Automatización de planes de misión para vehículos aéreos tripulados remotamente

Trabajo de Final de Grado

Ingeniería Aeroespacial

8 de septiembre de 2016

Asensio, Lorenzo Sempere

Tutor:
Juan, Vila Carbó

Cotutor:
Héctor, Usach Molina

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

On a given day, a given circumstance, you think you have a limit. And you then go for this limit and you touch this limit, and you think, 'Okay, this is the limit'. As soon as you touch this limit, something happens and you suddenly can go a little bit further.

Ayrton Senna

Agradecimientos

En primer lugar, agradecer a los profesores su magnífica labor, no solo a la hora de transmitir su conocimiento de la materia, sino por enseñarnos algo mucho más profundo que no se encuentra en los libros, y que desconocíamos cuando llegamos a aquí. En especial a mi tutor, Joan, por tenerme en cuenta desde el primer curso, por ofrecerme oportunidades para crecer y formarme a lo largo de la carrera, y por ayudarme a conseguir experiencia laboral.

Debo agradecer también a mis padres, Agustina y Asensio, por servirme de inspiración cada día; a mis hermanos, Agustín y María Jesús, por darme el cariño que solo te puede dar un hermano; y en general a toda mi familia porque son el entorno perfecto para crecer y desarrollarse como persona, por darme ánimos y apostar por mi.

Dar las gracias también a Pablo, Jorge, Anastasia, y a mi grupo de clase, por haber escuchado mis delirios y compartido los suyos, y por el intercambio de ánimos, apoyo, consejos y risas que han hecho de este viaje una experiencia inolvidable.

Y cómo no, a Adrián, mi fiel amigo. Porque aunque nuestros caminos se separaron hace unos años, siempre está ahí. Gracias también por ser siempre sincero conmigo, por recordarme mis defectos y ayudarme a pulirlos, y desde luego, por colaborar junto a mi padre en las revisiones de este proyecto.

Índice general

Índice de figuras

Índice de tablas

Resumen

El sector aeronáutico es un campo donde la innovación está a la orden del día. Sin embargo, tanto el periodo de diseño y validación, como el de amortización de las aeronaves provocan que su vida útil se extienda durante décadas. Esto requiere mantener la retrocompatibilidad para que las aeronaves que siguen en funcionamiento puedan compartir el espacio aéreo con los nuevos modelos.

Sin embargo, la aparición de vehículos RPAS en el ámbito civil y su rápido crecimiento, suponen un cambio en el paradigma de renovación debido a los cortos periodos de diseño y validación de estas aeronaves. Esto provoca que la frecuencia con la que las nuevas tecnologías se incorporan a los vehículos RPAS genere una brecha entre estas aeronaves y aquellas con las que debe compartir espacio aéreo.

Por esta razón, es necesario analizar los puntos débiles de este tipo de aeronaves, y extender los procedimientos actuales para que su funcionamiento sea tan rutinario como el del resto de aeronaves que vuelan en espacio aéreo integrado.

Con este trabajo, se pretende abordar varios de los problemas que plantea la introducción de vehículos RPAS en espacio aéreo controlado. En primer lugar, desde el punto de vista de la gestión del espacio aéreo, y en segundo lugar, respecto de la automatización y estandarización de la gestión de contingencias.

La solución elegida se basa en el desarrollo del concepto de Plan de Misión, similar al Plan de Vuelo, de tal manera que permita no solo especificar las intenciones de vuelo, sino también indicar otras rutas que la aeronave podría utilizar en caso de pérdida del enlace C2 u otras contingencias. Para completar este proyecto se diseña un sistema cuya finalidad es la de garantizar la seguridad de la aeronave durante la contingencia.

Objetivos

En este proyecto se pretende abarcar los siguientes objetivos:

- Investigar el funcionamiento del Autopiloto AP de una aeronave.
- Trabajar en el concepto "Plan de Misión" de manera que permita abarcar los misiones llevadas a cabo por Sistemas Aéreos Pilotados de forma Remota.
- Valorar las contingencias a las que tendrá que hacer frente un sistema Sistemas Aéreos Pilotados de forma Remota (RPAS), y su efecto nivel de Gestión del Tráfico Aéreo, en especial la pérdida del Enlace de Control y Comunicaciones.
- Estudiar la forma de proteger el sistema RPAS frente a estas contingencias.
- Implementar un sistema capaz de hacer frente a estas contingencias, en caso de pérdida del Enlace de Control y Comunicaciones.
- Validar dicho sistema bajo el software de simulación X-Plane

Índice de abreviaturas

ACAS Aircraft Collision Avoidance System. 16, 23, 42, IX

AFEP Automatic Flight Envelope Protection. 23, 25, 30, 32, 33, 44–47, 57, 61, 62, 88–93, VI, VIII–XI

AFGS Sistema de Guiado de Vuelo Automático. 10

AP Auto Pilot. 5, 8, 10, 12–14, 23, 25, 30, 43, V

ATC Controlador de Tráfico Aéreo. 4, 17, 18

ATM Gestión del Tráfico Aéreo. 18

ATS Servicios de Tráfico Aéreo. 1, 4

C2 Enlace de Control y Comunicaciones. 4, 12, 16–18, 23, 25, 27, 46, 50–52, 57, V

CFIT Controlled Flight Into Terrain. 14

DGAC Dirección General de Aviación Civil. 121

DISCA Departamento de Informática de Sistemas y Computadores Automáticos. 23

EPT Extended Path Terminator. 19, 22–25, 35, 37, 50, 51, 61, 62, 74, V–VII, IX, XI

FBW Fly-By-Wire. 14, 15, XI

FMC Flight Management Computer. 5, 9, 11, 42, IX

FMS Flight Management System. 9, 10, 23, 33, 38–42, V, VI, IX

GNSS Sistema Global de Navegación por Satélite. 23

IAC Instrumental Approach Chart. 6

IF Inital Fix. 21

LLA Latitud, Longitud y Altitud. 5, 49

LNAV Lateral Navigation. 25, 30, 35, 36, 43, 44, 81, 84, VII, VIII

MN milla náutica. 20, 26, 27, 67

MTOW Maximum Take-Off Weight. 67

NFZ No Flight Zones. 42

OACI Organización de Aviación Civil Internacional. 1, 17

PT Path Terminator. 6, 8, 9, 18, 19, 23, 35–37, V, VI, IX, XI

RA Resolution Advisory. 12, 15, 46, V

RNAV Navegacíon de Área. 4–6

RPA Aeronave Pilotada de forma Remota. 17, 20

RPAS Sistemas Aéreos Pilotados de forma Remota. 1, 2, 4, 10, 11, 14–19, 23, 47, 52, V

SESAR Single European Sky ATM Research. 15

SID Standard Instrumental Departure. 6

STAR Standard Arrival. 6

TAS True Air Speed. 55, X

TCAS Traffic Collision Avoidance System. 15, 16, IX

THR Throttle. 25, 30, 44, 82, 83, VII, VIII

VNAV Vertical Navigation. 25, 30, 35, 36, 43, 44, 82, 86, VII, VIII

Capítulo 1

Estudio del estado del arte

La automatización de Planes de Misión para RPAS representa un desafío tecnológico con varios frentes. Esto hace necesario introducir brevemente los siguientes temas por separado, dado que en su intersección se encuentra el foco de este trabajo.

1.1. Plan de vuelo

El Plan de vuelo es un documento de carácter aeronáutico, que se presenta previo a la realización de un vuelo y cuya finalidad es doble:

En primer lugar, servirá a los Servicios de Tráfico Aéreo (ATS) para autorizar, o no, dicho vuelo.
Y además, les ayudará a estar informados sobre los usuarios del espacio aéreo en cada momento, de modo que les sea posible realizar sus funciones a la hora de informar, apoyar y coordinar las operaciones de estos usuarios.

A su vez, es definido por la OACI [16] como:

> Información especificada que, respecto a un vuelo proyectado o a parte de un vuelo de una aeronave, se somete a las dependencias de los servicios de tránsito aéreo.

1.1.1. Contenido del Plan de Vuelo

El Plan de Vuelo debe incluir la información necesaria para que los servicios ATS tengan el mayor conocimiento del avión y de sus intenciones de vuelo, así como de la información básica respecto al mismo.

Solo de este modo es posible mantener el nivel de seguridad requerido por los estándares, y garantizar la viabilidad del vuelo.

Entre la información que debe incluir un plan de vuelo constan [17]:

- Identificación de la aeronave
- Reglas de vuelo y tipo de vuelo
- Número y tipos de aeronaves y categoría de estela turbulenta
- Equipo
- Aeródromo de salida
- Hora prevista de fuera calzos
- Velocidades de crucero
- Niveles de crucero
- Ruta que ha de seguirse
- Aeródromo de destino y duracíon total prevista
- Aeródromo de alternativa
- Autonomía
- Número total de personas a bordo
- Equipo de emergencia y de supervivencia
- Otros datos

1.1.2. Limitaciones del Plan de Vuelo

Al tratarse de un documento orientado principalmente al ámbito de la Aviación Civil, se centra en los vuelos cuya finalidad es la de desplazarse de un sitio a otro, normalmente con pasajeros a bordo.

El origen de este documento tiene lugar en un periodo en el que la aviación no tripulada no representaba un problema real, y los objetivos de cualquier vuelo abarcaban un rango de distancias inferior al actual. Sin embargo, la situación actual describe un panorama muy distinto.

El desarrollo de nuevos tipos de aeronaves y la expansión del uso de los RPAS, junto con nuevas funcionalidades que se implementan generación tras generación, ofrecen la posibilidad de emplear estas aeronaves para un rango más amplio de propósitos, lo que supone dos importantes limitaciones con respecto al Plan de Vuelo.

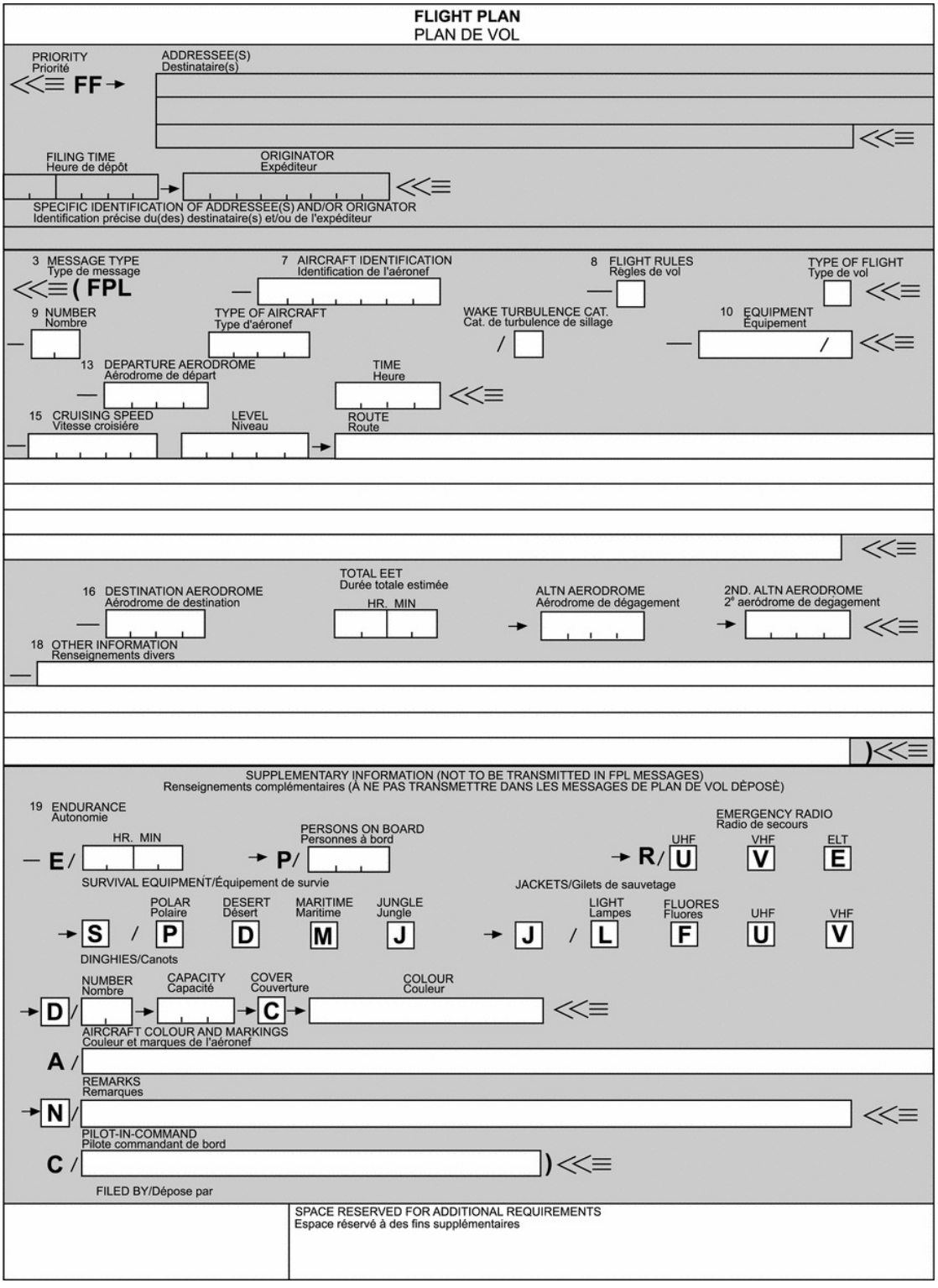

Figura 1.1: Modelo de Plan de Vuelo - OACI [1]

1. **Rigidez del formato:** Mientras que el formato del Plan de Vuelo permite a los proveedores de servicios ATS un rápido y sencillo acceso a la información relativa a la aeronave y al vuelo, las capacidades de las aeronaves actuales – entre ellas, especialmente las de los RPAS – pone en evidencia que el formato de plan de vuelo convencional (figura 1.1) no es capaz de abarcar en su totalidad las misiones que una aeronave puede realizar en espacio aéreo controlado, ya que estas van más allá de desplazar la aeronave y su carga de pago de un lugar a otro, al involucrar maniobras que no están definidos en el contexto actual.

2. **Resolución de contingencias:** El modelo actual está basado en que el piloto se encuentra dentro de la aeronave, y ante un posible conflicto, debe ser capaz de resolver la situación con la ayuda del Controlador de Tráfico Aéreo (ATC). Sin embargo, la integración de los RPAS en el espacio aéreo controlado implica eliminar este pilar, asumiendo que el piloto puede no estar dentro de la aeronave.
   La mayoría de incidencias en vuelo serán resueltas de forma similar tanto por el piloto que se encuentra dentro de la aeronave como por el operador remoto, siempre que exista el doble enlace (figura 1.2).

Éste es el principal problema añadido, dado que si en algún momento se pierde el Enlace de Control y Comunicaciones (C2), la aeronave tendrá que funcionar de forma autónoma y sin contacto con el operador remoto, siguiendo las leyes con las que fue programado hasta recuperar las comunicaciones o terminar el vuelo. Por tanto, **para mantener la predecibilidad de las aeronaves RPAS es necesario documentar una serie de procedimientos de contingencia** que contemplen, al menos, el procedimiento de actuación ante ciertas contingencias.

1.2. Estándar ARINC-424

1.2.1. Definición de Planes de vuelo y PBN

Desde la implantación de la Navegacíon de Área (RNAV) a nivel europeo en 1998 [18], todas la aeronaves deben contar con el equipamiento mínimo necesario para seguir el estándar RNAV-B (Básico). Esto permite que las aeronaves puedan generar radioayudas virtuales que les permitan describir cualquier ruta a partir de la información recibida de las radioayudas reales (VOR, NDB y GPS, entre otros) mediante técnicas de post-procesado. [4].

Este avance tecnológico, hace posible la creación de Planes de Vuelo basados en Waypoints (o Puntos de Control) en cuyas coordenadas no se encuentra física-

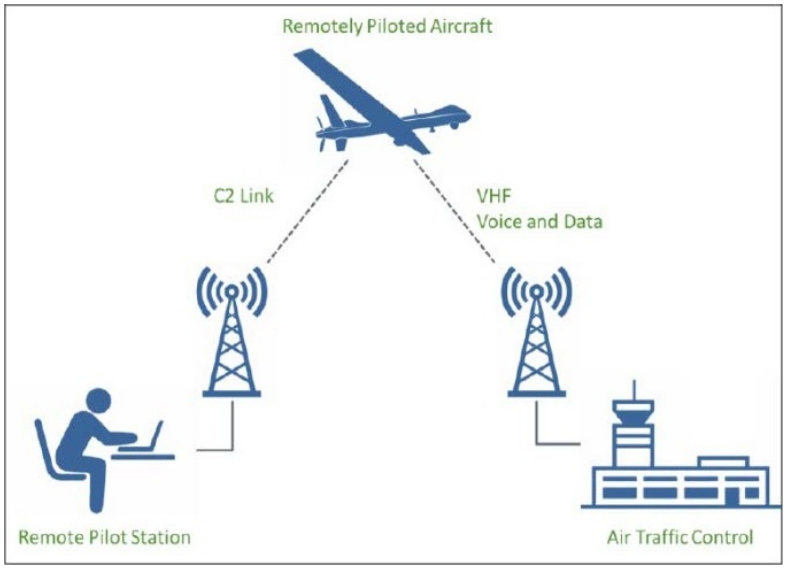

Figura 1.2: Doble enlace: Piloto Remoto - Aeronave - ATC [2]

mente la radioayuda. A estos procedimientos se los conoce como **procedimientos RNAV**, están definidos por coordenadas en formato Latitud, Longitud y Altitud (LLA), y permiten describir rutas en las tres dimensiones del espacio.

1.2.2. Bases de Datos de Aeronavegación - ARINC-424

Para mantener el sistema RNAV y poder construir los Planes de Vuelo, es necesario que algunos puntos se encuentren ya definidos en los sistemas aeronáuticos. Concretamente esta información se almacena en el Flight Management Computer (FMC) de la aeronave, y se actualiza periódicamente siguiendo el Modelo de Propagación de la Información de Aeronavegación (figura 1.4).

A través del FMC, el piloto seleciona los Waypoints que se indican en el Plan de Vuelo aprobado para indicar al Auto Pilot (AP) que deberá volar dicha ruta. [4]

Las Bases de Datos de Aeronavegación se crean siguiendo el formato establecido por el estándar ARINC-424, almacenan la formación relativa a los puntos de control aeronáuticos – tales como aeropuertos, helipuertos, pistas de aterrizaje, radioayudas, etc – en un formato de 132 bytes de longitud para cada radioayuda (figura 1.5).[19].

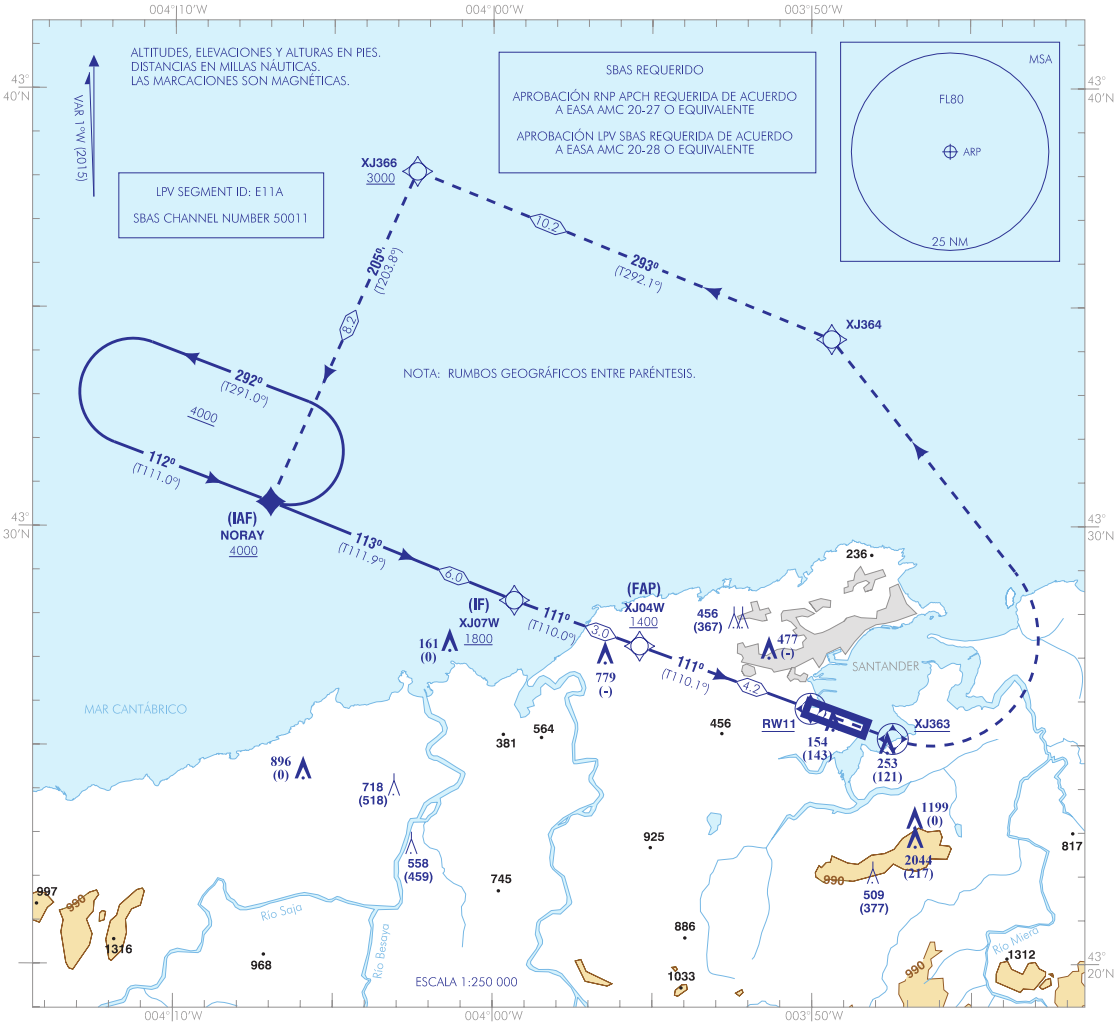

Figura 1.3: Carta de aproximación IAC RNAV Z - RWY 11 - Santander [3]

1.2.3. Path Terminators

Los Path Terminators son un concepto reciente propio de la navegación RNAV. Se definen como [20]:

> Un medio para permitir la codificación de los Procedimientos de Área Terminal, SID, STAR y de Aproximación.

A través de los Path Terminators (PTs) es posible describir un procedimiento – como un procedimiento Standard Arrival (STAR), Standard Instrumental Departure (SID) o Instrumental Approach Chart (IAC), como el de la figura 1.3 –

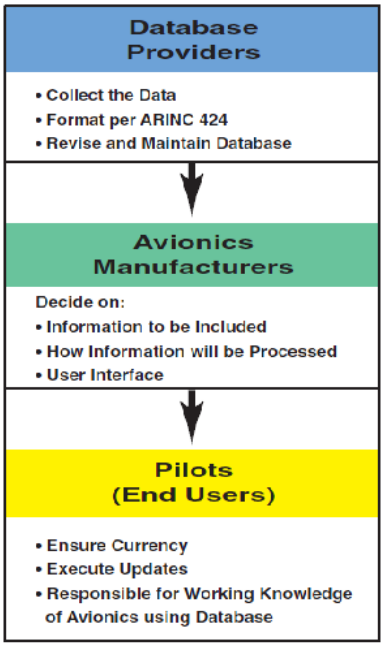

Figura 1.4: Propagación de la Información de Aeronavegación [4]

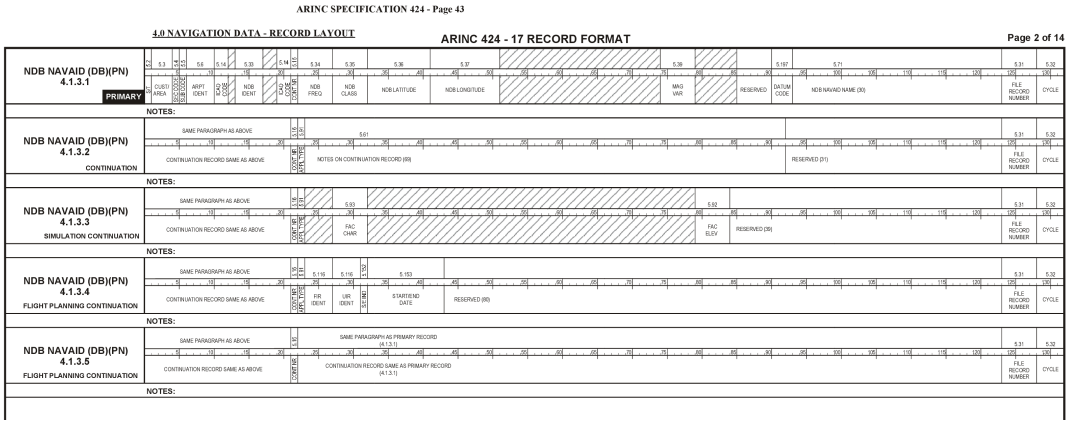

Figura 1.5: Estándar ARINC-424. Definición de un NDB [5]

mediante una **secuencia de procedimientos más sencillos** y estandarizados que indiquen cómo ir de un Waypoint a otro. [21]

Los Path Terminator más importantes, se describen a continuación:

- **IF**: Initial Fix
  Indica el Waypoint inicial de la ruta RNAV, por tanto, no representa la transición de un Waypoint a otro.

- **TF**: Track to Fix
  Describe un desplazamiento ortodrómico desde el Waypoint inicial – que será el punto final del PT anterior – hacia el Waypoint final. Supone un desplazamiento predecible.

- **DF**: Direct to Fix
  Representa el desplazamiento desde una posición inicial desconocida hacia el Waypoint final definido. Se puede utilizar para reducir el impacto ambiental de las fases de vuelo a baja altura.

- **CF**: Course to Fix
  Define una trayectoria tal que al alcanzar el Waypoint final se siga un rumbo concreto. Empleado mayormente en la fase de aproximación.

- **RF**: Radious to Fix
  Describe un giro con radio constante centrado en el Waypoint.

Los demás Path Terminator se pueden encontrar en la figura 1.1.

1.3. Vuelo automatizado

1.3.1. Piloto Automático (AP)

El Piloto Automático (AP) es un sistema de control de vuelo básico que permite mantener ciertos parámetros de vuelo constantes a lo largo del tiempo – representa el funcionamiento estático.

Representa la capa más baja a nivel de automatización, y permite mantener la aeronave en vuelo estable sin requerir una atención constante por parte del piloto, quien solo deberá indicar la velocidad y rumbo de vuelo deseada así como la velocidad vertical, dentro de los límites de la aeronave. Es decir, tiene la capacidad de mantener la aeronave en **estado estacionario**.

Path terminator	*Waypoint identifier*	*Flyover*	*Turn direction*	*Recommended Navaid*	*Distance from Navaid*	*Bearing from Navaid*	*Magnetic course*	*Path length*	*Altitude restriction 1*	*Altitude restriction 2*	*Speed limit*	*Vertical angle*	*Arc centre*
CA			O				✓		6		O		
CF	✓	1	O	✓	✓	✓	✓		O	O	O	O	
DF	✓	1	O	O	O	O			O	O	O		
FA	✓		O	✓	✓	✓	✓		6		O		
FM	✓		O	✓	✓	✓	✓		O		O		
HM	✓		O	O	O	O	✓	✓	O		O		
IF	✓			O	O	O			O	O	O		
RF	✓	O	✓	O		2	3	5	O	O	O	O	✓
TF	✓	O	O	O	O	O	O	O	O	O	O	O	
VA			O				4		6		O		
VI		O	O	O			4		O	O	O		
VM	O		O				4		O		O		

✓ — Required
O — Optional
1 — Required for CF/DF and DF/DF combinations only.
2 — Inbound tangential track
3 — Outbound tangential track
4 — Heading not course
5 — Along track distance
6 — Altitude at or above

Shaded spaces represent data that are not applicable to that path terminator.

Tabla 1.1: Tabla de los PT estándar y los datos requeridos [12]

1.3.2. Flight Management System

El Flight Management System (FMS) es un sistema embarcado de mayor nivel, y supone una segunda capa de automatización del vuelo, encargándose de gestionar el vuelo a lo largo de su recorrido – representa el funcionamiento dinámico.

A diferencia del Piloto Automático, que mantiene la actitud de la aeronave constante, el FMS recibe información de posición de la aeronave y actúa sobre el Piloto Automático para llevar a cabo la ruta programada. Es decir, lleva a cabo el **guiado de la aeronave a lo largo del tiempo**.

Sus funciones incluyen además de la ejecución de la mayor parte del vuelo, la programación de la ruta antes y durante el vuelo, la monitorización de las radioayudas y posición del avión, y el ajuste de los parámetros de la aeronave para maximizar la eficiencia.[22]

Para lograrlo, se compone de cuatro elementos:

- El Flight Management Computer, que es el ordenador en el que el piloto introduce la información del vuelo, y el centro de procesamiento del sistema (ver figura 1.7).

- Los sistemas de navegación, que se encargan de obtener la información de

Figura 1.6: Instrumentos de navegación básicos [6]

posición y actitud del avión a través de los sensores y radioayudas.

- La instrumentación de la aeronave, compuesta por los distintos indicadores del panel de control de los pilotos (ver figura 1.6)

- El Sistema de Guiado de Vuelo Automático, la parte del sistema que recibe y procesa la información de los otros componentes del sistema y aplica las leyes de control de forma similar al AP.

El uso del FMS es de especial utilidad en rutas largas ya que permite una reducción considerable de la carga de trabajo que debe asumir el piloto durante el vuelo. Esto permite que el piloto se concentre en supervisar el buen funcionamiento de la aeronave, evitando a su vez posibles errores propios del pilotaje manual.

1.3.3. Sistemas Aéreos Pilotados de forma Remota

El estado del arte del vuelo automatizado se materializa actualmente en los RPAS. Estas aeronaves, que pueden adquirir diversos aspectos y emplear diversas técnicas para el vuelo aprovechan la tecnología de los microprocesadores y la alta potencia de cómputo de los dispositivos actuales para realizar tareas que, hasta hace unos años, solo estaban al alcance de los pilotos.

La variabilidad de los formatos en los que se encuentran estas aeronaves, especialmente el tamaño, el peso, la autonomía, la eficiencia y su coste por hora de vuelo, así como las capacidades de las que se pueden dotar gracias al desarrollo de software – tales como reconocimiento del terreno, persecución de objetivos y vigilancia, entre otros – las sitúan en una posición muy aventajada respecto de las aeronaves convencionales.

Figura 1.7: FMC de un Boeing 737-300 [7]

Esta situación se evidencia en el hecho de que, aunque actualmente no todos los RPAS garantizan los mínimos de seguridad para aviación, ya hay numerosas empresas especializadas en el empleo de los mismos para realizar estas tareas debido a sus importantes ventas. Esto representa un cambio en el uso de las aeronaves y muestra que, en la actualidad, el principal propósito de la aviación ya no es el de transportar pasajeros y carga de un lugar a otro, sino que también se realizan todo tipo de misiones en el aire.

1.4. Contingencias

Durante el vuelo normal pueden tener lugar ciertos escenarios de fallo previsibles, a los que se conoce como contingencias. Estas situaciones involucran una serie de consecuencias que el piloto no debe ignorar, ya que a menudo son situaciones de emergencia que pueden derivar en accidentes de distinta magnitud.

En el ámbito de automatización del vuelo en sistemas RPAS, será de utilidad agruparlos en las siguientes categorías.

- Desconexión del autopiloto,

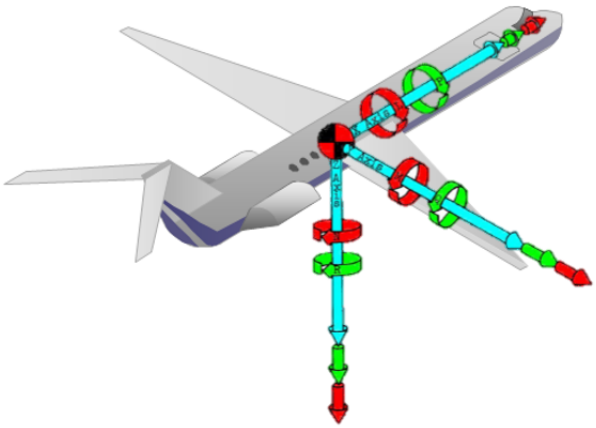

Figura 1.8: Posición de los ejes X, Y y Z en la aeronave

- Fallos de posicionamiento,
- Alerta de Tráfico (Resolution Advisory (RA)),
- Otras contingencias, y
- **Pérdida del Enlace C2**

1.4.1. Desconexión del autopiloto

Los sistemas AP convencionales están diseñados para garantizar un mínimo de seguridad operacional, por tanto, cuando no se encuentran dentro del límites de funcionamiento para los que fueron diseñados, deben desconectarse e informar al piloto para que tome el control de la aeronave.

Las **situaciones que provocan la desconexión del AP** se pueden agrupar, a su vez, en aquellas relacionadas con el movimiento longitudinal de la aeronave – desplazamiento, velocidades y aceleraciones en los ejes X y Z, y sus homólogos angulares respecto del eje Y –, y aquellas relacionadas con el movimiento lateral – desplazamiento, velocidad y aceleración en el eje Y, y sus homólogos angulares respecto de los ejes X y Z – como muestra la figura 1.8.

Según el **Movimiento Longitudinal**, la desconexión del AP se puede deber a:

1. **Velocidad y/o altura fuera de la envolvente de vuelo**:
   Bajo ciertas circunstancias, como el vuelo a gran altitud o los descensos a gran velocidad pueden alcanzarse situaciones en las que la aeronave saldría

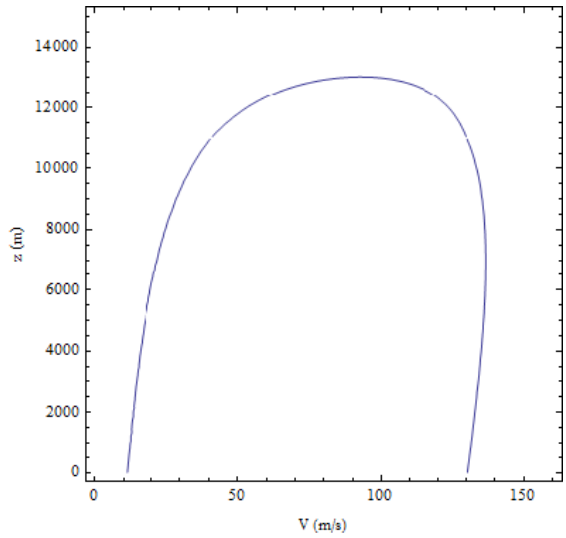

Figura 1.9: Ejemplo de envolvente de vuelo - Calculada con Mathematica

de los límites de su envolvente de vuelo (ver figura 1.9). Estas situaciones ponen en riesgo tanto la estructura como la estabilidad de la aeronave.

2. **Ángulo de ataque elevado**:
   Cuando el ángulo de ataque es demasiado elevado – ya sea positivo o negativo –, y ya que por diseño la mayoría de aeronaves no pueden sustentar todo su peso con solo el empuje motor, lo normal es que la velocidad se reduzca considerablemente por al exceso de resistencia aerodinámica. Esto puede hacer que la aeronave entre en pérdida, comprometiendo de forma similar la estructura y el control de la aeronave.

3. **Factor de carga elevado**:
   En situaciones en las que la velocidad vertical cambia rápidamente, la aeronave se ve expuesta a una aceleración, a, elevada. A esta aceleración, adimensionalizada respecto de la aceleración de la gravedad terrestre, g, se la conoce como factor de carga, n, siendo $n = a/g$. Al tratarse de aceleraciones respecto del eje z, el valor de n_z se puede obtener según la siguiente fórmula:

$$n_z = \cos\theta\cos\phi - \frac{1}{g}\left(\alpha\frac{dV}{dt} + V\left(\frac{d\alpha}{dt} - q + \alpha p\right)\right) \tag{1.1}$$

4. **Baja energía y** α_{suelo}:
   Este caso no supone una desconexión del AP, sino de una alerta, de modo que en caso de no seguir las indicaciones, la aeronave actúa de forma automática aumentando el empuje al máximo.

Esta alerta tiene lugar cuando la energía mecánica (cinética más potencial) son demasiado bajas para la fase de vuelo en la que se encuentra la aeronave, si el piloto no actúa para corregirlo, llegado a un cierto punto, el sistema hace funcionar los motores a máximo empuje a fin de evitar chocar contra el terreno – también conocido como evento Controlled Flight Into Terrain (CFIT), una de las causas más habituales de accidente actualmente.

En el otro grupo, según el **Movimiento Lateral**, la desconexión del AP puede deberse a:

1. **Velocidad de alabeo y guiñada elevadas**:
   La forma de evitar que la aeronave entre en pérdida en el plano lateral, desde el punto de vista de los Ángulos de Euler, es controlar la velocidad de alabeo y guiñada. Es decir, detectar si alguna de estas variables es inusualmente alta – lo cual podría deberse a ráfagas de viento o a un problema con las superficies de control.
   En el caso de que la aeronave no pueda reponerse de esta situación, lo siguiente opción es devolver el control al piloto, que estará más capacitado para afrontar una situación anormal y adaptarse a las nuevas circunstancias de la aeronave.
   En circunstancias normales, la velocidad de alabeo segura se considera de 15°/s.

2. **Angulo de alabeo excesivo**:
   Los giros en el plano horizontal se diseñan para velocidades de cambio de rumbo de 1.5°/s y 3°/s. Esto significa que en función de la velocidad de la aeronave empleará el angulo de alabeo (o *bank angle*) y el ratio de giro al que mejor se adapte. Nótese que la velocidad es una propiedad característica de cada aeronave, y dependerá de su masa y de sus cualidades aerodinámicas. En general un giro con ángulo de alabeo mayor de 30° se encuentra fuera de los estándares de cualquier ruta diseñada por un organismo de aviación. Aun así, considerando las capacidades de las aeronaves, los sistemas AP actuales permiten un alabeo de hasta $66^{\underline{o}}$ cuando el piloto fuerza el joystick al límite de su posición y mantiene dicha fuerza aplicada.
   En el caso a estudiar, se tomará como límite estándar $33^{\underline{o}}$, ya que se trata de maniobras llevadas a cabo por el propio RPAS, y no por el piloto.

Nota: Los casos expuestos anteriormente, guardan una estrecha relación con las limitaciones de actuación impuestas por los sistemas FBW, de modo que se pueden extrapolar los valores de las limitaciones como punto de partida (tabla 1.2. [23]

Variable	Normal Envolvente Normal	Envolvente Periférica
Factor de carga	$-1g <= n_z <= 2{,}5g$	
Ángulo de Cabezeo	$-15^\circ <= \theta <= 30^\circ$	
Ángulo de Ataque	$-5^\circ <= \alpha <= \alpha_{prot}$	$-5^\circ <= \alpha <= \alpha_{max}$
Altitud	$0 <= h <= 41000ft$	
Velocidad	$V_{\alpha_{prot}} <= V_{cas} <= V_{MO}$	$V_{\alpha_{max}} <= V_{cas} <= V_D$
Número de Mach	$0{,}2 <= M <= M_{MO}$	$0{,}2 <= M <= M_D$
Ángulo de Roll	$-33^\circ <= \phi <= 33^\circ$	$-66^\circ <= \phi <= 66^\circ$

Tabla 1.2: Definición de envolventes de vuelo de un sistema FBW [13]

1.4.2. Fallo de posicionamiento

El plan de actualización del espacio aéreo enmarcado en el proyecto Single European Sky ATM Research (SESAR) tiene como objetivo optimizar el uso del espacio aéreo mediante la asignación a cada tramo de cierto requerimientos de posicionamiento definidos por varias etiquetas – como RNAV-X o RNP-X. En estas segmentos de espacio aéreo, el fallo de una radioayuda puede dejar a la aeronave fuera de los límites de seguridad poniendo el riesgo el resto de aeronaves.

Por otro lado, en RPAS de menor envergadura donde solo se dispone del receptor GPS, debe tenerse en cuenta que el error de posicionamiento puede cambiar bruscamente debido a efectos como la reflexión de la señal en accidentes geográficos o en edificios. Por tanto en cuestión de minutos, los valores de precisión, integridad, continuidad y fiabilidad del sistema pueden pasar de cumplir los requisitos a ser insuficientes.

1.4.3. Alerta de tráfico - Resolution Advisory

Al utilizar espacio aéreo compartido, no es difícil encontrarse con otras aeronaves en el radio de actuación la aeronave. Para evitar una posible colisión entre aeronaves, el RPAS debe incorporar un sistema de TCAS.

Los sistemas TCAS son capaces de detectar otras aeronaves cercanas e indicar la mejor maniobra para evitar la colisión – pudiendo ser coordinada, o no coordinada entre las aeronaves – si se detecta que el punto de acercamiento máximo (Point of Closest Approach, o PCA) tendrá lugar dentro del volumen de protección de la aeronave. [24]

Figura 1.10: Representación de un TCAS - Un ejemplo de ACAS [8]

1.4.4. Otras contingencias

Cabe destacar dos contingencias más dado que, aunque no son el foco de este proyecto, representar un riesgo razonable y es prudente mencionarlas.

Problemas de potencia

Estos podrían incluir fallos en los motores, tanto eléctricos, como alternativos o a reacción, que irían desde la colisión de aves hasta la rotura o descompensado de las hélices.

El caso concreto, en el que se produce una situaciones de poco combustible o batería baja son eventos para los que se puede diseñar un procedimiento de contingencia, y sí son objeto de estudio.

Fallos de lectura de los sensores

Debido a la naturaleza de los sensores, existen multitud de razones para obtener lecturas de valores erróneos (o nulos). Éstas pueden ir desde la presencia de hielo hasta exceso de ruido electromagnético o vibraciones.

Este tema constituye por sí solo un bloque de trabajo distinto al de este proyecto, y se sugiere como posible ampliación.

1.4.5. Pérdida del Enlace C2

La pérdida del enlace C2, es el caso de **contingencia más característica** en los sistemas RPAS, y aunque se puede enfocar desde distintas perspectivas, siempre implica que el piloto remoto no podrá controlar la aeronave a través de los canales de comunicación previstos a tal efecto durante un tiempo, a priori, desconocido. Es decir, **la aeronave funcionará de forma autónoma**.

Según la Organización de Aviación Civil Internacional (OACI), en caso de pérdida del enlace C2 se contemplan cinco actuaciones posibles, aunque ninguna de ellas se válida en todos los casos [25]:

- **Continuar con el plan original**
  Aunque podría ser una opción a considerar para vuelos cortos, en la mayoría de circunstancias supone que la aeronave volará por sí sola durante demasiado tiempo, siendo una opción poco viable.

- **Aterrizar en el punto de aterrizaje definido más cercano**:
  Esta opción garantiza que el vuelo será, en principio, lo más corto posible. Puede ocurrir que el punto de aterrizaje definido más cercano sea el punto de aterrizaje final, y que el Aeronave Pilotada de forma Remota (RPA) se dirija directamente hacia él sin completar su misión.

- **Retorno al aeródromo o punto de partida**:
  De forma similar a la primera actuación, ésta puede ser apropiada para vuelos de corto alcance, sin embargo, en un vuelo de largo alcance carece de sentido tratar de volver al aeródromo de partida más allá de cierta distancia recorrida.

- **Terminación del vuelo**:
  Esta opción supone que el vuelo debe finalizar en ese punto. Esto se puede afrontar desde varias perspectivas en función del tipo de aeronave, aunque la práctica más recomendada consiste en la apertura de un paracaídas y detención de los motores.
  Por otro lado, se considera una maniobra peligrosa, ya que en función de la altura y la zona en la que se encuentre, el RPA podría dañar a otras aeronaves o causar daños materiales y humanos.

- **Ascender a una altitud que permita recuperar el enlace C2**:
  Aunque es la opción que garantiza en mayor medida el cumplimiento de la misión del RPAS, no puede ser considerada un práctica estándar, ya que en el espacio aéreo controlado la altitud de vuelo de las aeronaves está regulada por el controlador ATC a cargo, y no siempre va a ser posible reservar coordinar la maniobra de cambio de nivel de vuelo.
  Por tanto, esta opción debe considerarse junto a un procedimiento alternativo en caso de que las circunstancias no lo permitan.

Capítulo 2

Propuesta de solución

Una vez presentado los problemas que plantean los RPAS a nivel Gestión del Tráfico Aéreo (ATM), se propone el uso del Plan de Misión, y la utilización de Gestor de Misión sobre el que se implementa un código para proteger el vuelo frente a contingencias. Esto es, una ampliación del proyecto de Hèctor Molina Usach, con la participación de Alfonso Crespo y Pedro Yuste. [11]

2.1. Plan de Misión

El **Plan de Misión** es un concepto basado en el Plan de Vuelo, cuya finalidad es definir, a nivel de integración en el sistema automático del RPAS, la ruta a seguir así como las rutas alternativas en caso de pérdida del enlace C2. La principal diferencia entre el Plan de Vuelo y el Plan de Misión, es que el primero está está enfocado a la gestión del espacio aéreo por parte de los ATC.

En el Plan de Misión, se definen tanto el recorrido en espacio aéreo controlado como las maniobras en el área de operaciones donde se lleva acabo el objetivo de la misión del RPAS, aunque la mayoría de operaciones de éstos tienen lugar en espacio aéreo segregado.

Adicionalmente, el Plan de Misión podría servir de referencia para el personal ATC, ya que disponer de las rutas alternativas que debe utilizar el RPAS en caso de pérdida del enlace C2 es un dato importante del que el servicio de ATC debe disponer para cumplir su propósito.

2.1.1. Definición del Plan de Misión

En la implementación que se va a llevar a cabo del Plan de Misión , se empleará el estándar ARINC-424 presentado anteriormente, para definir las rutas – principal y alternativas – por medio de Path Terminators.

Sin embargo, el estándar ARINC-424 no está diseñado para la realización de misiones, y por tanto, no incluye todos los Path Terminators que podrían ser necesarios.

2.1.2. Extended Path Terminator

Por subsanar el problema anterior, es necesario el uso de Extended Path Terminators, una adición a la lista de PT que complementa este estándar, con funcionalidades características de los sistemas RPAS, tales como, escaneo en S del terreno, patrones de espera y orbitar alrededor de un punto, entre otros.

La lista completa de los Extended Path Terminator (EPT) propuestos se encuentra en la tabla 2.1

class	coding	fix1lat	fix1lon	fix2lat	fix2lon	altitude1	altitude2	speed limit	course	fly-over	radius	turn direction	limit value	vertical angle
Path Terminator	CA							O	✓				1	O
	CF	✓	✓			O	O	O	✓	O				
	DF	✓	✓			O	O	O		O				
	FA	✓	✓					O	✓				1	O
	FM	✓	✓					O	✓					
	IF	✓	✓			✓	O	O	O					
	RF	✓	✓			O	O	O			✓	✓		
	TF	✓	✓			O	O	O		O				
	VA							O	2				1	
	VI	3	3			O	O	O	2				4	
	VM							O	2					
Extended PT	LW	5	5			6		O	7					✓
	RA	8	8					O			✓	✓	1	O
	RL	8	8					O			✓	✓	9	
	RM	8	8					O			✓	✓		
	RT	8	8					O			✓	✓	10	
	SF	✓	✓			O	O	O	✓					
	SQ	✓	✓	✓	✓	O		O	11		✓			
Conting...	NFZ AVOIDANCE	✓	✓					O			✓	✓		
	COLLISICN AVOIDANCE							✓	✓					O
	STABILIZATION													
	FLIGHT TERMINATION													

Tabla 2.1: Lista de Extended Path Terminators sugeridos [14]

2.1.3. Caso de estudio

Para la validación de las propuestas de este trabajo, se propone una misión que sirva de ejemplo para mostrar el funcionamiento de los sistemas y estudiar sus resultados.

La misión, descrita sobre la figura 2.1 se compone de los siguientes tramos:

Figura 2.1: Plan de Misión para validar el sistema [9]

1. Salida (Initial Fix) directamente desde la pista 18 del aeródromo de Teruel – con código OACI LETL –, y el RPA autorizado a despegar con rumbo 180°.

2. Ascenso en espiral (maniobra ORBIT) hasta los 9500 pies entorno al punto con coordenadas (40.3164° N, 1.1856° O), para incorporarse a la aerovía R29.

3. Vuelo hasta el punto (FLY TO) situado a 40 milla náutica (MN) de CMA con rumbo 169° para interceptar la aerovía R29 en CMA169040/N0150F110 a nivel FL 110.

4. Vuelo con rumbo constante (Track to Fix) hasta el waypoint SOBRO.

5. Inspección (SCAN) de una superficie loxodrómica cuadrada sobre La Albufera definida por los vértices con coordenadas (39° 21’ N, 0° 24’ 30”W) y (39° 19’ N, 0° 18’ W) y una dirección de referencia de 160°, a 2000 pies.

6. Salida y ascenso (Course to Fix) desde La Albufera por el pasillo visual Sollana (440-797-LE_AD-2), con rumbo 340° hacia la radioayuda VLC (Valencia).

Figura 2.2: IAI SuperHeron en un expositor (Paris Air Show, 2009) [10]

7. Continuación del ascenso (Course to Fix) a través del pasillo visual El Puig (440-797-LE_AD-2) hasta las coordenadas (39° 35' 38"N, 0° 18' 01.º) con rumbo constante de 53°.

8. Continuación del ascenso (Course to Fix) a través del pasillo visual Sagunto (440-797-LE_AD-2) hasta las coordenadas (39° 39' 22"N, 0° 12' 44.º) con rumbo constante de 48°.

9. Vuelo directo (Fly To) hacia SOPET (39.8338° N, 0.0047° O).

10. Vuelo a rumbo constante (Course to Fix) hacia un waypoint virtual – antes de llegar a TORDU – con coordenadas (40.2581° N, 0.5883° E) con rumbo constante 47°.

11. Vuelo directo (Fly To) hacia el Inital Fix (IF) del aeropuerto de Castellón – con código OACI LECH.

12. Procedimiento de aterrizaje (LAND) en el aerpouerto de Castellón.

Para la simulación, se utilizará un modelo del RPAS israelí IAI Super Heron (como el que se puede ver en la imagen 2.2 – versión mejorada del model Heron anterior–, también conocido como *Majatz-1*, y desarrollado por la empresa Isarel Aerospace Industries. Su ficha técnica se encuentra en el anexo I.a.

2.2. Diseño de EPT

2.2.1. SCAN

El Extended Path Terminators SCAN se basa en el principio de escanear una sección rectangular de terrero realizando un patrón en S – desde un punto de vista loxodrómico. Es decir, la aeronave se desplaza en una dirección observando una franja rectangular de la superficie, gira, y vuelve paralelamente en sentido contrario mientras observa la siguiente franja de terreno. Esto se repite hasta cubrir toda la superficie.

Este procedimiento requiere una serie de entradas, las cuales siguiendo el estándar ARINC-424 serán:

- Fijo 1
- Fijo 2
- Rumbo

Dado que las entradas anteriores pueden resultar ambiguas y dar lugar a muchos procedimientos distintos, se definen una serie requerimientos, y un parámetro extra: el radio de giro (que se asume igual al radio de observación de la aeronave).

Los requerimientos para concretar el desarrollo del procedimiento y garantizar su predecibilidad son:

1. Los fijos 1 y 2, definen dos esquinas de la misma diagonal del rectángulo a escanear
2. La aeronave debe entrar próxima al fijo 1, y salir próxima al fijo 2 (definición coherente)
3. El rumbo debe indicar la dirección y sentido de entrada al rectángulo, que será siempre perpendicular a uno de los lados de éste.
    - Si el los fijos 1 y 2 no se definen de forma coherente los puntos de entrada y salida del área no serán los fijos 1 y 2, y el rumbo de entrada será el opuesto (+180°)
4. La distancia desde la esquinas hasta el punto de entrada y el punto de salida será equivalente al radio de giro
5. La aeronave debe entrar y salir con el mismo rumbo, próximo a esquinas opuestas (el numero de franjas será siempre impar)
    - La separación entre “legs” en una dirección y otra será del doble del radio, ajustado para cubrir toda la zona y que se cumpla la regla anterior.

2.3. Gestor de Misión

El Gestor de Misión que se utilizará es una rama del trabajo de Hèctor Usach Molina (07/03/2016), que a su vez es una continuación del trabajo de Borja Fons Albert (10/12/2012). Sobre este se implementará el código el código de Protección de Envolvente de Vuelo durante el vuelo Automático, o AFEP, cuya finalidad es la garantizar la viabilidad de los vuelos sistemáticos de sistemas RPAS.

Se trata de un Gestor de Misión diseñado sobre el software StateFlow incorporado en *Matlab* – Versión 2012b – de la cual el Departamento de Informática de Sistemas y Computadores Automáticos (DISCA) dispone de una licencia.

Este sistema, es capaz de interpretar un Plan de Misión definido por un archivo de *Matlab* en el que se describe la ruta en base a los Path Terminators que la componen. De forma similar se introducen las rutas alternativas / de contingencia que se deben emplear en función del estado de la misión, es decir, a partir de cierto punto, el plan de contingencia será uno u otro.

Para garantizar la predecibilidad de la operación en todo momento, cuando se cambia de una zona con un plan de contingencia a la siguiente zona, con un plan de contingencia diferente, la interfaz de usuario solicita al operador de la aeronave que "arme" el siguiente plan de contingencia (confirmación), de modo que, si se pierde en enlace C2 durante la transición, la aeronave siempre ejecutará el último plan armado por el operador.

Su funcionamiento está basado en el diseño de sistemas de tiempo real, y actúa como una capa intermedia entre el usuario, y el software de simulación, *X-Plane*, mediante la comunicación a través de puertos UDP.

2.3.1. Funcionalidad

Según el modo de funcionamiento que se encuentre activo, permite el by-pass de la capa de control, para el pilotaje manual, con o sin protección de la estabilidad de vuelo, así como un modo de funcionamiento "por comandos", muy similar al funcionamiento de un AP convencional, en el que solo es necesario indicar los parámetros de vuelo estacionarios (rumbo, velocidades y altura deseada).

Sin embargo, la característica principal, es el tercer modo: Misión. Que simula un sistema FMS avanzado, capaz de llevar a cabo el Plan de Misión a partir de los Extended Path Terminators. Además, es capaz de detectar contingencias – introducidas mediante una interfaz de usuario de testeo – tales como fallos del Sistema Global de Navegación por Satélite (GNSS) o del sistema ACAS, fallos de las receptores de radioayudas o del altímetro, y en especial, fallo del enlace C2.

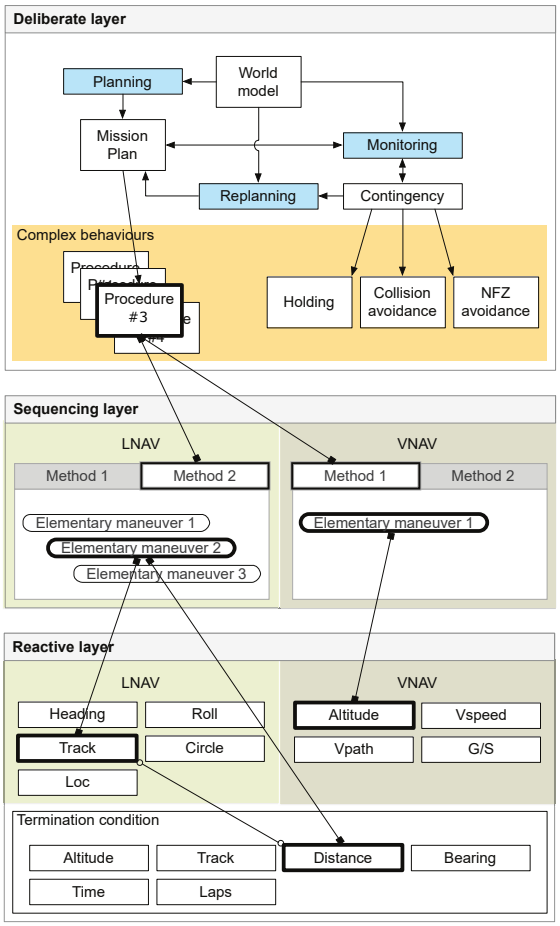

Figura 2.3: Arquitectura de 3 capas [11]

2.3.2. Arquitectura del Gestor de Misión

La arquitectura del Gestor de Misión está basada en una máquina de estados con 3 capas principales: Deliberativa, Secuenciadora y Reactiva, como se puede observar en la figura 2.3.

1. **Deliberativa**:

   Es la capa del FMS que funciona a más alto nivel, y se encarga de interpretar los EPT y activar la capa Secuenciadora cada vez que se completa un procedimiento, para que se inicie la siguiente maniobra. Detecta problemas generales de estabilidad, e informa de cuando es necesario actualizar el plan de contingencia armado.

2. **Secuenciadora**:

Es la capa intermedia entre la ejecución de la maniobra y la interpretación del Plan de Misión. Recibe el EPT que corresponde ejecutar en cada momento, así como los parámetros del mismo – tales como coordenadas iniciales y finales, altitudes y rumbos. Su función es descomponer los EPT en órdenes de más bajo nivel y detectar si se han cumplido las condiciones que determinan la finalización de la maniobra.
Estas órdenes se separan en tres bloques:

a) **LNAV** o **Navegación Lateral**: describe cada maniobra EPT en el plano lateral.

b) **VNAV** o **Navegación Vertical**: describe cada maniobra EPT en el plano vertical.

c) **THR** o **Empuje**: describe cómo debe ser la velocidad.

3. **Reactiva**:

   Es la capa de más bajo nivel. Se encarga de transmitir las órdenes de bajo nivel de cada bloque (LNAV, VNAV y THR) al AP de la aeronave. Sobre esta capa se implementará el AFEP, ya que es el sistema debe proteger a la aeronave, incluso de sus propios fallos de diseño.

2.4. Gestión de contingencias

Para garantizar la Automatización de Planes de Misión de los sistemas RPAS, se escoge una solución basada en tres elementos fundamentales, que garanticen la seguridad durante toda la operación:

1. El diseño seguro de Planes de Vuelo Alternativos, que permitan una vía de escape para las situaciones como pérdida del enlace C2.

2. El diseño seguro de la Transición entre Planes de Vuelo, que mantengan la aeronave en situación controlada independientemente del lugar donde se produzca la incidencia.

3. El diseño de un sistema de protección de envolvente (AFEP), que garantice la estabilidad de la aeronave durante todo el proceso.

2.4.1. Planes de Vuelo Alternativos

El diseño de Planes de Vuelo Alternativos tiene como objetivo garantizar una ruta segura en caso de pérdida del enlace C2. Se considera que el enlace C2 se

ha perdido cuando ya no es posible recuperar el control de las comunicaciones, pudiendo ser de forma temporal o definitiva. [26]

Esto implica que un Plan de Vuelo alternativo debe tener en especial consideración la ausencia de piloto. Por tanto, se deben evitar rutas con elevado tráfico o donde exista riesgo de colisión con los accidentes geográficos. Además, se considerarán las siguientes reglas para el diseño de Planes de Vuelo alternativos **durante la fase de planificación del vuelo**:

1. Si la distancia recorrida no es demasiado larga, se volverá al aeródromo de partida.

2. Si el aeródromo de destino se encuentra a una distancia razonable, se volará hacia el aeródromo de destino.

3. Si la distancia recorrida hace que volar hacia al aeródromo de origen o al de destino implique recorrer un distancia demasiado larga, se deben considerar otros aeródromos alternativos en los que finalizar el vuelo.

4. Si la distancia al aeródromo alternativo es excesiva, pero existe algún punto cercano en el que se puede realizar un circuito de espera, se realizará un circuito de espera para intentar recuperar el enlace de comunicaciones.

    a) Si el tiempo de espera supera cierto límite – en general, un límite impuesto por la capacidad de combustible / energía – se finalizará el vuelo abriendo el paracaídas.

5. Si no existe ninguna opción para aterrizar en un aeropuerto cercano, ni para realizar un circuito de espera, se finalizará el vuelo abriendo el paracaídas.

Caso de estudio

Para la validación del sistema implementado, es necesario definir los siguientes Planes de Vuelo alternativos.

Plan de Vuelo Alternativo 1

Desde el despegue y hasta llegar al waypoint SOBRO, el Plan de Vuelo alternativo consistirá en:

1. Volar (Course to Fix) hasta las coordenadas (40.3164° N, 1.1856° O) a una altitud de 5500 pies.

2. Una vez allí, orbitar (ORBIT) en sentido horario entorno a dichas coordenadas con un radio de 0.5 MN durante un tiempo máximo de 500 segundos.

Plan de Vuelo Alternativo 2
Desde SOBRO hasta terminar el procedimiento SCAN, el Plan de Vuelo alternativo consistirá en:

1. Volar directamente (Direct to Fix) hasta las coordenadas (40.1224° N, 0.2393° E) a una altitud de 5000 pies.

2. Una vez allí, orbitar (ORBIT) en sentido horario entorno a dichas coordenadas con un radio de 0.5 MN durante 200 segundos.

3. Si no se ha recuperado el enlace C2 en ese plazo, descender orbitando (ORBIT) hasta el terreno (flight termination) en dicha zona.

Plan de Vuelo Alternativo 3
Después de la fase de SCAN, y hasta finalizar el Plan de Vuelo de la lista que se encuentra en la sección 2.1.3, el Plan de Vuelo alternativo consistirá en:

1. Volar directamente (Direct to Fix) hasta las coordenadas (39.24° N, 0.075° O) a una altitud máxima 5500 pies.

2. Una vez allí, orbitar (ORBIT) en sentido horario entorno a dichas coordenadas con un radio de 0.5 MN durante 200 segundos.

2.4.2. Transición entre Planes de Vuelo

Las incidencias que provoca que la aeronave cambie del Plan de Vuelo principal al alternativo pueden ocurrir en cualquier punto de la ruta. Esto significa que, si solo se definen los Planes de Vuelo, la ruta no estará totalmente definida, ya que la transición de un Plan de Vuelo a otro vendrá sujeta a la implementación del Gestor de Misión. Por tanto, debe definirse una serie de requerimientos que determinen cómo será esta transición.

Estos requerimientos se basan en datos previos a cerca del comportamiento del Gestor de Misión utilizado:

- La transición debe hacerse a una altura tal que no implique riesgo de colisión contra el terreno.

- En el Gestor de Misión debe haber dos Planes de Vuelo en memoria: el Plan de Vuelo activado y el Plan de Vuelo armado – En funcionamiento normal el Plan de Vuelo activado es el Plan de Vuelo principal, mientras que el armado debe ser el Plan de Vuelo alternativo correspondiente a dicha fase de la ruta

- En caso de pérdida del enlace C2, el plan de vuelo a seguir pasará a ser el último Plan de Vuelo **armado manualmente**

    - Esto ocurre aunque el el último Plan de Vuelo armado no sea el que corresponde a esa fase del vuelo actual.
    - Con esto se pretende que, en las proximidades al lugar donde se cambia de un Plan de Vuelo alternativo al siguiente, no haya duda qué Plan de Vuelo seguirá la aeronave

- La incorporación a un Plan de Vuelo alternativo con origen en un punto lejano, se puede hacer desplazándose directamente al punto más cercano de dicho Plan de Vuelo, siempre que la incorporación se realice con un ángulo menor a 30°.

    - Esto se debe a que para giros de hasta 30°, los parámetros de estabilidad se mantienen en valores seguros, mientras que para giros mayores existe también un riesgo mayor de pérdida de control.

De este último punto se deduce que la incorporación al plan de vuelo alternativo se puede hacer con un patrón con forma de 3 simétrico respecto a la dirección de desplazamiento en el punto de la incorporación.

Para diseñar este patrón se descompone el procedimiento en fases:

- Fase de giro: se define como el giro máximo sin perder estabilidad. Se caracteriza por un ángulo girado, $\alpha_{\max}$ (se define anteriormente como 30°), y un radio de giro, que a su vez viene dado por la ecuación $r_G = \frac{1}{g}\tan(\phi)V^2$.
- Fase de estabilización: se define como un tramo recto después de cada giro para recuperar la estabilidad. Se caracteriza por una distancia de estabilización, l_E.

Para conocer el punto más cercano desde el que una aeronave con un rumbo dado puede iniciar el cambio de Plan de Vuelo, se parte del punto y la dirección finales, y se combina las dos fases descritas anteriormente hasta igualar el rumbo de la aeronave. Si el punto se encuentra en una posición más cercana al punto final, la aeronave tendrá que alejarse primero e iniciar el cambio de Plan de Vuelo con más margen de maniobra.

Para ilustrar este procedimiento, se calcula el punto de incorporación más cercano para una aeronave con rumbo 180° que desea incorporarse a una ruta con rumbo 270° – un giro de 90°. La elección de estas direcciones simplifica utilizar el sistema de ángulos cartesiano (en lugar del geográfico). Ver figura 2.4.

En primer lugar, es necesario describir la fase de giro mediante un vector de desplazamiento, cuyos parámetros serán:

- Módulo: $l_g = r_G\sqrt{2\left((1-\cos\alpha_{\max}\right)}$

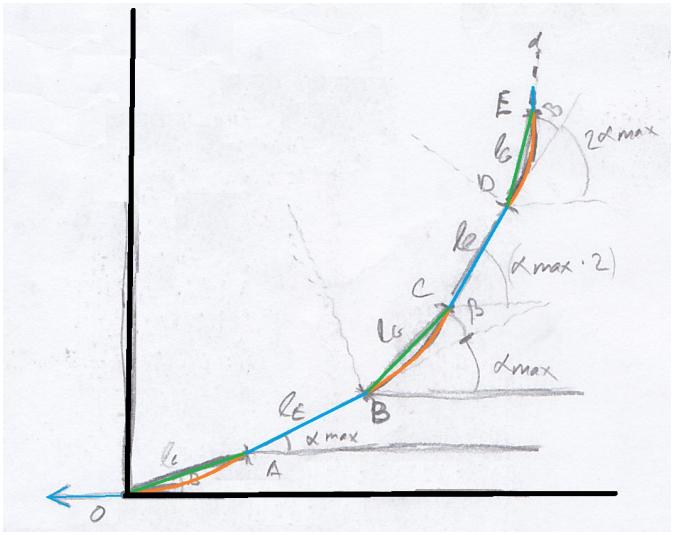

Figura 2.4: Diseño de cambio de Plan de Vuelo

- Ángulo: $\beta = \cot\left(\dfrac{1-\cos\alpha_{\max}}{\sin\alpha_{\max}}\right)$

En segundo lugar, se trazan los un vector de la fase de giro (O-A), seguido de un vector de la fase de estabilización (A-B), que ya va girado 30°.

$$A = \begin{bmatrix} l_G\cos(\beta) \\ l_G\sin(\beta) \end{bmatrix} \quad B = \begin{bmatrix} l_G\cos(\beta) + l_E\cos(\alpha_{\max}) \\ l_G\sin(\beta) + l_E\sin(\alpha_{\max})) \end{bmatrix} \tag{2.1}$$

Posteriormente se trata de añadir Fase de giro (B-C), Fase de estabilización (C-D) y Fase de giro (D-E), para terminar de descomponer el procedimiento:

$$C = B + \begin{bmatrix} l_G\cos(\beta+\alpha_{\max}) \\ l_G\sin(\beta+\alpha_{\max}) \end{bmatrix} \quad D = C + \begin{bmatrix} l_E\cos(2\alpha_{\max}) \\ l_E\sin(2\alpha_{\max})) \end{bmatrix} \quad E = D + \begin{bmatrix} l_G\cos(\beta+2\alpha_{\max}) \\ l_G\sin(\beta+2\alpha_{\max}) \end{bmatrix} \tag{2.2}$$

$$E = l_G \begin{bmatrix} \cos(\beta)+\cos(\beta+\alpha_{\max})+\cos(\beta+2\alpha_{\max}) \\ \sin(\beta)+\sin(\beta+\alpha_{\max})+\sin(\beta+2\alpha_{\max}) \end{bmatrix} + l_E \begin{bmatrix} \cos(\alpha_{\max})+\cos(2\alpha_{\max}) \\ \sin(\alpha_{\max})+\sin(2\alpha_{\max}) \end{bmatrix} \tag{2.3}$$

Una vez definido el punto de incorporación más cercano para un giro a 90°(el punto E), es fácil extender la secuencia, pudiéndose diseñar incorporaciones a 180° o incluso más.

Al dibujar el procedimiento completo, se pueden prolongar las fases de estabilización, tal y como se hace en la figura 2.5. Estas representan trayectorias de

entrada al patrón de incorporación desde posiciones más alejadas, lo que permite cubrir un espacio mayor.

2.4.3. Sistema de Protección de la Envolvente de Vuelo (AFEP)

La implementación del Gestor de Misión, pasa por utilizar el AP de la aeronave, como intérprete de las órdenes de bajo nivel que produce la *capa reactiva*.

Esto implica que, si los cálculos realizados por la capa reactiva son incorrectos – o se encuentran fuera de los límites de la aeronave–, o se produce un fallo en el control de la aeronave, o un elemento externo, como la lluvia o el viento, afecta a la estabilidad de la aeronave, el sistema podría desplazarse hacia un estado desde el cual el AP no es capaz de salir. Estos estados generalmente incluyen, la **entrada en pérdida** o que la **estructura de la aeronave** resulte **dañada**.

Por esta razón, se incorpora a la *capa reactiva* una nueva máquina de estados con capacidad de sobrescribir los valores de salida cuando sea necesario, que funciona paralelamente a las tres anteriores (LNAV, VNAV y THR).

El funcionamiento habitual de este sistema será, simplemente el de supervisar los valores de las protecciones implementadas y, solo en caso de incidencia, tendrá la capacidad de imponerse sobre la salida producida por la *capa reactiva*.

La forma de implementar estas protecciones, será tal que permita al usuario definir los límites de ciertos parámetros en función del resto de parámetros del vuelo, pudiendo definir, por ejemplo, funciones continuas a trozos que dependan de la altitud y la velocidad, mediante código de *Matlab*.

Protecciones

Las protecciones definidas se dividen en dos tipos, LNAV y VNAV, y en dos categorías, **Continuas**, y **Situacionales**.

Las protecciones **Continuas** se definen respecto de parámetros que dependen de una derivada temporal, como por ejemplo el factor de carga, N_z.

Mientra que las protecciones **Situacionales**, se definen respecto de parámetros puntuales como el ángulo de ataque, α.

Las protecciones que se implementan actúan para controlar los valores máximos de las variables de la tabla 2.2.

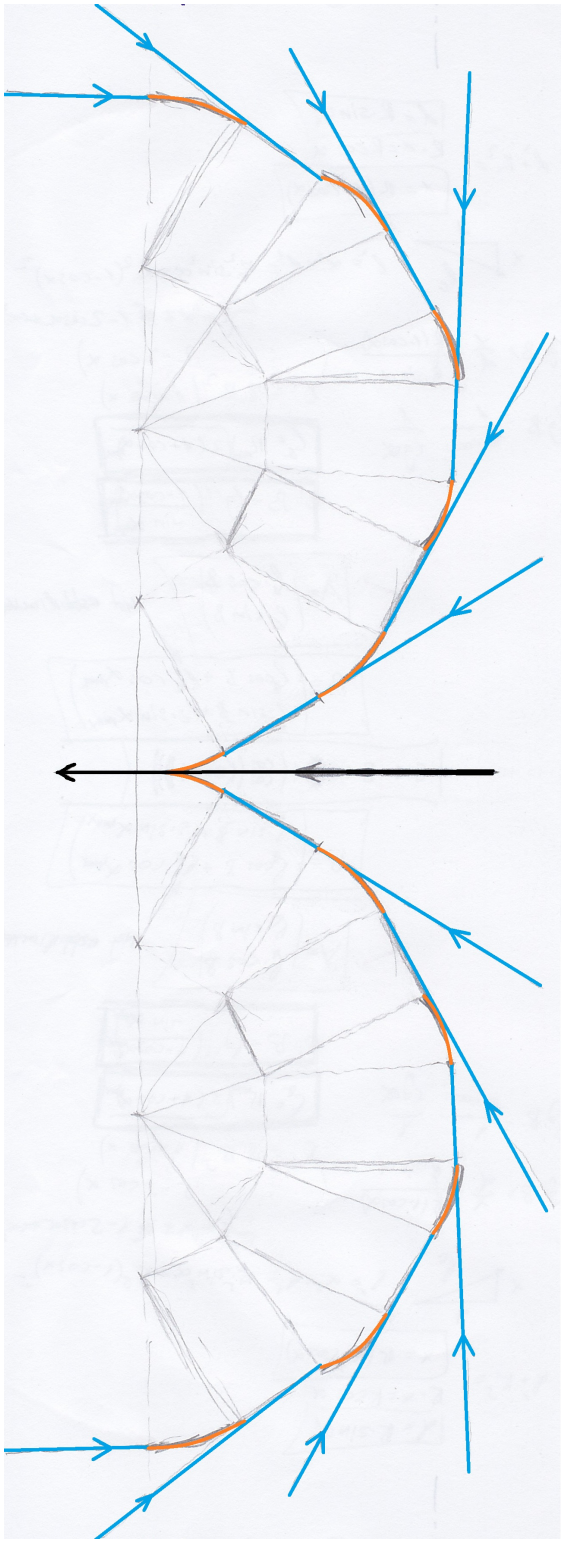

Figura 2.5: Patrón de incorporación a un waypoint con un rumbo dado

Protección	Tipo	Categoría
Ángulo de Ataque (AoA)	VNAV	Situacional
Velocidad	VNAV	Situacional
Factor de Carga (Normal Law)	VNAV	Continua
Alabeo	LNAV	Situacional
Aceleración del Alabeo (Lateral Law)	LNAV	Continua

Tabla 2.2: Protecciones implementadas en el AFEP

Los valores máximos de estas variables se pueden definir en funcion de los parámetros que se indican en la tabla 2.3.

Protección	Velocidad (V)	Altitud (z)
Ángulo de Ataque (AoA)	Sí	Sí
Velocidad	No	Sí
Factor de Carga (Normal Law)	Sí	Sí
Alabeo	Sí	Sí
Aceleración del Alabeo (Lateral Law)	Sí	Sí

Tabla 2.3: Variables de las que dependen las protecciones del AFEP

Para corregir aquellos parámetros que superen los valores máximos dados por las funciones definidas para cada variables, se utiliza un controlador proporcional, calibrado en cada caso.

Modos de funcionamiento

Los modos de funcionamiento dependen de si se trata de protecciones continuas o situacionales.

En el primer caso, las protecciones continuas, necesitan un estado de espera, en el que calcular el valor de la derivada temporal cuando el AFEP entra en funcionamiento.

En segundo lugar, las protecciones situacionales, requieren cierto margen de funcionamiento, ya que su comportamiento suele ser más amortiguado.

Adicionalmente, cada protección cuenta con dos estados de funcionamiento.

1. **Normal**: es el que se describe anteriormente
2. **Emergencia**:

Este modo permite exceder las limitaciones de rendimiento impuestas por el AFEP durante un breve periodo de tiempo, con la finalidad de evitar colisiones o realizar otras maniobras evasivas, cuando el FMS detecta el riesgo con margen de tiempo reducido.

Capítulo 3

Implementación

En este capítulo se resume brevemente los detalles de implementación de los sistemas descritos en el capítulo anterior.

El código y los diagramas completos se encuentran referenciados en los anexos.

3.1. Plan de Misión

3.1.1. Definición de waypoints

Ya que la ruta se diseña directamente en *Matlab*, los waypoints se pueden calcular en tiempo de ejecución. Esto permite simplificar y automatizar los cálculos de las coordenas, y otorga flexibilidad a la hora de definir puntos en función de cierta distancia y ángulo respecto de otro punto.

Esto se puede observar, en el anexo II.a, en la línea 32 del código.

3.1.2. Definición del Plan de Misión

El Plan de Misión, se puede encontrar en el anexo II, donde se encuentra dividido en

a) Plan de vuelo principal, anexo II.a

b) Plan de vuelo alternativo 1, anexo II.b

c) Plan de vuelo alternativo 2, anexo II.c

d) Plan de vuelo alternativo 3, anexo II.d

e) Transiciones entre Planes de Vuelo, anexo II.e

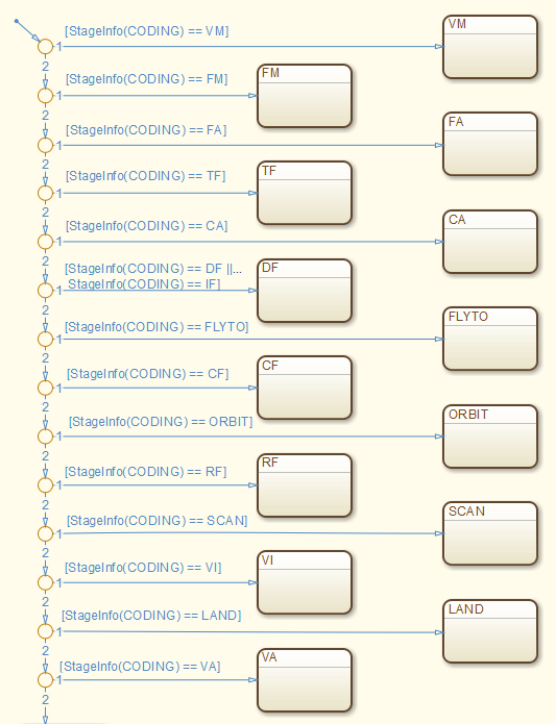

Figura 3.1: Implementación de los PT y EPT en la capa secuenciadora

Los procedimientos que se siguen son los mismos que los descritos en el apartado 2.1.3, con la excepción de la aproximación y aterrizaje, debido a la implementación del EPT LAND.

3.1.3. Path Terminators

La implementación de los Path Terminators se separa según su funcionamient en el plano horizontal (LNAV) y en el plano vertical (VNAV), y a su vez, cada una de estas, se define mediante bloques más pequeños, con diferentes propiedades.

Esto tiene lugar dentro de la *capa secuenciadora* donde se activa la máquina de estado correspondiente al PT (o EPT) que se debe ejecutar (ver figura 3.1).

El primero de estos bloques es un **método** que indica la forma de llevar a cabo la maniobra. Aunque en el caso de maniobras complejas se sustituye por código específico para dicho PT

En LNAV se utilizan los **métodos**:

- Track
- Heading

- Course
- Direct To

Mientras que en VNAV se utilizan:

- Vertical Speed
- Vertical Path (ángulo de ascenso)

Por otro lado, cada **método** va acompañado de una **condición**, que lleva asociada un **evento de terminación**, el cual se dispara cuando se cumple esta condición. Es decir, mientras la máquina se encuentra ejecutando un PT, internamente se activan dos máquinas, la primera de ellas ejecuta una de las maniobras de la lista anterior, y la segunda se encarga de detectar el alcance del objetivo (condición) y solicitar el siguiente PT.

En LNAV y VNAV se combinan las condiciones:

- Distance
- Altitude
- Detectores de terminación específicos:
  - Detector de llegada a un Fix
  - Detector de condiciones múltiples
  - Otros

Esta estructura, permite definir los Path Terminators de una forma sistemática y sencilla, como se puede ver en los PT de la figura 3.2.

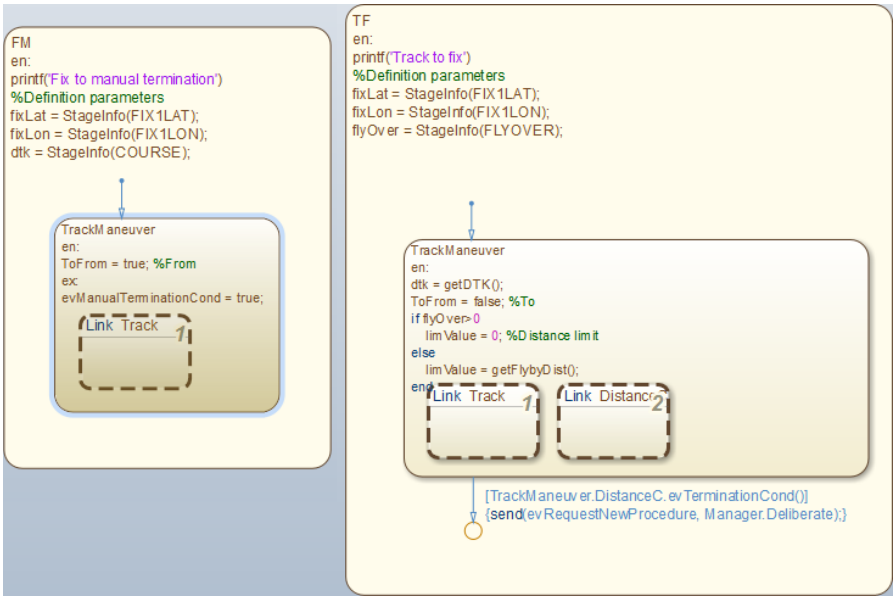

Figura 3.2: Implementación de varios PT (Fix to Manual y Track to Fix)

3.1.4. Extended Path Terminators

Los EPT se definen como secuencias de Path Terminators gestionados mediante combinaciones de código y máquinas de estado.

Tal es el caso del EPT de reconocimiento o es escaneo del terreno en S, llamado SCAN, como se puede ver en la figura 3.3.

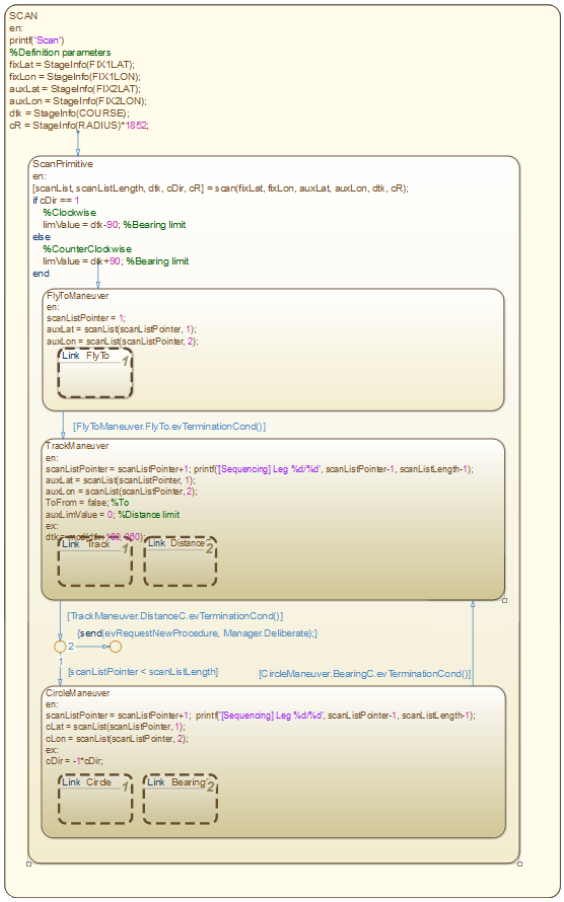

Figura 3.3: Interior de la máquina de estados SCAN

Y en el código del anexo III.a

3.2. Entorno de simulación superior

El conjunto total del sistema constituye un **modelo de Simulink** (*Matlab*) de bucle cerrado que se ejecuta periódicamente cada 1/16 s, la misma frecuencia que la tasa de transmisión de paquetes de *X-Plane*.

Este modelo representa el marco en el que ocurren los hechos, y no tiene una equivalente físico en la realidad, pero sirve para comunicar y permitir la interacción entre los distintos componentes de la simulación.

Está compuesto por dos bloques principales, el **Flight Management System** y el **Xplane Interface**, a los que se suma un tercer bloque, **HM Interface**, con el que se puede interactuar gráficamente durante la ejecución de la simulación. Ver figura 3.4.

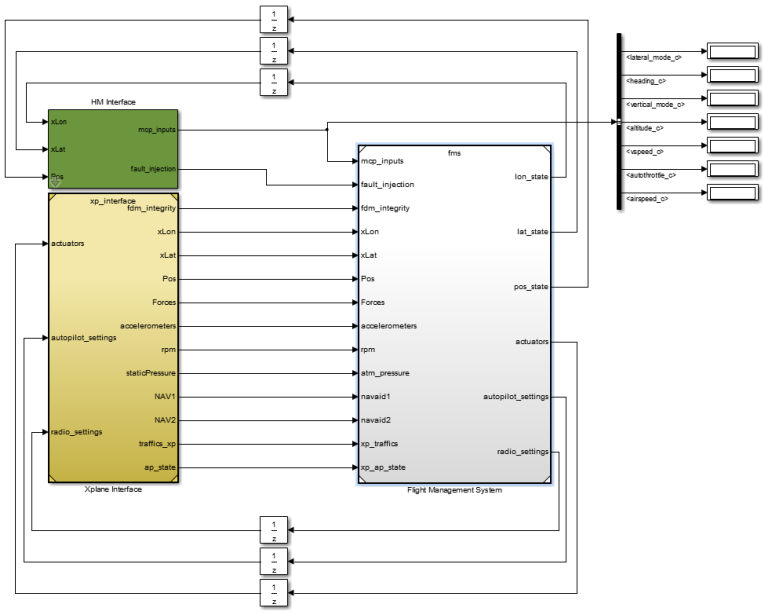

Figura 3.4: Aspecto del modelo de Simulink - Entorno de simulación superior

El Entorno de simulación superior está diseñado de tal forma que las salidas de datos de **HM Interface** y **Xplane Interface** son la entrada del **FMS**, y en el siguiente ciclo, la salida de **FMS** será la entrada de los dos bloques anteriores.

3.2.1. HM Interface

Este bloque añade a la simulación la interfaz gráfica que permite interactuar con la simulación – cambiar entre los modos Misión, Comandos y Manual, armar los planes de contingencia cuando se pasa de una zona a otra – e inyectar fallos para probar la respuesta del sistema, como se muestra en la figura 3.5.

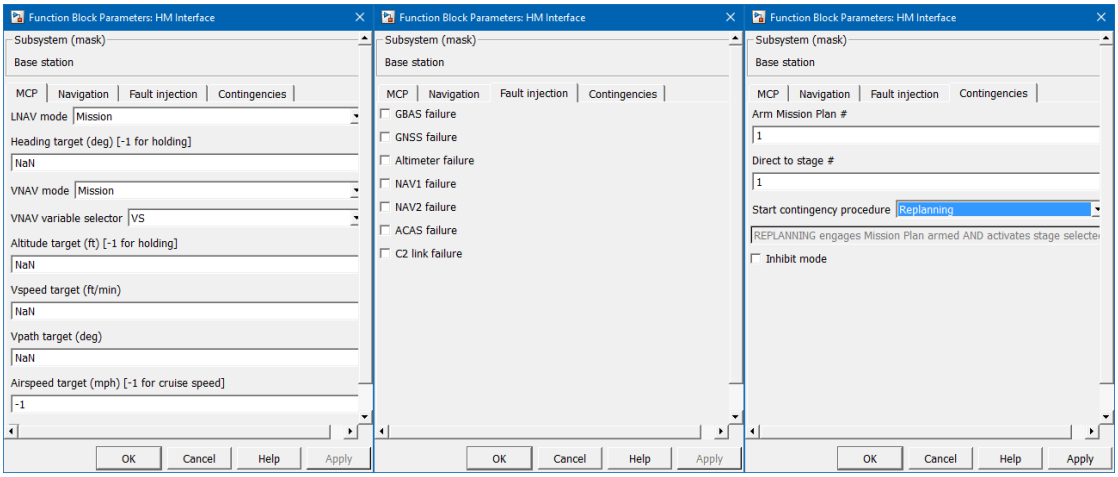

Figura 3.5: Aspecto del HM Interface - Pestañas MCP, Fault Injection y Contingency

3.2.2. X-Plane Interface

Este bloque recibe los datos del software de simulación *X-Plane*, y transmite la información que recibe del FMS calculada en el ciclo anterior.

Es imprescindible que en el archivo de configuración del proyecto de *Matlab* se establezcan las IPs y los puertos UDP correspondientes para la comunicación en red local. De forma similar, es necesario configurar el software de simulación *X-Plane* para que lea y escriba en los puertos en el sentido contrario a *Matlab* y Simulink, para permitir la comunicación bidireccional (ver figura 3.6).

3.3. Flight Management System

Es el bloque principal del Entorno de simulación superior (sección 3.2), y el de mayor nivel que sí representa un sistema físico de la simulación – el FMS.

Está compuesto a su vez por tres bloques de *Simulink*: Navigation, **Guidance** y XP_autopilot.

En este modelo se mezclan las entradas procedentes del Entorno de simulación superior con un bucle cerrado que mantiene la información de las órdenes de control del ciclo anterior mientras procesa las siguientes. Ver figura 3.7

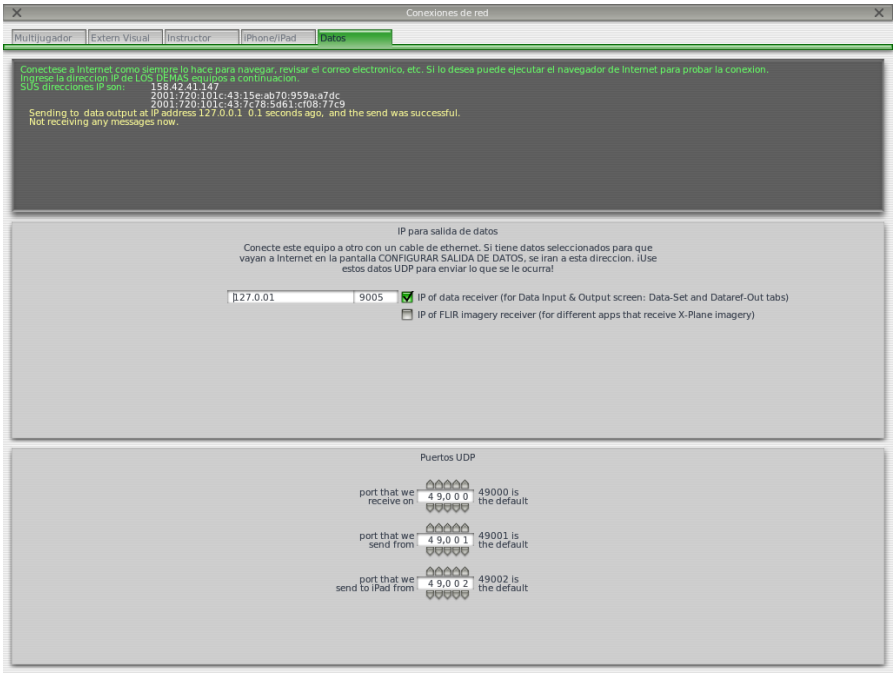

Figura 3.6: Interfaz de configuración de red de *X-Plane*

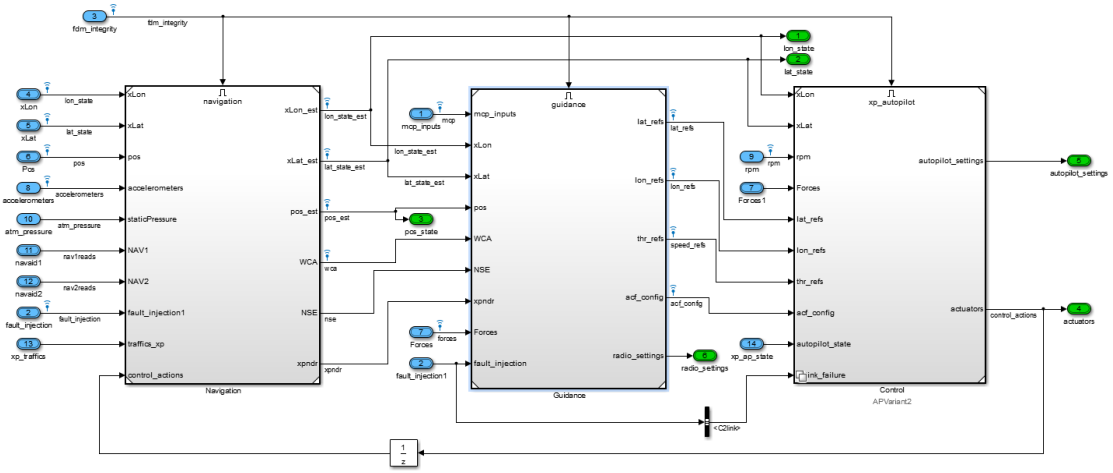

Figura 3.7: Bloque FMS de Simulink

3.3.1. Navigation

El bloque Navigation, permite interpretar la información de los sensores a bajo nivel, y gestionar procesos como la eliminación de ruido, o el procesado de infor-

mación de las radioayudas.

Físicamente, representa aquellos computadores de la red de la aeronave que funcionan de forma distribuida y que procesan la información analógica de los sensores para proveer una salida digital que finalmente utilizará el ordenador central.

No obstante, simular los defectos de los sensores supone un esfuerzo de cálculo sin sentido para el propósito de este proyecto, por tanto, este bloque se puede considerar deshabilitado a todos los efectos, sirviendo exclusivamente como puente entre sus entradas y sus salidas.

3.3.2. Guidance

Conforma el bloque central del FMS y agrupa los módulos de No Flight Zones (NFZ) y del ACAS. También es el lugar donde se sitúa el **Mission Manager** (ver figura 3.8), por tanto, se puede asumir que representa los dos sistemas anteriores y el FMC de la aeronave.

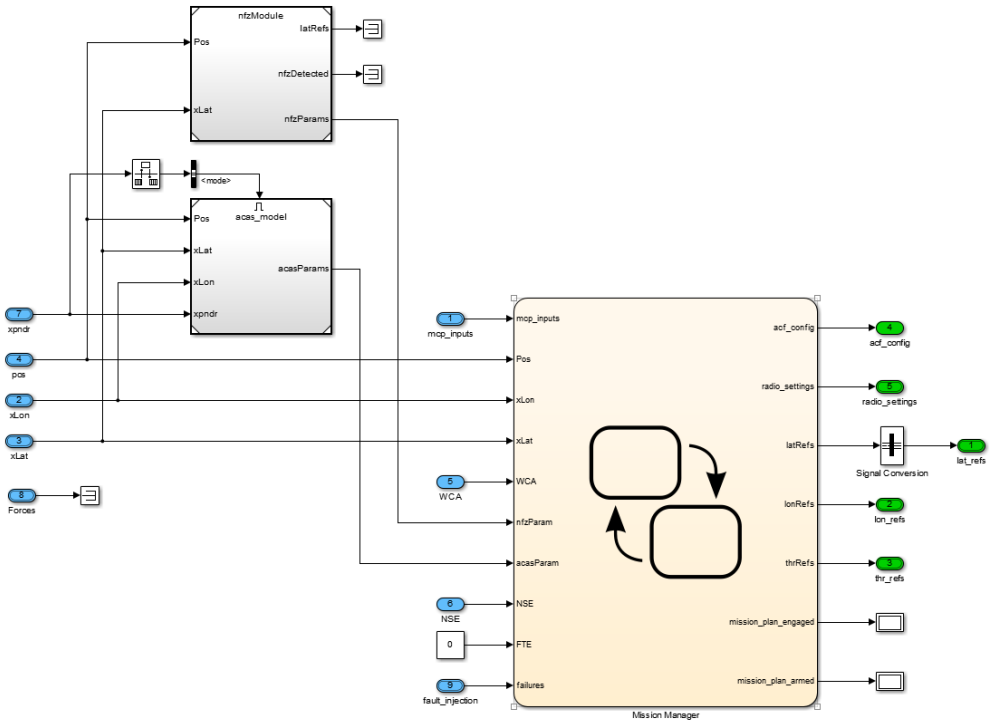

Figura 3.8: Bloque Guidance de Simulink

3.3.3. XP Autopilot

Es el bloque encargado de procesar la información de alto nivel procedente del bloque **Guidance**, y darle el formato apropiado para que sea procesada por el Auto Pilot de la aeronave en *X-Plane*.

Esta tarea consiste, principalmente, en transmitir las referencias de posición adecuadas en función de la ruta a seguir, así como activar o desactivar los sistemas – LNAV, VNAV, Altitud Hold, etc – correspondientes al modo de funcionamiento requerido en cada momento

3.4. Gestor de Misión - Mission Manager

La arquitectura de Stateflow permite definir máquinas que funcionan paralelamente dentro de otras máquinas. Este es el caso del Mission Manager, en cuyo interior se encuentran únicamente las funciones de uso general y las 3 capas de la arquitectura descritas en la sección 2.3.2, cuyo funcionamiento se esquematiza en la imagen 3.9.

Figura 3.9: Máquina de estados Mission Manager del bloque Guidance

3.4.1. Capa Deliberativa

Ver el anexo IV.a.

3.4.2. Capa Secuenciadora

Ver el anexo IV.b.

Modos LNAV, VNAV y THR

Ver los anexos IV.b.1, IV.b.2 y IV.b.3.
Dentro de estos se encuentra el modo *Mission* con máquinas de estado homólogas a la de la figura 3.1.

3.4.3. Capa Reactiva

Ver IV.c.
En esta capa se implementa el AFEP, núcleo principal del proyecto, que se detalla en el siguiente apartado.

3.5. Automatic Flight Envelope Protection

El AFEP se introduce como una máquina incorporada a la *Capa Reactiva* que funciona como módulo supervisor de esta (figura 3.10).

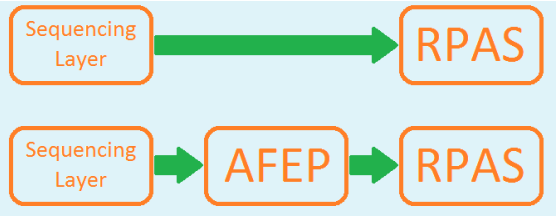

Figura 3.10: Introducción del AFEP en el sistema

Dado que la *Capa Reactiva* es la encargada de producir la salida de datos del Gestor de Misión, ésta es la única posición lógica para el AFEP. fig:machineAFEP
La implementación del AFEP pasa por modificar las salidas de datos directa que hace la *Capa Reactiva* para que pasen por un *buffer* legible por el AFEP.
Si el AFEP está activo, y detecta que los valores del buffer se encuentran por encima del límite definido en las funciones de las protecciones – que a su vez dependen de la altura, z, y la velocidad, V –, entonces aplica un regulador proporcional (K) que trata de compensar el exceso del parámetro medido (error, e) corrigiendo la salida de la capa.
Como se puede observar en la imagen 3.11, está compuesta por tres máquinas de estado. La primera de ellas monitoriza las condiciones de activación del AFEP,

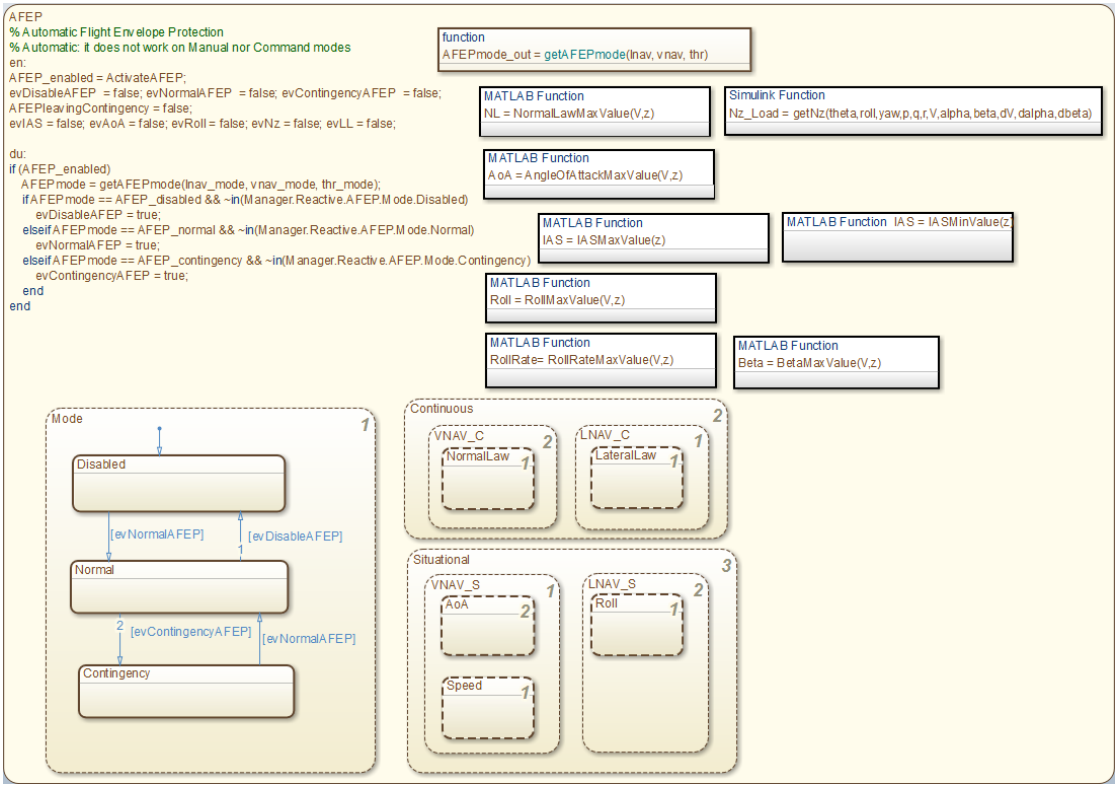

Figura 3.11: Máquina de estados AFEP dentro de la *Capa Reactiva*

minimizando el consumo de tiempo de procesamiento del sistema cuando el AFEP está inactivo. La segunda y la tercera corresponden respectivamente a las máquinas que gestionan las protecciones Continuas y Situacionales respectivamente, y su funcionamiento viene determinado por el bloque anterior

3.5.1. Mode

Constituye sencilla máquina de estados que puede alternar entre varios estados:

- **Disabled**: El AFEP no está funcionando y por tanto las capas de protección no se pueden activar. Este estado corresponde al vuelo en modo normal, cuando no se ha producido ningún fallo en los sistemas.
- **Normal** (Anexo V.a.1): El AFEP se encuentra **armado y en funcionamiento**, por tanto, la aeronave se rige por las límites impuestos por la *envolvente de vuelo segura* definida previamente.

- **Contingency** (Anexo V.a.2): El AFEP se encuentra **armado pero no aplica los límites**, esto significa que la aeronave ha perdido el enlace C2 y se ha producido alguna contingencia que requiere una maniobra correctiva urgente – como en el caso de alerta RA –, los límites de la aeronave vienen dados por sus propiedades físicas y aerodinámicas.

3.5.2. Protecciones Continuas

Las máquinas de estado de las protecciones Continuas tienen 3 estados, como se puede ver en las implementaciones de la Ley Normal (anexo V.b.1) y la Ley Lateral (anexo V.b.2).

- El primer de ellos, *Disabled*, es en el de inicialización, y no lleva acabo ninguna tarea. La transición al segundo estado ocurre en el primer ciclo, si se detecta que el AFEP se debe armar.
- El segundo, *Waiting*, calcula el valor de la derivada temporal, para que esté disponible en cualquier momento a partir de que el AFEP esté armado. La transición al tercer estado debe ocurrir si se produce una contingencia.
- En tal caso, el estado *Protection* entra en funcionamiento calculando el valor del factor de carga, n_z, y de la velocidad angular de alabeo, ω_x. Este mismo estado es el encargado de corregir las salidas de la *capa reactiva* si la aeronave sale de la envolvente de vuelo segura definida por el AFEP.

3.5.3. Protecciones Situacionales

Las protecciones situacionales tienen un diseño distinto, pero similar, basado en solo 2 estados, como se puede ver en las implementación de la protecciones del ángulo de ataque (anexo V.c.1), de velocidad máxima (anexo V.c.2) y del ángulo de alabeo (anexo V.c.3).

Esto se debe a que no es necesario un estado intermedio de espera en el que se calculen las derivadas temporales, por tanto los estados son:

- **Waiting**: es el estado de inicialización, y en el que se mantiene la máquina aunque esté armada. La transición al segundo estado se produce ante una contingencia.
- **Protecting**: es el estado capaz de sobrescribir la salida de la *capa reactiva* si se detecta que los parámetros de vuelo se encuentran fuera de la envolvente definida por el usuario.

3.5.4. Adaptabilidad de las Protecciones

La definición de envolvente de vuelo segura pueden variar sustancialmente de un RPAS a otro, y además, también puede tomar valores distintos en función de la altura y la velocidad a la que se encuentre. Por esta razón, se evita la definición de la envolvente de vuelo en el código fuente del AFEP.

En su lugar, se externaliza la definición de estas funciones a una serie de archivos de *Matlab* de modo que el proyecto las carga en su código cada vez que se inicia, permitiendo modificar su contenido con facilidad y simplificando el proceso de validación. El código que define la envolvente de vuelo segura mediante estas funciones se encuentra en el anexo V.d.

Capítulo 4

Validación del sistema

El último paso tras la implementación es comprobar que todos los sistemas funcionan.

Esto requiere previamente automatizar la ejecución de los vuelos, ya que el funcionamiento del software de simulación *X-Plane* trabajando en conjunto con el entorno Simulink (*Matlab*), sufre un pérdida de fidelidad al acelerar la simulación, el tiempo de simulación es razonablemente largo.

4.1. Automatizador de Misiones

El automatizador de misiones es un conjunto de programas de *Matlab* que se crean específicamente para esta tarea. Se compone de:

- Un **programa principal** en el que se define cada vuelo ejecutado, incluyendo la posición del avión, su actitud, su velocidad y las condiciones meteorológicas del vuelo, así como el control de la simulación pausando el software *X-Plane* durante la fase de compilación del código. Esto se realiza mediante una serie de funciones que explican en el apartado 4.1.1.
- Un **timer "watchdog"**, es decir, un detector que permite reiniciar el código del programa principal cuando la simulación ha terminado.

4.1.1. Funciones de Control de X-Plane

Mientras que ciertas funciones elementales de control de *X-Plane* ya estaban implementadas antes de empezar, otras tan sencillas como modificar la posición, la actitud o la velocidad de la aeronave, entre otras, no se encontraban implementadas.

A continuación se expone una lista de las funciones de control de *X-Plane* implementadas para este trabajo y, que se dejan a libre disposición de otros usuarios sin ninguna responsabilidad:

- *getPauseStatusXPLANE* (ver anexo VI.b.1)
  Detecta si simulador está en pausa o no.

- *togglePauseXPLANE* (ver anexo VI.b.2)
  Cambia el estado del simulador de Pausado a No-Pausado, y viceversa.

- *setPausePLANE* (ver anexo **??**)
  Mediante una combinación de las dos anteriores, permite al usuario elegir si quiere pausar o des-pausar el simulador. Si ya está en el estado deseado, no se realiza ninguna acción.

- *isScenaryLoadXPLANE* (ver anexo VI.b.4)
  Detecta si el simulador ha cargado los datos de topografía y gráficos del escenario.

- *getPosXPLANE* (ver anexo VI.b.5)
  Obtiene las coordenadas de la aeronave en el sistema local cartesiano (X, Y y Z) del sector.

- *setPosXPLANE* (ver anexo VI.b.6)
  Desplaza la aeronave a las coordenadas dadas según el sistema local cartesiano (X, Y y Z) del sector.

- *setCoordXPLANE* (ver anexo VI.b.7)
  Desplaza la aeronave a las coordenadas dadas según el sistema LLA.

- *setOrientationdXPLANE* (ver anexo VI.b.8)
  Establece la actitud y la velocidad de la aeronave de forma coherente.

- *setWeatherXPLANE* (ver anexo VI.b.9)
  Establece e interpola las condiciones de viento (dirección, altitud y velocidad).

- *fixPlaneXPLANE* (ver anexo VI.b.10)
  Repara los daños que haya sufrido la aeronave en el simulador.

4.2. Validación de Extended Path Terminators

4.2.1. SCAN

Como se puede observar en la figura 4.1, el procedimiento cumple con los requisitos de diseño (sección 2.2.1).

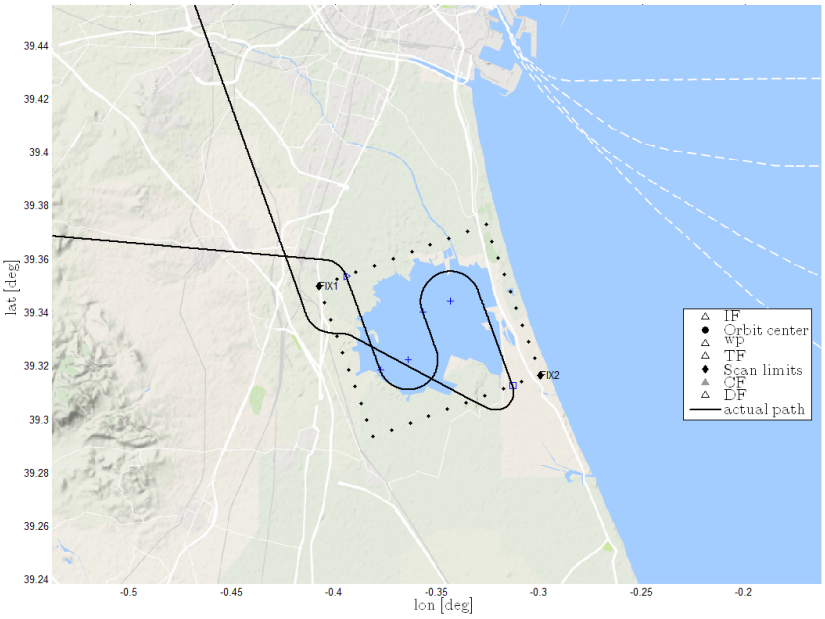

Figura 4.1: Ejecución del EPT SCAN

4.3. Prueba del sistema base

Para comprobar que el sistema funciona correctamente antes de la implementación se realizan 4 vuelos.

1. Vuelo completo del **Plan de vuelo principal**
2. Vuelo con pérdida del enlace C2 antes de llegar al waypoint SOBRO (**Plan alternativo 1**).
3. Vuelo con pérdida del enlace C2 entre el waypoint SOBRO y la duración de la maniobra SCAN (**Plan alternativo 2**).
4. Vuelo con pérdida del enlace C2 después la maniobra SCAN y la finalización del plan de vuelo (**Plan alternativo 3**).

4.3.1. Plan de Vuelo principal

A partir del Plan de Vuelo principal se obtiene abundante información de los sistemas y de los parámetros generales de funcionamiento de la aeronave (ésta continúa en la sección 4.3.5).

Debido a la implementación del EPT no se llega a realizar el aterrizaje, sin embargo, la realización del vuelo completo lleva a cabo la ruta que aparece en la figura 4.2.

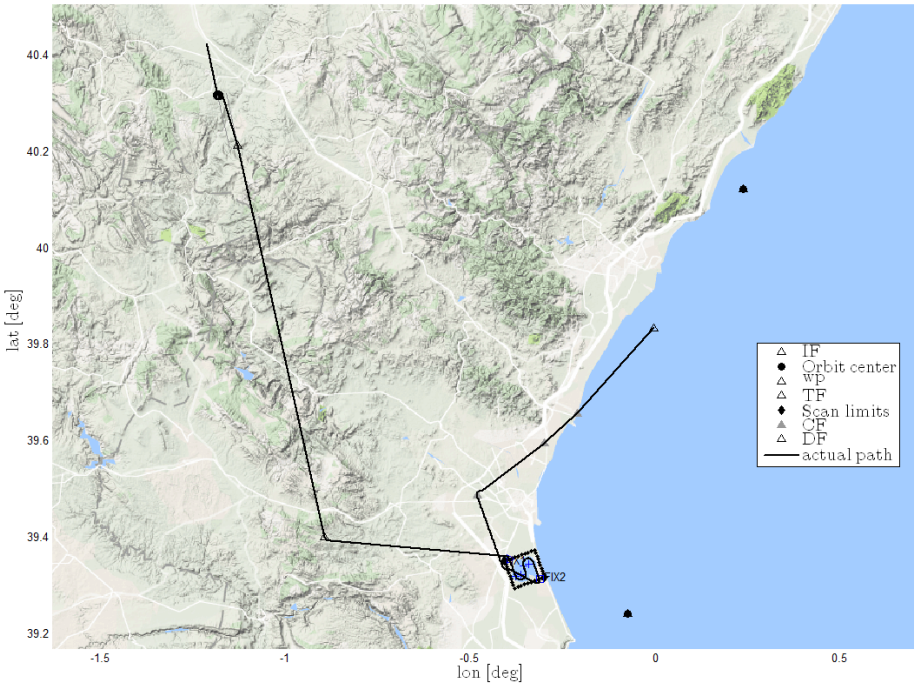

Figura 4.2: Perfil horizontal del Plan de Vuelo principal

4.3.2. Plan de Vuelo alternativo 1

En el Plan de Vuelo alternativo 1 se pierde el enlace C2 poco después del despegue, por tanto la aeronave queda orbitando en torno al waypoint definido, como se puede ver en la figura 4.3.

4.3.3. Plan de Vuelo alternativo 2

En el Plan de Vuelo alternativo 2 se pierde el enlace C2 poco después de pasar el waypoint SOBRO, por tanto la aeronave queda orbitando en torno al segundo

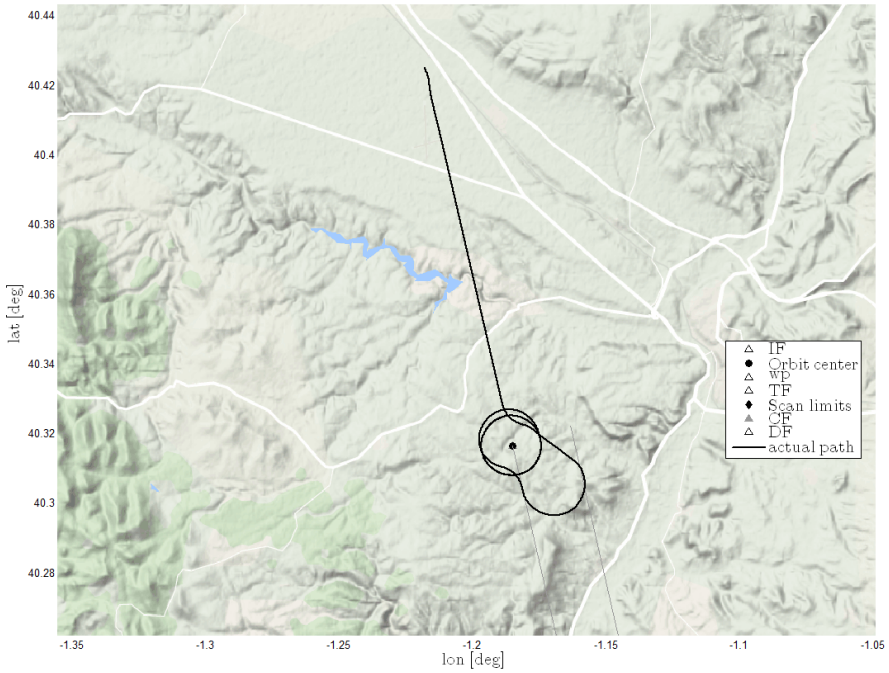

Figura 4.3: Perfil horizontal del Plan de Vuelo alternativo 1

punto de espera definido, cerca de La Albufera, figura 4.4.

4.3.4. Plan de Vuelo alternativo 3

En el Plan de Vuelo alternativo 3 se pierde el enlace C2 después de la maniobra SCAN sobre La Albufera,y el RPAS vuela hasta un punto cercano al aterrizaje, donde queda orbitando, ver figura 4.5.

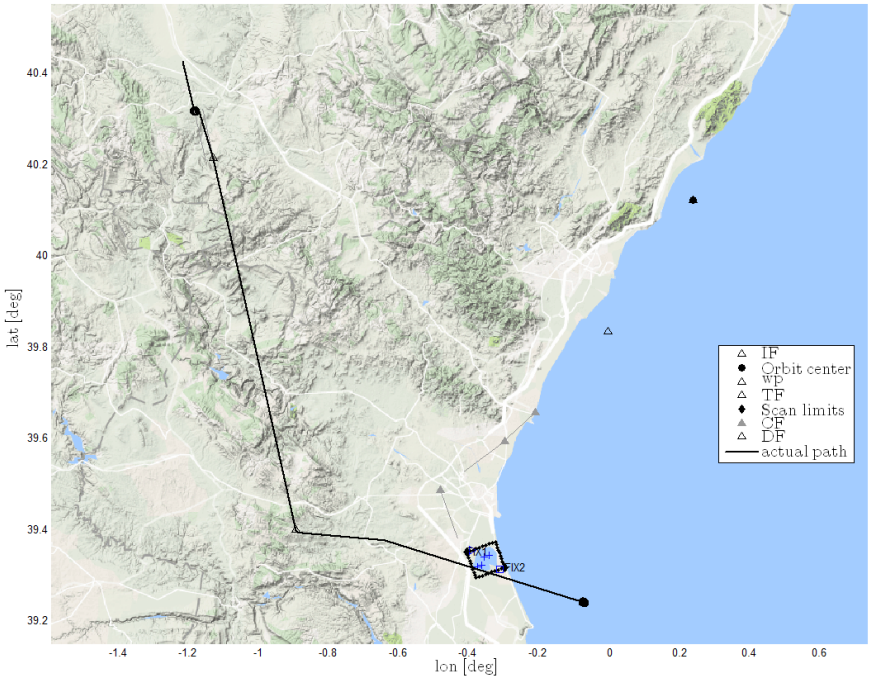

Figura 4.4: Perfil horizontal del Plan de Vuelo alternativo 2

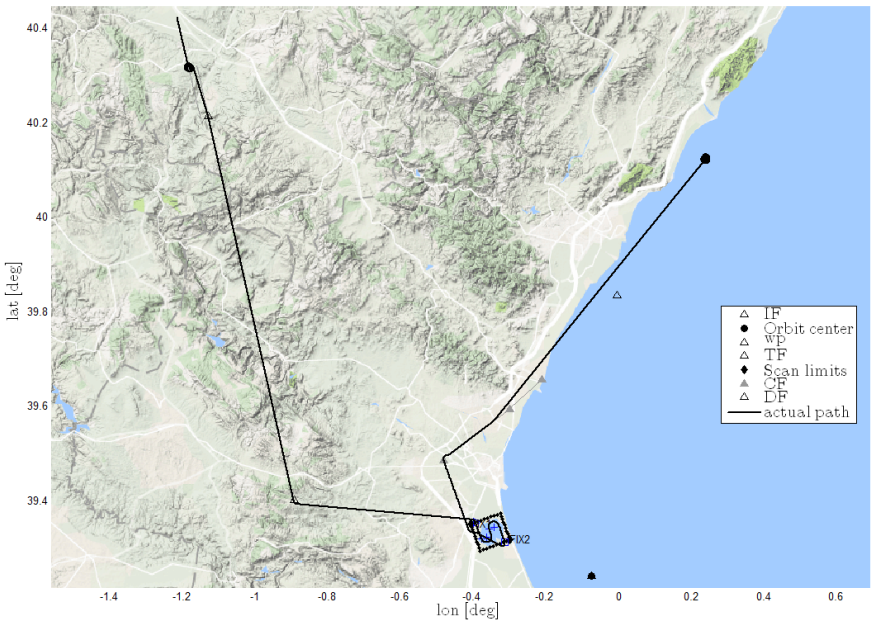

Figura 4.5: Perfil horizontal del Plan de Vuelo alternativo 3

4.3.5. Datos de vuelo (Plan de Vuelo principal)

Además del perfil horizontal, se obtienen otros datos de los vuelos que sirven para comprobar el funcionamiento del sistema. Estos se exponen a continuación:

Altitud

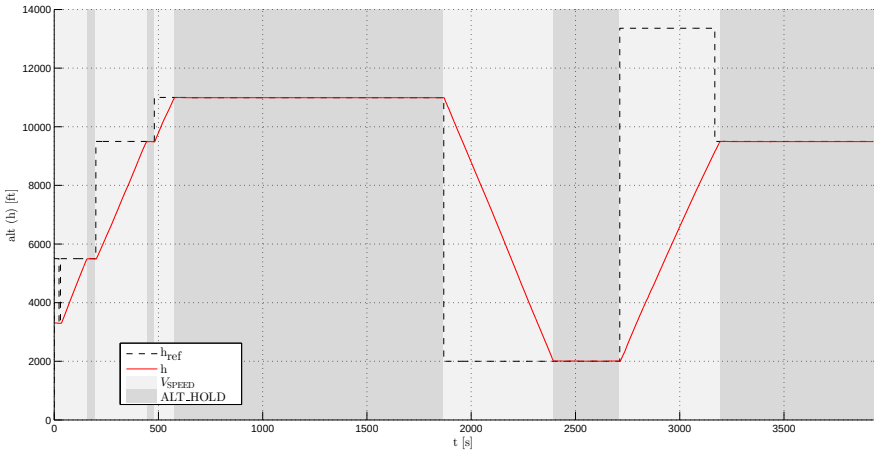

Figura 4.6: Perfil de alturas del Plan de Vuelo principal

Cabeceo, Ángulo de ataque y Ángulo de asiento horizontal

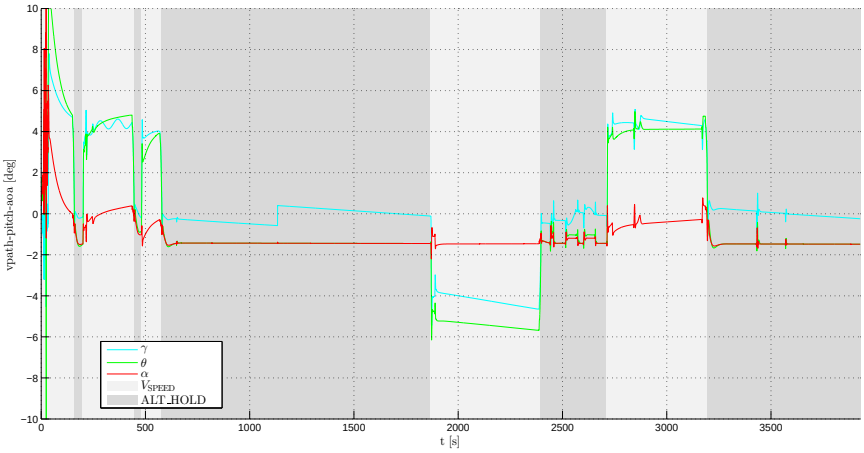

Figura 4.7: Ángulos del movimiento longitudinal del Plan de Vuelo principal

Velocidades

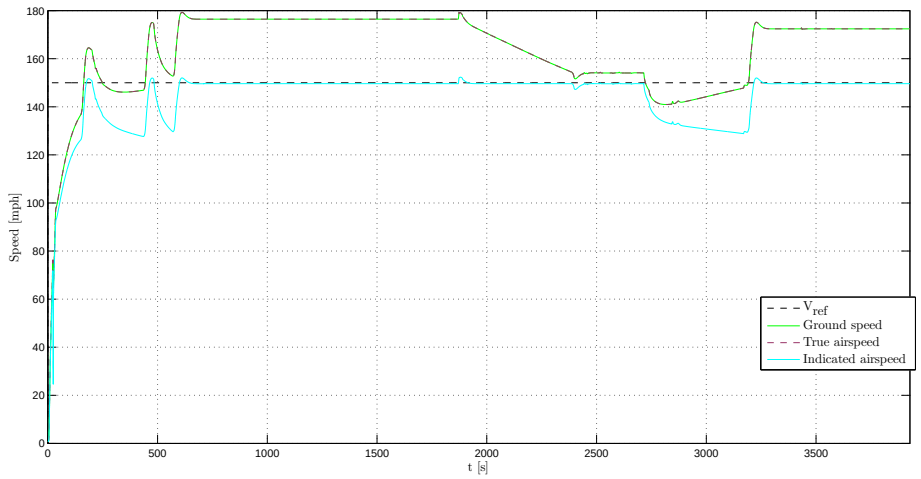

Figura 4.8: Velocidad Indicada, TAS, y Velocidad Proyectada del Plan de Vuelo principal

Velocidad vertical

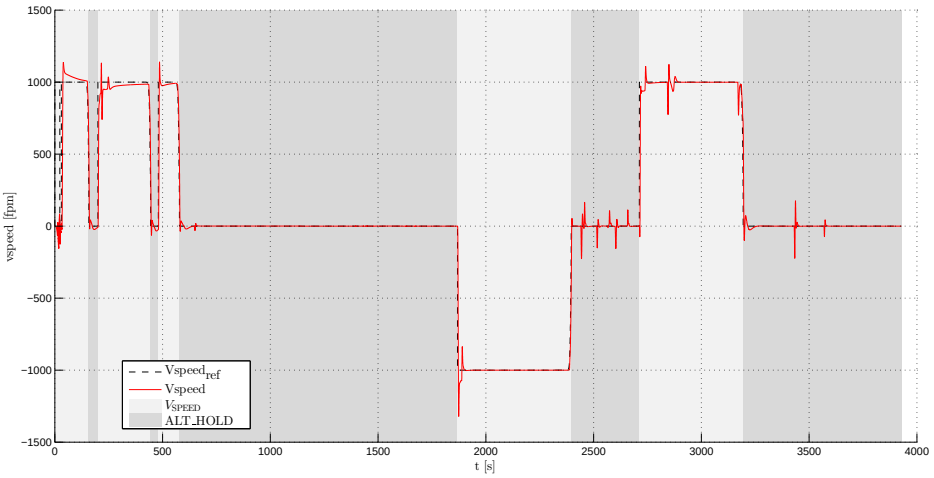

Figura 4.9: Velocidad vertical del Plan de Vuelo principal

Rumbo

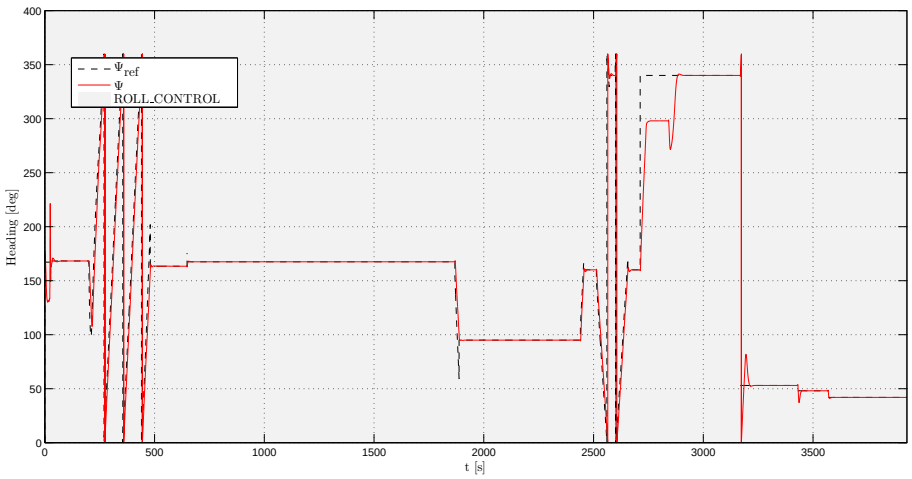

Figura 4.10: Rumbos del Plan de Vuelo principal

Ángulo de alabeo

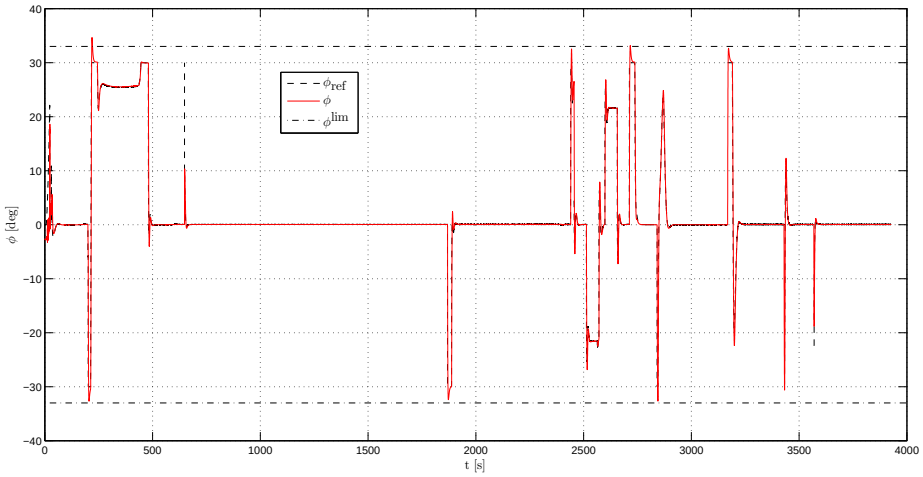

Figura 4.11: Ángulo de alabeo del Plan de Vuelo principal

Velocidad angular de alabeo

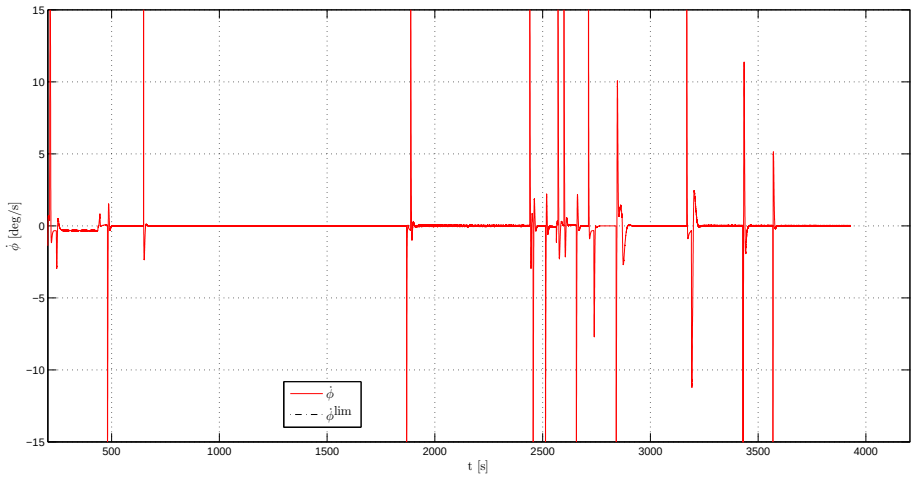

Figura 4.12: Velocidad angular de alabeo del Plan de Vuelo principal

4.4. Validación de los sistemas por separado

Para validar la protección de velocidad se llevan a cabo varios vuelos, en los que una vez pasada la fase de SCAN de La Albufera se simula la pérdida del enlace C2, de modo que se activa el Plan de Vuelo Alternativo 3.

La elección de esta zona viene dada por el hecho de que ofrece un recorrido largo y en linea recta que facilita la realización de las pruebas y un estudio limpio – sin interacciones con otros factores.

La validación por separado de los distintos sistemas, implica desactivar todas las capas de protección, salvo la que se quiere validar. Para lograr esto, se comenta el código correspondiente al evento que activa estas capas.

4.4.1. Protección de velocidad

Una vez está el avión posicionado y se encuentra en el **modo Normal** del AFEP, se provocan – a través de las funciones definidas en *Matlab* – varios picos de velocidad, para comprobar que el sistema funciona correctamente.

El resultado de la validación se muestra en la figura 4.13, donde se puede observar con gran detalle los saltos de velocidad introducidos virtualmente en

el simulador, así como la corrección de las referencias realizada por la capa de protección de velocidad.

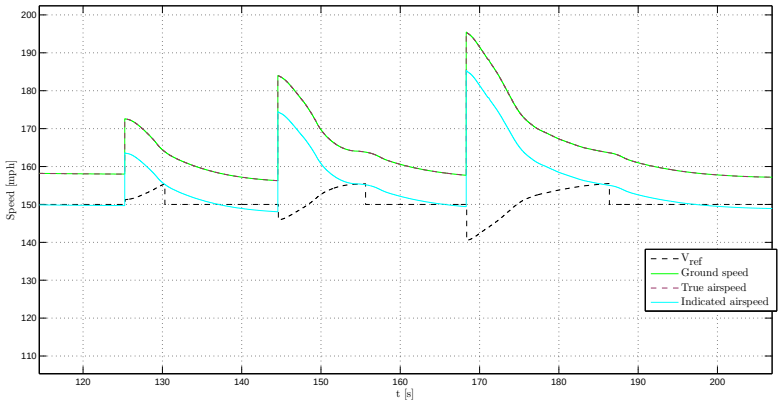

Figura 4.13: Capa de protección de velocidad

4.4.2. Protección del Ángulo de Ataque

De nuevo, con el avión posicionado y en el tramo anterior, se provoca mediante las funciones de *Matlab* que la aeronave aumente instantáneamente su ángulo de cabeceo, con la consecuencia directa de que el ángulo de ataque también aumenta.

Se puede observar en la figura 4.14 como la capa de protección del Ángulo de Ataque corrige en un breve periodo de tiempo los picos introducidos virtualmente en el ángulo de ataque.

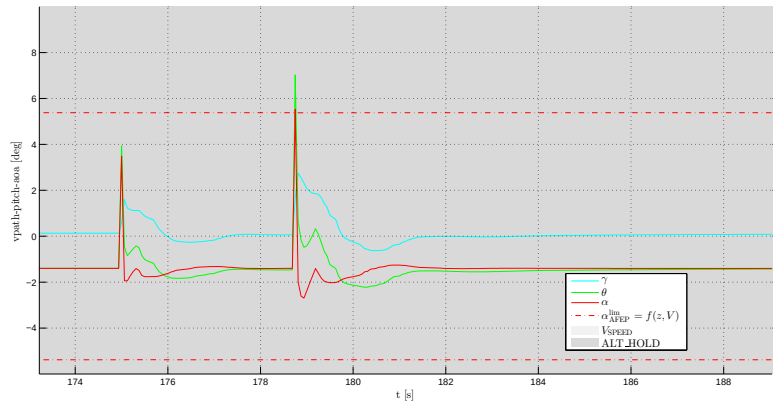

Figura 4.14: Capa de protección del ángulo de ataque

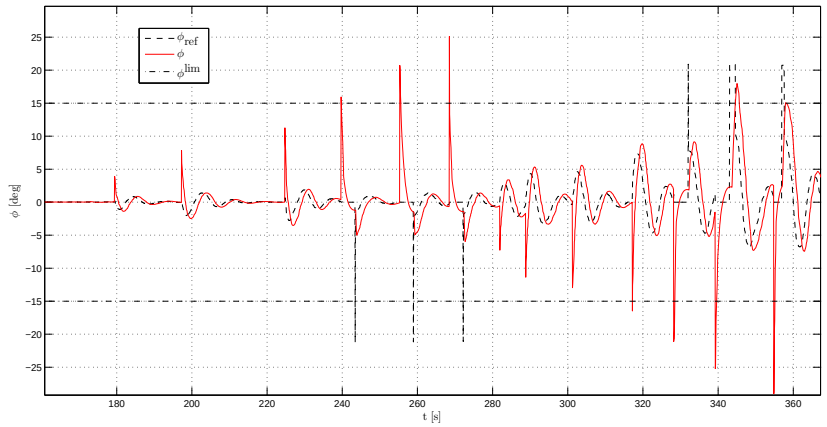

Figura 4.15: Capa de protección del ángulo de alabeo

4.4.3. Protección del Ángulo de Alabeo

De forma similar a los dos casos anteriores, se fuerza una variación instantánea del ángulo de alabeo mediante las funciones de *Matlab*.

En este caso, se puede observar en la figura 4.15, tanto la corrección positiva como la negativa de la referencia del ángulo de alabeo ante los sucesivos picos virtuales, introducidos de forma creciente.

4.4.4. Ley Normal, n_z

Para validar el correcto funcionamiento de la protección del factor de carga normal, se modifica la velocidad angular de cabeceo, ya que su efecto es bastante importante al ir multiplicado por la velocidad (ver ecuación 1.1).

El resultado de las pruebas se muestra en la figura 4.16, y en este se puede observar como, mediante la variación de la velocidad de ascenso/descenso, es posible reducir el valor del factor de carga normal, n_z.

4.4.5. Ley Lateral

La validación del la capa de protección *Ley Lateral*, requiere modificar el ángulo de alabeo de referencia, dado que esta protección actúa directamente sobre su derivada temporal, $\frac{d\phi}{dt}$.

Los resultados obtenidos se pueden se encuentran en la figura 4.17, donde se observa cómo el valor de la velocidad angular de alabeo retorna rápidamente a valores nominales después del pico introducido virtualmente.

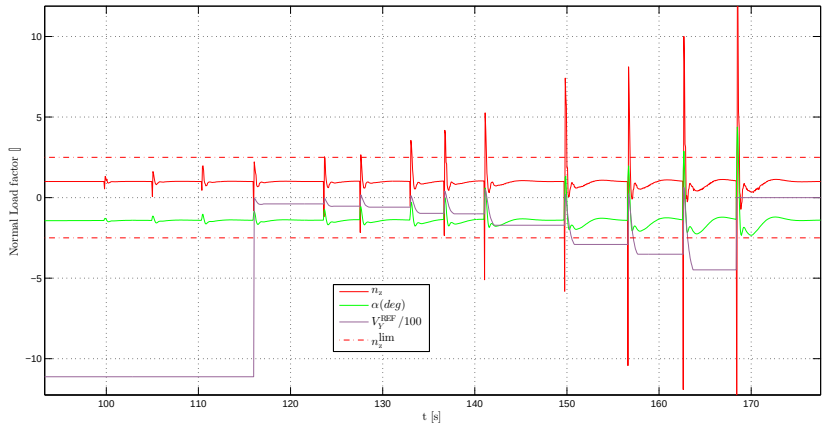

Figura 4.16: Capa de protección de la Ley Normal (factor de carga normal) y otras variables

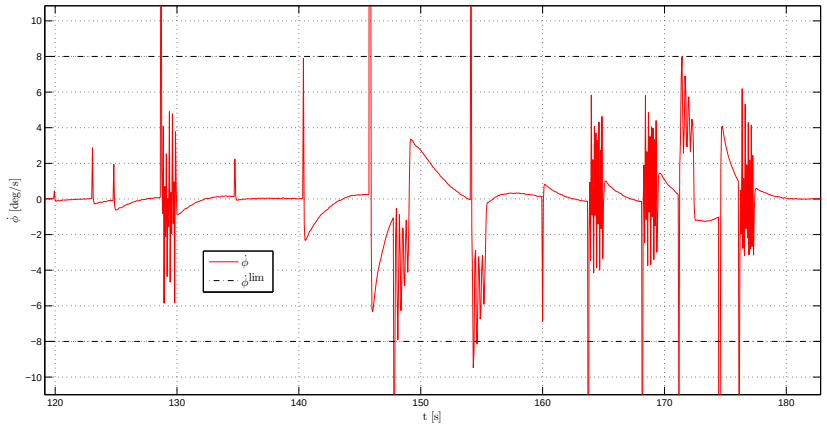

Figura 4.17: Capa de protección de la Ley Lateral (velocidad angular de alabeo)

Capítulo 5

Resumen de resultados

En este capítulo se resumen los resultados del proyecto.

5.1. Diseño

Se ha diseñado teóricamente los siguientes conceptos:

1. Planes de vuelo alternativos
    a) EPT: SCAN
2. Transición entre planes de vuelo
    a) Incorportación desde distintos puntos
3. Automatic Flight Envelope Protection
    a) Protección de velocidad
    b) Protección del ángulo de ataque
    c) Protección del ángulo de alabeo
    d) Protección del factor de carga normal
    e) Protección de la velocidad angular de alabeo

5.2. Implementación

Se han implementado los siguientes sistemas y programas:

1. Planes de vuelo alternativos

   a) EPT: SCAN (cálculo de las coordenadas de los puntos)

2. Automatic Flight Envelope Protection

   a) Protección de velocidad
   b) Protección del ángulo de ataque
   c) Protección del ángulo de alabeo
   d) Protección del factor de carga normal
   e) Protección de la velocidad angular de alabeo

3. + Validación

   a) Interfaz extendida de funciones de *Matlab* para *X-Plane*
   b) Simulador de múltiples Planes de Misión sobre *X-Plane*

5.3. Validación

Se han validado las siguientes implementaciones:

1. Planes de vuelo alternativos

   a) EPT: SCAN

2. Automatic Flight Envelope Protection

   a) Protección de velocidad
   b) Protección del ángulo de ataque
   c) Protección del ángulo de alabeo
   d) Protección del factor de carga normal
   e) Protección de la velocidad angular de alabeo

Capítulo 6

Recursos y Presupuesto

En este último capítulo se resumen los materiales, el software, el hardware y el personal asociado al desarrollo del proyecto, así como sus costes directos, y un resumen del presupuesto total.

A este se debe añadir el coste del trabajo previo realizado por Hèctor Usach y Borja Fons, el cual es desconocido.

Algunos de los costes, se encuentran originalmente en dólares americanos, en estos casos se mostrará el coste original entre paréntesis. El cambio de moneda utilizado es el disponible a fecha de 01/09/2016 (09:07 UTC) [27], y que corresponde a:

$$1\ USD \rightarrow 0{,}897142\ EUR \tag{6.1}$$

6.1. Desglose de gastos

- Espacios:

Descripción	Cantidad Requerida	Coste total
Aulas	1	No calculado

Tabla 6.1: Espacios empleados

- Ordenador de alta capacidad con Mandos de control:

Descripción	Coste total
Saitek Pro Flight Yoke System	134.00 €
Intel Core i5-6400 2.7GHz	175.00 €
Placa Base	159.00 €
GeForce GTX 750 Ti 2GB DDR5 (Tarjeta Gráfica)	126.00 €
Disco duro SSD 250GB	71.00 €
Monitor 1	69.95 €
Monitor 2	69.95 €
Fuente de alimentación	46.00 €
Disco duro HDD 1TB	44.95 €
Memoria Ram 8GB	37.00 €
Caja	23.95 €
Ventilador (Cooler)	19.95 €
Pack Teclado y Ratón	17.50 €
Total	924.30 €

Tabla 6.2: Coste del equipo informático (Hardware)

- Software Utilizado:

Descripción	Coste total
Entorno de desarrollo y compilador de textos, LaTeX	0.00 €
Editor de Imágenes, InkScape	0.00 €
X-Plane	53.81 € (59.99 $)
Matlab	2,000.00 €
Simulink (*Matlab*)	3,000.00 €
StateFlow (*Matlab*)	2,850.00 €
Total	7,903.81 €

Tabla 6.3: Espacios empleados

- Personal específico:

  Para los cálculos se emplea un valor de 1,840 horas trabajadas al año.

Descripción	Coste por hora	Número de horas	Coste Total
Profesor catedrático (40,558.28 €/año)	22.04 €/h	30 h	661.28 €
Ingeniero Técnico (21,000.00 €/año)	11.41 €/h	360 h	4,108.70 €
Total		4,769.98 €	

Tabla 6.4: Coste de personal

6.2. Resumen de presupuesto

Finalmente el resumen de los gastos es el siguiente:

Descripción	Coste
Coste del Hardware	924.30 €
Coste del Software	7,903.81 €
Coste del Personal	4,769.98 €
Total final	13,598.09 €

Tabla 6.5: Resumen del Presupuesto

El presupuesto de este proyecto asciende a 13,598.09 euros.

ANEXOS

Anexo I

Simulación

I.a. Ficha técnica IAI Super Heron

I.a.1. Características de la aeronave

Tripulación	2 (en tierra)
Longitud	8.5 m
Longitud alar	16.6 m
Maximum Take-Off Weight (MTOW)	1150 Kg
Motor	Rotax 914 (115 cv)

Tabla I.1: Características generales del IAI Super Heron [15]

I.a.2. Capacidad Operacional

Velocidad Máxima	113 kt (V)
Rango	189 MN
Techo de vuelo	FL 325
Tiempo máximo de vuelo	52 h
Carga de pago	250 Kg

Tabla I.2: Capacidad Operacional del IAI Super Heron [15]

Anexo II

Definición del Plan de Misión

II.a. Matlab: Plan de Vuelo Principal

```
% Plan de Mision de validacion
% Parametros iniciales
m_fuel      = 272; %Fuel weight (kg)
NUM_MPLANS = 4;    %Number of Stages
NUM_STAGES = 9;    %Number of Mission Plans
missionDefinition = NaN(NUM_STAGES, LONGTRAMA, NUM_MPLANS);
load chart_data_cv.mat;

%------------------------ MAIN PLAN ------------------------
PLAN=1; k=0;

k=k+1;
% IF RPAS loiter point
missionDefinition(k, CODING,    PLAN) = IF;
missionDefinition(k, FIX1LAT,   PLAN) = 40.3164;
missionDefinition(k, FIX1LON,   PLAN) = -1.1856;
missionDefinition(k, FIX1ALT,   PLAN) = 5500;
missionDefinition(k, LIMSPEED, PLAN) = 150/mph2knot;

k=k+1;
% ORBIT TO ALT: min altitude for AWY R29
missionDefinition(k, CODING,    PLAN) = ORBIT;
missionDefinition(k, FIX1LAT,   PLAN) = 40.3164;
missionDefinition(k, FIX1LON,   PLAN) = -1.1856;
missionDefinition(k, RADIUS,    PLAN) = .5; %nm
```

```
26 missionDefinition(k, DIR,       PLAN) = 1; %CW
27 missionDefinition(k, ENDTYPE,   PLAN) = ALT_TYPE;
28 missionDefinition(k, LIMVALUE,  PLAN) = 9500;
29
30 k=k+1;
31 %FLYTO intercept AWY R29, CMA169040/N0150F110
32 [lat,lon] = orto_reckon(navaid_cma.lat, navaid_cma.lon, 40*
       nm2m, 169);
33 [~, crs] = orto_distazi(lat, lon, wp_sobro.lat, wp_sobro.
       lon);
34 missionDefinition(k, CODING,    PLAN) = FLYTO;
35 missionDefinition(k, FIX1LAT,   PLAN) = lat;
36 missionDefinition(k, FIX1LON,   PLAN) = lon;
37 missionDefinition(k, COURSE,    PLAN) = crs;
38 missionDefinition(k, FIX1ALT,   PLAN) = 11000;
39 missionDefinition(k, FIX2ALT,   PLAN) = 11000;
40 missionDefinition(k, LIMSPEED, PLAN) = 150/mph2knot;
41
42 k=k+1;
43 %TF to SOBRO (antes de llegar)
44 missionDefinition(k, CODING,    PLAN) = TF;
45 missionDefinition(k, FIX1LAT,   PLAN) = wp_sobro.lat;
46 missionDefinition(k, FIX1LON,   PLAN) = wp_sobro.lon;
47 missionDefinition(k, LIMSPEED, PLAN) = 150/mph2knot;
48
49 k=k+1;
50 %SCAN Albufera
51 missionDefinition(k, CODING,    PLAN) = SCAN;
52 missionDefinition(k, FIX1LAT,   PLAN) = dms2degrees([39 21
       00]);
53 missionDefinition(k, FIX1LON,   PLAN) = dms2degrees([0 -24
       30]);
54 missionDefinition(k, FIX2LAT,   PLAN) = dms2degrees([39 19
       00]);
55 missionDefinition(k, FIX2LON,   PLAN) = dms2degrees([0 -18
       00]);
56 missionDefinition(k, COURSE,    PLAN) = 160;
57 missionDefinition(k, RADIUS,    PLAN) = 1000/1852; %NM
58 missionDefinition(k, FIX1ALT,   PLAN) = 2000;
59 missionDefinition(k, LIMSPEED, PLAN) = 100;
```

```
60
61 k=k+1;
62 %CF Pasillo visual Sollana
63 missionDefinition(k, CODING,    PLAN) = CF;
64 missionDefinition(k, FIX1LAT,   PLAN) = navaid_vlc.lat;
65 missionDefinition(k, FIX1LON,   PLAN) = navaid_vlc.lon;
66 missionDefinition(k, COURSE,    PLAN) = 340;
67 missionDefinition(k, FIX1ALT,   PLAN) = 6500+
       get_terrain_elevation(navaid_vlc.lat, navaid_vlc.lat);
68 missionDefinition(k, FIX2ALT,   PLAN) = 19500;
69 missionDefinition(k, LIMSPEED, PLAN) = 130/mph2knot;
70
71 k=k+1;
72 %CF Pasillo visual El Puig
73 missionDefinition(k, CODING,    PLAN) = CF;
74 missionDefinition(k, FIX1LAT,   PLAN) = sex2dec('393538N');
75 missionDefinition(k, FIX1LON,   PLAN) = sex2dec('0001801W');
76 missionDefinition(k, COURSE,    PLAN) = 53;
77 missionDefinition(k, FIX1ALT,   PLAN) = 9500;
78 missionDefinition(k, FIX2ALT,   PLAN) = 19500;
79 missionDefinition(k, LIMSPEED, PLAN) = 130/mph2knot;
80
81 k=k+1;
82 %CF Pasillo visual Sagunto
83 missionDefinition(k, CODING,    PLAN) = CF;
84 missionDefinition(k, FIX1LAT,   PLAN) = sex2dec('393922N');
85 missionDefinition(k, FIX1LON,   PLAN) = sex2dec('0001244W');
86 missionDefinition(k, COURSE,    PLAN) = 48;
87 missionDefinition(k, FIX1ALT,   PLAN) = 9500;
88 missionDefinition(k, FIX2ALT,   PLAN) = 19500;
89 missionDefinition(k, LIMSPEED, PLAN) = 130/mph2knot;
90 %
91 %%
92 k=k+1;
93 %FLYTO SOPET
94 missionDefinition(k, CODING,    PLAN) = FLYTO;
95 missionDefinition(k, FIX1LAT,   PLAN) = wp_sopet.lat;
96 missionDefinition(k, FIX1LON,   PLAN) = wp_sopet.lon;
97 missionDefinition(k, FIX1ALT,   PLAN) = 9500;
98 missionDefinition(k, LIMSPEED, PLAN) = 130/mph2knot;
```

II.b. Matlab: Plan de Vuelo Alternativo 1

```
%------------------- ALTERNATIVE PLANS -------------------
%------------------- Contingency Plan 1 -------------------
PLAN=2; k=0;

k=k+1;
%CF RPAS loiter point
missionDefinition(k, CODING,    PLAN) = CF;
missionDefinition(k, FIX1LAT,   PLAN) = 40.3164;
missionDefinition(k, FIX1LON,   PLAN) = -1.1856;
missionDefinition(k, FIX1ALT,   PLAN) = 5500;
missionDefinition(k, COURSE,    PLAN) = 167+180;

k=k+1;
%ORBIT until TIMEOUT occurs
missionDefinition(k, CODING,    PLAN) = ORBIT;
missionDefinition(k, FIX1LAT,   PLAN) = 40.3164;
missionDefinition(k, FIX1LON,   PLAN) = -1.1856;
missionDefinition(k, RADIUS,    PLAN) = .5; %nm
missionDefinition(k, DIR,       PLAN) = 1; %CW
missionDefinition(k, ENDTYPE,   PLAN) = TIME_TYPE;
missionDefinition(k, LIMVALUE,  PLAN) = 500; %s
```

II.c. Matlab: Plan de Vuelo Alternativo 2

```
%------------------- Contingency Plan 2 -------------------
PLAN=3; k=0;

k=k+1;
%DF to RPAS loiter point
missionDefinition(k, CODING,    PLAN) = DF;
missionDefinition(k, FIX1LAT,   PLAN) = 39.24;
missionDefinition(k, FIX1LON,   PLAN) = -0.075;
missionDefinition(k, FIX1ALT,   PLAN) = 3000;
missionDefinition(k, FIX2ALT,   PLAN) = 5500;

k=k+1;
%ORBIT trying to regain link
missionDefinition(k, CODING,    PLAN) = ORBIT;
```

```
missionDefinition(k, FIX1LAT,   PLAN) = 39.24;
missionDefinition(k, FIX1LON,   PLAN) = -0.075;
missionDefinition(k, RADIUS,    PLAN) = .5; %nm
missionDefinition(k, DIR,       PLAN) = 1; %CW
missionDefinition(k, ENDTYPE,   PLAN) = TIME_TYPE;
missionDefinition(k, LIMVALUE, PLAN) = 200; %s
```

II.d. Matlab: Plan de Vuelo Alternativo 3

```
%------------------ Contingency Plan 3 ------------------
PLAN=4; k=0;

k=k+1;
%DF to RPAS loiter point
missionDefinition(k, CODING,    PLAN) = DF;
missionDefinition(k, FIX1LAT,   PLAN) = 40.1224;
missionDefinition(k, FIX1LON,   PLAN) = 0.2393;
missionDefinition(k, FIX2ALT,   PLAN) = 5000;

k=k+1;
%ORBIT trying to regain link
missionDefinition(k, CODING,    PLAN) = ORBIT;
missionDefinition(k, FIX1LAT,   PLAN) = 40.1224;
missionDefinition(k, FIX1LON,   PLAN) = 0.2393;
missionDefinition(k, RADIUS,    PLAN) = .5; %nm
missionDefinition(k, DIR,       PLAN) = 1; %CW
missionDefinition(k, ENDTYPE,   PLAN) = TIME_TYPE;
missionDefinition(k, LIMVALUE, PLAN) = 200; %s

k=k+1;
%ORBIT to flight termination
missionDefinition(k, CODING,    PLAN) = ORBIT;
missionDefinition(k, FIX1LAT,   PLAN) = 40.1224;
missionDefinition(k, FIX1LON,   PLAN) = 0.2393;
missionDefinition(k, RADIUS,    PLAN) = .5; %nm
missionDefinition(k, DIR,       PLAN) = 1; %CW
missionDefinition(k, ENDTYPE,   PLAN) = ALT_TYPE;
missionDefinition(k, LIMVALUE, PLAN) = 0; %s
```

II.e. Matlab: Transiciones entre Planes de Vuelo

```
%————————— Mission Plan transition table —————————
% Transition conditions
stageVector = [4  5  9];
fteVector   = nan;

% Transition matrix
missionTransitionTable = NaN(length(fteVector)+1, length(
    stageVector)+1);
missionTransitionTable(1,2:end) = stageVector;
missionTransitionTable(2:end,1) = fteVector;

% Mission plans IDs
missionTransitionTable(2:end,2:end) = [2  3  4];
```

Anexo III

Definición de Extended Path Terminators

III.a. Matlab: Cálculo de coordenadas de referencia de SCAN

```
function [point_list, Npoints, first_rhumb, first_turn, coverage] = scan(
    lat1,lon1,lat3,lon3,crs,turn_radius)
%Scanning pattern
%Computes a scanning pattern as a sequence of straight and turn legs
%around a square defined by vertices (lat1, lon1) and (lat2, lon2).
%crs (decimal degrees) is the main direction of the scanning pattern.
%turn_radius (m) is the radius of the turn leg. It can be seen also as the
%semi-area covered in each pass.
%The function returns a 32x2 matrix containing the coordinates of the
%points which define the pattern: the entry point, the destination point
%of each straight segment, the turn center of each turn leg, and the exit
%point in the form [lat(i) lon(i)]. The scanning pattern shall have a
%maximum number of 32 points.
%Npoints is the actual number of points which define the scanning pattern.
%first_rhumb (decimal degrees) is the rhumb of the first straight leg.
%first_turn is the direction of the first turn segment (CW=1, CCW=-1)
%coverage (m) is the turn_radius corrected to optimize the scanning area.
%
%Asensio Lorenzo Sempere 9/2/2016

%Defines
CONSTNPOINTS = 32;
CW = 1;
CCW = -1;
%-----------------------------------------------------------------
%Square obtention
[lat2, lon2, lat4, lon4] = loxo_pt2sq(lat1,lon1,lat3,lon3,crs);

%Obtain the references distances and angles
[dist12, bearing12] = loxo_dist_bearing(lat1,lon1,lat2,lon2);
[dist14, bearing14] = loxo_dist_bearing(lat1,lon1,lat4,lon4);

%Obtain number of LONgitudinal divisions
N_largos_prov = floor(dist14 / (2*turn_radius));

%Correct the number of divisions (even / odd)
if (rem(N_largos_prov,2) == 0) %Even: Ends the wrong corner
    N_largos    = N_largos_prov -1;          %Divisions are reduced
    cob_efectiva = dist14 / (2*N_largos);    %Range is increased
else                          %Odd: Ends the right corner
    N_largos    = N_largos_prov;
    cob_efectiva = dist14 / (2*N_largos);    %Appropiated range is covered
end
Npoints = 2*N_largos;
%-----------------------------------------------------------------
%Definitions

point_list = zeros(CONSTNPOINTS,2);
tempwaypoint = zeros(1,2);
```

```
49 rev_bearing12 = mod(bearing12+180,360);pos = 0;
50
51 % Waypoints
52 [point_list(1,1), point_list(1,2)] = loxo_destination(lat1,lon1,dist14/
       Npoints,bearing14);
53 for i = 2:(Npoints-1)
54     if rem(i,2)==0  % a?? el punto de tangencia
55         if (pos == 0)          % Primero
56             [tempwaypoint(1),tempwaypoint(2)] = loxo_destination(lat2,lon2,
                   dist14*(i-1)/Npoints,bearing14);
57             [point_list(i,1),point_list(i,2)] = loxo_destination(
                   tempwaypoint(1),tempwaypoint(2),2*cob_efectiva,rev_bearing12
                   );
58             pos = 1;
59         elseif (pos == 1)    % Tercero
60             [tempwaypoint(1),tempwaypoint(2)] = loxo_destination(lat1,lon1,
                   dist14*(i-1)/Npoints,bearing14);
61             [point_list(i,1),point_list(i,2)] = loxo_destination(
                   tempwaypoint(1),tempwaypoint(2),2*cob_efectiva,bearing12);
62             pos = 0;
63         end
64     else              % a?? el punto de centro de circunferencia
65         if (pos == 1)          % Segundo
66             [tempwaypoint(1),tempwaypoint(2)] = loxo_destination(lat2,lon2,
                   dist14*(i-1)/Npoints,bearing14);
67             [point_list(i,1),point_list(i,2)] = loxo_destination(
                   tempwaypoint(1),tempwaypoint(2),2*cob_efectiva,rev_bearing12
                   );
68         elseif (pos == 0)    % Cuarto
69             [tempwaypoint(1),tempwaypoint(2)] = loxo_destination(lat1,lon1,
                   dist14*(i-1)/Npoints,bearing14);
70             [point_list(i,1),point_list(i,2)] = loxo_destination(
                   tempwaypoint(1),tempwaypoint(2),2*cob_efectiva,bearing12);
71         end
72     end
73 end
74 [point_list(Npoints,1), point_list(Npoints,2)] = loxo_destination(lat3,lon3
       ,dist14/Npoints,mod(bearing14+180,360));
75
76 % Obtain the first turn
77 objective_brg23 = unwrap(bearing14-bearing12, 180);
78 if objective_brg23>=0
79     first_turn = CW;
80 else
81     first_turn = CCW;
82 end
83
84 % First rhumb
85 first_rhumb = bearing12;
```

```
86
87 %Turn radius corrected
88 coverage = cob_efectiva; %m
89 end
90
91 %%=====================================================
92 function [lat2,lon2,lat4,lon4] = loxo_pt2sq(lat1,lon1,lat3,lon3,bearing)
93 % bearing = [0, 360[
94 % Graphic representation of the resulting points
95 %  1      main_dist      2
96 %   o----------------------o
97 %   |-->->->->->->->->->\  |
98 %   |                    |  |
99 %   | /<-<-<-<-<-<-<--/  | secondary_dist
100 %  |  |                 |
101 %  |  \->->->->->->->-->|
102 %  o----------------------o
103 %  4                      3
104 bearing = mod(bearing,360);
105
106 if (bearing == 90 || bearing == 270)
107     lat2 = lat1;
108     lon2 = lon3;
109     lat4 = lat3;
110     lon4 = lon1;
111 elseif (bearing == 0 || bearing == 180)
112     lat2 = lat3;
113     lon2 = lon1;
114     lat4 = lat1;
115     lon4 = lon3;
116 else
117     %if (mod(bearing,90) ~= 0)
118     [lat2, lon2] = loxo_90intersection(lat1,lon1,lat3,lon3,bearing);
119     [lat4, lon4] = loxo_90intersection(lat1,lon1,lat3,lon3,mod(bearing
        +90,360));
120 end
121 end
122
123 %%=====================================================
124 function [lat3,lon3]=loxo_90intersection(lat1,lon1,lat2,lon2,crs13)
125 lat1 = lat1*pi/180;
126 lon1 = lon1*pi/180;
127
128 lat2 = lat2*pi/180;
129 lon2 = lon2*pi/180;
130
131 m = tand(crs13);
132 lon3 = ((lon2*m+lon1/m)-log(tan(pi/4+lat1/2)/tan(pi/4+lat2/2)))/(m+1/m);
133 lat3 = 2*atan(exp((m^2*log(tan(pi/4+lat1/2))+m*(lon2-lon1)+log(tan(pi/4+
```

```
    lat2/2)))/(1+m^2)))-pi/2;

lat3 = lat3*180/pi;
lon3 = lon3*180/pi;
end

%%=========================================================
function [lat_fin, lon_fin] = loxo_destination(lat_ini, lon_ini, dist, brg)
R = 6371000;
lat_ini = lat_ini*pi/180;
lon_ini = lon_ini*pi/180;
brg = brg*pi/180;

alpha = dist/R;
lat_fin = lat_ini + alpha*cos(brg);
dLat = lat_fin - lat_ini;
dLatp = log(tan(pi/4+lat_fin/2)/tan(pi/4+lat_ini/2));
if (brg == pi/2)
    q = cos(lat_ini);
else
    q = dLat/dLatp;
end
dLon = alpha*sin(brg)/q;
lon_fin = mod(lon_ini + dLon + pi, 2*pi)-pi;
if (lat_fin > pi/2)
    lat_fin = pi - lat_fin;
elseif (lat_fin < -pi/2)
    lat_fin = -pi - lat_fin;
end

lat_fin = lat_fin*180/pi;
lon_fin = lon_fin*180/pi;
end

%%=========================================================
function [dist, brg] = loxo_dist_bearing(lat_ini, lon_ini, lat_fin, lon_fin
    )
lat_ini=lat_ini*pi/180;
lon_ini=lon_ini*pi/180;
lat_fin=lat_fin*pi/180;
lon_fin=lon_fin*pi/180;
Dphi = log(tan(pi/4+lat_fin/2)/tan(pi/4+lat_ini/2));
if lat_ini==lat_fin
    q=cos(lat_ini);
else
    q=(lat_fin-lat_ini)/(Dphi);
end
dlon = (lon_fin-lon_ini);
if dlon > pi
```

```
181     dlon = dlon-2*pi;
182 elseif dlon < -pi
183     dlon = dlon+2*pi;
184 end
185 dist = sqrt((lat_fin-lat_ini)^2 + q^2*dlon^2)*6371000;
186 brg  = atan2(dlon, (Dphi))*180/pi;
187 if(brg<0)
188     brg = brg + 360;
189 end
190 end
191
192 %%=====================================================
193 function out = unwrap(in, limit)
194 out = in;
195 if out > limit
196     out = out-2*limit;
197 end
198 if out < -limit
199     out = out+2*limit;
200 end
201 end
```

Anexo IV

Mission Manager

IV.a. Capa Deliberativa

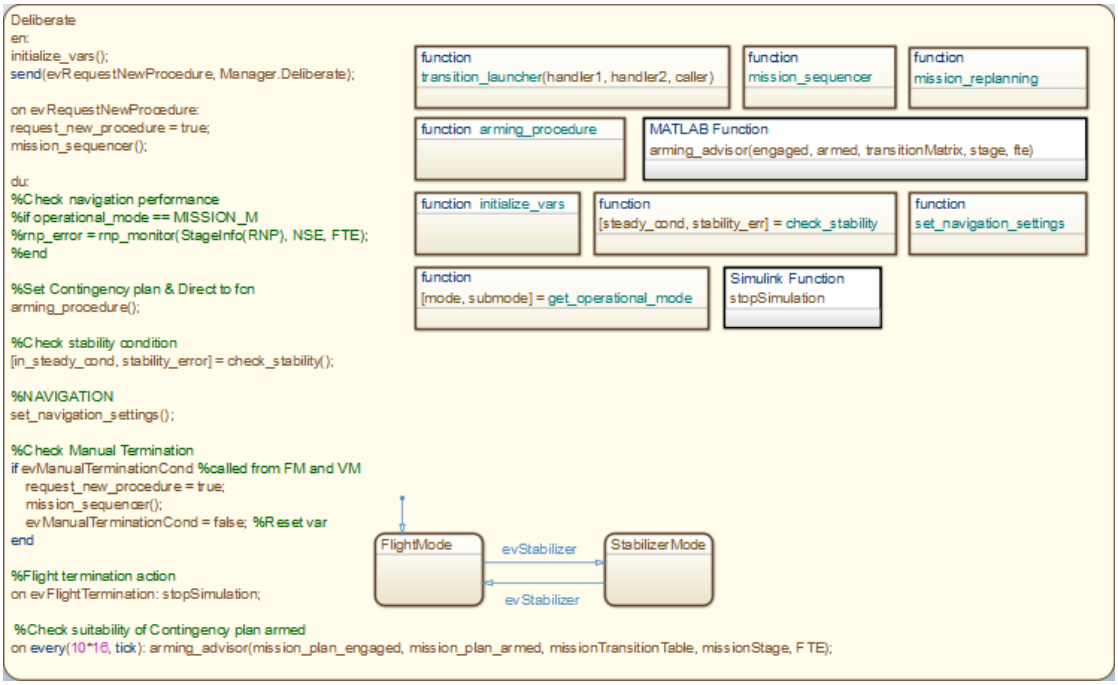

Figura IV.1: Máquina de estados Capa Deliberativa

IV.b. Capa Secuenciadora

Figura IV.2: Máquina de estados Capa Secuenciadora

IV.b.1. LNAV

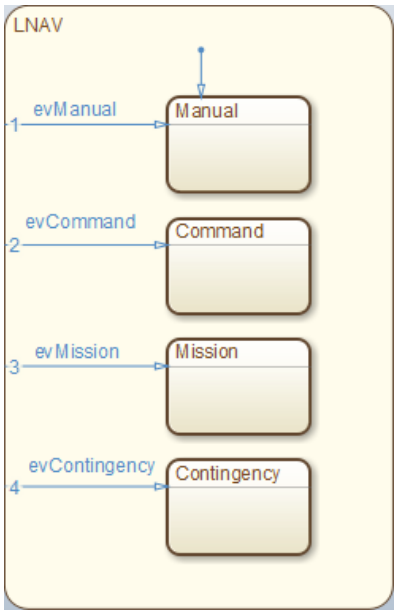

Figura IV.3: Modo LNAV de la capa Secuenciadora

IV.b.2. VNAV

Figura IV.4: Modo VNAV de la capa Secuenciadora

IV.b.3. THR

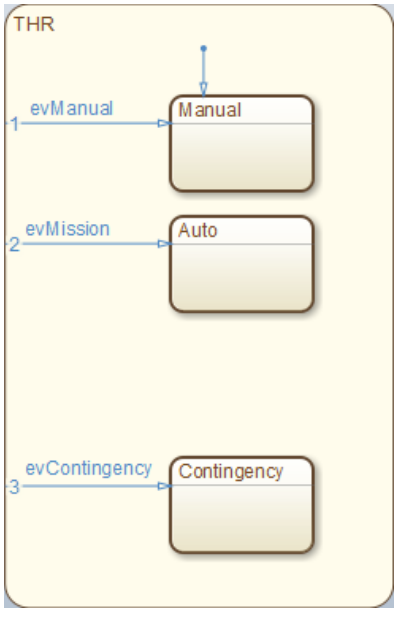

Figura IV.5: Modo THR de la capa Secuenciadora

IV.c. Capa Reactiva

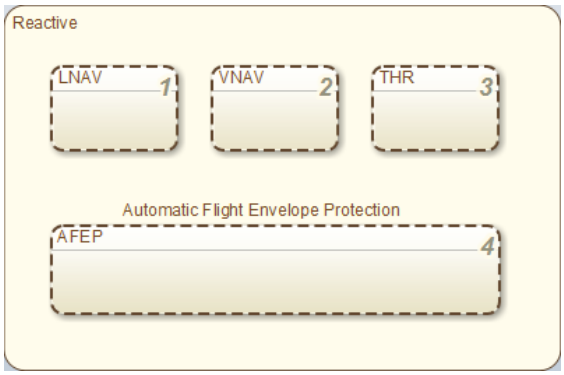

Figura IV.6: Máquina de estados Capa Reactiva

IV.c.1. THR

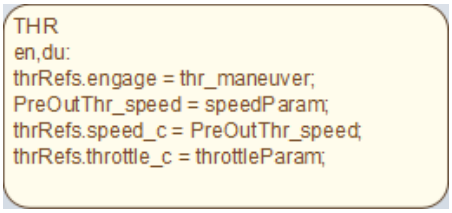

Figura IV.7: Modo THR de la capa Reactiva

IV.c.2. LNAV

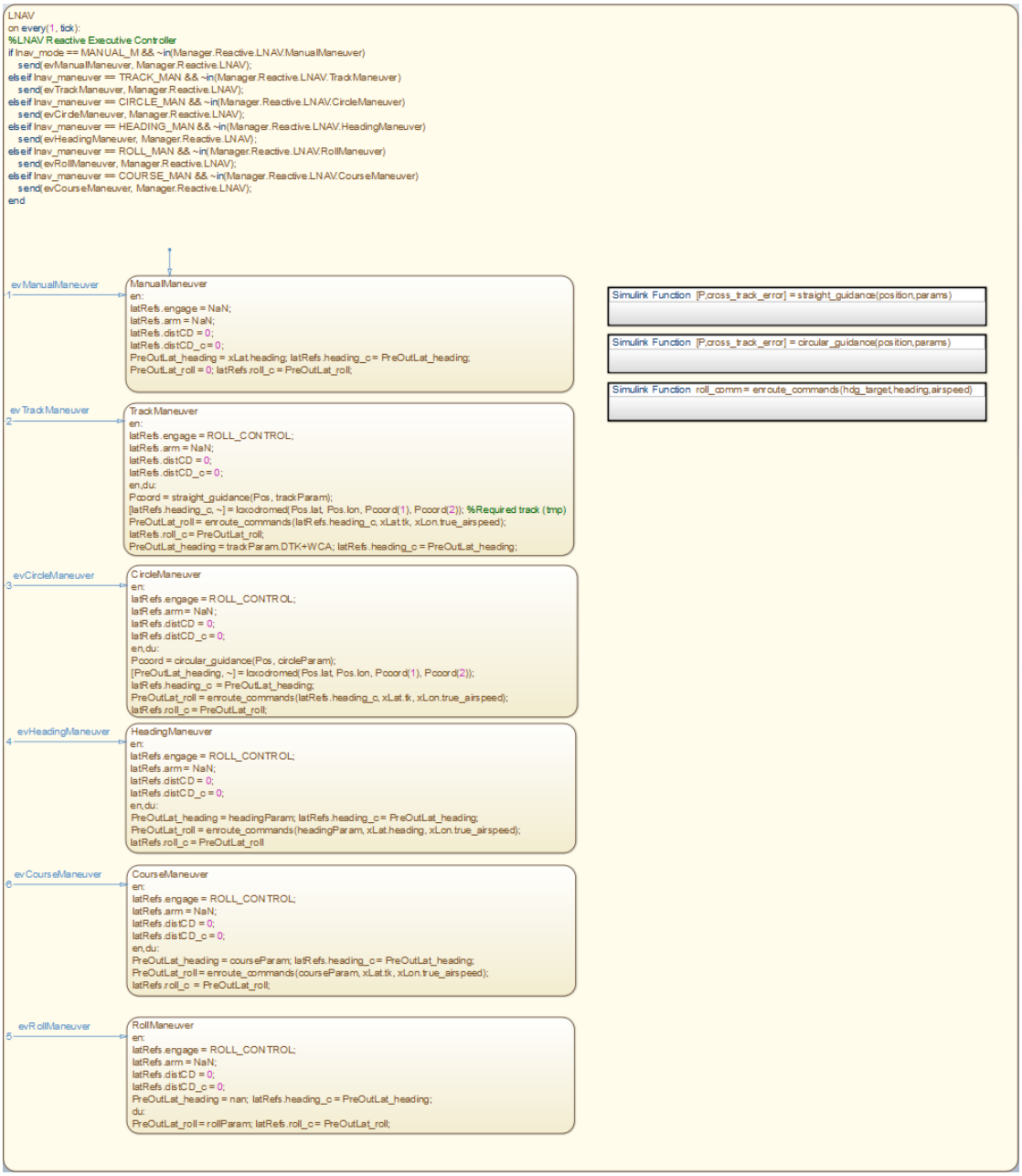

Figura IV.8: Modo LNAV de la capa Reactiva

IV.c.3. VNAV

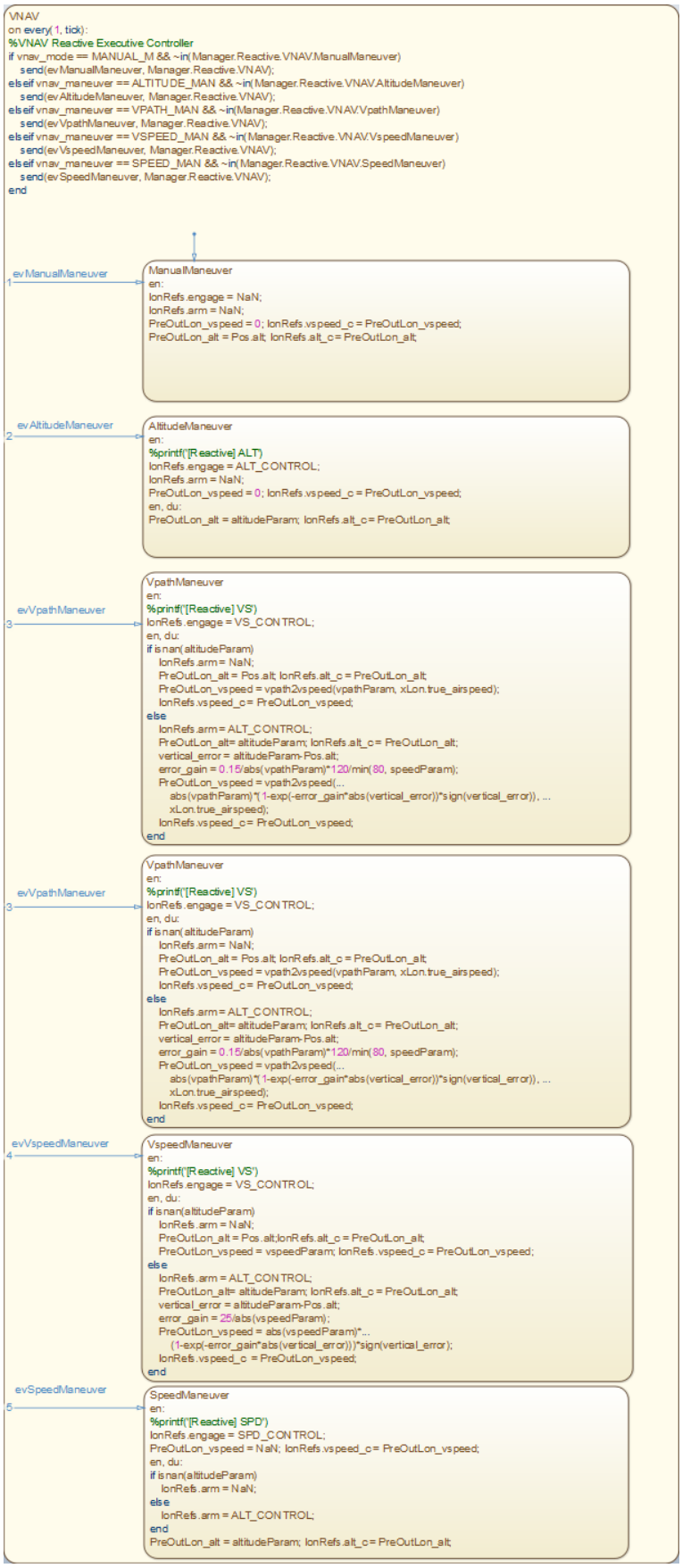

Figura IV.9: Modo VNAV de la capa Reactiva

Anexo V

Automatic Flight Envelope Protection

V.a. AFEP Modes

V.a.1. Modo Normal

```
Normal
en:
printf('[AFEP] Set to Normal Mode');
if (AFEPleavingContingency == true)
   printf('[AFEP] AFEP limitations reenabled');
   AFEPleavingContingency = false;
end
evNormalAFEP = false;

du:
% ** VNAV **

% Angle of Attack Protection
if (xLon.alpha > AngleOfAttackMaxValue(xLon.indicated_airspeed,Pos.alt))
   % evAoA = true;
end

% High Speed / Alpha Floor Protection
if (xLon.indicated_airspeed > IASMaxValue(Pos.alt))
   % evIAS = true;
end

% ** LNAV **
% Roll Attitude Protection
if (xLat.roll > RollMaxValue(xLon.indicated_airspeed,Pos.alt))
   % evRoll = true;
end
```

Figura V.1: Modo Normal de la máquina Modes del AFEP

V.a.2. Modo Seguro

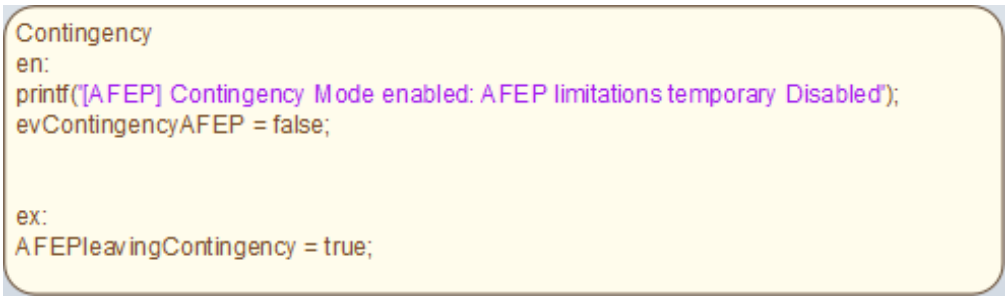

Figura V.2: Modo Seguro de la máquina Modes del AFEP

V.b. AFEP Protecciones Continuas

V.b.1. Normal Law

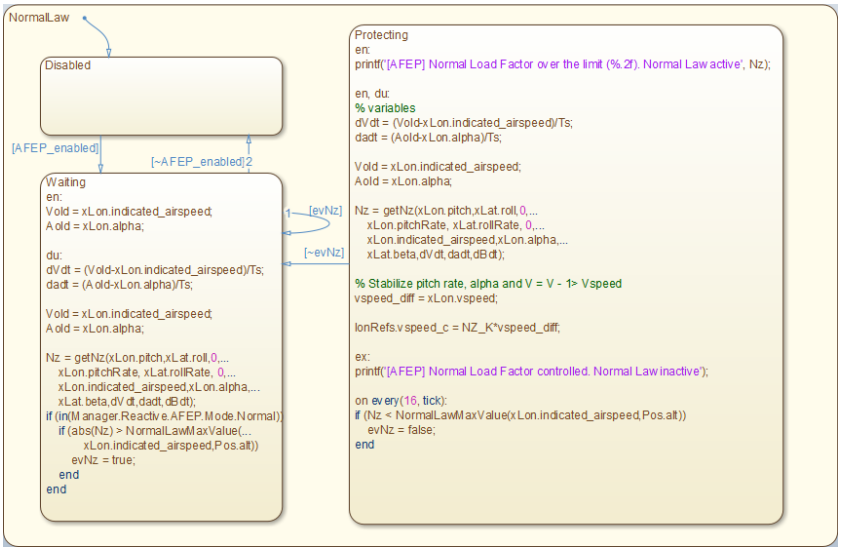

Figura V.3: Máquina de la protección Ley Normal (Continua) del AFEP

V.b.2. Lateral Law

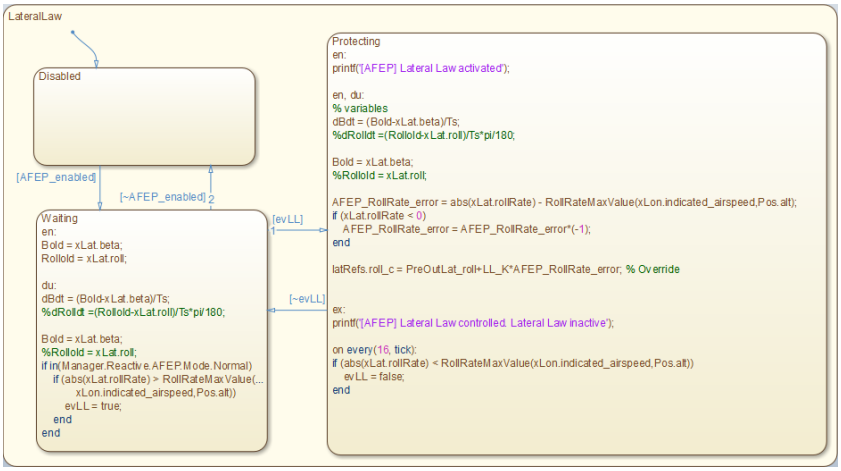

Figura V.4: Máquina de la protección Ley Lateral (Continua) del AFEP

V.c. AFEP Protecciones Situacionales

V.c.1. Protección del Ángulo de Ataque

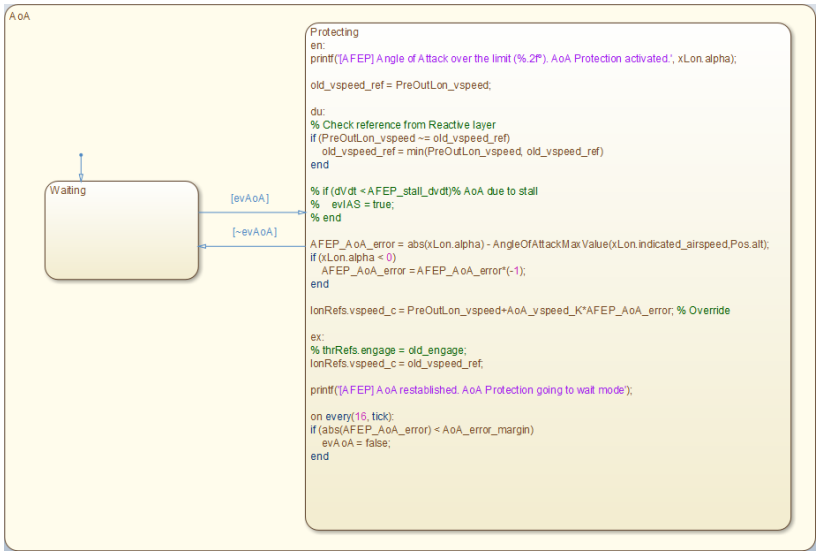

Figura V.5: Capa de protección del ángulo de ataque (α)

V.c.2. Protección de Velocidad

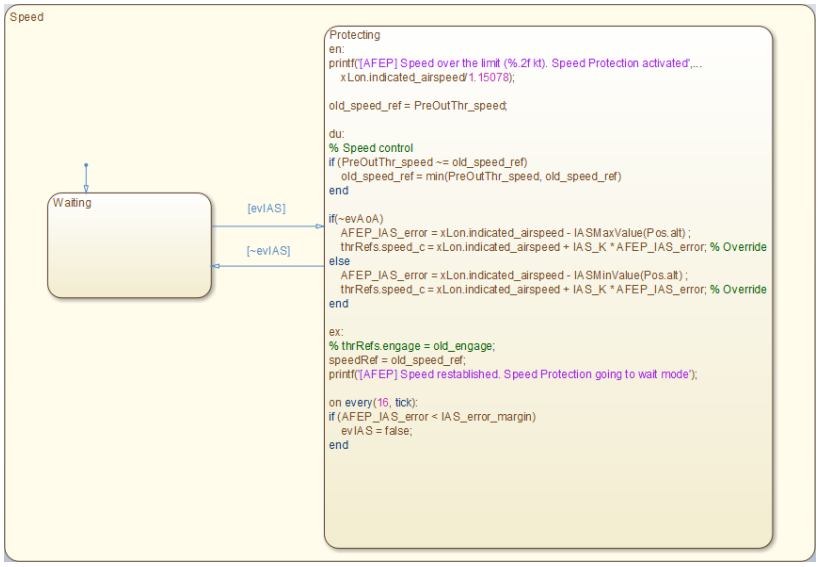

Figura V.6: Capa de protección de velocidad la máxima (V)

V.c.3. Protección del Ángulo de Alabeo

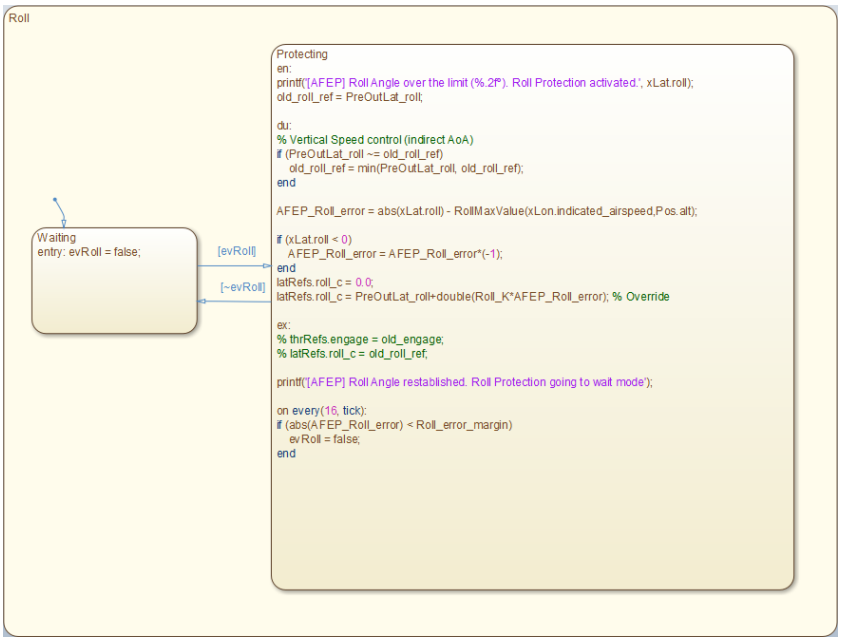

Figura V.7: Capa de protección del ángulo de alabeo (ϕ)

V.d. Defínción de la envolvente de vuelo segura del AFEP

V.d.1. Matlab: Configuración del AFEP

```
% Automatic Flight Envelope Protection configurations
ActivateAFEP  = 1; % 1 - Enabled / 0 - Disabled

AFEP_disabled        = 0;
AFEP_normal          = 1;
AFEP_contingency     = 2;

%% Continuous
% Nz Law (VNAV)
NZ_K = -1.4;

% Lateral Law (LNAV)
LL_K = -60;



%% Situational

AFEP_stall_dvdt = -0.8; %mph/s^2

%AoA (VNAV)
AoA_vspeed_K = -100;
AoA_error_margin = 1; %deg

%TAS (VNAV)
IAS_K = -1.5;
kt2mph = 1.15078;
IAS_error_margin = 3*kt2mph; %knot

%Roll (LNAV)
Roll_K = 1.4;
Roll_error_margin = 3; %deg
```

V.d.2. Matlab: Envolvente de Vuelo - Definición del ángulo de ataque

```
function AoA = AoA_MaxF(V,z)
% External Function
%V in knot / z in ft
AoA = 0;
if ((z>200) && ((V-60)+(z-200)^0.5)>0)
    AoA = 3+0.2*((V-60)+(z-200)^0.5)^0.5;
end

if (AoA < 3)
    AoA = 3;
end
```

V.d.3. Matlab: Envolvente de Vuelo - Definición de la velocidad

```
function IAS = IAS_MaxF(z)
% External Function
kt2mph = 1.15078;
IAS = 135 * kt2mph; %mph
```

V.d.4. Matlab: Envolvente de Vuelo - Definición del ángulo de alabeo

```
function Roll = Roll_MaxF(V, z)
% External Function
Roll = 15; %deg
```

V.d.5. Matlab: Envolvente de Vuelo - Definición de la Ley Normal

```
function NL = NL_MaxF(V,z)
% External Function
NL = 2.5;
```

V.d.6. Matlab: Envolvente de Vuelo - Definición de la Ley Lateral

```
function RollRate = RollRate_MaxF(V, z)
% External Function
RollRate = 8*pi/180; %deg/s
```

Anexo VI

Validación del sistema

VI.a. Automatizador de Misiones

VI.a.1. Matlab: Programa principal

```
function NL = NL_MaxF(V,z)
% External Function
NL = 2.5;
```

asdf

VI.a.2. Matlab: Watchdow

```
function NL = NL_MaxF(V,z)
% External Function
NL = 2.5;
```

asdf

VI.b. Funciones de Control de X-Plane

VI.b.1. Matlab: getPauseStatus

```
function isPaused = getPauseStatusXPLANE(ip, port_Xplane,
    port_Matlab)
% Asensio Lorenzo 15/06/16

freq=16;

dataref{1,1}='sim/time/paused';   dataref{1,2} = 130;
```

```
7  dref_requestXPLANE(ip,port_Xplane,port_Matlab,freq,dataref)
       ;
8  data   = dref_readXPLANE(port_Matlab, cell2mat(dataref(:,2))
       );
9  isPaused = data(1);
10 end
11
12 function sock=dref_requestXPLANE(ip,port_Xplane,port_Matlab
       ,freq,dataref)
13 n_dref= size(dataref,1);
14 sock = java.net.DatagramSocket(port_Matlab);
15
16 %Message header (the same for every request)
17 intro_bytes = uint8(['RREF' 0]);
18 freq_bytes = typecast(uint32(freq),'uint8');
19
20 for i=1:n_dref
21     dataref{i}(400) = 0;
22     %Indexing
23     if size(dataref,2) == 2
24         index_bytes = typecast(uint32(dataref{i,2}),'uint8'
               );
25     else
26         index_bytes = typecast(uint32(i),'uint8');
27     end
28     %Message constructor
29     send_msg = [intro_bytes freq_bytes index_bytes uint8(
           dataref{i,1})];
30     send_pkt = java.net.DatagramPacket(send_msg, length(
           send_msg),...
31                 java.net.InetAddress.getByName(ip),
                       port_Xplane);
32     %Send message
33     sock.send(send_pkt);
34 end
35 sock.close();
36 end
37
38 function data=dref_readXPLANE(port_Matlab, index)
39 max_rep = 10;
```

```
40 success = false;
41
42 rep = 1;
43 while (rep <= max_rep && ~success)
44     % Receive UDP data
45     sock = java.net.DatagramSocket(port_Matlab);
46     sock.setSoTimeout(2000);
47
48     recv_pkt = java.net.DatagramPacket(uint8(zeros(1,256)),
           256);
49     sock.receive(recv_pkt);
50     sock.close();
51
52     recv_data = recv_pkt.getData();
53     n = recv_pkt.getLength()-5;
54     recv_data = recv_data(6:end);
55     n = floor(n/8);
56
57     % Define variables (without overwriting the available
         data)
58     if ~exist('data','var')
59         data = zeros(length(index),1);
60     end
61     pindex = zeros(n,1);
62     pdata  = zeros(n,1);
63
64     % Import reived values
65     for i=1:n
66         pindex(i)= typecast(recv_data((i-1)*8+1:(i-1)*8+4),
               'uint32');
67         pdata(i) = typecast(recv_data((i-1)*8+5:(i-1)*8+8),
               'single');
68     end
69
70     % Fill required data with received values
71     len    = length(index);
72     filled = zeros(1,len);
73     for i = 1:len
74         matching_case = pindex == index(i);
75         if any(matching_case)
```

```
76                 data(i) = pdata(matching_case);
77                  filled(i) = true;
78             end
79         end
80
81         %Detect if all required data has been received
82         if all(filled)
83             success = true;
84         end
85
86         rep = rep+1;
87     end
88     end
```

VI.b.2. Matlab: togglePause

```
1  function togglePauseXPLANEy(ip,port)
2  %Asensio Lorenzo 15/06/16
3
4  ssock = java.net.DatagramSocket();
5
6  send_msg=uint8(zeros(1,5));
7  send_msg(1:4)=uint8('CMND');
8  send_msg(5)=uint8(0);
9
10 msg = uint8('sim/operation/pause_toggle');
11
12 send_msg = [send_msg msg];
13
14 len = length(send_msg);
15
16 send_pkt = java.net.DatagramPacket(send_msg, len, java.net.
       InetAddress.getByName(ip),port);
17 ssock.send(send_pkt);
18 close(ssock);
19 end
```

VI.b.3. Matlab: setPause

```
1  function setPauseXPLANE(ip, Xplane_port, Matlab_port, str)
2  %Asensio Lorenzo 15/06/16
3
```

```
4  isPaused = getPauseStatusXPLANE(ip, Xplane_port,
       Matlab_port);
5
6  if (any(strcmpi(str,{'pause', 'on'})) && ~isPaused)
7      togglePauseXPLANE(ip, Xplane_port);
8  elseif (any(strcmpi(str,{'unpause', 'off'})) && isPaused)
9      togglePauseXPLANE(ip, Xplane_port);
10 end
```

VI.b.4. Matlab: isScenaryLoad

```
function isLoad = isScenaryLoadXPLANE(ip, port_Xplane, port_Matlab)
% Asensio Lorenzo 15/06/16

freq=16;

dataref{1,1}='sim/graphics/scenery/async_scenery_load_in_progress';
   dataref{1,2} = 200;
dref_requestXPLANE(ip,port_Xplane,port_Matlab,freq,dataref);
data   = dref_readXPLANE(port_Matlab, cell2mat(dataref(:,2)) );
isLoad = ~data(1);
end

function sock=dref_requestXPLANE(ip,port_Xplane,port_Matlab,freq,dataref)
n_dref= size(dataref,1);
sock = java.net.DatagramSocket(port_Matlab);

% Message header (the same for every request)
intro_bytes = uint8(['RREF' 0]);
freq_bytes = typecast(uint32(freq),'uint8');

for i=1:n_dref
    dataref{i}(400) = 0;
    % Indexing
    if size(dataref,2) == 2
        index_bytes = typecast(uint32(dataref{i,2}),'uint8');
    else
        index_bytes = typecast(uint32(i),'uint8');
    end
    % Message constructor
    send_msg = [intro_bytes freq_bytes index_bytes uint8(dataref{i,1})];
    send_pkt = java.net.DatagramPacket(send_msg, length(send_msg),...
                java.net.InetAddress.getByName(ip),port_Xplane);
    % Send message
    sock.send(send_pkt);
end
sock.close();
end

function data=dref_readXPLANE(port_Matlab, index)
max_rep = 10;
success = false;

rep = 1;
while (rep <= max_rep && ~success)
    % Receive UDP data
    sock = java.net.DatagramSocket(port_Matlab);
    sock.setSoTimeout(2000);

    recv_pkt = java.net.DatagramPacket(uint8(zeros(1,256)), 256);
```

```
49      sock.receive(recv_pkt);
50      sock.close();
51
52      recv_data = recv_pkt.getData();
53      n = recv_pkt.getLength()-5;
54      recv_data = recv_data(6:end);
55      n = floor(n/8);
56
57      %Define variables (without overwriting the available data)
58      if ~exist('data','var')
59           data = zeros(length(index),1);
60      end
61      pindex = zeros(n,1);
62      pdata  = zeros(n,1);
63
64      %Import reived values
65      for i=1:n
66          pindex(i)= typecast(recv_data((i-1)*8+1:(i-1)*8+4),'uint32');
67          pdata(i) = typecast(recv_data((i-1)*8+5:(i-1)*8+8),'single');
68      end
69
70      %Fill required data with received values
71      len    = length(index);
72      filled = zeros(1,len);
73      for i = 1:len
74          matching_case = pindex == index(i);
75          if any(matching_case)
76               data(i) = pdata(matching_case);
77                filled(i) = true;
78          end
79      end
80
81      %Detect if all required data has been received
82      if all(filled)
83          success = true;
84      end
85
86      rep = rep+1;
87  end
88  end
```

VI.b.5. Matlab: getPos

```
function [lat, lon, alt, local_x, local_y, local_z] =
    getPosXPLANE(ip, port_Xplane, port_Matlab)
% Asensio Lorenzo 17/06/16
freq=16;

dataref{1,1}='sim/flightmodel/position/latitude'; dataref
    {1,2} = 140;
dataref{2,1}='sim/flightmodel/position/longitude'; dataref
    {2,2} = 141;
dataref{3,1}='sim/flightmodel/position/elevation'; dataref
    {3,2} = 142;
dataref{4,1}='sim/flightmodel/position/local_x'; dataref
    {4,2} = 143;
dataref{5,1}='sim/flightmodel/position/local_y'; dataref
    {5,2} = 144;
dataref{6,1}='sim/flightmodel/position/local_z'; dataref
    {6,2} = 145;

% Dataref Request (manual indexing)
dref_requestXPLANE(ip,port_Xplane,port_Matlab,freq,dataref)
    ;

% Dataref Reception (need to know the indexes, as several
    function may ask
% different datarefs)
[data, success] = dref_readXPLANE(port_Matlab, cell2mat(
    dataref(:,2)) );

if success
    lat = data(1);
    lon = data(2);
    alt = data(3);
    local_x = data(4);
    local_y = data(5);
    local_z = data(6);
else
    lat = NaN;
    lon = NaN;
    alt = NaN;
```

```
30      local_x = NaN;
31      local_y = NaN;
32      local_z = NaN;
33  end
34  end
35
36
37  function sock = dref_requestXPLANE(ip,port_Xplane,
        port_Matlab,freq,dataref)
38  % Asensio Lorenzo 15/06/16
39
40  n_dref= size(dataref,1);
41  sock = java.net.DatagramSocket(port_Matlab);
42
43  % Message header (the same for every request)
44  intro_bytes = uint8(['RREF' 0]);
45  freq_bytes = typecast(uint32(freq),'uint8');
46
47  for i=1:n_dref
48      dataref{i}(400) = 0; % Fixed length
49      % Indexing
50      if size(dataref,2) == 2
51          index_bytes = typecast(uint32(dataref{i,2}),'uint8'
                );
52      else
53          index_bytes = typecast(uint32(i),'uint8');
54      end
55      % Message constructor
56      send_msg = [intro_bytes freq_bytes index_bytes uint8(
            dataref{i,1})];
57      send_pkt = java.net.DatagramPacket(send_msg, length(
            send_msg),...
58                  java.net.InetAddress.getByName(ip),
                        port_Xplane);
59      % Send message
60      sock.send(send_pkt);
61  end
62  sock.close();
63  end
64
```

```
65
66 function [data, success] = dref_readXPLANE(port_Matlab,
       index)
67 max_rep = 10;
68 success = false;
69
70 rep = 1;
71 while (rep <= max_rep && ~success)
72     % Receive UDP data
73     sock = java.net.DatagramSocket(port_Matlab);
74     sock.setSoTimeout(2000);
75
76     recv_pkt = java.net.DatagramPacket(uint8(zeros(1,256)),
           256);
77     sock.receive(recv_pkt);
78     sock.close();
79
80     recv_data = recv_pkt.getData();
81     n = recv_pkt.getLength()-5;
82     recv_data = recv_data(6:end);
83     n = floor(n/8);
84
85     % Define variables (without overwriting the available
           data)
86     if ~exist('data','var')
87          data = zeros(length(index),1);
88     end
89     pindex = zeros(n,1);
90     pdata  = zeros(n,1);
91
92     % Import reived values
93     for i=1:n
94         pindex(i)= typecast(recv_data((i-1)*8+1:(i-1)*8+4),
               'uint32');
95         pdata(i) = typecast(recv_data((i-1)*8+5:(i-1)*8+8),
               'single');
96     end
97
98     % Fill required data with received values
99     len    = length(index);
```

```
100     filled = zeros(1,len);
101     for i = 1:len
102         matching_case = pindex == index(i);
103         if any(matching_case)
104             data(i) = pdata(matching_case);
105              filled(i) = true;
106         end
107     end
108
109     %Detect if all required data has been received
110     if all(filled)
111         success = true;
112     end
113
114     rep = rep+1;
115 end
116 end
```

VI.b.6. Matlab: setPos

```
1  function setPosXPLANE(ip,port,local_x,local_y,local_z)
2  %Asensio Lorenzo 15/06/16
3
4  %Modify Xplane plane position via UDP :
5  %
6  %local_x  / sim/flightmodel/position/local_x   double
7  %local_y  / sim/flightmodel/position/local_y   double
8  %local_z  / sim/flightmodel/position/local_z   double
9  %
10 %Out of range values are ignored by writeXPLANE.
11 %ip   : IP address of Xplane / Example: '127.0.0.1'
12 %port : port of Xplane / Integer / Example: 9005
13
14 dataref_x= 'sim/flightmodel/position/local_x';
15 dataref_y= 'sim/flightmodel/position/local_y';
16 dataref_z= 'sim/flightmodel/position/local_z';
17
18 ssock = java.net.DatagramSocket();
19 UDPsendDREFfloat(ssock,ip,port,dataref_x,local_x);
20 UDPsendDREFfloat(ssock,ip,port,dataref_y,local_y);
21 UDPsendDREFfloat(ssock,ip,port,dataref_z,local_z);
```

```matlab
ssock.close();

end

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function UDPsendDREFfloat(ssock,ip,port,dataref,value)
dataref(500) = 0; %Fixed length

intro_bytes   = uint8(['DREF' 0]);
value_bytes   = typecast(single(value),'uint8');
dataref_bytes = uint8(dataref);

send_msg = [intro_bytes value_bytes dataref_bytes];

send_pkt = java.net.DatagramPacket(send_msg, size(send_msg,2),...
           java.net.InetAddress.getByName(ip),port);
ssock.send(send_pkt);
end
```

VI.b.7. Matlab: setCoord

```matlab
function setCoordXPLANE(ip, port_Xplane, port_Matlab, req_lat, req_lon, req_alt, wait_mode)
% Asensio Lorenzo 15/06/16

% req_lat / req_lon (degrees)
% req_alt           (ft) [optional]
% wait_mode = 'wait' waits until map is fully load [optional]
%
% Asensio Lorenzo 17/06/16

long_pause = 1;
ft2m = 0.3048;
safe_local_y = 5000; %m
isBigJump = false;

%% Initialize
% Keep altitude if not defined
if (~exist('req_alt', 'var') || isempty(req_alt))
```

```
18      [~, ~, xp_alt] = getXZ(ip, port_Xplane, port_Matlab); %
            3rd output is Y
19      req_alt = xp_alt/ft2m;
20  end
21
22  % Place the plane in the center of the sector (the plane is
        safer at a high
23  % altitude)
24  setPosXPLANE(ip,port_Xplane,0  ,safe_local_y,0);
25  pause(long_pause);
26  [xp_lat, xp_lon] = getLatLon(ip, port_Xplane, port_Matlab);
27
28  %% Sector Positioning (Rough)
29  % Set Rough Longitude
30  while (abs(req_lon - xp_lon) > 0.55)
31      isBigJump = true;
32      if (req_lon > xp_lon)
33          moveFromSector(ip, port_Xplane, 'east');
34      else
35          moveFromSector(ip, port_Xplane, 'west');
36      end
37      setPosXPLANE(ip,port_Xplane,0  ,safe_local_y,0);
38      pause(long_pause);
39      [~, xp_lon] = getLatLon(ip, port_Xplane, port_Matlab);
40  end
41
42  % Set Rough Latitude
43  while (abs(req_lat - xp_lat) > 0.6)
44      isBigJump = true;
45      if (req_lat > xp_lat)
46          moveFromSector(ip, port_Xplane, 'north');
47      else
48          moveFromSector(ip, port_Xplane, 'south');
49      end
50      setPosXPLANE(ip,port_Xplane,0  ,safe_local_y,0);
51      pause(long_pause);
52      [xp_lat, xp_lon] = getLatLon(ip, port_Xplane,
            port_Matlab);
53  end
54
```

```
55 if isBigJump
56     fprintf('\tReloading scenary...');
57     pause(10);
58     fprintf('\tDone\n');
59 else
60     setPosXPLANE(ip,port_Xplane,0  ,safe_local_y,0);
61 end
62
63 %% Coordinate Positioning (Fine)
64 %D: Desired / R: Random / O: Obtained / C: Calculated
65 tolerance = 0.001;
66 x_jump_R = 10000 * sign(req_lon - xp_lon );
67 z_jump_R = 10000 * sign(xp_lat  - req_lat);
68
69 while (abs(req_lat - xp_lat) > tolerance || abs(req_lon -
       xp_lon) > tolerance)
70     % Set Fine Longitude
71             start_lon  = xp_lon;
72         [~, xp_lon] = moveWithinSector(ip, port_Xplane,
               port_Matlab, 'X', x_jump_R); fprintf('[setCoord]
                x_jump_R = %g\n',x_jump_R);
73             diff_lon_R = xp_lon  - start_lon;
74             diff_lon_D = req_lon - xp_lon;
75             x_jump_C   = x_jump_R / diff_lon_R * diff_lon_D
                   ; %(Regla de 3)
76             start_lon  = xp_lon;
77         [xp_lat, xp_lon] = moveWithinSector(ip, port_Xplane
               , port_Matlab, 'X', x_jump_C); fprintf('[
               setCoord] x_jump_C = %g\n',x_jump_C);
78             diff_lon_O = xp_lon - start_lon; % get
79
80     % Set Fine Latitude
81             start_lat = xp_lat;
82         [xp_lat, ~] = moveWithinSector(ip, port_Xplane,
               port_Matlab, 'Z', z_jump_R); fprintf('[setCoord]
                z_jump_R = %g\n',z_jump_R);
83             diff_lat_R = xp_lat  - start_lat;
84             diff_lat_D = req_lat - xp_lat;
85             z_jump_C   = z_jump_R / diff_lat_R * diff_lat_D
                   ; %(Regla de 3)
```

```
86              start_lat   = xp_lat;
87          [xp_lat, xp_lon] = moveWithinSector(ip, port_Xplane
                , port_Matlab, 'Z', z_jump_C); fprintf('[
               setCoord] z_jump_C = %g\n',z_jump_C);
88              diff_lat_O = xp_lat - start_lat;
89
90      %Recalculate first step jumps
91          diff_lon_D = req_lon - xp_lon;
92          x_jump_R = x_jump_C / diff_lon_O * diff_lon_D;
93          diff_lat_D = req_lat - xp_lat;
94          z_jump_R = z_jump_C / diff_lat_O * diff_lat_D;
95  end
96
97  %% Restablish Heigh
98  [X, Z] = getXZ(ip, port_Xplane, port_Matlab);
99  setPosXPLANE(ip,port_Xplane,X,req_alt*ft2m,Z);
100
101 %% Wait until scenary is fully load (optional)
102 if exist('wait_mode', 'var') && strcmpi(wait_mode,'wait')
103     while ~isScenaryLoadXPLANE(ip, port_Xplane, port_Matlab
            )
104         pause(1);
105     end
106 end
107 end
108
109 %Coordinates System
110 % +------->   X
111 % |
112 % |
113 % \|/
114 %     Z
115
116 function moveFromSector(ip, port_Xplane, dir)
117 long_pause = 1;
118 jump = 75000;
119 safe_local_y = 5000; %m
120
121 switch dir
122 case {'up','north'}
```

```
123     setPosXPLANE(ip,port_Xplane,0,safe_local_y,-jump);
124 case {'down','south'}
125     setPosXPLANE(ip,port_Xplane,0,safe_local_y,jump);
126 case {'right','east'}
127     setPosXPLANE(ip,port_Xplane,jump,safe_local_y,0);
128 case {'left','west'}
129     setPosXPLANE(ip,port_Xplane,-jump,safe_local_y,0);
130 end
131 pause(long_pause);
132 end
133
134 function [Lat, Lon, NewX, NewZ, sc] = moveWithinSector(ip,
       port_Xplane, port_Matlab, dir, dist)
135 safe_local_y = 5000; %m
136 HOR = 0; VER = 1;
137 short_pause = 0.5;
138
139 [Lat, Lon, x, z] = getLatLonXZ(ip, port_Xplane, port_Matlab
    );
140 if any(~isfinite([dist x z])) || abs(dist)>50000
141     disp('[setCoord] weird distance detected');
142     return
143 end
144
145 %Move the plane and check if sector has changed
146 switch dir
147     case {'x', 'X'}
148         dir_code = HOR; %Horizontal
149         setPosXPLANE(ip,port_Xplane,x+dist,safe_local_y,z);
150         pause(short_pause);
151         [Lat, Lon, NewX, NewZ] = getLatLonXZ(ip,
               port_Xplane, port_Matlab);
152         sc = sectorHasChanged(NewX, NewZ, x+dist, z);
153     case {'z', 'Z'}
154         dir_code = VER; %Vertical
155         setPosXPLANE(ip,port_Xplane,x,safe_local_y,z+dist);
156         pause(short_pause);
157         [Lat, Lon, NewX, NewZ] = getLatLonXZ(ip,
               port_Xplane, port_Matlab);
158         sc = sectorHasChanged(NewX, NewZ, x, z+dist);
```

```
159 end
160
161 % If Sector changed, then correct it
162 if sc
163     warning('Sector changed, staying at: (%.2f, %.2f) ',x,z
        )
164     if dir_code == HOR
165         if NewX < 0
166             moveFromSector(ip, port_Xplane, 'west');
167         else
168             moveFromSector(ip, port_Xplane, 'east');
169         end
170     elseif dir_code == VER
171         if NewZ < 0
172             moveFromSector(ip, port_Xplane, 'north');
173         else
174             moveFromSector(ip, port_Xplane, 'south');
175         end
176     end
177     setPosXPLANE(ip,port_Xplane,x,safe_local_y,z);
178 end
179 end
180
181 function ch = sectorHasChanged(new_x, new_z, x, z)
182 % Check if the sector has changed
183 if sqrt((x-new_x)^2+(z-new_z)^2) > 1000
184     ch = true;
185 else
186     ch = false;
187 end
188
189 end
190
191 function [Lat, Lon] = getLatLon(ip, port_Xplane,
       port_Matlab)
192 short_pause = 0.3;
193 trials  = 4;
194 intent  = 1;
195 success = false;
196 while intent <= trials && ~success
```

```
197     try
198         [Lat, Lon, ~, ~, ~, ~] = getPosXPLANE(ip,
                port_Xplane, port_Matlab);
199         if all(isfinite([Lat, Lon]))
200             success = true;
201         else
202             pause(short_pause);
203         end
204     catch err
205         intent = intent + 1;
206         pause(short_pause);
207             if intent == 4
208                 fprintf(2,'Connection Lost\n');
209                 rethrow(err)
210             end
211     end
212 end
213 end
214
215 function [X, Z, Y] = getXZ(ip, port_Xplane, port_Matlab)
216 short_pause = 0.3;
217 trials = 4;
218 intent = 1;
219 success = false;
220 while intent <= trials && ~success
221     try
222         [~, ~, ~, X, Y, Z] = getPosXPLANE(ip, port_Xplane,
                port_Matlab);
223         if all(isfinite([X, Z]))
224             success = true;
225         else
226             pause(short_pause);
227         end
228     catch err
229         intent = intent + 1;
230         pause(short_pause);
231             if intent == 4
232                 fprintf(2,'Connection Lost\n');
233                 rethrow(err)
234             end
```

```
235     end
236 end
237 end
238
239 function [Lat, Lon, X, Z] = getLatLonXZ(ip, port_Xplane,
        port_Matlab)
240 short_pause = 0.3;
241 trials = 4;
242 intent = 1;
243 success = false;
244 while intent <= trials && ~success
245     try
246         [Lat, Lon, ~, X, ~, Z] = getPosXPLANE(ip,
              port_Xplane, port_Matlab);
247         if all(isfinite([Lat, Lon, X, Z]))
248             success = true;
249         else
250             pause(short_pause);
251         end
252     catch err
253         intent = intent + 1;
254         pause(short_pause);
255             if intent == 4
256                 fprintf(2,'Connection Lost\n');
257                 rethrow(err)
258             end
259     end
260 end
261 end
```

VI.b.8. Matlab: setOrientation

```
1 function setOrientationXPLANE(ip,port,v,roll,pitch,heading)
2 %Modify Xplane plane orientation via UDP:
3 %Input:
4 %   -> X-Plane IP          (string)
5 %   -> Port                (integer)
6 %   -> TAS                 (knot)
7 %   -> Roll/Pitch/Heading  (Degrees)
8 %* Possible improvement: read local_v (x,y,z) and angles
      phi, psi, theta
```

```
9  %* and rotate speed vectors accordingly
10
11 kt2m = 1852;
12
13 S = [-1 0 0; 0 1 0; 0 0 1]; %Correct X-Plane coord system
14 TAS = v *kt2m/3600;          %Must be in m/s
15
16 phi   = roll    * pi /180; %X
17 theta = pitch   * pi /180; %Y
18 psi   = heading* pi /180; %Z
19
20 vTAS = S*mrotY(psi)*mrotX(theta)*[0 0 -TAS]'; %Coherent
       speed vector
21 q = angle2quat(phi,theta,psi,'XYZ');
22
23 variables = {'local_vx', 'local_vy', 'local_vz'};
24
25 ssock = java.net.DatagramSocket();
26 %Set speed
27 UDPsendDREFfloat(ssock,ip,port,'sim/flightmodel/position/
       local_vx',vTAS(1));
28 UDPsendDREFfloat(ssock,ip,port,'sim/flightmodel/position/
       local_vy',vTAS(2));
29 UDPsendDREFfloat(ssock,ip,port,'sim/flightmodel/position/
       local_vz',vTAS(3));
30
31 for i = 1:4
32     dataref= ['sim/flightmodel/position/q[' num2str(i-1) ']
           '];
33     value = q(i);
34     UDPsendDREFfloat(ssock,ip,port,dataref,value);
35 end
36
37
38 ssock.close();
39
40 end
41
42
43 %
```

```
    %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function UDPsendDREFfloat(ssock,ip,port,dataref,value)
send_msg=uint8(zeros(1,509));
send_msg(1:4)=uint8('DREF');
send_msg(5)=uint8(0);

value=single(value);
value_bytes=typecast(value,'uint8');
len=6;
for j=1:4
    send_msg(len)=value_bytes(j);
    len=len+1;
end
dataref_bytes=uint8(dataref);
for j=1:length(dataref)
    send_msg(len)=dataref_bytes(j);
    len=len+1;
end

send_pkt = java.net.DatagramPacket(send_msg, size(send_msg
    ,2),java.net.InetAddress.getByName(ip),port);
ssock.send(send_pkt);
clear send_pkt;
end

function m = mrotX(theta)
m = [1          0               0        ;...
     0     cos(theta)  -sin(theta);...
     0     sin(theta)   cos(theta)];
end

function m = mrotY(psi)
m = [ cos(psi)  0  sin(psi);...
          0       1      0      ;...
     -sin(psi)  0  cos(psi)];
end
```

VI.b.9. Matlab: setWeather

```
function setWeatherXPLANE(ip,port,data)
```

```
2  % Asensio Lorenzo 15/06/16
3
4  % Modify Xplane plane weather variables via UDP :
5  %
6  % Section: sim/weather/
7  % wind_altitude_msl_m[0]        int             meters >= 0
       The center altitude of this layer of wind in MSL
   meters.
8  % wind_altitude_msl_m[1]        int             meters >= 0
       The center altitude of this layer of wind in MSL
   meters.
9  % wind_altitude_msl_m[2]        int             meters >= 0
       The center altitude of this layer of wind in MSL
   meters.
10 % wind_direction_degt[0]        float   [0 - 360)       The
    direction the wind is blowing from in degrees from true
    north c lockwise.
11 % wind_direction_degt[1]        float   [0 - 360)       The
    direction the wind is blowing from in degrees from true
    north c lockwise.
12 % wind_direction_degt[2]        float   [0 - 360)       The
    direction the wind is blowing from in degrees from true
    north c lockwise.
13 % wind_speed_kt[0]          int         kts >= 0        The
    wind speed in knots.
14 % wind_speed_kt[1]          int         kts >= 0        The
    wind speed in knots.
15 % wind_speed_kt[2]        int           kts >= 0        The
    wind speed in knots.
16 %
17 % ip    : IP address of Xplane / String / Example:
   '127.0.0.1'
18 % port : port of Xplane / Integer / Example: 9005
19
20 section = 'sim/weather/';
21 allowed_dr = {...
22     'wind_altitude_msl_m',...
23     'wind_direction_degt',...
24     'wind_speed_kt'};
25
```

```
%Open the receiving socket
ssock = java.net.DatagramSocket();
fn = fieldnames(data); %Read field names of the structure
if iscell(fn)          %Check if it has only one field
    l = length(fn);
else
    l = 1;
end

for i = 1:l
    if any(strcmp(fn{i},allowed_dr))
        c_values = data.(fn{i});
        n_cells  = length(c_values);
        for j = 1:n_cells
            if ~isempty(c_values{j})
                dataref= [section fn{i}...
                    '[' num2str(j-1) ']'];
                value = c_values{j};
                UDPsendDREFfloat(ssock,ip,port,dataref,
                   value);
            end
        end
    end
end

ssock.close();

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function UDPsendDREFfloat(ssock,ip,port,dataref,value)
send_msg=uint8(zeros(1,509));
send_msg(1:4)=uint8('DREF');
send_msg(5)=uint8(0);

value=single(value);
value_bytes=typecast(value,'uint8');
len=6;
for j=1:4
```

```
65      send_msg(len)=value_bytes(j);
66      len=len+1;
67  end
68  dataref_bytes=uint8(dataref);
69  for j=1:length(dataref)
70      send_msg(len)=dataref_bytes(j);
71      len=len+1;
72  end
73  send_pkt = java.net.DatagramPacket(send_msg, size(send_msg
        ,2),java.net.InetAddress.getByName(ip),port);
74  ssock.send(send_pkt);
75  clear send_pkt;
76  end
77
78  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
79  function UDPsendDREFchar(ssock,ip,port,dataref,char)
80  send_msg=uint8(zeros(1,509));
81  send_msg(1:4)=uint8('DREF');
82  send_msg(5)=uint8(0);
83
84  value_bytes=typecast(char,'uint8');
85
86  send_msg(9)=value_bytes;
87  len=10;
88  dataref_bytes=uint8(dataref);
89  for j=1:length(dataref)
90      send_msg(len)=dataref_bytes(j);
91      len=len+1;
92  end
93  send_pkt = java.net.DatagramPacket(send_msg, size(send_msg
        ,2),java.net.InetAddress.getByName(ip),port);
94  ssock.send(send_pkt);
95  clear send_pkt;
96  end
97
98  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
99  % Other useful functions
100 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
101 function send_msg=UDPsendDATA(ip,port,index,data)
102 ssock = java.net.DatagramSocket();
```

```
103 n=length(index);
104 send_msg=uint8(zeros(1,41));
105 send_msg(1:4)=uint8('DATA');
106 send_msg(5)=uint8(0);
107
108 index=uint32(index);
109 data=single(data);
110
111 for j=1:n
112     index_bytes=typecast(index(j),'uint8');
113     len=6;
114     for k=1:4
115         send_msg(len)=index_bytes(k);
116         len=len+1;
117     end
118     for h=1:8
119         data_bytes=typecast(data((j-1)*8+h),'uint8');
120         for k=1:4
121             send_msg(len)=data_bytes(k);
122             len=len+1;
123         end
124     end
125     send_pkt = java.net.DatagramPacket(send_msg, 41, java.
        net.InetAddress.getByName(ip),port);
126     ssock.send(send_pkt);
127 end
128 close(ssock);
129 end
130
131 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
132 function UDPsendCHAR(ip,port,char)
133 ssock = java.net.DatagramSocket();
134 send_msg=uint8(zeros(1,6));
135 send_msg(1:4)=uint8('CHAR');
136 send_msg(5)=uint8(0);
137 send_msg(6)=uint8(char);
138 send_pkt = java.net.DatagramPacket(send_msg, size(send_msg
    ,2), java.net.InetAddress.getByName(ip),port);
139 ssock.send(send_pkt);
140 close(ssock);
```

```
141 end
```

VI.b.10. Matlab: fixPlane

```
1  function fixPlaneXPLANE(ip,port)
2  % Asensio Lorenzo 15/06/16
3
4  ssock = java.net.DatagramSocket();
5
6  send_msg=uint8(zeros(1,5));
7  send_msg(1:4)=uint8('CMND');
8  send_msg(5)=uint8(0);
9
10 msg = uint8('sim/operation/fix_all_systems');
11
12 send_msg = [send_msg msg];
13
14 len = length(send_msg);
15
16 send_pkt = java.net.DatagramPacket(send_msg, len, java.net.
       InetAddress.getByName(ip),port);
17 ssock.send(send_pkt);
18 close(ssock);
19 end
```

Bibliografía

[1] ICAO - International Civil Aviation Organization. ICAO Flight Plan, 2011. FAL - Free Art License 1.3. URL: https://commons.wikimedia.org/w/index.php?curid=15768935.

[2] Neale, M. *ICAO RPAS Manual C2 Link and Communications.* Table III-2-5-App-3. ICAO - 2015 RPAS Symposium - Day 2 Workshop 5.

[3] Enaire. Santander - IAC/7.1 - RNAV (GNSS) Z RWY 11, 2016. URL: http://www.enaire.es/csee/Satellite/navegacion-aerea/es/Page/1078418725163/?other=1083158950596#ancla3.

[4] Vila, J. A. *Transporte Navegación y Circulación Aérea. Unidad 8b. Advanced navigation (ii). Area navigation (RNAV) and Required Navigation Performace (RNP).* **Diciembre, 2013**, p. 38.

[5] ARINC. *ARINC Specification 424.* **2008**, p. 43.

[6] Meggar (WikiMedia). Six flight instruments in an aircraft cockpits, sometimes refered to as "basic-T" or "basic flight instruments", February, 2008. Licencia CC BY-SA 3.0. URL: http://www.enaire.es/csee/Satellite/navegacion-aerea/es/Page/1078418725163/?other=1083158950596#ancla3.

[7] PresLoiLoi (WikiMedia). FMS CDU of 737-300 plane, Mayo, 2012. Licencia CC BY-SA 3.0. URL: https://commons.wikimedia.org/w/index.php?curid=35964509.

[8] FlySafe Project (WikiMedia). FMS CDU of 737-300 plane, Febrero, 2011. Public Domain. URL: http://www.eu-flysafe.org/Project/Aviation-Hazards/Air-Traffic/current-systems.html.

[9] Carbó, J. V. *RPAS activities at Universitat Politécnica de València (UPV).* **Marzo, 2016**, pag. 13.

[10] Tangopaso (WikiMedia). IAI Heron on display at the Paris Air Show 2009, 2009. CC BY-SA 3.0. URL: `https://commons.wikimedia.org/w/index.php?curid=7100872`.

[11] Hèctor Usach Molina, A. C. y. P. Y. *A Highly-Automated RPAS Mission Manager for Integrated Airspace.* **Septiembre, 2015**.

[12] OACI. *Doc 8168. Procedures for Air Navigation Services. Volume II.* Table III-2-5-App-3, 6th edition.

[13] Andreas Varga, A. H. y. G. P. *Optimization Based Clearance of Flight COntrol Laws. A Civil Aircraft Application.* **2012**, Table 2.1, pag 15.

[14] Carbó, J. V. *RPAS activities at Universitat Politécnica de València (UPV).* **Marzo, 2016**, pag. 16.

[15] Israel Aerospace Industries. IAI Heron Specifications, Noviembre, 2012. URL: `http://www.israeli-weapons.com/weapons/aircraft/uav/heron/heron.html`.

[16] OACI. *Anexo 2 al Convenio sobre Aviación Civil Internacional.* Reglamento del aire. **Julio, 2005**, Cap 1 - 4, 10ª edición.

[17] OACI. *Anexo 2 al Convenio sobre Aviación Civil Internacional.* Reglamento del aire. **Julio, 2005**, Cap 3 - 5, 10ª edición.

[18] Dirección General de Aviación Civil (DGAC). *Aprobación operacional y Criterios de utilización de sistemas para la Navegación de Área Básica (RNAV Básica) en el espacio aéreo europeo.* **Abril, 1998**.

[19] EUROCONTROL. *ARINC 424 Specification.* **Abril, 2016**. URL: `http://www.eurocontrol.int/articles/arinc424-specification`.

[20] de Haag, D. M. U. *Flight Management Systems. Databases.* p. 27.

[21] OACI. *Doc 8168. Procedures for Air Navigation Services. Volume II.* **October, 2006**, p. III-5-1-1, 5th edition.

[22] Flight Management System, Agosto, 2016. URL: http://www.skybrary.aero/index.php/Flight_Management_System.

[23] Andreas Varga, A. H. y. G. P. *Optimization Based Clearance of Flight COntrol Laws. A Civil Aircraft Application.* **2012**, pags. 11-15.

[24] Airborne Collision Avoidance System (ACAS), Mayo, 2016. URL: http://www.skybrary.aero/index.php/Airborne_Collision_Avoidance_System_(ACAS).

[25] OACI. *Manual on Remotely Piloted Aircraft Sytems (RPAS).* **2015**, Cap 11, 6.23, 1ª edición.

[26] OACI. *Manual on Remotely Piloted Aircraft Sytems (RPAS).* **2015**, Cap 11, 6.4, 1ª edición.

[27] XE. Currency Converter: USD to EUR, 09:07 UTC, 01/09/2016. URL: http://www.xe.com/currencyconverter/convert/?From=USD&To=EUR.