UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Fin de Grado

Implementación y validación del sistema de control y navegación para una aeronave no tripulada dentro del proyecto HERMES-UPV

Autora: Laura Cristina Smith Ballester

Tutores: Francesc Xavier Blasco Ferragud
Sergio García-Nieto Rodríguez

Grado en Ingeniería Aeroespacial

Escuela Técnica Superior de Ingeniería del Diseño

Universitat Politècnica de València

Curso 2018/2019

Este trabajo contiene los siguientes documentos:

- Memoria
- Pliego de condiciones
- Presupuesto

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo Fin de Grado

Implementación y validación del sistema de control y navegación para una aeronave no tripulada dentro del proyecto HERMES-UPV

MEMORIA

Autora: Laura Cristina Smith Ballester

Tutores: Francesc Xavier Blasco Ferragud
Sergio García-Nieto Rodríguez

Grado en Ingeniería Aeroespacial

Escuela Técnica Superior de Ingeniería del Diseño

Universitat Politècnica de València

Curso 2018/2019

Resumen

Este trabajo se centra en la adaptación del código del piloto automático de código libre *ArduPilot* a los controladores diseñados para el vehículo aéreo no tripulado del proyecto HERMES-UPV. El objetivo de esta adaptación es la introducción del nuevo software en el ordenador de a bordo de la aeronave para que esta pueda realizar una misión de seguimiento de waypoints de forma autónoma. Esta implementación se consigue a través del rediseño del piloto automático mediante simulaciones en *Simulink* y empleando un algoritmo genético para la obtención de los parámetros necesarios. El proceso de rediseño incluye la realización de simulaciones de vuelo en tiempo real con la técnica SITL (*Software In The Loop*) para la validación de los controladores diseñados y de la ley de navegación.

Abstract

This project is focused on the adaptation of the open source automatic pilot *ArduPilot* to the previously designed controllers in the HERMES-UPV unmanned aerial vehicle project. This is done in order to install this new software in the aircraft's control board so the aircraft can fly autonomously following a defined mission that includes a series of waypoints. The implementation is achieved modifying the automatic pilot by simulating the behaviour of the aircraft with *Simulink* and using a genetic algorithm to obtain the new parameters. The re-design process includes real time simulations using the SITL (*Software In The Loop*) technique to validate the functioning of the controllers and the implemented navigation law.

Resum

Aquest treball es centra en l'adaptació del codi del pilot automàtic de codi lliure *ArduPilot* als controladors dissenyats per al vehicle aeri no tripulat del projecte HERMES-UPV. L'objectiu d'aquesta adaptació és la introducció del nou software en l'ordenador de bord de l'aeronau perquè esta puga realitzar una missió de seguiment de waypoints de forma autònoma. Aquesta implementació s'aconseguix a través del redisseny del pilot automàtic per mitjà de simulacions en *Simulink* i utilizant un algoritme genètic per a l'obtenció dels paràmetres necessaris. El procés de redisseny inclou la realització de simulacions de vol en temps real amb la tècnica SITL (*Software In The Loop*) per a la validació dels controladors dissenyats i de la llei de navegació.

Índice general

Índice de figuras

Índice de cuadros

Lista de símbolos

α	Ángulo de ataque
β	Ángulo de derrape
δA	Ángulo de deflexión de los alerones
δE	Ángulo de deflexión de los elevadores
δP	Posición de palanca de gases
δR	Ángulo de deflexión del timón de cola
ϕ	Ángulo de alabeo
Ψ	Ángulo de guiñada
ρ	Densidad
θ	Ángulo de cabeceo
ζ	Rumbo
b_w	Envergadura del ala
C_D	Coeficiente de resistencia o drag
C_L	Coeficiente de sustentación o lift
C_l	Coeficiente de momento de alabeo
C_M	Coeficiente de momento de cabeceo
C_N	Coeficiente de momento de guiñada
c_w	Cuerda media aerodinámica del ala
C_X	Coeficiente de fuerza en el eje X
C_Y	Coeficiente de fuerza en el eje Y o lateral
C_Z	Coeficiente de fuerza en el eje Z
C_{L_0}	Coeficiente de sustentación para ángulo de ataque nulo
C_{L_α}	Derivada del coeficiente de sustentación respecto al ángulo de ataque

C_{l_β} Derivada del coeficiente de momento de alabeo respecto al ángulo de derrape

$C_{L_{\dot{\alpha}}}$ Derivada del coeficiente de sustentación respecto a la variación del ángulo de ataque

$C_{l_{\dot{\beta}}}$ Derivada del coeficiente de momento de alabeo respecto a la variación del ángulo de derrape

C_{l_p} Derivada del coeficiente de momento de alabeo respecto a la velocidad de alabeo

C_{L_q} Derivada del coeficiente de sustentación respecto a la velocidad de cabeceo

C_{l_r} Derivada del coeficiente de momento de alabeo respecto a la velocidad de cabeceo

$C_{l_{\delta A}}$ Derivada del coeficiente de momento de alabeo respecto ala deflexión de los alerones

$C_{L_{\delta E}}$ Derivada del coeficiente de sustentación respecto a la deflexión de los elevadores

$C_{l_{\delta R}}$ Derivada del coeficiente de momento de alabeo respecto ala deflexión del timón de cola

C_{M_0} Coeficiente de momento de cabeceo para ángulo de ataque nulo

C_{M_α} Derivada del coeficiente de cabeceo respecto al ángulo de ataque

$C_{M_{\dot{\alpha}}}$ Derivada del coeficiente de cabeceo respecto a la variación del ángulo de ataque

C_{M_q} Derivada del coeficiente de cabeceo respecto a la velocidad de alabeo

$C_{M_{\delta E}}$ Derivada del coeficiente de cabeceo respecto a la deflexión de los elevadores

C_{N_β} Derivada del coeficiente de guiñada respecto al ángulo de derrape

$C_{N_{\dot{\beta}}}$ Derivada del coeficiente de guiñada respecto a la variación del ángulo de derrape

C_{N_p} Derivada del coeficiente de guiñada respecto a la velocidad de alabeo

C_{N_r} Derivada del coeficiente de guiñada respecto a la velocidad de cabeceo

$C_{N_{\delta A}}$ Derivada del coeficiente de guiñada respecto a la deflexión de los alerones

$C_{N_{\delta R}}$ Derivada del coeficiente de guiñada respecto a la deflexión dl timón de cola

C_{Y_β} Derivada del coeficiente de fuerza lateral respecto al ángulo de derrape

$C_{Y_{\dot{\beta}}}$ Derivada del coeficiente de fuerza lateral respecto a la variación del ángulo de derrape

C_{Y_p} Derivada del coeficiente de fuerza lateral respecto a la velocidad de alabeo

C_{Y_r} Derivada del coeficiente de fuerza lateral respecto a la velocidad de cabeceo

$C_{Y_{\delta A}}$ Derivada del coeficiente de fuerza lateral respecto ala deflexión de los alerones

$C_{Y_{\delta R}}$ Derivada del coeficiente de fuerza lateral respecto ala deflexión del timón de cola

F_A Fuerza aerodinámica

F_T Fuerza de empuje

GS Proyección del vector velocidad sobre el plano XY

I_{xx} Momento de inercia en eje X

I_{xy} Producto de inercia de XY

I_{xz} Producto de inercia de XZ

I_{yy} Momento de inercia en eje Y

I_{yz}	Producto de inercia de YZ
I_{zz}	Momento de inercia en eje Z
K_d	Ganancia derivativa
K_i	Ganancia integral
K_p	Ganancia proporcional
L	Momento de alabeo
M	Momento de cabeceo
m	Masa
M_A	Momento aerodinámico
M_T	Momento asociado al empuje
N	Momento de guiñada
Nu	Ángulo de error en la trayectoria según la ley de navegación L1
p	Velocidad angular de alabeo
q	Velocidad angular de cabeceo
r	Velocidad angular de guiñada
S_w	Superficie alar
u	Velocidad lineal en el eje cuerpo X
V	Velocidad aerodinámica
v	Velocidad lineal en el eje cuerpo Y
w	Velocidad lineal en el eje cuerpo Z
x	Coordenada norte (north) en el sistema NED
y	Coordenada este (east) en el sistema NED
z	Coordenada abajo (down) en el sistema NED

Capítulo 1

Objetivos y alcance del proyecto

1.1 Objetivos principales

Los objetivos principales de este proyecto, que se corresponden al mismo tiempo con el objetivo final de cada una de las dos fases principales del proyecto (implementación y validación), son los siguientes:

- Implementar los sistemas de control y de navegación necesarios para el vuelo del UAV (*Unmanned Aerial Vehicle*) HERMES-UPV de ala fija en un microcontrolador.
- Simular correctamente el vuelo del UAV para el que se ha implementado el sistema de control y de navegación diseñado, modificando los parámetros necesarios para este correcto funcionamiento.

1.2 Objetivos secundarios

Dentro de los objetivos secundarios pueden incluirse las fases previas necesarias para poder abordar las tareas que llevan a los objetivos principales y también los hitos a los que se debe llegar para alcanzar tales objetivos:

- Obtención de los coeficientes aerodinámicos del UAV de HERMES-UPV.
- Elaboración del modelo dinámico del UAV a partir de modelos como el aerodinámico y el propulsivo.
- Linealización del sistema de ecuaciones del modelo, que incluye la obtención de los puntos de funcionamiento.
- Desarrollo del esquema en bloques de *Simulink* del modelo completo de la aeronave, por una parte del modelo no lineal y por otra del linealizado.

- Diseño del sistema de control y navegación, que implica expresar el sistema de ecuaciones que describen el comportamiento de la aeronave en forma matricial de espacio de estados y un posterior ajuste de los controladores (PIDs).
- Validación del sistema de control en *Simulink*.
- Introducción al software de piloto automático *ArduPilot*.
- Obtención de parámetros de funcionamiento del sistema de control y de navegación definido en el software de *ArduPilot*.
- Configuración del programa de control en tierra *QGroundControl* mediante los parámetros óptimos obtenidos a partir del sistema de control y de navegación diseñado.
- Simulación de vuelo en *QGroundControl* y en *FlightGear* para validar los sistemas implementados.

1.3 Alcance del proyecto

Como se ha visto en los objetivos, al finalizar el proyecto se contará con un sistema de control y de navegación diseñado para el UAV de HERMES-UPV que estará preparado para ser introducido en la placa de control y realizar pruebas en vuelos reales.

Tras el modelado del UAV, linealización del modelo, diseño de los sistemas de control y de navegación de la aeronave y validación de estos, se procederá a transformar los controladores diseñados en unos nuevos compatibles con las estructuras de los del piloto automático *ArduPilot* y, por último, a validarlos mediante herramientas de simulación en tiempo real, como *Software In The Loop* (SITL) o *Hardware In The Loop* (HITL). Una vez validados los resultados del proyecto, el cual se habrá desarrollado mediante el uso de múltiples programas informáticos, como son *Matlab*, *Simulink*, *QGroundControl* o *FlightGear*, el sistema de control final estará disponible como base para el equipo de trabajo de HERMES-UPV, para que pueda tanto emplearlo para el vuelo como aplicar actualizaciones que optimicen el control de las superficies y las estrategias de vuelo. Por tanto, en las simulaciones se deberá comprobar que al modificar las variables de control se obtienen repuestas que aseguren la estabilidad de la aeronave.

Por otro lado, este proyecto servirá para confirmar si la aeronave ha sido diseñada correctamente en relación a los requisitos de vuelo que deberá cumplir una vez esté completamente ensamblada.

En este proyecto no se pretende modificar el código libre del piloto automático para mejorarlo, sino que se limitará a emplear el código existente para hacer posible que la aeronave vuele y pueda cumplir en el futuro, tras las modificaciones necesarias que tengan lugar fuera del presente trabajo, con la misión del proyecto HERMES-UPV.

Capítulo 2

Introducción y antecedentes

2.1 Introducción

El presente trabajo forma parte de un proyecto de carácter universitario dentro de la Universitat Politècnica de València (*UPV*) llamado HERMES-UPV, donde alumnos de diferentes ámbitos trabajan en colaboración con el Instituto Universitario de Automática e Informática Industrial (*ai2*) y el Grupo de Control Predictivo y Optimización Heurística (*CPOH*) para diseñar, fabricar y, como objetivo final, volar un UAV (*Unmanned Aerial Vehicle* o Vehículo Aéreo No Tripulado) de ala fija.

La motivación de la generación del proyecto HERMES-UPV es la presentación de la aeronave, que toma el mismo nombre que el proyecto, al concurso internacional *UAS Challenge* [1], con base en el Reino Unido, que propone como medio de unir el ámbito académico con el profesional el diseño de un UAV completamente autónomo que sea capaz de llevar a cabo misiones de ayuda humanitaria. Por tanto, para cumplir con esta misión, la aeronave deberá ser en primer lugar estable, pero también tener la mejor relación posible entre alcance y autonomía y cumplir con las tareas que se le especifiquen, como barrido de áreas, navegación siguiendo waypoints (coordenadas de referencia), descarga de paquetes de ayuda humanitaria y retorno a la base de origen. Todos estos aspectos se tienen en cuenta en las diferentes fases de diseño de la aeronave.

En el grupo de trabajo que conforma HERMES-UPV se encuentran diversos subgrupos, dedicados cada uno a tareas específicas, como son el diseño estructural y aerodinámico de la aeronave, fabricación y ensamblaje de esta, o el control y la simulación del modelo dinámico. El presente trabajo abarca la primera fase del proyecto de diseño de la aeronave, pero está enfocado específicamente al sistema de control y navegación (el trabajo tiene lugar dentro del Grupo de Modelado Dinámico, Simulación y Control de HERMES-UPV).

Por tanto, la aeronave sobre la cual se implementa el sistema de control y navegación en cuestión es proporcionada por el grupo de trabajo de diseño en CAD, y a partir de este diseño se desarrolla un modelo aerodinámico que sirve como base del sistema de control. Además del diseño de la aeronave y del sistema de control, es de vital importancia la validación de los resultados obtenidos, en primer lugar mediante simulaciones, que permiten comprobar el funcionamiento de la aeronave sin el riesgo de destruirla, como puede ser con el uso de un sistema *Harware In The Loop*, y más adelante en túnel de viento y ensayos de vuelo.

En relación a este Trabajo de Fin de Grado existen otros tres que han sido desarrollados de forma paralela y sobre el mismo sistema de control del HERMES-UPV, todos ellos dentro del Grupo de Modelado Dinámico, Simulación y Control mencionado anteriormente:

- "Desarrollo del Modelo Dinámico No Lineal de una aeronave no tripulada para el proyecto HERMES-UPV", por Álvaro Goterris Fuster [2].
- "Diseño del sistema de control y navegación para una aeronave no tripulada dentro del proyecto HERMES-UPV", por Daniel Villalibre Vilariño [3].
- "Implementación de un sistema de simulación Hardware-In-The-Loop (HIL) para la realización de tests no destructivos de una aeronave no tripulada dentro del proyecto HERMES-UPV", por Víctor Ribera Esplugues [4].

Especialmente los dos primeros trabajos están directamente relacionados con el presente, ya que son pasos necesarios para la implementación del sistema de control y navegación. Dado que los cuatro trabajos se han desarrollado al mismo tiempo, todos los componentes han participado de un modo u otro en cada uno de los proyectos de sus compañeros, aunque priorizando en todo momento el desarrollo del suyo propio.

2.2 Estado del arte

Como introducción a la aeronave de este proyecto, se propone una serie de definiciones y clasificaciones de los UAV, además de otros conceptos relacionados con el control de aeronaves, que pueden ayudan a comprender mejor cómo es el ámbito en el que se incluye el trabajo.

2.2.1 Definición de UAV y clasificación

Los UAV o Vehículos Aéreos No Tripulados son aeronaves que no necesitan un piloto a bordo que controle su comportamiento ni llevan pasajeros, aunque sí pueden llevar carga. Por tanto, este tipo de aeronave puede recibir las órdenes de control desde dentro del propio vehículo con un piloto automático, o recibirlas desde el exterior mediante un control remoto, pero en ningún caso a través de un humano a bordo. Son aeronaves que deben ser controlables, lo que implica por regla general que se les debe suministrar potencia para accionar los controles y para modificar la velocidad de vuelo.

Otro aspecto importante de los UAV es que se diseñan para ser recuperables, es decir, a diferencia de los misiles y otros vehículos balísticos, estos deben ser capaces de despegar y aterrizar de tal forma que puedan volver a utilizarse [5].

Este tipo de aeronave recibe además otros nombres, como son UAS (*Unmanned Aerial System*), que se suele emplear como sinónimo de UAV (únicamente el vehículo) o también haciendo referencia a todos los sistemas necesarios para que la aeronave pueda volar, como serían elementos relacionados con lanzaderas y la recuperación del vehículo al finalizar el vuelo, o los sistemas de comunicación; también reciben el nombre de RPV (*Remotely Piloted Vehicles*) o RPAS (*Remotely Piloted Aircaft System*), con diferencias entre ellos similares a las que habría entre UAV y UAS pero con la gran diferencia respecto a los UAV de no incluir vehículos completamente autónomos, por lo que todos los RPV son UAV pero no a la inversa; o dron, que es el nombre con el que se conoce popularmente a los UAV.

Un UAV está compuesto por diferentes subsistemas conectados entre sí [6]:

- La propia **aeronave**, que incluye el sistema propulsivo, controles y sistemas eléctricos.
- GCS (*Ground Control Station*) es la **estación de control en tierra**, que se encarga de procesar y mostrar la información recibida desde la aeronave, sea en forma de datos o de imágenes, por ejemplo, y de planificar misiones y controlar el UAV cuando sea necesario. Es, en términos generales, la conexión entre un UAV y los humanos, sea para visualizar datos o para darle instrucciones. Dos GCS de acceso libre y uso habitual hoy en día son *QGroundControl* y *Mission Planner*.
- El equipamiento de **lanzamiento y recuperación**, que depende del tipo de aeronave y su forma de despegar y aterrizar, pero puede incluir lanzaderas pirotécnicas o neumáticas, redes para recuperar aviones de ala fija, paracaídas, o sistemas de amarre.
- Payload o **carga de pago**, que puede incluir cámaras, radares, equipos de medidas meteorológicas, o paquetes que se entregan en un destino fijado.
- El **sistema de enlace de datos** (*data link*) es el responsable de comunicar los diferentes subsistemas entre sí de forma continua o bajo demanda, por lo que es imprescindible si hay una estación de control en tierra y se busca conocer el estado de la aeronave y transmitir instrucciones. La parte del sistema de datos que se encuentra en la aeronave incluye antenas, transmisores y receptores.

Aplicaciones

El hecho de no necesitar un piloto a bordo de un UAV conlleva diversas ventajas y desventajas decisivas para las aplicaciones de estos vehículos.

Una importante ventaja respecto a aeronaves tripuladas es, especialmente en aplicaciones militares o por ejemplo de control de incendios, la capacidad de realizar misiones de riesgo que resultarían peligrosas para un piloto. Otras ventajas son el menor impacto medioambiental, ya que son aeronaves más pequeñas y ligeras que pueden funcionar con energías más limpias, o el ahorro económico y de tiempo que implica no tener que depender directamente de un humano durante el vuelo.

Las desventajas de los UAV residen esencialmente en la posibilidad de que ocurran fallos de comunicación durante el vuelo, pero también en complicaciones derivadas de condiciones climatológicas adversas o por la dificultad de integración de estas aeronaves en el espacio aéreo.

La principal división entre diferentes aplicaciones de los UAV es según si pertenecen al ámbito militar o al ámbito civil, aunque también se puede tener en cuenta una categoría adicional de UAV relacionados con el medio ambiente.

Los UAV militares son los que más se han desarrollado a lo largo de los años debido a los conflictos bélicos, especialmente del siglo pasado, y tienen las siguientes aplicaciones:

- **Reconocimiento**: consiste en la recopilación de información mediante interceptación de señales, captura de imágenes y el uso de otros sensores sobre superficies que se quiere vigilar. Un UAV suele ser más difícil de detectar en estas aplicaciones que una aeronave tripulada, lo que aporta grandes ventajas.

- **Ataque**: los UAV empleados para misiones de ataque lanzan misiles o bombas durante el vuelo y tienen la ventaja de que no hay un piloto que corra riesgos.
- **Aprovisionamiento**: los UAV son de gran ayuda en la provisión de medicamentos u otros tipos de cargas de pago en zonas de difícil acceso, como pueden ser misiones de rescate o en zonas de guerra.
- **Entrenamiento de pilotos**: un uso que han tenido los UAV desde hace décadas es el de blanco para pilotos en formación dentro del ámbito militar.

En los últimos años ha aumentado notablemente el uso de vehículos no tripulados en el ámbito civil, debido a que son más accesibles y configurables según las necesidades del usuario.

Las principales aplicaciones civiles incluyen trabajos de logística, agricultura, reparto de paquetería, vigilancia, gestión de emergencias, minería, construcción o fotografía [7]. Más recientemente se ha popularizado la compra de los UAV para usos de ocio o como juguetes, aplicaciones que han hecho que la población en general pase a llamarlos drones en lugar de UAV.

Por último, las aplicaciones relacionadas con el medio ambiente pueden ser la monitorización de la calidad de aire, tierra y agua, inspección de montañas o conservación de la biodiversidad y hábitats.

Clasificación

Los UAV pueden clasificarse atendiendo a diferentes criterios, de los cuales varios se van a explicar a continuación.

Una primera clasificación es según la aerodinámica en la que se basa la aeronave, es decir, qué elementos proporcionan la sustentación y el empuje y cómo se controla su vuelo. Esta clasificación incluye aviones de ala fija, multicópteros, choppers, planeadores o VTOL (*Vertical Take Off and Landing*) entre otros. Por un lado, una aeronave de ala fija necesita una distancia en tierra para poder despegar y aterrizar, no puede permanecer quieta en el mismo punto durante del vuelo y su actitud (movimiento alrededor de sus ejes) es controlada por las superficies de control, mientras un cuadricóptero, por ejemplo, emplea sus diferentes hélices para controlar todos los ejes de la aeronave y así poder despegar y aterrizar en vertical y mantenerse volando en un punto fijo.

Una clasificación habitual también es según si el despegue y aterrizaje son verticales u horizontales, como muestra la figura 2.1 [8].

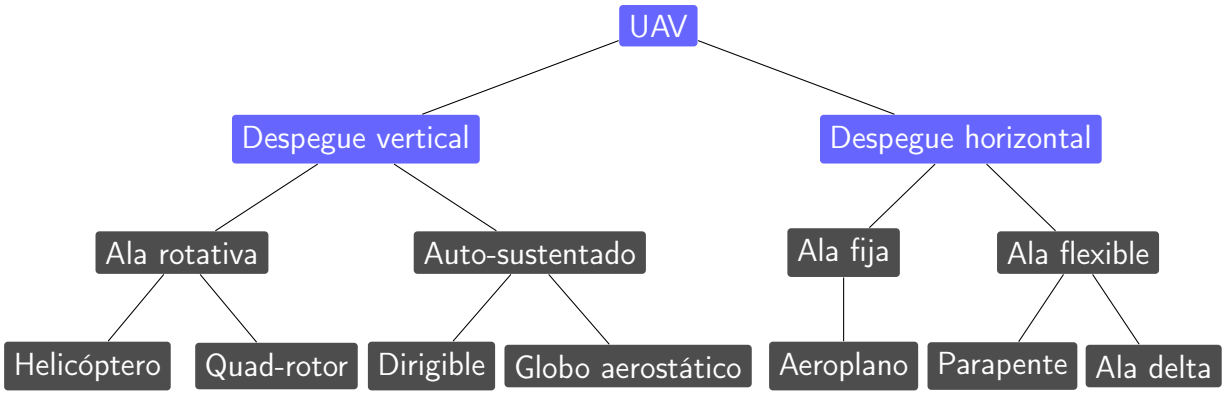

Figura 2.1: Esquema de clasificación de los UAV según despegue y aterrizaje

Por otro lado, también se puede clasificar un UAV en base a las capacidades de vuelo, es decir, la relación entre alcance, altitud, autonomía y carga. En la tabla 2.1 [8] se puede observar esta clasificación, que incluye tanto UAVs civiles como de aplicaciones militares.

Condiciones de vuelo UAV				
Categoría	Alcance (km)	Altitud (m)	Autonomía (h)	Carga máxima (kg)
Micro	< 10	250	1	< 5
Mini	<10	150 a 300	< 2	< 30
CR	10 a 30	3000	2 a 4	150
SR	30 a 70	3000	3 a 6	200
MR	70 a 200	5000	6 a 10	1250
LADP	> 250	50 a 9000	0.5 a 1	350
MRE	> 500	8000	10 a 18	1250
LALE	> 500	3000	> 24	< 30
MALE	> 500	14000	24 a 48	1500
HALE	> 2000	20000	24 a 48	12000
UCAV	1500	10000	2	10000
LETH	300	4000	3 a 4	250
DEC	0 a 500	5000	< 4	250
STRATO	> 2000	20000 a 30000	> 48	No disponible
EXO	No disponible	> 30000	No disponible	No disponible

Cuadro 2.1: Clasificación de UAV según capacidades de vuelo

2.2.2 Historia

Aunque los vehículos aéreos no tripulados tal como se conocen hoy en día se han desarrollado únicamente en décadas recientes, los vuelos no tripulados llevan teniendo lugar desde hace varios siglos. A través de un breve análisis de la historia de los UAV se puede comprender cómo han evolucionado a lo largo de los años para convertirse en los vehículos actuales.

La historia de los UAV puede dividirse en tres grandes bloques, el primero más extenso pero con un desarrollo más pausado, que abarca hasta el final del siglo XIX; en el segundo bloque, que ocupa la primer mitad aproximadamente del siglo XX, se puede considerar que está el origen de los UAV modernos; y en el tercer periodo de tiempo, desde la segunda mitad del siglo XX hasta la actualidad, destaca la importante y rápida evolución de estos vehículos.

Suele introducirse el comienzo de la historia de los UAV como una época alrededor de 400 años antes de Cristo, donde al mismo tiempo en la Antigua Grecia y en China comenzaron a fabricarse objetos que podían levantarse del suelo; se tiene constancia de que en China se comenzó a experimentar en esa época, mayoritariamente con intenciones militares, con máquinas como globos aerostáticos, cohetes o cometas [9]. Más adelante, a partir del siglo XIV se comenzó a entender cómo funcionaba el vuelo atmosférico a partir principalmente del estudio de las aves, y se intentó imitar su comportamiento con máquinas que simulaban el batimiento de las alas de un pájaro o el giro helicoidal que realizaban ciertas semillas en el aire, como hizo Leonardo Da Vinci (1452-1519).

Pero estos inventos no son lo que hoy conocemos como UAV, ya que los conocimientos sobre aerodinámica y propulsión eran escasos y estos vehículos no eran controlables. Hay que avanzar hasta principios del siglo XX para su verdadera introducción, donde se considera que empieza la segunda etapa mencionada anteriormente de su historia.

A principios del siglo XX los conocimientos sobre aerodinámica se habían ampliado y tuvo lugar el histórico vuelo de los hermanos Wright, aspectos que favorecieron el desarrollo durante los años posteriores de las aeronaves. Además, como ha sido habitual a lo largo de la historia, los principales avances tecnológicos relacionados con la aviación tuvieron lugar durante un conflicto bélico: la Primera Guerra Mundial. En el año 1916 se demostró el funcionamiento del *Aeroplano Automático Hewitt-Sperry*, posible gracias al trabajo de Elmer Ambrose Sperry con giroscopios para conseguir estabilizar la aeronave, y en el mismo año Archibald Low desarrolló el primer blanco aéreo de entrenamiento *Aerial Target*. Un año después, en el 1917, Charles Kettering presentó el *Kettering Aerial Torpedo*, también conocido como *Kettering Bug*, que era un UAV que podía programarse antes del vuelo para sobrevolar un objetivo y una vez encima deshacerse de sus alas para caer en picado y comportarse como una bomba.

Después de unos años en los que se perdió el interés por la idea de un vehículo automatizado, debido a las barreras tecnológicas del momento, se retomó el desarrollo durante las décadas de 1920 y 1930 de más UAV categorizados como blancos para entrenamiento, como el *Queen Bee* (figura 2.2 [1]) del Reino Unido en 1933, que demostró su efectividad al sobrevivir a decenas de misiones de entrenamiento.

Figura 2.2: UAV *Queen Bee K227*

Durante esta época no solo se desarrolló el vehículo en sí, sino también los demás componentes que conforman un sistema aéreo no tripulado. Un ejemplo del desarrollo de estos sistemas fue en 1924 el primer vuelo dirigido por radio control de la historia, de un UAV de Archibald Low, quien desarrolló el primer *data link*.

La tercera etapa comienza en la segunda mitad del siglo XX, tras la Segunda Guerra Mundial, después de la cual la atención se centró en los UAV de reconocimiento militar, como el *SD-1* de mitades de la década de los 50, que era controlado por radio control y contaba con una cámara. Además, la tecnología aplicada a misiles dirigibles durante la Segunda Guerra Mundial sirvió para mejorar el control de los UAV en los años posteriores [6].

El uso de los UAV de reconocimiento se extendió durante la Guerra de Vietnam (1955-1975), y estuvieron preparados para ser usados durante la Crisis de los Misiles de Cuba (1962), aunque finalmente no se emplearon. Estos últimos UAV, basados en los *target drones* de Ryan Firebee llamados *Fire Fly* fueron los principales vehículos aéreos utilizados en el sudeste asiático durante esta época.

En la década de los 70 se volvió a dar importancia a los UAV militares, y un ejemplo es el TADARS (T*arget Acquisition/ Designation and Aerial Reconnaissance Sysyem*), también llamado *Aquiles*, que estuvo en funcionamiento durante años en los Estados Unidos. En 1984 se consiguió un gran avance en los UAV de reconocimiento al demostrarse con el *Gray Wolf* el funcionamiento en ope-

[1] Recuperado de http://www.vintagewings.ca

raciones nocturnas. En las operaciones de la OTAN en Bosnia en el año 1995 fue imprescindible el reconocimiento nocturno para cumplir con las misiones, como el que realizaba el UAV *Predator*.

Otro escenario que favoreció el desarrollo de los UAV militares fue la guerra de Irak (2003-2011), donde se dio el paso de vehículos de reconocimiento y vigilancia que simplemente ayudaban en ciertas misiones, a UAV que eran armas decisivas en el resultado de los conflictos. Se usaron UAV como el *Pioneer*, *Shadow*, *Hunter* o *Pointer*.

En la actualidad, además de seguir empleándose los UAV en el ámbito militar, que siguen desarrollándose para cumplir misiones más exigentes, ha surgido un gran interés por los vehículos no tripulados en el ámbito civil. Como se ha visto en las aplicaciones, hoy en día los UAV también se utilizan para estudiar y proteger el medio ambiente, para ayudar en situaciones de emergencia o como entretenimiento.

2.3 Piloto automático

La forma de implementar un sistema de control del vuelo y de la navegación en este proyecto es mediante el uso de un piloto automático, el cual también incluya la opción de cambiar de modos de vuelo, como serían por ejemplo los modos manual o estabilizador, para así poder dirigir la aeronave mediante radio control o que navegue sin necesidad de pilotaje. En este apartado se va a explicar en qué consiste un piloto automático o *autopilot* y las opciones que existen actualmente para conseguir que un UAV vuele de forma autónoma.

Un piloto automático es un conjunto de software y hardware que genera instrucciones para llevar a una aeronave por una trayectoria deseada, por lo que incluye el cálculo de esta trayectoria y la determinación de las acciones necesarias para seguirla. Esta trayectoria puede estar determinada por una serie de waypoints o por comportamientos determinados, como mantener los ángulos de cabeceo, de alabeo o de guiñada, mantener la altitud y la velocidad, o realizar despegues y aterrizajes de forma automática.

Para que funcione un piloto automático es necesaria su conexión con una estación de control en tierra que le indique el modo de vuelo o los waypoints a seguir, por ejemplo; también una conexión GPS para conocer en todo momento la ubicación de la aeronave; y conexión con los servomotores para que lleven a cabo las acciones de control.

Por otro lado, un sistema de piloto automático puede dividirse en una parte de observación del estado de la aeronave, que puede incluir unidades IMU o sensores infrarrojos, entre otros, que combinados con la señal GPS aportan toda la información necesaria sobre la posición y orientación del UAV [10]; y en una parte dedicada al control, por lo general mediante la realimentación de los estados (la salida se redirige a la entrada para corregir los errores, es decir, la entrada depende del estado anterior), que puede estar basada en diferentes tipos de controladores, como se verá en el apartado 2.3.1.

Existen dos clases de pilotos automáticos disponibles en la actualidad para los UAV de ala fija de uso civil. Una clase son los que se pueden comprar como un conjunto de software y hardware, donde se incluyen en la misma placa el procesador y todos los sensores necesarios, con el software del controlador de vuelo y de navegación ya implementados en el hardware. Varios ejemplos de estos pilotos automáticos son las series de *Procerus Kestrel*, *MicroPilot* y *Cloud Cap Piccolo*, que cuestan varios miles de dólares americanos [11], o una versión más económica como el autopilot *UNAV 3500*, de 400 dólares.

La otra clase, que ha surgido con la bajada de precios de los componentes físicos de los UAV por su popularización, es la que incluye los pilotos automáticos de código abierto, que incluyen un software del control que puede ser modificado libremente por el usuario y que aportan una gran libertad a la hora de escoger sensores y procesadores, y a la hora de personalizar el piloto automático según los requerimientos de vuelo. Por tanto, debido al aspecto económico y al acceso disponible al código, el uso de este tipo de pilotos automáticos es el más apropiado para el ámbito académico, de investigación y de uso personal. En el desarrollo de este trabajo se ha optado por usar uno de este tipo, como se explicará en el capítulo 5.

2.3.1 Estrategias de control para un UAV

Aunque en este trabajo se va a emplear la primera estrategia mostrada, existen una gran variedad de estrategias diferentes que pueden aplicarse a los pilotos automáticos [11]:

- **PID**

  El mecanismo de control PID (Proporcional, Integral y Derivativo) se basa en la desviación existente entre un valor medido y un valor objetivo de una variable, y se compone de tres partes: la parte proporcional intenta que el error en estado estacionario disminuya lo máximo posible, produciendo una señal de control que es proporcional a la señal de error mediante la constante de proporcionalidad *Kp*; la parte integral busca disminuir y eliminar el error estacionario que la parte proporcional no puede corregir, y lo hace mediante la suma a la acción proporcional de la integral del error estacionario por la constante de integración *Ki*, a modo de promedio del error que se ha ido acumulando en el pasado debido al offset del control proporcional; y la parte derivativa, que se encarga de corregir los errores variables, como las perturbaciones, con la misma velocidad con la que se producen para que no aumenten, mediante la suma a las otras partes de control del producto de la derivada temporal del error por una constante derivativa *Kd*.

  Los parámetros de esta estrategia de control aplicada a los pilotos automáticos se pueden configurar antes del vuelo y reajustar durante este para mejorar el comportamiento. Estos controladores son los más empleados hoy en día en los autopilots comerciales debido a su fácil implementación y a que están muy desarrollados, pues el primer PID fue desarrollado en el 1911 por Elmer Sperry, ingeniero mencionado en el subapartado de la historia de los UAV en este capítulo. De todos modos, no es el controlador más robusto que existe y han surgido nuevas estrategias de control desde entonces.

- ***Fuzzy Based***

  Los controladores basados en la lógica *fuzzy* [12] o lógica difusa son una alternativa más moderna a los controladores tradicionales y que actualmente está en desarrollo para pilotos automáticos. La lógica difusa se puede comparar con la inteligencia artificial en el sentido de que no se basa únicamente en valores de entradas de 0 o 1 de la lógica booleana, sino que los valores de las variables pueden estar en un rango entre 0 y 1, ambos incluidos, mostrando el nivel de incertidumbre en las respuestas como lo haría una mente humana, que necesita contextualizar cada situación. Los resultados obtenidos con controladores de lógica difusa se obtienen más rápido y son más precisos que los obtenidos con controladores tradicionales PID, aunque son más difíciles de comprender.

- **Basado en redes neuronales (NN)**

  Mientras en la lógica difusa el control depende del modelo matemático del sistema al que controla, ya que relativiza las entradas, los controladores basados en redes neuronales (*Neural Network*) [11, 13] no dependen de ningún modelo, y además, no se limitan únicamente a sistemas SISO (*Single Input Single Output*). En los pilotos automáticos, el controlador basado en redes neuronales suele ser directo, por lo que aprende constantemente de la evaluación directa de la precisión del control y es capaz de adaptarse a situaciones nuevas.

- **Regulador lineal-cuadrático (LQR)**

  Un controlador LQR (*Linear-Quadratic Regulator*) [14] está basado en el problema LQ, donde las dinámica del sistema está descrita por un serie de ecuaciones diferenciales lineales y existe una función de coste cuadrática que se quiere minimizar. La solución del problema proporciona la configuración de los parámetros del controlador que minimizan las desviaciones de las variables de control. Este método de control es automático y sólo necesita la definición de la función de coste una vez se ha implementado el algoritmo, pero tiene el inconveniente de que desde el exterior no se ve claramente la relación que existe entre los diferentes parámetros.

- **Control predictivo por modelo (MPC)**

  Se trata de una estrategia de control empleada generalmente en sistemas dinámicos complejos, donde el modelo es capaz de predecir los cambios que pueden sufrir las variables dependientes del sistema causados por cambios que tienen lugar en las variables independientes, como las perturbaciones. Este controlador también busca minimizar una función de coste para obtener sus parámetros mediante procesos iterativos.

Capítulo 3

Plataforma de vuelo

3.1 Aeronave

Como se ha explicado en el apartado anterior, la aeronave en la que se implementa el sistema de control objeto de estudio ha sido diseñada dentro del proyecto HERMES-UPV, por lo que el presente trabajo parte de su arquitectura para diseñar e implementar el control y la navegación. El sistema de control se ha diseñado con el objetivo de conseguir que la aeronave sea estable, ligera, resistente y con una buena autonomía, al igual que el sistema de navegación empleado busca aprovechar las características de la aeronave para optimizar las trayectorias durante el vuelo.

Como se puede ver en la figura 3.1, se trata de una aeronave de una única ala fija, en configuración de ala alta y recta, que cuenta con un timón de cola, estabilizadores horizontales en configuración convencional y alerones en el ala, superficies accionadas por servos digitales. El tren de aterrizaje es de tipo triciclo, con el tren de morro dirigible para facilitar las maniobras en tierra.

Figura 3.1: Diseño exterior de la aeronave HERMES-UPV

Por otro lado, el empuje es proporcionado por un motor del fabricante *T-Motors* con una única hélice de dos palas APC17x10 (figura 3.2), y que es alimentado por una batería LiPo a través de un variador de velocidad. Más adelante, en la sección 3.2 se explicarán las características de estos componentes.

Figura 3.2: Hélice APC17x10 empleada

Tanto las dimensiones como la masa del UAV vienen limitadas por las condiciones que impone el concurso para el que la aeronave ha sido creada. Por tanto, una vez diseñada la aeronave, sus características dimensionales y el tensor de inercias se pueden obtener a partir del diseño en CAD realizado dentro del proyecto, que tienen en cuenta el efecto de todos los componentes y son las mostradas en la tabla 3.1:

Envergadura	2 m
Cuerda alar	0.25 m
Espesor máximo de la cuerda media	0.034 m
Superficie alar	0.5 m2
Aspect Ratio ala principal	8
Longitud fuselaje	1.25 m
Masa	7.443 kg

$$I = \begin{bmatrix} 0.609 & 0 & 0.093 \\ 0 & 1.294 & 0 \\ 0.093 & 0 & 1.718 \end{bmatrix} kgm2$$

Cuadro 3.1: Características de la aeronave y tensor de inercias

3.2 Hardware

El sistema de control diseñado para la aeronave se implementará en esta a través de un controlador, en este caso la placa *BeagleBone Blue* de *BeagleBoard*. Este ordenador hace posible la conexión entre los diferentes componentes electrónicos y la aeronave, y también la comunicación entre la aeronave y tierra mediante componentes de radio control. En este apartado se introducen las diferentes partes del hardware debido a que afectan al posterior modelado en cuanto a dimensiones y peso y también a limitaciones de funcionamiento.

3.2.1 BeagleBone Blue

Una de las principales características de esta placa que hacen que se haya escogido para el proyecto es que, al ser esencialmente un ordenador en una única placa, tiene mayor capacidad computacional que otros microcontroladores con un tamaño similar (la placa mide 8.89cm x 5.46cm) [15].

Otras características de este controlador que lo hacen interesante como ordenador de a bordo para un UAV son el uso del sistema operativo *Linux*, alrededor del cual está desarrollado el ordenador y que es de código abierto, y la posibilidad de conectar todos los periféricos necesarios para que el sistema de control funcione en la misma placa, como se puede ver en la figura 3.3[1]:

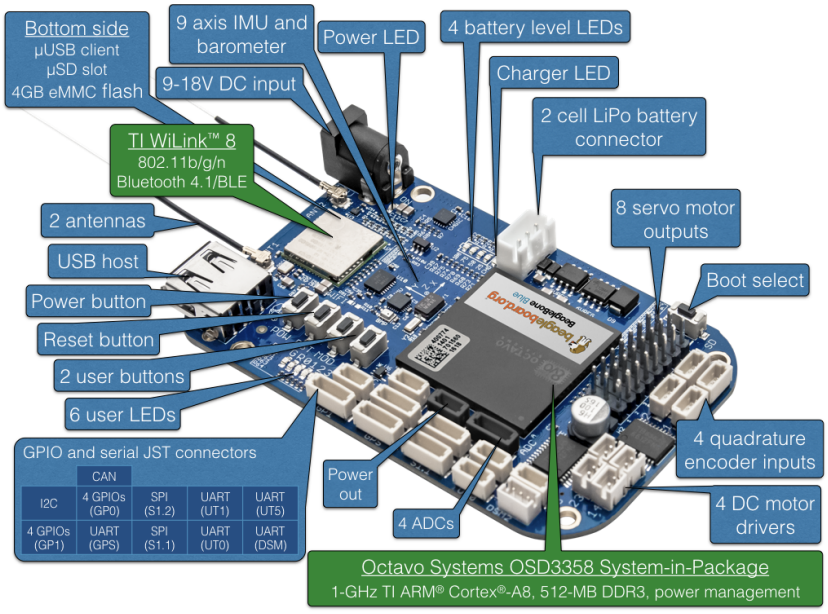

Figura 3.3: Puertos y conexiones de la placa BeagleBone Blue

Los principales elementos de la placa son los siguientes:

- Procesador *Octavo Systems* OSD3358 1GHz *ARM Cortex-A8*, con RAM DDR3 de 512MB.
- Conexión a batería LiPo de 2 celdas.
- Entrada de alimentación de 9 a 18V.
- Conexión wifi 802.11bgn, Bluetooth 4.1 y BLE.
- 8 salidas de 6V para servomotores.
- 4 salidas para motores DC.
- IMU (*Inertial Measurement Unit*) MPU9250 de 9 ejes (acelerómetro, giroscopio y magnetómetro).
- Barómetro BMP280.
- 11 luces LED y 2 botones programables.

[1] Recuperado de https://github.com/beagleboard/beaglebone-blue/wiki/Pinouts

– Conexiones de tipo GPIO (*General Purpose Input/Output*) y JST (conectores ampliamente utilizados para baterías y para servos, entre otras aplicaciones) que se pueden emplear para conectar más periféricos y buses, como GPS, conexión radio DSM2, UARTs, SPI, I2C...

3.2.2 Sensores

Debido a que la aeronave no se ha terminado de ensamblar durante la elaboración de este proyecto y aún cuando esté terminada sólo se dispondrá de una de ellas, se va a proceder a validar la implementación del sistema de control mediante una simulación en *QGroundControl* (QGC) para evitar perder el UAV en caso de fallo. Por tanto, los sensores reales serán reemplazados por los proporcionados en la simulación HIL dentro de QGC. De todos modos se va a explicar brevemente cuáles son estos sensores, tanto los que incluye la placa de control como los periféricos necesarios para volar, pues el sistema de control los tiene en cuenta, aunque no se detallará su instalación ni calibración.

IMU

La IMU MPU9250 integrada en la placa está compuesta por un acelerómetro de 3 ejes, giroscopio de 3 ejes y magnetómetro de 3 ejes, por lo que se puede decir que es una IMU de 9 ejes. Cuenta con un conversor analógico a digital de 16 bits para cada uno de los 9 ejes que digitalizan las salidas de cada unidad. Se puede elegir entre varios rangos de medida de los sensores, que van entre los $\pm 2g$ y $\pm 16g$ en el acelerómetro, entre $\pm 250°/s$ y $\pm 2000°/s$ en el giroscopio y los $\pm 4800 \mu T$.

Barómetro

El barómetro BMP280 es un sensor de presión absoluta empleado habitualmente en aplicaciones como teléfonos móviles o como en este caso vehículos autónomos, ya que por su bajo consumo puede alimentarse con las baterías de los dispositivos. Su precisión es adecuada para un UAV, ya que el error es de $\pm 0.12 hPa$, lo que equivale a un rango de error en la medida de la altitud de 1 metro.

3.2.3 Motor y variador de velocidad

El motor empleado en el proyecto es el AT4130 *Long Shaft* de *T-Motor* (figura 3.4) en su configuración de KV $300\ rpm/voltio$ y sin escobilla, adecuado junto con la hélice para las dimensiones y requisitos de funcionamiento de la aeronave.

Como se explicará en el apartado de modelado (sección 3.3), se puede obtener un modelo propulsivo aproximado a partir de estudios realizados en banco de ensayos por el propio fabricante disponibles en su página web. Este modelo se empleará para elaborar el sistema de control y después será simulado su aporte de empuje en QGC.

Respecto al variador de velocidad o ESC (*Electronic Speed Controller*), este es necesario para transformar la señal PWM (*Pulse Width Modulation*) de salida del controlador en la potencia necesaria para que el motor gire a la velocidad deseada. El ESC que se emlpeará en el proyecto es el *Favourite FVT Swallow Series 100A 2-6S*, que no tiene escobillas.

Figura 3.4: Motor AT4130 K230 de T-Motors

3.2.4 Servomotores

Los servomotores son dispositivos encargados de accionar las superficies de control (alerones, elevadores y timón de cola) a partir de las señales de tipo PWM que transmite el controlador. Los servos escogidos para el proyecto son los KST DS225MG HV de 10mm de espesor y peso de 26g, y los KST DS215MG HV V3 de 12mm de espesor y peso de 20g.

3.3 Modelado

Para poder abordar el problema del diseño de un sistema de control para un UAV, primero es necesario desarrollar un modelo matemático que represente de la forma más precisa posible el comportamiento dinámico que tendrá la aeronave durante el vuelo. El modelado se va a realizar a partir de las denominadas ecuaciones de Bryan, teniendo en cuenta 6 grados de libertad (3 de traslación y 3 de rotación) y obteniendo mediante relaciones teórico-empíricas, ensayos, técnicas de CFD o programas específicos los coeficientes aerodinámicos de estas ecuaciones diferenciales.

3.3.1 Ejes de referencia y ángulos de Euler

En primer lugar es necesario definir el sistema de ejes de coordenadas que se empleará en las ecuaciones para describir correctamente la dinámica del sistema. Existen varios tipos de ejes de coordenadas, de los cuales los más importantes se muestran a continuación, ya que a lo largo de este proyecto no se ha empleado un único sistema de referencia.

Ejes cuerpo

En este sistema de referencia el origen está aplicado en el centro de masas y los ejes se mantienen fijos en la aeronave de la manera que muestra la figura 3.5, donde el eje Xb es el longitudinal, que tiene la dirección del morro del avión; el eje Yb es perpendicular al Xb y la dirección es la del ala derecha; y el eje Zb, que es perpendicular a los dos anteriores y se dirige del centro de masas hacia abajo del avión. Como se verá en el siguiente apartado de los ángulos de Euler y en la figura 3.5 [2], estos ejes también suelen llamarse como los ángulos que se miden alrededor de ellos: roll, pitch y yaw. Para el modelado de una aeronave este tipo de sistema de referencia es el más adecuado, pues se mantiene fijo al avión y es no-inercial (tiene aceleraciones y rotaciones), y puede cuantificarse su traslación y rotación respecto al sistema de coordenadas inercial que estaría fijado a la Tierra. Por tanto, las variables de estado del sistema se medirán sobre el sistema de referencia de ejes cuerpo.

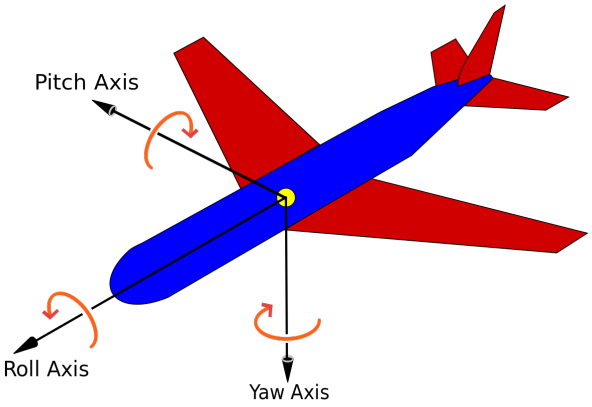

Figura 3.5: Ejes cuerpo de una aeronave

Otros ejes de referencia

Otro sistema de referencia sobre el que se puede medir es el de ejes viento, en el cual el eje Xw tiene la dirección del vector de velocidad aerodinámica de la aeronave, el eje Zw se encuentra en el plano de simetría de la aeronave mirando hacia abajo y el eje Yw es perpendicular a los otros dos ejes formando un triedro.

Por otro lado, los sistemas de ejes tierra son un tipo de sistema de referencia considerado inercial y fijo en la superfice terrestre, donde las direcciones de sus ejes son el Norte para Xt, Este para Yt y Zt perpendicular a los otros dos y hacia abajo, estando el origen de coordenadas en un punto fijo de la tierra con la cota cero al nivel medio del mar.

Por último, otro sistema de referencia de interés en el ámbito aeronáutico es el de ejes horizonte local, empleado en aspectos relacionados con la navegación, ya que todos sus ejes son paralelos a los del sistema de ejes tierra pero su origen de coordenadas está fijo a la aeronave, como muestra la figura 3.6. Este sistema de referencia puede llamarse también de forma abreviada NED (*North, East, Down*).

[2] Recuperado de https://es.wikipedia.org/wiki/Din%C3%A1mica_del_vuelo_(aeronaves_de_ala_fija)

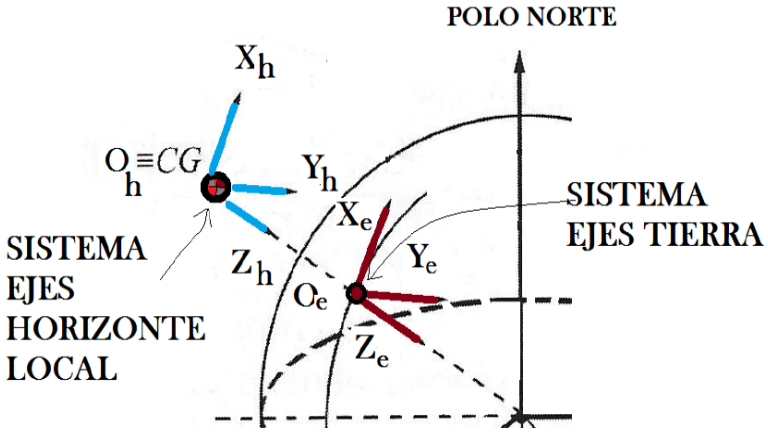

Figura 3.6: Ejes horizonte local de una aeronave

Ángulos de Euler

Estos ángulos definen la actitud de un avión, es decir la orientación de sus ejes cuerpo de referencia respecto a otros ejes inerciales: los ángulos de Euler son las rotaciones consecutivas a las que se someten los ejes cuerpo desde las direcciones de los inerciales. Estos tres ángulos son ϕ (phi), θ (theta) y ψ (psi), que se miden respectivamente sobre los ejes X, Y y Z, y que aplicados en este orden deshacen la transformación de rotación de los ejes cuerpo. Estos tres ángulos también pueden llamarse, en el mismo orden, ángulo de asiento lateral, de alabeo o roll; ángulo de asiento longitudinal, cabeceo o pitch; y ángulo de rumbo, guiñada o yaw.

De cara a la implementación en un UAV de ala fija, la forma de controlar los ángulos de Euler es esencialmente deflectando las superficies de control, aunque también influyen otros factores, como el empuje y la velocidad o por ejemplo la variación de la localización del centro de gravedad durante el vuelo. En el caso de este UAV el control de la actitud se realiza únicamente modificando la posición de las superficies de control, como son los alerones para el roll, los elevadores o estabilizador horizontal para el pitch, y el timón de cola para el yaw.

El ángulo de guiñada puede tener cualquier valor entre 0 y 360º, es positivo en el sentido de las agujas del reloj y tiene su origen en el norte; el ángulo de cabeceo es positivo por encima de la horizontal y negativo por debajo, y sólo se incluye en el rango entre $-\pi/2$ y $\pi/2$; y el ángulo de alabeo es positivo hacia la derecha del avión y toma valores entre $-\pi$ y π.

3.3.2 Ecuaciones de Bryan

Una vez definidos los ejes de referencia y los ángulos que describen la actitud de una aeronave se puede obtener una serie de ecuaciones basadas en las leyes básicas de la mecánica, especialmente en la Segunda Ley de Newton, aunque cabe introducir antes varias hipótesis y simplificaciones que facilitan el trabajo con estas ecuaciones.

- Se considera la aeronave como un cuerpo rígido, es decir, que la distancia que hay entre dos puntos cualquiera de la aeronave se va a mantener fija independientemente de la distribución de fuerzas que se apliquen al cuerpo. De este modo, el movimiento de la aeronave se puede describir como el movimiento del centro de gravedad de esta, tanto el movimiento de rotación como de traslación.

- Se toma el centro de masas como origen del sistema de ejes de coordenadas debido a que de ese modo las fuerzas gravitatorias no generan momentos, y además las ecuaciones de traslación y las de rotación son independientes entre sí. Por ello es interesante utilizar un sistema de ejes cuerpo, donde los ángulos de Euler se miden directamente sobre el origen.

- En las ecuaciones no se tiene en cuenta ni la rotación de la Tierra ni el efecto del viento o turbulencias presentes en la atmósfera sobre el comportamiento de la aeronave.

- Se considera que la aeronave tiene un plano de simetría en el eje XZ, por lo que los productos de inercia I_{yx} e I_{yz} son ambos nulos.

- La masa del avión se considera constante durante las maniobra.

Las ecuaciones de Bryan pueden separarse en ecuaciones de traslación, de rotación y en ecuaciones de los ángulos de Euler. No se va a desarrollar completamente la deducción de todas las ecuaciones, pero sí es conveniente comprender su origen.

La nomenclatura que se va a emplear en las ecuaciones será la siguiente (tabla 3.2):

Eje	Velocidad lineal	Velocidad angular	Momento aplicado	Fuerza aplicada	Distancia	Ángulo
X	u	p	F_x	L	x	ϕ
Y	v	q	F_y	M	y	θ
Z	w	r	F_z	N	z	ψ

Cuadro 3.2: Nomenclatura básica en las ecuaciones de Bryan

Ecuaciones de traslación

Para las ecuaciones de traslación se parte de la Segunda Ley de Newton (ecuación 3.1) considerando la masa constante, como se ha visto en las hipótesis anteriores:

$$\vec{F} = m\dot{\vec{V}}_{abs} \tag{3.1}$$

Donde la derivada de la velocidad absoluta es la mostrada en la ecuación 3.2:

$$\dot{\vec{V}}_{abs} = \dot{\vec{V}}_{rel} + \vec{\omega} \times \vec{V} = [\dot{u} \quad \dot{v} \quad \dot{w}] + [p \quad q \quad r] \times [u \quad v \quad w] \tag{3.2}$$

Si se incluye esta expresión en la Segunda Ley de Newton, se desarrollan los productos vectoriales y se añaden posteriormente las fuerzas gravitatorias obtenidas a partir de los ángulos de Euler, se obtienen las ecuaciones de traslación 3.3, 3.4 y 3.5.

$$m(\dot{u} + qw - rv) = F_{TX} + F_{AX} - mg\sin\theta \tag{3.3}$$

$$m(\dot{v} + ru - pw) = F_{TY} + F_{AY} + mg\cos\theta\sin\phi \tag{3.4}$$

$$m(\dot{w} + pv - qu) = F_{TZ} + F_{AZ} + mg\cos\theta\cos\phi \tag{3.5}$$

En estas expresiones se incluyen, por tanto, las fuerzas propulsivas (F_T), las aerodinámicas (F_A) y las debidas a la gravedad (mg), de las cuales las dos primeras se explicarán más a fondo en los próximos apartados del capítulo.

Ecuaciones de rotación

Las ecuaciones de rotación también parten de las Segunda Ley de Newton, pero en este caso aplicada a los momentos, donde la derivada temporal absoluta del momento angular ($\vec{H}_{abs}$) es igual al momento externo total aplicado sobre la aeronave (ecuación 3.6):

$$\vec{M} = \dot{\vec{H}}_{abs} \tag{3.6}$$

Teniendo en cuenta la relación de la ecuación 3.7 y desarrollando mediante el uso de los momentos y productos de inercia, se obtienen las ecuaciones 3.8, 3.9 y 3.10, que relacionan la velocidad angular de giro de la aeronave con los momentos externos aplicados sobre esta:

$$\dot{\vec{H}}_{abs} = \dot{\vec{H}}_{rel} + \vec{\omega} \times \vec{H} \tag{3.7}$$

$$L = I_{xx}\dot{p} - I_{xz}\dot{r} - I_{xz}pq + (I_{zz} - I_{yy})qr - I_{yz}(q^2 - r^2) - I_{xy}(\dot{q} - rp) \tag{3.8}$$

$$M = I_{yy}\dot{q} + I_{xz}(p^2 - r^2) + (I_{xx} - I_{zz})pr - I_{xy}(\dot{p} + qr) - I_{yz}(\dot{r} - pq) \tag{3.9}$$

$$N = I_{zz}\dot{r} - I_{xz}\dot{p} + I_{xz}rq + (I_{yy} - I_{xx})pq - I_{xy}(p^2 - q^2) - I_{yz}(\dot{q} - rp) \tag{3.10}$$

Estas expresiones se pueden simplificar teniendo en cuenta que los productos de inercia I_{xy} e I_{zy} son nulos por tener plano de simetría en la aeronave XZ. Además de esto, las ecuaciones de rotación suelen transformarse para que únicamente aparezca una derivada temporal de una velocidad angular en cada una de ellas, resultando en las ecuaciones 3.11, 3.12 y 3.13:

$$\dot{p} = \frac{I_{zz}}{A}L + \left(\frac{I_{xz}}{A}N + \frac{I_{xz}(I_{xx} - I_{yy} + I_{zz})}{A}\right)pq + \left(\frac{I_{zz}(I_{yy} - I_{zz}) - I_{xz}^2}{A}\right)rq \tag{3.11}$$

$$\dot{q} = \frac{M}{I_{yy}} + \left(\frac{I_{zz} - I_{xx}}{I_{yy}}\right)pr + \frac{I_{xz}}{I_{yy}}(r^2 - p^2) \tag{3.12}$$

$$\dot{r} = \frac{I_{xx}}{A}N + \frac{I_{xz}}{A}L + \left(\frac{I_{xx}(I_{xx} - I_{yy}) + I_{xz}^2)}{A}\right)pq + \left(\frac{I_{xz}(I_{yy} - I_{xx} - I_{zz})}{A}\right)rq \tag{3.13}$$

donde

$$A = I_{xx}\ I_{zz} - I_{xz}^2 \tag{3.14}$$

Ecuaciones de relación entre velocidades angulares y las derivadas de los ángulos de Euler

Las tres siguientes ecuaciones de Bryan indican la relación entre las velocidades angulares y las derivadas de los ángulos de Euler, como muestran las ecuaciones 3.15, 3.16 y 3.17.

$$\dot{\phi} = p + (q\sin\phi + r\cos\phi)\tan\theta \tag{3.15}$$

$$\dot{\theta} = q\cos\phi - r\sin\phi \tag{3.16}$$

$$\dot{\psi} = \frac{q\sin\phi + r\cos\phi}{\cos\theta} \tag{3.17}$$

Ecuaciones cinemáticas

Por último, las ecuaciones cinemáticas, que aportan información sobre las derivadas de las distancias recorridas a lo largo de los ejes, son las ecuaciones 3.18, 3.19 y 3.20:

$$\dot{x} = u\cos\psi\cos\theta + v(\cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi) + w(\sin\theta\cos\phi\cos\psi + \sin\phi\sin\psi) \tag{3.18}$$

$$\dot{y} = u\cos\theta\sin\psi + v(\cos\phi\cos\psi + \sin\theta\sin\phi\sin\psi) + w(-\cos\psi\sin\phi + \cos\phi + \sin\theta\sin\psi) \tag{3.19}$$

$$\dot{z} = -u\sin\theta + v\cos\theta\sin\phi + w\cos\theta\cos\phi \tag{3.20}$$

Estas últimas tres ecuaciones no se emplearán para linealizar el sistema, ya que no se controla específicamente la posición de la aeronave, sino el modo de llegar a ciertos puntos, pero sí son de gran utilidad para evaluar el funcionamiento de los controles comprobando que se siguen los waypoints definidos.

3.3.3 Fuerzas y momentos aerodinámicos

Las fuerzas y los momentos aerodinámicos que actúan sobre una aeronave pueden simplificarse en un modelo aerodinámico lineal debido a la hipótesis mencionada anteriormente de pequeñas perturbaciones, esto es, pueden representarse mediante la suma del producto de ciertos coeficientes o derivadas aerodinámicas por unas variables de derivación, más en algunos casos términos independientes que parten del diseño de la aeronave (las derivadas aerodinámicas se obtienen derivando los coeficientes aerodinámicos respecto a la variable de derivación correspondiente anulando el resto de variables).

Existen varias formas de calcular las fuerzas aerodinámicas que actúan sobre la aeronave incluidas en las ecuaciones de Bryan, una de ellas mediante las expresiones siguientes, que hacen referencia a las fuerzas que actúan sobre cada uno de los ejes:

$$F_{AX} = \frac{1}{2}\rho S_w V^2 C_X \tag{3.21}$$

$$F_{AY} = \frac{1}{2}\rho S_w V^2 C_Y \tag{3.22}$$

$$F_{AZ} = \frac{1}{2}\rho S_w V^2 C_Z \tag{3.23}$$

Los coeficientes aerodinámicos C_X, C_Y y C_Z se obtienen empleando las expresiones 3.24, 3.25 y 3.26 donde los coeficientes de sustentación y de resistencia C_L y C_D vienen dados por las ecuaciones

3.27 y 3.28, y teniendo en cuenta que C_Y se emplea tanto para las fuerzas aerodinámicas del eje Y como para los ejes X y Z (al final de este subapartado se expone una lista de los coeficientes que representan las derivadas aerodinámicas con respecto a las variables de estado y respecto a las variables de control). En estas expresiones, el segundo subíndice de los coeficientes representa la variable respecto a la que se ha derivado.

$$C_X = -\frac{\cos\alpha}{\cos\beta}C_D - \cos\alpha\tan\beta\ C_Y + \sin\alpha\ C_L \tag{3.24}$$

$$C_Y = C_{Y_\beta}\beta + C_{Y_{\dot\beta}}\dot\beta + C_{Y_p}\frac{b_w}{2V}p + C_{Y_r}\frac{b_w}{2V}r + C_{Y_{\delta A}}\delta A + C_{Y_{\delta R}}\delta R \tag{3.25}$$

$$C_Z = -\frac{\sin\alpha}{\cos\beta}C_D - \sin\alpha\tan\beta\ C_Y - \cos\alpha\ C_L \tag{3.26}$$

$$C_L = C_{L_0} + C_{L_\alpha}\alpha + C_{L_{\delta E}}\delta E + C_{L_{\dot\alpha}}\frac{c_w}{2V}\dot\alpha + C_{L_q}\frac{c_w}{2V}q \tag{3.27}$$

$$C_D = C_{D_0} + C_{D1}\ \alpha + C_{D2}\ \alpha^2 \tag{3.28}$$

Donde δA, δE y δR son los ángulos de deflexión de alerones, elevadores y timón de cola, S_w, b_w y c_w son respectivamente la superficie, envergadura y cuerda del ala, y existen las siguientes relaciones:

$$V = \sqrt{u^2 + v^2 + w^2} \tag{3.29}$$

$$\alpha = \arctan\frac{w}{u} \tag{3.30}$$

$$\dot\alpha = \frac{\dot w}{u} - \frac{\dot u w}{w^2 + u^2} \tag{3.31}$$

$$\beta\ \arcsin\frac{v}{V} \tag{3.32}$$

$$\dot\beta = \frac{\dot v V - v(\dot u u + \dot v v + \dot w w)}{V(\sqrt{V} - v^2)} \tag{3.33}$$

Por otro lado, los momentos aerodinámicos en cada eje tienen las expresiones 3.34, 3.35 y 3.36:

$$M_{AX} = \frac{1}{2}\rho S_w V^2 b_w C_l \tag{3.34}$$

$$M_{AY} = \frac{1}{2}\rho S_w V^2 b_w C_M \tag{3.35}$$

$$M_{AZ} = \frac{1}{2}\rho S_w V^2 b_w C_N \tag{3.36}$$

En las expresiones anteriores los coeficientes C_l, C_M y C_N se definen mediante las ecuaciones 3.37, 3.38 y 3.39:

$$C_l = C_{l_\beta}\beta + C_{l_{\dot\beta}}\dot\beta + C_{l_p}\frac{b_w}{2V}p + C_{l_r}\frac{b_w}{2V}r + C_{l_{\delta A}}\delta A + C_{l_{\delta R}}\delta R \tag{3.37}$$

$$C_M = C_{M_0} + C_{M_\alpha}\alpha + C_{M_{\delta E}}\delta E + C_{M_{\dot\alpha}}\frac{c_w}{2V}\dot\alpha + C_{M_q}\frac{c_w}{2V}q \tag{3.38}$$

$$C_N = C_{N_\beta}\beta + C_{N_{\dot\beta}}\dot\beta + C_{N_p}\frac{b_w}{2V}p + C_{N_r}\frac{b_w}{2V}r + C_{N_{\delta A}}\delta A + C_{N_{\delta R}}\delta R \tag{3.39}$$

Los valores de las derivadas aerodinámicas empleadas para definir las fuerzas y los momentos aerodinámicos del sistema con estas expresiones se han obtenido mediante el programa *Xflr5* [2], que es una herramienta de análisis de perfiles alares, alas y aeronaves a números de Reynolds bajos, basada en la Teoría de Líneas Sustentadoras de Prandtl, el método de *Vortex Lattice* y el método de los paneles en 3D. Los coeficientes aerodinámicos obtenidos son los mostrados en las tablas 3.3 y 3.4. Los valores de los coeficientes restantes, no introducidos en las tablas por ser términos independientes o relacionados únicamente con una variable, son $C_{L_0} = 0.3310$, $C_{M_0} = 0.2662$, $C_{D_0} = 0.039$, $C_{D1} = 0.3310$ y $C_{D2} = 1.4201$.

Variable	α	β	$\dot\alpha$	$\dot\beta$	V	p	q	r
C_L	4.8406	0	2.2396	0	0	0	10.1570	0
C_X	–	0	–	0	0	0	–	0
C_Y	0	−0.1437	0	0	0	0.0398	0	0.17380
C_Z	–	0	–	0	0	0	–	0
C_l	0	−0.0207	0	0	0	−0.5269	0	0.2224
C_M	−1.7800	0	−9.4711	0	0	0	−24.8790	0
C_N	0	0.0756	0	0	0	−0.1466	0	−0.0894

Cuadro 3.3: Derivadas aerodinámicas de las variables de estado

Variable	δA	δE	δP	δR
C_X	0	−0.0217	−0.0902	0
C_Y	−0.0155	0	0	0.1201
C_Z	0	−0.5551	−1.6182	0
C_l	0.4548	0	0	−0.0024
C_M	0	−2.2135	0.7259	0
C_N	0.0082	0	0	−0.0673

Cuadro 3.4: Derivadas aerodinámicas de las variables de control

3.3.4 Modelo propulsivo

Como se ha comentado anteriormente, el motor empleado para el modelo propulsivo es el AT4130 *Long Shaft* KV300 de *T-motors*. A raíz de datos de funcionamiento en un banco de ensayos proporcionados por el fabricante [16], mostrados en la tabla 3.5, es posible ajustar la relación entre la palanca aplicada y el empuje obtenido con una hélice APC17x10 a una curva de segundo orden, tal como se muestra en la figura 3.7.

Por tanto, a la hora de introducir el modelo propulsivo en el modelado completo de la aeronave, este tiene la siguiente forma, donde el empuje depende únicamente de la variable de control de la palanca y del cuadrado de esta y sólo se aplica en la dirección del movimiento, es decir, en el eje X (ecuación 3.40):

$$F_{TX} = 8.859\ \delta P + 58.362\ \delta P^2 \tag{3.40}$$

Debido a que el motor está ubicado ligeramente por encima del centro de gravedad (a una diferencia de altura de $h_{M_T} = 0.048m$), existe un momento de cabeceo asociado al empuje (ecuación 3.41).

Palanca	Empuje (g)	Empuje (N)
0	0	0
0.05	60	0.59
0.10	102	1.00
0.15	162	1.59
0.20	334	3.28
0.25	579	5.68
0.30	748	7.34
0.35	978	9.59
0.40	1175	11.53
0.45	1458	14.30
0.50	1903	18.67
0.55	2355	23.10
0.60	2773	27.20
0.65	3187	31.26
0.70	3696	36.26
0.75	4198	41.18
0.80	4662	45.73
0.90	5652	55.45
1	6577	64.52

Cuadro 3.5: Empuje proporcionado por el motor AT4130 con una hélice APC17x10 en función de la palanca

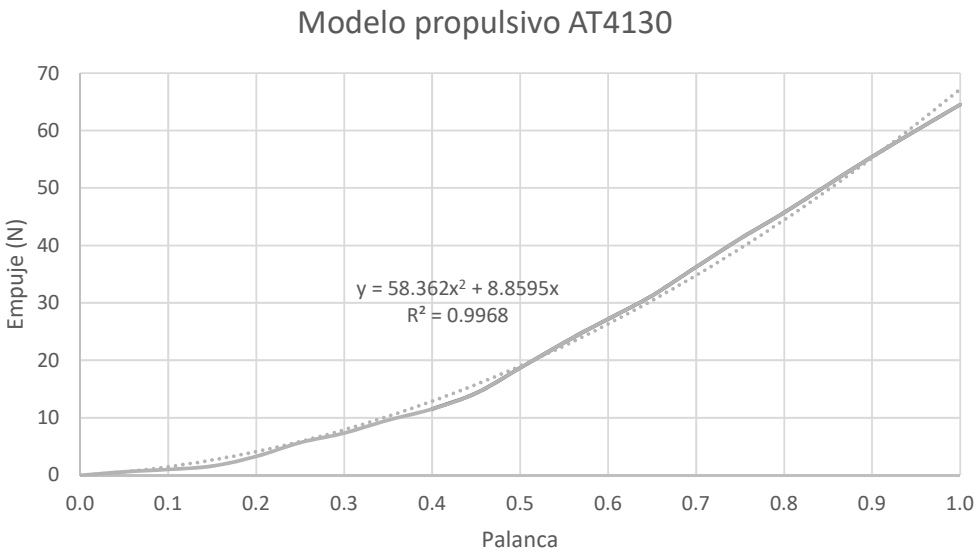

Figura 3.7: Modelo propulsivo del motor AT4130 con una hélice APC17x10

Por tanto, en este caso son nulas las fuerzas y momentos asociados al empuje F_{TY}, F_{TZ}, M_{TX} y M_{TZ}.

$$M_{TY} = -0.048\ F_{TX} \tag{3.41}$$

3.3.5 Simulink

Para poder posteriormente comparar el modelo no lineal con el linealizado y validar el sistema de control teórico se ha optado por introducir el modelo de la aeronave en el programa *Simulink* de *Matlab*, que emplea un sistema de bloques como implementación del modelo. El esquema de *Simulink* puede descomponerse en varios sub-bloques: modelo aerodinámico, modelo propulsivo, fuerzas gravitatorias, modelo de atmósfera, ecuaciones dinámicas y ecuaciones cinemáticas (o bloque de conversión a coordenadas NED).

El bloque del modelo aerodinámico se puede dividir en tres partes. La primera de ellas es la obtención de los coeficientes aerodinámicos, donde se abordan las ecuaciones de los coeficientes C_L, C_D y C_M como parte de la dinámica longitudinal, es decir para las fuerzas y momentos que actúan en el plano XZ; y C_Y, C_l y C_N para la dinámica lateral direccional (plano XY). Como se ha visto en las ecuaciones 3.24 y 3.26, los coeficientes C_L, C_D y C_Y, además de α y β, se emplean para obtener C_X y C_Z. Esta implementación de ecuaciones se puede ver en la figura 3.8, donde los bloques de funciones de *Matlab* incluyen las ecuaciones expuestas en el apartado anterior.

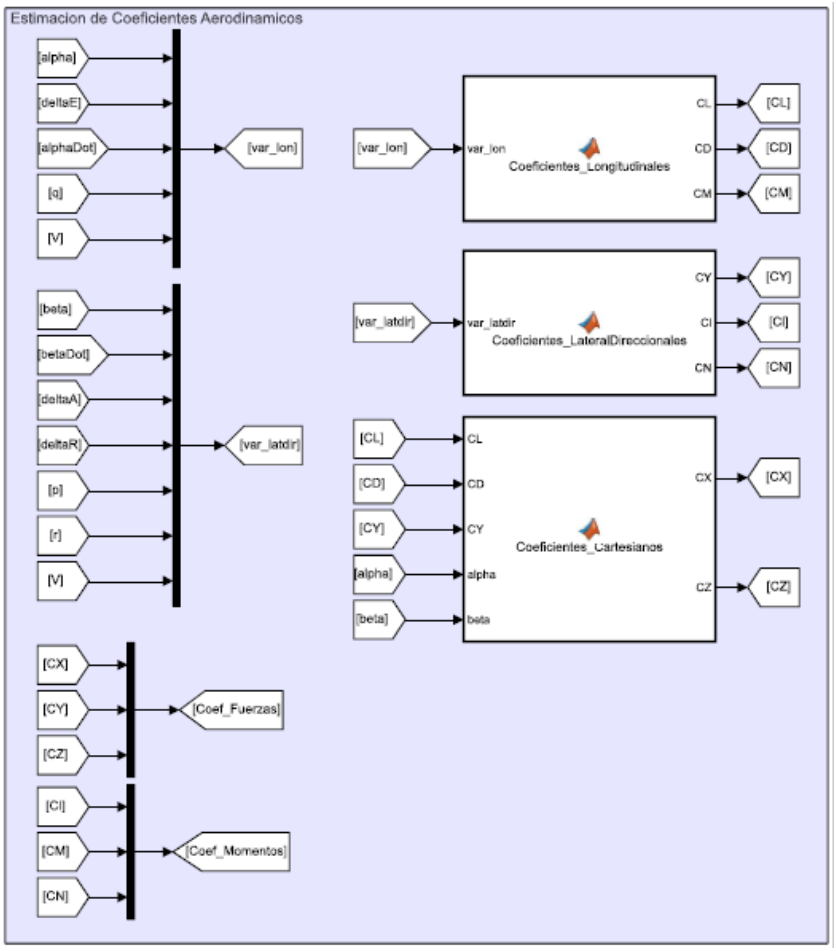

Figura 3.8: Bloque en *Simulink* de estimación de los coeficientes aerodinámicos

La segunda y la tercera parte del modelo aerodinámico emplean los coeficientes aerodinámicos obtenidos para calcular las fuerzas y los momentos aerodinámicos mediante sus ecuaciones correspondientes, como muestra la figura 3.9.

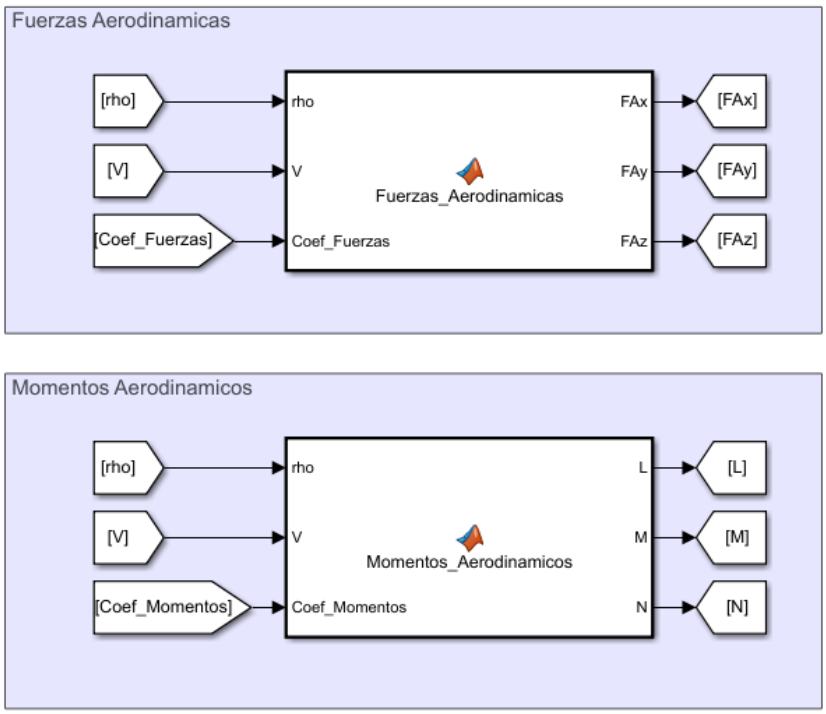

Figura 3.9: Bloques en *Simulink* de de obtención de fuerzas y momentos aerodinámicos

Por su parte, el bloque del modelo propulsivo tiene la estructura de la figura 3.10, mientras el bloque de fuerzas gravitatorias tiene como entradas ϕ y θ y calcula los elementos que incluyen el efecto de la gravedad en las ecuaciones 3.3, 3.4 y 3.5, y el bloque del modelo de la atmósfera emplea el modelo de atmósfera ISA para obtener en concreto la densidad del aire ρ, empleada en las ecuaciones de las fuerzas y momentos aerodinámicos.

La figura 3.11 muestra la estructura en *Simulink* de las ecuaciones dinámicas, donde las velocidades vel_lin y vel_ang vienen dadas por el bloque de la figura 3.12 y los bloques de funciones de *Matlab* calculan las aceleraciones lineales y aceleraciones angulares mediante las ecuaciones de traslación y de rotación.

Por último, las ecuaciones cinemáticas y las de relación entre las velocidades angulares y las derivadas de los ángulos de Euler se emplean en los bloques de conversión a coordenadas NED y de paso de velocidades angulares a ángulos de Euler mediante cuaterniones.

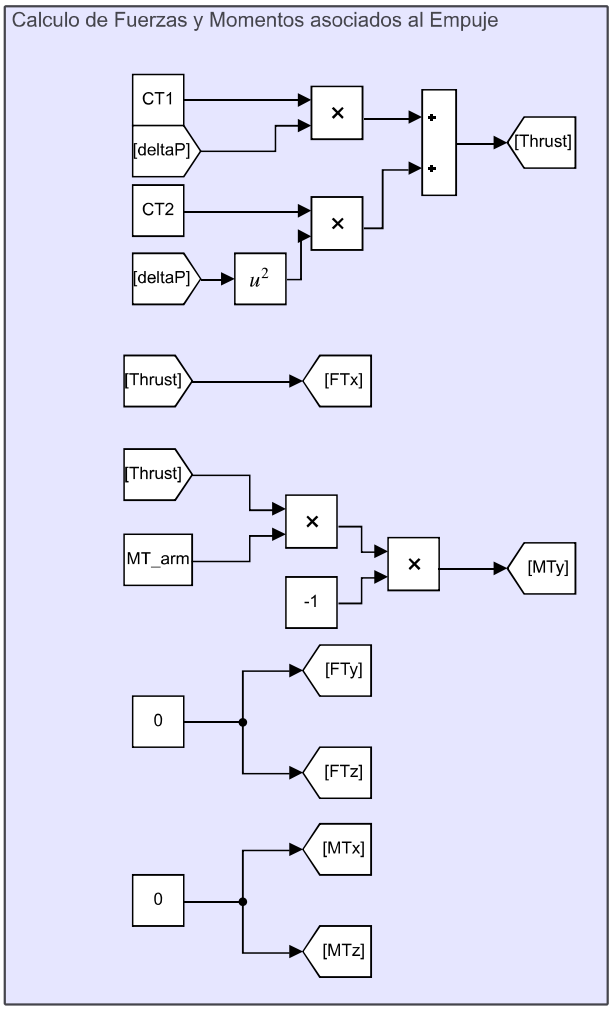

Figura 3.10: Bloques en *Simulink* de de obtención de fuerzas y momentos aerodinámicos

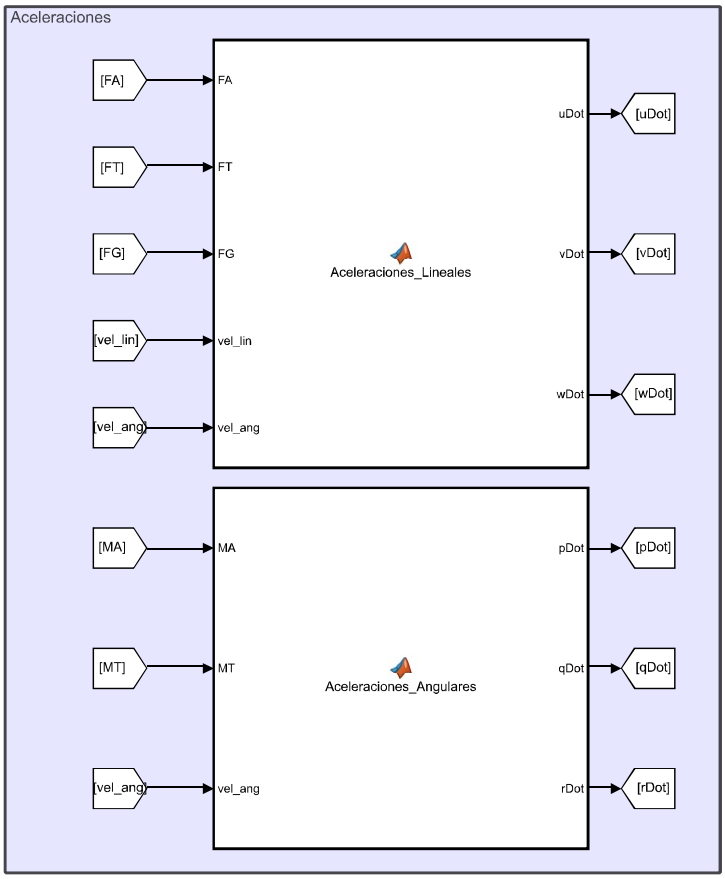

Figura 3.11: Bloques en *Simulink* de de obtención de fuerzas y momentos aerodinámicos

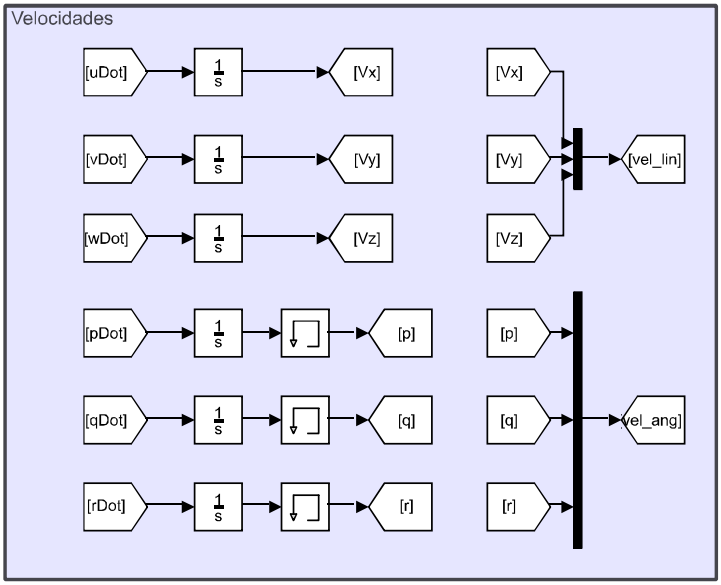

Figura 3.12: Bloques en Simulink de de obtención de fuerzas y momentos aerodinámicos

Capítulo 4

Diseño del sistema de control

4.1 Linealización del sistema

El diseño del control se realizará en espacio de estados, por lo que para este tipo de representación es necesario obtener un sistema de ecuaciones diferenciales de primer orden que represente este espacio de estados a partir del modelo no lineal, además de unos puntos de funcionamiento que sirvan como condiciones iniciales para llevar acabo esta linealización.

El motivo de la linealización del modelo es la dificultad de elaborar leyes de control para sistemas no lineales. Al linealizar el sistema se asume que este va a ser controlado en un punto de operación fijo o punto de funcionamiento y que cualquier cambio que ocurra en el sistema respecto a este punto de funcionamiento será pequeño, resultando en que cada variable de estado x_i puede representarse como $x_{i_0} + \delta x_i$, al igual que las entradas u_i equivalen a $u_{i_0} + \delta u_i$, donde el subíndice 0 indica punto de funcionamiento, el subíndice i la ecuación del sistema y δ (delta) indica una variación pequeña desde el punto de equilibrio.

Teniendo en cuenta que para el diseño del controlador se busca una representación en espacio de estados con la forma $\delta\dot{\underline{x}} = A\ \delta\underline{x} + B\ \delta\underline{u}$, esto se puede obtener mediante el desarrollo de Taylor de las derivadas de las variables de estado, como se muestra en la ecuación 4.1.

$$\frac{d}{dt}(x_0 + \delta x)_i = (\dot{x_0} + \delta\dot{x_0})_i = f_i(x_0 + \delta x_i, u_0 + \delta u) = f_i(x_0, u_0) + \\ + \delta x_1 \frac{\partial f_i}{\partial x_1}\bigg|_{x_0,u_0} + \delta x_2 \frac{\partial f_i}{\partial x_2}\bigg|_{x_0,u_0} + ... + \delta u_1 \frac{\partial f_i}{\partial u_1}\bigg|_{x_0,u_0} + ... \quad (4.1)$$

Por definición, la derivada temporal de las variables en el punto de funcionamiento es nula ($\dot{x_{i_0}} = 0 = f_i(x_{i_0}, u_{i_0})$), por lo que la linealización del sistema se puede obtener en forma matricial tal como indica la ecuación 4.2

$$\delta\underline{\dot{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \cdots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix} \delta\underline{x} + \begin{pmatrix} \frac{\partial f_1}{\partial u_1} & \frac{\partial f_1}{\partial u_2} & \cdots & \frac{\partial f_1}{\partial u_n} \\ \frac{\partial f_2}{\partial u_1} & \frac{\partial f_2}{\partial u_2} & \cdots & \frac{\partial f_2}{\partial u_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1} & \frac{\partial f_n}{\partial u_2} & \cdots & \frac{\partial f_n}{\partial u_n} \end{pmatrix} \delta\underline{u} \tag{4.2}$$

$$\delta\underline{\dot{x}} = A\ (\underline{x_0}, \underline{u_0})\delta\underline{x} + B\ (\underline{x_0}, \underline{u_0})\delta\underline{u}$$

Las variables de estado serán $\delta\underline{x} = (u\ v\ w\ p\ q\ r\ \phi\ \theta\ \psi)^T$, y las entradas $\delta\underline{u} = (\delta A\ \delta E\ \delta P\ \delta R)^T$, representando en ese orden la deflexión de los alerones, deflexión de los elevadores, nivel de palanca y deflexión del timón de cola.

Puntos de funcionamiento

Los puntos de funcionamiento del sistema representan el valor de las variables de estado y de las entradas cuando el vuelo es equilibrado, es decir, donde las derivadas temporales de estas variables y entradas son nulas. Durante el vuelo equilibrado se consigue una compensación de fuerzas y de momentos en la aeronave: la sustentación se compensa con el peso de la aeronave, el empuje se compensa con la resistencia aerodinámica, el momento de cabeceo debido a la posición del motor se iguala con el momento de cabeceo de la aeronave, y las fuerzas laterales en el eje Y y los momentos alrededor de los ejes X y Z son nulos. Por tanto, en el punto de funcionamiento los controles se mantienen fijos y el vuelo se mantiene recto y horizontal. Estas condiciones de equilibrio se observan en el conjunto de ecuaciones 4.3 , donde se han empleado los ángulos α y β por simplicidad, aunque los puntos de funcionamiento correspondientes a las variables de estado se obtendrán a partir de ellos.

$$\begin{aligned} mg\cos\theta + \frac{1}{2}\rho V_0^2 S_w C_Z &= 0 \\ F_{TX} + \frac{1}{2}\rho V_0^2 S_w C_X - mg\sin\theta &= 0 \\ \frac{C_M - M_{T,arm}F_{TX}}{\frac{1}{2}\rho V_0^2 S_w c_w} &= 0 \\ C_Y &= 0 \\ C_l &= 0 \\ C_N &= 0 \end{aligned} \tag{4.3}$$

Tras resolver δA, δE, δP, δR, α y β en este sistema de ecuaciones y teniendo en cuenta que la velocidad angular, la velocidad lineal en el eje Y, y los ángulos de alabeo y guiñada son nulos en el punto de equilibrio, los puntos de funcionamiento restantes que se obtienen son los mostrados en las ecuaciones 4.4 y 4.5. Para obtener los valores de las velocidades lineales iniciales u_0 y w_0 se han empleado las relaciones entre los ángulos α y β y las velocidad lineales expuestos en el apartado 3.3.3.

$$\underline{x_0} = \begin{pmatrix} 24.99 & 0 & -0.05 & 0 & 0 & 0 & 0 & -0.002 & 0 \end{pmatrix}^T \tag{4.4}$$

$$\underline{u_0} = \begin{pmatrix} 0 & 0.1185 & 0.2771 & 0 \end{pmatrix}^T \tag{4.5}$$

Linealización

El proceso seguido para obtener la representación en espacio de estados del sistema consiste en obtener en primer lugar las matrices correspondientes al sistema unilateral, es decir, el obtenido pasando todos los elementos de las 9 primeras ecuaciones de Bryan al mismo lado de la igualdad y empleando la expresión 4.2 para obtener una ecuación matricial como la ecuación 4.6 (donde MI, MD y MC son las matrices obtenidas derivando las ecuaciones respectivamente por las derivadas de las variables de estado, por las variables de estado en sí, y por las variables de control) y en segundo lugar transformar estas matrices a las definitivas A y B mediante las relaciones de la expresión 4.7. De este modo se obtiene el siguiente modelo lineal en espacio de estados y en forma matricial (expresión 4.8).

$$MI\ \delta\underline{\dot{x}} - MD\ \delta\underline{x} - MC\ \delta\underline{u} = 0 \tag{4.6}$$

$$\begin{aligned} A &= -MI^{-1}\ MD \\ B &= -MI^{-1}\ MC \end{aligned} \tag{4.7}$$

$$\delta\underline{\dot{x}} = \begin{pmatrix} -0.079 & 0 & 0.296 & 0 & 0.046 & 0 & 0 & -9.809 & 0 \\ 0 & -0.145 & 0 & -0.009 & 0 & -24.823 & 9.809 & 0 & 0 \\ -0.785 & 0 & -4.890 & 0 & 23.446 & 0 & 0 & 0.019 & 0 \\ 0 & -0.414 & 0 & -13.373 & 0 & 5.439 & 0 & 0 & 0 \\ 0.070 & 0 & -2.257 & 0 & -6.152 & 0 & 0 & 0 & 0 \\ 0 & 0.641 & 0 & -2.014 & 0 & -0.489 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -0.002 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix} \delta\underline{x} + \\ + \begin{pmatrix} 0 & -0.027 & 5.628 & 0 \\ -0.392 & 0 & 0 & 3.044 \\ 0 & -13.912 & 0 & 0 \\ 284.571 & 0 & 0 & -3.785 \\ 0 & -79.720 & -1.554 & 0 \\ 17.258 & 0 & 0 & -14.986 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \delta\underline{u} \tag{4.8}$$

Teniendo en cuenta que se busca que las salidas del sistema sean, además de las variables de estado, los valores de x, y, z y el rumbo de la aeronave ζ en este orden (definido el rumbo como $\zeta = \arctan\frac{\dot{y}}{\dot{x}}$), la matriz C es la mostrada en la expresión 4.9, y la matriz D que acompañaría a las entradas del sistema es una matriz de ceros de dimensiones 13x4.

$$\delta\underline{v} = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 1 & 0 & -0.002 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0.05 & 0 & 25 \\ 0.002 & 0 & 1 & 0 & 0 & 0 & 0 & -25 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \delta\underline{x} \tag{4.9}$$

4.2 Funciones de transferencia

El objetivo de linealizar el sistema es obtener de una forma clara la relación que existe entre las diferentes variables y entradas, y la mejor forma de analizar estas relaciones es mediante funciones de transferencia.

A partir de la representación en espacio de estados del sistema linealizado se puede emplear la función *ss2tf* de *Matlab* para obtener de forma automática las funciones de transferencia que relacionan las salidas con los controles de la aeronave o con las diferentes variables de estado, teniendo en cuenta que emplean los valores incrementales y no absolutos de las variables. Para el diseño del control automático son de interés las cuatro funciones de transferencia que contienen las variables de control ($\phi/\delta A$, $\theta/\delta E$, $u/\delta P$ y $\dot{v}/\delta R$), y las dos que permiten obtener la altitud y el rumbo necesarios para el guiado (z/θ y ζ/ϕ).

Función de transferencia $\phi/\delta A$

La función de transferencia entre el roll y la deflexión de los alerones es la mostrada en la ecuación 4.10. En esta función el valor de uno de los polos (raíces del denominador) reales es positivo e implica una dinámica inestable, pero dado que su valor de 0.1307 se encuentra muy próximo a 0 en comparación con el polo real negativo -12.7235, esta inestabilidad tendrá una dinámica lenta corregible por acciones de control suficientemente rápidas (tabla 4.1). Además, el comportamiento del ángulo de alabeo respecto al de los alerones será transitorio y con oscilaciones debidas a la existencia de polos con valores complejos.

$$\frac{\phi(s)}{\delta A(s)} = \frac{284.5s^2 + 275.6s + 4743}{s^4 + 14.01s^3 + 35.45s^2 + 240.4s - 32.06} \tag{4.10}$$

	Parte real	Parte imaginaria
Polos	0.1307	0
	-0.7082	-4.3333
	-0.7082	4.3333
	-12.7235	0
Ceros	-0.4843	-4.0540
	-0.4843	-4.0540
Ganancia	-147.9413	

Cuadro 4.1: Polos, ceros y ganancia de la función de transferencia de $\phi/\delta A$

Función de transferencia $\theta/\delta E$

La función de transferencia entre el pitch y el ángulo de deflexión de los elevadores es la representada en la ecuación 4.11, donde todos los polos son números complejos con partes reales negativas, por lo que el sistema es oscilatorio pero estable (tabla 4.2).

$$\frac{\theta(s)}{\delta E(s)} = \frac{-79.72s^2 - 364.8s - 47.5}{s^4 + 11.12s^3 + 84.14s^2 + 8.197s + 20.8} \tag{4.11}$$

	Parte real	Parte imaginaria
Polos	-0.0327	-0.4991
	-0.0327	0.4991
	-5.5291	-7.2521
	-5.5291	7.2521
Ceros	-0.1341	0
	-4.4424	0
Ganancia	-2.2836	

Cuadro 4.2: Polos, ceros y ganancia de la función de transferencia de $\theta/\delta E$

Función de transferencia $u/\delta P$

La ecuación 4.12 expresa la función de transferencia entre la velocidad u y la palanca de gases, que es estable debido a que todos los polos tienen su parte real negativa, aunque esta expresión se ha simplificado teniendo en cuenta que se puede anular una pareja de polos con otra de ceros por tener valores muy parecidos, como se observa en los dos últimos polos y dos últimos ceros de la tabla 4.3, obteniendo la función de transferencia de la ecuación 4.13.

$$\frac{u(s)}{\delta P(s)} = \frac{5.628s^3 + 62.08s^2 + 471.4s + 74.83}{s^4 + 11.12s^3 + 84.14s^2 + 8.197s + 20.8} \tag{4.12}$$

$$\frac{u(s)}{\delta P(s)} = \frac{5.628s + 0.9127}{s^2 + 0.06531s + 0.2501} \tag{4.13}$$

	Parte real	Parte imaginaria
Polos	-0.0327	-0.4991
	-0.0327	0.4991
	-5.5291	-7.2521
	-5.5291	7.2521
Ceros	-0.1622	0
	-5.4343	-7.2427
	-5.4343	7.2427
Ganancia	3.5975	

Cuadro 4.3: Polos, ceros y ganancia de la función de transferencia de $u/\delta P$

Función de transferencia $\dot{v}/\delta R$

La función que introduce el accionamiento del timón de cola en la aeronave es la de compensación de las aceleraciones laterales, representadas por $\dot{v}$, y la función de transferencia correspondiente es la mostrada en la ecuación 4.14. Esta función también puede simplificarse anulando un polo con un cero que están muy próximos entre sí (los situados alrededor de -12, como muestra la tabla 4.4) y empleando la función *minreal* de *Matlab*, obteniéndose la función de transferencia de la ecuación 4.15.

$$\frac{\dot{v}(s)}{\delta R(s)} = \frac{s(3.044s^3 + 414.3s^2 + 4803s - 814.1)}{s^4 + 14.01s^3 + 35.45s^2 + 240.4s - 32.06} \tag{4.14}$$

	Parte real	Parte imaginaria
Polos	0.1307	0
	-0.7082	-4.3333
	-0.7082	4.3333
	-12.7235	0
Ceros	0.1671	0
	0	0
	-12.9852	0
	-123.2638	0
Ganancia	0	

Cuadro 4.4: Polos, ceros y ganancia de la función de transferencia de $\dot{v}/\delta R$

$$\frac{\dot{v}(s)}{\delta R(s)} = \frac{3.044s^2 + 374.7s - 62.7}{s^2 + 1.416s + 19.28} \tag{4.15}$$

Función de transferencia z/θ

La estrategia empleada en la navegación de esta aeronave para alcanzar una altitud determinada es el uso de los elevadores para modificar el ángulo de cabeceo, por lo que interesa obtener la función de transferencia entre esta altitud y el ángulo de cabeceo, mostrada en la ecuación 4.16. Uno de los ceros de esta expresión es de fase no mínima (o con su parte real positiva), y otro cero está lejos del origen (polos 26.7879 y -24.0543 mostrados en la tabla 4.5). Se ha comprobado [3] que eliminando estos ceros de la función de transferencia y manteniendo la ganancia, la nueva expresión (ecuación 4.17) proporciona una respuesta similar a la de la expresión original.

$$\frac{z(s)}{\theta(s)} = \frac{0.1745s^3 - 0.4666s^2 - 112.5s - 7.153}{s(s^2 + 4.577s + 0.5958)} \tag{4.16}$$

	Parte real	Parte imaginaria
Polos	0	0
	-0.1341	0
	-4.4424	0
Ceros	26.7879	0
	-0.0636	0
	-24.0543	0
Ganancia	-12.0077	

Cuadro 4.5: Polos, ceros y ganancia de la función de transferencia de z/θ

$$\frac{z(s)}{\theta(s)} = \frac{-112.5s - 7.153}{s(s^2 + 4.577s + 0.5958)} \tag{4.17}$$

Función de transferencia ζ/ϕ

Por último, dado que la trayectoria durante la navegación se consigue modificando el rumbo, la función de transferencia que interesa obtener es la mostrada en la ecuación 4.18, donde se encuentran dos ceros de fase no mínima que no se han modificado debido a que no afectan negativamente al diseño del control. Los polos y ceros del sistema se han introducido en la tabla 4.6.

$$\frac{\zeta(s)}{\phi(s)} = \frac{0.061s^3 - 1.195s^2 - 0.195s + 6.542}{s(s^2 + 0.969s + 16.67)} \tag{4.18}$$

	Parte real	Parte imaginaria
Polos	0	0
	-0.4843	-4.0540
	-0.4843	4.0540
Ceros	19.5924	0
	2.4055	0
	-2.2885	0
Ganancia	0.3924	

Cuadro 4.6: Polos, ceros y ganancia de la función de transferencia de ζ/ϕ

4.3 Diseño del control

El siguiente paso tras la obtención de las funciones de transferencia de la sección anterior, es aplicarlas para el diseño de los controladores del piloto automático, por un lado para el guiado de la aeronave a través de las leyes de navegación y por otro para el accionamiento de las superficies de control. Estas dos divisiones de los controladores se corresponden con los denominados lazos de control externo e interno respectivamente, donde el externo genera las referencias de las variables de estado que se quieren controlar y el interno emplea estas referencias para accionar las superficies de control correspondientes. Dado que en este trabajo se están diseñando al mismo tiempo el sistema

de control y el de navegación, las referencias de las variables vendrán dadas por el lazo de control externo, pero en el modo manual de vuelo las referencias serán proporcionadas por radio control.

Un aspecto importante que hay que tener en cuenta en este desarrollo es que las funciones de transferencia trabajan con diferencias y no con valores absolutos de las variables, por lo que a las salidas de cada bloque de control hay que sumarles los valores de los puntos de equilibrio correspondientes.

Dado que el piloto automático empleado durante este proyecto es *ArduPilot* y los controladores que usa son de tipo PID (se verán con más profundidad en la sección 5.1), también serán de este tipo los diseñados para el modelo dinámico de la aeronave. Además, como se ha explicado anteriormente, los controladores PID son sencillos de utilizar y se puede encontrar fácilmente documentación sobre su funcionamiento y diseño debido a su uso extendido en los ámbitos académico y profesional.

Para obtener las ganancias de los múltiples controladores se ha empleado la herramienta *pidTuner* de *Matlab*, que a partir de requisitos de robustez y rapidez de las respuestas calcula valores apropiados de estos parámetros, teniendo en cuenta en todo momento que las funciones de transferencia se han obtenido a partir del modelo linealizado y no del modelo no-lineal que representa la realidad. No procede en este proyecto la explicación del funcionamiento de esta herramienta ni el desarrollo completo de cómo se ha obtenido cada parámetro, pues este es solo uno de los diversos métodos de diseño que se pueden emplear hoy en día. A continuación se muestran los parámetros obtenidos para los lazos de control interno y externo por separado.

4.3.1 Lazo de control interno

Como se ha explicado en la introducción a los controladores de tipo PID de la subsección 2.3.1, estos se basan en la diferencia existente entre el valor de referencia de una variable y su valor actual.

En primer lugar, los controladores del ángulo de alabeo y del ángulo de cabeceo tienen una estructura similar, pues ambos cuentan con partes proporcional, integrativa y derivativa. La entrada a los controladores es el error entre el ángulo de referencia indicado por el sistema de navegación o lazo externo y el valor instantáneo del ángulo de roll y de pitch, respectivamente para alerones y para elevadores. Los valores obtenidos con *pidTuner* que ayudan a que este error sea lo más pequeño posible son los mostrados en la tabla 4.7, obtenidos a partir de las funciones de transferencia de $\phi/\delta A$ y $\theta/\delta E$ y correspondientes a una estructura en paralelo de controlador PID como la mostrada en la ecuación 4.19, donde N es el coeficiente del filtro del derivador, que toma el valor de la unidad en este caso.

	K_p	K_i	K_d
Alabeo	3.193	10.330	0.247
Cabeceo	-12.823	-21.490	-1.913

Cuadro 4.7: Coeficientes de los controladores de alabeo y de cabeceo diseñados

$$PID(s) = K_p + K_i \, \frac{1}{s} + K_d \, \frac{N}{1 + N\frac{1}{s}} \tag{4.19}$$

Por otro lado, el controlador de la aceleración lateral, correspondiente a la deflexión del timón de cola, es un controlador de tipo PI que tiene como objetivo que esta aceleración sea nula. Empleando la función de transferencia de $\dot{v}/\delta R$ se obtienen las ganancias proporcional e integral de la tabla 4.8.

	K_p	K_i
Acel. lateral	0.149	43.638

Cuadro 4.8: Coeficientes del controlador de la aceleración lateral diseñado

Como último controlador del lazo interno se encuentra el de la velocidad, controlada por la palanca de gases, con la referencia indicada por los waypoints durante la navegación y basada en la función de transferencia $u/\delta P$. Este controlador también es un PI y los valores de sus ganancias se muestran en la tabla 4.9.

	K_p	K_i
Velocidad	1.370	0.186

Cuadro 4.9: Coeficientes del controlador de velocidad diseñado

4.3.2 Lazo de control externo

Uno de los controladores del lazo externo es el que proporciona el ángulo de cabeceo necesario para alcanzar una altitud de referencia indicada por las leyes de navegación. A partir de la función de transferencia z/θ se obtiene un controlador PI con los parámetros de la tabla 4.10.

	K_p	K_i
Altitud	-0.065	-1.755e-3

Cuadro 4.10: Coeficientes del controlador de altitud diseñado

El otro controlador del lazo externo es el del rumbo, responsable de generar la trayectoria que debe seguir la aeronave para alcanzar una serie de waypoints. Más adelante se explicará la ley de navegación que se sigue en este trabajo (navegación L_1), pero esencialmente busca que la diferencia entre el rumbo objetivo, que une la aeronave con un punto de referencia, y el rumbo instantáneo obtenido a partir de la posición y la velocidad de la aeronave sea mínima. El controlador escogido en este caso es de tipo PD, con las ganancias proporcional y derivativa (mostradas en la tabla 4.11) obtenidas a partir de la función de transferencia ζ/ϕ, teniendo presente que este controlador cuenta con un filtro de paso alto para que los cambios en la referencia del ángulo de alabeo no sean demasiado bruscos.

	K_p	K_d
Rumbo	0.361	2.500

Cuadro 4.11: Coeficientes del controlador de rumbo diseñado

Todos los controladores diseñados se emplearán, como se explica en el siguiente capítulo, para trasladar las acciones deseadas de la aeronave al piloto automática real *ArduPilot*.

Capítulo 5

Implementación inicial del control

En el capítulo anterior se ha diseñado un sistema de control teórico que permite al modelo del UAV responder correctamente tanto a instrucciones directas del piloto como a una serie de waypoints introducidos. A continuación se procederá a trasladar estos resultados a un modelo basado en los controladores que contiene el piloto automático *ArduPilot* y a obtener los parámetros que los definen.

5.1 *ArduPilot*

ArduPilot es un proyecto de código libre, es decir, es un código accesible para todo aquel que quiera usarlo y que puede ser modificado por el usuario, el cual pasa a formar parte del llamado equipo DEV (*Development Team*) si comparte con el resto de usuarios su código. Por tanto, el contenido del proyecto está en constante desarrollo para cumplir con las necesidades de los usuarios y para estar al día con cualquier novedad relacionada con los vehículos en los que se puede implementar el código, especialmente aeronaves, pero también rovers (vehículos terrestres de exploración), y vehículos acuáticos, incluidos submarinos.

A lo largo de este trabajo, el nombre *ArduPilot* no se va a emplear como referencia al proyecto, sino al firmware dedicado a aeronaves, el código desarrollado dentro del proyecto que hace posible el vuelo autónomo de una aeronave. Este firmware es un tipo de software desarrollado para ser únicamente de lectura una vez instalado en el ordenador de a bordo (esta grabación en el hardware se llama realizar un *flash*) y que está, por tanto, programado para proporcionarle instrucciones constantemente.

El código de *ArduPilot* se encuentra en la plataforma de desarrollo de software *GitHub*, desde donde se puede descargar todo el repositorio mediante la clonación del directorio para su uso inmediato en un ordenador o directamente en un controlador, siempre que el código esté compilado. Dentro de este repositorio se incluyen todas las librerías, configuraciones y códigos que proporcionan instrucciones a cada uno de los tipos de vehículos en los que se puede implementar, además de documentos informativos sobre todos estos archivos.

Para la implementación del piloto automático dentro de la placa de control del UAV se empleará un código compilado previamente (*build*) de *ArduPlane*, que es la parte del piloto automático dirigida a aeroplanos, y que proporciona el propio proyecto, tanto por facilidad como por el hecho de que el código existente se adecúa a las características y necesidades de la aeronave HERMES-UPV. Como

se ha explicado con anterioridad, la modificación del código base del piloto automático no es un objetivo de este proyecto.

5.2 Obtención de parámetros

En la documentación de *ArduPilot* [17] puede encontrarse información sobre cómo configurar los controladores de las superficies de control y el empuje de forma manual, es decir, modificando uno a uno los parámetros hasta que el comportamiento de la aeronave sea el deseado, pero dado el carácter académico de este proyecto y como modo de comprensión del modelo de cara a optimizaciones del UAV futuras, se ha optado por realizar un análisis de los controladores de *ArduPilot* para obtener los parámetros de una forma justificada.

La forma de abordar esta tarea es mediante la elaboración de un nuevo modelo en *Simulink*, equivalente al modelo original con el que se ha trabajado en el capítulo anterior, donde los controladores se sustituyen por unos nuevos que representan a los de *ArduPilot*. El objetivo será comparar las respuestas obtenidas ante las mismas entradas en ambos modelos y minimizar el error entre estas, pero para ello es necesario simular los modelos con diferentes valores y combinaciones de parámetros. Por tanto, se ha optado por emplear un algoritmo genético que busca la mejor combinación de parámetros para minimizar una función objetivo.

5.2.1 Algoritmo Genético

Una opción para ajustar un controlador en base al comportamiento deseado del sistema que lo contiene es el uso de un algoritmo de optimización que sea capaz de minimizar una función $J(x)$. En el caso del ajuste de un PID hace falta contar con un modelo, por ejemplo en *Simulink*, y definir la función $J(x)$ como indicador de la calidad del proceso de optimización.

Dado que en este caso se busca minimizar el error entre dos respuestas, se pueden emplear varios indicadores basados en la diferencia entre una referencia (modelo original) y la variable que se quiere controlar (modelo *ArduPilot*). El tipo de error en el que se ha basado la optimización en este caso ha sido el *Integral Absolute Error* o IAE, donde se integra a lo largo del tiempo de simulación el valor absoluto del error obtenido entre la referencia y la variable de control, como se muestra en la ecuación 5.1, donde la x representa las variables a ajustar, $e(x,t)$ es el error entre las dos respuestas y $tsim$ es el tiempo de simulación del modelo.

$$J_{IAE}(x) = \int_0^{tsim} |e(x,t)|\ dt \tag{5.1}$$

El motivo por el que el algoritmo empleado en este proyecto se denomina algoritmo genético es porque está basado en la evolución biológica. Esto quiere decir que se simula y estudia la evolución de una población a lo largo de varias generaciones, donde los individuos que surgen en cada generación se originan por recombinaciones genéticas o por mutaciones y pasan a generaciones posteriores los que más se ajustan a unos criterios específicos de selección. De este modo, la gran mayoría de los individuos de la última generación tenderán a ser similares entre sí y a cumplir esos criterios de selección. El paralelismo entre la evolución biológica y la optimización de parámetros de un PID se encuentra en la identificación de elementos: cada individuo equivale a una simulación con una combinación diferente de valores de parámetros, y cada generación sería una nueva iteración donde las nuevas combinaciones de parámetros parten de las que más se han ajustado a los criterios en la generación anterior, más algunas combinaciones aleatorias que representan mutaciones. Por tanto,

en cada generación se harán tantas simulaciones como individuos contenga esta, y en las últimas generaciones se podrá considerar que se ha alcanzado un óptimo local, ya que es un algoritmo basado en estadística que no garantiza hallar un óptimo global.

Este algoritmo se ha implementado en *Matlab* mediante la función *ga.m* proporcionada por el CPOH (figuras 9.1 y 9.2 en el anexo), en la cual se generan los individuos de la población y se crean nuevas generaciones, teniendo en cuenta que es necesario definir rangos de valores entre los que se deben encontrar los parámetros y que se puede introducir una serie de valores iniciales alrededor de los cuales se espera que los parámetros óptimos se encuentren, los cuales servirán para generar los primeros individuos. También se puede especificar el tamaño de la población y el número de generaciones que se quiere estudiar. Dentro de la función del algoritmo genético se llama a la función de optimización, que en este caso es la llamada IAE debido al error en el que se basa mencionado anteriormente, la cual simula el modelo en Simulink y obtiene el valor de la función de error. En cada iteración se escoge la serie de parámetros que lleve al valor mínimo de la función de error y entre todas las iteraciones se escoge de nuevo el valor más pequeño.

5.2.2 Modelo de Ardupilot en Simulink

Dentro del repositorio de *ArduPilot* [18] se encuentran varios archivos *.ccp* que contienen en lenguaje C++ información de la estructura de los PID del controlador, incluyendo definiciones de parámetros y las funciones que los emplean. A partir de esta información y la disponible en la documentación de *ArduPilot* en la web, es posible diseñar un modelo en *Simulink* similar al que se encontraría en el controlador de la aeronave, donde las diferencias con el modelo creado a lo largo de este trabajo son los bloques de los controladores y de navegación. Debido al elevado cálculo computacional que sería necesario para optimizar todos los parámetros al mismo tiempo, se ha dividido la optimización en diferentes partes, estando cada una de ellas dedicada a un controlador en concreto y donde el resto de controladores se mantienen iguales que en el modelo original.

Teniendo en cuenta que los controladores en *ArduPilot* por lo general no emplean radianes como unidad de medida de los ángulos, sino grados, y asignándole el valor de la unidad al elemento escalador que emplea *ArduPilot* para representar un cambio en el comportamiento de las superficies con la variación de la velocidad de vuelo, los bloques creados son los siguientes:

Controladores del lazo interno

- Controlador del ángulo de alabeo

El bloque de control del roll tiene la estructura mostrada en la figura 5.1, donde a partir de la diferencia entre el ángulo de alabeo instantáneo y el de referencia y la velocidad de cambio de este ángulo (derivada temporal del roll) se obtiene la deflexión necesaria en radianes de los alerones.

La estrategia de control que sigue este bloque es un PID donde cada uno de los tres parámetros P, I y D parten de una combinación de los parámetros $RLL2SRV\_P$, que es la ganancia proporcional entre la demanda de roll y la deflexión de alerones; $RLL2SRV\_I$, que es la ganancia del integrador que corrige las desviaciones del ángulo de roll a largo plazo; $RLL2SRV\_D$, que es la ganancia de amortiguamiento entre la aceleración del ángulo de roll y los alerones; y $RLL2SRV\_TCONST$, que es el tiempo que se espera que tarde el controlador en pasar de un ángulo de roll específico al roll de referencia. El único valor que se mantiene igual es la ganancia derivativa $K\_D = RLL2SRV\_D$, mientras las nuevas K_I y K_P serán las mostradas en las ecuaciones 5.2 y 5.3:

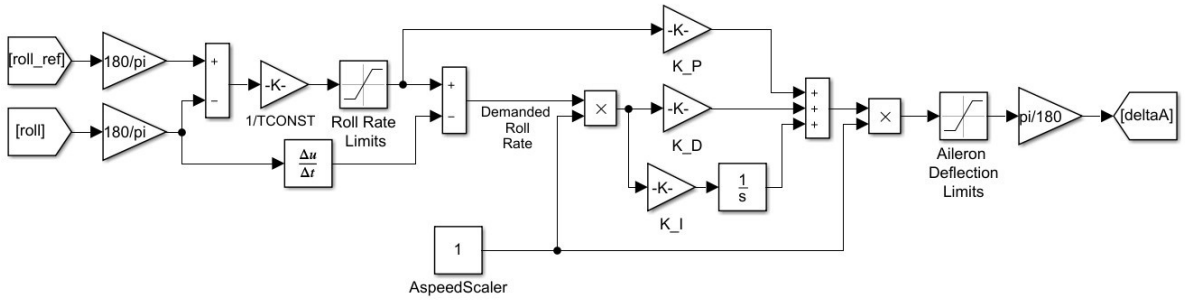

Figura 5.1: Simulink Roll *ArduPilot*

$$K\_I{=}RLL2SRV\_I\ RLL2SRV\_TCONST \tag{5.2}$$

$$K\_P{=}(RLL2SRV\_P{-}K\_I)\ RLL2SRV\_TCONST{-}K\_D \tag{5.3}$$

En el modelo de *Simulink* se incluyen bloques de saturación que limitan por un lado la velocidad máxima en valores absolutos de cambio del ángulo de roll a 60º/s y por otro lado la deflexión máxima de los alerones a 30º. Dado que los controladores de *ArduPilot* no se han obtenido a partir de funciones de transferencia de un modelo linealizado, los resultados obtenidos de ellos no son diferencias de valores, sino que son directamente los valores que deben tomar las variables. Por tanto, no es necesario sumar en los controladores de alabeo, cabeceo y aceleración lateral estos valores en el punto de equilibrio.

– Controlador del ángulo de cabeceo

La estructura del bloque de control del ángulo de cabeceo (figura 5.2) es como la del de control de alabeo pero con una demanda adicional de ángulo de pitch introducida para compensar la pérdida de altitud al realizar giros coordinados. Esta compensación se obtiene mediante la expresión 5.4 introducida en el subsistema nombrado *Roll Compensation* en el modelo de *Simulink* multiplicada por una ganancia $PTCH2SRV\_RLL$, que es una de las variables a optimizar en el controlador. Las otras variables son $PTCH2SRV\_P$, $PTCH2SRV\_I$, $PTCH2SRV\_D$ y $PTCH2SRV\_TCONST$, empleadas para obtener las ganancias proporcional, derivativa e integrativa del mismo modo que en el caso del control del alabeo, tal como muestran las expresiones 5.5, 5.6 y 5.7.

$$roll\_comp = \cos\theta \left| \frac{g}{V} \tan\phi \sin\phi \right| \tag{5.4}$$

$$K\_D{=}PTCH2SRV\_D \tag{5.5}$$

$$K\_I{=}PTCH2SRV\_I\ PTCH2SRV\_TCONST \tag{5.6}$$

$$K\_P{=}(PTCH2SRV\_P{-}K\_I)\ PTCH2SRV\_TCONST{-}K\_D \tag{5.7}$$

En este controlador también se han introducido límites en la deflexión máxima de los elevadores, de 30º en valor absoluto, y en la velocidad de cambio del ángulo de pitch, limitado también a 30º en valor absoluto.

– Controlador de la aceleración lateral

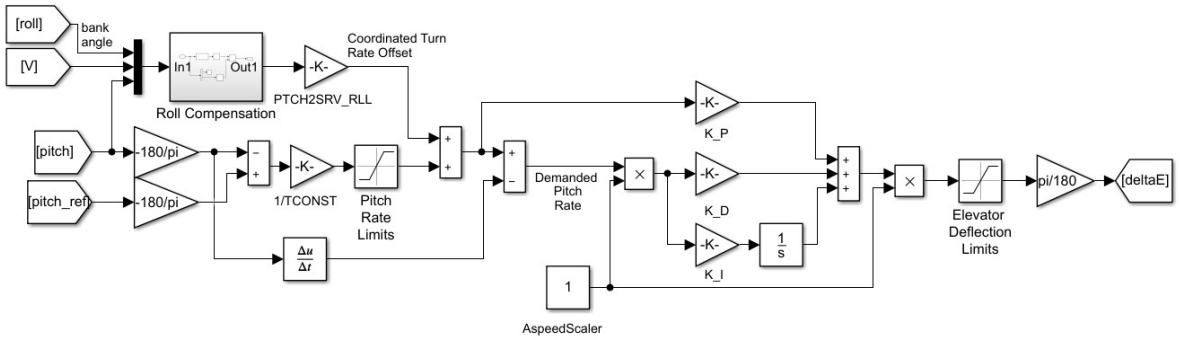

Figura 5.2: Simulink Pitch *ArduPilot*

El controlador de la aceleración lateral o *vDot* busca obtener la deflexión necesaria del timón de cola para compensar las aceleraciones laterales que puedan surgir durante un giro. Por tanto, la aceleración lateral de referencia es 0, y por ello *vDot* se resta en el diagrama de *Simulink* de la figura 5.3. Como se ve en el esquema, el controlador trabaja con cambios del ángulo de guiñada, y mediante la ganancia $YAW2SRV\_SLIP$ se obtiene el cambio en el yaw necesario para que la aceleración lateral sea nula.

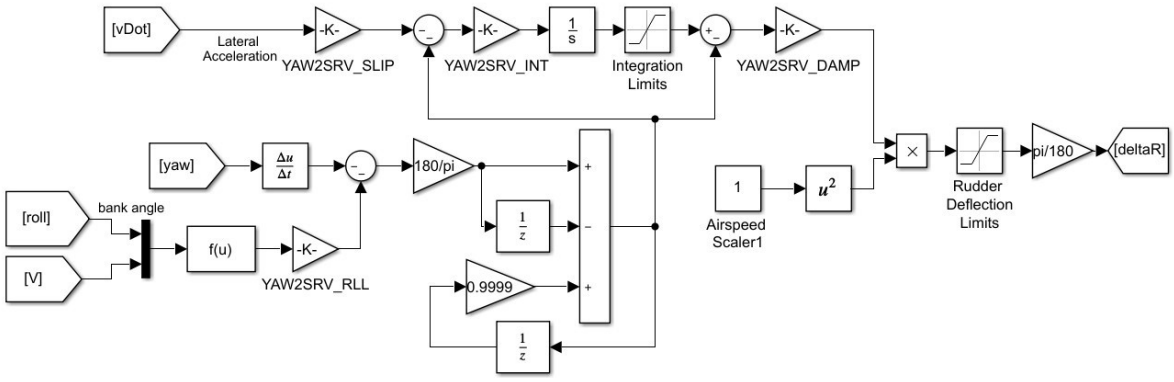

Figura 5.3: Simulink Yaw *ArduPilot*

Además de tener en cuenta la aceleración lateral propia de la aeronave, debida al viento, por ejemplo, el controlador también debe ser capaz de compensar los cambios en el ángulo de guiñada debidos a cambios en el ángulo de alabeo: el controlador hace así que los giros sean coordinados y no se pierda altura. Este efecto se añade al cambio del ángulo de yaw necesario mediante un filtro de paso alto. Este filtro se aplica a la suma de la variación del ángulo de guiñada y del factor de compensación del roll, obtenido con la expresión 5.8, donde $YAW2SRV\_RLL$ es la ganancia de coordinación del yaw, un parámetro que hay que obtener.

$$roll\_comp = \frac{g}{V} \sin\phi \; YAW2SRV\_RLL \tag{5.8}$$

Una vez obtenida la velocidad de cambio del ángulo de yaw deseada, el valor de la deflexión del timón de cola necesaria se obtiene mediante su integración con una ganancia integral $YAW2SRV\_INT$ y mediante la ganancia proporcional $YAW2SRV\_DAMP$.

Los bloques de saturación incluidos en el diagrama limitan la deflexión del timón de cola a 45° en valor absoluto.

– Controlador de la velocidad

El controlador de la velocidad se basa en la estrategia TECS (*Total Energy Control System*), que reparte la energía total disponible del sistema en energía empleada para modificar la velocidad y en energía empleada para controlar la altitud. En este proyecto se ha optado por controlar (con un reparto equitativo de energía) la altitud con los elevadores y la velocidad con la palanca de gases, pero también se podría haber abordado el problema modificando la altitud con el empuje.

En el esquema de la figura 5.4 se observa el diagrama de *Simulink* empleado para obtener la palanca necesaria para alcanzar una velocidad de referencia, proporcionada por el lazo de control externo (subsección 5.2.2). El algoritmo de *Ardupilot* para controlar la velocidad se encarga en primer lugar de que la palanca de gases proporcione suficiente empuje, y por tanto, suficiente energía mecánica (suma de energía potencial y cinética) como para compensar la fuerza de arrastre en la aeronave y sobrepasarla hasta alcanzar la velocidad y la altitud deseadas; y por otro lado se encarga de que el balance entre energía potencial y energía cinética sea apropiado, que en este caso tienen el mismo valor. La ganancia $K\_STE2Thr$ junto con la constante $TIME\_CONST$ sirve para relacionar la energía específica necesaria con el nivel de la palanca correspondiente.

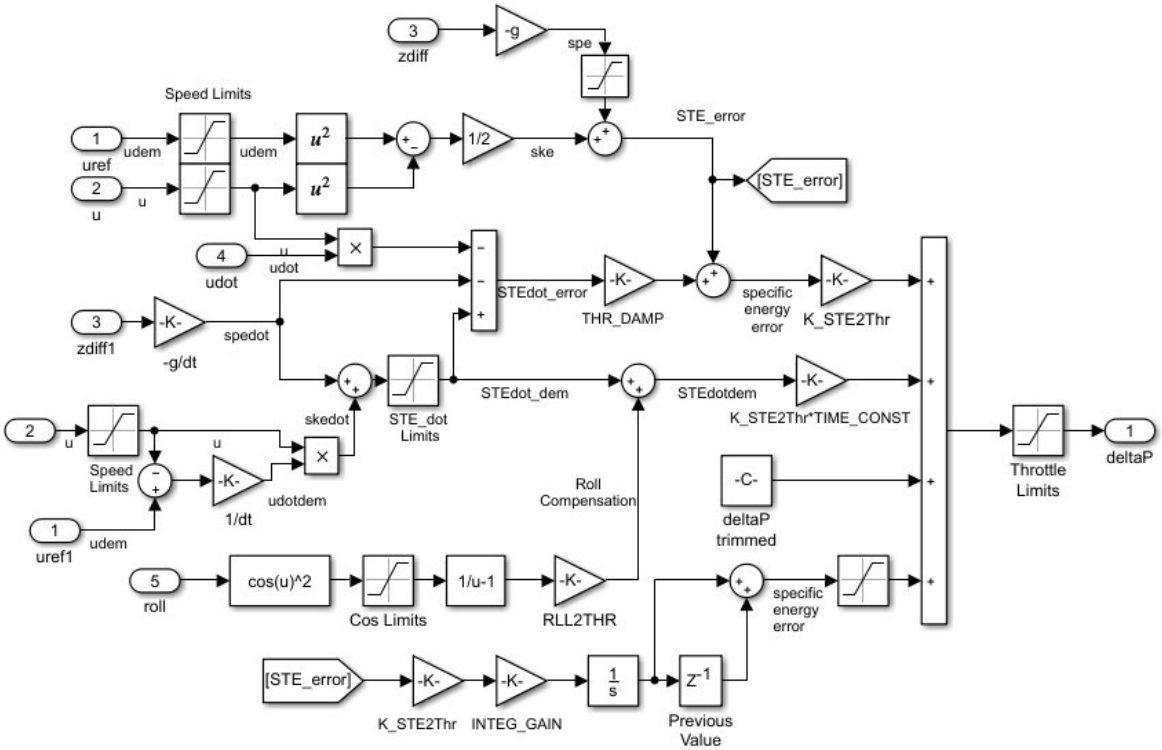

Figura 5.4: Simulink Throttle ArduPilot

En el diagrama se puede ver como en este caso hay que añadirle a la posición de la palanca de gases necesaria la del punto de equilibrio, ya que con la diferencia de energía total se obtiene únicamente el aumento de palanca necesaria, pero no el valor final. Por último, igual que la deflexión de las

superficies de control está limitada por ciertos valores, la posición de la palanca también se limita para situarse en todo momento entre 0 y 1.

Controladores del lazo externo (obtención de valores de referencia)

– Ángulo de alabeo de referencia

El controlador de la trayectoria está diseñado para proporcionar el ángulo de alabeo de referencia necesario para que la aeronave siga una serie de waypoints, y está basado en una versión ajustada a *ArduPilot* de la ley de navegación L_1 [19]. Esta ley de navegación calcula, por un lado, el ángulo de error existente entre el vector de la velocidad de la aeronave y la recta que une el vehículo con un punto de referencia L_1, el cual se sitúa sobre la línea que une un waypoint con el siguiente (ángulo Nu), y por otro obtiene la magnitud de la velocidad de la aeronave proyectada sobre el suelo o *groundspeed* (GS). Como muestra el esquema en *Simulink* de la figura 5.5, en la transformación entre el error en la trayectoria y el ángulo de alabeo necesario para corregirlo intervienen además de Nu y GS, por un lado, los parámetros $L1\_damping$ y $L1\_period$, que representan respectivamente el amortiguamiento en el cálculo de la trayectoria y el tiempo que debe tardar la aeronave en realizar un giro, o lo que es lo mismo, la agresividad de giro; y por otro lado, el coseno del ángulo de ataque, que se emplea para escalar el ángulo de alabeo.

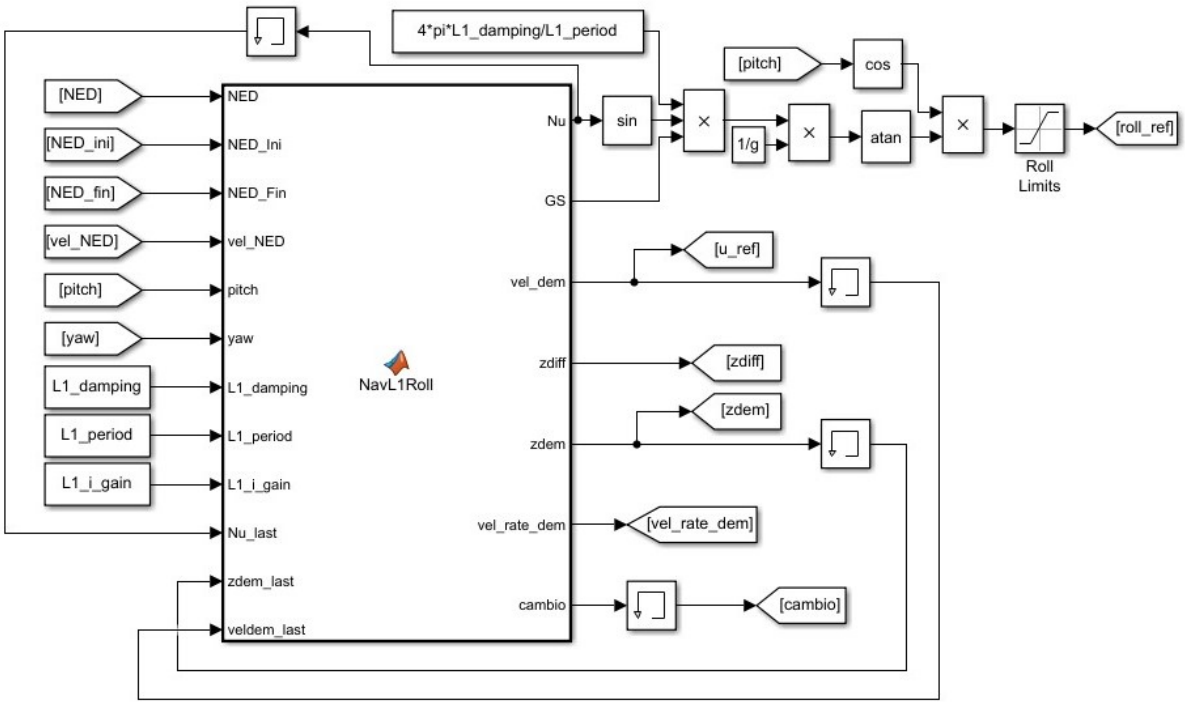

Figura 5.5: Esquema en *Simulink* de la trayectoria de referencia para *ArduPilot*

La función *NavL1Roll* creada en el entorno de trabajo de *Simulink* (cuyo código se encuentra en las figuras 9.3 y 9.4 del anexo) y basada en la ley de navegación L_1 obtiene el ángulo Nu dependiendo de la región en la que se encuentre la aeronave. La figura 5.6 resume las tres posibles regiones y las expresiones de la ecuación 5.9 introducen definiciones básicas de los vectores que se emplean en los cálculos, teniendo en cuenta que A y B representan las coordenadas (longitud y latitud) de los waypoints de origen y destino respectivamente, y C son las coordenadas en las que se encuentra la aeronave en cada instante. El vector $\vec{GS}$ que se emplea a lo largo de los cálculos es la proyección en el plano XY del vector velocidad de la aeronave en coordenadas NED.

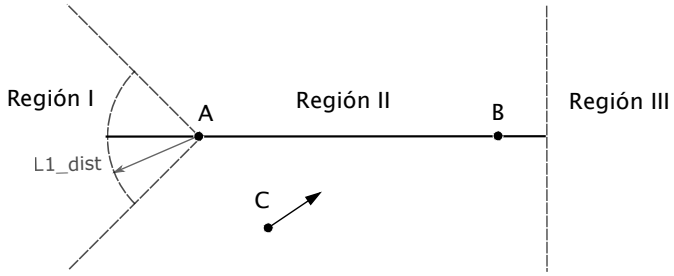

Figura 5.6: Posibles regiones de ubicación de la aeronave según la ley de navegación L_1

$$\vec{AB} = \frac{B-A}{|B-A|}; \quad \vec{AC} = \frac{C-A}{|C-A|}; \quad \vec{BC} = \frac{C-B}{|C-B|} \tag{5.9}$$

- Región I: la aeronave se encuentra a una distancia mayor que la distancia $L1\_dist$ (ecuación 5.10) de referencia del punto A y dentro del arco de circunferencia entre 135º y -135º centrado en A. En este caso, aunque sí se considera que la aeronave ha alcanzado este waypoint, el punto de referencia de la trayectoria es el propio waypoint A, y la diferencia de ángulo que tiene que corregir el vector velocidad es, como muestra la ecuación 5.13, la arcotangente entre la componente de la velocidad perpendicular al vector $\vec{AB}$ (ecuación 5.11) y la componente paralela o longitudinal (ecuación 5.12).

$$L1\_dist = \frac{1}{\pi} L1\_damping \cdot L1\_period \cdot GS \tag{5.10}$$

$$xtrackVel = \vec{GS}_{lon} \cdot -\vec{AC}_{lat} - \vec{GS}_{lat} \cdot -\vec{AC}_{lon} \tag{5.11}$$

$$ltrackVel = \vec{GS} \cdot -\vec{AC} \tag{5.12}$$

$$Nu = atan2(xtrackVel, ltrackVel) \tag{5.13}$$

- Región II: la aeronave se encuentra a una distancia del waypoint sobre la prolongación de la recta que une A y B equivalente a tres segundos de vuelo o más, por lo que no se ha alcanzado este punto debido a que la distancia al punto B es mayor que el radio definido de 50 metros. En este caso, el punto de referencia es el waypoint B y el ángulo de error en la trayectoria se obtiene con el mismo procedimiento que en la región I (ecuaciones 5.14, 5.15 y 5.16).

$$xtrackVel = \vec{GS}_{lon} \cdot -\vec{BC}_{lat} - \vec{GS}_{lat} \cdot -\vec{BC}_{lon} \tag{5.14}$$

$$ltrackVel = \vec{GS} \cdot -\vec{BC} \tag{5.15}$$

$$Nu = atan2(xtrackVel, ltrackVel) \tag{5.16}$$

- Región III: la aeronave se encuentra en cualquier punto no incluido en las dos regiones anteriores, por lo que se sigue la ley general de navegación L_1, donde el punto de referencia de la trayectoria se encuentra sobre la recta que une los waypoints A y B. En esta región, el ángulo de error de la trayectoria se obtiene mediante la suma de los ángulos Nu_1 y Nu_2: el ángulo Nu_2 es el ángulo existente entre el vector de velocidad de la aeronave y el vector unitario $\vec{AB}$, obtenido del mismo modo que el ángulo Nu en las regiones I y II (ecuaciones 5.17, 5.18 y 5.20); mientras el ángulo Nu_1 se encuentra entre el vector $\vec{AB}$ y la línea que une el punto C con el punto de referencia L_1 a una distancia $L1\_dist$ sobre la recta entre A y B (ecuaciones 5.19 y 5.21). Dado que es posible que la distancia $L1\_dist$ sea menor que la distancia entre C y la recta mencionada, el ángulo Nu_1 sólo puede tomar valores entre -45º y 45º. En el caso de que el ángulo Nu_1 sea menor que 5º, se aplica un integrador para que la distancia entre la aeronave y la recta entre A y B (*crosstrack error*) tienda a cero. Por tanto, si se cumple esta condición, la ley de navegación le suma i_{xtrack} a Nu_1, que es el producto entre Nu_1, una ganancia $L1\_i\_gain$ y el tiempo de paso del modelo dt.

$$xtrackVel = \vec{GS}_{lon} \cdot \vec{AB}_{lat} - \vec{GS}_{lat} \cdot \vec{AB}_{lon} \tag{5.17}$$

$$ltrackVel = \vec{GS} \cdot \vec{AB} \tag{5.18}$$

$$Nu_1 = asin\left(\frac{\vec{AC} \times \vec{AB}}{L1\_dist}\right) \tag{5.19}$$

$$Nu_2 = atan2(xtrackVel, ltrackVel) \tag{5.20}$$

$$Nu = Nu_1 + Nu_2 + i_{xtrack} \tag{5.21}$$

– Velocidad de referencia

La velocidad de referencia se calcula dentro de la función *NavL1Roll* directamente, ya que sólo depende de la velocidad objetivo definida para cada waypoint y de los cambios de velocidad mínimo y máximo definidos según la energía total disponible en el sistema. Por tanto, la velocidad objetivo en cada tramo de la misión es la velocidad del waypoint de referencia: si la aeronave se encuentra en la región I, será la velocidad del waypoint A, y si se encuentra en la región II o en la región III, la velocidad de referencia será la del waypoint B. Dado que el modelo en *Simulink* tiene un tiempo de paso determinado, es necesario definir una velocidad demandada para cada uno de los intervalos de tiempo, por lo que la aeronave puede acelerar o decelerar según los límites *velRateMax* y *velRateMin*, que dependen cada uno de la velocidad de ascenso máximo y la velocidad de descenso mínima, respectivamente, y de la gravedad y la velocidad en cada momento (ecuaciones 5.22 y 5.23).

$$velRateMax = \frac{1}{2}\frac{Climb_{max} \cdot g}{u} \tag{5.22}$$

$$velRateMin = -\frac{1}{2}\frac{Sink_{min} \cdot g}{u} \tag{5.23}$$

De este modo, si el incremento de velocidad demandado en un intervalo de tiempo es mayor o menor que los límites *velRateMax* y *velRateMin* respectivamente, la velocidad vendrá determinada por ellos, y si el incremento se sitúa dentro de los límites, la velocidad demandada será directamente la velocidad objetivo.

– Ángulo de cabeceo de referencia

Por último, el ángulo de cabeceo de referencia o $pitch\_ref$ se obtiene mediante las funciones creadas *NavL1Pitch* (figura 9.5 en el anexo) e *integ* (figura 9.6 en el anexo) en *Simulink* y las operaciones mostradas en el esquema de la figura 5.7. En la primera función se obtiene esencialmente el error en el balance de energía específica del sistema, es decir, se calcula la diferencia entre la energía demandada y la disponible SEB_error (*Specific Energy Balance*) a partir de las energías cinética y potencial obtenidas como parte de la estrategia TECS, explicada anteriormente. Una vez obtenido el error del balance de energía específica, este se multiplica por una ganancia llamada i_gain, se integra con el objetivo de minimizarlo y mediante la función *integ* se limita entre los valores $integ\_SEB\_min$ y $integ\_SEB\_max$, definidos en la función *NavL1Pitch*.

Por último, para obtener el ángulo de cabeceo de referencia, a la salida del integrador se le añade la variable $temp$, definida en la ecuación 5.24, y este resultado es dividido entre la variable $gainInv$, definida en la ecuación 5.25. En el esquema se han introducido unos límites inferior y superior al ángulo demandado de cabeceo de 15° y -15° para evitar que la aeronave entre en pérdida.

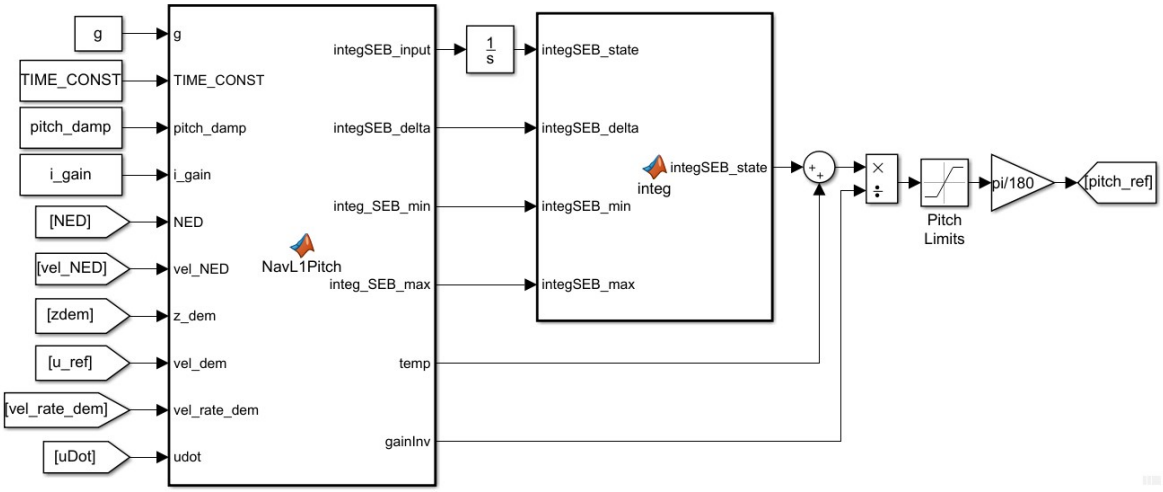

Figura 5.7: Esquema en *Simulink* de la altitud de referencia para *ArduPilot*

$$temp = SEB\_error + SEBdot\_dem \cdot TIME\_CONST + SEBdot\_error \cdot pitch\_damp \quad (5.24)$$

$$gainInv = u \cdot TIME\_CONST \cdot g \quad (5.25)$$

Cabe destacar que la función *NavL1Pitch* necesita recibir como entrada una altitud de referencia, y esta se calcula dentro de la función *NavL1Roll* mediante un procedimiento equivalente al de la obtención de la velocidad de referencia, donde la velocidad vertical de la aeronave se ve limitada por la velocidad mínima de descenso y la velocidad máxima de ascenso.

5.2.3 Optimización

Una vez se ha implementado el modelo de *ArduPilot* en *Simulink* se puede ejecutar la optimización en *Matlab* para cada uno de los bloques de control. Tras obtener los valores de los parámetros que minimizan la función de error en cada caso, se debe simular de nuevo el modelo con estos valores y comprobar que el comportamiento del sistema de *ArduPilot* es similar al del sistema de referencia.

Controlador del ángulo de alabeo

Para el caso del roll, los parámetros que hace falta optimizar son $RLL2SRV\_P$, $RLL2SRV\_I$, $RLL2SRV\_D$ y $RLL2SRV\_TCONST$. El script de *Matlab* que llama al algoritmo genético es *optimizRoll.m*, e incluye las definiciones de los campos de *gaDat* necesarios para la optimización y valores de parámetros constantes que impone el usuario para limitar los resultados a ciertos valores mínimos y máximos, como es en este caso $RLL2SRV\_RMAX$ (figura 5.8). Se ha optado por contar con 10 generaciones con 30 individuos cada una, además de la población inicial, en la cual uno de los individuos tiene los valores de los parámetros mostrados en *gaDat.indini*, que son los valores predeterminados en *ArduPilot*. En todas las optimizaciones del resto de controladores se han empleado los mismos valores de generaciones, individuos y tiempo de simulación, que es de 110 segundos.

```
%Mejor combinacion de 4 parámetros de roll: P, I, D y TCONST
%para que el error entre el roll resultante en ARDUPILOT y el del modelo
%ETSID sea mínimo.

RLL2SRV_RMAX=60;
RLL2SRV_Pmin=0.1; RLL2SRV_Pmax=4.0;
RLL2SRV_Imin=0; RLL2SRV_Imax=1.0;
RLL2SRV_Dmin=0; RLL2SRV_Dmax=0.1;
RLL2SRV_TCONSTmin=0.4; RLL2SRV_TCONSTmax=1.0;
gaDat.FieldD=[RLL2SRV_Pmin RLL2SRV_Imin RLL2SRV_Dmin RLL2SRV_TCONSTmin;
              RLL2SRV_Pmax RLL2SRV_Imax RLL2SRV_Dmax RLL2SRV_TCONSTmax];
gaDat.MAXGEN=10;
gaDat.NIND=30;
gaDat.indini=[1 0.3 0.08 0.5];
```

Figura 5.8: Definición de variables y límites de variables del controlador del roll en el script *optimizRoll.m*

Una vez terminan las iteraciones, los resultados de los parámetros que minimizan la función de error se simulan en el modelo de Simulink a modo de comprobación, obteniendo una gráfica que muestra al mismo tiempo el roll en el modelo llamado ETSID y en el modelo llamado *ArduPilot*. Como se puede observar en la figura 5.9, las dos curvas son prácticamente coincidentes, por lo que los parámetros obtenidos de roll son apropiados para la implementación incial. Los valores de estos parámetros son $RLL2SRV\_P = 0.66$, $RLL2SRV\_I = 0.10$, $RLL2SRV\_D = 0.08$ y $RLL2SRV\_TCONST = 1.00$.

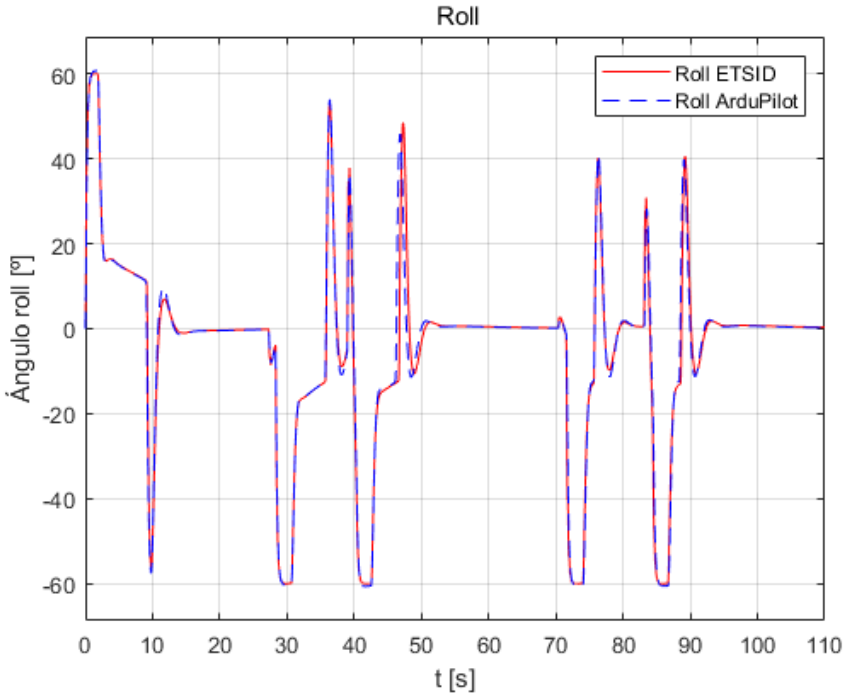

Figura 5.9: Comparación roll obtenido en el modelo ETSID con el obtenido en el modelo *ArduPilot*

Controlador del ángulo de cabeceo

En el caso del pitch, los parámetros que se quieren optimizar son $PTCH2SRV\_P$, $PTCH2SRV\_I$, $PTCH2SRV\_D$, $PTCH2SRV\_TCONST$ y $PTCH2SRV\_RLL$, y esta tarea se lleva a cabo mediante el script de Matlab *optimizPitch.m*, del cual se muestra un extracto en la figura 5.10.

```
%Mejor combinacion de 5 parámetros de pitch: P, I, D, TCONST y RLL
%para que el error entre el pitch resultante en ARDUPILOT y el del modelo
%ETSID sea mínimo.

PTCH2SRV_RMAX_UP=30;
PTCH2SRV_RMAX_DN=30;
PTCH2SRV_IMAX=30;
PTCH2SRV_Pmin=0.1; PTCH2SRV_Pmax=3.0;
PTCH2SRV_Imin=0.15; PTCH2SRV_Imax=0.5;
PTCH2SRV_Dmin=0; PTCH2SRV_Dmax=0.1;
PTCH2SRV_TCONSTmin=0.4; PTCH2SRV_TCONSTmax=1.0;
PTCH2SRV_RLLmin=0.7; PTCH2SRV_RLLmax=1.5;
gaDat.FieldD=[PTCH2SRV_Pmin PTCH2SRV_Imin PTCH2SRV_Dmin PTCH2SRV_TCONSTmin PTCH2SRV_RLLmin;
              PTCH2SRV_Pmax PTCH2SRV_Imax PTCH2SRV_Dmax PTCH2SRV_TCONSTmax PTCH2SRV_RLLmax];
gaDat.MAXGEN=10;
gaDat.NIND=30;
gaDat.indini=[1 0.3 0.04 0.5 1];
```

Figura 5.10: Definición de variables y límites de variables del controlador del pitch en el script *optimizPitch.m*

Los resultados obtenidos tras la optimización se muestran en la gráfica de la figura 5.11, donde se observa que el comportamiento del controlador de ArduPilot es similar al del controlador del modelo ETSID. Los valores de los parámetros que proporcionan este resultado son $PTCH2SRV\_P = 3.00$, $PTCH2SRV\_I = 0.23$, $PTCH2SRV\_D = 0.01$, $PTCH2SRV\_TCONST = 0.40$ y $PTCH2SRV\_RLL = 0.75$. Si se comparan estas ganancias con las equivalentes del controlador

inicial, se observan diferencias en los signos debido a que los ejes Z se han considerado con signos opuestos en cada uno de los controladores.

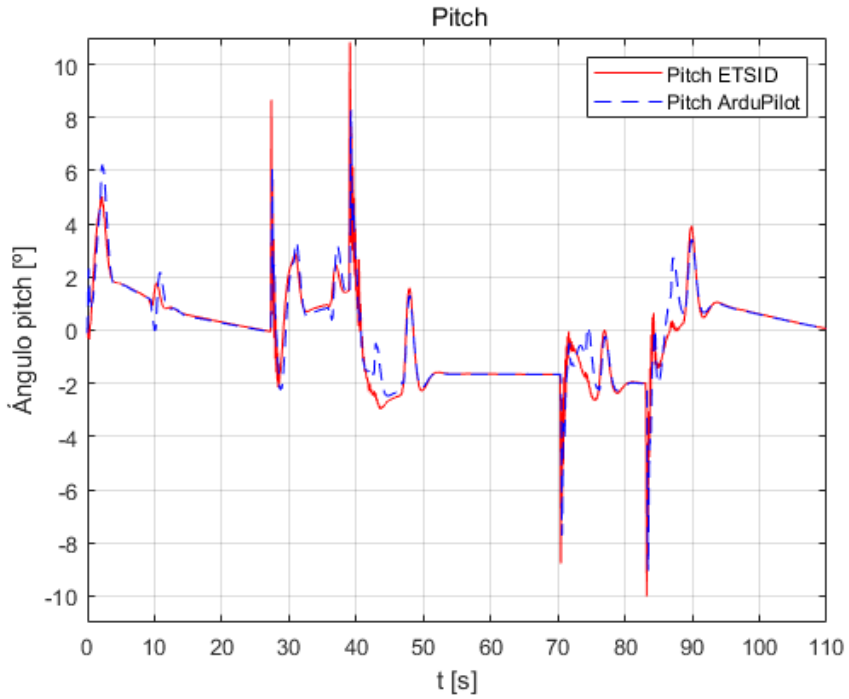

Figura 5.11: Comparación pitch obtenido en el modelo ETSID con el obtenido en el modelo *ArduPilot*

Controlador de la aceleración lateral

Los parámetros necesarios para el control de la aceleración lateral (también referido como controlador de yaw en la documentación) se han obtenido de una forma similar al control del roll, ya que la estructura es esencialmente la misma. El script de optimización *optimizYaw.m*, mostrado en la figura 5.12 define los rangos de los parámetros optimizados.

```
%Mejor combinacion de 4 parámetros de aceleración lateral: SLIP, INT, DAMP
%y RLL para que el error entre la aceleración lateral resultante en ARDUPILOT
%y la del modelo ETSID sea mínimo.

RLL2SRV_IMAXyaw=15;
YAW2SRV_SLIPmin=0; YAW2SRV_SLIPmax=4.0;
YAW2SRV_INTmin=0; YAW2SRv_INTmax=2.0;
YAW2SRV_DAMPmin=0; YAW2SRV_DAMPmax=2.0;
YAW2SRV_RLLmin=0.9; YAW2SRV_RLLmax=1.1;
gaDat.FieldD=[YAW2SRV_SLIPmin YAW2SRV_INTmin YAW2SRV_DAMPmin YAW2SRV_RLLmin;
              YAW2SRV_SLIPmax YAW2SRv_INTmax YAW2SRV_DAMPmax YAW2SRV_RLLmax];
gaDat.MAXGEN=10;
gaDat.NIND=30;
gaDat.indini=[0.52 1.47 0.11 0.99];
```

Figura 5.12: Definición de variables y límites de variables del controlador de la aceleración lateral en el script *optimizYaw.m*

Los valores de los parámetros obtenidos son $YAW2SRV\_SLIP = 4.00$, $YAW2SRV\_INT = 2.00$, $YAW2SRV\_DAMP = 0.15$ y $YAW2SRV\_RLL = 1.10$.

Una vez simulado el modelo con los nuevos valores, los resultados obtenidos no son exactamente iguales que en el modelo ETSID, ya que la aceleración horizontal es notablemente mayor y con más oscilaciones debido a que depende también del ángulo de alabeo y de guiñada, aunque tras unos segundos se estabiliza su valor y vuelve a tener un valor nulo, como se puede ver en la figura 5.13. Por tanto, aunque la respuesta no sea idéntica a la del modelo ETSID, el nuevo controlador tiene un comportamiento dentro de lo esperado.

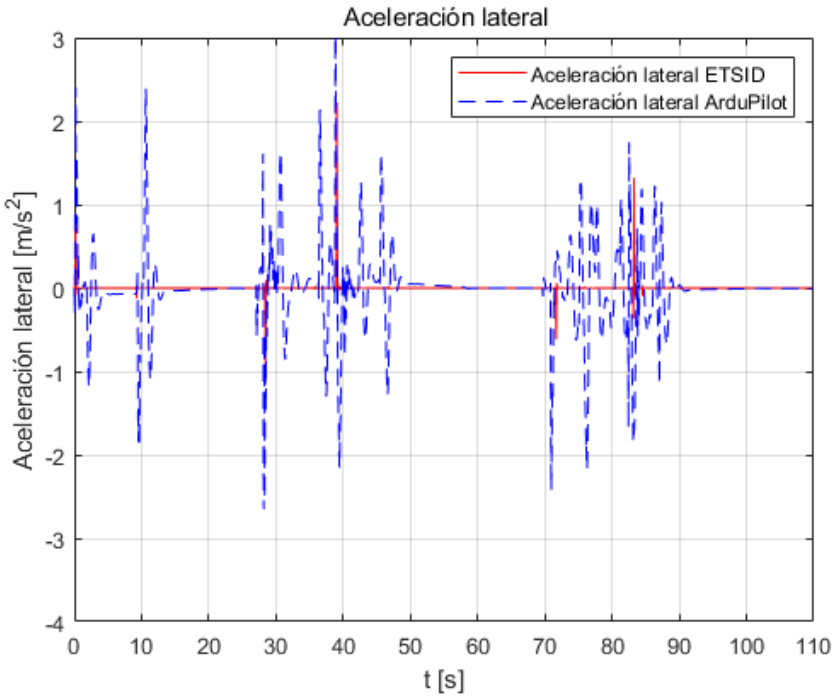

Figura 5.13: Comparación aceleración lateral obtenida en el modelo ETSID con la obtenida en el modelo *ArduPilot*

Controlador de la velocidad

Los parámetros a optimizar para el control de la velocidad son $TIME\_CONST$, THR_DAMP, $RLL2THR$ e $INTEG\_GAIN$, que tienen los rangos de valores especificados en el script *optimizThrotlle.mat*, mostrado parcialmente en la figura 5.14.

Los valores óptimos de los parámetros obtenidos son $TIME\_CONST = 3.00$, $THR\_DAMP = 0.20$, $RLL2THR = 5.00$ e $INTEG\_GAIN = 0.00$, que generan la gráfica mostrada en la figura 5.15, donde las diferencias entre los dos controladores son pequeñas y debidas esencialmente a la dependencia de la altitud y el ángulo de alabeo del controlador.

```
%Mejor combinacion de 4 parámetros de throttle: TIME_CONST,
%THR_DAMP, RLL2THR y INTEG_GAIN para que el error entre la velocidad
%resultante en ARDUPILOT y la del modelo ETSID sea mínimo.

CLMB_MAX=5; SINK_MIN=2; STEdot_MAX=CLMB_MAX*g; STEdot_MIN=-SINK_MIN*g;
THR_MIN=0.05; THR_MAX=1;
TIME_CONSTmin=3; TIME_CONSTmax=10;
THR_DAMPmin=0.1; THR_DAMPmax=1;
RLL2THRmin=5; RLL2THRmax=30;
INTEG_GAINmin=0; INTEG_GAINmax=0.5;
gaDat.FieldD=[TIME_CONSTmin THR_DAMPmin RLL2THRmin INTEG_GAINmin;
              TIME_CONSTmax THR_DAMPmax RLL2THRmax INTEG_GAINmax];
gaDat.MAXGEN=10;
gaDat.NIND=30;
gaDat.indini=[5 0.5 10 0.1];
```

Figura 5.14: Definición de variables y límites de variables del controlador de la velocidad en el script *optimizThrottle.m*

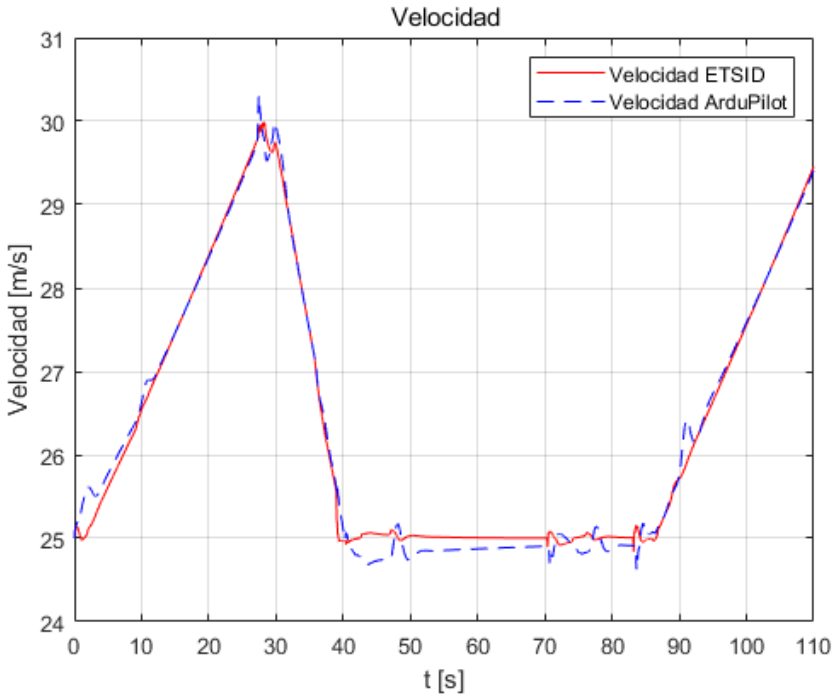

Figura 5.15: Comparación velocidad obtenida en el modelo ETSID con la obtenida en el modelo *ArduPilot*

Controladores del lazo externo (trayectoria y altitud)

Debido a que la ley de navegación implementada en este trabajo es ligeramente diferente a la ley aplicada en el sistema original, no resulta efectivo comparar la trayectoria o la altitud en cada punto con los valores obtenidos en el sistema original, ya que, por ejemplo, la velocidad no es la misma en cada instante y con un modelo se alcanza un waypoint en un momento diferente al otro modelo. Por tanto, la función de error a minimizar estaría comparando coordenadas que no tienen nada que ver entre sí y la optimización de parámetros no sería precisa.

Se ha optado por emplear los parámetros de control de la trayectoria y de la altitud predeterminados para aeronaves de ala fija, con los cuales el comportamiento de la aeronave es correcto: $L1\_damping = 0.70$, $L1\_period = 15.00$, $L1\_i\_gain = 0.08$, $pitch\_damp = 0.90$ e $i\_gain = 0.10$.

Si se combinan todos los bloques con los nuevos parámetros, se obtiene la gráfica de la figura 5.16, que compara el recorrido de la aeronave empleando todos los controladores iniciales y la ley de navegación original con los nuevos controladores y ley de navegación.

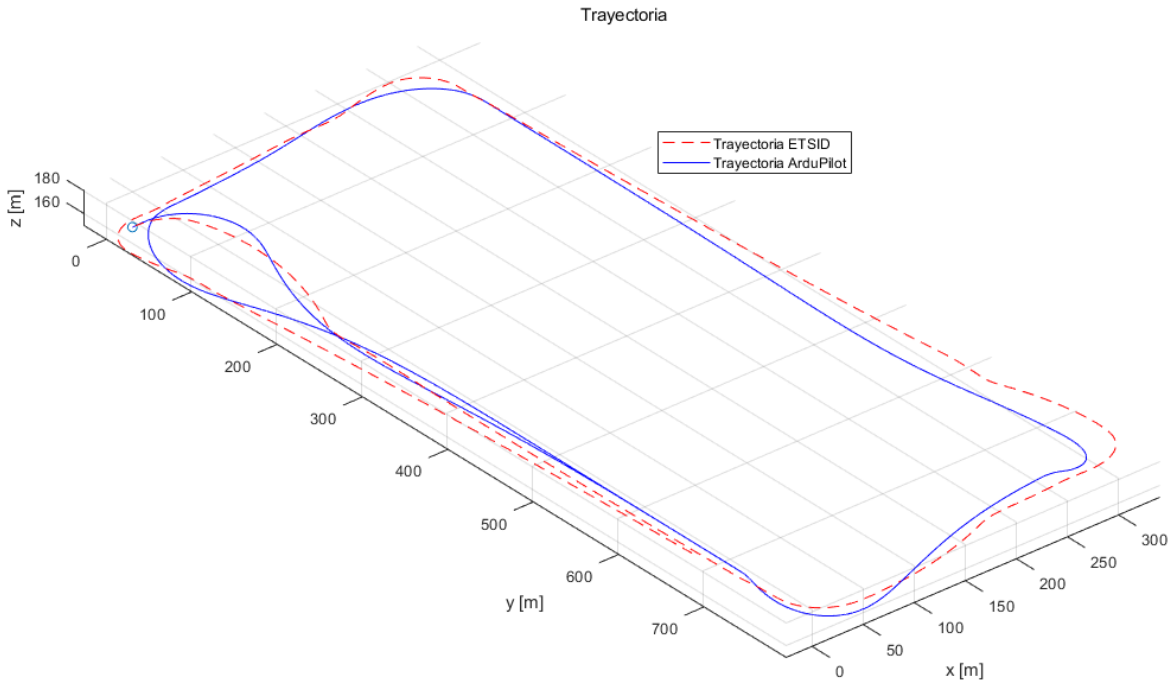

Figura 5.16: Comparación de trayectoria en el modelo ETSID con la obtenida en el modelo ARDU

5.3 Parámetros modificados en *QGroundControl*

Como último paso antes de la validación del sistema de control se deben redefinir los parámetros para los que se ha realizado la optimización, además de modificar parámetros adicionales que están relacionados con el tipo de aeronave. Esta definición de parámetros se puede hacer de dos modos: por una parte se pueden introducir directamente en la interfaz gráfica de *QGroundControl*, el funcionamiento de la cual se explica en el capítulo siguiente (Capítulo 6), o por otra se puede modificar el archivo *plane.parm* dentro del repositorio de *ArduPilot*, donde se definen los parámetros que aparecen por defecto en el piloto automático. Dado que la interfaz de *QGroundControl* es más

intuitiva, se ha optado por introducir los parámetros directamente ahí, los cuales además se pueden guardar para poder ser recuperados en caso de modificaciones.

Por tanto, a modo de resumen, los parámetros que se han obtenido en la optimización y se han introducido en el piloto automático son los mostrados en la tabla tabla 5.1.

Parámetro	Valor
$RLL2SRV\_P$	0.66
$RLL2SRV\_I$	0.10
$RLL2SRV\_D$	0.08
$RLL2SRV\_TCONST$	1.00
$PTCH2SRV\_P$	3.00
$PTCH2SRV\_I$	0.23
$PTCH2SRV\_D$	0.01
$PTCH2SRV\_TCONST$	0.40
$PTCH2SRV\_RLL$	0.75
$YAW2SRV\_SLIP$	4.00
$YAW2SRV\_INT$	2.00
$YAW2SRV\_DAMP$	0.15
$YAW2SRV\_RLL$	1.10
$TIME\_CONST$	3.00
THR_DAMP	0.20
$RLL2THR$	5.00
$INTEG\_GAIN$	0.00
$L1\_damping$	0.70
$L1\_period$	15.00
$L1\_i\_gain$	0.08
$pitch\_damp$	0.90
i_gain	0.10

Cuadro 5.1: Parámetros introducidos en *QGroundControl*

Capítulo 6

Validación mediante simulación de vuelo

Una forma de validar los parámetros obtenidos en el capítulo anterior es evaluar el comportamiento de la aeronave con estos nuevos valores. Dado que el proyecto se encuentra en una fase inicial de desarrollo no es posible emplear la aeronave con sus componentes para la validación del sistema de control, ya que el equipo todavía no cuenta con todos los sensores y la electrónica necesarios para realizar un primer vuelo. Por tanto, una alternativa a un vuelo real es la simulación de la aeronave (tanto con el software como con el hardware) que represente fielmente el comportamiento de un vuelo de la aeronave de este proyecto en una misión de seguimiento de waypoints.

Este capítulo introduce, en primer lugar, el proceso seguido para la simulación de vuelo mediante software y sin contar con el hardware real (*Software In The Loop*) y, en segundo lugar, la transmisión de esta información a la estación de control en tierra *QGroundControl* con una posterior simulación en *FlightGear*.

6.1 *Software In The Loop*

El repositorio de *ArduPilot* incluye una versión de la compilación del código del piloto automático donde se sustituye al hardware de la aeronave por más líneas de código. Este archivo es el *Software In The Loop* empleado para simular el comportamiento de la aeronave. Dado que el código se desarrolló inicialmente en *Linux* y el proyecto se está realizando en un ordenador con el sistema operativo de *Windows*, es necesario emplear aplicaciones adicionales que compatibilicen el SITL con el ordenador, como puede ser la herramienta *Cygwin*.

6.1.1 Cygwin

Cygwin es una herramienta que posibilita el uso de ejecutables creados para sistemas *UNIX*, como *Linux*, en sistemas *Windows*, y en este proyecto se emplea su terminal para ejecutar el simulador en *MAVProxy* (subsección 6.1.2).

Una forma eficiente de realizar la instalación de *Cygwin* es introduciendo la línea de código mostrada en la figura 6.1 en la ventana de comandos de *Windows*. Con esta instrucción se ejecuta el instalador del programa y se indican todos los paquetes adicionales que deben instalarse para realizar la simulación. Los paquetes que contienen la palabra *python* son necesarios para que *MAVProxy* pueda leer el SITL, ya que el programa emplea este lenguaje.

```
C:\Users\laura>C:\Users\laura\Downloads>setup-x86_64.exe -P autoconf,automake,ccache,
gcc-g++,git,libtool,make,gawk,libexpat-devel,libxml2-devel,python27,python27-future,
python27-libxml2,python27-pip,libxslt-devel,python27-devel,procps-ng,zip,gdb,ddd
```

Figura 6.1: Línea de código para la instalación de *Cygwin64*

Una vez terminada la instalación, se debe abrir y cerrar la terminal de *Cygwin64* recién instalada para generar y guardar archivos de inicialización, y a continuación se vuelve a abrir introduciendo la línea de código de la figura 6.2 para instalar paquetes adicionales relacionados con el lenguaje *python*.

```
pip2 install empy pyserial pymavlink
```

Figura 6.2: Línea de código para la instalación de paquetes de *python* en *Cygwin64*

El último paso antes de la simulación consiste en clonar el repositorio de *ArduPilot* en el directorio de *Cygwin* y actualizar sus submódulos, todo ello con las líneas de la figura 6.3.

```
git clone https://github.com/ardupilot/ardupilot.git
cd ardupilot
git submodule update --init --recursive
```

Figura 6.3: Líneas de código para la clonación del repositorio de *ArduPilot* en *Cygwin*

6.1.2 MAVProxy

MAVProxy es un paquete de software que forma una estación de control en tierra basada en el protocolo de comunicación con UAVs de tipo *MAVLink*, como es el que emplea el piloto automático de *ArduPilot* para comunicar a la aeronave con tierra o a diferentes componentes del mismo UAV entre sí. La aplicación de *MAVProxy* está basada en una consola con líneas de comandos sencilla y que ocupa poca memoria, por lo que puede ejecutarse en cualquier sistema operativo que cuente con *python* y *pyserial*, incluyendo tanto *Windows* como *Linux* y *OS X*. Además, cuenta con herramientas adicionales que facilitan la visualización de la simulación, de las cuales se emplean en este proyecto el mapa con vista satélite de la localización de la aeronave, que también se puede emplear para guiarla punto a punto, y la consola, que muestra el estado del UAV y mensajes generados a raíz de los comandos introducidos.

Una vez instalado *MAVProxy* desde el instalador proporcionado en la documentación de *ArduPilot*, la aplicación se ejecuta mediante la introducción de los comandos de la figura 6.4 en la terminal de *Cygwin*. La segunda línea ejecuta el archivo *sim_vehicle.py* en lenguaje *python* del SITL y abre las herramientas del mapa y la consola. Las tres ventanas que se abren son las mostradas en la figura 6.5.

```
cd ~/ardupilot/ArduPlane
../Tools/autotest/sim_vehicle.py --map --console
```

Figura 6.4: Líneas de código en *Cygwin* para la ejecución de la simulación

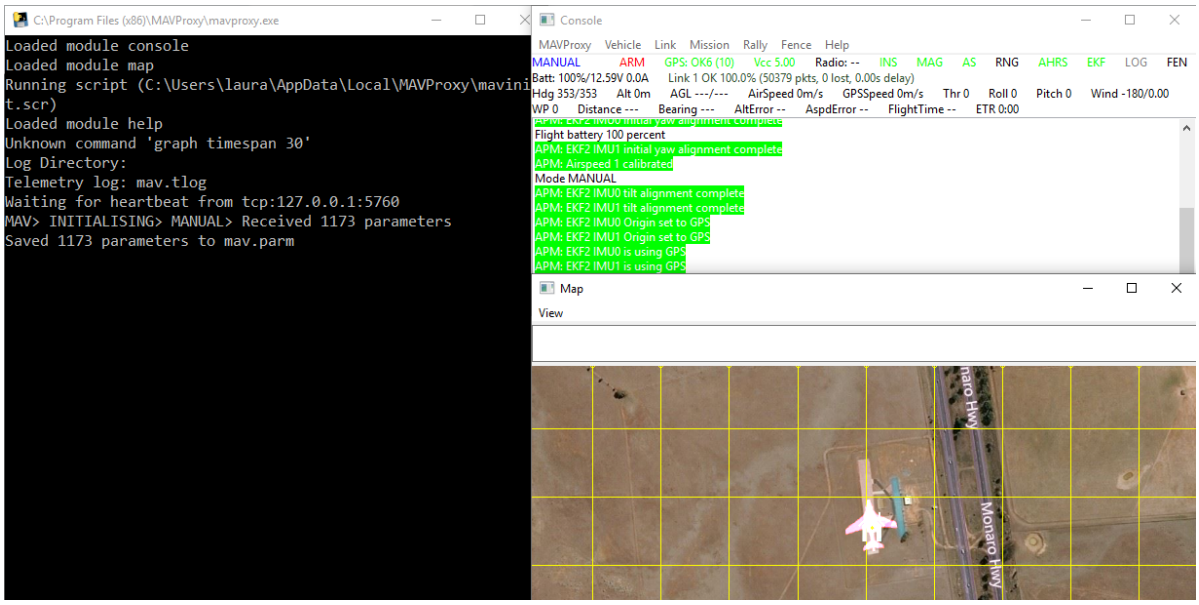

Figura 6.5: Ventanas de *MAVProxy*: ventana de comandos, consola y mapa

La misión formada por waypoints que seguirá la aeronave se introduce en la ventana de comandos con la instrucción *wp load*, que lee una serie de coordenadas y otras características deseadas durante el vuelo de un archivo tipo *.txt*, llamado en este caso *ListWP.txt* (figura 9.7 en el anexo). En esta lista la primera columna enumera las filas para poder hacer referencia a ellas en cualquier momento, teniendo en cuenta que el número 0 se asigna a la base de la aeronave; la segunda columna indica si cierto punto se debe seguir directamente si otros waypoints anteriores a este se encuentran más lejos, indicando el 1 esta prioridad y el 0 el seguimiento normal de los puntos; la tercera columna representa el sistema de coordenadas, que suele tener el valor de 3 para el sistema de coordenadas global y de 0 para el sistema de coordenadas local; y la cuarta columna indica el tipo de comando (16 es seguimiento de waypoint, 22 es despegue, 178 es cambio de velocidad y 177 indica a qué punto dirigirse para realizar un circuito de waypoints). Las siete columnas siguientes son una serie de parámetros que dependen del comando que se ha indicado en la cuarta columna, como el ángulo de cabeceo durante el despegue (comando 22), la velocidad que se desea alcanzar (comando 178), el radio de aceptación de waypoint (comando 16) o las coordenadas y altitud a seguir (comandos 16 y 22), y la última columna de todas indica si se continua empleando el modo automático o no.

Una vez se ha cargado el archivo que contiene los waypoints, la instrucción *wp list* envía esta información al documento *way.txt*, que es el que lee *MAVProxy* para realizar la simulación. Introduciendo los comandos *mode auto* y *arm throttle* la aeronave entra en modo automático y sigue las instrucciones de *way.txt*, incluyendo las de despegue y repetición de circuitos. En la figura 6.6 se ve claramente la sucesión de waypoints representados en el mapa, comenzando por el despegue desde la base, y su seguimiento. El siguiente paso consiste en transmitir los datos de *MAVProxy* a la estación de control en tierra *QGrounControl*.

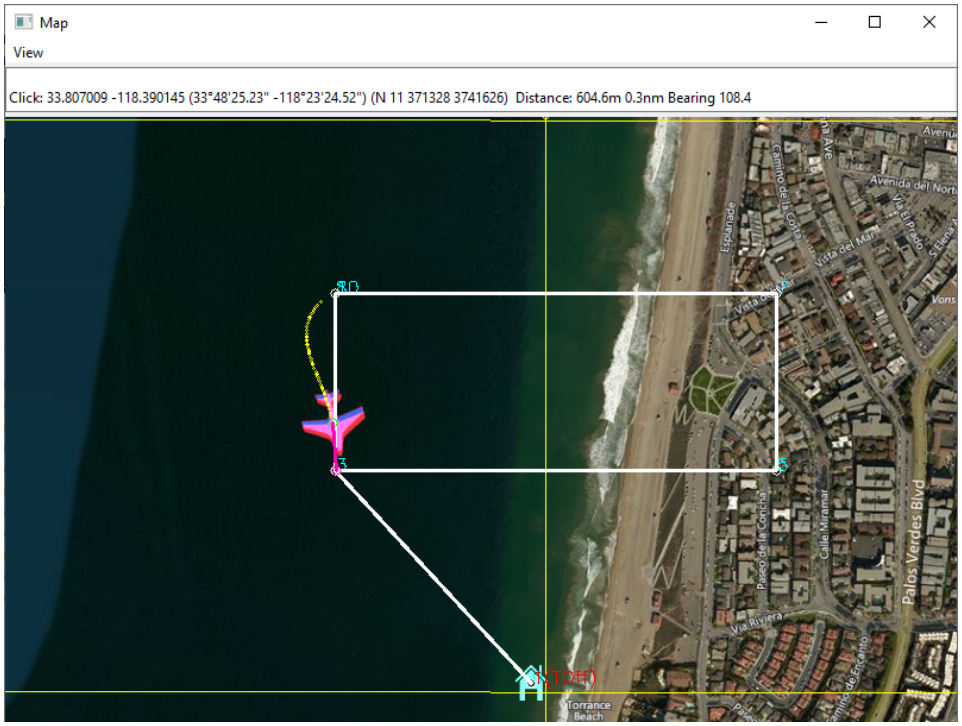

Figura 6.6: Seguimiento de waypoints en el mapa de *MAVProxy*

6.2 *QGroundControl*

6.2.1 QGroundControl

Como se ha comentado en el primer capítulo de este proyecto, la estación de control en tierra es de gran utilidad para visualizar el estado de la aeronave en todo momento y planear sus misiones de una forma más intuitiva que usando únicamente una consola. *MAVProxy* cumple con estas características, pero la estación de control *QGroundControl* cuenta con una interfaz gráfica más desarrollada y de uso generalizado en el mundo de los UAV que permite controlar a estos vehículos desde un ordenador, un teléfono móvil o una tableta. La opción adicional de simular un vuelo en tres dimensiones en *FlightGear* que ofrece *QGroundControl* hace que este programa sea incluso más apropiado para este proyecto.

MAVProxy transmite paquetes de información a *QGroundControl* a través de un puerto UDP 14550, como se puede ver en la figura 6.7, que muestra la estructura de comunicaciones que existe entre el piloto automático y las diferentes estaciones de control en tierra y plataformas de simulación. Para que la información circule por este puerto es necesario indicar en la ventana de comandos de *MAVProxy* el número del canal mediante la instrucción *output add 1:127.0.0.1:14550*. Una vez establecida la conexión y definida la lista de waypoints, ya es posible controlar la aeronave completamente desde *QGroundControl*.

En el mapa de *QGroundControl* aparecen representados los diferentes waypoints y las líneas que los unen, y en la misma pantalla se encuentran un indicador de actitud, una brújula y un cuadro con los valores de altitud, velocidad, duración del vuelo y otros parámetros seleccionables. Una vez todos los parámetros del piloto automático se han introducido en la pestaña de parámetros de *QGroundControl* (sección 5.3), un deslizable situado en la parte inferior de la pantalla activa el modo automático, que lleva a cabo el despegue y se encarga de que la aeronave siga todos los

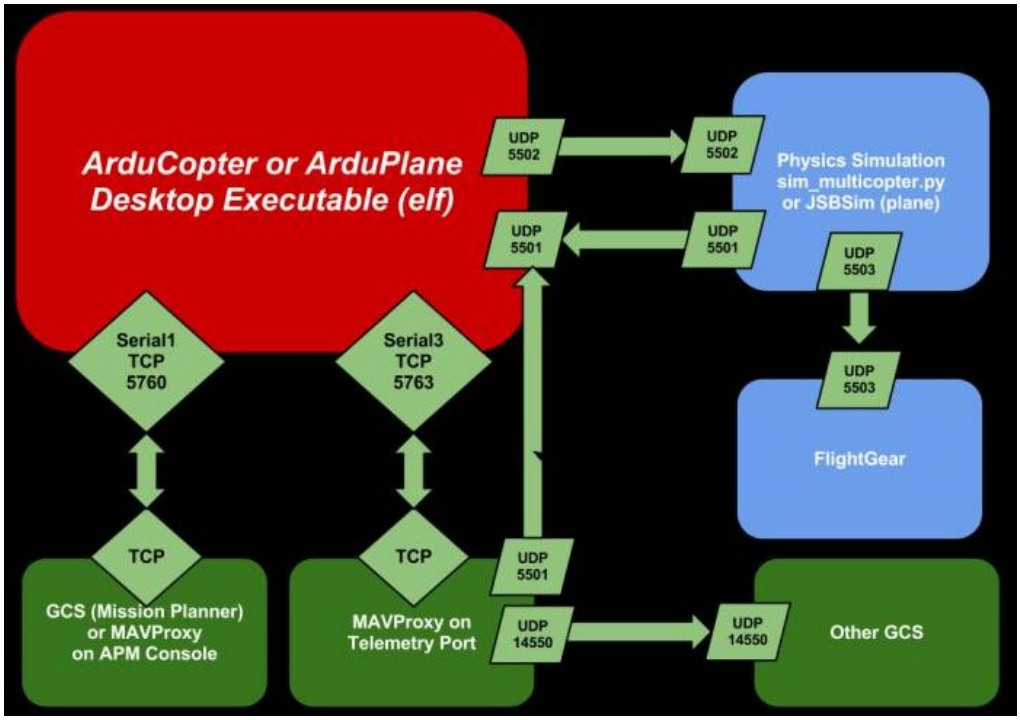

Figura 6.7: Arquitectura de comunicaciones en el *Software In The Loop* de *ArduPilot*

waypoints. Durante la simulación, todos los instrumentos de vuelo y los indicadores se comportan como lo harían en un vuelo real. La figura 6.8 muestra una captura de la ventana de *QGroundControl* durante la simulación.

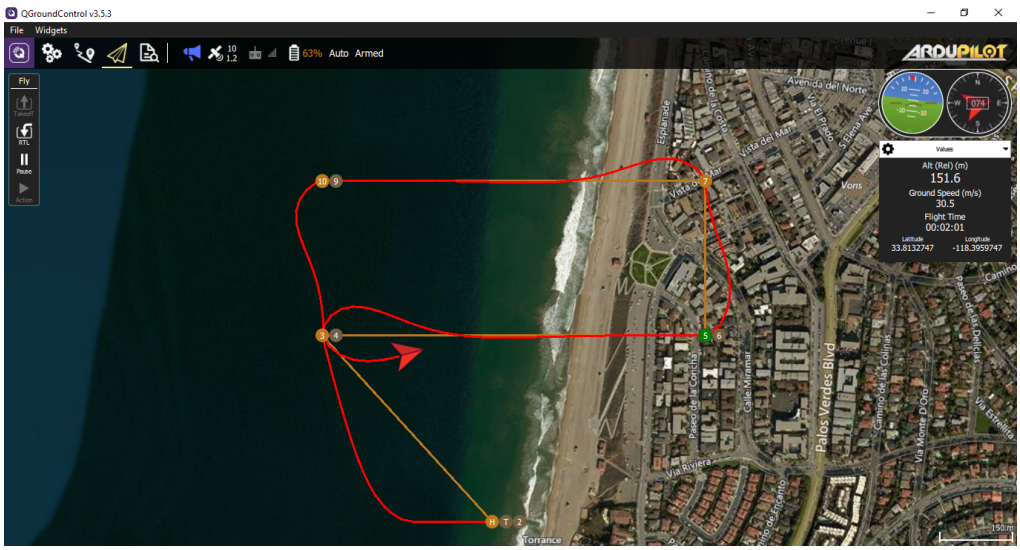

Figura 6.8: Seguimiento de waypoints en el mapa de *QGroundControl*

6.2.2 FlightGear

Por último, para comprobar el correcto funcionamiento del SITL y visualizar fuera de un mapa en 2D el vuelo, este se ha simulado en *FlightGear*, un software de simulación de vuelo de código libre y multiplataforma, donde se ha introducido un modelo tridimensional del HERMES-UPV diseñado por el grupo de trabajo [4].

La simulación en *FlightGear* se pone en marcha mediante la ventana *HIL Config* contenida en el desplegable de *Widgets* dentro de *QGroundControl*, en donde se selecciona la aeronave *Rascal110-JSBSim* (que es realmente la aeronave HERMES-UPV guardada en una carpeta con este nombre) y se introducen una serie de instrucciones, como el uso del puerto UDP 5303 para transmitir datos a *FlightGear*, tal como muestra la figura 6.9. La opción de *Hardware In The Loop* suele emplearse conectando la aeronave con todos sus sensores a la estación de control, ya que el HIL tiene el papel de simular la dinámica de la aeronave a partir de modelos matemáticos, pero al no contar con todos los componentes del UAV en la realidad, el SITL sustituye el hardware y los sensores reales.

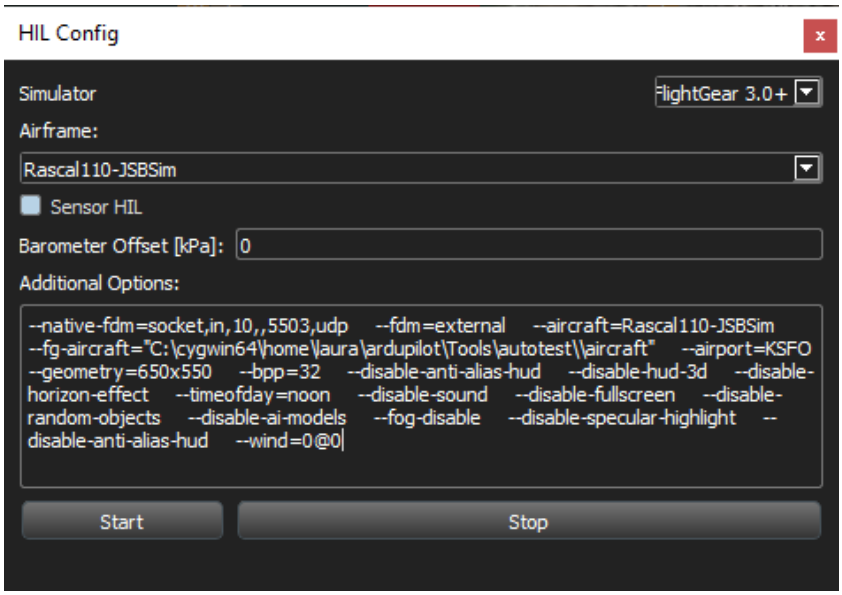

Figura 6.9: Configuración de la simulación en *FlightGear*

En la figura 6.10 se observa cómo la simulación en *FlightGear* tiene el mismo comportamiento durante los giros que la simulación en el mapa de *QGroundControl*.

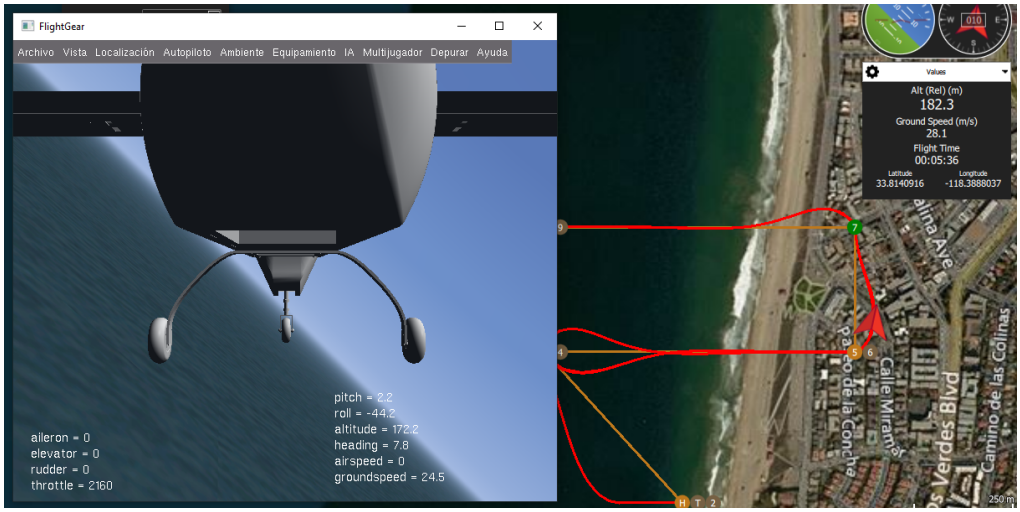

Figura 6.10: Visualización del mismo giro en *FlightGear* y en *QGroundControl*

Capítulo 7

Resultados de la simulación

La estación de control *QGroundControl* ofrece la opción de analizar información de la aeronave en tiempo real mediante la herramienta *Analyze* dentro del desplegable *Widgets*. Como se muestra en la figura 7.1, se puede realizar la representación gráfica de múltiples datos al mismo tiempo, y mediante el botón de *Start logging* todos estos valores se guardan en un documento de texto que sirve para representarlos posteriormente de forma individual o en grupos en *Matlab*.

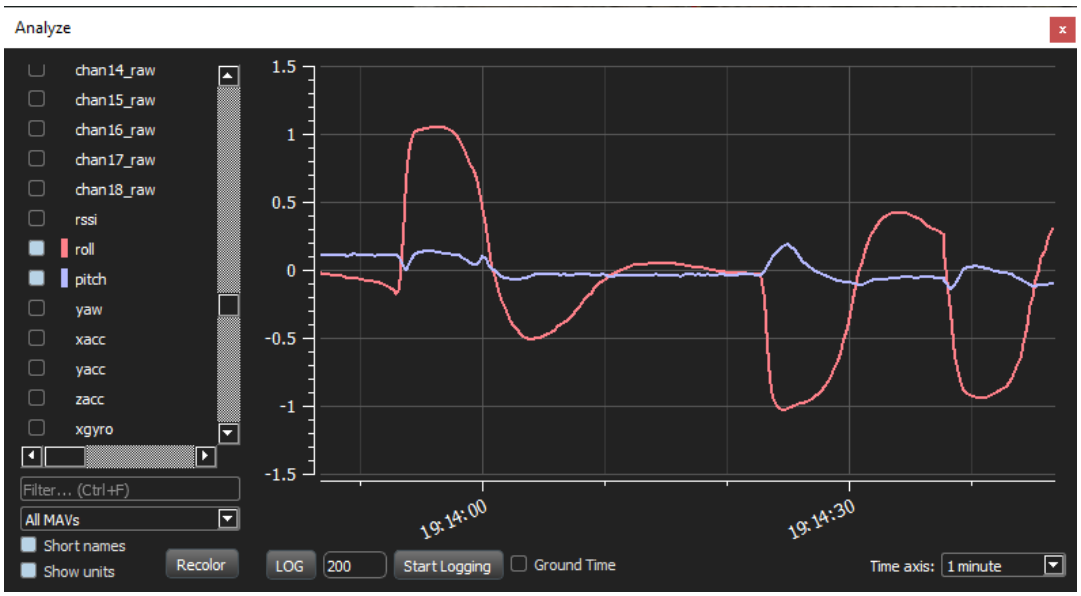

Figura 7.1: Herramienta *Analyze* en *QGroundControl*

Las figuras 7.2, 7.3, 7.4 y 7.5 muestran respectivamente las comparaciones entre el roll, pitch, aceleración lateral y velocidad de la simulación en *Simulink* frente a los resultados del SITL obtenidos mediante *QGroundControl*. El comportamiento de las variables es similar en los dos casos, especialmente en las gráficas del ángulo de alabeo y de la velocidad, aunque los resultados no son exactamente iguales debido a que la simulación SITL obtiene los datos de los sensores de una forma más elaborada que la simulación en *Simulink* y tiene en cuenta la presencia de ruido en las medidas.

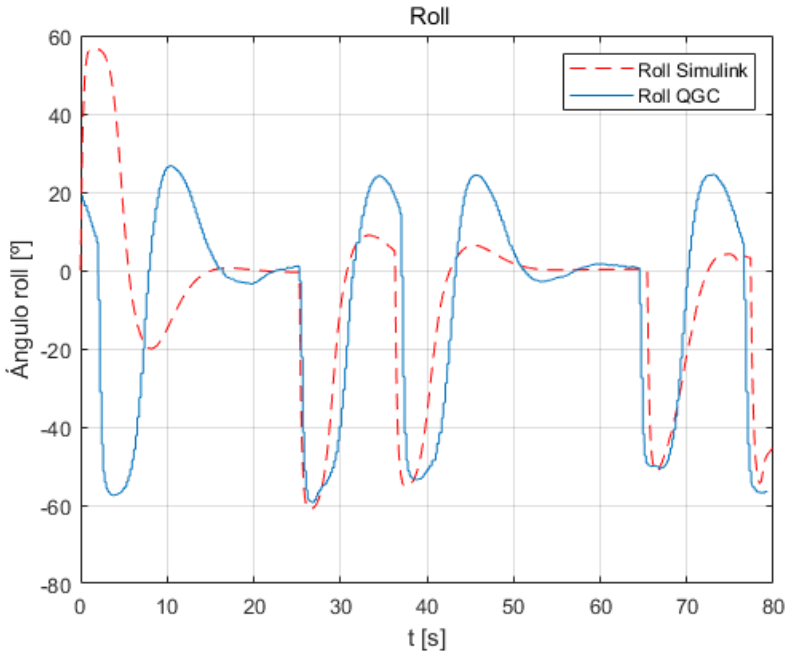

Figura 7.2: Comparación de roll obtenido en *Simulink* con el obtenido mediante SITL

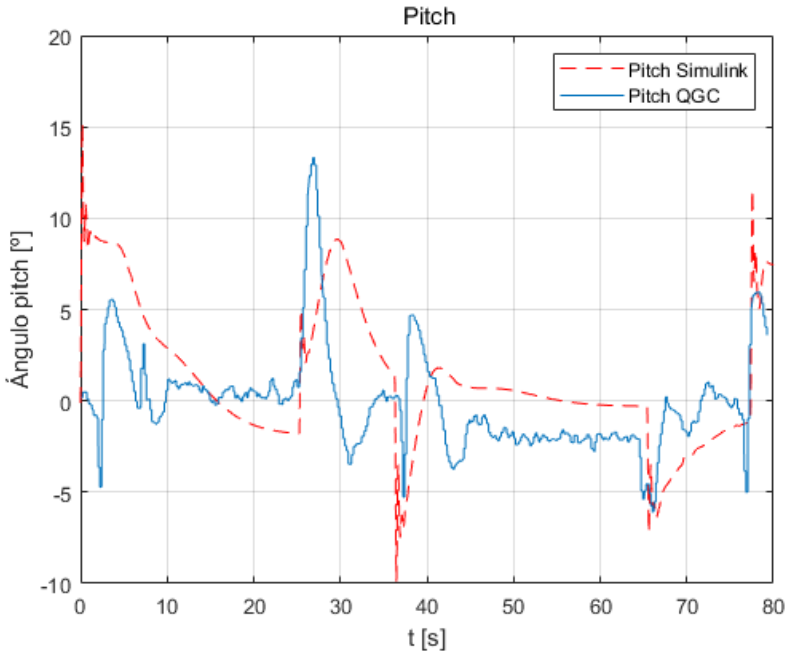

Figura 7.3: Comparación de pitch obtenido en *Simulink* con el obtenido mediante SITL

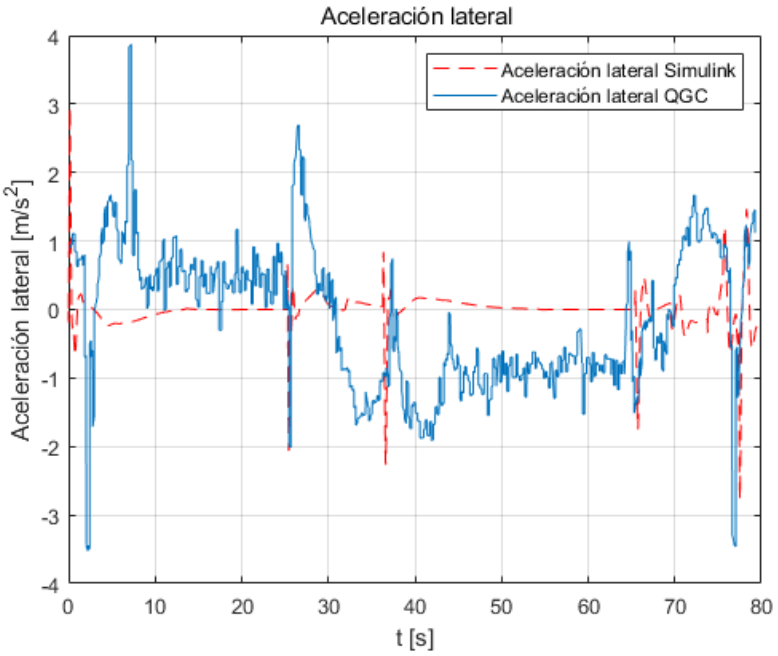

Figura 7.4: Comparación de aceleración lateral obtenida en *Simulink* con la obtenida mediante SITL

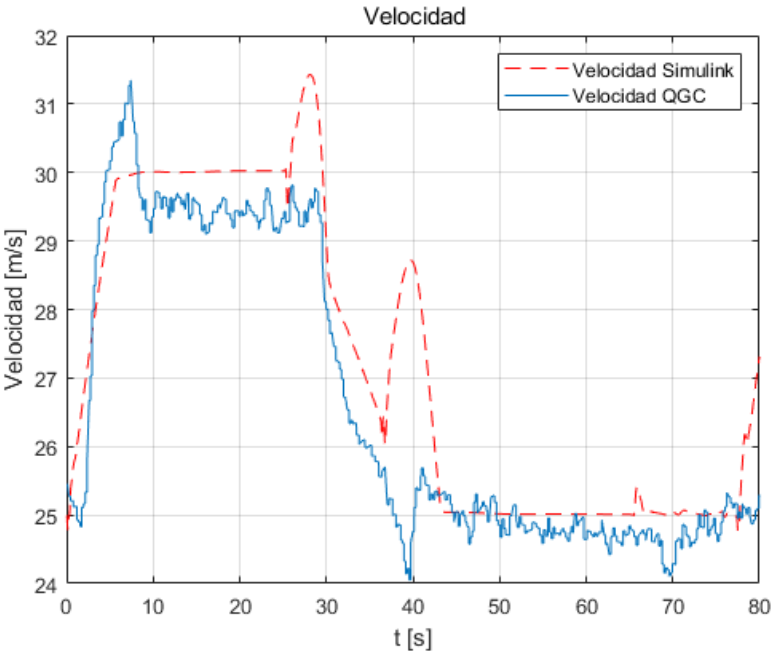

Figura 7.5: Comparación de velocidad lateral obtenida en *Simulink* con la obtenida mediante SITL

Dado que las variables del sistema tienen un comportamiento estable dentro de lo esperado y la aeronave cumple con la misión de seguimiento de waypoints exigida (la figura 6.8 del capítulo anterior muestra en un mapa el cumplimiento de la misión), se considera validado el sistema de control y navegación diseñado e implementado para la aeronave de HERMES-UPV.

Capítulo 8

Conclusiones y futuros trabajos

Mediante la realización de este trabajo se ha conseguido imitar el comportamiento de un piloto automático real en un ordenador mediante *Simulink* y posteriormente mediante *MAVProxy*, proceso que ha servido, no solo para obtener unos parámetros iniciales para la implementación del piloto automático en el ordenador de a bordo, sino también como herramienta para desarrollar nuevos pilotos automáticos dentro del proyecto HERMES-UPV que se adapten de la mejor manera a su misión.

Tras la validación de los sistemas de control y de navegación, la configuración del piloto automático obtenida podría emplearse directamente para realizar unas pruebas de vuelo iniciales, ya que, aunque la configuración de los parámetros seguramente no sea la definitiva, los controladores son capaces de estabilizar la aeronave. Por tanto, una manera de dar un paso más en el proceso de validación de los controladores sería evaluar la respuesta de la aeronave a estos durante un vuelo real. De todos modos, es importante tener presente que a lo largo del trabajo se han introducido varias simplificaciones y que el comportamiento real de una aeronave no tiene por qué corresponderse exactamente con una simulación, por lo que estudiar más a fondo el problema y realizar simulaciones más precisas puede ayudar a que el vuelo real sea más seguro.

A lo largo de este proyecto se ha seguido todo el proceso de diseño de los controladores de la aeronave y del sistema de navegación desde el principio, es decir, desde el modelado dinámico de la aeronave hasta la simulación de su comportamiento, por lo que cada una de las fases puede aprovecharse para crear un nuevo piloto automático, ya que la resolución de problemas relacionados con la interpretación del código o con las herramientas empleadas, por ejemplo, puede aplicarse a futuros proyectos. La comprensión completa del funcionamiento de un software de uso generalizado en el mundo de los UAV como es *ArduPilot* facilita la transición a pilotos automáticos más complejos y recientes, como el de código libre *PX4* que se pretende utilizar en el futuro en la aeronave.

Capítulo 9

Anexo

```
function gaDat=ga(g)
%
% Basic Genetic Algorithm
%
%    gaDat=ga(gaDat)
%    gaDat : Data structure used by the algorithm.
%
% Data structure:
% Parameters that have to be defined by user
% gaDat.FieldD=[lb; ub]; % lower (lb) and upper (up) bounds of the search space.
%                        % each dimension of the search space requires bounds
% gaDat.Objfun='costFunction'; % Name of the 0bjective function to be minimize
%
% Parameters that could be defined by user, in other case, there is a default value
% gaDat.MAXGEN={gaDat.NVAR*20+10}; % Number of generation, gaDat.NVAR*20+10 by default
% gaDat.NIND={gaDat.NVAR*50} ;   % Size of the population, gaDat.NVAR*50 by default
% gaDat.alfa={0};                % Parameter for linear crossover, 0 by default
% gaDat.Pc={0.9};                % Crossover probability, 0.9 by default
% gaDat.Pm={0.1};                % Mutation probability, 0.1 by default
% gaDat.ObjfunPar={[]};          % Additional parameters of the objective function
%                                %  have to be packed in a structure, empty by default
% gaDat.indini={[]};             % Initialized members of the initial population,↙
empty
%                                %  by default
%
% Grupo de Control Predictivo y Optimización - CPOH
% Universitat Politècnica de València.
% http://cpoh.upv.es
% (c) CPOH  1995 - 2012

if nargin==1
    gaDat=g;
else
    error('It is necessary to pass a data structure: gaDat.FieldD and gaDat.Objfun')
end
% If the parameter doesn't exist in the data structure it is created with the default↙
value
if ~isfield(gaDat,'NVAR')
    gaDat.NVAR=size(gaDat.FieldD,2);
end
if ~isfield(gaDat,'MAXGEN')
    gaDat.MAXGEN=gaDat.NVAR*20+10;
end
if ~isfield(gaDat,'NIND')
    gaDat.NIND=gaDat.NVAR*50;
end
if ~isfield(gaDat,'alfa')
    gaDat.alfa=0;
end
if ~isfield(gaDat,'Pc')
    gaDat.Pc=0.9;
end
if ~isfield(gaDat,'Pm')
    gaDat.Pm=0.1;
end
```

Figura 9.1: Función de matlab *ga.m* que contiene el algoritmo genético empleado en la optimización (I)

```
if ~isfield(gaDat,'ObjfunPar')
    gaDat.ObjfunPar=[];
end
if ~isfield(gaDat,'indini')
    gaDat.indini=[];
end

% Internal parameters
gaDat.Chrom=[];
gaDat.ObjV=[];
gaDat.xmin=[];
gaDat.fxmin=inf;
gaDat.xmingen=[];
gaDat.fxmingen=[];
gaDat.rf=(1:gaDat.NIND)';
gaDat.gen=0;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Main loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Generation counter
gen=0;

% Initial population      ---------------------------------------
gaDat.Chrom=crtrp(gaDat.NIND,gaDat.FieldD);   % Real codification
% Individuals of gaDat.indini are randomly added in the initial population
if not(isempty(gaDat.indini))
    nind0=size(gaDat.indini,1);
    posicion0=ceil(rand(1,nind0)*gaDat.NIND);
    gaDat.Chrom(posicion0,:)=gaDat.indini;
end

while (gaDat.gen<gaDat.MAXGEN),
    gaDat.gen=gen;
    gaDat=gaevolucion(gaDat);
    % Increase generation counter        ------------------
    gaDat.xmingen(gen+1,:)=gaDat.xmin;
    gaDat.fxmingen(gen+1,:)=gaDat.fxmin;
    gen=gen+1;
    datetime
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% End main loop
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Present final results
garesults(gaDat)
```

Figura 9.2: Función de matlab *ga.m* que contiene el algoritmo genético empleado en la optimización (II)

```
function [Nu, GS, vel_dem, zdiff, zdem, vel_rate_dem,cambio] = ...
NavL1Roll(NED, NED_Ini, NED_Fin, vel_NED, pitch, yaw, L1_damping, L1_period,↙
L1_i_gain, Nu_last, zdem_last, veldem_last)

groundspeed_vector=cos(pitch)*[vel_NED(1),vel_NED(2), 0];
groundSpeed=norm(groundspeed_vector);
GS=groundSpeed;

L1_dist = 1/pi * L1_damping * L1_period * groundSpeed;
dt=0.0005;
cambio=0;

A=[NED_Ini(1) NED_Ini(2) 0];
B=[NED_Fin(1) NED_Fin(2) 0];
C=[NED(1) NED(2) 0];

z=-NED(3);
vel=norm(vel_NED);

AB=(B-A);
AB=AB/norm(AB); %vector normalizado de la trayectoria A hasta B
AC=C-A;
CB=B-C;
BC=C-B;
WP_A_dist=norm(AC);

alongTrackDist = dot(AC, AB);
crosstrack_error=AC(1)*AB(2)-AC(2)*AB(1);

if WP_A_dist > L1_dist && alongTrackDist/max(WP_A_dist, 1.0) < cos(3*pi/4) %Región I
    AC_unit=AC./WP_A_dist;
    xtrackVel = groundspeed_vector(1)*(-AC_unit(2))-groundspeed_vector(2)*(-AC_unit↙
(1)); % Velocity across line
    ltrackVel = dot(groundspeed_vector, (-AC_unit)); % Velocity along line
    Nu = atan2(xtrackVel,ltrackVel);

    zdem=-NED_Ini(3);
    veldem=NED_Ini(4);

elseif alongTrackDist > norm(B-A) + groundSpeed*3  %Región II
    BC_unit=BC./norm(BC);
    xtrackVel = groundspeed_vector(1)*(-BC_unit(2))-groundspeed_vector(2)*(-BC_unit↙
(1)); % Velocity across line
    ltrackVel = dot(groundspeed_vector, (-BC_unit)); % Velocity along line
    Nu = atan2(xtrackVel,ltrackVel);

    zdem=-NED_Fin(3);
    veldem=NED_Fin(4);

else   %Región III
    xtrackVel = groundspeed_vector(1)*AB(2)-groundspeed_vector(2)*AB(1); % Velocity↙
across line
    ltrackVel = dot(groundspeed_vector, AB); % Velocity along line
    Nu2 = atan2(xtrackVel,ltrackVel);
```

Figura 9.3: Función de matlab *NavL1Roll* para el cálculo del ángulo de alabeo de referencia y de la velocidad de referencia (I)

```
    Nu1=asin(min(max(crosstrack_error/max(L1_dist,0.1),sin(-pi/4)),sin(pi/4)));
    if abs(Nu1)<deg2rad(5)
        L1_xtrack_i = min(max(Nu1 *L1_i_gain * dt,-0.1),0.1);

        Nu1=Nu1+L1_xtrack_i;
     end
    Nu=Nu1+Nu2;

    zdem=-NED_Fin(3);
    veldem=NED_Fin(4);
end

  target_bearing=pi/2+atan2(-CB(1),CB(2));
    if target_bearing<0
        target_bearing=target_bearing+2*pi;
    end
    if abs(Nu)>0.9*pi&&abs(Nu_last)>0.9*pi&&abs(target_bearing-yaw)>deg2rad(120)↙
&&Nu*Nu_last<0
        Nu=Nu_last;
    end
Nu=min(max(Nu,-pi/2),pi/2);

%Velocidad de referencia
CLMB_MAX=5; SINK_MIN=2; velRateMax=0.5*CLMB_MAX*9.81/vel; velRateMin=-0.5*SINK_MIN*9.↙
81/vel;
if veldem-veldem_last>velRateMax*dt
    vel_dem=veldem_last+velRateMax*dt;
    vel_rate_dem=velRateMax;
elseif veldem-veldem_last<velRateMin*dt
    vel_dem=veldem_last+velRateMin*dt;
    vel_rate_dem=velRateMin;
else
    vel_dem=veldem;
    vel_rate_dem=(vel_dem-veldem_last)/dt;
end

%Altitud de referencia
maxClimbRate=CLMB_MAX; minSinkRate=SINK_MIN;
if zdem-zdem_last>maxClimbRate*dt
    z_dem=zdem_last+maxClimbRate*dt;
elseif zdem-zdem_last<-minSinkRate*dt
    z_dem=zdem_last-minSinkRate*dt;
else
    z_dem=zdem;
end

zdiff=z_dem-z;
%Determina si se ha alcanzado el waypoint
if norm([NED_Fin(1)-NED(1) NED_Fin(2)-NED(2)]) <= 50
    cambio = 1;
else
    cambio = 0;
end

end
```

Figura 9.4: Función de matlab *NavL1Roll* para el cálculo del ángulo de alabeo de referencia y de la velocidad de referencia (II)

```
function [integSEB_input, integSEB_delta, integ_SEB_min, integ_SEB_max, temp, gainInv]
=NavL1Pitch(g,TIME_CONST, pitch_damp, i_gain, NED, vel_NED, z_dem, vel_dem,
vel_rate_dem, udot)

dt=0.1;
g=9.81;
TIME_CONST=3;
pitch_damp=0.4;
z=-NED(3);
vel=norm(vel_NED);
pitchmin=-deg2rad(15);
pitchmax=deg2rad(15);
climb=-vel_NED(3);

%Energías específica, potencial y cinética
SEB_dem=z_dem*g-vel_dem^2*0.5;
SPE_est=z*g;
SKE_est=vel^2*0.5;
SEB_error=SEB_dem-(SPE_est-SKE_est);
SPEdot=climb*g;
SKEdot=vel*udot;
SEB_dot_dem=(z_dem-z)/dt*g-vel*vel_rate_dem;
SEB_dot_error=SEB_dot_dem-(SPEdot-SKEdot);

%Límites aplicados tras la integración
gainInv = (vel * TIME_CONST * g);
temp = SEB_error + SEB_dot_dem * TIME_CONST + SEB_dot_error*pitch_damp;
pitchmin=-deg2rad(15);pitchmax=deg2rad(15);
integ_SEB_min=gainInv*(pitchmin-0.0783)-temp;
integ_SEB_max=gainInv*(pitchmax+0.0783)-temp;
integSEB_range=integ_SEB_max-integ_SEB_min;
integSEB_input=SEB_error*i_gain;
integSEB_delta=min(max(SEB_error*dt,-integSEB_range*0.1),integSEB_range*0.1);
end
```

Figura 9.5: Función de matlab *NavL1Pitch* para el cálculo del ángulo *Nu* de error en la trayectoria

```
function integSEB_state = integ(integSEB_state, integSEB_delta,integSEB_min,
integSEB_max)

integSEB_min=min(integSEB_state, integSEB_min);
integSEB_max=max(integSEB_state, integSEB_max);
integSEB_state=min(max(integSEB_state+integSEB_delta, integSEB_min), integSEB_max);
```

Figura 9.6: Función de matlab *integ* para el cálculo del ángulo de cabeceo de referencia

```
QGC WPL 110
0       0       0       16      0.000000        0.000000        0.000000        0.000000        -35.362938      149.165085      150.000000      1
1       0       3       22      15.000000       0.000000        0.000000        0.000000        -35.359833      149.164703      41.029999       1
2       0       3       178     0.000000        25.00000        0.000000        0.000000        0.000000        0.000000        0.000000        1
3       0       3       16      0.000000        0.000000        0.000000        0.000000        -35.359585      149.161392      150.000000      1
4       0       3       178     0.000000        30.00000        0.000000        0.000000        0.000000        0.000000        0.000000        1
5       1       3       16      0.000000        0.000000        0.000000        0.000000        -35.359585      149.169725      170.000000      1
6       0       3       178     0.000000        25.00000        0.000000        0.000000        0.000000        0.000000        0.000000        1
7       0       3       16      0.000000        0.000000        0.000000        0.000000        -35.356808      149.169725      180.000000      1
8       0       3       16      0.000000        0.000000        0.000000        0.000000        -35.356808      149.161392      160.000000      1
9       0       3       177     3.000000        -1.000000       0.000000        0.000000        0.000000        0.000000        0.000000        1
10      0       3       16      0.000000        0.000000        0.000000        0.000000        -35.356808      149.161392      160.000000      1
```

Figura 9.7: Archivo *ListWP.txt*

Bibliografía

[1] Institution of Mechanical Engineers. *Página web informativa sobre el UAS Challenge*. 2019. URL: https://www.imeche.org/events/challenges/uas-challenge (visitado 28-07-2019) (vid. pág. 3).

[2] Álvaro Goterris Fuster. *Desarrollo del Modelo Dinámico No Lineal de una Aeronave No Tripulada para el Proyecto HERMES-UPV*. Trabajo fin de grado. Valencia, 2019 (vid. págs. 4, 24).

[3] Daniel Villalibre Vilariño. *Diseño del Sistema de Control y Navegación para una Aeronave No Tripulada dentro del Proyecto HERMES-UPV*. Trabajo fin de grado. Valencia, 2019 (vid. págs. 4, 36).

[4] Víctor Ribera Esplugues. *Implementación de un Sistema de Simulación Hardware-In-The-Loop (HIL) para la Realización de Test No Destructivos de una Aeronave No Tripulada dentro del Proyecto HERMES-UPV*. Trabajo fin de grado. Valencia, 2019 (vid. págs. 4, 64).

[5] NATO STANAG. "4671 Unmanned Aerial Vehicles Systems Airworthiness Requirements (USAR)". En: *NSA/0976* (2009) (vid. pág. 4).

[6] Paul Fahlstrom y Thomas Gleason. *Introduction to UAV systems*. John Wiley & Sons, 2012 (vid. págs. 5, 8).

[7] Antonio Barrientos y col. "Vehículos aéreos no tripulados para uso civil. Tecnología y aplicaciones". En: *Universidad Politécnica de Madrid, Madrid* (2007) (vid. pág. 6).

[8] Gaurav Singhal, Babankumar Bansod y Lini Mathew. "Unmanned Aerial Vehicle Classification, Applications and Challenges: A Review". En: (2018) (vid. págs. 6, 7).

[9] Kimon P Valavanis y George J Vachtsevanos. *Handbook of unmanned aerial vehicles*. Springer, 2015 (vid. pág. 7).

[10] Gabriel Hugh Elkaim, Fidelis Adhika Pradipta Lie y Demoz Gebre-Egziabher. "Principles of guidance, navigation, and control of UAVs". En: *Handbook of Unmanned Aerial Vehicles* (2015), págs. 347-380 (vid. pág. 9).

[11] HaiYang Chao, YongCan Cao y YangQuan Chen. "Autopilots for small unmanned aerial vehicles: a survey". En: *International Journal of Control, Automation and Systems* 8.1 (2010), págs. 36-44 (vid. págs. 9-11).

[12] Adel Rawea y Shabana Urooj. "Design of fuzzy logic controller to drive autopilot altitude in landing phase". En: *Emerging ICT for Bridging the Future-Proceedings of the 49th Annual Convention of the Computer Society of India CSI Volume 2*. Springer. 2015, págs. 111-117 (vid. pág. 10).

[13] Grant E Hearn, Y Zhang y P Sen. "Alternative designs of neural network based autopilots: A comparative study". En: *IFAC Proceedings Volumes* 30.22 (1997), págs. 83-88 (vid. pág. 11).

[14] ASK Annamalai y A Motwani. "A comparison between LQG and MPC autopilots for inclusion in a navigation, guidance and control system". En: *Marine and Industrial Dynamic Analysis, School of Marine Science and Engineering, Plymouth University* (2013), págs. 1-19 (vid. pág. 11).

[15] BeagleBoard. *Página web de BeagleBone Blue*. 2018. URL: https://beagleboard.org/blue/ (visitado 28-07-2019) (vid. pág. 15).

[16] T-motor. *Página web de especificaciones del motor AT4130*. 2019. URL: http://store-en.tmotor.com/goods.php?id=828 (visitado 28-07-2019) (vid. pág. 24).

[17] ArduPilot Dev Team. *Documentación online de ArduPilot Plane*. 2019. URL: http://ardupilot.org/plane/ (visitado 28-07-2019) (vid. pág. 42).

[18] GitHub. *Repositorio en GitHub del firmware de ArduPilot*. 2019. URL: https://github.com/ArduPilot/ardupilot (visitado 28-07-2019) (vid. pág. 43).

[19] Sanghyuk Park, John Deyst y Jonathan How. "A new nonlinear guidance logic for trajectory tracking". En: *AIAA guidance, navigation, and control conference and exhibit*. 2004, pág. 4900 (vid. pág. 47).

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Trabajo Fin de Grado

Implementación y validación del sistema de control y navegación para una aeronave no tripulada dentro del proyecto HERMES-UPV

PLIEGO DE CONDICIONES

Autora: Laura Cristina Smith Ballester

Tutores: Francesc Xavier Blasco Ferragud
Sergio García-Nieto Rodríguez

Grado en Ingeniería Aeroespacial

Escuela Técnica Superior de Ingeniería del Diseño

Universitat Politècnica de València

Curso 2018/2019

Índice general

Capítulo 1

Introducción

Este documento contiene el pliego de condiciones correspondiente al trabajo de fin de grado con el título "Implementación y validación del sistema de control y navegación para una aeronave no tripulada dentro del proyecto HERMES-UPV".

El objetivo de la redacción de un pliego de condiciones en este proyecto es agrupar las condiciones técnicas y legales que deben cumplirse para desarrollar de forma correcta sus contenidos. Dado que en el desarrollo del trabajo se abordan especialmente aspectos relacionados con el uso de programas informáticos, este documento establece las condiciones de uso generales y particulares del software relacionado con este proyecto que se deben seguir.

Capítulo 2

Condiciones particulares y disposiciones técnicas

2.1 Condiciones particulares

La documentación generada en este proyecto estará disponible de modo que sirva de guía para el desarrollo de nuevos trabajos, mejoras en el código y resolución de posibles problemas. Los productos de la ejecución del presente trabajo son los siguientes:

- Modelo dinámico no lineal y linealizado mediante *Matlab* de la aeronave, implementado en la aplicación *Simulink*.
- Implementación en *Simulink* de los controladores PID y la ley de navegación inicial del piloto automático diseñado.
- Implementación en *Simulink* de los controladores específicos para aeronaves de ala fija incluidos dentro del piloto automático *ArduPilot* junto con la ley de navegación L_1 que se aplica en su software.
- Parámetros de configuración de *ArduPilot* para la simulación en *Software In The Loop* (SITL) y su método de obtención mediante un algoritmo genético.
- Programación de las interfaces empleadas para la simulación de vuelo en tiempo real, siendo estas *Cygwin*, *MAVProxy*, *QGroundControl* y *FlightGear*.

2.2 Disposiciones técnicas

2.2.1 ArduPilot

El código libre proporcionado por el proyecto *ArduPilot* se ha empleado a lo largo del proyecto y su uso se rige por la licencia *Created Commons Attribution-Share Alike 3.0 Unported* para la protección de los derechos de autor y la licencia GPLv3 (*General Public License version 3*) [1] específicamente adoptada por tratarse de software de código libre. Los aspectos más relevantes de las condiciones de uso del código y de la documentación se resumen a continuación:

- Se permite al usuario reproducir el contenido, incorporarlo en una o más recopilaciones de contenido, y reproducirlo una vez incluido en recopilaciones.
- Se permite crear y reproducir adaptaciones siempre que en dicha adaptación (incluyendo traducciones) se tomen acciones para identificar claramente las modificaciones realizadas sobre el contenido original. Estas adaptaciones pueden realizarse con cualquier propósito, incluido el comercial.
- Se permite la distribución y el uso público del contenido original y de sus adaptaciones, siempre que se incluya la referencia a esta licencia y se respete la autoría del contenido. No es necesaria la distribución del contenido original junto con el contenido adaptado.

2.2.2 Software empleado

En este trabajo se cumplen las condiciones de uso de los siguientes programas y herramientas informáticas empleados para la elaboración del proyecto:

- *Matlab*: se ha empleado la versión R2019a de *Matlab* con las herramientas *Simulink*, *pidTuner* y *Aerospace Toolbox* para realizar todos los cálculos numéricos, por lo que la versión empleada debe incluir estas herramientas. La licencia de pago es de uso individual y puede emplearse en un máximo de cuatro ordenadores.
- *Cygwin*: la licencia de uso de este programa, empleado para ejecutar el SITL, es de tipo GNU LGPLv3 (*GNU Lesser General Public License version 3*). Esta licencia permite que los usuarios y desarrolladores utilicen e integren el software en sus propios proyectos, con la condición de que esté disponible al público cualquier versión modificada del código inicial junto con la misma licencia LGPLv3.
- *MAVProxy*: este programa de código libre, relacionado directamente con los desarrolladores de *ArduPilot*, emplea las mismas licencias: *GNU Lesser General Public License version 3* para el programa y *CC Attribution-Share Alike 3.0 Unported* para la documentación.
- *QGroundControl*: el código de esta estación de control en tierra está protegido también por la licencia GPLv3 y adicionalmente por *Apache 2.0*, que requiere avisos de derechos de autor pero no necesita la redistribución del código fuente al crear modificaciones.
- *FlightGear*: dado que el código de este programa también es libre, la licencia que emplea es de nuevo la GPLv3.

[1] `http://ardupilot.org/dev/docs/license-gplv3.html`

- *Overleaf*: editor de texto online empleado para generar los documentos del trabajo. Las condiciones de uso se exponen en el apartado legal de su página web. [2]

- *Microsoft Windows 10 Home*: los términos de uso de la licencia del software de *Microsoft* se exponen en la página web `https://www.microsoft.com/en-us/Useterms/Retail/Windows/10/UseTerms_Retail_Windows_10_Spanish.htm`.

2.2.3 Bibliografía

A continuación se exponen las condiciones de uso de las fuentes bibliográficas empleadas en la elaboración del proyecto:

- La documentación de *ArduPilot* y *MAVProxy* es generada por la plataforma *GitHub*. Las términos de uso generales de los repositorios de *GitHub* y de la documentación que ofrece se encuentran en su página web (`https://help.github.com/en/articles/github-terms-of-service`), aunque cada documento puede estar sometido a condiciones de uso específicas.

- Las disposiciones legales de consulta de documentos en Internet o en libros son las especificadas en cada una de estas publicaciones, incluidas las consultadas mediante la *RedIRIS*, que es la red de recursos informáticos española a la que los miembros de la Universitat Politècnica de València tienen acceso.

[2] `https://www.overleaf.com/legal`

Capítulo 3

Condiciones de utilización

Al tratar el presente proyecto la implementación de un sistema de control y de navegación en una vehículo aéreo no tripulado, el usuario debe cumplir con las regulaciones existentes que afectan a estos vehículos en el país en el que se lleva a cabo el uso. Estas disposiciones se hallan en el caso de España en el Real Decreto 1036/2017, donde se regula la utilización civil de los UAV y se modifican el Real Decreto 552/2014 sobre el Reglamento del aire y disposiciones operativas comunes para los servicios y procedimientos de navegación aérea, y el Real Decreto 57/2002 sobre la aprobación del Reglamento de Circulación Aérea.

Por último, los documentos generados durante la realización del presente proyecto están regulados según las condiciones de uso y distribución establecidas en la plataforma *Riunet* de la Universitat Politècnica de València.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Trabajo Fin de Grado

Implementación y validación del sistema de control y navegación para una aeronave no tripulada dentro del proyecto HERMES-UPV

PRESUPUESTO

Autora: Laura Cristina Smith Ballester

Tutores: Francesc Xavier Blasco Ferragud
Sergio García-Nieto Rodríguez

Grado en Ingeniería Aeroespacial

Escuela Técnica Superior de Ingeniería del Diseño

Universitat Politècnica de València

Curso 2018/2019

Índice general

Índice de cuadros

Capítulo 1

Introducción

En este documento se presenta el presupuesto establecido para llevar a cabo el contenido del trabajo fin de grado titulado "Implementación y validación del sistema de control y navegación para una aeronave no tripulada dentro del proyecto HERMES-UPV".

En primer lugar, se muestra un desglose de los presupuestos parciales, dedicados esencialmente al software, material empleado, personal o licencias adquiridas, y en segundo lugar, se expone el presupuesto global del proyecto.

Capítulo 2

Presupuesto parcial

Para calcular el presupuesto parcial se tiene en cuenta que el proyecto se ha desarrollado a lo largo de tres meses completos en los que se ha trabajado una media de 5 horas diarias durante los días de la semana laborables. Por tanto, el número de horas orientativo necesario para desarrollar el proyecto que se va a emplear para desglosar los costes es de 300 horas.

2.1 Costes de material

Uno de los materiales necesarios para el desarrollo del proyecto es un ordenador capaz de ejecutar los programas necesarios. En este caso, el empleado es un ordenador portátil *Lenovo Ideapad 310-15IKB*, con procesador *i5-7200U Dual Core* de 2.5 GHz y 3.1GHz, 8GB de memoria RAM y 256GB de memoria en disco SSD, cuyo importe se muestra en la tabla 2.1. Para obtener el coste de este material se ha considerado un periodo de amortización del ordenador de 5 años, por lo que el coste tras los tres de meses de trabajo resulta de la división del coste inicial del portátil entre 20. En la tercera columna de la tabla se incluye un precio orientativo por hora obtenido dividiendo el periodo de 3 meses en las 300 horas de trabajo mencionadas anteriormente. En la tabla también se incluye el coste de la placa *BeagleBone Blue* en la que se implementa el piloto automático [1].

Concepto	Coste adquisición (€)	Coste horario (€/hora)	Coste total (€)
Lenovo Ideapad 310-15IKB	645.00	0.1075	**32.25**
BeagleBone Blue	86.73	0.2891	**86.73**
Coste total de material			**118.98**

Cuadro 2.1: Costes asociados al material empleado

[1] El presupuesto de esta placa se ha obtenido del vendedor de la siguiente página (en donde el IVA no se incluye en el precio): https://es.farnell.com/beagleboard/bbone-blue/beaglebone-blue-robotics-platform/dp/2612583

2.2 Costes de licencias de software

En la tabla 2.2 se especifica el coste de cada una de las licencias del software empleado durante el desarrollo de este trabajo. Como se puede observar, todas las licencias excepto dos son gratuitas gracias a la elección de programas de código libre, y por este motivo el tipo de licencia de estos programas es de duración perpetua.

Los costes de las licencias de pago de la tabla se han obtenido del mismo modo que en la sección anterior, calculando el coste horario tras 300 horas de trabajo y considerando un periodo de amortización de 5 años en el caso de las licencias de duración perpetua.

Concepto	Tipo de licencia	Coste (€)
Matlab	Perpetua (2000 €)	100.00
Simulink	Perpetua (3000 €)	150.00
Aerospace Toolbox	Perpetua (1150 €)	57.50
Control System Toolbox	Perpetua (1000 €)	50.00
Cygwin	Perpetua	0.00
MAVProxy	Perpetua	0.00
ArduPilot	Perpetua	0.00
FlightGear	Perpetua	0.00
QGroundControl	Perpetua	0.00
Microsoft Windows 10 Home	Anual (145 €)	4.97
Overleaf	Perpetua	0.00
Coste total de licencias		**362.47**

Cuadro 2.2: Costes asociados a licencias de software

2.3 Costes de personal

Los costes de personal se han calculado a partir de un sueldo de 19.70 €/hora de un ingeniero aeroespacial junior [2]. En la tabla 2.3 se relacionan las tareas principales del proyecto y su duración con el coste neto correspondiente.

Tarea	Duración (horas)	Coste (€)
Revisión bibliográfica	30	591.00
Diseño inicial de controladores y sistema de navegación	40	788.00
Diseño de controladores y sistema de navegación de *ArduPilot* en *Simulink*	85	1674.50
Simulación de vuelo y validación de la implementación	55	1083.50
Redacción del proyecto	90	1773.00
Coste total de personal		**5910.00**

Cuadro 2.3: Costes asociados al personal

[2] Salario anual de 23618.28 €. Resolución de 30 de diciembre de 2016, de la Dirección General de Empleo. *Boletín Oficial del Estado*, 18 de enero de 2017, núm. 15, pp. 4356-4382.

Capítulo 3

Presupuesto total

En la tabla 3.1 se resumen los costes totales brutos para la elaboración del proyecto, obtenidos mediante la suma de los costes de material, licencias de software y personal. También se incluye este coste tras la aplicación del beneficio industrial (BI) del 6 % y la adición de gastos generales (GG) del 13 %.

Concepto	Coste total (€)
Presupuesto material	118.98
Presupuesto licencias	362.47
Presupuesto personal	5910.00
Coste total bruto	**6391.45**
BI+GG (6 %+13 %)	1214.37
Coste total bruto + BI + GG	**7605.82**

Cuadro 3.1: Costes totales brutos

Para obtener el presupuesto neto final se debe añadir al coste bruto el IVA correspondiente del 21 %. Este presupuesto global se encuentra en la tabla 3.2.

Concepto	Coste (€)
Coste total bruto	7605.82
IVA (21 %)	1597.22
Presupuesto total neto	**9203.04**

Cuadro 3.2: Presupuesto total neto