F103.
- Cuenta con giróscopo ST Micro L3GD20H de 16 bits, acelerómetro/magnetómetro ST Micro LSM303D de 14 bits, acelerómetros de 3 ejes/giróscopo Invensense MPU 6000, y un barómetro MEAS MS5611.
- Dispone de 5 puertos serie (uno de alta capacidad, dos de control de flujo de datos), dos puertos de protocolo CAN (Controller Area Network), entrada compatible con los DSM de Spektrum (radiocontrol), entrada y salida dedicada al Futaba S.BUS (conexión de múltiples servos), entrada de señal PPM, entrada para PWM o voltaje, I2C, SPI, entrada para GPS, convertidores analógico-digital de 3.3 V y 6.6 V y puerto interno microUSB.
- 14 salidas PWM, seis auxiliares y de alta potencia [6].

2.2.1.2 OpenPilot CopterControl y OpenPilot Revolution

Plataformas pertenecientes al Proyecto OpenPilot, nacido en 2010 de la mano de una comunidad de usuarios de todo el mundo. Dicho proyecto, aunque lleva pocos años en activo, es bastante completo, y supone una interesante opción a la hora de adquirir un sistema autopiloto. Además de desarrollar su propio firmware y software de adquisición de datos, el Proyecto OpenPilot sostiene como estandartes dos plataformas, por encima de otros componentes electrónicos propios, como sensores GPS y telemetría:

- OpenPilot CopterControl: Consiste en una plataforma autopiloto para la estabilización de aeronave multirotor, helicópteros y aviones no tripulados. Su versión actualizada, la plataforma OpenPilot CC3D, consta de las siguientes características:

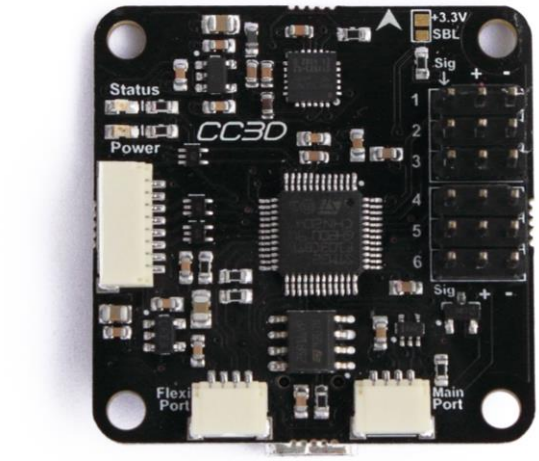

Figura 2.3. Plataforma OpenPilot CC3D

- Procesador ARM STM32F1 de 32 bits Cortex M3, con 512 Kb de FLASH y 64 Kb de RAM, de 72 MHz [7].
- Giróscopo/Acelerómetro MPU 6000. No posee barómetro.
- 6 canales PWM; incluye PPM, soporte para Spektrum DSM2, DSMJ, DSMX; soporte para Futaba S.BUS, entrada para telemetría y GPS (diseñado por el propio Proyecto OpenPilot), I2C (en desarrollo), 10 salidas PWM con compatibilidad con cámara y entrada USB para fácil configuración [8].

- OpenPilot Revolution: El Proyecto OpenPilot decide mejorar considerablemente las capacidades de la plataforma CopterControl mediante la creación de la OpenPilot Revolution, que incluye una mayor cantidad de conexiones útiles, un procesador de mayor rapidez y memoria y la implementación de sensores a día de hoy muy utilizados como son el barómetro o el sónar (añade la posibilidad de conectarlo). Las características generales son:

Figura 2.4. Plataforma OpenPilot Revolution

- Procesador ARM de 32 bits STM32F4 Cortex M4, de 168 MHz, con 192 Kb de memoria RAM y 1 Mb de FLASH, con FPU.
- Sensor acelerómetro/giróscopo de 3 ejes MPU 6000, barómetro MS5611-01BA03 con sensibilidad mínima de 10 cm y magnetómetro Honeywell HMC5883L con sensibilidad de 1-2º.
- 6 canales PWM de recepción junto con dos GPIO, un canal PPM de recepción junto con cinco puertos entrada/salida PWM para conexión de motores, servos y/o sensores; cuatro pines SPI, un pin UART, protocolo CAN, soporte para I2C, telemetría, GPS, Spektrum DSM, Futaba S.BUS y sónar; puerto SWD para facilidad de depuración para desarrolladores, seis salidas específicas PWM y microUSB [9].

2.2.1.3 Arsov AUAV3

Esta plataforma, menos conocida que las anteriores, está inspirada en el Proyecto PX4 y en su plataforma. El objetivo de este autopiloto es, sin embargo, suponer una alternativa seria y económica a su análogo aportado por el Proyecto PX4, igualando en gran medida sus prestaciones y reduciendo su precio. También es remarcable la compatibilidad con los firmwares desarrollados para el ArduPilot, aunque ya se está trabajando en un firmware exclusivo para la plataforma AUAV3. Algunas de sus especificaciones son:

Figura 2.5. Plataforma AUAV3

- Microprocesador dsPIC33EP512MU810 Microchip de 16 bits de 70 MHz, 512 Kb de RAM y 32 Mb de FLASH AT45DB321 incorporado [10] [11].
- Sensores IMU MPU6000, magnetómetro HMC5883 y barómetro BMP180.
- 3 pines UART, dos de ellos destinados a la telemetría y al OSD; 8 canales de entrada y salida, 4 entradas analógicas, 3 pines I/O digitales, soporte para GPS, I2C, SPI y protocolo CAN (driver MAX3051), entre otros [11].

2.2.1.4 Proyecto Paparazzi

El Proyecto Paparazzi es uno de los proyectos públicos relacionados con los vehículos aéreos no tripulados más completos que se pueden encontrar. Abarca todo el sistema autopiloto, desde la plataforma hasta el firmware, pasando por los distintos periféricos que se pueden usar en conjunto con ambos.

Desde su fundación en 2003, miles de desarrolladores se han sentido identificados con las pretensiones del proyecto, muchos de ellos vinculados a universidades de Francia, Holanda o Estados Unidos. También, a raíz

del proyecto, han surgido, de la mano de desarrolladores, empresas que comercializan productos relacionados con el proyecto o se especializan en el desarrollo de los mismos.

No se puede decir, sin embargo, que exista una plataforma específica y característica del proyecto, como se ha visto en los anteriores. El Proyecto Paparazzi ha llevado a cabo una labor de experimentación en multitud de tarjetas para validar el firmware propio desarrollado. La lista, relativamente extensa, se puede consultar en la página principal del proyecto [12].

2.2.2 Proyecto ArduPilot

La plataforma ArduPilot es resultado de años de trabajo en el denominado Proyecto ArduPilot. El nombre proviene de 'Arduino', ya que las primeras versiones de la plataforma (APM 1) se basaban en el entorno de desarrollo Arduino. Conforme se fue actualizando dicha plataforma, el firmware asociado a la misma evolucionaba paralelamente, y era compatible con más tipos de hardware, y no sólo con aquellos compatibles con Arduino, como el PIXHAWK. Por ello, hoy día la plataforma recibe el nombre de APM (ArduPilot Mega), al igual que los firmwares asociados (APMCopter, APMPlane…), siendo sus versiones más recientes APM 2.5 y APM 2.6 [13].

El proyecto abarca el desarrollo de plataformas autopiloto, la programación de firmwares para el control de distintos vehículos de uso popular como son el avión, el multirotor o los rovers; y la creación de un software que haga de conexión entre el usuario y el hardware, llamado Mission Planner. En las siguientes subsecciones se repasará de forma general cada uno de los componentes principales del Proyecto Ardupilot, posponiendo el estudio en profundidad al siguiente capítulo.

2.2.2.1 Plataforma ArduPilot

El cerebro del AirWhale. Esta placa será la que lleve a cabo las operaciones necesarias para que los rotores actúen adecuadamente y la aeronave pueda volar según las leyes de control diseñadas. La versión usada en el proyecto es el ArduPilot Mega 2.6 o APM 2.6, pero la plataforma ha experimentado muchos cambios para llegar a disponer de las características de las que goza a día de hoy. En la Figura 2.6 se repasa una evolución de la plataforma desde 2009 hasta 2012.

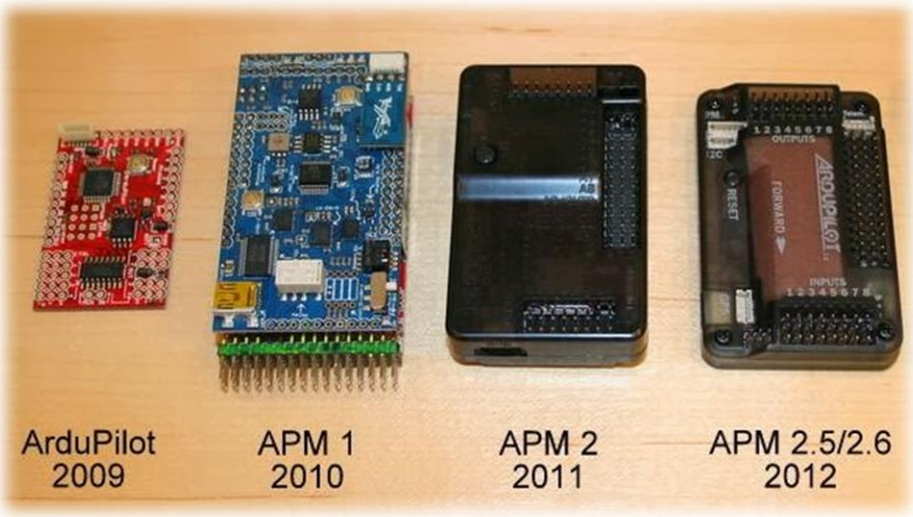

Figura 2.6. Evolución visual de la plataforma ArduPilot, desde su primera versión hasta la 2.6 [13].

Las especificaciones técnicas de las que dispone la versión APM 2.6 son las siguientes:

- Procesador Atmel ATMEGA2560 de 8 bits, con memoria FLASH de 256 Kb, 8 Kb de RAM y 16 MHz [14]. También cuenta con un coprocesador Atmel ATMEGA32U2 de 8 bits, con memoria FLASH de 32 bits, 1 Kb de RAM y 16 MHz [15] y un chip DataFlash de 4 Mb para el registro de datos automático.
- Cuenta con tres sensores a bordo: un acelerómetro/giróscopo MPU6000 y un barómetro MS5611-01BA03 de Measurements Specialties. No cuenta con un magnetómetro a bordo, como su predecesor APM 2.5, sino que está adaptado para que el conecte externamente, evitando así las interferencias magnéticas que pueda producir la propia placa [13].
- 8 salidas y entradas digitales, cada una de ellas con pines polarizados y el neutro. Pensando en la práctica, estos pines se emplean con dos cometidos:
    - Las salidas se usan para alimentar a los motores, el sensor de flujo óptico (no usado en este proyecto), para actuar como puerto RSSI y para manejar la cámara (servos). En el momento de detallar el firmware ArduCopter, se entrará en mayor profundidad en la función de estos pines.
    - Las entradas son, en su mayoría, utilizadas para los canales de radiocontrol. Normalmente unos pocos suelen quedar sin función, en ese caso se usan como puertos auxiliares para otros cometidos (se verá detallado en las siguientes secciones) [13].
- Entradas adaptadas para el GPS, la alimentación mediante un módulo de alimentación, I2C (normalmente usado para la conexión de un magnetómetro externo) y telemetría [13].
- 9 entradas analógicas (A0-A8) para utilidades generales (sónar, sensor de velocidad del aire…) y otras cinco, dos de ellas como puertos SPI (A12-A13) y las otras tres como entrada/salida (A9-A11) [13].

La figura 2.7 muestra un esquema general de las conexiones descritas.

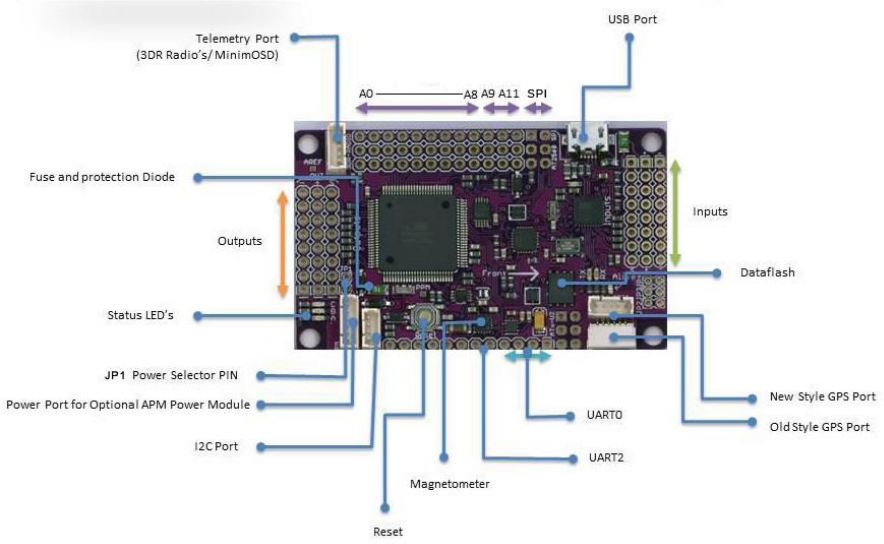

Figura 2.7. Esquema de las conexiones de la placa APM 2.6 [13]

En el Capítulo 4 se enumerarán y detallarán los dispositivos que se conectarán a la plataforma.

2.2.2.2 Software Mission Planner

La interfaz placa-usuario. Este programa recoge un sinfín de opciones e información disponible sobre los disintos aspectos del hardware y firmware del ArduPilot. El mismo se puede descargar gratuitamente desde el

enlace [16]. En la Figura 2.8 puede apreciarse la interfaz principal del programa Mission Planner.

Figura 2.8. Interfaz principal del software Mission Planner.

Se explicarán, en orden de numeración, los distintos módulos que aparecen en la Figura 2.8:

1. Menú principal: Aquí aparecen las principales secciones que componen el programa. Se describirán generalmente las más importantes:

    - Flight Data (seleccionada)
    - Flight Plan: Permite planificar el recorrido de la aeronave seleccionando puntos clave del camino a seguir o 'waypoints', y otras opciones relacionadas con las misiones.
    - Initial Setup: Proporciona indicaciones para configurar la plataforma ArduPilot, incluyendo la actualización del firmware o la correcta calibración de los sensores.
    - Config/Tuning: Permite configurar varias opciones relacionadas con el software y firmware (PIDs, parámetros que permiten habilitar o deshabilitar hardware…).
    - Simulation: Sección donde se puede observar una simulación basada en otros softwares de simulación (FlightGear, X-Plane, JSBSim…) si está activado el hardware-in-the-loop.

2. Visualización de piloto: Visión 'a bordo' del movimiento de la aeronave. Se detallan los giros en las tres direcciones, la altitud, la intensidad de señal en porcentaje, el tiempo de vuelo, la velocidad del aire (en caso de no disponer de sensor de velocidad del aire, marca la velocidad del suelo), el estado de la batería y del GPS, distancia al siguiente 'waypoint' y más información.
3. Registro de datos: Muestra distintos datos relacionados con el vuelo: medidas de sensores (acelerómetro, giróscopo, sensor de presión…), velocidades, posición o actuaciones. También pueden revisarse mensajes (recibidos por protocolo Mavlink), descargar registros de datos almacenados y emplearse otras interfaces visuales.
4. Mapa: Muestra la situación actual de la aeronave en un mapa proporcionado por Google Maps. Esto es posible gracias a los datos proporcionados por el GPS.
5. Conexión APM: Aquí se selecciona el puerto serie donde se encuentra la plataforma ArduPilot, la velocidad a la que se obtienen los datos por este puerto y se activa la conexión. Al activar la conexión, nuevas opciones pueden aparecer en el software.

Aunque la utilidad del software es relevante, para este proyecto no se usará demasiado, limitando su utilidad a la comprobación del buen funcionamiento de los sensores (aunque fue realizado por el alumno, no se detallará en este trabajo; se demostrará el funcionamiento óptimo de los sensores a utilizar a nivel de código, en el Capítulo 5).

2.2.2.3 Firmware ArduPilot

La inteligencia de la aeronave. Consiste en un código en lenguaje C/C++ que recoge las sentencias precisas para el funcionamiento correcto de la placa APM 2.6, para una comunicación adecuada con el software Mission Planner y el uso deseado de los dispositivos electrónicos conectados a la plataforma. Este firmware es muy extenso, abarcando cientos de archivos de código divididos en grandes grupos pertenecientes a cada vehículo soportado por la comunidad de desarrolladores. Estos grupos son:

- APMrover: Tiene como objetivo la creación de un firmware estable y eficaz para vehículos terrestres no tripulados.
- ArduPlane: Busca el desarrollo de un firmware adecuado para aeronaves con configuración parecida a la de un avión clásico.
- ArduCopter: El firmware que atañe a este escrito. Aporta un repositorio de archivos muy extenso para hacer volar un multirotor de forma óptima.

Además, existen, de forma común, varios ficheros relacionados con librerías, utilidades varias para los desarrolladores y módulos relacionados con otras plataformas (PIXHAWK).

Concretamente, el firmware ArduCopter consta de 69 archivos, entre los que se hallan archivos de código C++, librerías concretas para el ArduCopter y otros. Por otro lado, las librerías comunes, recogen conceptos relacionados con sensores, comunicación con Mission Planner, declaraciones a nivel de placa, controladores como el PID, funciones matemáticas o sentencias necesarias para la compatibilidad con otras plataformas autopiloto.

Más adelante se detallarán en mayor profundidad los archivos relativos al presente trabajo.

3 MODELO DINÁMICO DEL AIRWHALE

Conocer cómo debe comportarse la aeronave AirWhale es uno de los hitos a alcanzar por el equipo Automático. Por ello, uno de los trabajos que se expondrán aquí es la obtención de unas ecuaciones que definan completamente la dinámica de una primera versión del AirWhale (prototipo a construir). Este trabajo, como ya se ha mencionado anteriormente, ha sido desarrollado conjuntamente con el alumno José Luis Holgado Álvarez (se podrá encontrar el mismo trabajo en su escrito [4]) y se ha basado en el modelo dinámico desarrollado por nuestro compañero del equipo Mecánico, Javier Eduardo Mitjavila Samayoa. A continuación se expondrá dicho modelo a modo de introducción y, posteriormente, se desarrollarán las transformaciones necesarias para obtener el modelo susceptible a traducción en Simulink.

3.1 Modelo dinámico del AirWhale (formulación cartesiana y transformación a coordenadas de Euler)

El modelo dinámico del AirWhale en coordenadas cartesianas ha sido desarrollado íntegramente por el alumno Javier Mitjavila en su Trabajo Fin de Grado [3]. Este modelo tiene en cuenta todas las fuerzas que actúan sobre la aeronave, pero aquí se hará una simplificación para reducir la complejidad del modelo que se simulará posteriormente sin reducir significativamente la precisión, teniendo en cuenta que lo que se pretende construir es un prototipo de dinámica parecida al AirWhale. En ese sentido, no se han tenido en cuenta la sustentación de las alas (debido al pobre diseño de las mismas en el prototipo y que se supone que el ángulo de cabeceo se controlará para que no sufra variaciones significativas) ni las fuerzas ni momentos de masa añadida (ver [3] para más información).

El modelo parte de la siguiente ecuación general:

$$M_{masa} \cdot \ddot{q} = \tau_I + \tau_G + \tau_C + \tau_{AS} + \tau_{AD} = \tau_{TOTAL} \quad (1)$$

Donde:

- M_{masa} es una matriz 6x6 que modela la masa del AirWhale.
- El vector de 6 componentes τ_I representa las fuerzas de inercia que afectan a la aeronave.
- τ_G corresponde a las fuerzas de gravedad.
- τ_C recoge las fuerzas de control.
- τ_{AS} representa las fuerzas de sustentación del helio.
- τ_{AD} corresponde a las fuerzas aerodinámicas que sufre el AirWhale.
- $\ddot{q}$ representa la segunda derivada temporal del vector de estados.

La ecuación (1) y siguientes se basan en los sistemas de coordenadas que aparecen en la Figura 3.1. En ella se puede observar cómo el eje X del sistema referido al centro de volumen sigue la dirección longitudinal del AirWhale, apuntando al morro de la aeronave, y el eje Z apunta hacia debajo de forma perpendicular.

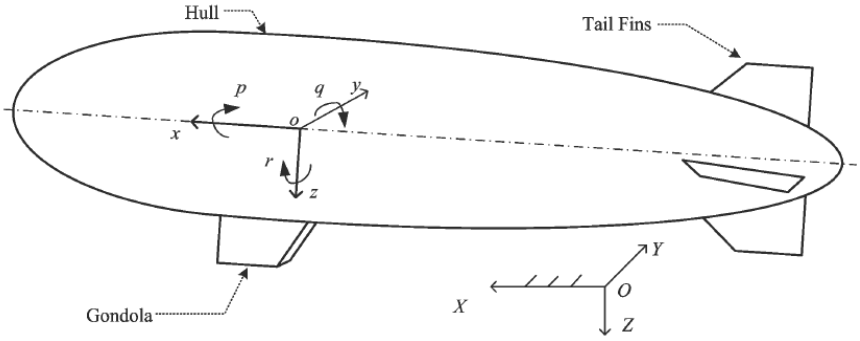

Figura 3.1. Ilustración de los sistemas de coordenadas empleados [3]

En este sentido, se define el vector de estados como:

$$q = \begin{pmatrix} x \\ y \\ z \\ p \\ q \\ r \end{pmatrix} \tag{2}$$

Donde x, y y z representan la posición del centro de volumen y p, q y r son los giros del propio sistema. Continuando con la definición de elementos, se expondrá a continuación la equivalencia a cada uno de los términos de la ecuación (1), puntualizando antes la siguiente igualdad:

$$e = \begin{pmatrix} e_1 \\ e_2 \\ e_3 \end{pmatrix};\ e^{\times} = \begin{pmatrix} 0 & -e_3 & e_1 \\ e_3 & 0 & -e_1 \\ -e_2 & e_1 & 0 \end{pmatrix} \tag{3}$$

Y definiendo la matriz de rotación en función de las coordenadas de Euler, que servirá para convertir las ecuaciones a coordenadas de Euler:

$$R = \begin{pmatrix} cos\psi cos\theta & sin\psi cos\theta & -sin\theta \\ cos\psi sin\theta sin\phi - sin\psi cos\phi & sin\psi sin\theta sin\phi - cos\psi cos\phi & cos\theta sin\phi \\ cos\psi sin\theta cos\phi - sin\psi sin\phi & sin\psi sin\theta cos\phi - cos\psi sin\phi & cos\theta cos\phi \end{pmatrix} \tag{4}$$

Así:

- Matriz de masa:

$$M_{masa} = \begin{pmatrix} m \cdot \boldsymbol{I} & -m \cdot \boldsymbol{r_G}^{\times} \\ m \cdot \boldsymbol{r_G}^{\times} & \boldsymbol{J} \end{pmatrix} \tag{5}$$

Donde m es la masa del AirWhale, r_G el vector de posición del origen de coordenadas (supuesto nulo), $\boldsymbol{I}$ la matriz identidad y $\boldsymbol{J}$ la matriz de inercia, que se ha simplificado a su forma diagonal.

- Fuerzas de inercia:

$$\tau_I = \begin{pmatrix} \boldsymbol{R} \cdot (-m \cdot \boldsymbol{w}^{\times} \cdot \boldsymbol{v} + m \cdot \boldsymbol{w}^{\times} \cdot \boldsymbol{r_G}^{\times} \cdot \boldsymbol{w}) \\ \boldsymbol{R} \cdot (-m \cdot \boldsymbol{r_G}^{\times} \cdot \boldsymbol{w} \cdot \boldsymbol{v} - \boldsymbol{w}^{\times} \cdot \boldsymbol{J} \cdot \boldsymbol{w}) \end{pmatrix} \quad (6)$$

Donde $\boldsymbol{v}$ representa el vector velocidad y $\boldsymbol{w}$ es el vector velocidad angular:

$$\boldsymbol{v} = \begin{pmatrix} U \\ V \\ W \end{pmatrix} \quad (7)$$

$$\boldsymbol{w} = \begin{pmatrix} p \\ q \\ r \end{pmatrix} \quad (8)$$

- Fuerzas gravitatorias:

$$\tau_G = \begin{pmatrix} m \cdot \boldsymbol{g} \\ \boldsymbol{r_G} \cdot m \cdot \boldsymbol{g} \end{pmatrix} \quad (9)$$

Siendo $\boldsymbol{g}$ el vector gravedad en coordenadas de Euler:

$$\boldsymbol{g} = \begin{pmatrix} -g \cdot sin\theta \\ g \cdot cos\theta \cdot sin\phi \\ g \cdot cos\theta \cdot cos\phi \end{pmatrix} \quad (10)$$

- Fuerzas de control:

$$\tau_C = \begin{pmatrix} \boldsymbol{R} \cdot \boldsymbol{F_m} \\ \boldsymbol{R} \cdot \boldsymbol{M_t} \end{pmatrix} \quad (11)$$

Donde $\boldsymbol{F_m}$ y $\boldsymbol{M_t}$ son los vectores de fuerzas motoras y pares en los ejes de coordenadas, respectivamente, de control, definidos como:

$$\boldsymbol{F_m} = \begin{pmatrix} f_x \\ 0 \\ -f_z \end{pmatrix} \quad (12)$$

$$\boldsymbol{M_t} = \begin{pmatrix} mt_x \\ mt_y \\ mt_z \end{pmatrix} \quad (13)$$

Se ha considerado la componente de control en Y de las fuerzas motoras nula, debido a que no se tiene ningún actuador que ejerza alguna fuerza en esa dirección.

- Fuerzas ejercidas por la masa de helio:

$$\tau_{AS} = m_{He} \cdot g \cdot \begin{pmatrix} \boldsymbol{R} \cdot \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{-1} \end{bmatrix} \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix} \tag{14}$$

Donde m_{He} es la masa que sustenta el helio del globo.

- Fuerzas aerodinámicas:

$$\tau_{AD} = \begin{pmatrix} -\frac{1}{2} \cdot \rho_{aire} \cdot U^2 \cdot S_{hx} \cdot C_d \\ -\frac{1}{2} \cdot \rho_{aire} \cdot V^2 \cdot S_{hy} \cdot C_d \\ -\frac{1}{2} \cdot \rho_{aire} \cdot W^2 \cdot S_{hz} \cdot C_d \\ 0 \\ 0 \\ 0 \end{pmatrix} \tag{15}$$

Donde ρ_{aire} es la densidad del aire; S_{hx}, S_{hy} y S_{hz} son las superficies frontales del AirWhale en cada uno de las direcciones cartesianas y C_d el coeficiente aerodinámico. Se ha supuesto que el 'drag' no afecta al giro de la aeronave y que el coeficiente aerodinámico que se aplica en las tres direcciones es el mismo. A este vector no se le aplica la matriz de rotación.

3.2 Modelo dinámico del AirWhale (espacio de estados)

Una vez definidos cada uno de los términos de la ecuación (1) a través de las ecuaciones (2~15), hay que marcar una nueva formulación más adecuada para la simulación y control de la misma. Esta formulación será la del espacio de estados en coordenadas de Euler:

$$\begin{cases} \dot{x}(t) = Ax(t) + Bu(t) \\ y(t) = Cx(t) + Du(t) \end{cases} \tag{16}$$

Donde $x(t)$ representa el vector de espacio de estados, $u(t)$ representa el vector de actuaciones que se aplica al sistema que modela el espacio de estados, $y(t)$ marca la salida del sistema y $A, B, C\ y\ D$ son matrices constantes que marcan la relación entre los vectores definidos anteriormente. Por simplicidad, en los cálculos siguientes se omitirá la notación que indica la dependencia temporal.

El nuevo vector de estados deseado es el siguiente:

$$\boldsymbol{x} = \begin{pmatrix} x \\ y \\ z \\ U \\ V \\ W \\ \phi \\ \theta \\ \psi \\ p \\ q \\ r \end{pmatrix} \tag{17}$$

Para proceder a obtener el espacio de estados del sistema, se hallará $\dot{x}(t)$. Para ello, basta con usar la ecuación (1) y la ecuación (18), que relaciona la derivada temporal de los ángulos de Euler con las coordenadas de estado p, q y r:

$$\dot{\boldsymbol{R}} = \boldsymbol{R} \cdot \boldsymbol{w} \tag{18}$$

Así, se obtienen las siguientes ecuaciones, que componen el espacio de estados no lineal:

$$\dot{\boldsymbol{x}} = \begin{cases} \dot{x} = U \\ \dot{y} = V \\ \dot{z} = W \\ \dot{U} = \left(\frac{1}{m}\right) \cdot \tau_{TOTAL_1} \\ \dot{V} = \left(\frac{1}{m}\right) \cdot \tau_{TOTAL_2} \\ \dot{W} = \left(\frac{1}{m}\right) \cdot \tau_{TOTAL_3} \\ \dot{\phi} = p + q \cdot sin\,\phi\, tan\,\theta + r \cdot cos\,\phi\, tan\,\theta \\ \dot{\theta} = q \cdot cos\,\phi - r \cdot sin\,\phi \\ \dot{\psi} = q \cdot sin\,\phi\, sec\,\theta + r \cdot cos\,\phi\, sec\,\theta \\ \dot{p} = \left(\frac{1}{I_{xx}}\right) \cdot \tau_{TOTAL_4} \\ \dot{q} = \left(\frac{1}{I_{yy}}\right) \cdot \tau_{TOTAL_5} \\ \dot{r} = \left(\frac{1}{I_{zz}}\right) \cdot \tau_{TOTAL_6} \end{cases} \tag{19}$$

Donde τ_{TOTAL_i} es la componente i-ésima del vector de fuerzas totales τ_{TOTAL}.

Las tres primeras ecuaciones son triviales: la derivada temporal de la posición es igual a la velocidad. Las siguientes tres ecuaciones y las tres últimas tienen su origen en la ecuación (1). Las tres ecuaciones restantes, las correspondientes a las derivadas temporales de los ángulos de Euler, se han obtenido partiendo de la ecuación (18).

El conjunto de ecuaciones (19) proporciona el espacio de estados no lineal de la aeronave. Esto quiere decir que se dispone de un modelo dinámico dependiente de los ángulos de Euler no lineal, y es posible simularlo. En el Anexo A se muestra el programa creado con el software Matlab que calcula la expresión completa de las

ecuaciones anteriores.

Por otro lado, para conseguir el espacio de estados lineal, hay que derivar parcialmente todas las ecuaciones reunidas en (19) con respecto a todas las variables de control y de estado, evaluarlas en el estado de equilibrio y multiplicarlas por una variable que represente las variaciones de la variable de control o de estado, con respecto a la que se ha derivado, en torno su valor de equilibrio. Es decir:

$$\dot{x}_{lin_i} = \sum_{k=1}^{n} \frac{\partial \dot{x}_i}{\partial u_k}\bigg|_{u^{eq},x^{eq}} \cdot {u^p}_k + \sum_{j=1}^{m} \frac{\partial \dot{x}_i}{\partial x_j}\bigg|_{u^{eq},x^{eq}} \cdot {x^p}_j \tag{20}$$

Donde $i = 1, \dots, 12$ representa la componente i-ésima del vector derivado de variables de estado; $m = 12$ representa el número de componentes del vector de variables de estados y $n = 5$ el número de actuaciones.

El vector $\dot{\boldsymbol{x}}_{\boldsymbol{lin}}$ reúne las ecuaciones que definen cada una de las variables de estado derivadas temporalmente linealizadas, $\dot{\boldsymbol{x}}$ ídem con las ecuaciones sin derivar, $\boldsymbol{u^{eq}}$ y $\boldsymbol{x^{eq}}$ son los vectores de actuaciones y de variables de estado particularizadas en el punto de funcionamiento, y $\boldsymbol{u^p}$ y $\boldsymbol{x^p}$ son los vectores que reúnen las variaciones en torno el punto de funcionamiento de las actuaciones y de las variables de estado.

Cabe mencionar un aspecto muy importante acerca del modelo. Este sistema se puede dividir en dos subsistemas: un subsistema que modela la traslación de la aeronave y otro subsistema que modela la rotación de la misma. Lógicamente, el problema se simplifica, pero hay que tener en cuenta que las variables de estado y de control de cada submodelo no serán los mismos.

En el caso del subsistema de traslación, las ecuaciones que modelan el comportamiento son las relativas a la traslación, es decir, las que definen las velocidades y aceleraciones lineales. En este sentido, las variables de estado serían x, y, z, U, V y W; y se considerarían variables de control las fuerzas f_x y f_z, y los ángulos de Euler. Para ilustrar este concepto, se presenta la Figura 3.2.

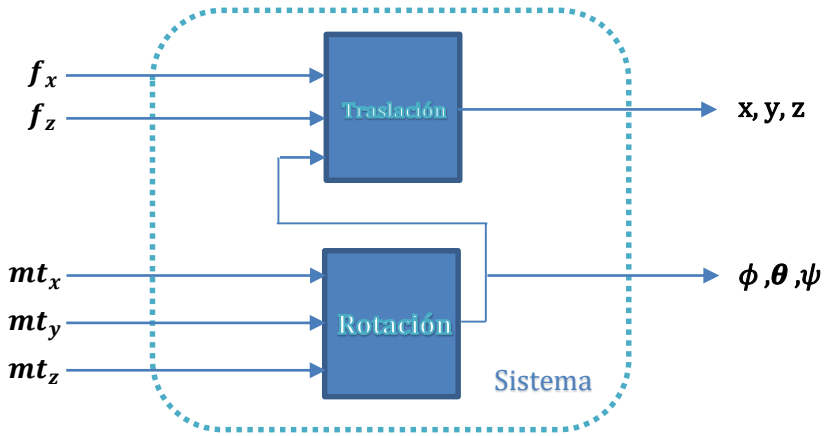

Figura 3.2. Esquema de la relación entre subsistemas de traslación y rotación

Los ángulos, por ejemplo, serán variables de control (o de entrada) en el subsistema de traslación aunque en el sistema global sean consideradas variables de estado. Este concepto afecta directamente a la definición del espacio de estados lineal. En el caso del subsistema de rotación, se tiene como variables de estado los ángulos de Euler y las variables p, q y r; mientras que las variables de control se identifican con los pares en cada eje: mt_x, mt_y y mt_x.

Realizando la operación definida en (20), que también realiza el archivo mostrado en el Anexo A; teniendo en cuenta lo expuesto anteriormente sobre las variables de estado y de control de cada subsistema, y suponiendo el punto de funcionamiento dinámico siguiente:

$$\boldsymbol{p^{eq}} = \begin{pmatrix} f_x^{eq} \\ f_z^{eq} \\ mt_x^{eq} \\ mt_y^{eq} \\ mt_z^{eq} \\ U^{eq} \\ V^{eq} \\ W^{eq} \\ \phi^{eq} \\ \theta^{eq} \\ \psi^{eq} \\ p^{eq} \\ q^{eq} \\ r^{eq} \end{pmatrix} = \begin{pmatrix} \frac{1}{2} \cdot \rho_{aire} \cdot U^2 \cdot S_{hx} \cdot C_d \\ g \cdot m - g \cdot m_{He} \\ 0 \\ 0 \\ 0 \\ U^{op} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

Se calculan las siguientes ecuaciones que definen el espacio de estados lineal:

$$\begin{cases} \dot{x} = U^p \\ \dot{y} = V^p \\ \dot{z} = W^p \\ \dot{U} = \frac{f_x^p}{m} + \frac{(g \cdot m_{He} - 2 \cdot m_{He} + g \cdot m) \cdot \theta^p}{m} - \frac{(C_d \cdot S_{hx} \cdot U^{op} \cdot \rho_{aire}) \cdot U^p}{m} \\ \dot{V} = -\frac{(g \cdot m_{He} - 2 \cdot m_{He} + g \cdot m) \cdot \phi^p}{m} - \frac{\left(C_d \cdot S_{hx} \cdot U^{op\,2} \cdot \rho_{aire}\right) \cdot \psi^p}{m} \\ \dot{W} = -\frac{f_z^p}{m} + \frac{\left(C_d \cdot S_{hx} \cdot U^{op\,2} \cdot \rho_{aire}\right) \cdot \theta^p}{m} \\ \dot{\phi} = p^p \\ \dot{\theta} = q^p \\ \dot{\psi} = r^p \\ \dot{p} = \frac{mt_x^p}{I_{xx}} \\ \dot{q} = \frac{mt_y^p}{I_{yy}} \\ \dot{r} = \frac{mt_z^p}{I_{zz}} \end{cases} \tag{22}$$

Debido a la separación del sistema en dos subsistemas mencionados anteriormente, también puede asociarse a cada uno de ellos un subespacio de estados. Así, se presenta el espacio de estados correspondiente al subsistema de traslación:

$$\begin{cases} \dot{\boldsymbol{x}}_{\boldsymbol{t}}(t) = \boldsymbol{A_t x_t}(t) + \boldsymbol{B_t u_t}(t) \\ \boldsymbol{y}(t) = \boldsymbol{C_t x_t}(t) \end{cases} \tag{23}$$

Donde el subíndice ‘t’ indica ahora la pertenencia al subsistema de traslación:

$$\dot{\boldsymbol{x}}_{\boldsymbol{t}}(t) = \begin{pmatrix} x^p(t) \\ y^p(t) \\ z^p(t) \\ U^p(t) \\ V^p(t) \\ W^p(t) \end{pmatrix} \tag{24}$$

$$\boldsymbol{u}_{\boldsymbol{t}}(t) = \begin{pmatrix} {f_x}^p(t) \\ {f_z}^p(t) \\ \phi^p(t) \\ \theta^p(t) \\ \psi^p(t) \end{pmatrix} \tag{25}$$

$$\boldsymbol{y}_{\boldsymbol{t}}(t) = \begin{pmatrix} x^p(t) \\ y^p(t) \\ z^p(t) \end{pmatrix} \tag{26}$$

$$\boldsymbol{A}_{\boldsymbol{t}} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\dfrac{(C_d \cdot S_{hx} \cdot U^{op} \cdot \rho_{aire})}{m} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{27}$$

$$\boldsymbol{B}_{\boldsymbol{t}} = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ \dfrac{1}{m} & 0 & 0 & \dfrac{(g \cdot m_{He} - 2 \cdot m_{He} + g \cdot m)}{m} & 0 \\ 0 & 0 & -\dfrac{(g \cdot m_{He} - 2 \cdot m_{He} + g \cdot m)}{m} & 0 & -\dfrac{\left(C_d \cdot S_{hx} \cdot U^{op^2} \cdot \rho_{aire}\right) \cdot \psi^p}{m} \\ 0 & -\dfrac{1}{m} & 0 & \dfrac{\left(C_d \cdot S_{hx} \cdot U^{op^2} \cdot \rho_{aire}\right)}{m} & 0 \end{pmatrix} \tag{28}$$

$$\boldsymbol{C}_{\boldsymbol{t}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{29}$$

De igual forma, se puede obtener el espacio de estados correspondiente al subsistema de rotación:

$$\begin{cases} \dot{\boldsymbol{x}}_{\boldsymbol{r}}(t) = \boldsymbol{A}_{\boldsymbol{r}}\boldsymbol{x}_{\boldsymbol{r}}(t) + \boldsymbol{B}_{\boldsymbol{r}}\boldsymbol{u}_{\boldsymbol{r}}(t) \\ \boldsymbol{y}(t) = \boldsymbol{C}_{\boldsymbol{r}}\boldsymbol{x}_{\boldsymbol{r}}(t) \end{cases} \tag{30}$$

Donde el subíndice ‘r’ marca su particularización al subsistema de rotación:

$$\dot{\boldsymbol{x}}_{\boldsymbol{r}}(t) = \begin{pmatrix} \phi^p(t) \\ \theta^p(t) \\ \psi^p(t) \\ p^p(t) \\ q^p(t) \\ r^p(t) \end{pmatrix} \tag{31}$$

$$\boldsymbol{u}_{\boldsymbol{r}}(t) = \begin{pmatrix} mt_x^p(t) \\ mt_y^p(t) \\ mt_z^p(t) \end{pmatrix} \tag{32}$$

$$\boldsymbol{y}_{\boldsymbol{r}}(t) = \begin{pmatrix} \phi^p(t) \\ \theta^p(t) \\ \psi^p(t) \end{pmatrix} \tag{33}$$

$$\boldsymbol{A}_{\boldsymbol{r}} = \begin{pmatrix} 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \tag{34}$$

$$\boldsymbol{B}_{\boldsymbol{r}} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \frac{1}{I_{xx}} & 0 & 0 \\ 0 & \frac{1}{I_{yy}} & 0 \\ 0 & 0 & \frac{1}{I_{zz}} \end{pmatrix} \tag{35}$$

$$\boldsymbol{C}_{\boldsymbol{r}} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \tag{36}$$

Las ecuaciones (23~36) definen completamente los espacios de estados lineales asociados a los subsistemas de traslación y rotación y, por ende, definen el sistema linealizado completo.

La simulación del sistema mostrado durante este capítulo se ha decidido incluir en el trabajo de la referencia [4]. Todo el equipo Automático ha estado realizando simulaciones activamente sobre este sistema debido a su trascendencia para todos los trabajos individuales, pero quizá sea más trascendente para la creación de los controladores PID, por lo que se ha decidido mostrar la simulación y la validación del modelo dinámico, tanto en su forma lineal como en su forma no lineal, en el trabajo de José Luis Holgado Álvarez.

4 SELECCIÓN Y COMPRA DE LOS COMPONENTES ELECTRÓNICOS

Definido el modelo dinámico que representa fielmente el comportamiento de la aeronave a construir, guardando grandes similitudes con el AirWhale definido en los trabajos del equipo Mecánico, cabe abordar qué componentes electrónicos se usarán para el vuelo de la misma. Esta electrónica hace referencia tanto a los actuadores (motores y servos), como los sensores (GPS, sónar…) y conexionado entre unos y otros (cableado, adaptadores, distribuidores de potencia…), además de la alimentación de todos ellos (baterías). Todos los componentes han sido seleccionados cuidadosamente junto con el alumno Juan Carlos Martín Rodríguez, teniendo en mente la construcción del prototipo, realizándose cambios de última hora en la misma (se indicarán en su respectivo capítulo); y de la versión final del AirWhale. En este apartado se hará una revisión de todas las compras realizadas por la asociación EsiTech y el Departamento de Ingeniería de Sistemas y Automática, justificándolas.

4.1 Plataforma ArduPilot

La plataforma autopiloto fue escogida específicamente por el Departamento de Ingeniería de Sistemas y Automática. La razón fue el conocimiento de las bondades de su uso por parte del testimonio del profesor Francisco Gavilán Jiménez, que, en esos momentos, tutelaba el proyecto del alumno Juan Antonio García Fernández, estudiante del Grado en Ingeniería Aeroespacial [17]. En él, el alumno desarrollaba un sistema hardware-in-the-loop usando la plataforma ArduPilot y otros componentes para observar una simulación en vuelo.

El profesor Francisco Gavilán recomendó la compra y uso de la plataforma ArduPilot, y así se hizo. El alumno que escribe este texto ha tenido que trabajar con la plataforma sin conocer previamente cómo gestiona sus recursos y cómo se programa en ella, aún entendiéndose mejor con otras plataformas igual de válidas e, incluso, más potentes.

La plataforma pesa, aproximadamente, 28 gramos. Este dato es importante de cara a la construcción del prototipo y a la selección de motores y baterías, y también se indicará en los siguientes dispositivos.

4.2 Sensores

Junto con la plataforma ArduPilot, se compraron los sensores externos a ella. Estos también fueron escogidos por el Departamento de Ingeniería de Sistemas y Automática por recomendación del profesor Francisco Gavilán. Estos sensores son:

- 3DR uBlox GPS: Un módulo que incorpora al ArduPilot un sensor GPS y un magnetómetro externo. Lógicamente, aporta al aparato un dato de posición en todo momento, siempre y cuando su vuelo sea en exterioresDispone de un puerto de 6 pines para conectar el sensor GPS a la placa (en el puerto de uso exclusivo para ello) y un puerto de 4 pines para conectar el magnetómetro (puerto I2C).

Figura 4.1. Módulo 3DR uBlox GPS con magnetómetro incorporado

La flecha que aparece en el dispositivo marca la posición en la que se debe colocar en el momento del montaje en la aeronave (eje X).

Las características generales del módulo son [18]:

- Módulo uBlox NEO7, con refresco de 5 Hz.
- Filtros LNA (Low Noise Amplifier) y SAW (Surface Acoustic Wave) incorporados para mejorar la señal proveniente de la antena, además de un regulador de tensión de ruido bajo de 3.3 V.
- Tamaño: 38x38x8.5mm. Peso: 16.8 gramos.

- 3DR Power Module: Su función principal no es la de sensor, sino la de suponer un puente entre la alimentación principal y la plataforma ArduPilot. Dispone de dos conectadores XT60 (hembra-macho), uno para la conexión de la/s batería/s y otro en el que se conectarán los motores mediante algún tipo de conexión casera o una placa de distribución de potencia. También cuenta con un sensor de corriente/voltaje asociado a la batería para informar de la carga restante, útil para finalizar el vuelo antes de la descarga completa de la misma y para medir la autonomía de forma fiable y directa.

Figura 4.2. 3DR Power Module

En concreto, dispone de las siguientes especificaciones de interés [19]:

- Tensión máxima admisible de entrada: 18 V (equivalente a una batería LiPo de 4 celdas). Tensión mínima admisible: 4.5 V. Intensidad máxima admisible: 90 A (para la placa PIXHAWK, esta corriente baja a 60 A).
- Proporciona 5.3 V y 2.25 A como máximo, regulados.
- Puerto de 6 pines para la conexión con la plataforma ArduPilot (en el puerto destinado para ello), alimentando a la placa y aportando datos sobre la carga de la batería en tiempo real.
- Tamaño: 25x21x9mm. Peso: 17 gramos.

- 3DR Radio Set: La necesidad de la monitorización de los distintos datos administrados por la plataforma ArduPilot en vuelo supone el uso de un sistema de telemetría. El producto incluye dos módulos de radio con antenas, un cable microUSB, un cable adaptador OTG para Android y cables conectores con APM. Estas son sus especificaciones [20]:
    - 433 MHz de frecuencia, de acuerdo con la legislación vigente del país y de la Unión Europea. También puede trabajar a 915 MHz en adaptación con las leyes americanas.
    - Puerto de 6 pines para la conexión con la plataforma ArduPilot (en el puerto específico para ello).
    - 100 mW de potencia de salida máxima, además de una sensibilidad en la recepción de -117 dBm y el uso de la técnica TMD para la comunicación.
    - Conector RP-SMA para la incorporación de una antena.
    - Modulación FHSS, interfaz UART, soporte del protocolo Mavlink y configurable mediante Mission Planner.
    - Voltaje de alimentación 3.7-6 V en continua, corriente en transmisión de 20 mA a 20 dB, corriente en recepción de 25 mA y tensión en interfaz UART de 3.3 V.
    - Tamaño: 26.7x55.5x13.3mm. Peso: ~ 20 gramos.

Figura 4.3. 3DR Radio Set

- Sónar XL-MAXSONAR-EZ/AE MB1240: Este sensor se usa frecuentemente para la monitorización de la altitud (dirigiéndolo al suelo) o para la detección de obstáculos. Al disponer la plataforma ArduPilot de un barómetro que calcula automáticamente la altitud a partir de la presión, la utilidad de este sensor, en caso de emplearse, será la de detección de elementos con los que la aeronave pueda chocar.

Figura 4.4. Sónar XL-MAXSONAR-EZ MB1240

Algunas de sus especificaciones son [21]:

- Resolución de 1cm y lectura a 10 Hz.
- En general, los objetos a distancias menores de 25 cm se detectan a 25 cm.
- Actúa a 3.3-5.5V y a 3.4 mA, y se puede leer la distancia, directa o indirectamente, de tres salidas propias del sónar: voltaje analógico, en forma de puerto serie con formato RS232 y PWM, realizando ajustes en los pines del sensor.
- Rango de funcionamiento de temperaturas de 0 a 65ºC, calibración automática en tiempo real y filtrado.
- Detección de obstáculos a una distancia máxima de 765 cm.
- Tamaño: 22.1x19.9x25.1mm. Peso: 6.1 gramos.

4.3 Almacenamiento

Para almacenamiento de datos en vuelo, viene incluido en las compras realizadas por el Departamento un dispositivo de almacenamiento USB: DataTraveler microDuo 3.0, de Kingston Technology. Se caracteriza por [22]:

- 32 GB de almacenamiento.
- 70 MB por segundo de lectura y 15 MB por segundo de escritura.
- Interfaz compatible con USB 3.0 y microUSB.
- Tamaño: 27.63x16.46x8.56mm. Peso: 45.36 g.

Figura 4.5. Dispositivo de almacenamiento DataTraveler microDuo 3.0

4.4 Alimentación

Las baterías son uno de los componentes más importantes del proyecto, debido al objetivo principal del mismo: aumentar la autonomía de los UAV actuales. Las baterías LiPo que se pueden encontrar en el mercado están, generalmente, clasificadas según la tensión que aportan a la salida de sus bornes, y este parámetro determina el número de celdas que poseen (cada celda posee un voltaje de 3.7 V). A su vez, la tensión determina el regulador y el motor que se van a conectar a dicha batería. Este último factor se debe unir a los ya importantes parámetros del peso y la capacidad de la batería. Por ello, hay que llegar a una solución de compromiso, siempre teniendo en mente conseguir la mayor autonomía posible.

En primera instancia, repasemos la capacidad de la batería. Normalmente ésta se da en miliamperios por hora (mAh) y es el factor más determinante en relación a la autonomía. Cuanta más capacidad tiene la batería, mayor autonomía de vuelo. Hay multitud de baterías en el mercado, todas con una gran variedad de capacidades, desde 600 mAh (más baratas) hasta 6000 mAh (más caras). Hay valores en el mercado que se salen de este rango, pero abandonan los precios y la autonomía deseados, por lo que no son de interés en este proyecto.

Conforme aumenta la capacidad de la batería, generalmente también lo hace el peso. Afortunadamente, como se verá en la siguiente sección, los elementos estructurales y la electrónica presentada anteriormente no suponen un gran aumento del peso de la aeronave, y el helio lo contrarresta completamente, por lo que hay margen para seleccionar baterías de gran capacidad aunque su peso sea medianamente alto.

Bajo estas premisas, y después de buscar detenidamente en el mercado, se han seleccionado las baterías Zippy 5000mAh 3S (11.1V) 20C-30C.

Figura 4.6. Batería Zippy 5000mAh 3S 20C-30C

Por el nombre, puede saberse que posee una capacidad de 5 amperios por hora y proporciona 11.1 V a la salida (3S = tres celdas). El último término, 20C-30C, hace referencia al índice de descarga.

El primer término (20C) hace referencia a la intensidad de descarga constante máxima que es capaz de proporcionar la batería. Es decir, en este caso: 20·5000mA = 100000mA = 100 A. Este dato determina que la batería puede proporcionar una corriente constante de 100 A, si el sistema así lo requiriese. Este índice, por tanto, debe ser alto, y normalmente guardar un margen de seguridad del 50% al 100%.

El segundo término (30C) se refiere a la intensidad de descarga puntual máxima que puede proporcionar la batería. En este caso: 30·5000mA = 150000mA = 150 A, es decir, de forma puntual y esporádica, la batería podría llegar a dar una intensidad de descarga de 150 A.

Las dimensiones de la batería son 138x47x25mm y tiene un peso aproximado de 300 gramos.

4.5 Actuadores

Los dispositivos electrónicos actuadores, aquellos que llevarán a cabo las fuerzas y pares necesarios para el control óptimo de la aeronave, se seleccionan, principalmente, a partir de un dato clave: el peso de la estructura. Es claro que, si lo que se desea es que algún vehículo vuele, cuánto más pese, mayor debe ser la fuerza que ejercen los actuadores para conseguir el vuelo. En una primera aproximación, dado que la intención es que los actuadores que se seleccionen también puedan emplearse en la versión final del AirWhale, se va a partir del dato del peso de la estructura reflejado en los cálculos del equipo Mecánico-Aeronáutico y de la electrónica anteriormente presentada, además de la que está por presentar.

En cuanto a la estructura del AirWhale, mediante los trabajos del equipo Mecánico-Aeronáutico, se pueden recoger los siguientes pesos [2]:

- Peso del cuerpo y estabilizadores: 1.3 Kg.
- Peso de las alas: 0.3 Kg.

Además, la alumna Inmaculada Gómez calcula en su trabajo [2] la sustentación que ejerce el volumen de helio confinado por la estructura: 2.11 Kg. Esto, contando sólo la estructura propia del AirWhale, sin añadir en la ecuación el peso de la electrónica, deja un balance negativo de 510 gramos. Es decir, olvidando la electrónica, la estructura volaría por sí misma gracias a la acción del helio.

Ahora se reflejará el peso de la electrónica presentada hasta ahora, esto es: plataforma autopiloto, sensores, dispositivos de almacenamiento y baterías:

- ArduPilot: 28 gramos.
- GPS: 16.8 gramos.
- Módulo de alimentación: 17 gramos.
- Telemetría: 20 gramos.
- Sónar: 6.1 gramos.
- Dispositivo almacenamiento USB: 45.36 gramos.
- 4 baterías Zippy 5000 mAh 3S 20-30C (se decidio usar una por cada motor): 1200 gramos.

Lo que hace un total de 1333.46 gramos. Esto, en el balance general, supone un total de 823.46 gramos a levantar por los motores, sin contar el peso de los mismos y de los variadores/reguladores que les acompañan, de los cables y de otros componentes.

Para el control de la aeronave, hay que recordar que son necesarios cuatro motores, dos verticales y dos horizontales, y cuatro servomotores (con control de posición incorporado) para los cuatro estabilizadores mostrados en el diseño final. Posteriormente, en el montaje del prototipo, se podrá comprobar cómo el número de servomotores se reduce a dos, por simplicidad y para habilitar el control del cabeceo de la aeronave.

4.5.1 Motores principales

Empecemos seleccionando los motores. Una vez escogidas las baterías y determinado el peso que hay que levantar, es fácil encontrar un motor adecuado para cumplir la misión de este trabajo. De las baterías se sabe que los motores deben funcionar a una tensión máxima de 11.1 V, en concordancia con la máxima que puede aportar una batería de las seleccionadas. En cuanto al peso, se sabe que, mínimo, se necesitarán levantar aproximadamente 825 gramos, cantidad que aumentará con el peso de los motores escogidos y demás componentes. Por ello, hay que buscar un motor poco pesado que no aumente demasiado el peso total de la estructura, que ejerza una fuerza suficiente como para levantar la aeronave y que consuma poca potencia, de manera que no exija demasiado a las baterías y no afecte considerablemente a la autonomía.

Sabiendo todos estos factores, se decide adquirir el motor Brushless outrunner EMAX BL2210/30 - 1300kV, mostrado en la Figura 4.7.

Figura 4.7. Motor Brushless EMAX BL2210/30

Las características de interés de este motor se resumen en:

- Tensión de entrada: 7.4-11.1 V.
- Consumo: 166 W. 1300 RPM por voltio.
- Peso: 45 gramos.
- Corriente máxima: 16.5 A. El 75% de esta corriente da la máxima eficiencia.
- Empuje máximo: 800 gramos (implementando hélices de 9x4.7 pulgadas).

Dados los datos anteriores, añadir estos cuatro motores a la estructura supone una sustentación máxima de 1600 gramos y un peso extra de 180 gramos. En total, una estructura de 1003.46 gramos totalmente cubiertos por la acción de los motores.

4.5.2 Hélices propulsoras

Los motores seleccionados anteriormente no podrían elevar ninguna aeronave sin acoplarles hélices que ejerzan la fuerza necesaria de sustentación. Para seleccionar qué tipo de hélices son las correctas con el tipo de motor seleccionado, el propio fabricante ofrece recomendaciones sobre las dimensiones de las mismas, añadiendo el empuje que generaría el tándem motor-hélice.

En el caso de emplear una hélice de tamaño 9x4.7 pulgadas, el motor seleccionado aporta el empuje máximo de 800 gramos, así que será éste el tamaño buscado. Con respecto al eje de la hélice, sería deseable que guardase gran similitud con el diámetro del eje del motor, pero puede no ser necesario dependiendo de la conexión usada entre hélice y motor. Con los motores se incluyen gomillas, elemento que permite una unión que no necesita imprescindiblemente la semejanza de ejes. Por último, resta decidir el material de las hélices. Existe varios materiales: madera, plástico ABS, fibra de carbono… Debido a su rigidez, que supone resistencia a vibraciones, y poco peso, se ha optado por hélices de fibra de carbono.

Las hélices escogidas son de fibra de carbono, 9x4.7 pulgadas y eje de 8mm. Cada una pesa 10 gramos, y al añadir una por cada motor, también se añaden 40 gramos más a la estructura: 1043.46 gramos en total. Se hará uso de hélices rotatorias y contrarotatorias.

Figura 4.8. Hélice 9x4.7 de fibra de carbono.

4.5.3 Servomotores

Los servomotores, por otro lado, deben ser capaces de soportar la fuerza que pueda ejercer el aire en la superficie de control, además del propio peso de la misma, y es conveniente que consuman lo menos posible. Los servomotores mostrados en el mercado suelen venir acompañados de dos parámetros principales: el 'torque' y la velocidad de actuación, y cada uno de ellos depende de la tensión de alimentación que reciba el servo.

El 'torque' es una medida referida a la fuerza que puede ejercer el servomotor. Concretamente, se mide en kilogramos por centímetro (Kg/cm), e indica los kilogramos de fuerza que ejerce el servo al girar al final del brazo por centímetro de brazo, asemejándose al concepto tradicional de momento. Conviene, por tanto, que sea lo más alto posible.

La velocidad de actuación es un concepto más intuitivo; se mide en segundos por cada sesenta grados (s/60º) e indica el tiempo que tarda el servomotor en moverse 60 grados. Este dato determina completamente la velocidad del servo en su giro, e interesa que sea lo más rápido posible, sobre todo si de estos dispositivos depende el control de alguna variable de estado de la aeronave.

El voltaje al que se alimenta el servomotor influye en lo anterior. En las especificaciones facilitadas por los fabricantes, suelen indicar el 'torque' y la velocidad de actuación a dos voltajes comunes: 4.8 V y 6.0 V. En nuestro caso, como no es posible alimentar directamente los servomotores desde las baterías (debido a su alto voltaje), se alimentarán directamente desde la placa.

El servomotor seleccionado es el TowerPro SG92r 9G, que dispone de las siguientes características:

- 'Torque': 1.5 Kg/cm a 4.7 V, 1.6 Kg/cm a 6 V.
- Velocidad de actuación: 0.12 a 4.7 V.
- Rango de voltaje admitido: 3.0-7.2 V.
- Tamaño: 22x11.5x27mm. Peso: 9 gramos.

Lo que supone añadir a la estructura 36 gramos más: 1079.43 gramos en total.

Figura 4.9. Servomotor TowerPro SG92r 9G

4.6 Variadores/reguladores

Los variadores/reguladores son dispositivos que se encargan de variar la velocidad del motor desde su mínimo hasta su máximo, y de transformar la corriente continua que le llega desde la batería a trifásica, que es la que solicita el motor. Además, supone la conexión de las salidas PWM de la placa con el motor. Estos dispositivos son muy variados y pueden disponer de multitud de características, algunas dependientes de la batería a la que se conecten y otras del motor al que alimenta.

La intensidad que ofrecen estos dispositivos depende exclusivamente de la intensidad máxima que aceptan los motores a los que están conectados. Es recomendable guardar un margen de seguridad con respecto a este valor para seleccionar el variador, siendo frecuente que el fabricante del motor recomiende el tipo de variador a usar. En el caso de nuestro motor, se recomienda un variador con una intensidad continua de 18 A como mínimo. Existe un amperaje máximo adicional que puede proporcionar el variador de forma puntual, mayor que el valor continuo.

Normalmente, el fabricante del variador también indica qué tipo de batería recomienda conectar con el mismo, concretamente determinando el número de celdas (voltaje) que debe caracterizar a la batería.

Aparte de estos parámetros, los variadores pueden disponer de BEC (Battery Elimination Circuit). Es, básicamente, un regulador que obtiene la tensión e intensidad provenientes de la batería y los convierte en un voltaje y amperaje estables, de manera que puede llegar a alimentar elementos que requieran poca potencia para funcionar. Esta alimentación acompaña al cable de señal proveniente de la placa, y es posible usarla para la alimentación de la plataforma ArduPilot, como se detallará en la construcción del prototipo y conexionado. Se optará por un variador que incluya este sistema de alimentación, en pro de la versatilidad que ofrece.

El seleccionado es el regulador Brushless EMAX 18A Budget, mostrado en la Figura 4.10.

Figura 4.10. Regulador Brushless EMAX 18A Budget

Caracterizado por:

- Amperaje continuo: 18 A. Amperaje máximo: 23 A.
- Compatible con baterías 2-3S.
- Dispone de BEC que suministra 5 V y 2 A.
- Tamaño: 55x30x15mm. Peso: 24 gramos.

Como debe acompañar uno a cada motor, el peso que se añade a la estructura es de 96 gramos. Esto hace un total de 1175.43 gramos.

Por último, añadir que este variador es posible programarlo, pudiendo elegir distintas opciones relacionadas con la alimentación BEC, la velocidad de cambio de velocidad del motor y otros. Para una sencilla programación, se hará uso de la EMAX Program Card.

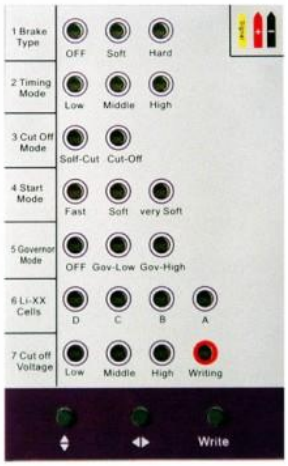

Figura 4.11. EMAX Program Card.

4.7 Conexionado y otros componentes

Para la conexión de los dispositivos anteriores, se ha visto necesaria la compra de ciertos componentes que se nombran a continuación:

- Placa de distribución de corriente: Para la conexión efectiva y sencilla de los cuatro motores en paralelo con las baterías, es útil el uso de una placa de distribución, a la cual se conectarían los variadores y la batería (o en su defecto el módulo 3DR Power Module). También dispone de conectores destinados a las salidas de la placa, permitiendo, así, el control PWM de los motores y la alimentación de la misma.

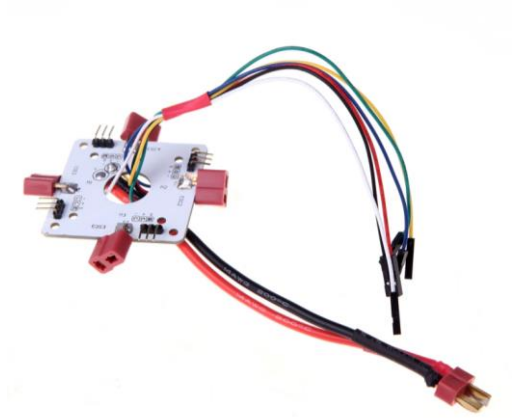

Figura 4.12. Placa de distribución de corriente

Dispone de:

- Cuatro conectores macho para la conexión del BEC y el cable de señal de los variadores.
- Cuatro conectores t-dean para la conexión de alimentación de los variadores.
- Seis conexiones dirigidas al APM, cuatro de señal PWM y dos de alimentación.
- Conexión t-dean macho para la batería o el módulo de alimentación 3DR Power Module.
- Peso: 27 gramos aproximadamente.

- Adaptadores de conexión: Para el correcto conexionado de los distintos componentes, es necesario adaptadores para que coincidan las terminaciones de ciertos cables. Estos se enumeran a continuación, detallando en qué conexión son necesarios:
  - Adaptador de XT60 macho a t-dean hembra: necesario para la conexión del 3DR Power Module con la placa de distribución de corriente.

Figura 4.13. Adaptador XT60 macho t-dean hembra.

  - Conector t-dean macho-hembra: Necesario para la conexión de los variadores con la placa de distribución de corriente (sólo necesaria la conexión macho).

Figura 4.14. Conector t-dean macho-hembra.

- Conector XT60 macho-hembra: Necesario para la conexión de la batería con el 3DR Power Module o con la placa de distribución de corriente (la batería trae terminaciones banana de 3mm).

Figura 4.15. Conector XT60 macho-hembra.

- Cableado 16 AWG: Debido a que las dimensiones de la aeronave son considerables y mucho mayores que las tradicionales de un quadrotor, los variadores y la placa APM no vienen preparados para conexiones tan alejadas entre sí. Es por ello que es necesaria la compra de cableado adicional para alargar las conexiones.
- Cargador IMAX B6AC: Para la recarga de las baterías y su reutilización, se ha visto conveniente la compra de este cargador capaz de alimentar baterías de hasta 6 celdas.

Figura 4.16. Cargador IMAX B6AC.

4.8 Balance de pesos

En forma de resumen, se expone a continuación, en la Tabla 4.1, un balance de los pesos de todos los elementos que se añadirán a la estructura de la versión final del AirWhale. En el Capítulo 6 se detallará este mismo balance para el prototipo construido, ya que no se usan todos los materiales en la misma cantidad y los pesos de la estructura cambian.

Tabla 4-A. Pesos de los distintos elementos de la aeronave AirWhale y balance

Elemento	Peso/Sustentación (gramos)
Cuerpo y estabilizadores	1200
Alas	300
Plataforma ArduPilot Mega 2.6	28
GPS 3DR uBlox	16.8
3DR Power Module	17
3DR Radio Set	20
Sónar XL-MAXSONAR-EZ MB1240	6.1
Dispositivo de almacenamiento USB	45.36
Batería Zippy 5000mAh (x4)	1200
Motor Brushless EMAX BL2210/30	180
Hélice 9x4.7 pulgadas fibra de carbono	40
Servomotor TowerPro SG92r 9G	36
Regulador Brushless EMAX 18A Budget	96
Placa de distribución de corriente	27
Otros componentes de conexionado	~ 40
TOTAL	**3252.26**
Sustentación del helio	2110
Sustentación de los motores	1600
TOTAL TRAS SUSTENTACIÓN	**457.74 (de sustentación)**

Es de destacar el margen de sustentación que proporciona tanto el volumen de helio como la actuación de los motores verticales a máxima potencia. Es de suponer que los motores no estarán funcionando para proporcionar el máximo empuje posible, sino sólo para la estabilidad de la altitud de la nave. Ello se consigue haciendo que el balance de pesos sea nulo, es decir, que el empuje proporcionado por los motores sea **1600 - 457.74 = 1142.26** gramos, equivalente a 571.13 gramos aportados por cada motor vertical en sustentación[1].

4.9 Presupuesto final

El precio de cada uno de los elementos descritos anteriormente se detalla a continuación. El presupuesto detallado, provisto de los enlaces web donde se puede realizar la compra de cada producto y encontrar sus características, las facturas de dichas compras, y otros detalles, se pueden repasar en el Anexo B.

Obsérvese que hay productos de los que, aún estando previsto su uso en la versión final del AirWhale, no se han comprado las unidades necesarias, debido a que no era urgente; las compras realizadas tienen como objetivo

[1] Estas cantidades están sujetas a posibles modificaciones en el momento de la construcción y vuelo de la versión final del AirWhale

permitir la construcción completa del prototipo desarrollado en el Capítulo 6, en el que se podrá conocer los productos usados.

El dinero total desembolsado proviene del fondo concedido a la asociación EsiTech por parte de la Universidad de Sevilla.

Tabla 4-B. Presupuesto final

Nombre	Cantidad	Precio Ud. (€)	Precio total (€)
Motor Brushless EMAX BL2210/30	6	18.90	113.4
Regulador Brushless EMAX 18A Budget	6	15.90	95.4
Bateria Zippy 5000mAh 3S	2	36.95	73.9
Adaptador XT60 macho a t-dean hembra	1	2.90	2.90
Cable de silicona 16AWG (1 metro)	4	2.50	10.00
Conector tdean macho hembra	4	4.00	4.00
Conector XT60 hembra macho	1	1.65	1.65
Cargador IMAX B6AC	1	37.99	37.99
Servo TowerPro SG92r 9G	2	4.95	9.90
EMAX Program Card	1	4.90	4.90
Placa de distribución de corriente	1	6.50	6.50
Hélice de Fibra de Carbono 9x4,7" (CCW)	4	7.90	31.6
Hélice de Fibra de Carbono 9x4,7" (CW)	4	7.90	31.6
TOTAL			**423.74**

5 FIRMWARE ARDUCOPTER Y CREACIÓN DE UN FIRMWARE PARA LA AERONAVE AIRWHALE

La causa principal de que el AirWhale opere según leyes de control óptimas es el firmware añadido a la plataforma ArduPilot. Durante años se han ido desarrollando multitud de archivos por parte de la comunidad asociada al proyecto ArduPilot que, en conjunto, son capaces de gestionar completamente todos los periféricos disponibles para la plataforma y los recursos con los que cuenta. La especificación alcanza un nivel en el que existen diferentes firmwares según el tipo de aeronave a usar, y actualmente existen muchos otros en desarrollo para nuevos vehículos aéreos en uso.

Como ya se adelantó en páginas anteriores, el vehículo AirWhale no dispone de un firmware específico dentro del proyecto ArduPilot, como era de esperar. Por ello, urge la programación de un código útil capaz de llevar a cabo el cometido del Proyecto AirWhale. En este sentido, la línea de trabajo a seguir comienza en la revisión del código ArduCopter para una comprensión, en una primera aproximación, de las funciones que se usarán posteriormente en el código desarrollado para el AirWhale, obtenidas de las librerías desarrolladas por la comunidad.

El siguiente paso consistirá en la programación de un código eficaz en el vuelo controlado de la aeronave AirWhale, explicando detenidamente todas las tareas y funciones que llevará a cabo la plataforma ArduPilot y todo aquello que esté relacionado con el firmware desarrollado.

La dificultad de este trabajo radica en la alta encapsulación y modularidad del código, además de la falta de documentación o guía para poder entenderlo. Existen comentarios incluidos en el código, aportados por todos los colaboradores e integrantes de la comunidad, pero debido, precisamente, al alto número de programadores, se pueden encontrar a veces contradicciones o líneas de código pendientes de eliminar que llevan a 'callejones sin salida', ya que no existe una colaboración estrecha entre todos ellos. También hay que remarcar que el código está escrito en su totalidad en C++, lenguaje de programación no impartido en el grado. En este sentido, el alumno ha tenido que buscar personalmente conceptos pertenecientes a este lenguaje para entender el firmware lo suficiente para usarlo a su favor.

5.1 El firmware ArduCopter

El código ArduCopter es el firmware desarrollado por la comunidad del Proyecto ArduPilot que permite el vuelo, y su monitorización, de aeronaves que basan su funcionamiento, generalmente, en motores eléctricos con hélices que ejerzan la fuerza actuadora. Esta característica reúne vehículos aéreos como los quadrotors, hexarotors, trirotors, octarotors o helicópteros.

El firmware completo está compuesto por 630 archivos: 63 de ellos componen los archivos específicos del firmware ArduCopter y los archivos restantes componen la biblioteca común a todos los firmwares. Muchos de los archivos que componen la biblioteca no son de interés en este trabajo, definiendo los más importantes en las siguientes secciones. En el caso de los archivos específicos del firmware ArduCopter, la mayoría son de extensión .pde, siendo todos compilados en conjunto. Abarcan funciones de control, gestión de actuadores y sensores, labores de comunicación, selección de modos de vuelo y un largo etcétera de tareas que, como se verá luego, no son imprescindibles a la hora de adaptar el firmware al Proyecto AirWhale; la gran mayoría de funciones que aporta el firmware no se implementarán en el código del AirWhale, ya sea por inutilidad, por falta de tiempo o por alta complejidad.

En este sentido, sólo se explicarán de forma detallada las funciones útiles para el código a desarrollar, estudiando de forma general las que no se adoptarán. Para un trabajo posterior, podría plantearse el adaptar completamente el código ArduCopter a la aeronave AirWhale, manteniendo todas las funciones; en un sentido práctico, se ha desechado ese trabajo en este escrito. Tampoco se detallará el código a bajo nivel, es decir, la

programación relacionada con los pines del microprocesador.

La última versión estable del código ArduCopter a estudiar se puede descargar íntegramente desde el repositorio oficial del Proyecto ArduPilot, concretamente desde el enlace [23].

5.1.1 Librerías

Como se ha apuntado, existe un gran número de archivos que constituye la librería común a todos los firmwares. Este gran número es el resultado de la necesidad de cubrir multitud de tareas que aparecen en cada uno de los firmwares específicos y la definición de funciones necesarias para la compatibilidad con otras plataformas autopiloto. A continuación se enumerarán y detallarán brevemente cada una de ellas, dejando una explicación más profunda de las librerías útiles, para el código a construir, para secciones posteriores.

5.1.1.1 Definición general de librerías

Se presentarán las librerías según el nombre de la carpeta que las engloba:

- **AC_AttitudeControl**: Este conjunto de librerías introduce variables y funciones relacionadas con el control de la orientación y posición de la aeronave, sea cual sea el firmware. Esta relacionado directamente con los archivos 'control_X.pde' del firmware ArduCopter. Para el objetivo de este trabajo, no van a ser necesarias, ya que se programarán las leyes de control desarolladas por el equipo Automático, y por tanto tampoco se hará uso de los archivos 'control_X.pde'. También incluyen librerías específicas para el control de la orientación y la posición del helicóptero, el cual consta de particularidades lo suficientemente importantes como para el desarrollo de archivos específicos para suplirlas.
- **AC_Fence**: Una librería que define límites geométricos al vuelo de la aeronave, tanto en altitud como distancia al punto inicial o 'home'. Concretamente, se define una circunferencia límite en el plano XY en la cual la aeronave puede operar sin problemas, y valor de altitud límite. Cuando la aeronave supera estos límites, el firmware cambia el estado de la aeronave o la función a realizar en pro de la seguridad. Muy útil en caso de arrastre por causa del viento. No se usará debido a que no se realizarán pruebas en exteriores, y el vuelo será en un entorno controlado.
- **AC_PID**: Librerías que incluyen las funciones que definen los algoritmos de los controladores proporcionales (P) e integrales y derivativos (PID), y las particularizaciones para el helicóptero. No se usará, es preferible reprogramarlo.
- **AC_Sprayer**: Librería que se encarga del control del 'crop sprayer' o rociador de cultivos, si la aeronave lo tuviese. No se usará, por motivos obvios.
- **AC_WPNav**: Conjunto de librerías que contienen las funciones y variables necesarias para la correcta navegación de la aeronave a partir de balizas o 'waypoints'. También incluyen lo propio para una navegación en círculo. Aunque es interesante dotar a la aeronave de la posibilidad de la navegación por 'waypoints', en esta primera versión de código no se implementará. En líneas de trabajo posteriores se podría abordar esta tarea.
- **AP_ADC** y **AP_ADC_AnalogSource**: Estas librerías contienen el código específico para el uso de los convertidores analógico-digitales que pueda usar la placa Ardupilot. Se ha observado el uso de esta librería en el código particular ArduCopter y no se encuentran grandes usos. Parece ser necesario si se dispone de una placa APM 1, pero al poseer una plataforma APM 2.6, no se tendrá en cuenta.
- **AP_AHRS**: Estas librerías contienen los algoritmos necesarios para implementar el llamado 'Attitude and Heading Reference System' (AHRS). Este tipo de sistemas proporciona datos sobre la orientación, el rumbo y la posición de la aeronave. Esto se consigue gracias a la fusión de la medida de giróscopo, acelerómetro y magnetómetro (en algunos casos, también GPS) mediante la aplicación de algún tipo de

filtro. En el caso de esta librería, se usan dos filtros: el famoso Filtro de Kalman Extendido (EKF), y otro basado en el DCM (Direction Cosine Matrix). Según la documentación encontrada en la web oficial del Proyecto ArduPilot [24], el filtro de Kalman no puede aplicarse en ninguna de las versiones de la plataforma ArduPilot, por el momento, debido a su reducida capacidad de cálculo.

- **AP_Airspeed**: Aportan los archivos necesarios para la implementación de funciones y variables clave para el uso correcto y calibración de sensores de velocidad del aire alrededor de la aeronave, todo ello para distintas placas autopiloto. Dado que la aeronave a construir no operará a velocidades muy altas, se ha prescindido tanto del sensor como de las librerías asociadas.
- **AP_Arming**: Define algunas funciones de comunicación con la estación en tierra de verificación, permitiendo o no el armado final de la aeronave según la comprobación de aspectos relacionados con los sensores, radio y otros. En principio, no será necesario este conjunto de archivos.
- **AP_Baro**: Aporta las funciones necesarias para manejar los sensores barométricos compatibles con las distintas plataformas disponibles. Estas funciones incluyen tareas de inicialización del barómetro, obtención de datos proporcionados por el sensor (temperatura y presión), su calibración y demás acciones.
- **AP_BattMonitor**: De forma análoga a la anterior librería, este conjunto define las funciones para gestionar los datos obtenidos por el sensor que incluye el 3DR Power Module.
- **AP_BoardConfig**: Relativo a opciones varias sobre pines y variables relacionadas con las plataformas autopiloto PX4, PIXHAWK Y VBRAIN. No se detallará.
- **AP_Buffer**: Librería que aporta funciones para manejar filas o colas de datos, tales como la definición de la cola, adición y sustracción de datos, identificar el orden del dato o conocer el número de elementos de la fila.
- **AP_Camera**: Librería relacionada con las acciones realizables con la cámara que pueda añadirse a la aeronave. En una primera versión del firmware, no se incluirá.
- **AP_Common**: Definición de funciones comunes, usualmente utilizadas, como la conversión de grados a radianes, y viceversa; fijar la velocidad de transmisión de datos por el puerto serie y funciones para eliminar errores y avisos comunes del compilador. Las funciones de conversión de ángulos serán usadas con asiduidad, aunque sean fáciles de reprogramar. También incluye declaraciones relacionadas con testeos del firmware y compatibilidad Arduino-C++.
- **AP_Compass**: Incluye todas las funciones necesarias para la obtención y gestión de datos proporcionados por los modelos de magnetómetro admitidos por las placas disponibles. En el caso del AirWhale, se dispone de un magnetómetro junto al módulo de GPS, contemplado por la biblioteca.
- **AP_Curve**: Librería que aporta las funciones necesarias para definir la curva que relaciona los valores PWM con la fuerza de sustentación del motor relacionado (añadir/eliminar puntos a/de la curva, cálculo de los valores de PWM o de la fuerza de los motores por aproximación lineal…). Es usada frecuentemente por el firmware ArduCopter, y es clave en la definición de los motores y los PWM a mandar a cada uno**.** No se va a usar, ya que se realizará aparte un experimento para encontrar esa relación necesaria para el control óptimo de la aeronave.
- **AP_EPM**: Biblioteca que permite la gestión y uso adecuado del EPM_CargoGripper, un dispositivo magnetoelectrónico que ejerce un campo magnético para el agarre de cargas metálicas de peso máximo de un kilogramo [25]. Como este dispositivo no se usará, se prescinde de esta librería.
- **AP_Frsky_Telem**: Aporta variables y funciones que realizan acciones relacionadas con la telemetría conectada a la placa disponible, en su mayoría de comunicación.
- **AP_GPS**: Conjunto de bibliotecas que habilitan distintos modelos de GPS. Aunque se hará uso del GPS sólo porque el estimador que se usará para la obtención de los ángulos de navegación lo necesita, no se hará la lectura del mismo de forma directa. Las funciones que se usarán se definirán en su momento.
- **AP_HAL**: Uno de los conjuntos de librerías más importantes del firmware. Se encarga de la definición

de las variables y funciones relacionadas con la plataforma autopiloto en sí: lectura, definición y escritura en pines de entrada y salida, puertos SPI, I2C, almacenaje en memoria, gestión de tiempo de cómputo de distintas tareas, comunicación por puerto serie y consola del IDE y muchas más otras tareas. Ampliamente usada por el firmware y, evidentemente, indirectamente por el código creado para el AirWhale, pero no se entrará en detalles sobre las distintas librerías que se encuentran aquí debido a su gran extensión. Las funciones que aparezcan posteriormente relacionadas con estas bibliotecas serán definidas para su fácil comprensión.

- **AP_HAL_AVR, AP_HAL_AVR_SITL, AP_HAL_Empty, AP_HAL_FLYMAPLE, AP_HAL_Linux, AP_HAL_PX4, AP_HAL_VRBRAIN**: Todas estas bibliotecas se encargan de lo mismo que el conjunto AP_HAL, pero para distintas placas. En este sentido, no se explicarán. En conjunto, suponen el mayor número de archivos de librería de todos.
- **AP_InertialNav**: Al igual que la biblioteca AP_AHRS, este conjunto incluye algoritmos necesarios para obtener datos precisos sobre la posición y altitud de la aeronave, fusionando datos de los sensores acelerómetro, GPS y barómetro. Existen particularidades dependiendo del método de integración y fusión de sensores empleados (DCM o EKF).
- **AP_InertialSensor**: Funciones necesarias para gestionar correctamente el sensor acelerómetro y giróscopo integrados en la placa APM 2.6.
- **AP_L1_Control**: Biblioteca necesaria para el control adaptativo L1. No se entrará en detalles, ya que no es cometido de este trabajo.
- **AP_Limits**: Impone límites a la localización de la aeronave imponiendo 'geovallas', tal y como pudo verse en AP_Fence; y otros parámetros susceptibles a restricciones relacionados con el GPS o la altitud. No se usará.
- **AP_Math**: Bibliotecas que introducen funciones matemáticas comunes y otras más elaboradas, permitiendo el uso de vectores, matrices, operaciones de conversión de unidades, trigonometría, cambio de variables, geometría y muchas otras más tareas. No se entrará en más detalles, salvo cuando se deba detallar más adelante.
- **AP_Motors**: De gran importancia en el firmware, define los parámetros y aporta las funciones para la definición y gestión de los motores en el código. Contempla distintas disposiciones de los motores (quadrotor en X o en cruz, hexarotor…) e incluye algoritmos para la definición del valor de PWM que debe recibir cada motor según los valores que deberían tener las variables de estado de la aeronave.
- **AP_Mission**: Esta librería lee y escribe comandos relacionados con las misiones que se definen en Mission Planner, además de permitir el fácil acceso a parámetros relacionados con los 'waypoints'. Como se dijo anteriormente, se va a prescindir en esta primera versión del firmware de la operación de vuelo por balizas.
- **AP_Mount**: Librería para el control del montaje de la cámara u otro dispositivo con un montaje de dos o tres ejes móviles. No se detallará.
- **AP_NavEKF**: Definición de las variables y funciones que hacen posible la aplicación del filtro de Kalman extendido (Extended Kalman Filter). Debido a la cantidad de recursos a nivel de procesador que requiere la aplicación de este filtro, no puede utilizarse bajo la placa ArduPilot. Su programación está dirigida a la placa PX4/PIXHAWK.
- **AP_Navigation**: Declaración de funciones relacionadas con la navegación por 'waypoints', como la modificación de ángulos de orientación de la aeronave con respecto a las balizas, la aceleración lateral de la aeronave en movimiento a la baliza o la comunicación con las librerías AC_WPNav, actualizando el estado de controlador de navegación dados el anterior y próximo 'waypoint'. No se detallará en profundidad.
- **AP_Notify**: Gestión de la iluminación de los leds de la placa en función de la notificación que se quiera dar al usuario, además de la definición de variables de estado que propinan el aviso de distintos eventos (GPS mal medido, firmware armado, fallo del filtro de Kalman programado, activaciones del modo seguro de un sensor dado…).

- **AP_OpticalFlow**: Se encargan del funcionamiento adecuado del sensor de flujo óptico, si lo hubiese. Este sensor es capaz de medir la velocidad de movimiento de la aeronave comparando el movimiento de un fotograma captado en un instante con respecto al mismo fotograma en un instante justo anterior. Como no se usará, esta biblioteca no es de interés en este trabajo.
- **AP_Parachute**: Librería encargada del funcionamiento de un paracaídas, si la aeronave dispusiese de uno. No es el caso, por ello no se detallará más.
- **AP_Param**: Otro de los conjuntos más importantes. Se encarga de la gestión y definición de variables comúnmente y ampliamente usadas por el firmware. Son usadas grandes estructuras de datos para guardar las variables necesarias, y son susceptibles de ser parámetros de funciones de escritura, copia, búsqueda, modificación, obtención de información y demás tratamiento de variables. Debido a su gran extensión, no se entrará en más detalle, aunque las funciones a usar se basen en variables y funciones de estas bibliotecas.
- **AP_PerfMon**: Define funciones para la captura de tiempos de cómputo de funciones basándose en funciones más simples de captura de tiempos. Estos datos se guardan en una lista de variables temporales que serán manipuladas por otras funciones descritas. No se usará.
- **AP_ProgMem**: Observando los archivos, se puede comprobar que aporta funciones relacionadas con la captura del tamaño de variables de distintos tipos, funciones para conocer la longitud de cadenas de caracteres, funciones de copia de cadenas y demás funciones sencillas. Existen funciones definidas en esta biblioteca que aparecen en el envío de PWM a los motores (se indicará en su momento).
- **AP_Rally**: Incorpora al firmware la capacidad de crear puntos virtuales geométricos al que la aeronave acudiría en caso de que active su estado RTL (Return To Launch), normalmente por una decisión en pro de la seguridad. Es útil cuando la vuelta a la posición 'home' (acción por defecto en el modo RTL) resulta ser una operación comprometida.
- **AP_RangeFinder**: Bibliotecas que definen funciones para gestionar las medidas proporcionadas por los distintos modelos de sónar más usados o soportados por las placas que soportan el firmware. Interesante su uso, ya que se dispone, como se vio en el Capítulo 4, de un sensor sónar. Sin embargo, en una primera versión del firmware, no será objetivo principal el uso del sónar, y se pospone para versiones posteriores.
- **AP_RCMapper**: Necesaria para la conversión de las entradas de radiocontrol en empuje, par en roll, pitch y yaw. En un principio, como no se hace uso de un radiocontrol ni de un receptor en los pines de entrada, no se detallará.
- **AP_Relay**: Define el driver para los pines usados como pines 'relay': pines digitales que proporcionan 0 V o 5 V (en el caso de la placa APM). Según el código, un máximo de cuatro pines se pueden definir como 'relays', cualquiera de los pines A0-A8 de la placa. Puede ser útil para la alimentación de servomotores. No se usará en esta primera versión.
- **AP_Scheduler**: Se encarga de la gestión eficaz del tiempo del que dispone cada proceso para completarse y el periodo de repetición del mismo mediante un arcaico sistema de ejecutivo cíclico, ya que la placa APM 2.6 no soporta el 'threading'.
- **AP_ServoRelayEvents**: Relacionado con el anterior conjunto de librerías; define funciones para mandar señales a través de los pines de 'relay', concretando según el dispositivo conectado a ellos.
- **AP_SpdHgtControl**: Una interfaz genérica para todos los controladores de velocidad y altitud. Define funciones generales a ellos, y supone la clase común a todos. Luego, para cada controlador, existirá una clase específica basada en la definida en esta librería. Al usar controladores propios, no se detallará.
- **AP_TECS**: Librería bastante extensa que introduce las funciones necesarias para un control combinado de la altura y la velocidad, usando el empuje total para controlar la energía total y el ángulo 'pitch' para repartir esa energía entre potencial y cinética, según la prioridad. Supone una subclase de la definida en la anterior librería. No se detallará por los mismos motivos anteriores.
- **AP_Terrain**: Librería que aporta funcionalidades dirigidas a conocer el estado del terreno que se sobrevuela. Lleva a cabo tareas de reconocimiento de altitud en la posición actual sobre el terreno,

velocidad relativa, diferencia de parámetros del terreno con respecto al encontrado en 'home', almacenaje de datos y reconocimiento de parámetros relacionados con el terreno y con los 'waypoints' y puntos 'rally'.

- **AP_Vehicle**: Proporciona parámetros específicos para cada tipo de vehículo (en este caso, vehículos con ala o multicópteros), además de otras variables necesarias para la selección de directorios de compilación.
- **APM_Control**: Habilita la función 'AutoTune', que se encarga de actualizar cada 10 segundos las ganancias de los controladores PID usados para el control de la aeronave. Dispone de una clase general (AP_AutoTune) y clases más concretas relacionadas con cada variable a controlar (p.e: AP_YawController).
- **AP_OBC**: Se encarga de aportar funciones de seguridad, de tal manera que si la aeronave supera los límites geométricos establecidos por AP_Fence y AP_Limits, algún sensor crítico deja de funcionar correctamente (GPS, barómetro) o se pierde comunicación con la estación de tierra, se activa el modo seguro (Failsafe).
- **AP_PI**: Define algoritmos para la definición de un controlador PI.
- **Dataflash**: Librería que define funciones necesarias para operar con la placa de almacenamiento que incluye la plataforma ArduPilot. Muy útil para guardar datos que en el momento no puedan leerse, pudiendo extraerlos posteriormente. Recordar que la placa de almacenamiento dispone de un total de 4 MB, por lo que es posible obtener una gran cantidad de datos de vuelo sin requerir su lectura hasta el fin del mismo.
- **Filter**: Librerías donde se definen los distintos filtros que se usarán en las demás bibliotecas que la necesiten (en su mayoría, las relacionadas con la obtención de datos provenientes de los sensores). En total, reúne cuatro tipos de filtros: derivativo, paso bajo, paso bajo de segundo orden y filtro de Butterworth.
- **GCS_Console** y **GCS_MAVLink**: Librerías que se encargan de la comunicación efectiva con la estación de tierra y el control mediante la misma, todo ello bajo el protocolo MAVLink.
- **RC_Channel**: Conjunto de bibliotecas que se dedican a gestionar todo lo relacionado con los puertos de entrada y salida, es decir: los relacionados con la entrada de datos por el receptor de radio y la salida de PWM dirigida a los motores. Contienen funciones limitadoras de los valores a enviar, definiciones de pines, etc.
- **SITL**: Introduce el sistema llamado 'Software In The Loop', que permite gestionar eficientemente los datos relacionados con la simulación de sensores por parte del simulador que esté realizándolo. Unido estrechamente con el sistema HIL.
- **StorageManager**: Gestión de la memoria de la placa, que incluye definir los sectores de la misma para el número de 'waypoints', de parámetros, de puntos 'rally' y de puntos que definan los límites geométricos.

5.1.1.2 AP_Baro

El sensor barométrico es imprescindible para conocer la altura a la que se encuentra la aeronave, ya que utiliza la medida realizada de presión y temperatura para calcular datos como la altitud o, indirectamente, la densidad del aire. El sensor a utilizar es el incluido en la plataforma ArduPilot, el MS5611-01BA03 de Measurements Specialties, y el fabricante ofrece documentación suficiente para la conexión y tratamiento de dicho sensor [26]. En la biblioteca referente al barómetro, se puede encontrar una librería específica para el conexionado y correcta gestión del sensor MS5611, y se usará de forma íntegra, definiendo detalladamente las funciones que aparezcan explícitamente en el código a desarrollar.

AP_Baro.h/.cpp: Descripción y funcionamiento general

El conjunto incluye una librería general llamada AP_Baro.h, y su archivo .cpp asociado, donde se define una clase pública general llamada AP_Baro que incluye funciones relacionadas con la calibración, cálculo de altitud, velocidad relativa en Z y otros. Estas funciones usarán otras definidas en las sucesivas subclases que aparecen en las librerías específicas para cada modelo de barómetro (se verá a continuación). En ningún momento, esta librería general define cómo obtener medidas del sensor, esta tarea se detalla en las librerías específicas, ya que depende de cada modelo.

AP_Baro.h/.cpp: Funciones

Las funciones que aparecen en AP_Baro.h/.cpp son:

- `void calibrate (void)`: Como su nombre indica, se encarga de la calibración correcta del sensor de presión. El funcionamiento es sencillo: se toma una medida inicial de la presión y la temperatura (valiéndose de las funciones definidas en las librerías específicas), que se guardarán como valores correspondientes al 'suelo virtual' para el firmware.

  Después, se mantiene el sensor tomando medidas inútiles durante un segundo. El motivo es que, precisamente, el modelo MS5611 toma medidas con un error de aproximadamente un metro durante el primer segundo de funcionamiento.

  Por último, se toman cinco medidas para asegurar una buena calibración, realizando la siguiente ponderación después de cada una:

  $$valor_{suelo} = (valor_{suelo} \cdot 0.8) + (valor_{medido} \cdot 0.2)$$

  El valor final del calibrado se obtendrá después de la quinta medida, finalizando la función.

- `void update_calibrate (void)`: Ignora el proceso anterior y guarda como valores de 'suelo' los obtenidos en ese momento, lo cual resulta útil para mantener actualizada la calibración del dispositivo.

- `float get_altitude_difference (float, float) const`: Esta función constante (no puede modificar variables) realiza el cálculo de la diferencia entre la altitud en función de una presión dada (segundo parámetro) y la altitud en función de una presión base (primer parámetro), y devuelve como una variable *float*. Es la función que se usará mayoritariamente para calcular la altitud de la aeronave, teniendo como segundo parámetro la presión medida al momento y como primero la presión guardada a nivel de 'suelo'.

  En ella, se realizan varios cálculos, dependiendo de la plataforma que se disponga. En el caso de la plataforma APM 2.6 (velocidad de 16 MHz), el cálculo realizado es el siguiente:

  $$altitud = ln\left(\frac{presion_{base}}{presion_{medida}}\right) \cdot temperatura \cdot 29.271269$$

  Buscando, se ha identificado este cálculo con el cálculo de la presión a partir de la altura y la temperatura suponiendo atmósfera estándar, dado por la ecuación (1):

  $$p = p_0 \cdot e^{-\frac{g}{RT} \cdot (h - h_0)} \quad (1)$$

  Donde p y p_0 son las presiones medida y base, respectivamente, g es la gravedad, R es la constante universal de los gases expresada en $\frac{m^2}{s^2 \cdot K}$, T es la temperatura medida y h y h_0 son la altura a calcular y la altura inicial (nula, considerada a nivel de suelo), respectivamente [27].

  Esta función devolverá cero si no ha habido una calibración previa.

- `float get_altitude (void)`: Realiza automáticamente la medición de un valor de presión en el instante en el que se ha usado, llama a la función *get_altitute_difference*, pasándole como parámetros el valor medido y el valor de presión base; y devuelve el valor de la altitud calculada.

  También realiza funciones de comprobación, como, por ejemplo, si el valor para la presión base es distinto de cero (en caso contrario, la función se habría llamado sin una calibración previa) o si realmente existe una nueva medida a tratar. Para esta última comprobación, se utilizan dos variables temporales que van guardando el momento actual de lectura de un valor del sensor y el anterior; si los valores de estas dos variables coinciden, conlleva que no ha existido lectura (no se ha actualizado la variable) y la función acaba.

Existen otras dos funciones relacionadas con el cálculo de la velocidad estimada, a través de la variación de la presión teniendo en cuenta las últimas medidas empleando un filtro derivativo, pero no se usarán y, por tanto, tampoco se entrará en más detalle.

AP_Baro_MS5611.h/.cpp: Descripción y funciones

Esta librería, subordinada a la anterior, define las funciones necesarias para gestionar de forma correcta el modelo de sensor barómetrico del que se dispone, el MS5611. Todas las funciones realizan tareas a bajo nivel, por lo que no se entrará en mucho detalle; se proporcionará una descripción general de las más trascendentes, a modo de introducción y con el objetivo de entender qué tarea desempeña cada función.

Estas funciones son:

- `void init (void)`: Habilita el puerto serie y lo asigna al barómetro. Esto puede llevar a problemas, ya que el acelerómetro y el barómetro comparten puerto SPI. Para inicializarlo, se debe asignar el puerto a los dos de forma alternada exteriormente a la función, como se comprobará en el código desarrollado por el alumno. Para la gestión conjunta, se le asigna al puerto SPI un semáforo. Habilitado y asignado el puerto SPI, se le asigna una velocidad de 8 MHz.

  Posteriormente, se le envía un comando al barómetro para que se reinicie, y posteriormente se obtiene la calibración del sensor aportada por el fabricante. Se inicia el proceso para obtener una medida, y si esa medida se tarda en obtener más de un segundo, se considera que el barómetro tiene problemas y se avisa al usuario sobre ello.

- `void _update (void)`: Lee los datos extraidos por el sensor, y la función es llamada, de media, cada 0.0075 segundos (no se permite que el proceso de medida dure más de 0.01 segundos). Para realizar esta tarea, se utiliza una máquina de estados de cinco estados: en el primero se lee la temperatura y en los cuatro posteriores se lee la presión; con cada llamada a la función el estado aumenta en una unidad. La razón es que la temperatura no necesita leerse tantas veces, ya que varía menos que la presión. Si no se llegan a tratar los datos, por la correspondiente función de tratamiento, antes de que exista un número determinado de datos acumulados (en el caso de la temperatura 32 valores, y en el de la presión 128) se perderán datos acumulados. En el momento que existe un dato acumulado, se avisa al sistema mediante un 'flag' de que hay medidas disponibles para tratar.

- `void read (void)`: Función que trata los datos obtenidos del sensor. Comprueba si la anterior función ha activado el 'flag' que indica que existen datos acumulados y, en caso afirmativo, captura esos datos y los reinicia para su uso en la función *_update*. Si los datos acumulados no son nulos (comprobación por seguridad), realiza una media de los datos acumulados, destinada a tratar posteriormente para obtener la presión y la temperatura en sus unidades correspondientes. Esta labor la lleva a cabo la función *_calculate*.

- `void _calculate (void)`: Función que calcula la presión en pascales y la temperatura en grados Celsius. Las operaciones no se detallarán en este escrito; para encontrar el detallado procedimiento de cálculo se puede acudir al 'datasheet' ofrecido por el fabricante [33]. Terminados los cálculos, la función almacena los resultados en dos variables: 'temp' y 'press', temperatura y presión, respectivamente.

- `float get_pressure (void)`: Esta función es usada frecuentemente por la librería AP_Baro.h/.cpp. Es, por denominarlo de forma sencilla, la interfaz entre el tratamiento a bajo nivel del sensor de presión

y el tratamiento del mismo a alto nivel. Esta función sólo realiza una tarea: devolver la variable 'press'.

- `float get_temperature (void)`: Esta función, al igual que la anterior, es usada activamente por la librería superior. Realiza la misma función, pero relativa a la temperatura: devolver la variable 'temp'.

5.1.1.3 AP_InertialSensor

El conjunto de bibliotecas asociado a las acciones relacionadas con el acelerómetro y el giróscopo. Al igual que se expuso en los anteriores sensores, esta biblioteca contiene numerosos archivos destinados a definir el modo de actuación con respecto a cada uno de los modelos soportados por las plataformas que pueden cargar el firmware ArduCopter. En el caso de la APM 2.6, se dispone del popular acelerómetro/giróscopo MPU6000. A continuación se realizará un estudio de la librería general de este conjunto, AP_InertialSensor.h/.cpp, con descripciones de las funciones más relevantes.

Funcionamiento general

El funcionamiento aportado por la librería es bastante confuso. En una primera aproximación, podríamos definir tres niveles de tareas:

- Tareas generales a todos los modelos de acelerómetros soportados, incluidas en AP_InertialSensor.h/.cpp.
- Tareas específicas de cada modelo, definidas en sus respectivas bibliotecas (por ejemplo, AP_InertialSensor_MPU6000.h/.cpp).
- Una librería que trabaja específicamente como nexo de unión entre el primer y segundo nivel, definiendo variables asociadas a cada sensor, llamada AP_InertialSensor_Backend.h/.cpp. Es la que proporciona directamente las medidas al primer nivel.

Así, las tareas específicas de cada modelo de sensor suponen una clase subordinada a la definida en AP_InertialSensor_Backend, por lo que es esta biblioteca la que trata con el primer nivel. A continuación se realizará un estudio de la librería general de este conjunto, AP_InertialSensor.h/.cpp.

AP_InertialSensor.h/.cpp: Funciones

Las funciones que aparecen en AP_InertialSensor.h/.cpp son, entre otras:

- `void init (style, simple rate)`: Inicia el sensor. Los parámetros que se le pasan son la manera de iniciar el sensor (primer parámetro) y el periodo de muestreo del sensor (segundo parámetro). El primer parámetro puede ser WARM_START, que supone la no calibración del acelerómetro y del giróscopo, o COLD_START, que supone la calibración del giróscopo por parte de la función.

  La función realiza, además, la asociación de una clase definida en AP_InertialSensor_Backend al sensor inicializado, que permite formalizar ese nexo de unión con el bajo nivel. También reinicia variables temporales útiles en las demás funciones, escalas a aplicar a las medidas obtenidas por el sensor y otras tareas.

- `void add_backend (AP_InertialSensor_Backend)`: Registra la definición de una clase puente entre alto y bajo nivel (parámetro), definida en AP_InertialSensor_Backend. Si se realiza con éxito, aumenta un contador de este tipo de clases en uno, existiendo un valor límite.

- `void detect_backend (void)`: Además de realizar la tarea especificada por *add_backend*, detecta qué modelo de sensor se está tratando y asocia la clase puente con la biblioteca de dicho modelo, cerrando la conexión entre bajo y alto nivel completamente.

- `void init_accel (void)`: Inicializa el acelerómetro calibrado. Para ello, realiza los siguientes pasos:

  1. Se elimina el calibrado anterior, si lo hubiese.
  2. Calibra el sensor con valores 'offset' muy altos, de tal manera que obliga al algoritmo tomar

varias medidas para obtener un buen calibrado.

3. Comienza un bucle de calibración, donde se realizarán iteraciones hasta que se obtenga una calibración aceptable. Los siguientes pasos se realizarán dentro de una iteración.
4. Se toman medidas del sensor (función *update*) y se guarda como 'offset_nuevo', previo guardado del anterior como 'offset_antiguo'.
5. A continuación, se tomarán cincuenta medidas para ir corrigiendo el 'offset_nuevo', realizando después de cada medición el siguiente cálculo:

$$medida_{antigua} = \left(medida_{antigua} \cdot 0.9\right) + \left(medida_{nueva} \cdot 0.2\right)$$

El resultado de este cálculo será el nuevo 'offset_nuevo' candidato a adoptarse.

6. Se compara el resultado con el 'offset_antiguo', realizando las siguientes comprobaciones:

$$cambio\ total = \left|\mathrm{offset}_{\mathrm{antiguo}}.\mathrm{x} - \mathrm{offset}_{\mathrm{nuevo}}.\mathrm{x}\right| + \left|\mathrm{offset}_{\mathrm{antiguo}}.\mathrm{y} - \mathrm{offset}_{\mathrm{nuevo}}.\mathrm{y}\right| + \left|\mathrm{offset}_{\mathrm{antiguo}}.\mathrm{z} - \mathrm{offset}_{\mathrm{nuevo}}.\mathrm{z}\right| < 4$$

$$offset\ maximo = máx\left(\left|\mathrm{offset}_{\mathrm{antiguo}}.\mathrm{x} - \mathrm{offset}_{\mathrm{nuevo}}.\mathrm{x}\right|, \left|\mathrm{offset}_{\mathrm{antiguo}}.\mathrm{x} - \mathrm{offset}_{\mathrm{nuevo}}.\mathrm{x}\right|, \left|\mathrm{offset}_{\mathrm{antiguo}}.\mathrm{x} - \mathrm{offset}_{\mathrm{nuevo}}.\mathrm{x}\right|\right) < 250$$

Es decir, si el cambio total de medidas en cada eje entre una medida y otra es menor que 4, y el mayor cambio registrado entre medidas, sea cual sea su dirección cartesiana, es menor que 250, se da por buena la calibración y se guardan estos valores como 'offset'. Si no se cumpliesen estas desigualdades, el algoritmo vuelve al paso 4.

- `void init_gyro (void)`: Inicializa, al igual que la anterior, el giróscopo. Realiza distintas operaciones, pero la idea puede considerarse semejante a la vista en la función *init_accel*. No se va a detallar.
- `void update (void)`: Se comunica con la clase asociada de tipo AP_InertialSensor_Backend para que realice las gestiones y cálculos concretos necesarios para obtener una medida de aceleración lineal y velocidad angular. Concretamente, la función esperará al siguiente instante de muestreo y ejecutará la función *update*, definida en AP_InertialSensor_Backend y, indirectamente, en cada una de las librerías asociadas a cada modelo de sensor.

  Una vez realizado esto, y otras tareas, avisa al sistema de que ya se ha iniciado el proceso para tomar las medidas.
- `Vector3f get_accel (void)`: Devuelve el vector de aceleraciones lineales actualizado, el cual está declarado globalmente y se modifica automáticamente con el uso de las demás funciones. Esta función SÓLO devuelve el vector.
- `Vector3f get_gyro (void)`: Devuelve el vector de velocidades angulares actualizado, el cual está declarado globalmente y se modifica automáticamente con el uso de las demás funciones. Esta función SÓLO devuelve el vector.

AP_InertialSensor_Backend.h/.cpp: Descripción y funciones

En esta librería se define la clase AP_InertialSensor_Backend, que añade la interfaz de variable y funciones necesarias entre alto y bajo nivel. Esta clase recibe como parámetro la clase definida en AP_InertialSensor, recibiendo sus variables para poder modificarlas. Las funciones definidas en esta librería más importantes son:

- `void rotate_and_offset_gyro (uint8_t, const Vector3f)`: Copia las medidas obtenidas por el giróscopo (segundo argumento) en la variable '_gyro' del sensor correspondiente (primer argumento), definida en AP_InertialSensor; y la trata, orientando correctamente el vector de velocidades angulares y restándole los valores 'offset'. Esta función es llamada en la librería AP_InertialSensor_MPU6000.h/.cpp.
- `void rotate_and_offset_accel (uint8_t, const Vector3f)`: Ídem a *rotate_and_offset_gyro*,

pero con las aceleraciones lineales. Además de los tratamientos mencionados, también aplica un escalado a las medidas. También se la llama en AP_InertialSensor_MPU6000.h/.cpp.

- `void set_accel_error_count (uint8_t, uint32_t)`: Copia el error obtenido en las medidas (segundo parámetro), del acelerómetro, en la variable correspondiente definida en AP_InertialSensor.
- `void set_gyro_error_count (uint8_t, uint32_t)`: Ídem a *set_accel_error_count*, pero con el error en las medidas del giróscopo.

AP_InertialSensor_MPU6000.h/.cpp: Descripción y funciones

Constituye la gestión del sensor a bajo nivel: definición de puertos, manejo de registros… y obtención directa de las medidas provenientes del MPU6000. Para ello, define una clase subordinada a la definida en la librería anterior: AP_InertialSensor_Backend. Así, cuando se llama desde alto nivel, por ejemplo, a la función *update*, indirectamente se llama a la función *update* definida en esta librería.

Algunas de las funciones que se pueden encontrar en AP_InertialSensor_MPU6000.h/.cpp son:

- `bool init_sensor (void)`: Inicia el sensor a bajo nivel. Esto conlleva la inicialización del puerto SPI, la obtención de semáforos para la correcta gestión del mismo y la comprobación de que el sensor se ha iniciado correctamente, comprobando si se obtienen medidas por el puerto. También permite el inicio del muestreo continuo de medidas según el periodo de muestreo definido. Devuelve '*true*' si la inicialización se ha realizado con éxito.
- `void poll_data (void)`: Esta función se ejecutará con cada periodo de muestreo, y es la que se encarga de tomar medidas por el sensor. Las medidas se guardan en variables tratadas, luego, por la función *update*.
- `bool update (void)`: Cuando es llamada, trata las variables obtenidas por la función *poll_data* para pasarlas a alto nivel, usando las funciones *rotate_and_offset_gyro* y *rotate_and_offset_accel*.

5.1.1.4 AP_AHRS

Como se adelantaba anteriormente, esta biblioteca contiene el sistema que fusiona los datos de los tres sensores principales (acelerómetro, giróscopo y magnetómetro), filtrándolos para obtener la orientación en tiempo real de la aeronave (ángulos roll, pitch y yaw). Este filtrado está disponible en dos versiones:

- Filtrado a través del Filtro de Kalman Extendido: Un filtrado eficaz capaz de proporcionar datos provenientes de los sensores ponderándolos con los datos que se deberían obtener mediante el modelo dinámico del sistema. El algoritmo de este filtro es bastante elaborado, y supone operaciones con matrices que, en caso de contar con multitud de variables de estado, implican una gran capacidad de cálculo. Su programación y uso está dirigida exclusivamente a placas con una capacidad de cómputo elevada, como PX4/PIXHAWK. Desafortunadamente, la plataforma APM 2.6 no dispone de capacidad suficiente como para poder implementar este filtro, por lo que no se ha hecho uso del mismo.
- Filtrado DCM: Este tipo de filtrado, no muy conocido, es posible implementarlo en la plataforma APM 2.6. El funcionamiento básico se muestra en la Figura 5.1.

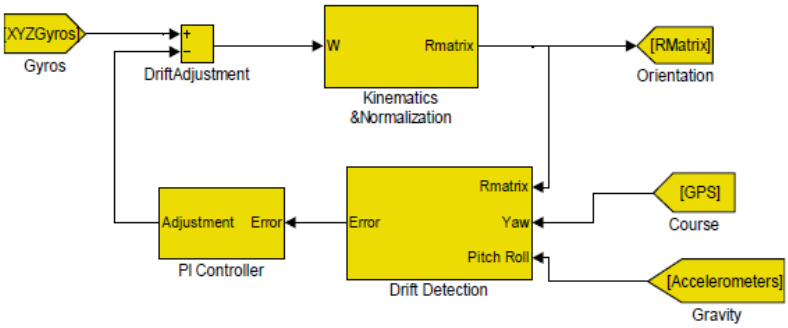

Figura 5.1. Esquema principal del filtrado DCM [28]

El filtro parte de los datos obtenidos de las velocidades angulares a través del giróscopo, dando un valor a la matriz de giro en coordenadas Euler para el instante actual en función de su valor en el anterior instante de muestreo y en función de las velocidades angulares adquiridas. Al integrar esta matriz, se obtienen los ángulos de orientación, los cuales serán contrastados con el GPS para el ángulo sobre el eje Z y el acelerómetro para los ángulos sobre X e Y. El error relativo entre las medidas se introduce en un controlador PI, que dará como salida una corrección del vector de velocidades angulares que se añadirá a las medidas tomadas del giróscopo para la siguiente iteración. En siguientes secciones se expondrán experimentos realizados para comprobar las medidas de ángulos obtenidas por el filtro DCM.

No se entrará en mayor detalle, ya que no es el cometido de este trabajo saber en qué consiste este filtro.

AP_AHRS.h/.cpp: Descripción y funciones utilizadas

La librería principal AP_AHRS se basa en la definición de una clase general, que abarca, con la definición de variables y funciones, las subclases asociadas a cada biblioteca individual de filtrado (AP_AHRS_DCM y AP_AHRS_NavEFK), como se ha visto, por ejemplo, con las librerías propias del acelerómetro/giróscopo. Esta clase aporta funciones de uso permitido por el usuario, que resultan ser un enlace con las funciones análogas que aparecen en las subclases. No se entrará en tantos detalles como se hizo con las anteriores bibliotecas, ya que, en su mayoría, las funciones cubren cálculos y algoritmos relacionados con el filtro.

A continuación se nombran las funciones usadas relacionadas, todas incluidas en AP_AHRS.h/.cpp, explicando, así, las análogas en la AP_AHRS_DCM.h/.cpp:

- `void init (void)`: Inicializa el algoritmo, estableciendo una orientación inicial de la aeronave a partir de unas medidas iniciales del acelerómetro/giróscopo.
- `void update (void)`: Esta función realiza, en su totalidad, el algoritmo que implementa el filtro, en este caso el filtrado DCM. Concretando para el DCM, utiliza otras funciones definidas en la librería asociada al filtro, tales como *matrix_update* (integración de la matriz de giro), *normalize* (normalizar la matriz de giro una vez integrada), *drift_correction* (realiza la comprobación del error con respecto a los datos aportados por GPS y acelerómetro) o *euler_angles* (cálculo de los ángulos de Euler, obteniendo la orientación de la aeronave).

5.1.1.5 AP_Motors

La librería AP_Motors no supone una gran dificultad añadida a la revisión del firmware si se piensa en qué se emplea finalmente de dicha librería; sin embargo, se encontró una dificultad con respecto a los motores en los primeros acercamientos al código ArduPilot.

Debido a la complejidad de gestión de los mismos en el código ArduCopter, una de las primera ideas fue modificar la librería AP_Motors para conseguir una configuración parecida a la aeronave AirWhale,

conservando las funcionalidades de control incluidas en el firmware y, por ende, el procedimiento de envío de valores PWM a los motores, junto a todas las demás tareas (comunicación, modos de vuelo, interfaz en programa Mission Planner o sistema HIL). Esta labor, en un principio, se intentó, realizando modificaciones sin ninguna guía o consejos sobre qué habría que cambiar o qué elementos se deberían conservar para conseguir el objetivo propuesto. Tuvo que ser un usuario de la comunidad ArduPilot quien paró los pies al alumno, y le aconsejó que abandonase la idea, por altamente compleja. La misma tarea, con distintos modelos de aeronaves, ya había sido realizada por usuarios anteriormente, y tardaron meses en finalizarla, teniendo estos un mayor conocimiento sobre el código ArduCopter. Esto ocurrió aproximadamente a principios del mes de julio.

Ante esta contrariedad, el alumno probó a seguir los mismos pasos que se llevó a cabo con los sensores: ir a la biblioteca relacionada con los motores y buscar qué era necesario para poder controlar los motores. Para ello, se hace una suposición: la aeronave tiene una configuración de quadrotor. De primeras, podría pensarse que esto afectará negativamente al vuelo del AirWhale, ya que, a nivel de código ArduCopter, suponer una configuración de quadrotor implica numerosas sentencias y particularidades en el control, la definición de PWM, el tratamiento de sensores…. Sin embargo, ya que en este trabajo no se usará la arquitectura ArduCopter, suponer una configuración quadrotor es, incluso, positivo. Rechazando controladores, métodos de cálculo PWM y demás sentencias que dependen directamente de la configuración escogida, en el código desarrollado sólo realiza una definición de canales de salida (los pines a los que se conectarán los motores) y habilita el envío PWM. Dicho de otra manera, sólo le comunica a la plataforma APM 2.6 que cuenta con cuatro motores que gestionar.

En secciones posteriores, podrá comprobarse cómo se usan las funciones asociadas a AP_Motors.

AP_Motors.h/.cpp: Descripción y funciones utilizadas

El funcionamiento de la librería, como en las anteriores, está distribuido por niveles o capas:

- AP_Motors_Class: Primer nivel de funcionamiento. Se encarga de gestionar las variables y funciones comunes a todas las clases subordinadas a ella, de tal manera que supone un nexo de unión con las funciones definidas en el segundo y tercer nivel. Aquí, aparecen variables de limitación de valores a enviar a los motores, funciones dedicadas a la asociación de los motores con pines de salida determinados, velocidad de envío de datos por los pines asociados a cada motor…
- AP_Motors_Matrix: Segundo nivel de funcionamiento, cuya clase es subordinada al primer nivel. Se encarga de incluir las funciones propias de envío, de inicialización de los motores, del añadido de motores al firmware con sus respectivos parámetros funcionales…
- AP_Motors_Quad/Octa/Hexa…: El tercer nivel. Es el menos extenso, ya que se dedica a añadir motores al firmware según parámetros geométricos y de contribución al cambio de las variables de orientación y el empuje. Estos motores se añaden según el tipo de configuración de aeronave del que se disponga (quadrotor, hexarotor…).

Resumiendo, el primer nivel añade parámetros y funciones necesarias para definir los datos más generales al control de motores, el segundo nivel es más concreto, aportando funciones de envío de PWM a motores y tareas asociadas, y el tercer nivel se encarga de definir los motores a usar para cada tipo de aeronave.

El control de los motores es bastante complejo y poco intuitivo, ya que este depende de parámetros relacionados con su aportación a la variación de las variables de estado. Es decir, si se escoge una configuración de quadrotor, se tendrá un control de PWM sobre motor muy eficiente, asociado a los archivos de control de la aeronave, pero con una modularidad muy baja; los parámetros de control aparecen en gran cantidad de líneas de código, siendo muy difícil adaptar esta librería al AirWhale sin tener antes que modificar gran parte de la biblioteca AP_Motors.

Esto llevó a revisar la librería para encontrar la función ejecutora del envío de PWM a los motores: la función *output_armed*. En ella, se realiza un gran número de tareas previas al envío del valor final de tensión a los motores conectados a la plataforma ArduPilot. Al término de dicha función, aparece la función buscada: ***hal.rcout->write(channel, PWM)***. Alojada en la librería *RCOutput.h* (incluida en AP_HAL), esta función envía al pin de salida '*channel*' el valor '*PWM*', estando este, según las indicaciones escritas en el código, entre 1000 y 2000. Esta función se usará directamente, sin intermediarios, de tal manera que se controlará en todo momento el valor PWM a enviar a cada motor.

Las otras funciones útiles, incluidas en la librería AP_Motors (AP_Motors_Class, AP_Motors_Matrix y AP_Motors_Quad) son:

- `void Init (void)`: Función incluida en AP_Motors_Class y AP_Motors_Matriz, inicializa los motores, definiendo una curva de relación PWM-fuerza de los motores, añadiendo los motores según la función *setup_motors* (incluida en AP_Motors_Quad) y estableciendo una velocidad de comunicación por los pines de salida con la función *set_update_rate*.
- `void enable (void)`: Sencillamente permite la comunicación a través de los pines de salida definidos para cada motor (usando la función ***hal.rcout->enable_ch(channel)***).

Para declarar los pines que se usarán para mandar los valores PWM a cada motor, se hará uso de la clase definida en la biblioteca *RC_Channel.h*, RC_Channel, la cual incluye las variables y funciones para definir como pines de salida el número que se le pasa como parámetro. En ese sentido, si se desea disponer de cuatro pines de salida para controlar cuatro motores, se puede realizar la declaración de cuatro clases RC_Channel: rc1(0), rc2(1), rc3(2) y rc4(3). Este método descrito será el usado en el código desarrollado.

Por su parte, la clase asociada a AP_Motors recibe como parámetros los canales de salida de los motores. Por ello, en el código creado se declara una clase AP_Motors_Quad a la que se le pasa las cuatro clases RC_Channel anteriormente nombradas. Esto habilita, a través de las funciones definidas antes, el control PWM correcto de los motores.

5.1.1.6 AP_Scheduler

Una de las bibliotecas más importantes del firmware, y no de las más extensas, precisamente. La gestión de las tareas, su duración y su periodo de activación son gestionados por esta librería. Por así decirlo, es el organizador de los distintos trabajadores que se encuentran en el firmware: les dice cuándo deben actuar y qué tiempo tienen para realizar la tarea que les ha sido encomendada.

En el firmware ArduCopter es totalmente imprescindible la actuación de este organizador de tiempo pseudo-real, debido a la alta cantidad de tareas que deben realizarse en un corto periodo de tiempo: lectura de sensores, comunicación a través de radio/puerto serie, gestión de los actuadores, actualización de controladores… y muchas otras que, de forma introductoria, se han definido junto a su conjunto de librerías correspondiente. En el caso del código desarrollado, se pensó en usar para el código a desarrollar, pero, por falta de tiempo, tuvo que desecharse la idea. Por suerte, no es imprescindible su actuación, pero, de cara a una posible mejora del mismo y de la creación de futuras versiones, se ha visto apropiado explicar su librería asociada.

Así, como se ha ido haciendo con las demás bibliotecas usadas, se hará una revisión detallada de la librería AP_Scheduler.h/.cpp.

AP_Scheduler.h/.cpp: Descripción y funcionamiento general

La librería AP_Scheduler aporta las variables y funciones necesarias para definir un administrador de tareas a realizar en el firmware mediante un sistema de ejecutivo cíclico. El motivo es que la plataforma ArduPilot no soporta 'threading', al contrario que otras plataformas que incluyen compatibilidad con 'threading' POSIX (PX4, por ejemplo) [29]. La biblioteca define una clase donde se recogen las funciones y variables útiles para gestionar este arcaico sistema de ejecutivo cíclico.

Entre estos elementos destaca la estructura ***struct Task***, la cual está compuesta de tres elementos:

- Una variable que guarda el nombre de la función o tarea a ejecutar.
- Una variable que guarda el periodo, en unidades de 10 ms cada una, de ejecución de la tarea. Por ejemplo, si esta variable vale 200, significa que la tarea se ejecuta cada dos segundos.
- Una variable que guarda el tiempo que se espera que tarde esta tarea en ejecutarse, en microsegundos.

En el archivo principal del firmware ArduCopter (ArduCopter.pde) se define un vector compuesto por un número determinado de estas estructuras. El número, evidentemente, coincide con el número de tareas que se

desea gestionar con el ejecutivo cíclico, suponiendo el vector una especie de lista de las tareas incluidas en el firmware y los datos temporales relacionados con las mismas. Es el procedimiento recomendado para el funcionamiento del administrador de tareas, ya que las funciones que se van a explicar a continuación cuentan con la definición de un vector de estructuras ***Task***. Concretamente, las variables temporales guardadas en ***Task*** serán de suma importancia en el cálculo de tiempos restantes, tiempos disponibles, orden en la ejecución de las tareas…

El ejecutivo cíclico funcionará mediante la actualización de un contador llamado 'tick', que marcará los intervalos de tiempo máximos para ejecutar las tareas. Cuando este contador llega a un valor igual al periodo de alguna tarea, en principio, debe dar paso a su código. En la explicación de las funciones se observará qué tipo de restricciones temporales se pueden encontrar.

AP_Scheduler.h/.cpp: Funciones

Las funciones que pueden revisarse en la librería AP_Scheduler.h/.cpp son:

- `void init (const AP_Scheduler::Task, uint8_t)`: Pone a punto el ejecutivo cíclico para su funcionamiento. El primer parámetro que recibe la función es una variable de tipo ***const struct Task*** que representa la/s tarea/s que debe gestionar el administrador. Este parámetro, como se ha comentado anteriormente, suele ser un vector que guarda un número determinado de tareas. El segundo parámetro representa el número de tareas incluido en el primer parámetro.

  La librería incluye ciertas variables propias, tales como una estructura tipo ***Task*** para realizar una copia de seguridad de la lista de tareas recibidas u otra para hacer lo propio con el segundo parámetro de la función *init*. En concreto, esta función actualiza variables con el primer y segundo parámetro que le llega a la función.

  Pondrá, además, el contador 'tick' a cero y creará un vector de tamaño igual al número de tareas que se utilizará para guardar en qué momento (en qué valor de tick) se inicia una tarea. El motivo se conocerá en breve.

- `void tick (void)`: Simplemente aumenta en uno el contador 'tick'. Es útil mantener esta sentencia en una función aparte, ya que es una tarea muy recurrida al ser demandada cada vez que la función *run*, explicada a continuación, acaba.

- `void run (uint16_t) const`: Ejecuta una o varias tareas antes de que el tiempo pasado como argumento (*uint16_t*) acabe. Es decir, el parámetro indica de cuánto tiempo dispone el ejecutivo cíclico para realizar el mayor número de tareas posible antes de que se vuelva a llamar a la función *run*. Este tiempo depende de la zona del código donde se esté llamando a esta función. En el código ArduCopter, esta llamada está insertada en un bucle que itera cada 100 Hz (0.01 s), por lo que el tiempo disponible que se tiene, como máximo, disponible para ejecutar las tareas, es de 100 Hz. Este tiempo viene impuesto por el tiempo de refresco del acelerómetro, que supone la tarea más rápida de todas. Normalmente el periodo de ejecución de *run* se hace coincidir con el periodo de ejecución de la tarea más rápida.

  Por ello, esta función debe revisar cuántos microsegundos conlleva realizar una y otra tarea, y si es posible ejecutarla atendiendo al tiempo disponible, se lleva a cabo. En este sentido, la función realiza los siguientes pasos:

  1. Guarda el instante en el que empieza a ejecutarse la función *run*.
  2. Inicia un bucle *for* con un número de iteraciones igual al número de tareas a gestionar. Los pasos que se describirán a continuación serán realizados con los datos de la tarea *i*, donde *i* representa el número de la iteración.
  3. Calcula la diferencia entre el valor del contador 'tick' y el valor del mismo cuando se ejecutó por última vez la tarea *i* (valor guardado en el vector definido por la función *init*).
  4. Si esta diferencia es mayor o igual al periodo de ejecución descrito para la tarea *i*, significa que el periodo se ha cumplido y la tarea debe ejecutarse. En caso de que la diferencia sea mayor o igual al doble del periodo, existe la opción de avisar al usuario de que se ha perdido un ciclo de

ejecución de la tarea.

5. Se comprueba si el tiempo de ejecución previsto para la tarea *i* es menor que el tiempo disponible de ejecución (parámetro de la función).
6. Si es menor, la función ejecuta la tarea, guardando el instante de tiempo en el que se ejecuta y la tarea que se ejecuta. Guarda también el valor del contador 'tick' para la siguiente comprobación y calcula el tiempo que ha tardado la tarea en ejecutarse.
    - Si este tiempo es mayor que el tiempo de ejecución previsto para la tarea, se da la opción de avisar al usuario (para que se plantee cambiar el tiempo de ejecución prevista de la tarea) y, además, podría darse el caso de que se haya superado también el tiempo disponible de ejecución.
    - Si no es mayor, se resta al tiempo de ejecución disponible, obteniendo el tiempo de ejecución restante disponible para ejecutar más tareas, si se pudiese. Se acaba la iteración y pasa a comprobarse otra tarea (paso 3 con i+1).
7. Si no es menor, significa que no se dispone de tiempo de ejecución para esa tarea y se pasa a comprobar otra (paso 3 con i+1).

- `uint16_t time_available_usec (void)`: Calcula y devuelve la diferencia entre el tiempo que lleva la tarea ejecutándose y el tiempo de ejecución previsto de la misma. Si se ha superado el valor previsto, se devuelve cero. Útil para labores de 'debug'.

Existen una función más que, en vistas al código a desarollar, no es muy útil. Calcula la carga de trabajo del administrador de tareas y, según se ha revisado en el código, sólo se usa para comunicar al usuario de dicha carga de trabajo.

5.1.2 Arducopter.pde

El archivo Arducopter.pde hospeda el código principal que se ejecutará en la plataforma ArduPilot, el principal agente ejecutor en el vuelo de cualquiera de las aeronaves soportadas. Este código, compuesto de 1612 líneas, lleva a cabo, en un primer nivel, todo lo necesario para el vuelo correcto de la aeronave, incluidas funciones de comunicación, lectura de sensores, distribución de los recursos disponibles entre procesos (Scheduler), y demás tareas que se irán comentando. Alrededor a él, existen multitud de archivos .pde que son compilados junto a él y que, debido a la gran importancia que tienen, es necesario, al menos, un acercamiento general a sus cometidos; hablamos de control de la aeronave, inicialización de diversos dispositivos, guardado de datos, comunicación con estación en tierra…

Para no alargar la explicación, se decide mostrar el diagrama de flujo siguiente, que da idea general del funcionamiento del archivo. Posteriormente, se hará un breve comentario de los aspectos importantes. Hay que recalcar que estos archivos han sido revisados en mucho menos detalle que las librerías explicadas en la anterior sección, ya que la metodología es lo único extraíble de Arducopter.pde, los demás archivos no son muy interesantes si se centra la vista en los códigos a desarrollar. Como se ha dicho varias veces, el objetivo no es mantener todas las funcionalidades del código ArduCopter, sino extraer los conceptos útiles y usarlos a favor.

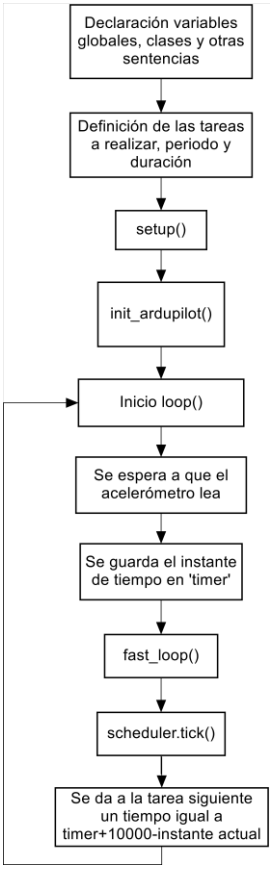

Figura 5.2. Diagrama de flujo principal del código Arducopter.pde

Cada una de las partes se explica a continuación:

- La definición de las clases y variables asociadas a las librerías explicadas en la anterior sección. Las variables tienen objetivos distintos dentro del código que no se detallarán. Debido al gran número de librerías (es decir, de tareas a realizar), esta parte del código supone prácticamente la mitad del mismo. Estas clases son, principalmente:
    - Clase asociada a AP_Hal: definición de pines, recursos, características de placa…
    - Clase asociada a AP_Scheduler: ejecutivo cíclico y funciones.
    - Clase asociada a AP_Notify.
    - Clase asociada a Dataflash.
    - Clase asociada a AP_GPS (si hubiese GPS).
    - Clase asociada a AP_Baro (si hubiese barómetro).
    - Clase asociada a AP_Compass (si hubiese magnetómetro).
    - Clase asociada a AP_InertialSensor.
    - Clase asociada a AP_AHRS.
    - Clase asociada a AP_Mission.
    - Clase asociada a GCS_MAVLINK.
    - Clase asociada a AP_OpticalFlow (si hubiese sensor de flujo óptico).
    - Clase asociada a RangeFinder (si hubiese sónar).
    - Clase asociada a RCMapper.

  - Clase asociada a AP_BoardConfig.
  - Clase asociada a AP_Motors.
  - Clases asociadas a las librerías de control.
  - Clase asociada a AP_BattMonitor.
  - Clase asociada a AP_InertialNav.
  - Clases relacionadas con otras funciones secundarias (AP_Relay, AP_Camera, AP_Sprayer…).

- Definición de las tareas a realizar conjuntamente y a gestionar por el ejecutivo cíclico definido en AP_Scheduler. Para cada una de las dos velocidades que engloba cada una de las placas soportadas (400 Hz y 100 Hz, esta última la disponible en APM 2.6) existe una lista de acciones a controlar por el administrador de tareas. La lista es bastante larga, no se realizará una enumeración: cada una de las que aparecen disponen de distintos periodos y duraciones, y resulta interesante destacar los siguientes hechos:
  - A la velocidad más alta (100 Hz), se realiza la lectura del estimador DCM y del navegador inercial (InertialNav), se activan los controladores que requieren los datos de la IMU y los que ajustarán la orientación de la aeronave y se leen los datos del receptor de radio.
  - A la velocidad de 50 Hz, se lee la altitud y la velocidad en Z desde el navegador inercial, se comprueba si la aeronave está en tierra o ha aterrizado y hace comprobaciones relacionadas con el modo 'Auto'. Además, si hay una cámara instalada, actualiza los controles de los servos para su movimiento y lleva a cabo labores de comunicación y guardado de datos. Por otro lado, actualiza la lectura del GPS
  - A la velocidad de 10 Hz, lee los datos sobre la batería y el magnetómetro, además del guardado de datos. También se actualiza los datos sobre la altitud a partir del barómetro y el sónar.
  - A la velocidad de 3.3 Hz, se realizan labores menos importantes o que requieran menos tiempo de refresco, como comprobar si se ha perdido comunicación con la estación en tierra o tareas relacionadas con AP_Sprayer o AP_Fence.

- En el *setup*, se inicia la función *init_ardupilot*, la cual está definida en otro archivo .pde (system.pde), que se encarga de inicializar todo lo necesario para que Arducopter.pde pueda funcionar correctamente.
- En el *loop*, el programa espera a la lectura del acelerómetro/giróscopo, que se recuerda que se realiza cada flanco de reloj de la APM 2.6. Una vez se ha realizado, se llama a la función *fast_loop* (que incluye lo definido anteriormente a 100 Hz) y se le comunica al administrador que puede dar paso a otra tarea (*scheduler.tick*) con tiempo disponible igual al momento en el que se leyó el acelerómetro, más el tiempo del reloj en microsegundos (100 Hz = 10000 μs) y menos el instante en el que se da paso a la tarea (tiempo de muestreo menos tiempo de ejecución en instante de muestreo).

No se detallarán los archivos paralelos al Arducopter.pde, ya que no han aportado ninguna idea o concepto a aplicar al código desarrollado. Se recomienda, en cambio, revisarlos para conocer en profundidad el funcionamiento de Arducopter.pde y de los controladores.

5.2 Códigos desarrollados

Una vez repasado de forma general el funcionamiento del código ArduCopter, se presenta el código desarrollado para volar la nave AirWhale. Este código se divide, principalmente, en tres archivos; cada uno de ellos incluye una funcionalidad diferente, según la fase de desarrollo del prototipo expuesto en el Capítulo 6. La razón es obvia: es necesaria la implementación del código en el ArduPilot Mega 2.6 y, además, la verificación del mismo con los motores conectados. Conforme el prototipo iba avanzando, el código lo hacía con él, aumentando sus posibilidades y tareas.

Es necesario, pues, la definición de una línea temporal para marcar los hechos que rodean al código desarrollado, ya que el mismo depende fuertemente del contexto en el que se encuentra. Esta línea temporal, en

la que se puede repasar aproximadamente el contexto temporal de todo el trabajo presentado, aparece en la Figura 5.3.

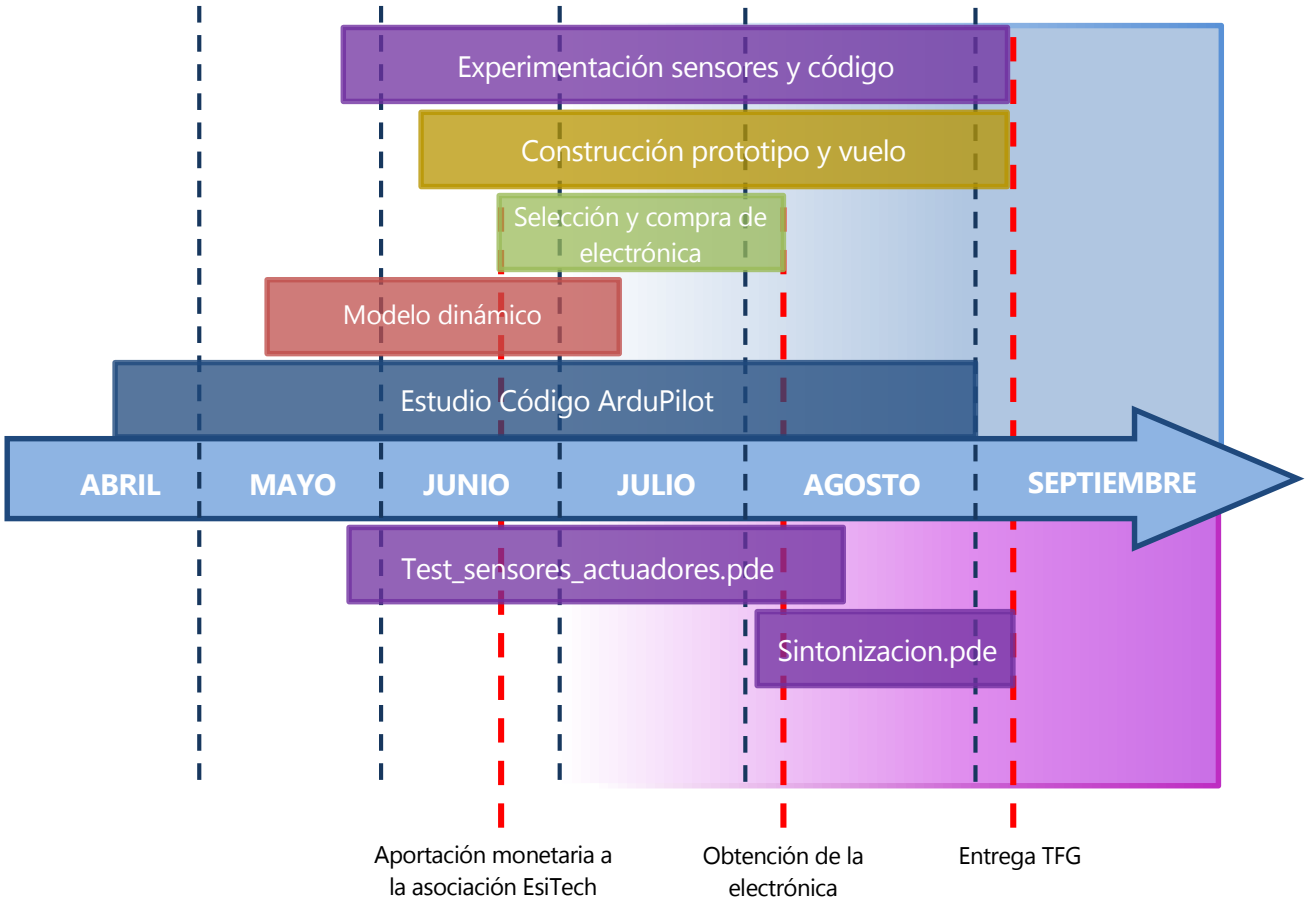

Figura 5.3. Cronograma del trabajo desarrollado en este escrito, detallando los archivos de código creados

Como puede verificarse, existen dos archivos de código creados exclusivamente para el prototipo AirWhale, y se desea mejorar estos archivos y aumentar el número de los mismos en el curso que entra. Estos archivos son los siguientes:

- **Test_sensores_actuadores.pde**: Código realizado para testear los sensores disponibles y a usar con el prototipo. Estos sensores son: acelerómetro, giróscopo, GPS y barómetro. También se incluye una opción de testeo de los motores conectados a la plataforma ArduPilot.
- **Sintonización.pde**: Código utilizado para la sintonización y estabilización sin vuelo del prototipo. En él se definen por primera vez los controladores de los pares en X e Y, y el control de la altura Z.

En la línea temporal se ve cómo el archivo de prueba de sensores y actuadores no pudo finalizarse hasta que no se dispusiese de los motores, ya que sin ellos no podía comprobarse si las funciones de control de actuadores eran eficaces. También, en este mismo sentido, el archivo de sintonización no pudo avanzar mucho en su desarrollo sin disponer de los actuadores, que suponen una de las componentes importantes en el proceso de sintonización *in situ*.

5.2.1 Código *test_sensores_actuadores.pde*

Conforme se iba aprendiendo sobre el funcionamiento de las librerías del firmware ArduPilot, explicadas anteriormente, se fue desarrollando un código sencillo para el testeo y monitorización de las medidas ofrecidas por los sensores. Es un archivo que, pese a su sencillez, cobra gran importancia si se piensa en la trascendencia de los sensores en los códigos posteriores. El trabajo realizado con este código ahorra muchos quebraderos de cabeza a la hora de crear un código completamente desde cero, sin haber probado anteriormente si los sensores funcionan correctamente con las funciones adoptadas desde las bibliotecas. La creación de los siguientes códigos no se vio retrasada por problemas con los sensores.

En una primera aproximación al código, su función se limita a, básicamente, obtener medidas de los sensores e imprimirlas por pantalla. En este sentido, se buscó familiarizarse con los procesos de inicialización y calibrado de los mismos, además de los modos de obtención de medidas. También se hace uso del estimador para obtener un estado de la orientación de la plataforma APM 2.6, y, con ello, la monitorización de las medidas de los sensores que se usan en el mismo (acelerómetro, giróscopo, magnetómetro y barómetro).

Aparte de esto, se puede realizar un testeo de envío de valores PWM a los motores conectados a los pines de salida de la APM 2.6. Esta tarea fue de gran ayuda para cerciorarse de que los motores y los variadores se encontraban en un estado óptimo y, además, para asegurarse de que se conocía perfectamente el modo de controlar los motores mediante PWM.

Funcionamiento general: Diagrama de flujo

El funcionamiento general del programa se puede repasar en el diagrama de flujo de la Figura 5.4. Se pueden apreciar tres niveles generales:

- Definición de las clases necesarias paa la gestión de sensores y actuadores.
- Petición de acción deseada al usuario, que introducirá la opción por teclado.
- Llamada a subrutinas que realizarán las medidas e impresión por pantalla de cada uno de los sensores. En el caso de los actuadores, se habla de una subrutina que envía PWM a cada motor.

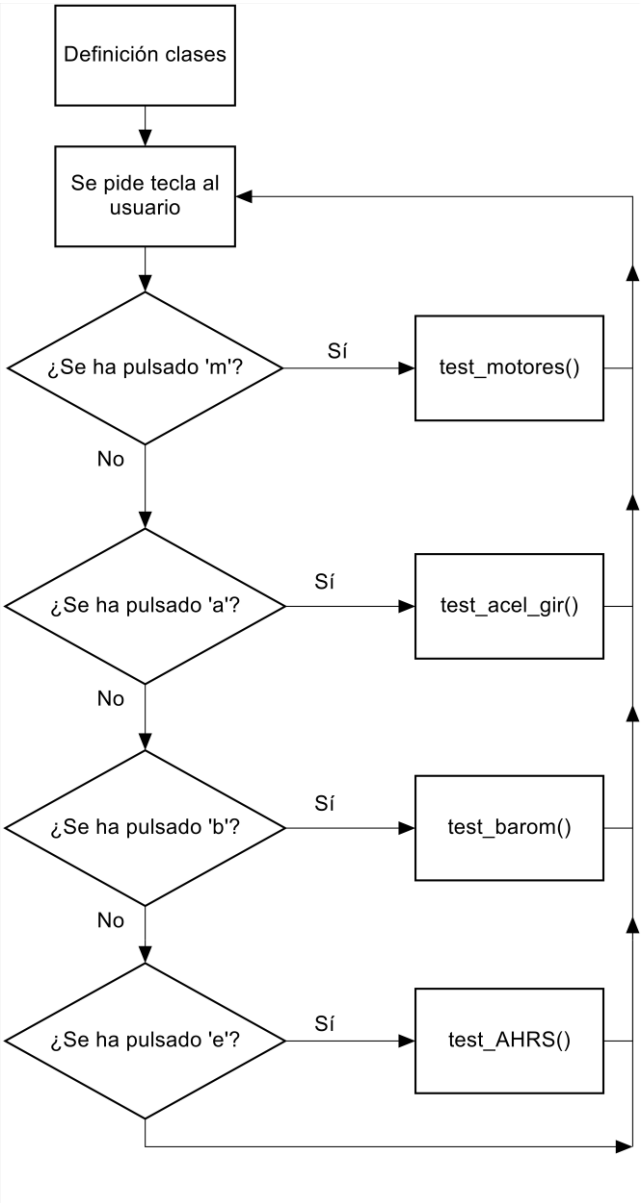

Figura 5.4. Diagrama de flujo del funcionamiento general de *test_sensores_actuadores.pde*

Las clases definidas permiten acceder directamente a las funciones y variables definidas en cada una de las librerías asociadas a la misma. Estas son:

- **AP_Baro_MS5611** *barometer*
- **AP_Compass_HMC5843** *compass*
- **AP_InertialSensor** *ins*
- **AP_AHRS** *ahrs*
- **RC_Channel** *rc1*, *rc2*, *rc3* y *rc4*
- **AP_MotorsQuad** *motors*

El programa solicita al usuario que pulse una letra en el teclado, según la tarea que desea realizar. Las tareas se enumeran como:

- **test_motores**: Realiza una prueba del funcionamiento de los motores.
- **test_acel_gir**: Realiza un testeo de la medición y cálculos relativos al acelerómetro/giróscopo.

- **test_barom**: Realiza una prueba de las medidas aportadas por el barómetro.
- **test_AHRS**: Aporta información sobre la orientación de la placa, fusionando datos del magnetómetro, acelerómetro/giróscopo y barómetro.

Test_motores: funcionamiento y descripción detallada

El objetivo principal de esta subrutina fue la de aprender exactamente cómo entregar el valor deseado de PWM a los motores. En un comienzo, la falta de conocimientos sobre el código ArduCopter y sobre la placa disponible, llevaban a la necesidad de probar si el procedimiento en mente concebido para controlar los motores era el correcto. Las dificultades encontradas sobre la gestión de los actuadores ya se han comentado en su respectiva librería.

Se presenta el diagrama de flujo de la subrutina en cuestión (Figura 5.5). El programa lleva a cabo una solicitud de valores PWM a enviar a cada motor conectado a la plataforma APM 2.6. Para guardar los valores PWM tecleados por el usuario, se usa la función *captura_datos*, explicada más abajo.

Una vez guardados los valores que se deben enviar a cada motor, el programa usa la función ***hal.rcout -> write (channel, PWM)***, la cual, como se explicó anteriormente en este escrito, envía por el pin de salida determinado por el primer parámetro el valor PWM determinado por el segundo parámetro. Para que esta función pueda realizar dicha tarea, debe haberse antes inicializado los motores con ***motors.init*** y ***motors.enable***. Así, usa la función para enviar el valor PWM, espera dos segundos (*delay*) y envía un valor PWM nulo al motor para detenerlo. Repite el proceso para cada motor y, una vez acabada la secuencia, pregunta al usuario si desea repetir el test con distintos valores de PWM (instándole a que pulse 'T'). En caso afirmativo, volverá a repetir el test, de lo contrario la subrutina acabará.

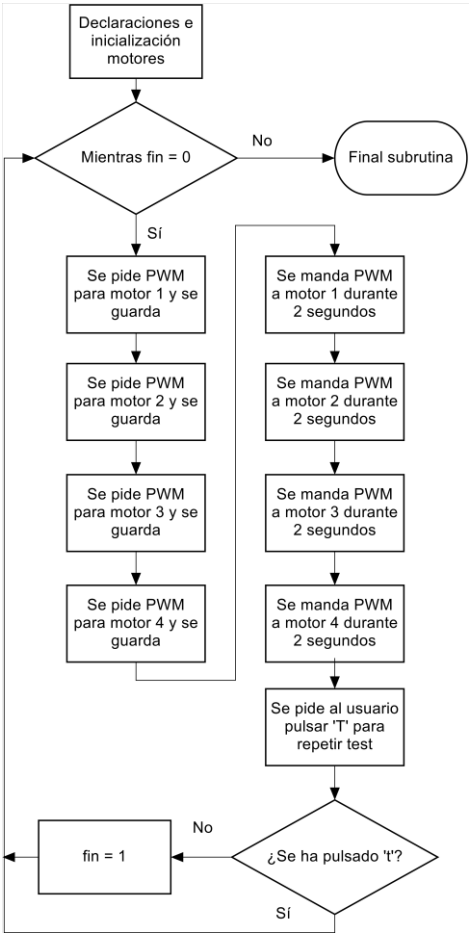

Figura 5.5. Diagrama de flujo de la subrutina test_motores

Test_acel_gir: funcionamiento y descripción detallada

Una subrutina capaz de medir el acelerómetro y el giróscopo, sin calibrar y una vez calibrado, para observar los resultados por pantalla. Este hecho supone la comprobación exhaustiva de las medidas del acelerómetro/giróscopo según unos movimientos predeterminados; estos experimentos podrán repasarse más adelante.

El programa, para empezar, inicializa el acelerómetro. Aquí cabe realizar una puntualización: el acelerómetro/giróscopo, al entrar en conflicto con el barómetro en el puerto SPI, debe inicializarse activando el pin CS asociado al mismo para una correcta inicialización. El pin CS se destina a dar paso a un dispositivo que comparta conexión SPI con otros dispositivos. Cada uno de los dispositivos dispone de un pin CS, de tal manera que, activado uno de ellos, la comunicación micro-dispositivo por SPI sólo se da para ese dispositivo. De igual manera se debe hacer con el barómetro, como se verá más adelante. Una vez inicializados, la recogida de medidas de uno y otro no supondrá ningún problema parecido. Esta tarea se realiza con las funciones ***hal.gpio -> pinMode (número pin, modo de transmisión del pin)*** y ***hal.gpio -> write (número pin, modo de transmisión del pin)***. Para el acelerómetro, el pin CS es el número 40.

El diagrama de flujo se indica en la Figura 5.6.

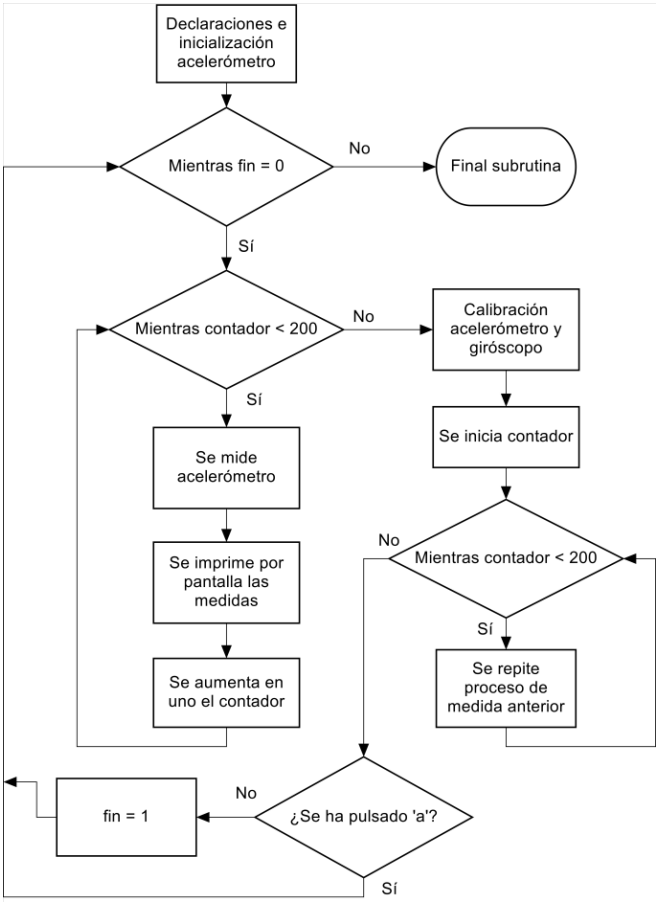

Figura 5.6. Diagrama de flujo de la subrutina test_acel_gir

El funcionamiento es sencillo: se inicializa el sensor en modo WARM_START (como se pudo ver en la correspondiente librería, significa inicio sin calibración de ningún tipo) y a 100 Hz, con la función ***ins.init***. Sin calibrar, se realizan doscientas medidas (tantas como indica el límite del contador impuesto) y se imprimirán por pantalla para su posterior análisis. Para ello, se usa la función ***ins.update***, ***ins.get_accel*** e ***ins.get_gyro***. Terminadas estas medidas, se reinicia el contador y se calibra el sensor, con ***ins.init_accel*** e ***ins.init_gyro***. Con la calibración, se volverán a realizar doscientas medidas. Por último, se comunicará al usuario de que pulsando

'A' puede volver a realizar el test.

En posteriores secciones se podrá comprobar los resultados obtenidos a partir de las medidas de esta subrutina ante movimiento predeterminados de la placa.

Test_barom: funcionamiento y descripción detallada

El funcionamiento de esta subrutina es semejante a la descrita anteriormente para el acelerómetro. Se inicializa el barómetro (***barometer.init***), teniendo en cuenta la multiconexión en el puerto SPI (en este caso, el pin CS para el barómetro es el 63), y se realizan doscientas medidas sin calibrar (***barometer.read***) y doscientas medidas habiendo calibrado el sensor. Todas ellas se imprimirán por pantalla para su posterior análisis. Esta vez, la letra para volver a realizar el test es 'B'. El diagrama de flujo se muestra a continuación.

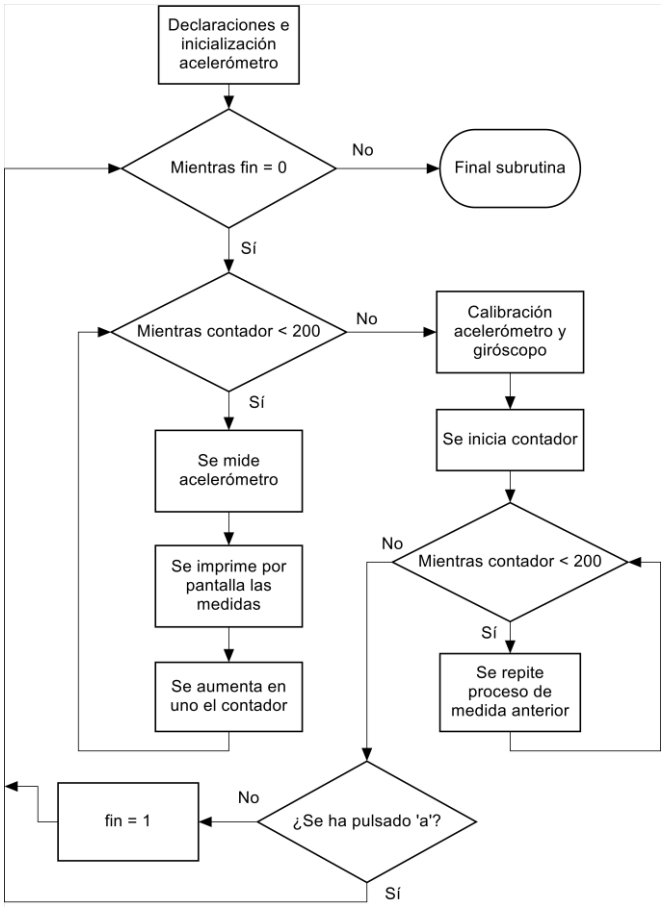

Figura 5.7. Diagrama de flujo de la subrutina test_barom

Test_AHRS: funcionamiento y descripción detallada

El funcionamiento correcto del estimador pasa por inicializar los sensores GPS (***GPS.init***, no definida anteriormente debido a que sólo será utilizado para que el estimador pueda iniciarse, no se tendrán en cuenta sus medidas en este trabajo), acelerómetro y barómetro. Posteriormente, el programa será semejante a los anteriores, usando la función ***ahrs.init*** y accediendo a las variables globales definidas en la librería AP_AHRS que contienen los ángulos (***ahrs.roll***, ***ahrs.pitch*** y ***ahrs.yaw***).

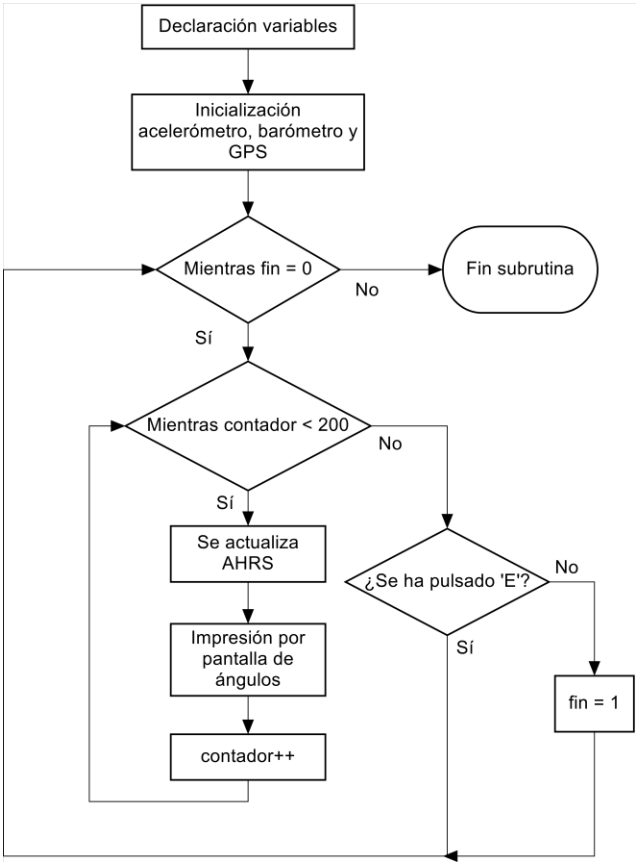

Figura 5.8. Diagrama de flujo de la subrutina test_AHRS.

5.2.1.1 Experimentos realizados con el barómetro

Los datos obtenidos mediante el programa descrito anteriormente sobre el barómetro son:

- Presión, en Pa.
- Altura, en metros.
- Temperatura, en grados Celsius.

Así, se van a realizar dos experimentos:

- **Primer experimento**: Se elevará lentamente la placa hasta aproximadamente un metro y diez centímetros de altura y volverá a dejarse sobre la altura cero. Se mostrarán los datos sin calibrar y los datos calibrados.

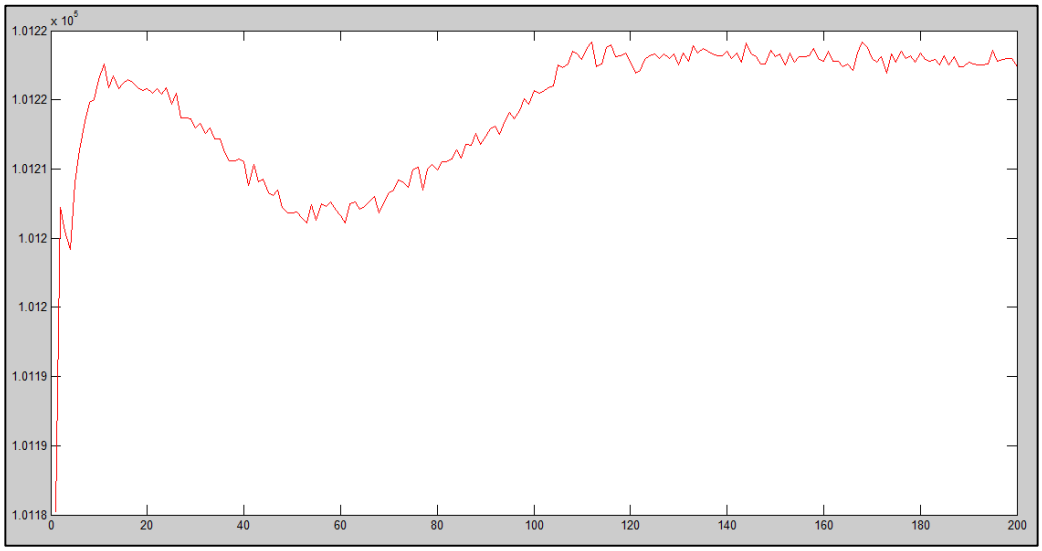

Figura 5.9. Resultados experimento 1: presión sin calibrar

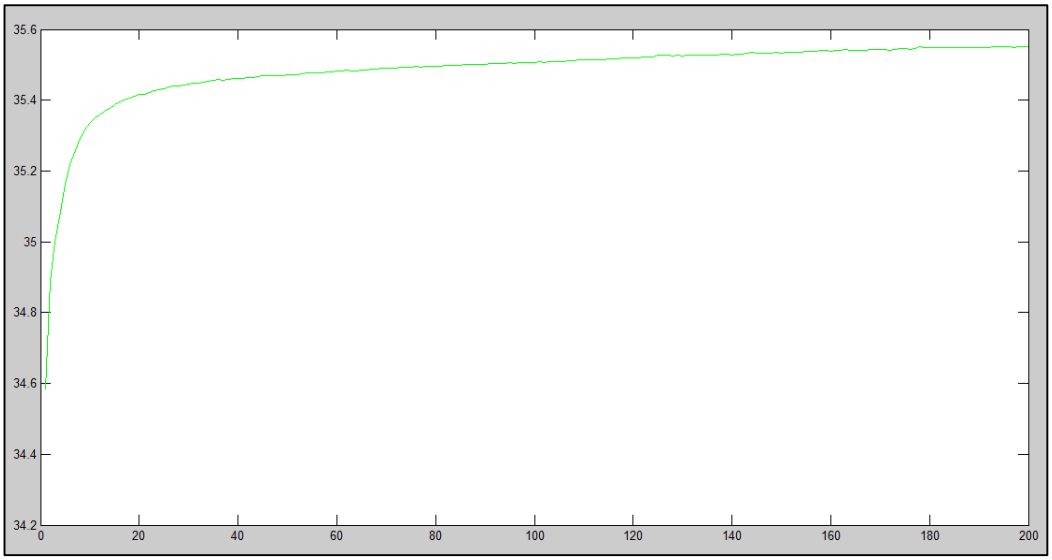

Figura 5.10. Resultados experimento 1: temperatura sin calibrar

La presión, al no estar calibrada inicialmente, debe sufrir un proceso estabilización en los primeros instantes de medida, ya que se parte desde una presión base ficticia nula. Posteriormente, se puede observar cómo la presión reduce su valor conforme se va elevando la plataforma APM 2.6. Cuando la plataforma va reduciendo su altura, la presión aumenta. Se ha intentado elevar y bajar la plataforma al mismo ritmo, hecho que se refleja en las medidas.

Por su parte, la temperatura también debe partir desde valores más bajos, y aumenta considerablemente en los primeros instantes para, luego, y estabilizándose en un valor cercano a los 35,6 grados. La altitud no se ha representado, ya que sin una calibración previa, las funciones de cálculo de la altitud devuelven una altitud nula, por cuestiones de seguridad. Se debe calibrar el sensor para obtener medidas de la altura de la aeronave, obligatoriamente.

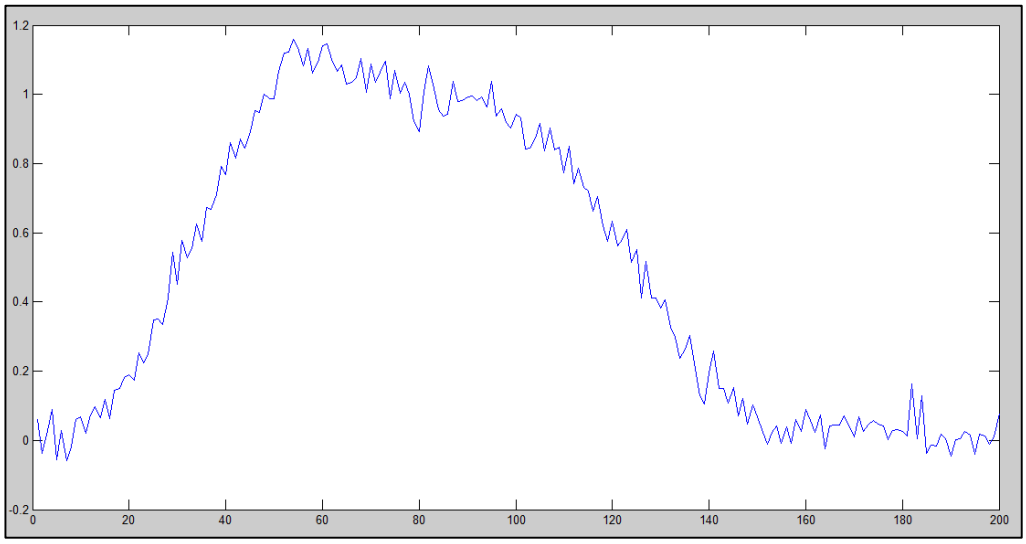

Figura 5.11. Experimento 1: altitud calibrada.

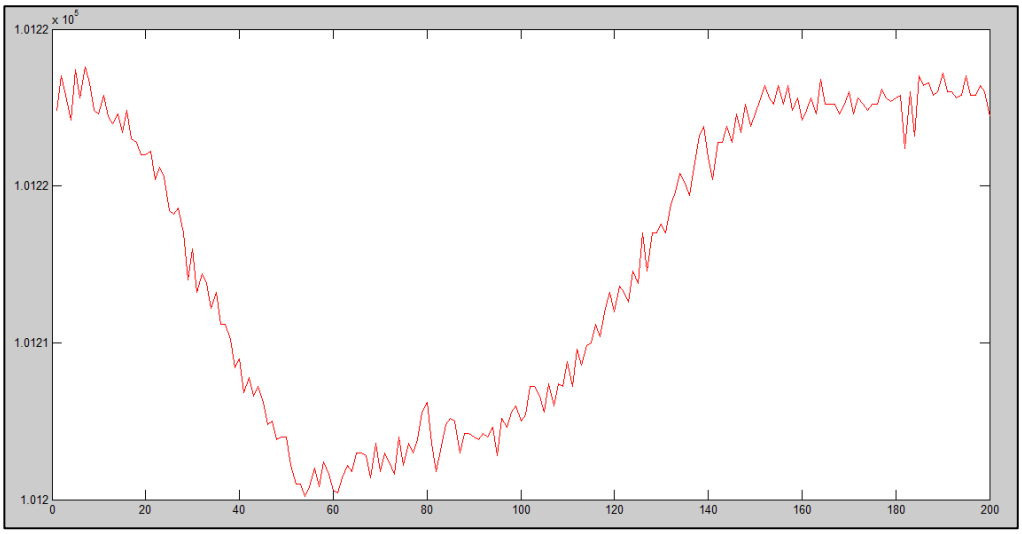

Figura 5.12. Experimento 1: presión calibrada

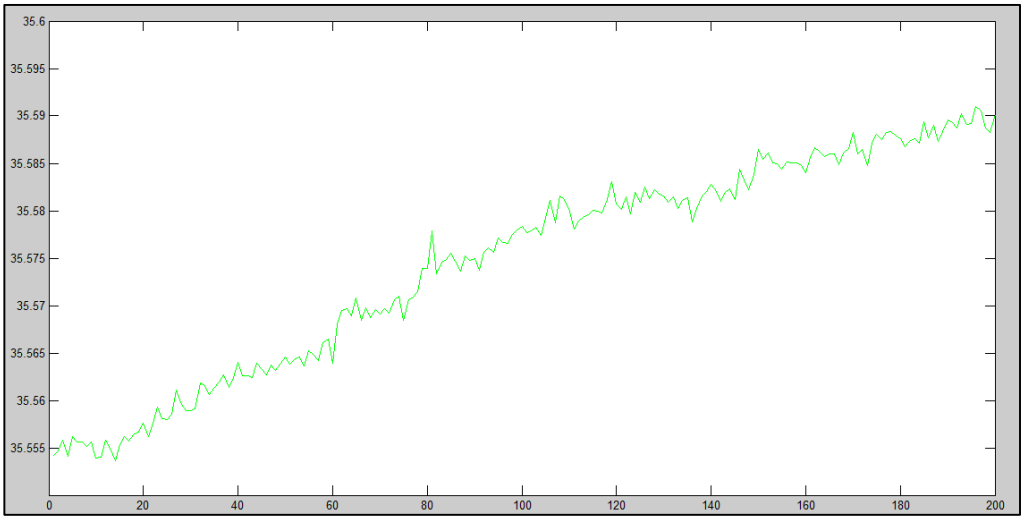

Figura 5.13. Experimento 1: temperatura calibrada

La calibración elimina el periodo de estabilización en los primeros instantes de medida, iniciando las mismas en los valores 'offset'. Así, se puede observar cómo la altitud aumenta a un ritmo constante, deja de crecer al llegar a la altura prevista, y decrece a ritmo aproximadamente constante y de forma semejante al crecimiento, hasta llegar a la altura base. De forma contraria, evoluciona la presión. Hay que añadir el considerable ruido que acompaña a las medidas, hecho que puede ser producido por multitud de factores (forma de agarrar la placa al realizar los experimentos, atmósfera de la habitación donde se realizan los mismos…). Según el datasheet del fabricante [33], el error, en milibares, es aproximadamente de ±2.5 mbar, lo que se traduce en ±250 Pa.

En cuanto a la temperatura, sufre un aumento aproximadamente constante, debido, seguramente, al agarre de la mano sobre la placa.

- **Segundo experimento**: Se elevará rápidamente la placa hasta aproximadamente un metro y diez centímetros de altura y volverá a dejarse sobre la altura cero. Se ejecutará este movimiento cinco veces. Se mostrarán sólo los datos calibrados.

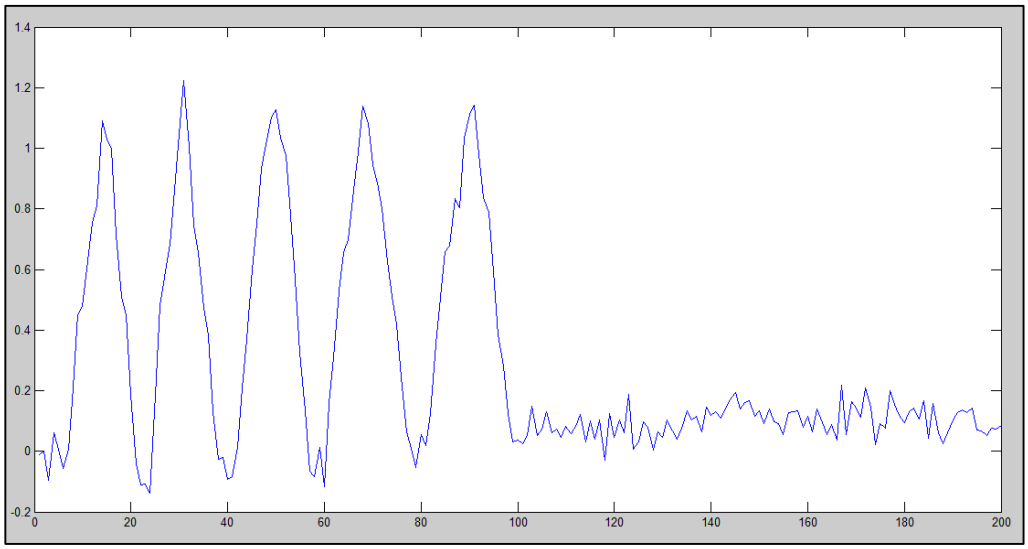

Figura 5.14. Experimento 2: altitud calibrada

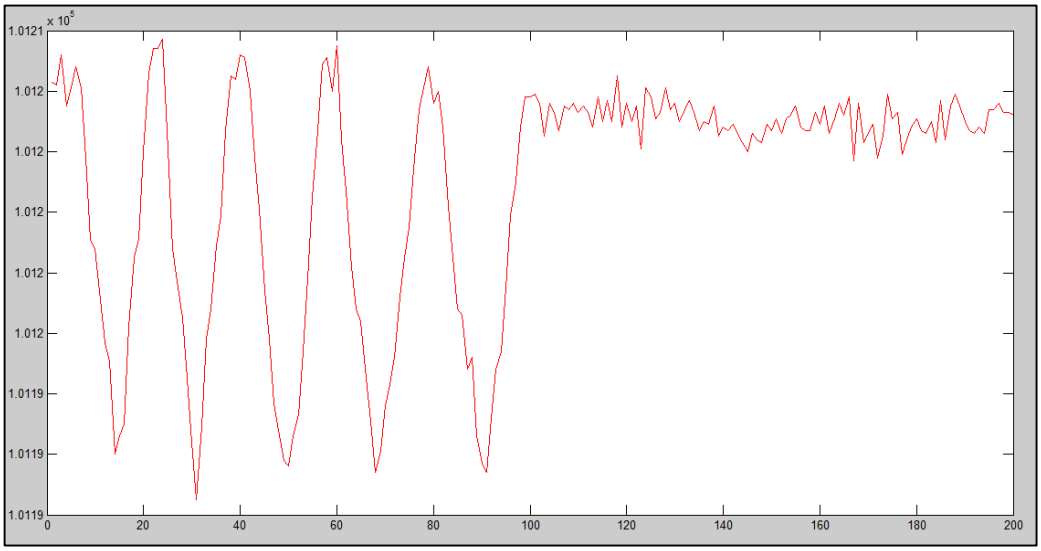

Figura 5.15. Experimento 2: presión calibrada

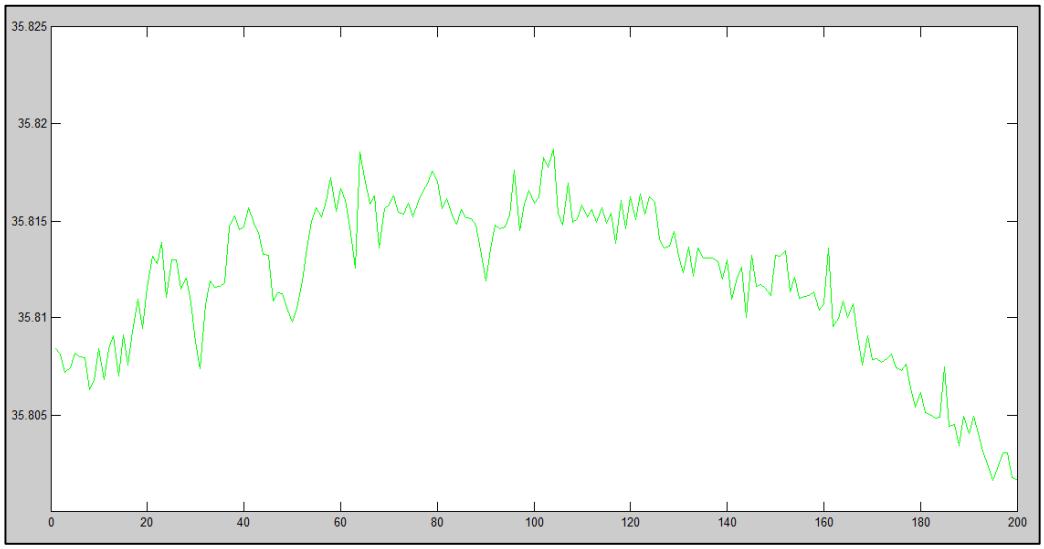

Figura 5.16. Experimento 2: temperatura calibrada

Independientemente del ruido, que es inherente a la medida, se observa un comportamiento bastante fiel a la realidad, teniendo en cuenta que en el experimento realizado no se ha elevado con total exactitud la placa a 1,1 metros. La evolución de la altitud en cada uno de los cinco movimientos es rápida, observándose bastante exactitud en la altura máxima de los tres últimos movimientos. Una vez la placa se abandona en la altura base, se observa un 'offset' espúreo que aumenta la medida en pocos centímetros. La presión, como se ha observado en las anteriores gráficas, evoluciona a la par que la altitud. La temperatura aumenta, reduciendo su crecimiento y disminuyendo aproximadamente a la mitad del experimento, pudiendo ser debido al enfriamiento de la placa debido al rápido movimiento de la misma.

5.2.1.2 Experimentos realizados con el acelerómetro/giróscopo

Los datos que se toman de las medidas del acelerómetro/giróscopo son:

- Aceleraciones lineales en X, Y y Z, en m/s^2.
- Velocidades angulares en los giros sobre X, Y y Z, en rad/s.

Así, como se hizo con el barómetro, se van a realizar dos experimentos:

- **Primer experimento**: Se va a girar el sensor sobre cada uno de los ejes X e Y, en las dos direcciones de giro, de forma suave, 90 grados aproximadamente, como se muestra en la Figura 5.17. Se mostrarán las medidas sin calibrar y calibradas.

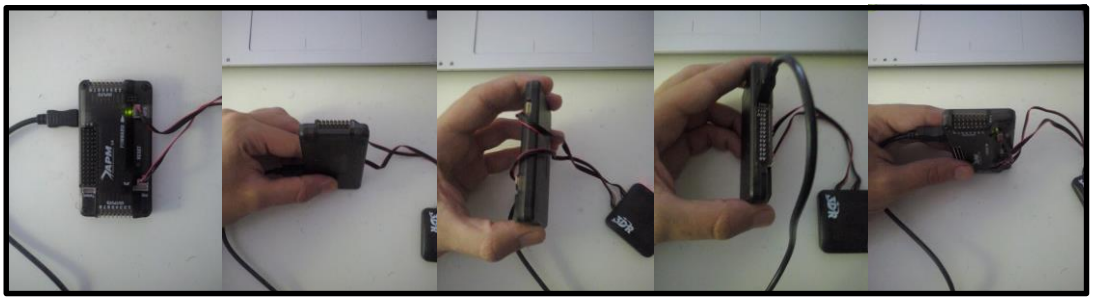

Figura 5.17. Secuencia de movimientos para los experimentos con el acelerómetro/giróscopo

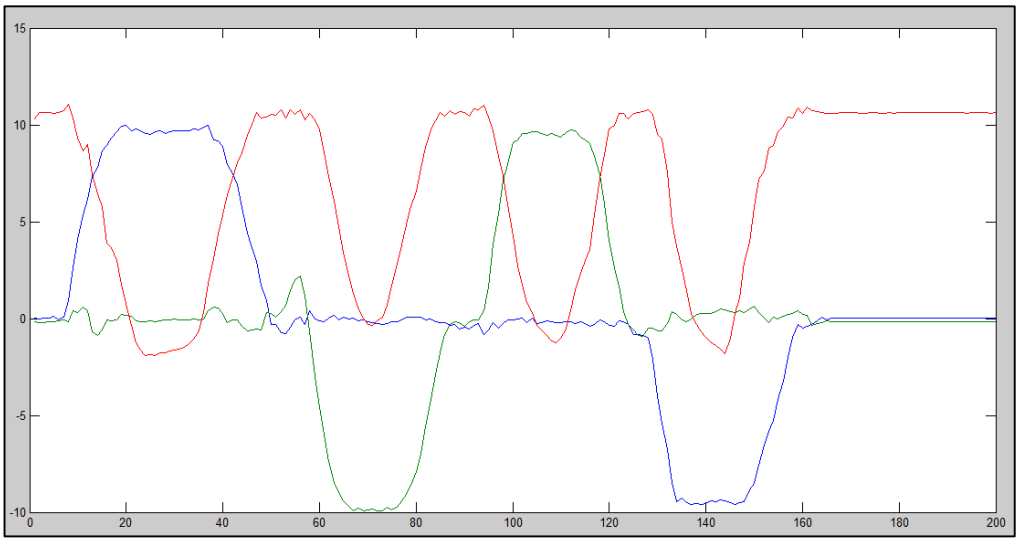

Figura 5.18. Experimento 1: aceleraciones lineales sin calibrar

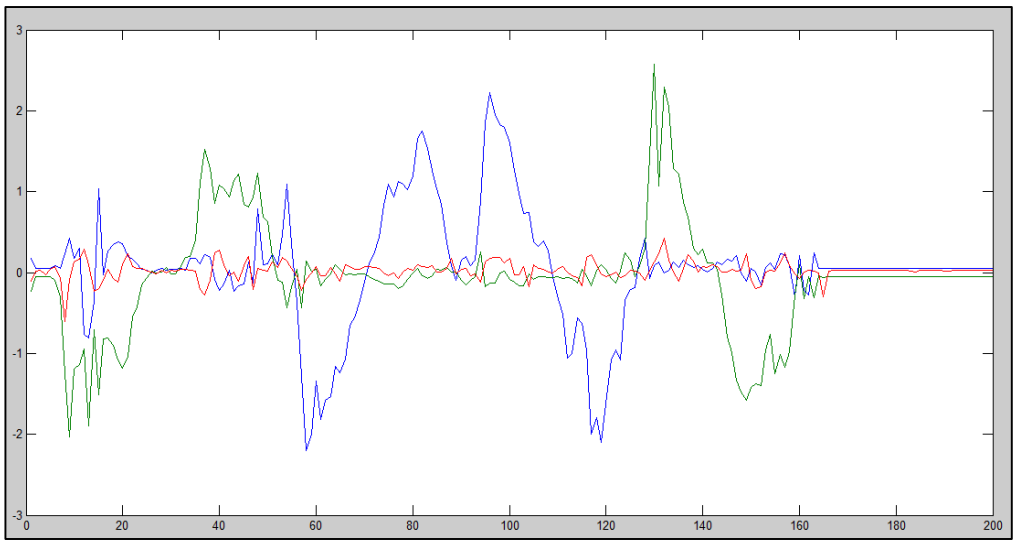

Figura 5.19. Experimento 2: velocidades angulares sin calibrar

Las medidas en aceleraciones parecen tener sentido, ya que en el primer giro en Y la aceleración de la gravedad (recordar el sistema de coordenadas local de la aeronave, Z apuntaba haca abajo) pasa a tener componente única en X positiva; del mismo modo, se puede observar que en el último giro en Y pasa a tener componente X única negativa. Así, el primer giro sobre el eje X supone que la componente de la gravedad, positiva en Z, se convierta completamente en componente negativa en Y. El último giro en Z supone que la gravedad se convierta en componente positiva en Y. Sin embargo, y ahí es donde se nota la ausencia de calibrado, el módulo de la gravedad tiene un valor por encima de 10 m/s^2. También se observa al final del experimento cómo las aceleraciones en X e Y no son exactamente cero.

En velocidades angulares, también se encuentran medidas lógicas, obteniendo velocidades negativas en Y en el primer giro de 90 grados (supone un par negativo), con su posterior valor nulo (se mantiene la placa en ese ángulo un momento) y velocidad positiva al volver al estado inicial (supone un par positivo). Cuando se realiza el giro en X final, se obtiene justo lo contrario a lo anteriormente explicado, ya que supone un par positivo y luego un par negativo en X. De igual manera, se puede razonar el valor y sentido de la velocidad angular en X. La velocidad en Z, al no haber realizado ningún giro significativo sobre el eje Z, se mantiene aproximadamente constante. En cuanto a la calibración, al final del experimento se observa que ninguna de las velocidades es

exactamente cero, y que la velocidad en Y se desmarca bastante de las otras dos.

El sensor parece ser bastante preciso, dentro de lo que cabe. Afortunadamente, se verá en los experimentos con el estimador, que el ruido de los sensores fusionados se reduce considerablemente.

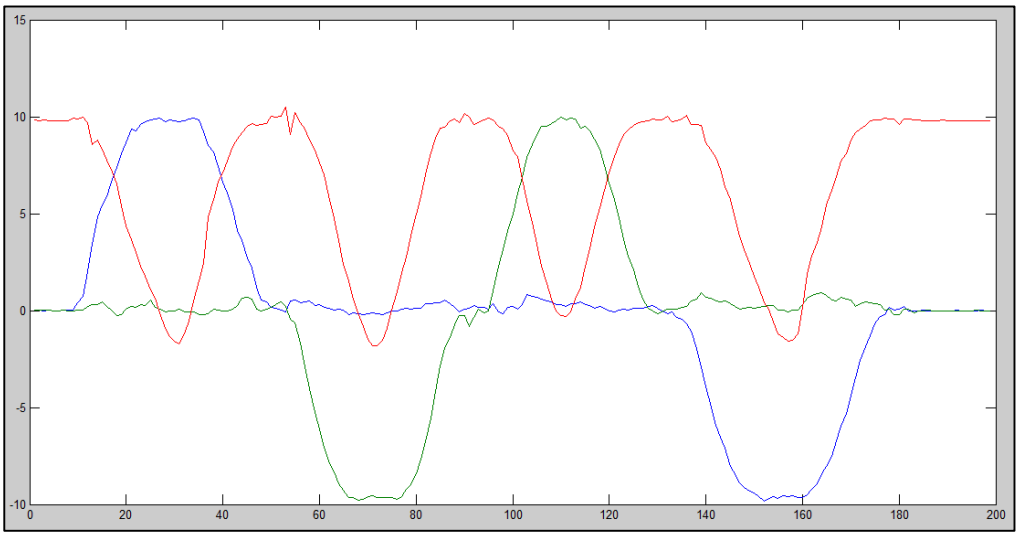

Figura 5.20. Experimento 1: aceleraciones lineales calibradas.

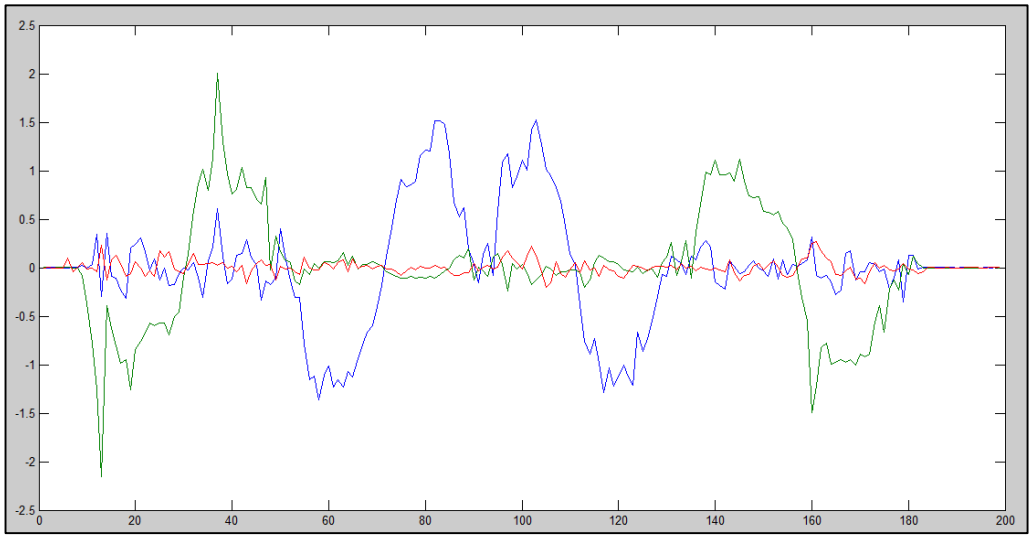

Figura 5.21. Experimento 1: velocidades angulares calibradas.

En las aceleraciones angulares se observa que se ha añadido el 'offset' que elimina la componente espúrea y la aceleración de la gravedad vuelve a tener un valor parecido a 9,81 m/s^2. En cuando a las velocidades, se observa que, al final del experimento, las tres velocidades confluyen en aproximadamente 0 rad/s.

- **Segundo experimento**: Igual que el anterior, pero de una forma más violenta. Sólo se mostrarán las medidas calibradas.

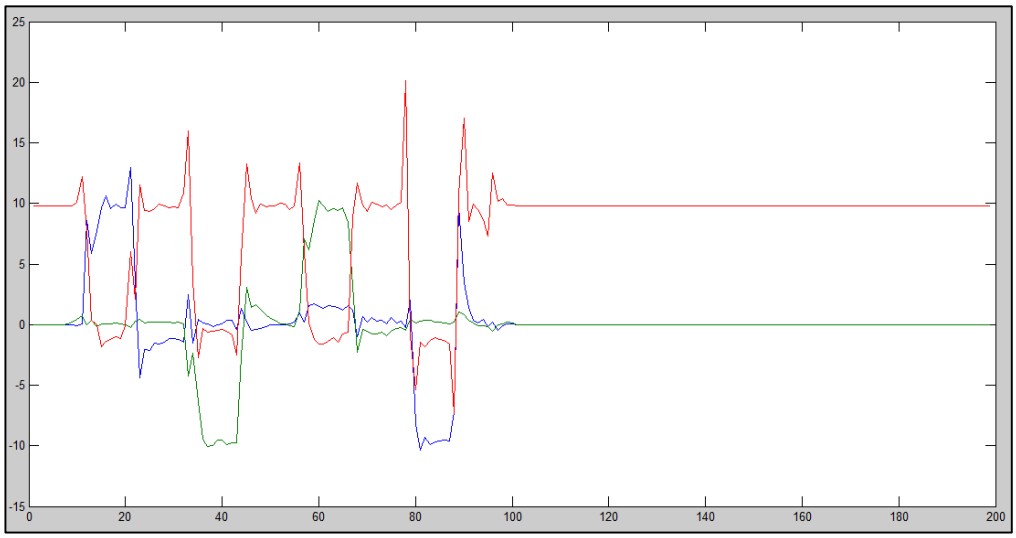

Figura 5.22. Experimento 2: aceleraciones lineales calibradas.

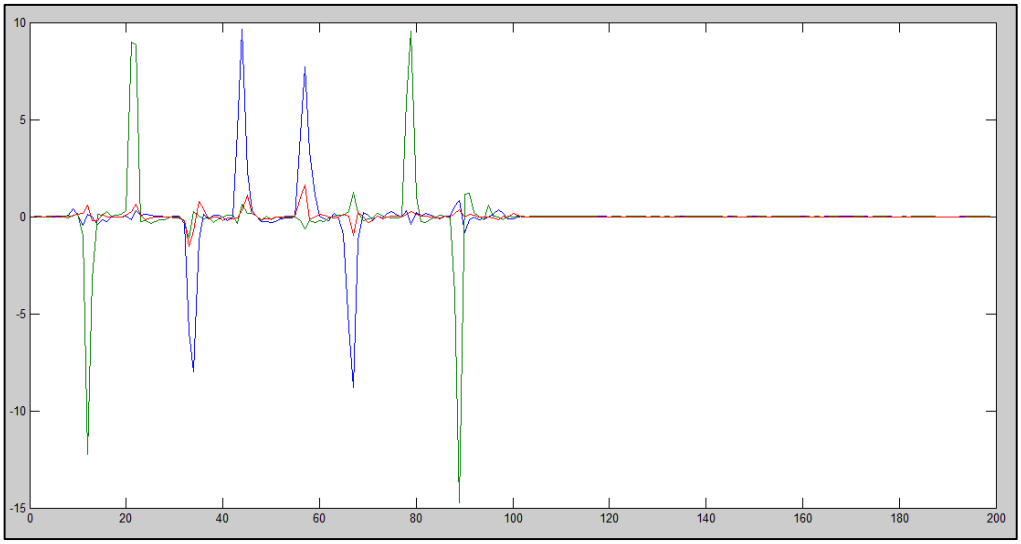

Figura 5.23. Experimento 2: velocidades angulares calibradas.

Al ser más violentos los movimientos, la aceleración aumenta considerablemente su valor debido a la componente que se genera en el arranque y en el freno, llegando a registrarse hasta 20 m/s^2 de aceleración. Estas componentes se registran en todos los ejes, como se puede observar. En el caso de las velocidades, no hay mucho más que decir: al ser movimientos bruscos, el tiempo de evolución es mucho menor, pero el módulo aumenta considerablemente, hasta en un 1000%.

5.2.1.3 Experimentos realizados con el estimador DCM

Los experimentos a realizar son semejantes a los realizados con el acelerómetro/giróscopo (Figura 5.17), exceptuando que no se realizará un experimentocon movimientos bruscos, para simplificar el proceso. Los datos a mostrar serán los ángulos de orientación de la aeronave: ángulo roll, pitch y yaw.

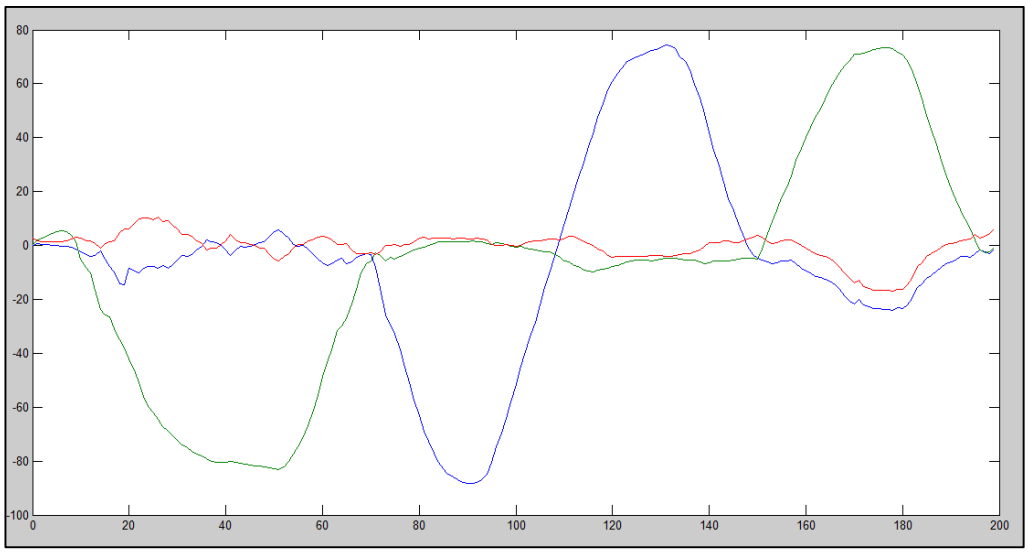

Figura 5.24. Resultados del experimento con el estimador DCM.

Como puede comprobarse, la medida de los ángulos es precisa (los movimientos no lo son tanto, desafortunadamente), registrándose un aumento negativo del ángulo pitch (verde) en el primer movimiento sobre el eje Y, y un aumento positivo en el último movimiento. En el caso del ángulo roll, se obtiene un resultado semejante, siendo los signos de la variación totalmente correctos.

5.2.2 Código *sintonizacion.pde*

La estabilización deseada de variables de estado determinadas es, claramente, uno de los objetivos principales a la hora de programar el firmware. Es necesario el control de ciertas variables inherentes al vehículo para que pueda operar sin incidencias. En el caso del AirWhale (representado por el prototipo mostrado en el Capítulo 6) existen tres variables de estado principales que hay que fijar a un valor determinado mediante el uso de controladores:

- Ángulo de alabeo o ángulo ϕ , relacionado con el movimiento de *roll* (par en X).
- Ángulo de cabeceo o ángulo $\boldsymbol{\theta}$, relacionado con el movimiento de *roll* (par en X).
- Coordenada Z o altitud de la aeronave.

En este caso, se desea una estabilización completa de la orientación en el plano XY con un ángulo de 0 grados. Por otro lado, el control en Z depende directamente de la altura que se desee alcanzar. Es importante que los ángulos alabeo y cabeceo estén muy próximos a cero, ya que, en caso contrario, sería muy complicado mantener la dirección efectiva de empuje de los motores en vertical. No se necesita que los motores verticales estén aportando una componente de fuerza distinta a la que aportan en dirección Z; para moverse a lo largo del plano XY se dispone de los motores horizontales (en el Capítulo 6 se detalla esta configuración de actuadores).

Resumiendo, hay que crear un programa capaz de realizar esta estabilización deseada en alabeo y cabeceo, y en altura. Es necesaria, pues, la acción de los sensores capaces de obtener la orientación de la aeronave en tiempo real y la altura de la misma. Estos sensores son:

- Acelerómetro/giróscopo, barómetro y magnetómetro, que hacen funcionar el estimador DCM que devuelve los ángulos de orientación con respecto a cada eje de la placa.
- Barómetro, para obtener en cada iteración de control la altura de la aeronave en ese instante.

Obtenida las medidas, procede realizar un control sobre ellas, enviando la actuación necesaria a los motores

verticales para hacer que esas medidas tiendan al valor deseado. Cada uno de los motores influye de manera específica en cada una de las variables de estado y, por ende, en las variables que se desean controlar. La relación entre actuación individual de cada motor y las fuerzas y pares de actuación se exponen a continuación.

El archivo *sintonización.pde* realiza la tarea descrita.

Funcionamiento general: Diagrama de flujo

El diagrama de flujo que representa el funcionamiento general del programa se muestra en la Figura 5.21. Principalmente está compuesto de los siguientes procedimientos:

- Declaración de las variables globales necesarias y de las clases correspondientes a cada sensor, a los motores actuadores y al estimador ARHS. Además, se realiza una inicialización de cada uno de ellos de la siguiente manera:
    - El acelerómetro se inicializa en modo COLD_START (previa selección de su respectivo Chip Select), esto es, con la calibración implícita del giróscopo. Por ello, se añade en la siguiente línea de código la calibración sólo del acelerómetro.
    - Se inicia el barómetro seleccionando su CS y se calibra.
    - Inicialización del magnetómetro.
    - Inicialización de los motores y envío de un PWM nulo para la inicialización de los variadores.
- Petición de cada uno de los parámetros de definen el control de cada variable de estado a controlar:
    - K, Ti y Td del controlador PID y referencia relacionados con el ángulo de cabeceo.
    - K, Ti y Td del controlador PID y referencia relacionados con el ángulo de alabeo.
    - K, Ti y Td del controlador PID y referencia relacionados con la altitud de la aeronave.

Estos parámetros tendrán un valor distinto dependiendo del carácter de la actuación que proporcionen los controladores PID. Esto es muy importante a la hora de aplicar un valor de PWM a cada motor, ya que una sintonización y tratamiento posterior de las actuaciones no coordinadas entre sí puede llevar a la insuficiente fuerza de los motores para conseguir levantar la aeronave o el descontrol de la misma debido al valor desmesurado del PWM a pasar al motor. En el Capítulo 6 se indicarán los parámetros usados en los experimentos de estabilidad, el tratamiento de las actuaciones obtenidas y sus resultados, tanto a nivel teórico como a nivel de código.

La obtención de los parámetros se realiza mediante la función *captura_datos*.

- Inicio del bucle de control, donde se lleva a cabo:
    - La lectura del barómetro para obtener la altitud de la aeronave en tiempo real y la actualización del estimador para obtener los ángulos de orientación del vehículo en tiempo real. Estas medidas son necesarias para calcular el error, entre las medidas reales y las referencias introducidas, que será la entrada al PID.
    - El cálculo de las actuaciones con la aplicación de los controladores PID.
    - Tratamiento de las actuaciones según el carácter de las mismas.

La medición del barómetro, como pudo apuntarse en la definición de su correspondiente librería, se realiza, máximo, a 0.01 segundos. Por otro lado, la estimación de los ángulos mediante el filtrado DCM puede realizarse sin dificultades a la misma velocidad. Sabiendo esto, el tiempo de muestreo de los controladores debe ser, mínimo, el valor del sensor de mayor periodo de lectura (en este caso, el barómetro). Es muy importante sincronizar estos tiempos, y, además, planificar el valor del tiempo de muestreo, por tres razones:

1. Es obvio que si el tiempo de muestreo de los controladores es menor que el periodo de medida de los sensores, existirán bucles de control con la misma medida del bucle anterior, ya que no habrá dado tiempo a obtener una medida nueva. Esto, evidentemente, no es deseable.

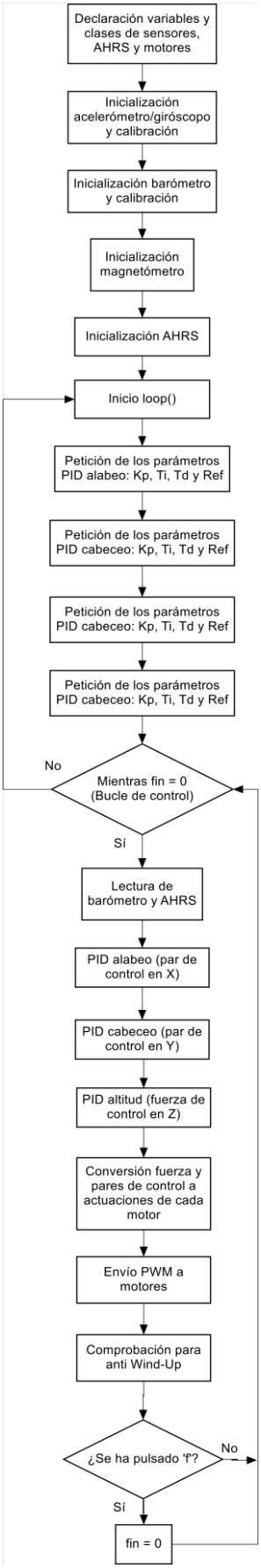

Figura 5.25. Diagrama de flujo del funcionamiento general de sintonizacion.pde

2. El tiempo de muestreo debe tener un valor lo suficientemente bajo como para aproximar el resultado de los controladores programados a los resultados extraídos de los controladores continuos basados en el modelo dinámico del sistema mostrado en el Capítulo 3.
3. El tiempo de muestreo debe ser mayor al tiempo de cálculo de Matlab, y el tiempo de envío y recepción de datos por puerto serie con el programa, en el caso del sistema 'hardware in the loop'. El motivo es sencillo: si no es mayor, los datos aportados por el programa, que corresponden con las medidas de hipotéticos sensores, serían los mismos en varios bucles de control, y no es deseable.

Como consecuencia de esto, hay que remarcar que el algoritmo programado correspondiente con la aplicación de los controladores PID depende estrechamente de la precisión del mismo con respecto al tiempo continuo. El algoritmo empleado para implementar el control PID se muestra en la ecuación (2):

$$U_k = K_p \cdot \left[E_k \cdot \left(\frac{T}{T_i} \cdot I_k \right) \cdot \left(\frac{T_d}{T} \cdot (E_k - E_{k-1}) \right) \right] \tag{2}$$

Donde K_p, T_i y T_d son los parámetros característicos del controlador PID, U_k es la actuación o salida del PID en el instante actual de muestreo, E_k y E_{k-1} son el error entre la medida y la referencia en el instante actual de muestreo y el anterior instante de muestreo, respectivamente; T es el tiempo de muestreo del controlador e I_k es el valor de la integración del error en cada instante de muestreo, y es igual a $I_k = I_{k-1} + E_k$, donde I_{k-1} es el valor de la integración en el instante anterior de muestreo.

El tratamiento de las actuaciones, salida de cada PID, es imprescindible, ya que representan los pares y fuerzas de control y no las actuaciones individuales de cada motor. Para obtener qué acción debe realizar cada motor para conseguir los pares y fuerzas deseadas, se usa el conjunto de ecuaciones mostradas en la sección 6.1 del presente trabajo, ya que están asociadas con la geometría del prototipo.

La comprobación anti Wind-Up se basa en chequear si las actuaciones han llegado a saturar, momento en el que no es deseable que la componente integral del controlador PID siga integrando el error. Este hecho puede llevar al retardo en la respuesta instantes posteriores a un cambio en la referencia que necesite una actuación menor a la de saturación. Si la actuación satura, se activa un flag que determina que la integral del error del controlador asociado a esa actuación deja de actualizarse, manteniendo el valor de ese instante mientras la actuación siga saturando. En el momento en el que deja de saturar, el flag se desactiva y el efecto integral vuelve a actualizarse.

Los experimentos de la estabilización de las variables de estado a controlar se muestran también en el Capítulo 6, donde se usará este código para obtener datos de validación.

5.2.3 Función *captura_datos*

La función captura_datos es necesaria para la obtención de datos provenientes del usuario. La función que obtiene datos por el puerto serie, definida en el firmware ArduPilot, es ***hal.console->read***. Esta sentencia recoge los datos obtenidos por teclado, pero en código ASCII. Esto es un problema, ya que el código ASCII no es un número fiel a lo introducido por el usuario. Por ello, se ha visto conveniente la creación de una función capaz de obtener estos datos y convertirlos en un número útil para las funciones de PWM y los controladores PID. El concepto es sencillo:

- Se capturan los datos del teclado. En el caso de valores PWM (archivo *test_sensores_actuadores*) el número podrá ser un número entero entre 9999 y 0, mientras que en el caso de los controladores (archivo *sintonizacion*) podrá ser un número entre 99.99 y 00.00, siempre con dos decimales. El motivo se verá a continuación.
- Se va obteniendo cada dato por teclado, de tal manera que se asociará el primer número leído como los millares (PWM) o decenas (controladores), el segundo número leído como centenas (PWM) o unidades (controladores), el tercer número como decenas o décimas y el último número como unidades o

centésimas. Cualquier otro dato ASCII ajeno a un número se desecha.

- Una vez obtenidos, se realizan las siguientes conversiones, teniendo en cuenta que los números entre 0 y 9 estarán, en ASCII, incluidos en el intervalo 48-57:
  - En el caso de PWM:

$$PWMfinal = \big(1000 * (millares - 48)\big) + \big(100 * (centenas - 48)\big) + \big(10 * (decenas - 48)\big) + \big(1 * (unidades - 48)\big)$$

  - En el caso de los controladores:

$$Paramfinal = \big(10 * (decenas - 48)\big) + \big(1 * (unidades - 48)\big) + \big(0.1 * (décimas - 48)\big) + \big(0.01 * (centésimas - 48)\big)$$

$PWMfinal$ y $Paramfinal$ son los valores a usar por las funciones de PWM y los controladores, respectivamente.

6 CONSTRUCCIÓN DEL PROTOTIPO AIRWHALE Y EXPERIMENTACIÓN

La intención de construir y volar la aeronave AirWhale siempre ha sido el principal objetivo del equipo de desarrollo de la asociación EsiTech. Desde las primeras reuniones del equipo la idea de construir una versión final del AirWhale ha rondado la mente de todos los integrantes. Por desgracia, debido a la falta de tiempo y de recursos, se desechó la posibilidad de poder construir el vehículo.

Sin embargo, el alumno José Luis Holgado Álvarez y yo vimos necesaria la construcción de una aeronave capaz de realizar movimientos parecidos al AirWhale y con una dinámica parecida, para la verificación del trabajo de José Luis y el mío, descrito en el Capítulo 5: el diseño teórico de controladores del estado dinámico de la aeronave AirWhale y la creación de un firmware que gestione adecuadamente los sensores y actuadores, además de otras funciones, a través de una plataforma autopiloto. Este trabajo tenía una evidente componente experimental en la verificación, y se contaba, en un principio, con la versión final de la aeronave AirWhale para pruebas de vuelo. Como ya se ha dicho, no ha sido así.

Por ello, se decidió construir un prototipo con dinámica parecida y que pudiese operar con los dispositivos seleccionados para la versión final del AirWhale, de manera que los experimentos llevados a cabo con esta primera versión supusiesen una primera aproximación a lo que podría aportar el AirWhale en su versión final. Es decir: ensayar con una aeronave menos costosa, tanto en tiempo como en recursos de construcción, semejante en comportamiento dinámico y observar qué resultados (logros, problemas, posibilidades...) se pueden extraer de ello.

En las siguientes secciones se repasarán todos los aspectos que rodean al prototipo: disposición geométrica, materiales y dispositivos electrónicos a usar, proceso de construcción, conexionado, estabilización estática y resultados. El diseño ha sido llevado a cabo por el alumno José Luis Holgado Álvarez.

6.1 Introducción: geometría y movimientos del prototipo

La geometría del prototipo, debido a que se desea que guarde similitud dinámica con el AirWhale original, dispone de un globo de helio cilíndrico, con extremos semiesféricos, en el centro, rodeado de soportes para los motores verticales y horizontales, y demás electrónica. También dispone de una estructura de apoyo que evita que el globo contacte con el suelo. El diseño en CATIA del prototipo, realizado por el alumno José Luis Holgado Álvarez, se muestra en la Figura 6.1.

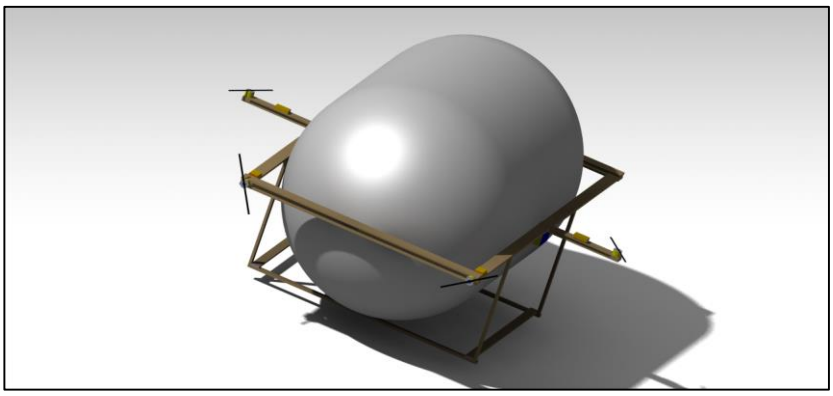

Figura 6.1. Diseño conceptual del prototipo en CATIA [4]

Este diseño es orientativo; se verá en el proceso de construcción que varía con decisiones conjuntas tomadas durante el mismo.

Como se puede observar, la disposición de los motores permiten realizar los mismos movimientos que la aeronave AirWhale: propulsión en X y Z, y giros en X, Y y Z. Los motores con eje vertical permiten controlar la propulsión en Z y el par en X, mientras que los motores con eje horizontal habilitan el control de la propulsión en X y el par en Z. El esquema inicial del prototipo, junto a sus movimientos, se muestra en la Figura 6.2.

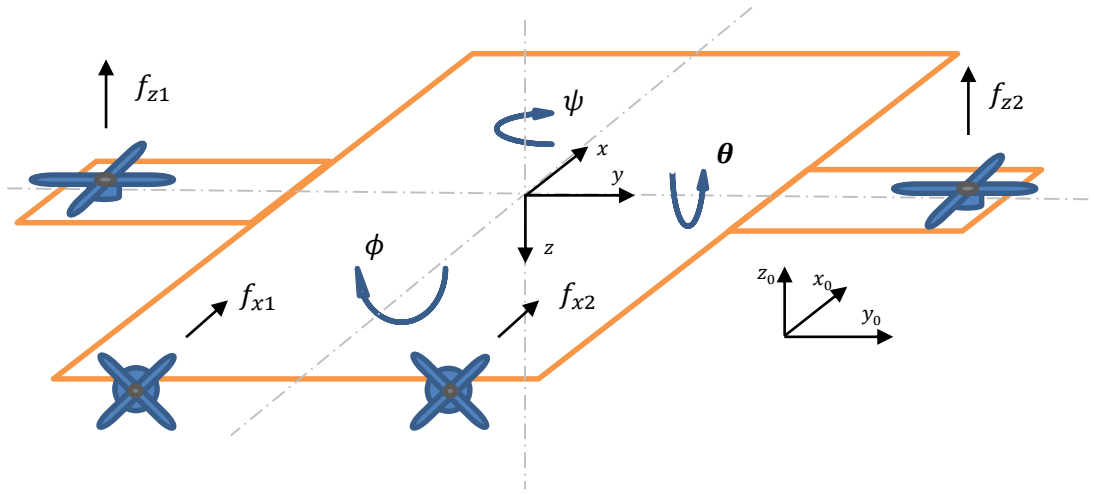

Figura 6.2. Esquema básico e inicial del prototipo

Para el control del par en el eje Y, la primera idea que se barajó es la de usar servomotores conectados físicamente a los motores traseros, de tal forma que, con un sencillo control en posición de ángulo, se pudiese controlar la dirección de la fuerza de dichos motores. Ello conlleva el diseño de un actuador capaz de realizar ese cambio de orientación, y el uso de dos servomotores para convertir en independientes el giro de cada motor. La razón de esto radica en el par de alabeo (par en X) residual que podría aparecer si la aeronave está actuando sobre el par en Z y, a la vez, sobre el par en Y.

Controlar el par en Z supone la desigualdad de fuerzas proporcionadas por los motores traseros, y el control del par en Y supone el cambio de dirección de la fuerza ejercida por dichos motores. El resultado es equivalente a dos fuerzas de componente vertical distintas que generarían un par en X. Este concepto se explica en la Figura 6.3.

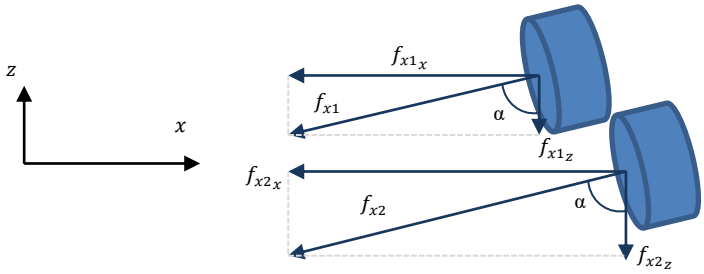

Figura 6.3. Esquema de la causa del par en X no deseable al girar los motores traseros el mismo ángulo

En la figura, la componente f_{x2_z} es mayor que f_{x1_z}, lo que supondría un desequilibrio de pares generados por estas dos fuerzas en el eje X y su consecuente resultante. Para realizar un control deseado del par en Y, las fuerzas en su componente vertical deben ser iguales. El ángulo α representa el ángulo de giro ejercido por los servos, y es el elemento que determina la magnitud de la componente vertical de ambas fuerzas. Para eliminar este efecto, el ángulo de giro en uno y otro motor deben ser distintos. Esto generaría, por otro lado, un

desequilibrio añadido en las componentes horizontales de ambas fuerzas, por lo que el par que se desea generar en Z también se verá afectado, aumentando su valor.

Por dicha complejidad, y por otros motivos que se verán en siguientes secciones, se ha optado por el uso de cuatro motores verticales en sustentación: dos para controlar el par en X (alabeo) y otros dos para controlar el par en Y (cabeceo), como se muestra en la Figura 6.4. El uso de servomotores en los actuadores traseros se pospondrá para trabajos posteriores.

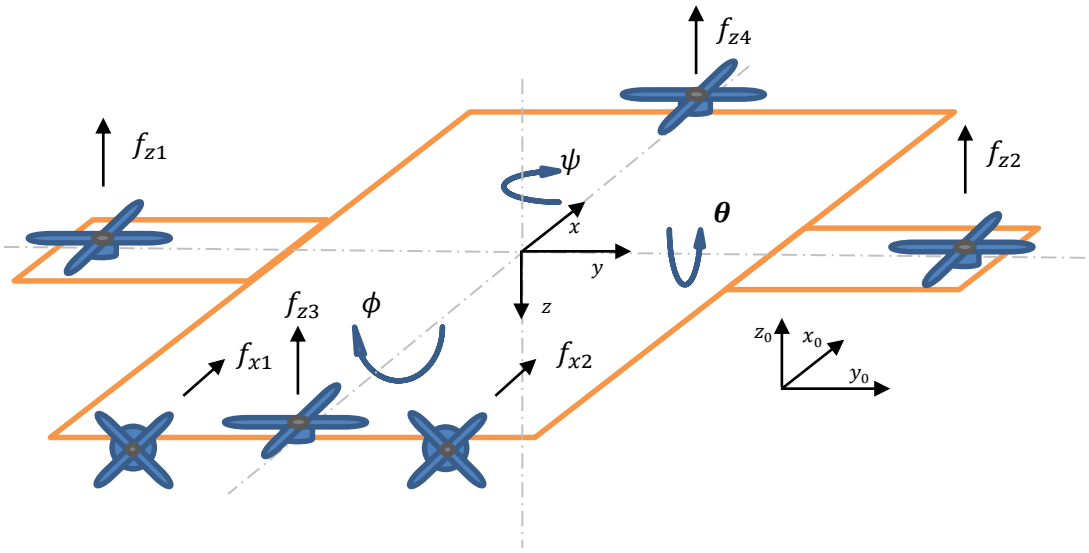

Figura 6.4. Esquema básico y final del prototipo

La relación, por tanto, entre pares y fuerzas de actuación, y las actuaciones individuales de cada motor, se muestra en la agrupación de ecuaciones (1):

$$\begin{aligned} PWM\ motor\ 1 &= \frac{L_1 \cdot U_Z - 2 \cdot U_\theta}{4 \cdot L_1} \\ PWM\ motor\ 2 &= \frac{L_2 \cdot U_Z + 2 \cdot U_\phi}{4 \cdot L_2} \\ PWM\ motor\ 3 &= \frac{L_1 \cdot U_Z + 2 \cdot U_\theta}{4 \cdot L_1} \\ PWM\ motor\ 4 &= \frac{L_2 \cdot U_Z - 2 \cdot U_\phi}{4 \cdot L_2} \end{aligned} \tag{1}$$

Donde L_1 y L_2 son las distancias entre motores en el eje mayor y menor, respectivamente; y U_Z, U_θ y U_ϕ son la fuerza vertical de control, el par sobre el eje Y de actuación y el par sobre el eje X de actuación, respectivamente.

6.2 Materiales de construcción y dispositivos electrónicos a usar

El prototipo está contruido, en su totalidad, con madera de balsa administrado por el Departamento de Ingeniería Aeroespacial y Mecánica de Fluidos (aprovecho para dar las gracias al profesor Sergio Esteban Roncero por su colaboración y comprensión). Las piezas de madera usada cuentan con las siguientes características:

- Madera de balsa sheets de 101.6 x 914.4 x 3.175 mm, de volumen 294.97 cm^3, peso por unidad de 40.44 gramos y 0.1371 g/cm^3 de densidad. Serán las piezas usadas para las superficies mayores de las alas y la plataforma donde se colocará la placa ArduPilot.
- Madera de balsa sheets de 101.6 x 914.4 x 6.35 mm, de volumen 589.93 cm^3, peso por unidad de 76.00 gramos y 0.1288 g/cm^3 de densidad. Serán las piezas usadas para conformar los lados menores de la estructura principal y los listones que compondrán la estructura de apoyo del prototipo.
- Madera de balsa strips de 9.525 x 914.4 x 3.175 mm, de volumen 27.65 cm^3, peso por unidad de 5.85 gramos y 0.2117 g/cm^3 de densidad. Serán, por tanto, los larguerillos usados para colocar en el lado menor, creando el perfil en I, y como superficie menor de las alas.
- Madera de balsa sheets de 76.2 x 914.4 x 9.525 mm, de volumen 663.68 cm^3, peso por unidad de 61.57 gramos y 0.0928 g/cm^3 de densidad. Serán las piezas usadas para formas los lados mayores de la estructura principal.

La electrónica a usar se enumera a continuación, encontrándose una explicación detallada de sus características en el Capítulo 4:

- Plataforma ArduPilot, con módulo 3DR uBlox GPS.
- Variadores EMAX 18 A Budget x 4.
- Motores EMAX BL2210/30 x 4.
- 4 metros de cable AWG para el correcto conexionado entre los variadores y la batería.
- Un batería Zippy 5000mAh.
- Cable fino para la conexión entre variadores y plataforma autopiloto.
- Dos conectores macho y uno hembra T-DEAN.
- Un conector macho XT60.
- Un adaptador T-DEAN – XT60.

La razón por la que se ha escogido una batería en vez de dos es porque asociar una alimentación distinta a cada motor puede llegar a suponer distintos valores de empuje en los mismos. Hay que remarcar que, aunque el fabricante asegure que la batería tiene unas características predefinidas, no tiene por qué ser así: una y otra batería pueden disponer de parámetros distintos, y si esos parámetros (velocidad de descarga, carga…) son cruciales en vistas a la potencia proporcionada a los motores, los resultados en estabilización y vuelo pueden ser dispares. Por ello, se ha considerado el uso de una única alimentación, igualando las condiciones de alimentación para cada motor y eliminando ese posible problema.

También se ha prescindido de la mayoría de elementos de conexionado, para facilitar el mismo. En este sentido, se han usado dos conectores macho-hembra T-DEAN para la conexión de alimentación del variador alimentador (se verá más adelante) y un conector macho para la conexión del cableado de alimentación con la batería. El adaptador T-DEAN – XT60 supone la unión entre el conector XT60 macho soldado a los cables de la batería y el conector T-DEAN del cableado de alimentación. Se verá con más detalle en la sección de conexionado.

6.3 Proceso de construcción

El prototipo ha sido construido a lo largo de varios meses. El diseño del mismo, realizado en CATIA, dista bastante del vehículo elaborado finalmente, debido a decisiones que se han ido tomando a lo largo del proceso.

Por otro lado, las dificultades que se han ido encontrando a lo largo de los meses han afectado directa o indirectamente a la forma final del prototipo. Todo ello se expone a continuación, en forma de pasos cronológicos:

- En primer lugar, se midieron las longitudes de los listones de madera necesarios para los soportes principales del prototipo y las alas del mismo, donde se alojan dos de los motores. Las alas están compuestas de láminas de madera X (área superior e inferior) y de varillas de madera X. Los soportes principales están compuestos de dos listones de madera X (lado más largo) y de dos listones de madera X unidos por un larguerillo X, en forma de perfil en I.
- Una vez medidas y cortadas, se ensamblan las maderas para crear los soportes principales y las alas. Las láminas de madera que conforman las alas se unen a los dos listones que formarán el lado superior del prototipo asociado a dicho ala, por la parte superior e inferior de los listones. En el caso de los lados menores, las superficies superior e inferior del perfil en I creado se unen a la superficie superior e inferior del lado mayor.

Figura 6.5. Construcción del ala

Figura 6.6. Medidas de los dos lados superiores con las alas colocadas

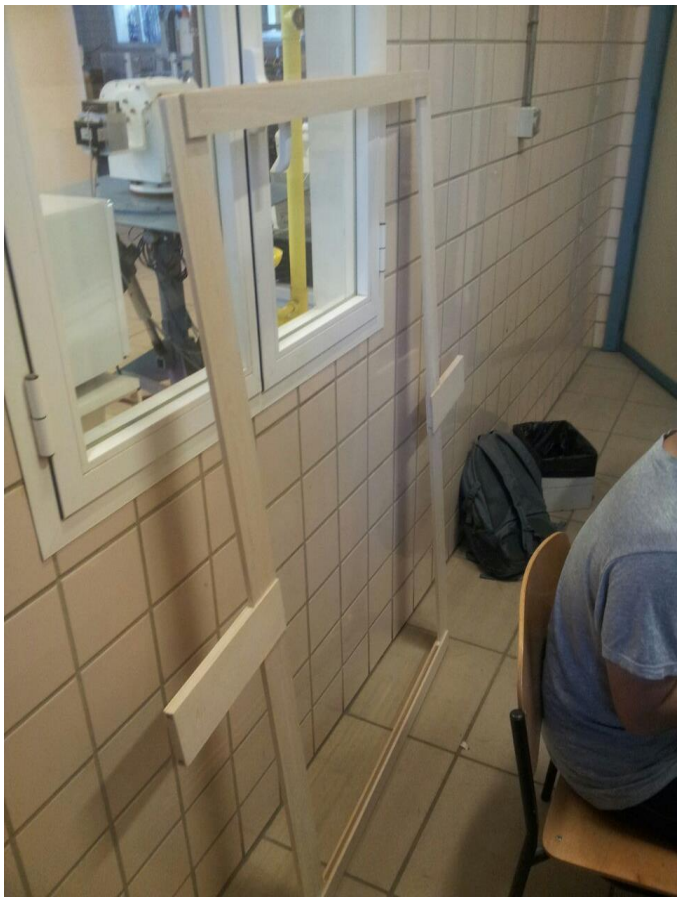

Figura 6.7. Estructura principal

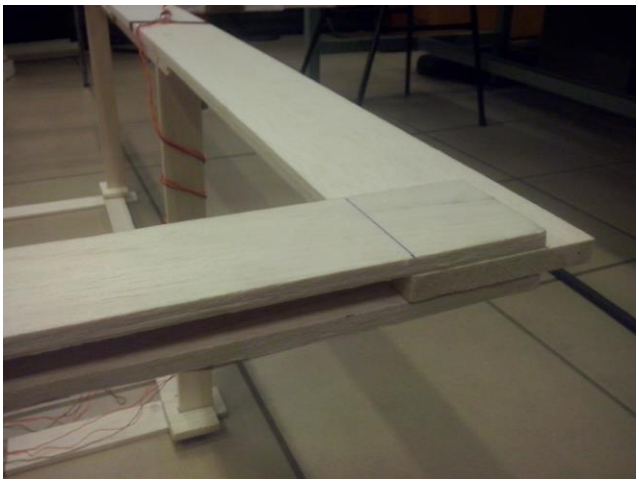

Figura 6.8. Detalle de la unión entre lado menor y mayor de la estructura principal

- Posteriormente, y ya obtenida la estructura principal del prototipo (la que confina el globo), se miden y se cortan las maderas necesarias para el soporte del prototipo en el suelo, que evitará que el globo entre en contacto con el mismo.
- Por otro lado, se colocan los dos motores laterales en las alas. Estos se atornillan lo más cerca del extremo del ala posible, para crear el máximo par, y coincidentes con el eje menor del prototipo para no crear un par espúreo en Y.
- Se unen los soportes del prototipo a la estructura principal que confina el globo, usando una pequeña

pieza de madera con una muesca realizada por los alumnos para reforzar la unión aumentando la superficie de adherencia.

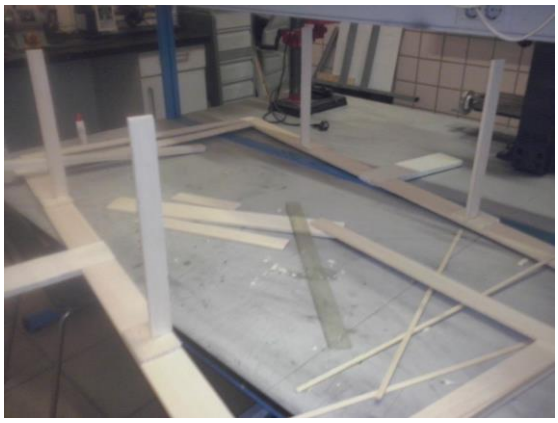

Figura 6.9. Soportes verticales colocados en la estructura principal

Figura 6.10. Trozo de madera con muesca indicada, usada para la unión de los soportes

Figura 6.11. Detalle de la unión de los soportes con la estructura principal

- Mediante uniones entrelazadas se fabrica una plataforma inferior para el fácil apoyo de la aeronave en el suelo.

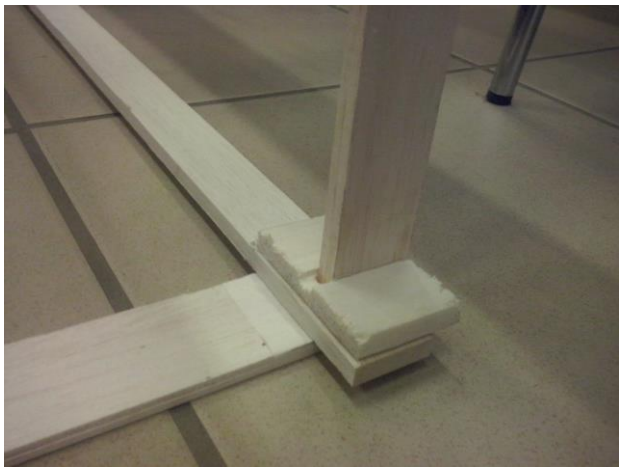

Figura 6.12. Detalle de la unión de la plataforma inferior con los soportes (cara interior)

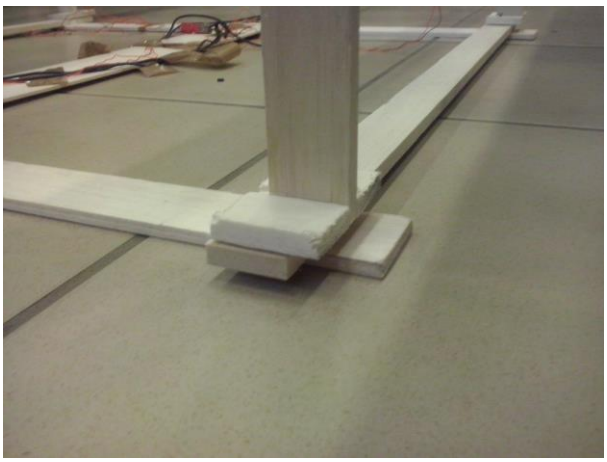

Figura 6.13. Detalle de la unión de la plataforma inferior con los soportes (cara exterior)

- Se colocan los dos motores verticales restantes en el centro de cada lado menor de la estructura principal, evitando la aparición de un par espúreo en X.

Los problemas que se han ido afrontando conforme se realizaban experimentos o se construía son los siguientes:

- El principal problema, a tener mucho en cuenta, fue un daño a la estructura principal que se descubrió al empezar a trabajar en el prototipo al comenzar el mes de agosto. El prototipo aparecía con una fisura en una de las alas que la dividía en dos casi por la mitad, no realizada por los alumnos asociados al trabajo, evidentemente. La figura 6.15 muestra el estado del ala después de la fisura descubierta. Esto ha traido consecuencias a mencionar, tales como el descenso de la resistencia a fuerzas de dicha ala y, en consonancia, la excesiva deformación por el esfuerzo axil producido por el motor actuador situado en dicha ala, con el efecto que ello trae en la actuación de dicho motor (variación de par no deseada).

Figura 6.14. Fisura encontrada al retomar la construcción del prototipo

Figura 6.15. Ala fijada de nuevo y reforzada

- En consecuencia de lo anterior, se observó que el listón perteneciente a ese ala también había perdido resistencia frente a las fuerzas ejercidas por los motores verticales situados en el eje mayor (lado menor de la estructura), por lo que se optó por colocar en la cara inferior de dicho listón, y en la del otro para compensar pesos, una lámina de madera, formando un perfil en T y aumentando la resistencia de los dos listones ante esfuerzos comprometedores.

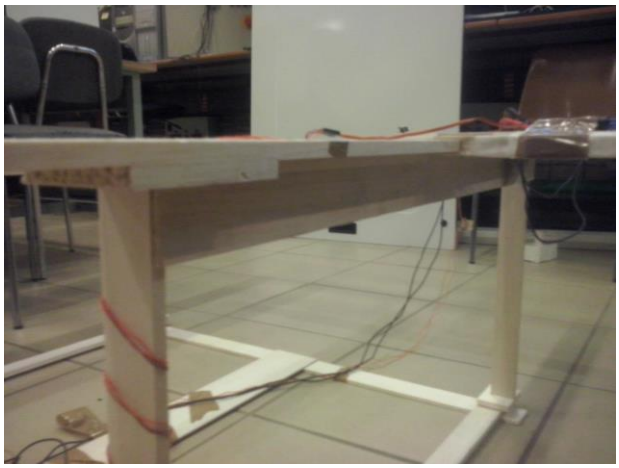

Figura 6.16. Listón colocado para formar un perfil en T junto con los listones del lado mayor

- Un fallo evidente, y que se observó bien avanzado el proceso de construcción, fue la excesiva superficie del ala, que restaba gran parte del empuje producido por los motores situados en cada una. La solución pasó por usar una sierra y cortar las alas, reduciendo el ancho de las mismas para aumentar el empuje de los actuadores.

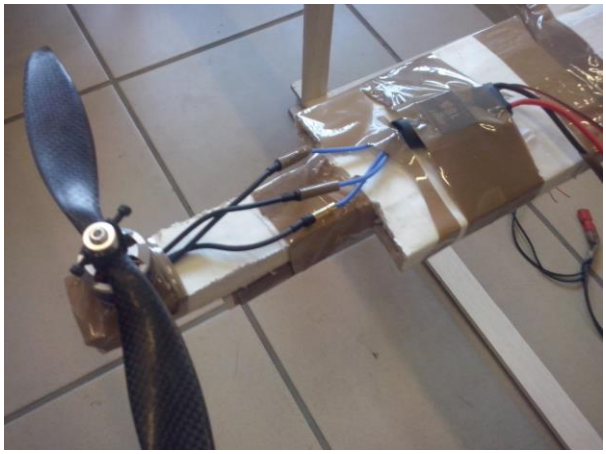

Figura 6.17. Corte realizado en el ala para eliminar la pérdida de sustentación aportada por el motor

- Los motores verticales que contribuyen al par en Y no podían mantenerse atornillados a los lados menores de la estructura principal, ya que no era posible ponerlos en marcha sin que la hélice de fibra de carbono tocase el globo confinado, por lo que se optó por usar unas superficies de madera introducidas en el hueco del perfil en Y, reforzando el lado menor ante las torsiones producidas por la actuación de los motores en su nueva configuración con un taco de madera colocado en el hueco del perfil en Y de la cara inferior.

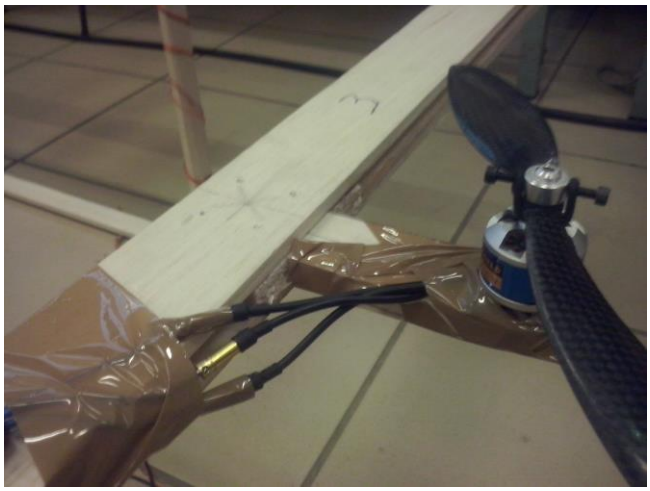

Figura 6.18. Plataforma de madera usada para albergar los motores verticales contribuyentes al par en Y. Nótese la marca de la posición anterior de dichos motores.

- Se olvidó un lugar para poder colocar la plataforma Ardupilot y la batería única de alimentación, el cual se añadió con una lámina fina de madera colocada en la plataforma inferior de apoyo. Se puede observar en la Figura 6.16.

El resultado final se muestra en la Figura 6.19.

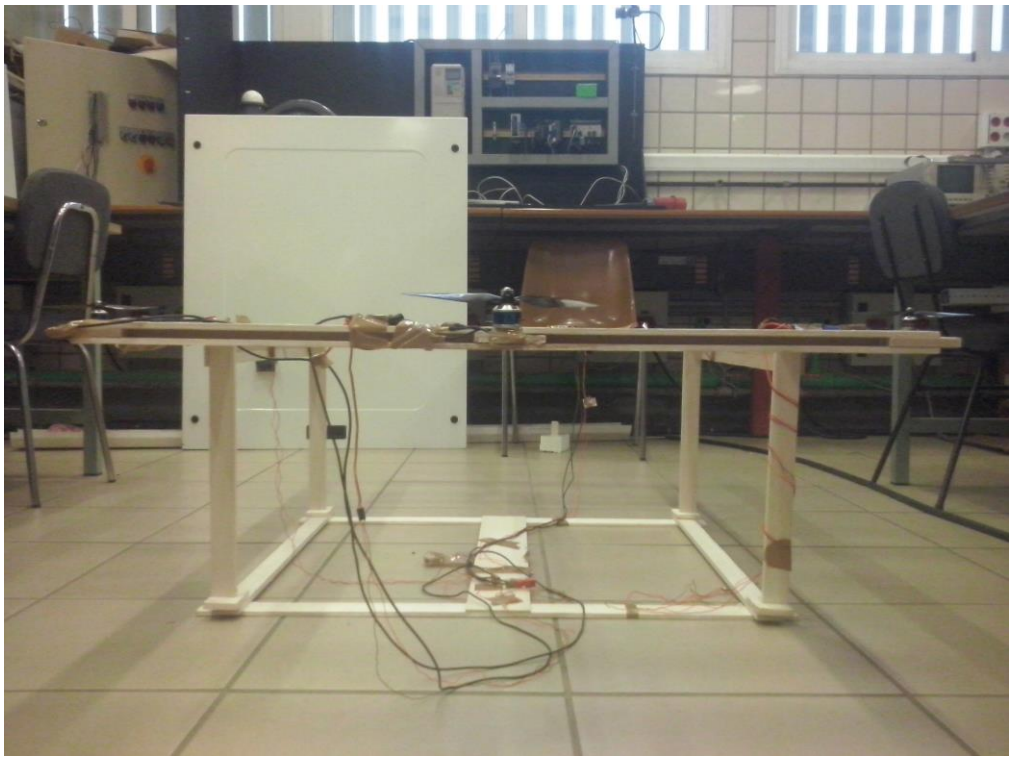

Figura 6.19. Prototipo construido

6.4 Conexionado

La conexión de los distintos componentes electrónicos con la alimentación se ha realizado, en su totalidad, mediante empalmes en paralelo entre los variadores con la batería.

Los polos positivos de cada uno de los variadores se sueldan a un cable del mismo calibre, y lo mismo se hace con los polos negativos. Cada uno de los cables coincidentes en polaridad se suelda a un cable común, que se unirá a uno de los polos de un terminador T-DEAN macho. Este terminador se engancha a un conversor T-DEAN - XT60, que será el enganche final con la batería, colocada en el centro inferior de la estructura. Esta conexión reparte la potencia suministrada por la batería entre los cuatro variadores y motores a usar en las primeras pruebas. También se dispone de conectores T-DEAN macho-hembra para la conexión y desconexión del variador que alimentará, específicamente, a los demás (mediante la plataforma autopiloto). Un esquema se puede revisar en la Figura 6.20.

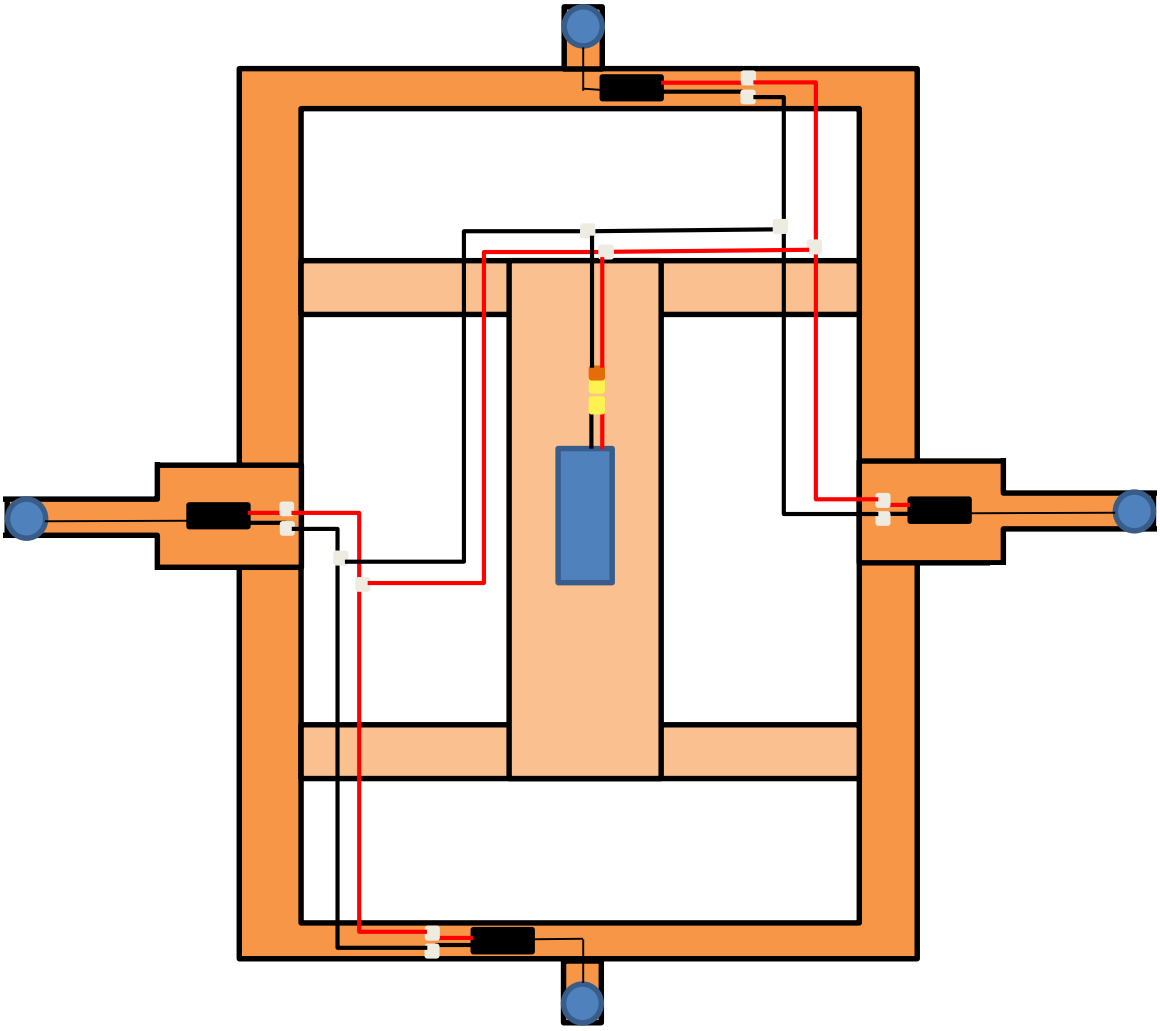

Figura 6.20. Conexionado de los variadores y motores en paralelo con la batería

El conexionado de los motores con la plataforma ArduPilot se realiza a través de los cables de señal y alimentación (BEC) de los variadores. En el caso que nos ocupa, todos los variadores se van a alimentar mediante el BEC de uno de ellos. Para ello, se conectan los tres cables de uno de los variadores (señal, alimentación y tierra) a una de las ternas de pines de salida del ArduPilot. Los demás variadores se conectarán a la plataforma mediante el cable de señal y de tierra, omitiendo la conexión de la alimentación. Hay que remarcar que la plataforma se alimentará de forma exclusiva mediante puerto USB, por lo que no se debe colocar el *jumper* que permite la alimentación de la placa mediante el BEC del variador (ello produciría daños potencialmente

peligrosos en la toma del USB). En la Figura 6.21 se puede observar cómo existen tres conexiones a través de dos cables y una sólo con tres cables (alimentación BEC).

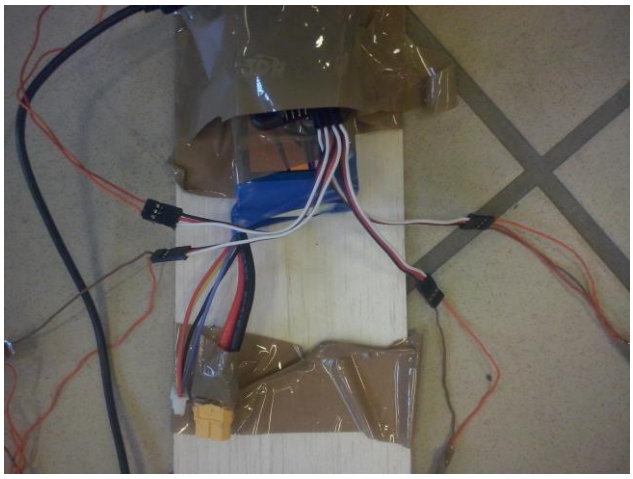

Figura 6.21. Conexionado placa-variadores

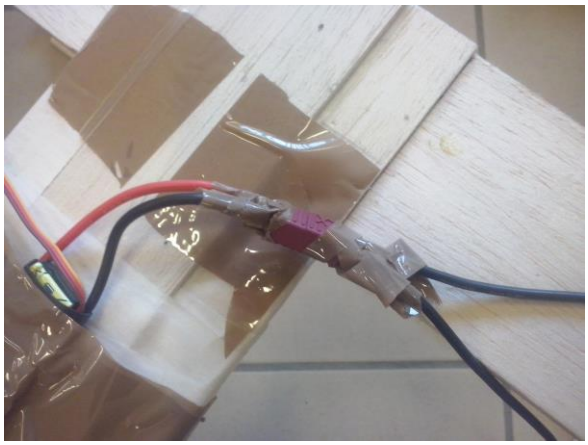

Figura 6.22. Conexión T-DEAN para la alimentación del variador alimentador por BEC.

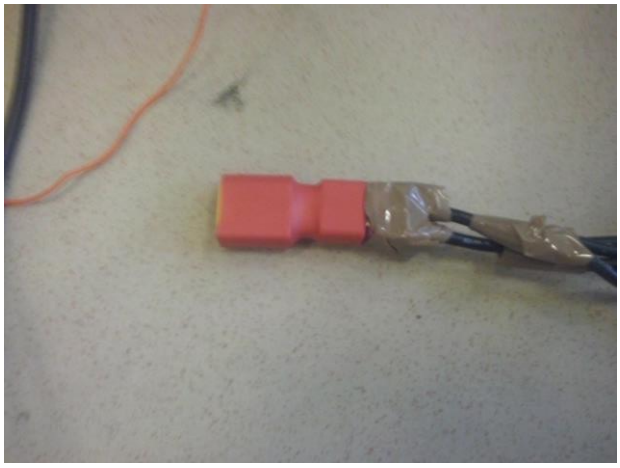

Figura 6.23. Adaptador XT60 – T-DEAN conectado al cableado de alimentación

6.5 Experimentos realizados

Una vez se dispone del prototipo finalmente construido y con sus elementos electrónicos adecuadamente conectados, se exponen a continuación los distintos experimentos realizados con el mismo. Estos abarcan varios aspectos imprescindibles para el correcto vuelo de la aeronave, tales como la estabilización estática, identificación del sistema real para el uso de un controlador apropiado para el prototipo y estabilización dinámica.

6.5.1 Estabilización estática

La estabilización de pesos del prototipo es un concepto muy importante si tenemos en cuenta la estabilización obligada de los ángulos de orientación de la aeronave en vuelo. Si estos ángulos tienen, de partida, un valor no corregido, se puede experimentar desviaciones indeseadas en el despegue y vuelo del vehículo.

Para conseguir una estabilización estática, se ha fabricado un arcaico sistema para poder suspender la aeronave sobre dos puntos colocados en los extremos de sus dos ejes de simetría, como se puede observar en la figura.

Figura 6.24. Experimentos realizados para la estabilización estática del vehículo sobre el eje Y

En la imagen, el prototipo se suspende sobre su eje Y para observar qué desequilibrio de pesos existe inherente al mismo. Como puede comprobarse, el prototipo no se mantenía con un ángulo nulo sostenido por su eje Y, por lo que se optó por situar las baterías laterales un poco más hacia la izquierda. Cabe anotar que, aunque pueda parecer contradictorio el hecho de usar dos baterías teniendo en cuenta el conexionado anteriormente explicado, al introducir la electrónica usada ya se puntualizó el motivo por el que se rechazó la idea de usar dos baterías en vez de una. Colocadas las baterías correctamente, el vehículo se estabilizó.

Figura 6.25. Estabilización sobre el eje Y conseguida

Para la estabilización del ángulo relacionado con el eje X, se colocó el prototipo suspendido sobre su eje X, como se observa en la siguiente figura. Colocando un peso debajo del ala del lado levantado para igualar el exceso del otro lado, se consigue la estabilización.

Figura 6.26. Experimento realizado para la estabilización estática sobre el eje X

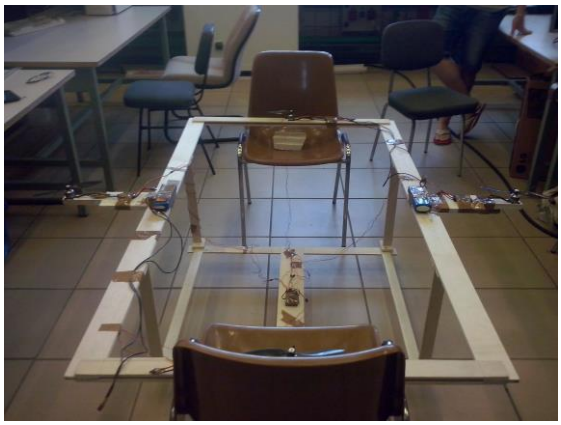

Figura 6.27. Estabilización estática sobre eje X conseguida

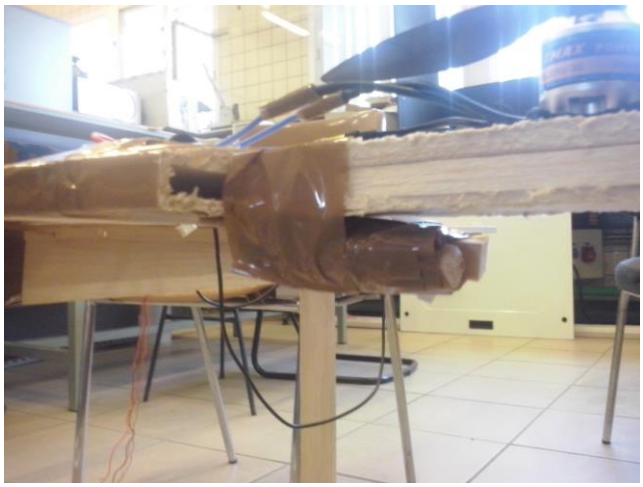

Figura 6.28. Peso colocado debajo del ala para la estabilización

6.5.2 Identificación del prototipo: sistema real

La necesidad de una identificación en bucle abierto de la dinámica que ofrece el prototipo en su movimiento por la acción de pares es evidente. Se hizo patente al observar que la implementación de los controladores diseñados en el trabajo [4] era imposible, debido a que, al ser diseñados sobre un modelo teórico sin ruido, suponía un control muy rápido y brusco incapaz de controlar un sistema con ruido sin saturar constantemente. Es obvio que existen multitud de diferencias entre los modelos teóricos desarrollados y el modelo real representado por el prototipo, por lo que se necesitaba una representación del mismo en forma de función de transferencia para poder diseñar y aplicar controladores más conservadores y cercanos al sistema real.

Por otro lado, el tratamiento de dichos controladores en términos de fuerza suponía una dificultad añadida a la hora de la implementación en el código desarrollado, ya que los valores enviados a los motores están en términos de PWM. En primera instancia, se barajó la idea de encontrar la característica estática entre el PWM que se mandaba a un motor y la fuerza que ejercía. Esta idea se simplificó a encontrar dos puntos de dicha característica y suponer una recta como relación entre PWM-fuerza. Este método no dio grandes resultados, por lo que identificar el sistema mediante una función de transferencia donde la entrada es PWM y la salida son los ángulos en cuestión parece más que adecuado.

Para ello, se realizaron dos experimentos: la aplicación de un par escogido sobre el eje X del prototipo, estando este suspendido sobre dicho eje, y el proceso análogo con el eje Y. Para realizarlo, se modificó el código *test_sensores_actuadores.pde*, de tal manera que, en el test de motores, se pudiese dar un valor PWM a cada motor y mantener el valor de cada uno durante un tiempo determinado, mientras se imprimía por pantalla el valor de los ángulos roll y pitch.

- Identificación de la dinámica del prototipo par en X - ángulo roll: El par escogido de actuación será de 50 PWM, que, por las relaciones indicadas en (1), supone 36 y -36 PWM en los motores implicados en la aplicación de dicho par. Sumando los valores indicados a un valor inicial enviado a los motores implicados (1200 PWM) y leyendo en cada instante de muestreo (50 ms cada uno) el ángulo roll, se obtiene el siguiente resultado:

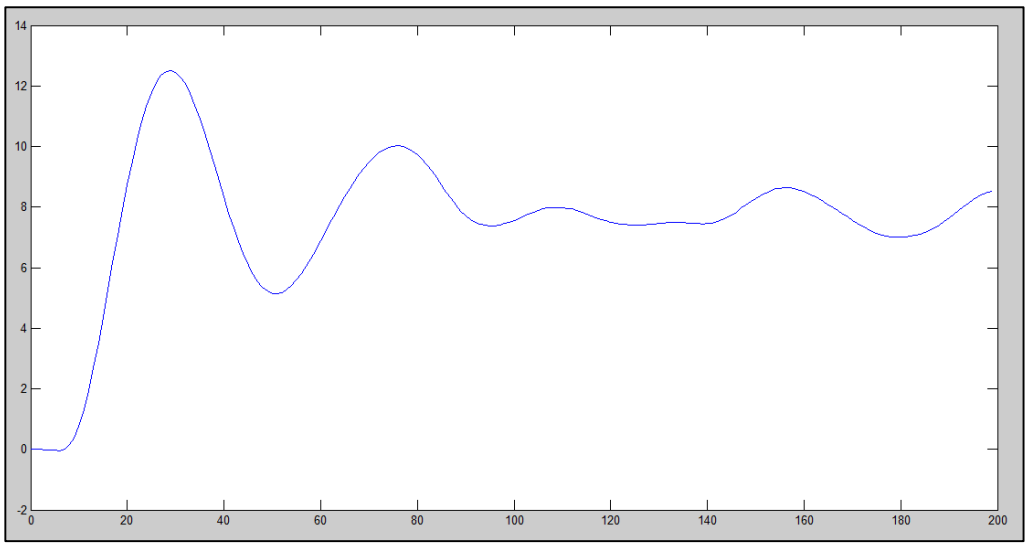

Figura 6.29. Representación del valor del ángulo roll ante la aplicación en bucle abierto de un par en X

Se puede observar su gran parecido con un sistema de segundo orden subamortiguado que tiende al valor de siete grados, aproximadamente. La añadida oscilación del final se debe a una ráfaga de aire.

- Identificación de la dinámica del prototipo par en Y - ángulo pitch: El par escogido de actuación será de 60 PWM, que, por las relaciones indicadas en (1), supone 37.5 y -37.5 PWM en los motores implicados en la aplicación de dicho par. De forma análoga al anterior caso, pero con el ángulo pitch, se obtiene el siguiente resultado:

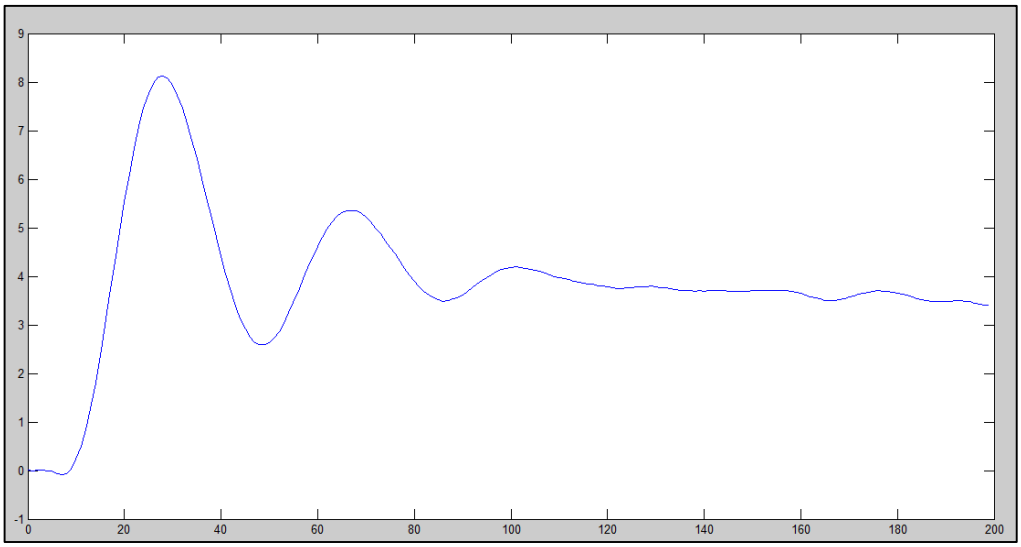

Figura 6.30. Representación del valor del ángulo pitch ante la aplicación en bucle abierto de un par en Y

El comportamiento de la respuesta se asemeja bastante a la obtenida en el caso del roll, pero con magnitudes

inferiores. La perturbación, en este caso, parece reducir un poco el valor en régimen permanente de la respuesta.

El cálculo de las funciones de transferencia y de los controladores a aplicar se puede revisar en el trabajo [4].

6.5.3 Estabilización dinámica de los ángulos roll y pitch

El anterior experimento arroja, finalmente, unos controladores adecuados para el sistema real que representa el prototipo.

Aplicando los parámetros de los controladores hallados en el código *sintonizacion.pde* (usado de forma íntegra, sin modificaciones), y dándoles un valor inicial de PWM a los motores de 1200, se observan los siguientes resultados, añadiendo, además, sendas perturbaciones al control:

- Estabilización del ángulo roll:

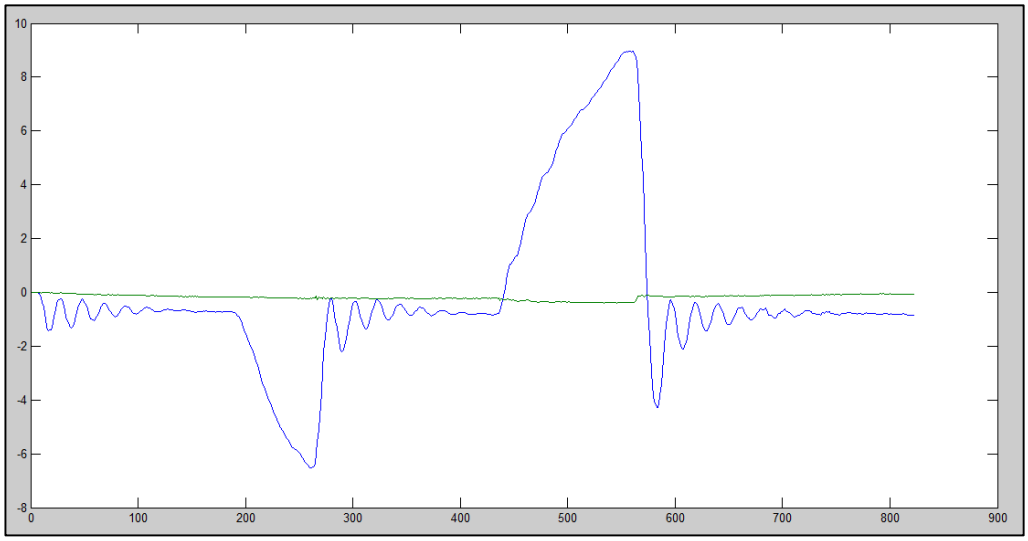

Figura 6.31. Evolución del ángulo roll y pitch durante la estabilización del ángulo roll y perturbaciones

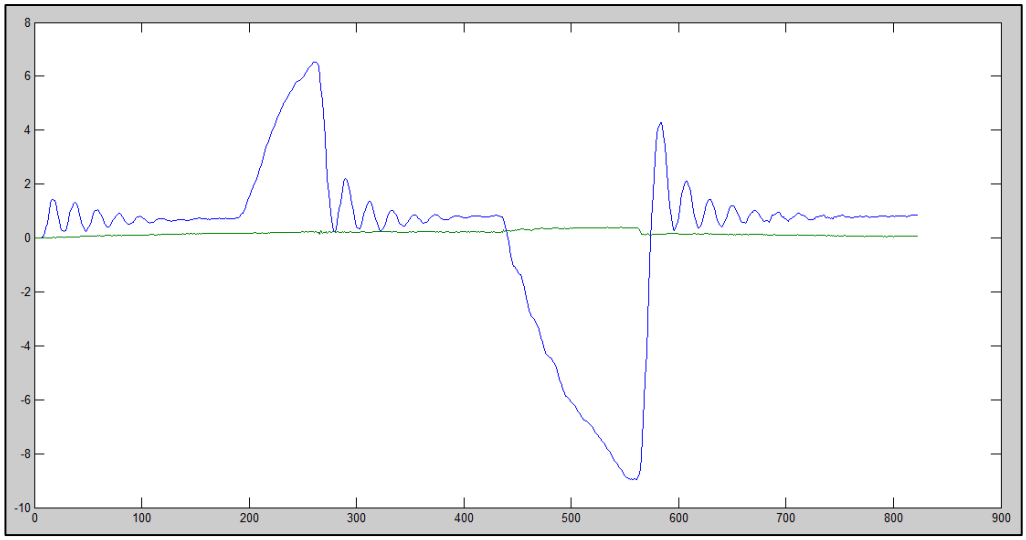

Figura 6.32. Evolución del error de los ángulos con respecto a la referencia durante la estabilización del ángulo roll

La estabilización del ángulo roll es lenta, como puede comprobarse, encontrándose una sobreoscilación cercana al 50% y un tiempo de establecimiento superior a los 100 instantes de muestreo. También se puede observar que estos valores aumentan ante una perturbación como la que se aparece a los 200 y 450 instantes de muestreo. Por otro lado, no se observa que el valor del ángulo iguale la referencia nula que se le ha impuesto. Estas conclusiones no son positivas en vistas de un futuro vuelo.

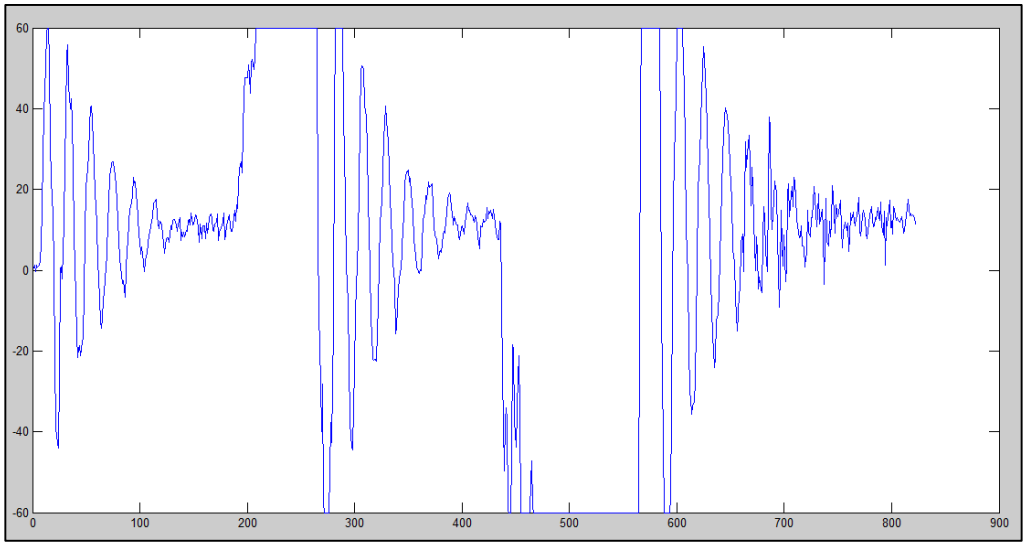

Figura 6.33. Par de actuación durante la estabilización del ángulo roll y perturbaciones

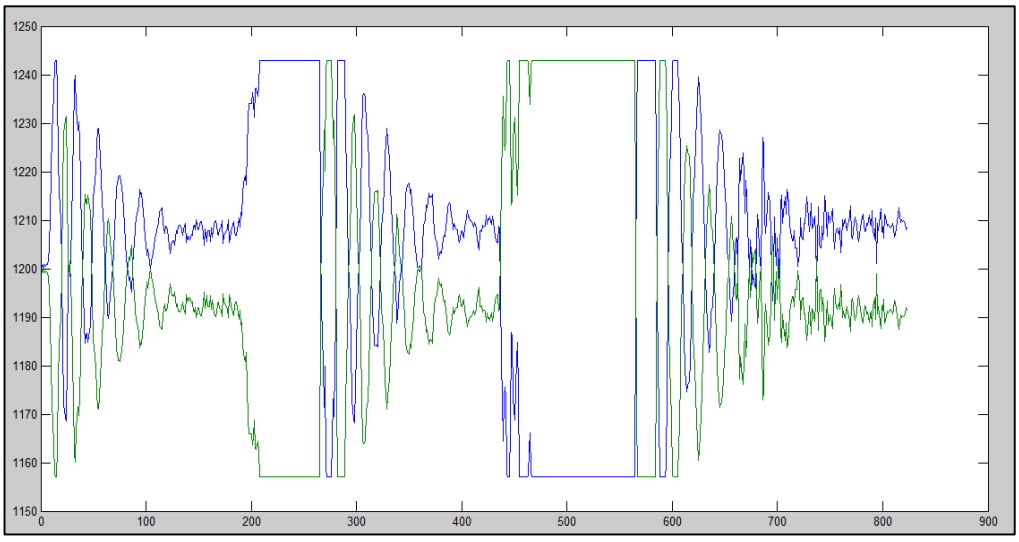

Figura 6.34. PWM mandados a los motores laterales durante la estabilización del ángulo roll

Los valores más altos de las actuaciones coinciden, obviamente, con los momentos en los que existe una gran perturbación. Conforme se va llegando a la referencia, las actuaciones disminuyen a valores aproximadamente de 15-16 PWM, en el caso del par de actuación, y 10 PWM para los motores individuales; intentarán mantener ese valor debido a la referencia distinta de cero. La saturación implementada a los actuadores son 60 y -60 PWM para el par de actuación, que se traduce en 43 y -43 PWM para cada motor.

- Estabilización del ángulo pitch:

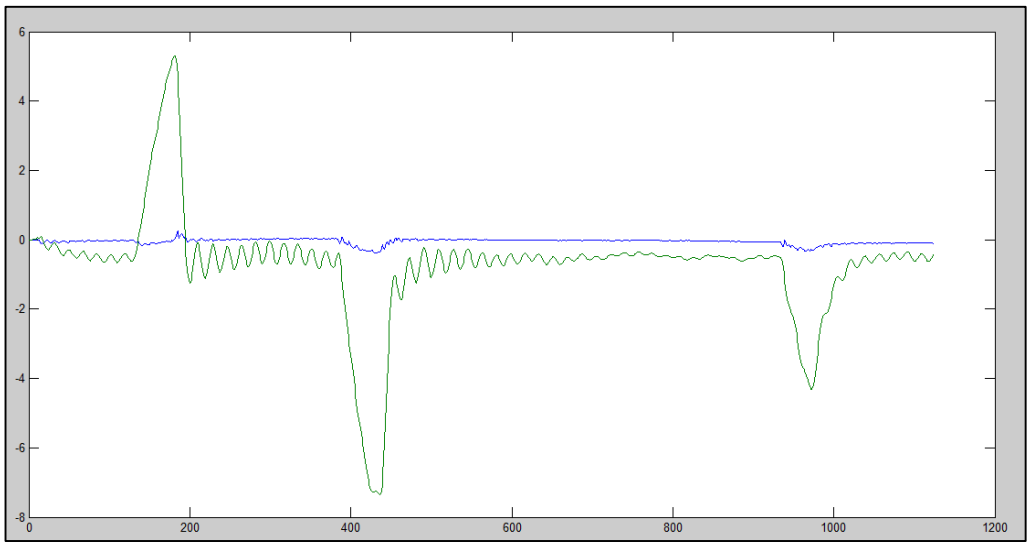

Figura 6.35. Evolución de los ángulos roll y pitch en la estabilización del ángulo pitch y perturbaciones

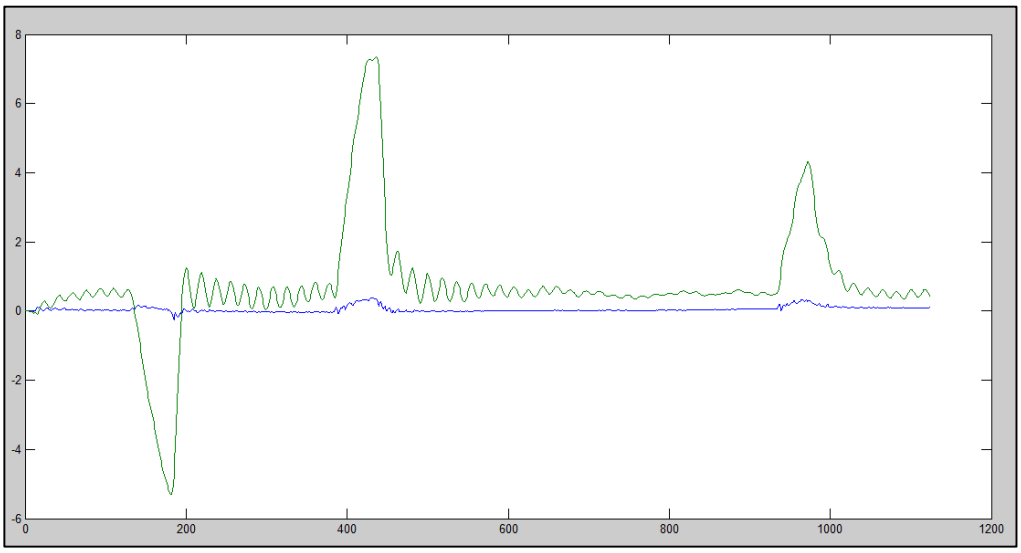

Figura 6.36. Evolución del error de los ángulos con respecto a la referencia durante la estabilización del ángulo pitch

El ángulo pitch, en este caso, presenta menos sobreoscilación, pero un mayor tiempo de establecimiento (alrededor de 250 instantes de muestreo). Tampoco llega a valer cero, quedándose en cerca de un grado de error con la referencia. Suponen resultados bastante mejorables.

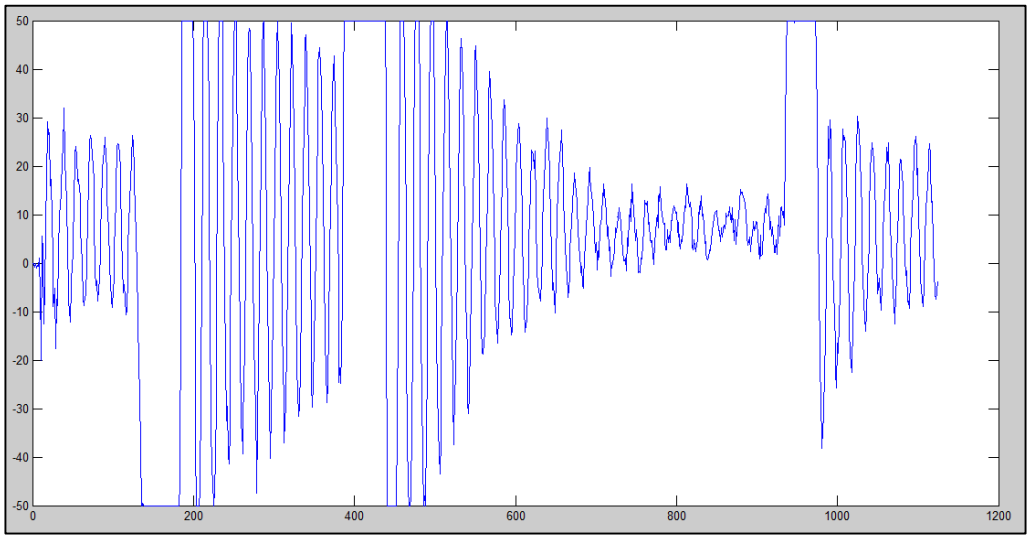

Figura 6.37. Par de actuación durante la estabilización del ángulo pitch y perturbaciones

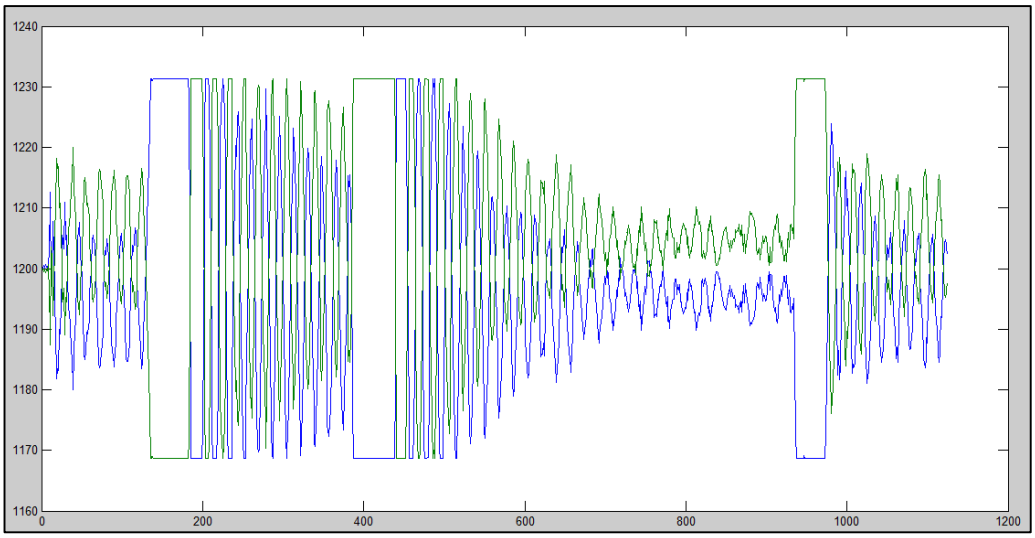

Figura 6.38. PWM mandados a los motores laterales durante la estabilización del ángulo pitch

Las actuaciones siguen el mismo comportamiento mostrado en la anterior estabilización, siendo, en este caso, menor la velocidad con la que disminuye el valor PWM de las mismas (mayor sobreoscilación, mayor tiempo de establecimiento). Como puede comprobarse aquí también, se mantiene un valor residual debido al error con la referencia. La saturación del par en -50 – 50 PWM supone una saturación en la actuación individual de los motores de -33 – 33 PWM aproximadamente.

En todos los experimentos se aprecia el efecto positivo del anti Wind-Up. La saturación de las actuaciones está presente en gran parte del experimento, no apreciándose apenas un retardo provocado por la integración del error en saturación. El sistema programado en el archivo *sintonizacion.pde* ofrece, pues, buenos resultados.

Las estabilizaciones de los ángulos relacionadas con estos experimentos se pueden observar en los vídeos adjuntos a la memoria escrita.

6.5.4 Comprobación de la dinámica ante un escalón dado en la fuerza de actuación vertical

Debido a los problemas encontrados al intentar volar el prototipo con el control de ángulos activo, se vio necesaria la experimentación del comportamiento de la aeronave ante un escalón dado en la actuación en dirección Z. Este experimento es el más cercano al vuelo controlado que se ha realizado durante el trabajo presentado. Desgraciadamente, no se ha podido controlar la aeronave en un vuelo de forma decente: no se encontraba un control de ángulos efectivo, como se ha comentado en la anterior subsección. La aeronvae, al implementar la fuerza de actuación en Z, se descontrolaba bastante, haciendo imposible el vuelo normal de la misma.

Dicho esto, se presentan los resultados de un experimento donde se aplica un escalón de 137.5 PWM a la actuación vertical (1750 PWM a cada motor individual, partiendo de un valor inicial de la actuación individual de 1200 PWM) y se controlan los ángulos, donde, como se puede ver en el vídeo adjunto, dos alumnos sujetan la aeronave para que la misma no se descontrole y se convierta en un elemento peligroso para las personas cercanas.

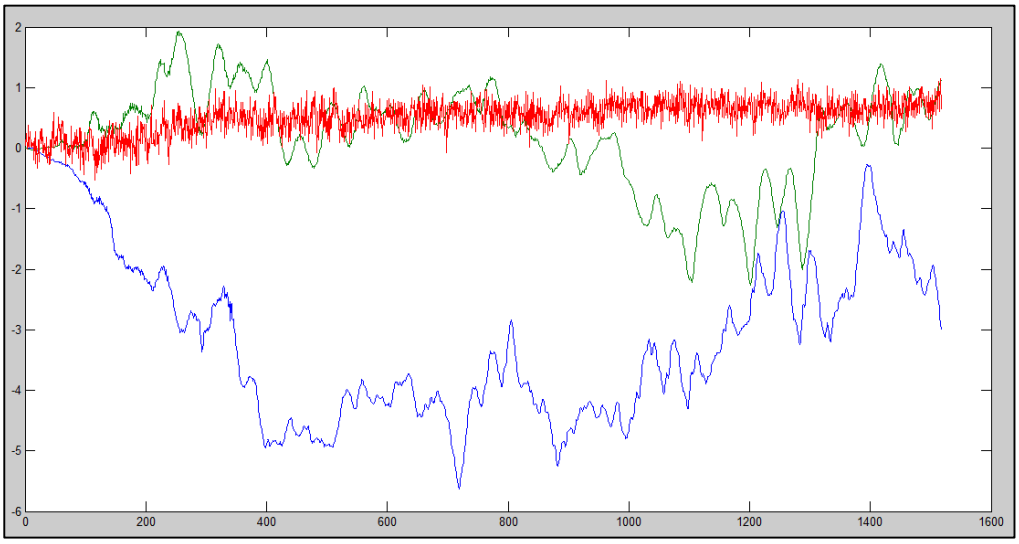

Figura 6.39. Evolución de los ángulos roll (azul) y pitch (verde), y de la altitud (rojo)

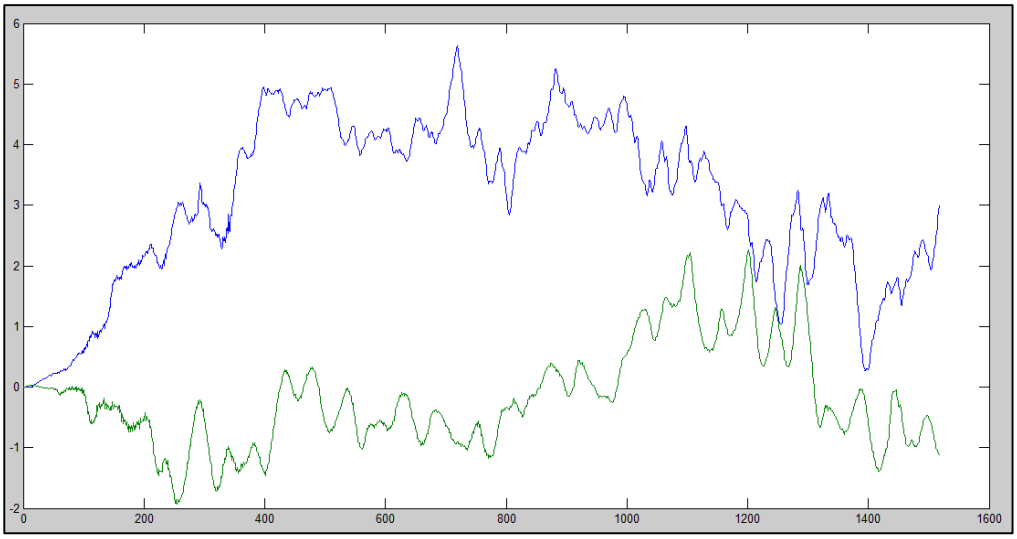

Figura 6.40. Evolución del error con respecto a la referencia de los ángulos roll y pitch

Los ángulos disponen de una evolución irregular debida a la acción de los alumnos que sujetan la aeronave. Por otro lado, la altitud, que posee bastante ruido, sube conforme el experimento avanza, teniendo sentido con lo observable en el vídeo. El error con respecto a la referencia (nulas para el control de los ángulos) es bastante grande, debido al componente humano ya mencionado.

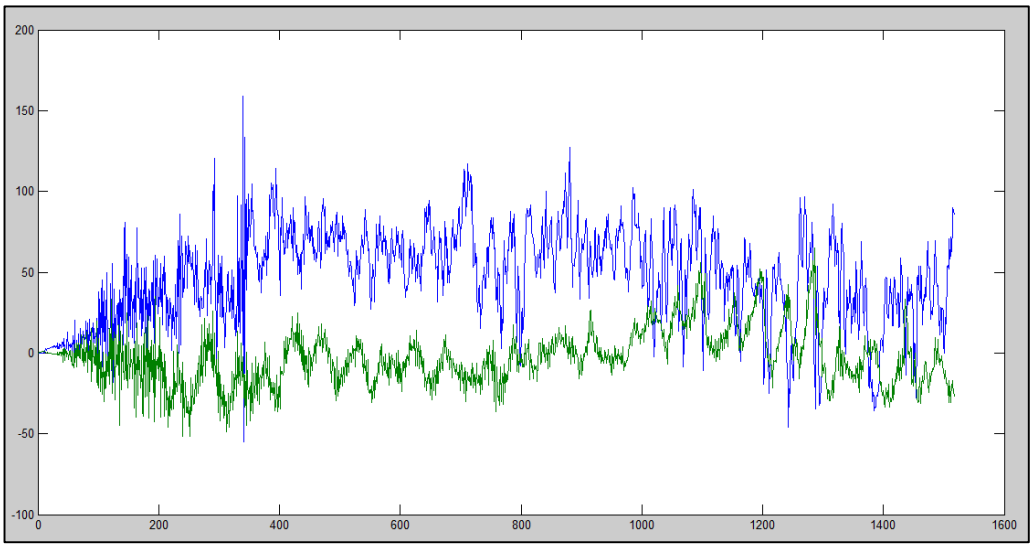

Figura 6.41. Evolución de los pares de actuación asociados al control del roll (azul) y del pitch (verde)

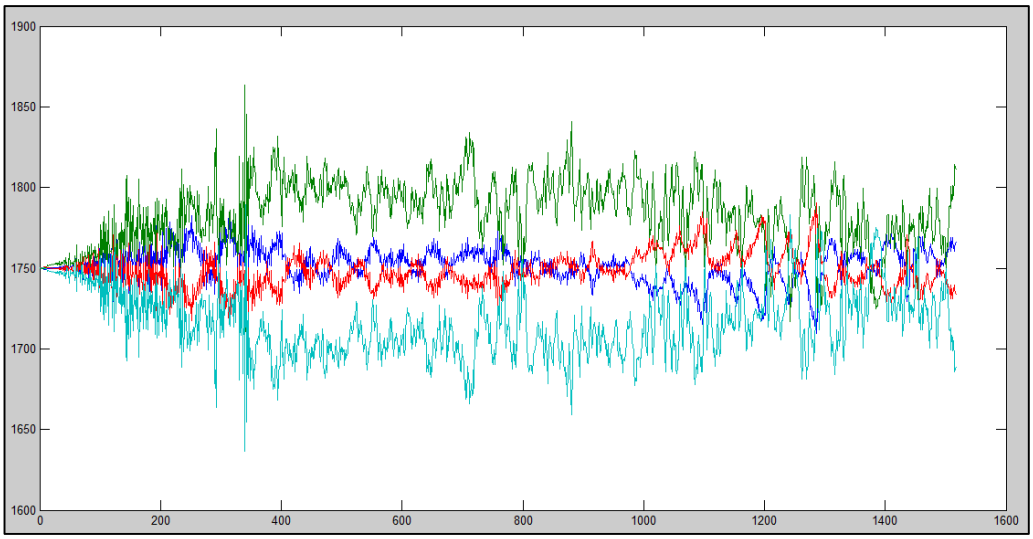

Figura 6.42. Evolución de las actuaciones individuales de cada uno de los cuatro motores: dos motores contribuyentes al par en X (verde y cián) y dos motores contribuyentes al par en Y (azul y rojo)

La actuación evoluciona de forma concordante con el error respecto a la referencia nula, siendo bastante irregular. Los picos de PWM observados en las actuaciones individuales coinciden con las variaciones máximas con respecto a la referencia que se observan en la Figura 6.40. Se puede observar que el valor al que tendrían que tender las actuaciones individuales es 1750 PWM, donde inician la evolución todas ellas. Aclarar que no se ha incluido una saturación, y que el código *sintonizacion.pde* se ha modificado para este experimento, permitiendo introducir el escalón en la fuerza vertical de actuación.

7 CONCLUSIONES Y POSIBLES TRABAJOS FUTUROS

"El 90% del éxito se basa, simplemente, en insistir"

Woody Allen

Cuando se ofreció al alumno que escribe este trabajo la oportunidad de poder realizar algo de dimensiones tan trascendentes, en muchos sentidos, no pudo, sino, unirse a la aventura. No se necesitaba nada más que ganas de trabajar, aprender y conocer qué se siente al cooperar con un equipo de grandes estudiantes (y mejores personas) con un fin y, a veces, enemigo común. Algo así no se enseña. Algo así debe vivirse.

El trabajo en equipo es, más que propicio, necesario. Un proyecto que nació hará un año, a día de hoy va cogiendo forma y entrando en su mediana edad: la del montaje. Muchos de los integrantes de la asociación EsiTech hemos confirmado la intención de continuar para ver volar a nuestro querido AirWhale, pero nunca podríamos haber pensado en tales miras sin habernos apoyado los unos en los otros, tanto en las ideas brillantes como en los desastres más sonoros. Somos parte de un todo, y si cae uno, caemos con él. Ha sido una experiencia muy reveladora a nivel académico y, con la mente en el futuro, profesional. Recomiendo a todo estudiante que quiera enfrentarse a un reto de estas magnitudes, aceptar; merece la pena.

La ciencia de los UAVs está en imparable auge. Muchos son los que piensan ya en futuros donde nada ni nadie puede escapar del ojo de estos bichos; EsiTech cree en las ventajas que aguardan los drones. Todo el trabajo realizado durante el presente curso es la primera piedra del camino a seguir para conseguir ver nuestro vehículo surcando los cielos. Falta mucho, y no será fácil, pero esperamos incorporar un mayor número de estudiantes interesados en ayudar en la causa de construir el AirWhale o dispuestos a aportar nuevas ideas para próximas líneas de trabajo.

En cuanto al trabajo del equipo Automático, hay mucho por hacer. Los controladores mostrados en los últimos capítulos, que controlan los ángulos de orientación, no son los más adecuados. Tampoco se ha llevado a cabo una experimentación con la versión final del prototipo, con 6 motores en vez de cuatro, habilitando el control en yaw y el movimiento a través del plano XY. El sistema 'hardware-in-the-loop' debería ser seguido de un sistema 'software-in-the-loop', en el que Matlab puede ser, en este caso, el controlador completo de los actuadores conectados a la plataforma ArduPilot durante el vuelo, monitorizándolo simultáneamente.

El firmware necesita mejoras y nuevas funciones, mucho más robustas y fiables, con miras a la versión final del AirWhale. En este sentido, podría experimentarse, de forma introductoria, con la administración de tareas simultáneas del ejecutivo cíclico, con el GPS y el navegador inercial para vuelo en exteriores o recintos amplios, con la comunicación por telemetría y el control añadido del ángulo yaw, del movimiento en el plano XY y un recorrido basado en 'waypoints' u otro método práctico. Muchas posibilidades, incluso cambiar de plataforma autopiloto para aumentar la potencia y añadir filtrados más sofisticados para la obtención de ángulos más precisos con la realidad y otros datos afectados por ruido.

ANEXO A

Se presenta en este Anexo, el código del *script* de Matlab para el cálculo y obtención de las ecuaciones de modelo dinámico del AirWhale en coordenadas de Euler y su versión lineal:

Código 1. *Script de Matlab para el cálculo de las ecuaciones dinámica lineales y no lineales del AirWhale*

```
%% DEFINICION VARIABLES SIMBOLICAS

syms m % masa del prototipo
syms g % gravedad
syms Ixx Iyy Izz % momentos de inercia
syms rgx rgy rgz % coordenadas del centro de volumen
syms p q r % componentes de velocidad angular
syms Fm % fuerza de control
syms V U W % velocidades lineales
syms ro % densidad
% syms Vhe % volumen de helio
syms La lc ll % distancia entre motores y centro
syms fx fz % fuerza de control en x y z
syms mtx mty mtz % pares de control
syms he % fuerza producida por el helio
syms thetap phip psip fxp fzp pp qp rp Up Vp Wp mtxp mtyp mtzp fy  Ug Vg Wg
% incrementos de variables
syms q0 Cd Shx Shy Shz % presión dinámica, coeficiente aerodinámico y
superficie afectada por drag
syms theta phi psi

%% Datos relativos al prototipo
%he=0.6; g=9.8; Shx=0.5; Shy=1.45; Shz=Shy; Ixx=0.11; Iyy=0.295; Izz=0.26;
m=1.5; Cd=0.15;

%% Distancia de los motores al centro de masa
rm01=[0;-La;0];
rm02=[0;La;0];
rm03=[-lc;-ll;0];
rm04=[-lc;ll;0];

%% Gravedad
gr=g*[-sin(theta);cos(theta)*sin(phi);cos(theta)*cos(phi)];

%% Identidad
I=eye(3,3);

%% Centro de coordenadas
r0=[0;0;0];

%% Matriz de transformacion
R=[cos(psi)*cos(theta) sin(psi)*cos(theta) -sin(theta);
   cos(psi)*sin(theta)*sin(phi)-sin(psi)*cos(phi)
sin(psi)*sin(theta)*sin(phi)+cos(psi)*cos(phi) cos(theta)*sin(phi);
   cos(psi)*sin(theta)*cos(phi)+sin(psi)*sin(phi)
sin(psi)*sin(theta)*cos(phi)-cos(psi)*sin(phi) cos(theta)*cos(phi)];
```

```
%% Fuerza de control
Fm=R*[fx;0;-fz];


%% Matrices antisimetricas para producto vectorial (longitud entre motores y
centro de masa)
rm11=rm01;
rm12=rm02;
rm13=rm03;
rm14=rm04;
rm1=[0 -rm11(3) rm11(2);rm11(3) 0 -rm11(1);-rm11(2) rm11(1) 0];
rm2=[0 -rm12(3) rm12(2);rm12(3) 0 -rm12(1);-rm12(2) rm12(1) 0];
rm3=[0 -rm13(3) rm13(2);rm13(3) 0 -rm13(1);-rm13(2) rm13(1) 0];
rm4=[0 -rm14(3) rm14(2);rm14(3) 0 -rm14(1);-rm14(2) rm14(1) 0];

%% Velocidad lineal
vo=[U;V;W];

%% Centro de volumen en coordenadas de euler

rgp=zeros(3,3);

%% Definicion velocidades angulares y antisimétrica
w0=[p;q;r];
w=w0;
w1x=w(1);
w1y=w(2);
w1z=w(3);
W2=[0 -w1z w1y;w1z 0 -w1x;-w1y w1x 0];

%% Matriz de inercias
J=[Ixx 0 0;
   0 Iyy 0;
   0 0 Izz];

%% Matriz de masas
M=[m*I,-m*rgp;
   m*rgp,  J];

%% Fuerzas de inercia
ti=[R*(-m*W2*vo+m*W2*rgp*w);
    R*(-m*rgp*W2*vo-W2*J*w)];

%% Fuerzas gravitatorias
tg=[m*gr;
    m*rgp*gr];

%% Pares de control
Mt=[mtx;mty;mtz];

%% Fuerza y pares de control
tc=[Fm;
    R*(Mt)];

%% Fuerza de sustentación del helio
the=he*g*[R*[0;0;-1];0;0;0];

%% Drag
```

```
D=[(-1.225*0.5*U^2*Cd*Shx);(-1.225*0.5*V^2*Cd*Shy);(-
1.225*0.5*W^2*Cd*Shz);0;0;0];

%% Fuerza total (igual a M*q'dospuntos')
Tt=ti+tg+tc+the+D;

%% Ecuaciones iguales a xpunto, ypunto, zpunto, phipunto, thetapunto y
psipunto
xd=U;
yd=V;
zd=W;
ud=(1/m)*Tt(1);
vd=(1/m)*Tt(2);
wd=(1/m)*Tt(3);
pd=(1/Ixx)*Tt(4);
qd=(1/Iyy)*Tt(5);
rd=(1/Izz)*Tt(6);
phid= p + q*sin(phi)*tan(theta)+r*cos(phi)*tan(theta);
thetad= q*cos(phi)-r*sin(phi);
psid= q*sin(phi)*sec(theta)+r*cos(phi)*sec(theta);

%% Linealización de las ecuaciones: fuerzas totales y ecuaciones del vector
EC (12 ecuaciones)
xdl=Up;
ydl=Vp;
zdl=Wp;

udl=diff(ud,fx)*fxp+diff(ud,fz)*fzp+diff(ud,theta)*thetap+...
...diff(ud,phi)*phip+diff(ud,psi)*psip+diff(ud,U)*Up+diff(ud,V)*Vp+...
...diff(ud,W)*Wp;

vdl=diff(vd,fx)*fxp+diff(vd,fz)*fzp+diff(vd,theta)*thetap+...
...diff(vd,phi)*phip+diff(vd,psi)*psip+diff(vd,U)*Up+diff(vd,V)*Vp+...
...diff(vd,W)*Wp;

wdl=diff(wd,fx)*fxp+diff(wd,fz)*fzp+diff(wd,theta)*thetap+...
...diff(wd,phi)*phip+diff(wd,psi)*psip+diff(wd,U)*Up+diff(wd,V)*Vp+...
...diff(wd,W)*Wp;

pdl=diff(pd,mtx)*mtxp+diff(pd,mtz)*mtzp+diff(pd,mty)*mtyp+...
...diff(pd,theta)*thetap+diff(pd,phi)*phip+diff(pd,psi)*psip+...
...diff(pd,p)*pp+diff(pd,q)*qp+diff(pd,r)*rp;

qdl=diff(qd,mtx)*mtxp+diff(qd,mtz)*mtzp+diff(qd,mty)*mtyp+...
...diff(qd,theta)*thetap+diff(qd,phi)*phip+diff(qd,psi)*psip+...
...diff(qd,p)*pp+diff(qd,q)*qp+diff(qd,r)*rp;

rdl=diff(rd,mtx)*mtxp+diff(rd,mtz)*mtzp+diff(rd,mty)*mtyp+...
...diff(rd,theta)*thetap+diff(rd,phi)*phip+diff(rd,psi)*psip+...
...diff(rd,p)*pp+diff(rd,q)*qp+diff(rd,r)*rp;

phidl=diff(phid,theta)*thetap+diff(phid,phi)*phip+diff(phid,psi)*psip+...
...diff(phid,p)*pp+diff(phid,q)*qp+diff(phid,r)*rp;

thetadl=diff(thetad,theta)*thetap+diff(thetad,phi)*phip+...
...diff(thetad,psi)*psip+diff(thetad,p)*pp+diff(thetad,q)*qp+...
...diff(thetad,r)*rp;

psidl=diff(psid,theta)*thetap+diff(psid,phi)*phip+diff(psid,psi)*psip+...
...diff(psid,p)*pp+diff(psid,q)*qp+diff(psid,r)*rp;
```

```
%% Ecuaciones linealizadas evaluando el punto de equilibrio
ecl(1,1)=subs(xdl,{theta,phi,psi,U,V,W,mtx,mty,mtz,p,q,r,fx,fz},{0,0,0,5,0,0,
0,0,0,0,0,0,1.225*0.5*5^2*Cd*Shx,-he+g*m});

ecl(2,1)=subs(ydl,{theta,phi,psi,U,V,W,mtx,mty,mtz,p,q,r,fx,fz},{0,0,0,5,0,0,
0,0,0,0,0,0,1.225*0.5*5^2*Cd*Shx,-he+g*m});

ecl(3,1)=subs(zdl,{theta,phi,psi,U,V,W,mtx,mty,mtz,p,q,r,fx,fz},{0,0,0,5,0,0,
0,0,0,0,0,0,1.225*0.5*5^2*Cd*Shx,-he+g*m});

ecl(4,1)=subs(udl,{theta,phi,psi,U,V,W,mtx,mty,mtz,p,q,r,fx,fz},{0,0,0,5,0,0,
0,0,0,0,0,0,1.225*0.5*5^2*Cd*Shx,-he+g*m});

ecl(5,1)=subs(vdl,{theta,phi,psi,U,V,W,mtx,mty,mtz,p,q,r,fx,fz},{0,0,0,5,0,0,
0,0,0,0,0,0,1.225*0.5*5^2*Cd*Shx,-he+g*m});

ecl(6,1)=subs(wdl,{theta,phi,psi,U,V,W,mtx,mty,mtz,p,q,r,fx,fz},{0,0,0,5,0,0,
0,0,0,0,0,0,1.225*0.5*5^2*Cd*Shx,-he+g*m});

ecl(7,1)=subs(pdl,{theta,phi,psi,U,V,W,mtx,mty,mtz,p,q,r,fx,fz},{0,0,0,5,0,0,
0,0,0,0,0,0,1.225*0.5*5^2*Cd*Shx,-he+g*m});

ecl(8,1)=subs(qdl,{theta,phi,psi,U,V,W,mtx,mty,mtz,p,q,r,fx,fz},{0,0,0,5,0,0,
0,0,0,0,0,0,1.225*0.5*5^2*Cd*Shx,-he+g*m});

ecl(9,1)=subs(rdl,{theta,phi,psi,U,V,W,mtx,mty,mtz,p,q,r,fx,fz},{0,0,0,5,0,0,
0,0,0,0,0,0,1.225*0.5*5^2*Cd*Shx,-he+g*m});

ecl(10,1)=subs(phidl,{theta,phi,psi,U,V,W,mtx,mty,mtz,p,q,r,fx,fz},{0,0,0,5,0
,0,0,0,0,0,0,0,1.225*0.5*5^2*Cd*Shx,-he+g*m});

ecl(11,1)=subs(thetadl,{theta,phi,psi,U,V,W,mtx,mty,mtz,p,q,r,fx,fz},{0,0,0,5
,0,0,0,0,0,0,0,0,1.225*0.5*5^2*Cd*Shx,-he+g*m});

ecl(12,1)=subs(psidl,{theta,phi,psi,U,V,W,mtx,mty,mtz,p,q,r,fx,fz},{0,0,0,5,0
,0,0,0,0,0,0,0,1.225*0.5*5^2*Cd*Shx,-he+g*m});

ecl
```

ANEXO B

La siguiente tabla muestra los productos comprados junto a su enlace de compra:

Tabla 0.A. Enlaces web de compras de los productos definidos en el Capítulo 4

Nombre	Enlace
Motor Brushless EMAX BL2210/30	http://www.rctecnic.com/es/motores-brushless-para-aviones-rc/557-motor-brushless-outrunner-emax-bl221030-1300kv
Regulador Brushless EMAX 18A Budget	http://www.rctecnic.com/es/reguladores/2271-regulador-brushless-emax-18a-budget-esc-2a-bec-2-3s
Bateria Zippy 5000mAh 3S	http://www.hobbyjess.es/es/product/bateria-zippy-5000mah-3s111v-20-30c
Adaptador XT60 macho a t-dean hembra	http://www.hobbyjess.es/es/product/adaptador-de-xt60-macho-a-tdean-hembra-1
Cable de silicona 16AWG (1 metro)	http://www.rctecnic.com/es/cable-de-silicona/801-cable-de-silicona-diametro-29mm-16awg-1-metro
Conector tdean macho hembra	http://www.hobbyjess.es/es/product/conector-tipo-dean-t-rojo-macho-hembra
Conector XT60 hembra macho	http://www.hobbyjess.es/es/product/conector-xt60-hembra-macho
Cargador IMAX B6AC	http://www.hobbyjess.es/es/product/cargador-balanceador-hk-50w-5a-12v-digital
Servo TowerPro SG92r 9G	http://www.hobbyjess.es/es/product/servo-towerpro-sg90-9g
EMAX Program Card	http://www.rctecnic.com/es/reguladores/1910-emax-program-card-para-reguladores-brushless-emax
Placa de distribución de corriente	http://www.rctecnic.com/es/electronica-accesorios-y-fpv-para-multicopteros/3845-placa-de-distribucion-de-corriente-para-cuadricoptero-con-conector-t-dean
Hélice de Fibra de Carbono 9x4,7" (CCW)	http://www.rctecnic.com/es/helices-de-fibra-de-carbono/3582-helice-de-fibra-de-carbono-9x47-ccw-sentido-normal-dji
Hélice de Fibra de Carbono 9x4,7" (CW)	http://www.rctecnic.com/es/helices-de-fibra-de-carbono/3581-helice-de-fibra-de-carbono-9x47-cw-sentido-contrarrotatorio-dji

Las facturas de las compras realizadas se muestran a continuación. Se hicieron varias compras a lo largo de los meses, por lo que los gastos de envío aparecen varias veces en ellas. No se asocia ninguna identificación a las mismas.

RCTecnic
31/07/2015
Invoice #CU15002341

Facturación y Dirección de entrega.
Jose Luis Holgado Alvarez
Calle Avión cuatro vientos numero 9 6º1 Sevilla
41013 Sevilla
España

Número de pedido:
RKAUYVKIO

Fecha de pedido
31/07/2015

Método de pago
TPV CECA 220.20 €

Carrier:
24h Express

Producto / Referencia	Arriba (tasas no incluídas)	Arriba IVA incluído	Descuento	Cant.	Total IVA incluído
MOTOR BRUSHLESS OUTRUNNER EMAX BL2210/30 - 1300KV (Reference: BL2210/30)	15.62 €	18.90 €	--	6	113.40 €
REGULADOR BRUSHLESS EMAX 18A Budget ESC 2A BEC 2-3S (Reference: BudgetESC-18A)	13.15 €	15.90 €	--	6	95.40 €
EMAX Program Card para Reguladores Brushless EMAX (Reference: Program Card)	4.05 €	4.90 €	--	1	4.90 €
Placa de distribución de corriente para cuadricópteros (Reference: YY-PLACA DISTRIBUCIÓN QUAD)	5.38 €	6.50 €	--	1	6.50 €

Producto Total (Sin IVA).	181.98 €
Total del producto (IVA incl.)	220.20 €
Total tasa	38.22 €
Total	220.20 €

Detalle de tasa	Grado de Tasa	Total tasa inc.	Total tasa
Productos	21.000 %	181.98 €	38.22 €

RCTecnic
03/08/2015
Invoice #CU15002355

Facturación y Dirección de entrega.
Jose Luis Holgado Alvarez
Calle Avión cuatro vientos numero 9 6º1 Sevilla
41013 Sevilla
España

Número de pedido:
UUOBQDOMS

Fecha de pedido
03/08/2015

Método de pago
TPV CECA 69.10 €

Carrier:
24h Express

Producto / Referencia	Arriba (tasas no incluídas)	Arriba IVA incluído	Descuento	Cant.	Total IVA incluído
Hélice de Fibra de Carbono 9x4,7" (CCW, sentido normal) (DJI) (Reference: CFP-9x47-DJI-CCW)	6.53 €	7.90 €	--	4	31.60 €
Hélice de Fibra de Carbono 9x4,7" (CW, sentido contrarrotatorio) (DJI) (Reference: CFP-9x47-DJI)	6.53 €	7.90 €	--	4	31.60 €

Producto Total (Sin IVA).	52.24 €
Total del producto (IVA incl.)	63.20 €
Shipping Cost (Tax Incl.)	5.90 €
Total tasa	11.98 €
Total	69.10 €

Detalle de tasa	Grado de Tasa	Total tasa inc.	Total tasa
Productos	21.000 %	52.24 €	10.96 €
Transporte	21.000 %	4.88 €	1.02 €

RCTecnic
05/08/2015
Invoice #CU15002382

Facturación y Dirección de entrega.
Jose Luis Holgado Alvarez
Calle Avión cuatro vientos numero 9 6º1 Sevilla
41013 Sevilla
España

Número de pedido:
KYOJQZVYV

Fecha de pedido
05/08/2015

Método de pago
TPV CECA 15.90 €

Carrier:
24h Express

Producto / Referencia	Arriba (tasas no incluídas)	Arriba IVA incluído	Descuento	Cant.	Total IVA incluído
Cable de silicona diámetro 2.9mm - 16AWG (1 metro) (Reference: YY-M-048 S 16AWG)	2.07 €	2.50 €	--	4	10.00 €

Producto Total (Sin IVA).	8.28 €
Total del producto (IVA incl.)	10.00 €
Shipping Cost (Tax Incl.)	5.90 €
Total tasa	2.74 €
Total	15.90 €

Detalle de tasa	Grado de Tasa	Total tasa inc.	Total tasa
Productos	21.000 %	8.28 €	1.72 €
Transporte	21.000 %	4.88 €	1.02 €

HoBByJess.es

Justificante del pedido realizado

Hola jose sluis,

Su pedido con referencia '3201 ' está preparado para ser procesado.

Tus productos	Precio unitario	Cantidad	Precio
Bateria Zippy 5000mAh 3S(11,1V) 20-30c Ref: Z50003S1P-20	36.95€	2	73.90€
ADAPTADOR DE XT60 MACHO A TDEAN HEMBRA Ref: 258000018	2.90€	1	2.90€
CONECTOR TIPO DEAN T ROJO MACHO HEMBRA Ref: 606A-606B	1.00€	4	4.00€
Conector XT60 hembra macho Ref: 601Bx5	1.65€	1	1.65€
		Subtotal	**82.45€**
		Forma de envío	5.50€
		Total	**87.95€**

HoBByJess.es

Justificante del pedido realizado

Hola jose sluis,

Su pedido con referencia '3187 ' está preparado para ser procesado.

Tus productos	Precio unitario	Cantidad	Precio
CARGADOR BALANCEADOR HK ECO 220V Y 12V Ref: EC06	36.95€	1	36.95€
		Subtotal	**36.95€**
		Forma de envío	5.50€
		Total	**42.45€**

RCTecnic
26/08/2015
Invoice #CU15002553

Facturación y Dirección de entrega.
Alejandro Romero Galán
C/ Avión Cuatro Vientos nº9 Sexto 1
41013 Sevilla
España

Número de pedido:
RJBFBWQWM

Fecha de pedido
26/08/2015

Método de pago
TPV CECA 40.40 €

Carrier:
24h Express

Producto / Referencia	Arriba (tasas no incluidas)	Arriba IVA incluido	Descuento	Cant.	Total IVA incluido
REGULADOR BRUSHLESS EMAX Simon Series 20A para MULTICOPTEROS (Reference: SimonESC-20A)	14.05 €	17.00 €	--	1	17.00 €
Cable de silicona diámetro 2.9mm - 16AWG (1 metro) (Reference: YY-M-048 S 16AWG)	2.07 €	2.50 €	--	7	17.50 €

Producto Total (Sin IVA).	28.54 €
Total del producto (IVA incl.)	34.50 €
Shipping Cost (Tax Incl.)	5.90 €
Total tasa	6.98 €
Total	40.40 €

Detalle de tasa	Grado de Tasa	Total tasa inc.	Total tasa
Productos	21.000 %	28.54 €	5.96 €
Transporte	21.000 %	4.88 €	1.02 €

ANEXO C

Se presentan los códigos incluidos en *test_sensores_actuadores.pde* y *sintonizacion.pde*.

Código 1. *Código incluido en* ***test_sensores_actuadores.pde***

```
/*
  CODIGO ARDUPILOT AIRWHALE - TEST SENSORES Y ACTUADORES - ALUMNO: ALEJANDRO
                                ROMERO GALAN
 Este codigo ha sido desarrollado integramente por el alumno Alejandro Romero
                                Galan
    como parte de su Trabajo Fin de Grado en Ingenieria de las Tecnologias
                                Industriales
                    de la Universidad de Sevilla (ETSI).
*/


#include <AP_Common.h>
#include <stdarg.h>
#include <AP_Progmem.h>
#include <AP_Param.h>
#include <AP_HAL.h>
#include <AP_HAL_AVR.h>
#include <AP_HAL_PX4.h>
```

```
#include <AP_HAL_Linux.h>
#include <AP_HAL_Empty.h>
#include <AP_Math.h>
#include <RC_Channel.h>
#include <AP_Motors.h>
#include <AP_Curve.h>
#include <AP_Notify.h>
#include <AP_GPS.h>
#include <AP_Mission.h>
#include <DataFlash.h>
#include <AP_InertialSensor.h>
#include <AP_ADC.h>
#include <GCS_MAVLink.h>
#include <AP_Baro.h>
#include <Filter.h>
#include <AP_AHRS.h>
#include <AP_Compass.h>
#include <AP_Declination.h>
#include <AP_Airspeed.h>
#include <AP_Vehicle.h>
#include <AP_ADC_AnalogSource.h>
#include <AP_Mission.h>
#include <StorageManager.h>
#include <AP_Terrain.h>
#include <AP_NavEKF.h>
#include <AP_ADC.h>
#include <AP_Rally.h>




////////////////////////////////////////////////////////////////////////////
///// AP_Hal 'instance'
////////////////////////////////////////////////////////////////////////////

const AP_HAL::HAL& hal = AP_HAL_BOARD_DRIVER;

////////////////////////////////////////////////////////////////////////////
///// Barómetro
////////////////////////////////////////////////////////////////////////////

static AP_Baro_MS5611 barometer(&AP_Baro_MS5611::spi);

////////////////////////////////////////////////////////////////////////////
///// Magnetómetro
////////////////////////////////////////////////////////////////////////////

static AP_Compass_HMC5843 compass;

////////////////////////////////////////////////////////////////////////////
///// Acelerómetro
////////////////////////////////////////////////////////////////////////////

AP_InertialSensor ins;

////////////////////////////////////////////////////////////////////////////
///// Motores
////////////////////////////////////////////////////////////////////////////

RC_Channel rc1(0), rc2(1), rc3(2), rc4(3);
AP_MotorsQuad motors(rc1, rc2, rc3, rc4);
```

```
////////////////////////////////////////////////////////////////////////
///// GPS
////////////////////////////////////////////////////////////////////////

static AP_GPS  gps;

void setup()
{
    // Inicialización motores
    motors.Init();
    motors.enable();

    hal.scheduler->delay(1000);
}

void loop()
{
    // Definición
    int16_t opcion;

    // Opciones iniciales
    hal.console->println("Pulse la tecla indicada para la accion que desea
realizar:\n");
    hal.console->println("Pulsa T para probar los motores\n");
    hal.console->println("Pulsa A para probar el acelerometro/giroscopo\n");
    hal.console->println("Pulsa B para probar el barometro\n");
    hal.console->println("Pulsa B para probar el estimador DCM\n");

    // Se espera a que se teclee algo
    while( !hal.console->available() ) {
        hal.scheduler->delay(20);
    }

    // Se obtiene el valor tecleado
    opcion=hal.console->read();

    // Se comprueba la letra pulsada
    if (opcion == 't' || opcion == 'T') {
        test_motores();
    }
    if (opcion == 'a' || opcion == 'A') {
        test_acel_gir();
    }
    if (opcion == 'b' || opcion == 'B') {
        test_barom();
    }
    if (opcion == 'e' || opcion == 'E') {
        test_AHRS();
    }
}

void test_motores(void){
  int16_t pwm_1;
  int16_t pwm_2;
  int16_t pwm_3;
  int16_t pwm_4;
  int fin = 0;
  uint16_t opcion;

  // Comienza el test
    while(fin==0){
```

```
      // Se pide los valores PWM para cada motor
      hal.console->println("Introduce un valor de PWM para el motor 1 (aprox.
1000-2000)\n");
      pwm_1 = captura_datos();

      hal.console->println("Introduce un valor de PWM para el motor 2 (aprox.
1000-2000)\n");
      pwm_2 = captura_datos();

      hal.console->println("Introduce un valor de PWM para el motor 3 (aprox.
1000-2000)\n");
      pwm_3 = captura_datos();

      hal.console->println("Introduce un valor de PWM para el motor 4 (aprox.
1000-2000)\n");
      pwm_4 = captura_datos();

      hal.console->printf("\n");
      hal.console->printf("\n");

      // Se manda los valores adquiridos a cada motor durante dos segundos de
forma individual
      hal.console->printf("Mandando PWM %d a motor 1\n", (int)pwm_1);
      hal.rcout->write(0,(uint16_t)pwm_1);
      hal.scheduler->delay(2000);
      hal.rcout->write(0,0);

      hal.console->printf("Mandando PWM %d a motor 2\n", (int)pwm_2);
      hal.rcout->write(1,(uint16_t)pwm_2);
      hal.scheduler->delay(2000);
      hal.rcout->write(0,0);

      hal.console->printf("Mandando PWM %d a motor 3\n", (int)pwm_3);
      hal.rcout->write(2,(uint16_t)pwm_3);
      hal.scheduler->delay(2000);
      hal.rcout->write(0,0);

      hal.console->printf("Mandando PWM %d a motor 4\n", (int)pwm_4);
      hal.rcout->write(3,(uint16_t)pwm_4);
      hal.scheduler->delay(2000);
      hal.rcout->write(0,0);

      hal.console->println("Fin del test, pulsa T si quieres volver a
realizar un test\n");

      while( hal.console->available() ) {
        hal.console->read();
        hal.scheduler->delay(20);
      }
      while( !hal.console->available() ) {
        hal.scheduler->delay(20);
      }

      hal.console->printf("\n");
      hal.console->printf("\n");

      opcion=hal.console->read();

      // Si es 't', se vuelve a realizar el test
      if(opcion == 't' || opcion == 'T'){
      }
```

```
      else{
        fin = 1;
      }
      hal.scheduler->delay(2000);
    }
    hal.console->println("Fin del test de motores");
}

void test_barom(void){
    int contador=0;
    float altura=0;
    float presion=0;
    float temperatura=0;
    int fin = 0;
    uint16_t opcion;

    // Se activa el pin del barómetro y se desactiva el del acelerometro
    hal.gpio->pinMode(63, HAL_GPIO_OUTPUT);
    hal.gpio->write(63, 1);

    // Se inicia el barómetro
    barometer.init();

    // Se inicia el test
    while(fin==0){

    contador = 0;
    hal.console->printf("Se realizaran 200 medidas (0.1 ms cada una) sin
calibrar\n");
    hal.console->printf("\n");
    hal.console->printf("\n");
    hal.console->printf("Altura: \t Presion: \t Temperatura: \n");

    // Se realizan 200 medidas
    while(contador<200){
      hal.scheduler->delay(100);

      // Se lee el barómetro
      barometer.read();

      // Se extrae los datos de presión, temperatura y altitud
      altura = barometer.get_altitude();
      presion = barometer.get_pressure();
      temperatura = barometer.get_temperature();

      // Se imprimen por pantalla
      hal.console->printf("%f %f %f;\n", altura, presion, temperatura);
      contador++;
    }

    hal.console->printf("\n");
    hal.console->printf("\n");

    // Se calibra el barómetro
    barometer.calibrate();

    // Se lee 200 veces el barómetro
    hal.console->printf("Se realizaran 200 medidas (0.1 ms cada una),
calibrado el sensor\n");
    hal.console->printf("\n");
    hal.console->printf("\n");
    hal.console->printf("Altura: \t Presion: \t Temperatura: \n");
```

```
    contador = 0;
    while(contador<200){
      hal.scheduler->delay(100);
      barometer.read();
      altura = barometer.get_altitude();
      presion = barometer.get_pressure();
      temperatura = barometer.get_temperature();
      hal.console->printf("%f %f %f;\n", altura, presion, temperatura);
      contador++;
    }

    hal.console->printf("\n");
    hal.console->printf("\n");

    hal.console->println("Fin del test, pulsa B si quieres volver a realizar
un test\n");

      while( hal.console->available() ) {
        hal.console->read();
        hal.scheduler->delay(20);
      }
      while( !hal.console->available() ) {
        hal.scheduler->delay(20);
      }

      hal.console->printf("\n");
      hal.console->printf("\n");

      opcion=hal.console->read();

      // Si se pulsa 'b', se vuelve a empezar el test
      if(opcion == 'b' || opcion == 'B'){
      }
      else{
        fin = 1;
      }
      hal.scheduler->delay(2000);
    }
    hal.console->println("Fin del test de barometro");
}

void test_acel_gir(void){

    // Se activa el pin del acelerómetro y se desactiva el del barometro
    hal.gpio->pinMode(40, HAL_GPIO_OUTPUT);
    hal.gpio->write(40, 1);

    // Se inicia el acelerometro
    ins.init(AP_InertialSensor::WARM_START, AP_InertialSensor::RATE_100HZ);

    // Definicion
    Vector3f aceleraciones;
    Vector3f giros;
    int contador=0;
    uint16_t opcion;
    int fin=0;

    // Inicio del test
    while(fin==0){

    contador = 0;
```

```
    // Se lee 200 veces el acelerometro
    hal.console->printf("Se realizaran 200 medidas (0.1 s cada una) sin
calibrar\n");
    hal.console->printf("\n");
    hal.console->printf("\n");
    hal.console->printf("Aceleracion en X: \t Aceleracion en Y: \t
Aceleracion en Z: A. Giro en X: \t A. Giro en Y: \t A. Giro en Z: \n");

    while(contador<200){
      // Se espera a que el acelerómetro tenga una medida
      ins.wait_for_sample();

      // Se actualiza la medida
      ins.update();

      // Se extraen las aceleraciones y velocidades
      aceleraciones = ins.get_accel();
      giros = ins.get_gyro();

      hal.scheduler->delay(100);

      // Se imprime por pantalla los resultados
      hal.console->printf("%f %f %f %f %f %f;\n", aceleraciones.x,
aceleraciones.y, aceleraciones.z, giros.x, giros.y, giros.z);
      contador++;
    }

    hal.console->printf("Se realizaran 200 medidas (0.1 s cada una),
calibrado el sensor\n");
    hal.console->printf("\n");
    hal.console->printf("\n");
    hal.console->printf("Aceleracion en X: \t Aceleracion en Y: \t
Aceleracion en Z: A. Giro en X: \t A. Giro en Y: \t A. Giro en Z: \n");

    // Se calibra el acelerometro/giroscopo
    ins.init_gyro();
    ins.init_accel();

    contador=0;

    while(contador<200){
      ins.wait_for_sample();
      ins.update();
      aceleraciones = ins.get_accel();
      giros = ins.get_gyro();
      hal.scheduler->delay(100);
      hal.console->printf("%f %f %f %f %f %f;\n", aceleraciones.x,
aceleraciones.y, aceleraciones.z, giros.x, giros.y, giros.z);
      contador++;
    }

    hal.console->printf("\n");
    hal.console->printf("\n");

    hal.console->println("Fin del test, pulsa A si quieres volver a realizar
un test\n");

      while( hal.console->available() ) {
        hal.console->read();
        hal.scheduler->delay(20);
      }
```

```
      while( !hal.console->available() ) {
        hal.scheduler->delay(20);
      }

      hal.console->printf("\n");
      hal.console->printf("\n");

      opcion=hal.console->read();

      // Si se pulsa 'a', se vuelve a hacer el test
      if(opcion == 'a' || opcion == 'A'){
      }
      else{
        fin = 1;
      }
      hal.scheduler->delay(2000);
    }
    hal.console->println("Fin del test de acelerometro");
}

void test_AHRS(void){

    int contador=0;
    float altura=0;
    float latitud=0;
    float longitud=0;
    int fin = 0;
    uint16_t opcion;

    // Inicio de acelerómetro, barómetro y AHRS
    hal.gpio->pinMode(40, HAL_GPIO_OUTPUT);
    hal.gpio->write(40, 1);
    ins.init(AP_InertialSensor::COLD_START, AP_InertialSensor::RATE_100HZ);
    ins.init_accel();

    hal.gpio->pinMode(63, HAL_GPIO_OUTPUT);
    hal.gpio->write(63, 1);
    barometer.init();

    ahrs.init();

    // Se inicia el test
    while(fin==0){

    contador = 0;
    hal.console->printf("Se realizaran 200 medidas (0.1 ms cada una)\n");
    hal.console->printf("\n");
    hal.console->printf("\n");
    hal.console->printf("Roll: \t Pitch: \t Yaw: \n");

    // Se realizan las 200 medidas
    while(contador<200){
      // Se actualiza el filtrado DCM
      ahrs.update();

      // Se imprime por pantalla los ángulos
      hal.console->printf("%f %f %f\n",
ToDeg(ahrs.roll),ToDeg(ahrs.pitch),ToDeg(ahrs.yaw));

      contador++;
    }
```

```
    hal.console->printf("\n");
    hal.console->printf("\n");

    hal.console->println("Fin del test, pulsa E si quieres volver a realizar
un test\n");

      while( hal.console->available() ) {
        hal.console->read();
        hal.scheduler->delay(20);
      }
      while( !hal.console->available() ) {
        hal.scheduler->delay(20);
      }

      hal.console->printf("\n");
      hal.console->printf("\n");

      opcion=hal.console->read();

      // Si se pulsa 'e', se inicia el test de nuevo
      if(opcion == 'e' || opcion == 'E'){
      }
      else{
        fin = 1;
      }
      hal.scheduler->delay(2000);
    }
    hal.console->println("Fin del test del estimador DCM");
}



float captura_datos(void){

  int16_t valor_consola;
  int16_t pwm_mil;
  int16_t pwm_cien;
  int16_t pwm_diez;
  int16_t pwm_uno;
  float pwm_final;
  int contador = 0;
  int fin = 0;

  // Se captura los datos del puerto serie
    hal.scheduler->delay(200);
        while(fin == 0){
        while( !hal.console->available() ) {
          hal.scheduler->delay(20);
        }
        valor_consola = hal.console->read();
        if(valor_consola == 13 || (valor_consola<48 && valor_consola>57)){
          hal.console->println("Dato a desechar");
          if(contador > 3)
          {
            fin=1;
            hal.console->println("Fin lectura");
          }
        }
        else{
          if(contador == 0){
            pwm_mil = valor_consola;
            hal.console->println("Leido dato decena");
```

```
          }
          if(contador == 1){
            pwm_cien = valor_consola;
            hal.console->println("Leido dato unidad");
          }
          if(contador == 2){
            pwm_diez = valor_consola;
            hal.console->println("Leido dato decima");
          }
          if(contador == 3){
            pwm_uno=valor_consola;
            hal.console->println("Leido dato centesima");
          }
          contador++;
        }
        }

    // Se halla el valor para el cálculo en PID
    pwm_final=(1000*(pwm_mil-48))+(100*(pwm_cien-48))+(10*(pwm_diez-
48))+(pwm_uno-48);
    return pwm_final;
}

AP_HAL_MAIN();
```

Código 2. *Código incluido en* ***sintonizacion.pde***

```
/*
  CODIGO ARDUPILOT AIRWHALE - SINTONIZACION - ALUMNO: ALEJANDRO ROMERO GALAN
 Este codigo ha sido desarrollado integramente por el alumno Alejandro Romero
                                    Galan
    como parte de su Trabajo Fin de Grado en Ingenieria de las Tecnologias
                                    Industriales
                      de la Universidad de Sevilla (ETSI).
*/



#include <AP_Common.h>
#include <stdarg.h>
#include <AP_Progmem.h>
#include <AP_Param.h>
#include <AP_HAL.h>
#include <AP_HAL_AVR.h>
```

```
#include <AP_HAL_PX4.h>
#include <AP_HAL_Linux.h>
#include <AP_HAL_Empty.h>
#include <AP_Math.h>
#include <RC_Channel.h>
#include <AP_Motors.h>
#include <AP_Curve.h>
#include <AP_Notify.h>
#include <AP_GPS.h>
#include <AP_Mission.h>
#include <DataFlash.h>
#include <AP_InertialSensor.h>
#include <AP_ADC.h>
#include <GCS_MAVLink.h>
#include <AP_Baro.h>
#include <Filter.h>
#include <AP_AHRS.h>
#include <AP_Compass.h>
#include <AP_Declination.h>
#include <AP_Airspeed.h>
#include <AP_Vehicle.h>
#include <AP_ADC_AnalogSource.h>
#include <AP_Mission.h>
#include <StorageManager.h>
#include <AP_Terrain.h>
#include <AP_NavEKF.h>
#include <AP_ADC.h>
#include <AP_Rally.h>
#include <AP_Scheduler.h>
#include <AP_Compass_HMC5843.h>
#include <AP_InertialNav.h>
#include <AP_Buffer.h>

////////////////////////////////////////////////////////////////////////
///// AP_Hal 'instance'
////////////////////////////////////////////////////////////////////////

const AP_HAL::HAL& hal = AP_HAL_BOARD_DRIVER;

////////////////////////////////////////////////////////////////////////
///// Barómetro
```

```
////////////////////////////////////////////////////////////////////////

static AP_Baro_MS5611 barometer(&AP_Baro_MS5611::spi);

////////////////////////////////////////////////////////////////////////
///// Magnetómetro
////////////////////////////////////////////////////////////////////////

static AP_Compass_HMC5843 compass;

////////////////////////////////////////////////////////////////////////
///// Acelerómetro
////////////////////////////////////////////////////////////////////////

AP_InertialSensor ins;

////////////////////////////////////////////////////////////////////////
///// Motores
////////////////////////////////////////////////////////////////////////

RC_Channel rc1(0), rc2(1), rc3(2), rc4(3);
AP_MotorsQuad   motors(rc1, rc2, rc3, rc4);

////////////////////////////////////////////////////////////////////////
///// GPS
////////////////////////////////////////////////////////////////////////

static AP_GPS  gps;

////////////////////////////////////////////////////////////////////////
///// Estimador DCM
////////////////////////////////////////////////////////////////////////

AP_AHRS_DCM ahrs(ins, barometer, gps);

////////////////////////////////////////////////////////////////////////
///// Variables globales
////////////////////////////////////////////////////////////////////////

float Ref_roll = 0;  // Referencia roll
float Ref_pitch = 0;  // Referencia pitch
```

```
float Ref_z; // Referencia Z
float Kp_roll, Ti_roll, Td_roll;  // Parámetros del PID del controlador Roll
float Kp_pitch, Ti_pitch, Td_pitch;  // Parámetros del PID del controlador Pitch
float Kp_z, Ti_z, Td_z;  // Parámetros del PID del controlador altura en Z
int T_roll = 20;  // Tiempo de muestreo control roll
int T_pitch = 20;  // Tiempo de muestreo control pitch
int T_z = 20;  // Tiempo de muestreo control Z
float Uk_roll;  // Variable de actuación (salida del PID control roll)
float Uk_pitch;  // Variable de actuación (salida del PID control pitch)
float Uk_z;  // Variable de actuación (salida del PID control Z)
float Ik_roll, Ik1_roll;  // Variables para el término integral
float Ik_pitch, Ik1_pitch;  // Variables para el término integral
float Ik_z, Ik1_z;  // Variables para el término integral
float Ek_roll, Ek1_roll;  // Variables para el error
float Ek_pitch, Ek1_pitch;  // Variables para el error
float Ek_z, Ek1_z;  // Variables para el error



////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
  SETUP
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////


void setup()
{
    // Inicio acelerometro

    hal.gpio->pinMode(40, HAL_GPIO_OUTPUT);
    hal.gpio->write(40, 1);
    ins.init(AP_InertialSensor::COLD_START, AP_InertialSensor::RATE_100HZ);

    // Inicio barometro

    hal.gpio->pinMode(63, HAL_GPIO_OUTPUT);
    hal.gpio->write(63, 1);
    barometer.init();
    barometer.calibrate();

    // Inicio magnetómetro
```

```
    compass.init();

    // Inicio motores

    motors.Init();
    motors.enable();
    hal.rcout->write(0,0);
    hal.rcout->write(1,0);
    hal.rcout->write(2,0);
    hal.rcout->write(3,0);

    // Inicio estimador

    ahrs.init();

    hal.scheduler->delay(1000);
}



////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
          Bucle principal
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////


void loop()
{
    // Definiciones
    float altura;
    float PWM_motor1 = 0;
    float PWM_motor2 = 0;
    float PWM_motor3 = 0;
    float PWM_motor4 = 0;
    float L1 = 0.8;
    float L2 = 0.7;
    int fin = 0;
    int anti_roll = 0;
    int anti_pitch = 0;
    int anti_z = 0;
    int16_t tecla;
    float Fz=0;
```

```
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////


Sintonización

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////


    hal.console->println("Sintonizacion de controladores. Valores pedidos: Kp,
Ti, Td y referencia\n");
    hal.console->println("Sintonizacion controlador PID para el roll\n");


    ///////////////
    /// Kp_roll ///
    ///////////////

    // Se pide un valor para los parámetros de los controladores
    hal.console->println("Introduce un valor de Kp_roll\n");
    Kp_roll = captura_datos();

    ///////////////
    /// Ti_roll ///
    ///////////////

    // Se pide un valor para los parámetros de los controladores
    hal.console->println("Introduce un valor de Ti_roll\n");
    Ti_roll = captura_datos();

    ///////////////
    /// Td_roll ///
    ///////////////

    // Se pide un valor para los parámetros de los controladores
    hal.console->println("Introduce un valor de Td_roll\n");
    Td_roll = captura_datos();

    ////////////////
    /// Ref_roll ///
    ////////////////
```

```
// Se pide un valor para los parámetros de los controladores
hal.console->println("Introduce un valor de Ref_roll (en grados)\n");
Ref_roll = captura_datos();

////////////////
/// Kp_pitch ///
////////////////

hal.console->println("Sintonizacion controlador PID para el pitch\n");

// Se pide un valor para los parámetros de los controladores
hal.console->println("Introduce un valor de Kp\n");
Kp_pitch = captura_datos();

////////////////
/// Ti_pitch ///
////////////////

// Se pide un valor para los parámetros de los controladores
hal.console->println("Introduce un valor de Ti\n");
Ti_pitch = captura_datos();

////////////////
/// Td_pitch ///
////////////////

// Se pide un valor para los parámetros de los controladores
hal.console->println("Introduce un valor de Td\n");
Td_pitch = captura_datos();

/////////////////
/// Ref_pitch ///
/////////////////

// Se pide un valor para los parámetros de los controladores
hal.console->println("Introduce un valor de Ref_pitch (en grados)\n");
Ref_pitch = captura_datos();

////////////
/// Kp_z ///
////////////
```

```
hal.console->println("Sintonizacion controlador PID para Z\n");

// Se pide un valor para los parámetros de los controladores
hal.console->println("Introduce un valor de Kp\n");
Kp_z = captura_datos();

////////////
/// Ti_z ///
////////////

// Se pide un valor para los parámetros de los controladores
hal.console->println("Introduce un valor de Ti\n");
Ti_z = captura_datos();

////////////
/// Td_z ///
////////////

// Se pide un valor para los parámetros de los controladores
hal.console->println("Introduce un valor de Td\n");
Td_z = captura_datos();

/////////////
/// Ref_z ///
/////////////

// Se pide un valor para los parámetros de los controladores
hal.console->println("Introduce un valor de Ref_z (en metros)\n");
Ref_z = captura_datos();

//// OPCION DE PEDIR ESCALÓN PARA FZ ////

//    hal.console->println("Introduce un valor de Fz (en PWM)\n");
//    Fz = captura_datos();
//    Fz = 100*Fz;

hal.console->println("Fin de sintonizacion. Iniciando controladores\n");
```

```
////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////
Controladores

////////////////////////////////////////////////////////////////////////////
////////////////////////////////////////////////////////////////////////////


      hal.console->printf("Roll: \t Pitch: \t Altura: \t Error Roll: \t Error
Pitch: \t Error Z: \t Actuacion Roll: \t Actuacion Pitch: \t Actuacion Z: \t
Fuerza motor 1: \t Fuerza motor 2: \t Fuerza motor 3: \t Fuerza motor 4: \n");
      hal.rcout->write(0,0);
      hal.rcout->write(1,0);
      hal.rcout->write(2,0);
      hal.rcout->write(3,0);


      //// INICIO BUCLE DE CONTROL ////









      while(fin == 0){

      /////////////////
      //// Lectura sensor
      /////////////////

      ahrs.update();  // Se activa el estimador, obteniendo los ángulos de Euler
      barometer.read();  // Se lee barómetro
      altura = barometer.get_altitude();  // Se guarda la altitud

      /////////////////
      //// Controlador Roll
      /////////////////

      Ek_roll = Ref_roll - ToDeg(ahrs.roll);  // Se calcula el error
      if(anti_roll == 1){  // Comprobación de anti Wind-Up
        Ik_roll = Ik1_roll;
      }else{
      Ik_roll = Ik1_roll + Ek_roll;  // Se actualiza integrador
      }
      Uk_roll  =  (Kp_roll  *  (Ek_roll  +  (((T_roll/1000)/Ti_roll)*Ik_roll)  +
```

```
(((Td_roll*1000)/T_roll)*(Ek_roll-Ek1_roll))));  // Se halla actuación
      Ik1_roll = Ik_roll;  // Se actualizan valores de la iteración anterior
para la siguiente iteración
      Ek1_roll = Ek_roll;

      /////////////////
      //// Controlador Pitch
      /////////////////

      Ek_pitch = Ref_pitch - ToDeg(ahrs.pitch);
      if(anti_pitch == 1){
        Ik_pitch = Ik1_pitch;
      }else{
      Ik_pitch = Ik1_pitch + Ek_pitch;
      }
      Uk_pitch = ((Kp_pitch)*(((Ek_pitch)) +
(((T_pitch/1000)/(Ti_pitch))*(Ik_pitch)) +
(((Td_pitch*1000)/(T_pitch)*(Ek_pitch-Ek1_pitch)))));
      Ik1_pitch = Ik_pitch;
      Ek1_pitch = Ek_pitch;

      /////////////////
      //// Controlador Z
      /////////////////

      Ek_z = Ref_z - altura;
      if(anti_z == 1){
        Ik_z = Ik1_z;
      }else{
      Ik_z = Ik1_z + Ek_z;
      }

      Uk_z = ((Kp_z)*(((Ek_z)) + (((T_z/1000)/(Ti_z))*(Ik_z)) +
(((Td_z*1000)/(T_z)*(Ek_z-Ek1_z)))));
      Ik1_z = Ik_z;
      Ek1_z = Ek_z;

      // Comprobación saturación y anti Wind-Up
      if(Uk_roll<-50){
        Uk_roll = -50;
        anti_roll = 1;
      }else{
```

```
        anti_roll = 0;
      }
      if(Uk_pitch<-60){
        Uk_pitch = -60;
        anti_pitch = 1;
      }else{
        anti_pitch = 0;
      }
      if(Uk_roll>50){
        Uk_roll = 50;
        anti_roll = 1;
      }else{
        anti_roll = 0;
      }
      if(Uk_pitch>60){
        Uk_pitch = 60;
        anti_pitch = 1;
      }else{
        anti_pitch = 0;
      }

      /////////////////
      //// PWM motores
      /////////////////

      PWM_motor1 = ((L1*(Uk_z)) - (2*Uk_pitch)) / (4*L1);
      PWM_motor2 = ((L2*(Uk_z)) + (2*Uk_roll)) / (4*L2);
      PWM_motor3 = ((L1*(Uk_z)) + (2*Uk_pitch)) / (4*L1);
      PWM_motor4 = ((L2*(Uk_z)) - (2*Uk_roll)) / (4*L2);

      PWM_motor1 = ((PWM_motor1)+1200+Fz);
      PWM_motor2 = ((PWM_motor2)+1200+Fz);
      PWM_motor3 = ((PWM_motor3)+1200+Fz);
      PWM_motor4 = ((PWM_motor4)+1200+Fz);

      hal.console->printf("%f %f %f %f %f %f %f %f %f %f %f %f %f;\n",
ToDeg(ahrs.roll), ToDeg(ahrs.pitch), altura, Ek_roll, Ek_pitch, Ek_z, Uk_roll,
Uk_pitch, Uk_z, PWM_motor1, PWM_motor2, PWM_motor3, PWM_motor4);
      hal.console->printf("\n");

      hal.rcout->write(0,(int16_t)(PWM_motor1));
      hal.rcout->write(1,(int16_t)(PWM_motor2));
```

```
      hal.rcout->write(2,(int16_t)(PWM_motor3));
      hal.rcout->write(3,(int16_t)(PWM_motor4));

      // Comprobación de fin bucle, útil por seguridad

      if(hal.console->available()){
        tecla = hal.console->read();
        if(tecla == 'f' || tecla == 'F'){
          fin = 1;
        }
      }
      }

      // Se detienen los motores
      hal.rcout->write(0,0);
      hal.rcout->write(1,0);
      hal.rcout->write(2,0);
      hal.rcout->write(3,0);

      hal.console->printf("Fin del bucle de control. Se pedira otra
sintonizacion de los controladores para la siguiente prueba\n");
      hal.console->printf("\n");
      hal.console->printf("\n");
      hal.console->printf("\n");
}

float captura_datos(void){

  int16_t valor_consola;
  int16_t valor_diez;
  int16_t valor_uno;
  int16_t valor_decima;
  int16_t valor_centesima;
  float valor_final;
  int contador = 0;
  int fin = 0;
  // Se captura los datos del puerto serie
    hal.scheduler->delay(200);
        while(fin == 0){
        while( !hal.console->available() ) {
          hal.scheduler->delay(20);
```

```
        }
        valor_consola = hal.console->read();
        if(valor_consola == 13 || (valor_consola<48 && valor_consola>57)){
          hal.console->println("Dato a desechar");
          if(contador > 3)
          {
            fin=1;
            hal.console->println("Fin lectura");
          }
        }
        else{
          if(contador == 0){
            valor_diez = valor_consola;
            hal.console->println("Leido dato decena");
          }
          if(contador == 1){
            valor_uno = valor_consola;
            hal.console->println("Leido dato unidad");
          }
          if(contador == 2){
            valor_decima = valor_consola;
            hal.console->println("Leido dato decima");
          }
          if(contador == 3){
            valor_centesima=valor_consola;
            hal.console->println("Leido dato centesima");
          }
          contador++;
        }
        }


    // Se halla el valor para el cálculo en PID
    valor_final    =    (10*(valor_diez-48))    +    (1*(valor_uno-48))    +
(0.1*(valor_decima-48)) + (0.01*(valor_centesima-48));
    return valor_final;
}


AP_HAL_MAIN();
```

REFERENCIAS

[1] J. C. Mancebo, «Diseño Asistido por Ordenador del AirWhale y Cálculo Estructural de sus Alas,» Sevilla, 2015.

[2] I. G. Vázquez, «Diseño y Cálculo de las Características Aerodinámicas y de Estabilidad de un Dirigible Híbrido: Proyecto AirWhale,» Sevilla, 2015.

[3] J. E. M. Samayoa, «Diseño Básico, Bussiness Case y Prototipado Estructural del AirWhale,» Sevilla, 2015.

[4] J. L. H. Álvarez, Sevilla, 2015.

[5] E. M. Caballero, Sevilla, 2015.

[6] Página principal PX4 Autopilot, módulo PIXHAWK.

[7] Wiki del Proyecto OpenPilot, especificaciones de productos.

[8] Página principal OpenPilot, detalles sobre el producto CopterControl.

[9] Página principal del Proyecto OpenPilot, detalles sobre el producto Revolution.

[10] Página web MicroChip, especificaciones del microprocesador dsPIC33EP512MU810.

[11] Página web Arsov RC Technology, detalles producto AUAV3.

[12] Página principal Proyecto Paparazzi, plataformas autopiloto contempladas.

[13] Página principal Proyecto ArduPilot.

[14] Página web Atmel, especificaciones ATMEGA2560.

[15] Página web Atmel, especificaciones ATMEGA32U2.

[16] Enlace de descarga del software Mission Planner.

[17] J. A. G. Fernández, «Desarrollo de una Plataforma RPAS para Simulación Hardware In The Loop y Experimentación en Vuelo,» Sevilla, 2015.

[18] Página web 3D Robotics, detalles del producto 3DR uBlox GPS.

[19] Página web 3D Robotics, detalles del producto 3DR Power Module.

[20] Página web 3D Robotics, detalles del producto 3DR Radio Set.

[21] Datasheet XL-MAXSONAR EZ Series.

[22] Datasheet DataTraveler microDuo 3.0.

[23] Enlace de descarga Firmware ArduPilot.

[24] Página web Proyecto ArduPilot, introducción al EKF.

[25] Página web Proyecto ArduPilot, introducción al EPM.

[26] Datasheet barómetro MS5611-01BA03.

[27] Teoría sobre la atmósfera estándar.

[28] W. Premerlani y P. Bizard, «Direction Cosine Matrix IMU: Theory».

[29] Página web Proyecto ArduPilot, detalles sobre 'threading'.