UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA AERONÁUTICA Y DEL ESPACIO

GRADO EN INGENIERÍA AEROESPACIAL

TRABAJO FIN DE GRADO

“Diseño y Desarrollo de una Herramienta para Optimización de Sistemas de Control de Vuelo”

AUTOR: Fidel ECHEVARRÍA CORRALES

ESPECIALIDAD: Vehículos Aeroespaciales

TUTOR PROFESIONAL: Enrique BABIO FERNÁNDEZ

TUTOR ACADÉMICO: Javier CUBAS CANO

Septiembre de 2016

Agradecimientos

El trabajo de fin de grado que aquí se presenta es por razones administrativas una aportación unipersonal. Sin embargo, la realidad es que el trabajo realizado es más bien el resultado de la aportación de muchas personas, cada una desde ángulos diferentes, sin las cuales no hubiera sido posible llegar a este final. Mi agradecimiento a todas ellas, aunque me gustaría particularizar este agradecimiento en algunos casos:

En primer lugar me gustaría agradecer a la Escuela Técnica Superior de Ingeniería Aeronáutica y del Espacio, y principalmente a sus profesores, por todo el conocimiento y capacidades que me han transmitido durante estos años. Especialmente estoy agradecido al profesor Javier Cubas, doctor ingeniero aeronáutico del Instituto Universitario de Microgravedad "Ignacio Da Riva", por haber aceptado ejercer de tutor académico haciendo posible la realización de este trabajo y por haber prestado tan dedicada atención a su desarrollo.

Un agradecimiento imprescindible es el que dedico a la empresa "UAV Navigation", en la que realicé durante el primer trimestre del pasado curso las prácticas de empresa organizadas desde la ETSIAE. Este ejercicio de prácticas de empresa se extendió después a prácticas extracurriculares, y actualmente a mi primer contrato laboral. De la experiencia adquirida en esta empresa nace el trabajo Fin de Grado que aquí se presenta. Las oportunidades que se me han dado en "UAV Navigation", el trato humano recibido y la confianza que se ha depositado en mí me llevan a agradecer sinceramente a todo su personal. En especial, me gustaría personalizar este agradecimiento en el responsable de la empresa Juan Luque Suárez, quien está atravesando una etapa difícil y a quien deseo una pronta recuperación. Así mismo quisiera agradecer a quienes han sido mis contactos más cercanos en el desarrollo de este trabajo fin de grado. Enrique Babio Fernández, como tutor profesional, ha ejercido un papel fundamental en mi formación relativa a sistemas de control; Agradezco también a Miguel Ángel de Frutos Carro, director del departamento de control de vuelo, por su trato atento y cercano que me ha facilitado la integración en este equipo. También quiero agradecer a mis compañeros en este último año de trabajo en la empresa que me han hecho sentir como en casa.

En un terreno más personal me gustaría agradecer a una red de amistades que me ha ayudado enormemente a lo largo de estos años, incluyendo mis amigos de El Puerto de Santa María y los más recientes de Madrid. Finalmente, aunque no por ello menos importante, me gustaría agradecer a mi familia por su incondicional y constante apoyo, ya que han sido ellos los principales culpables de que haya llegado hasta aquí.

Índice

Tabla de contenidos

Capítulo 1

Introducción

Los sistemas de control de vuelo automáticos son fundamentales en aviación. Desde la primera época, los pioneros de la aviación ya ponen de manifiesto la necesidad de disponer de este tipo de sistemas [1].

Este Trabajo de Fin de Grado describe de forma secuencial el proceso que se ha seguido para desarrollar una herramienta de software destinada a simular y optimizar sistemas de control de vuelo de aeronaves.

Este primer capítulo de introducción comienza con un breve análisis de la necesidad de este tipo de sistemas en aviación y las funciones que desempeñan. Seguidamente, se presenta una breve revisión histórica en la que se repasa la evolución de los sistemas automáticos de control desde los orígenes de la aviación. Al final de este capítulo introductorio se presentan y desarrollan los objetivos del trabajo.

1.1 Necesidad de los sistemas de control automáticos

La evolución de las aeronaves modernas ha obligado a dar respuesta a tres grandes necesidades [1]:

- Necesidad de superficies aerodinámicas de control actuadas total o parcialmente por motores. Esta necesidad surgió principalmente por dos factores. Por un lado, las crecientes cargas aerodinámicas en las superficies de control asociadas con aviones más grandes y actuaciones mejores. Por otro la enorme cantidad de tiempo de vuelo que suponía equilibrar adecuadamente las superficies de control. Este proceso de equilibrado era necesario para proveer al piloto de una sensación de control adecuada.

- Necesidad de sistemas de piloto automático que aliviaran la fatiga causada por la atención constante de los pilotos a los controles, un factor que era necesario para volar de forma segura durante los primeros días de la aviación.

- La cada vez más amplia envolvente de vuelo creó la necesidad de aumentar la estabilidad de las aeronaves cuando volaban en diversas zonas de dicha envolvente. Esta necesidad de aumentar la estabilidad se debe a que la envolvente cubría un rango cada vez mayor de presiones dinámicas. Debido a los grandes cambios en la dinámica del avión entre un punto de la envolvente y otro, un modo dinámico estable y adecuadamente amortiguado en una condición de vuelo puede convertirse en inestable, o al menos inadecuadamente amortiguado, en otra condición de vuelo. Un modo oscilatorio poco amortiguado puede causar incomodidad a los pasajeros o dificultar el control de la trayectoria a los pilotos.

Los problemas anteriormente expuestos se resuelven utilizando control de realimentación para modificar la dinámica del avión. Las variables de estado de la aeronave se miden y se utilizan para generar señales que alimentan a los actuadores de las superficies de control aerodinámicas, modificando así el comportamiento dinámico.

1.2 Funciones de los sistemas de control automáticos

Los modos dinámicos de las aeronaves suelen estar divididos en dos categorías. Por un lado se encuentran los modos que están relacionados principalmente con los grados de libertad de rotación (modo de corto periodo, modo de convergencia en balance y modo de balanceo holandés). Sus frecuencias naturales (o constantes de tiempo, si son puramente exponenciales) están determinadas por los momentos de inercia de la aeronave y por los momentos generados por las superficies aerodinámicas. Su amortiguamiento está determinado por los momentos aerodinámicos ocasionados por las velocidades angulares, son modos de corto periodo. Los modos restantes (modo fugoide y modo espiral) están relacionados con cambios en la trayectoria de vuelo y corresponden a modos mucho más lentos. El modo fugoide está relacionado con los grados de libertad de translación y depende de las fuerzas aerodinámicas de sustentación y resistencia, y la variación de éstos con la velocidad. El modo espiral depende de los momentos aerodinámicos, pero las fuerzas aerodinámicas no toman un papel importante.

La sensibilidad de una aeronave a los comandos de maniobra está determinada en parte por la velocidad de los modos rotacionales. Las frecuencias de estos modos suelen ser lo suficientemente altas como para que un piloto encuentre difícil (o imposible) controlar el avión en caso de que los modos fueran poco amortiguados o inestables. Por lo tanto, es necesario desarrollar sistemas automáticos de control que consigan proporcionar a estos modos amortiguamentos y frecuencias naturales adecuadas. Este tipo de sistemas de control se denominan *sistemas de aumento de estabilidad* (“Stability Augmentation Systems”, SAS). En cambio, si el sistema está diseñado para controlar el modo (en lugar de evitarlo) y para proveer al piloto de un determinado tipo de respuesta a los inputs de control, entonces se denomina *sistema de aumento de control* (“Control Augmentation Systems”, CAS). Un ejemplo de CAS puede ser el CAS de aceleración normal, que consigue que los inputs del piloto controlen la aceleración generada en el eje z de la aeronave.

Los modos lentos (fugoide y espiral) son controlables por los pilotos. Sin embargo, no es recomendable que los pilotos tengan que prestar atención constante para controlar estos

modos, por lo que se necesita un sistema de control automático para disminuir la fatiga del piloto. Un *autopiloto* es un sistema de control que provee funciones de reducción de la fatiga de los pilotos además de funciones especiales como podría ser el aterrizaje automático.

A continuación se recogen las funciones típicas de los SAS, CAS y autopilotos (Tabla 1).

Tabla 1: Funciones típicas de los sistemas de control automáticos

SAS	CAS	Autopilotos
Amortiguador de roll	Velocidad angular de roll	Ángulo de pitch fijo
Amortiguador de pitch	Velocidad angular de pitch	Altitud fija
Amortiguador de yaw	Aceleración normal	Velocidad/Mach fijos
	Acoplamiento lateral/direccional	Aterrizaje automático
		Ángulo de roll fijo
		Giro coordinado
		Ángulo de yaw/subida fijos

1.3 Revisión histórica

Los sistemas automáticos de control han sido una necesidad desde los primeros tiempos de la aviación [1]. El éxito de los hermanos Wright al realizar el primer vuelo propulsado en Diciembre de 1903 se suele atribuir a dos factores: Por una parte a su enfoque de diseño sistemático (por ejemplo construyeron y utilizaron un túnel de viento); Por otro lado al énfasis que mostraron en hacer su aeronave controlable por el piloto en lugar de inherentemente estable. Sin embargo, las dificultades que presentaba controlar las primeras aeronaves y el progreso hacia tiempos de vuelo mayores impulsaron el desarrollo de un sistema automático de control. De esta manera, en el año 1912 la empresa “Sperry Gyroscope Company” desarrolló el primer autopiloto y lo probó en un hidroavión Curtiss. En 1914 el llamado “estabilizador de aeronaves Sperry” había alcanzado un estado de desarrollo avanzado que permitió que se realizara una demostración pública del sistema, en la que un mecánico caminó por el ala mientras el piloto apartaba sus manos de los controles (Figura 1).

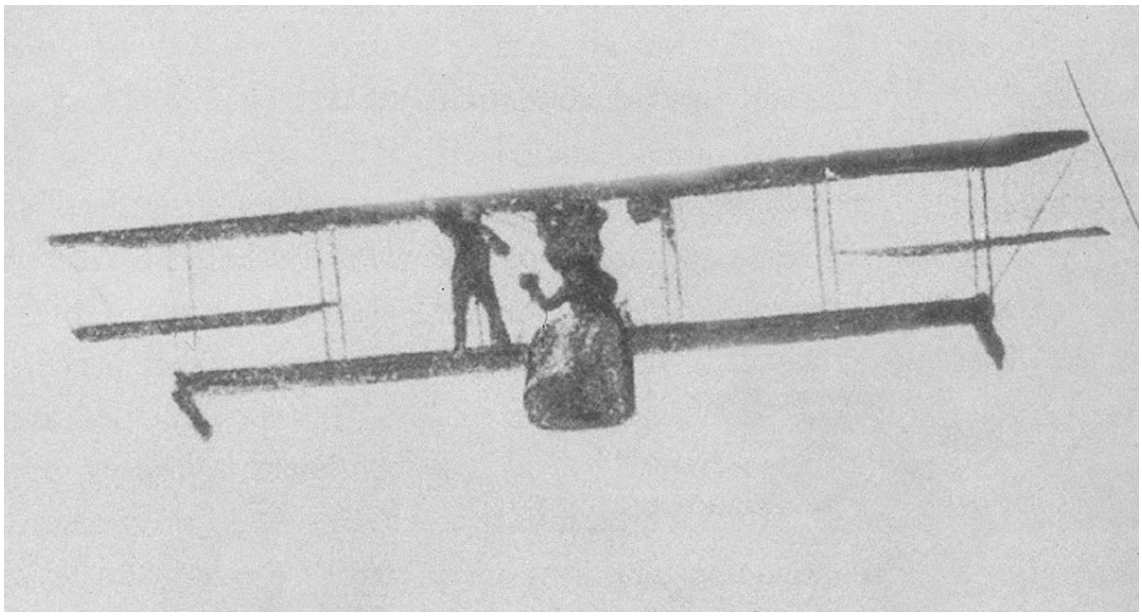

Figura 1: Demostración del estabilizador de aeronaves Sperry (1933)

Durante la Primera Guerra Mundial (1914-1918) el diseño de aeronaves experimentó un desarrollo notable. Sin embargo, un piloto humano era perfectamente capaz de ejercer las funciones de estabilización y control de las aeronaves de la época, por lo que este periodo no supuso un gran desarrollo en control automático. Se había desarrollado la teoría de pequeñas perturbaciones de la dinámica de aviones (Bryan, 1911) y en los años 20 se midieron y calcularon derivadas de estabilidad, por lo que se confirmó la teoría mediante ensayos de vuelo. Sin embargo, esta teoría tuvo escasa aplicación ya que incluso hallar las raíces de una ecuación de cuarto grado era un proceso difícil en la época. El desarrollo de los autopilotos continuó, utilizando giróscopos como sensores de referencia y servomecanismos neumáticos para posicionar las superficies de control. En el año 1933 un autopiloto Sperry ayudó a Wiley Post a volar alrededor del mundo en menos de ocho días.

A finales de los años 30 del siglo pasado se empezó a desarrollar la teoría clásica de control. La necesidad de diseñar repetidores amplificadores de teléfono con valores de ganancia muy controlados condujo al trabajo de Black sobre “la teoría de la regeneración” y al criterio de estabilidad en el dominio de la frecuencia de Nyquist. El mismo estímulo también llevó a la teoría en el dominio de la frecuencia de Bode para las relaciones entre ganancia y fase, y sus diagramas logarítmicos de ganancia y fase. Como ha ocurrido innumerables veces a lo largo de la historia, los periodos bélicos demostraron ser impulsores de áreas tecnológicas clave, en este caso diseño de aviones. La gran expansión de la envolvente de velocidad y altitud, y la necesidad de transportar y deshacerse en vuelo de cargas pesadas llevó a cambios drásticos en la dinámica de aviones, creando una necesidad de analizar el comportamiento dinámico. El aumento de masa de los aviones trajo como requerimiento superficies de control actuadas mediante motores, lo que derivó en desarrollos en el campo de los servomecanismos hidráulicos. Además, la necesidad de volar durante la noche y en malas condiciones meteorológicas trajo desarrollos en ayudas de radio-navegación y una necesidad de acoplarlas con el autopiloto.

A finales de los años 40, los conceptos de respuesta en frecuencia y función de transferencia eran más conocidos y estaban surgiendo las primeras computadoras analógicas para uso militar. La técnica del lugar de las raíces, publicada por W. R. Evans en 1948, supuso un enorme avance en el campo de análisis y diseño de sistemas de control (incluso es más útil en la actualidad gracias a los ordenadores modernos). Las compañías aéreas comenzaron a realizar análisis de la estabilidad y actuaciones de aeronaves con sistemas de control automáticos más a menudo. La envolvente de velocidad y altitud comenzó a expandirse a un ritmo muy alto (la serie "X" en los EEUU). Esto llevó, entre otros, al descubrimiento por accidente del fenómeno de acoplamiento inercial. Estos descubrimientos contribuyeron a la necesidad de un enfoque más analítico a los problemas de estabilidad y control de aeronaves. Los cambios en las propiedades másicas de los aviones, junto con la necesidad de reducir el área de las superficies aerodinámicas (para menor resistencia a altas velocidades), causaron cambios en los modos naturales de las aeronaves, lo que los convertía en incontrolables por los pilotos. Además, el amortiguamiento de los modos naturales tendía a decrecer a medida que los límites de altitud iban aumentando. Estos factores dieron más importancia a predecir la frecuencia y el amortiguamiento de los modos analíticamente. Adicionalmente, la expansión de la envolvente de velocidad y altitud derivó en que durante el vuelo existieran unas variaciones mucho más drásticas en la dinámica de las aeronaves.

Debido a las crecientes cargas aerodinámicas asociadas con aviones más grandes con mejores actuaciones se introdujeron las superficies aerodinámicas actuadas total o parcialmente por servomecanismos. Otro de los motivos que impulsó este cambio fue que eliminaba la gran cantidad de tiempo de vuelo que era necesario para equilibrar adecuadamente las superficies de control. La antigua sensación de control de los pilotos a través de los controles se podía emular utilizando muelles y contrapesos, o también mediante servomecanismos. De esta manera se podía conseguir que la sensación de los controles fuera la adecuada a lo largo de toda la envolvente de vuelo. Además, los servomecanismos de los actuadores posibilitaban realimentar las señales de los estados a los propios actuadores para aumentar la estabilidad (SAS).

Durante los años 50 hubo un rápido desarrollo tecnológico en el área de aerodinámica de altas velocidades, así como en aerotermodinámica y en combustibles y materiales especiales. Aeronaves experimentales tripuladas expandieron la envolvente de velocidad y altitud a velocidades superiores a Mach 6 y altitudes superiores a los 300,000 pies (X-15).

A principios de los años 60 los aviones de combate pequeños estaban alcanzando velocidades cercanas a Mach 2. Se produjo también un desarrollo de aviones supersónicos de transporte de pasajeros (Concorde). El ordenador digital comenzaba a causar un gran impacto en el área de la ingeniería. Las técnicas de análisis numérico ganaron mucha importancia, lo que derivó en un crecimiento de la teoría de control moderna a mediados de los 60. A finales de los años 60 hubo avances en tecnologías de control para la reentrada de vehículos en la atmósfera.

Gracias al ordenador digital, los años 70 vieron enormes avances en dinámica de fluidos computacional, análisis estructural y de bataneo (flutter), simulación de sistemas dinámicos complejos, y la aplicación de las teoría de guiado y de control en los procesadores de a bordo de las aeronaves. Algunas técnicas de simulación hicieron posible entrenar a los pilotos de forma

realista en tierra, y los sistemas automáticos de control a bordo de las aeronaves permitían simular el comportamiento dinámico de una aeronave totalmente diferente (Figura 2). Los avances en tecnologías de control de vuelo permitieron diseñar aeronaves actuadas totalmente por sistemas eléctricos (fly-by-wire).

Figura 2: "Shuttle Training Aircraft", vehículo de entrenamiento de la NASA que simulaba el comportamiento dinámico del "Space Shuttle" (2003)

Durante los años 80 se aprendió mucho sobre el control de los vehículos hipersónicos. Las trayectorias de los vehículos se deben controlar con mucha precisión, ya que el calor producido por las fuerzas de fricción puede crear temperaturas enormes en puntos críticos del vehículo. El control manual es muy difícil o no factible en la mayoría de las fases de vuelo. La trayectoria se puede controlar comparando mediante realimentación con una trayectoria de referencia previamente computada, o con métodos de predicción de trayectoria a tiempo real.

Puede haber grandes incertidumbres en los coeficientes aerodinámicos de los vehículos hipersónicos, y esto complica el diseño de los sistemas de control automáticos. Estos sistemas deben adaptarse a lo largo del vuelo para compensar los efectos producidos por las enormes variaciones en la dinámica del vehículo a lo largo de la envolvente. Los sensores externos de velocidad y Mach no funcionan correctamente a estas velocidades, por lo que los datos del aire se obtienen usando información derivada del sistema de navegación, y un modelo atmosférico. A estas altitudes pueden existir grandes variaciones de densidad repentinas que son muy difíciles de predecir, por lo que los sistemas de control se tienen que diseñar con capacidad de tolerar estas variaciones.

La era de los verdaderos vehículos “aeroespaciales” introduce nuevos retos para los ingenieros de control. Hoy en día se debe pensar en términos de guiado y control, algoritmos y simulación, y métodos numéricos e implementación digital. Se necesitan muchas técnicas analíticas relativamente nuevas, que incluyen optimización numérica, análisis sensitivos y de robustez ante las variaciones de parámetros, técnicas adaptativas, y control multivariable. El ingeniero de

control no puede trabajar más de una manera aislada; muchas otras tecnologías se integran en el diseño, y muchos campos imponen restricciones (por ejemplo estructural, térmico, propulsivo, administración energética, factores humanos, etc.)

1.4 Objetivos y estructura del trabajo

La mayoría del progreso que se llevó a cabo en análisis de dinámica de vuelo de aviones durante los primeros días de la aviación se consiguió mediante la resolución de ecuaciones algebraicas lineales. Hoy en día es habitual que el análisis del comportamiento dinámico de una aeronave con sus sistemas de control se realice mediante simulaciones no lineales, que se resuelven utilizando métodos numéricos. Este proceso únicamente es eficiente si se utilizan las capacidades de los procesadores digitales modernos.

El objetivo general de este trabajo es "**desarrollar una herramienta de software que permita optimizar sistemas de control de vuelo de aeronaves de manera eficiente**". Este objetivo implica el diseño y desarrollo de dos elementos principales.

- Un **simulador** de vuelo de la aeronave seleccionada. Este simulador se desarrollará mediante la herramienta de software *Simulink®* [2], un entorno gráfico de programación orientado a los procesos de modelado, simulación y análisis de sistemas dinámicos. Para permitir una adecuada visualización de los resultados en tiempo real, se utilizará el simulador de vuelo ***FlightGear*** *Flight Simulator* (nombre comúnmente abreviado a *FlightGear*) como motor gráfico. *FlightGear* es un simulador de vuelo multiplataforma de código abierto gratuito (software libre). El simulador se presenta en el capítulo 3, mientras que el software de visualización se detalla en el capítulo 4.

- Una **interfaz gráfica** que tenga la capacidad de interactuar con el simulador. Para ello, se utilizará la herramienta incluida con el software *MATLAB®* llamada *GUIDE* (*Graphical User Interface Design Environment*) [3][4][5][6]. Esta interfaz facilitará la rápida interacción con el simulador. Se detalla su desarrollo en el capítulo 5.

Con el fin de mostrar las capacidades de la plataforma a lo largo del desarrollo, estos dos elementos principales se aplicarán a una **aeronave** no tripulada, cuyas propiedades se detallan en el capítulo 2.

El esquema que se presenta en la Figura 3 resume y sintetiza la estructura del trabajo, con indicación de los elementos que se presentarán a lo largo de los distintos capítulos.

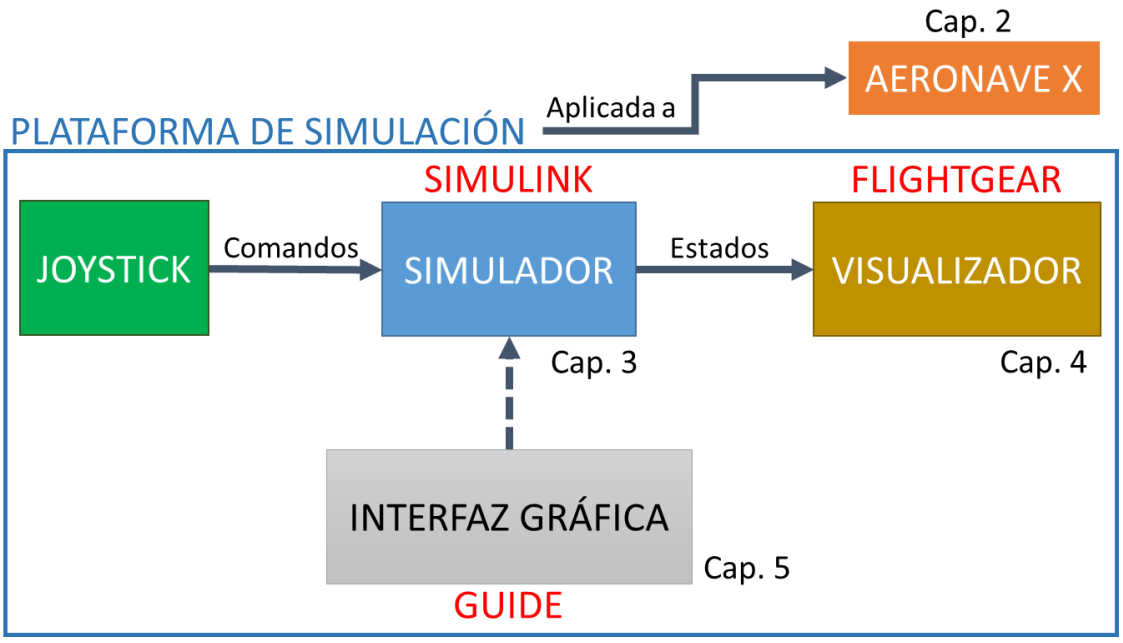

Figura 3: Estructura de la plataforma de simulación

Además de estos capítulos, se incluye un capítulo de **resultados** donde se muestra el comportamiento dinámico de la aeronave X en la plataforma de simulación, un capítulo final de **conclusiones** y dos **anexos**. El primer anexo describe el desarrollo de un simulador de vuelo independiente de la plataforma. Este simulador tiene funcionalidades que no se han incluido en la plataforma. El segundo anexo corresponde a un breve manual de instalación y puesta a punto de la plataforma de simulación.

Capítulo 2

Aeronave seleccionada

Siendo el objetivo del presente trabajo el diseño y desarrollo de una plataforma para la optimización de sistemas de control de vuelo, es necesario en este punto seleccionar una aeronave a simular, que servirá a modo de demostración de las capacidades de la plataforma.

La aeronave seleccionada corresponde a un "Unmanned Aerial Vehicle" (aeronave no tripulada) ficticio, cuyas características aerodinámicas, geométricas y másicas han sido estimadas utilizando como referencia datos de aeronaves reales de características similares. Dado que se trata de una aeronave ficticia, en el presente trabajo se hará referencia a ella como "aeronave X". La aeronave X tiene forma de ala volante, lo que probablemente derive en unas propiedades inerciales no comunes en otros tipos de aeronaves.

Se ha llevado a cabo un proceso de selección y ajuste de las características geométricas, másicas, aerodinámicas, propulsivas y de los parámetros de control para conseguir un adecuado funcionamiento del sistema en su totalidad. Dichas características se exponen a continuación.

2.1 Características geométricas y másicas de la aeronave X

El diseño geométrico de la aeronave X tiene dos propósitos. Por un lado, este modelo geométrico será utilizado en la visualización mediante el motor gráfico FlightGear para conseguir unos resultados visuales más realistas. Por otro, la geometría permitirá estimar las propiedades inerciales de la aeronave como se detallará más adelante.

La aeronave X utilizará propulsión a hélice, aunque ésta no formará parte de la geometría que se va a diseñar, ya que aumenta la complejidad del diseño innecesariamente (se considerará despreciable el efecto de su masa en las propiedades inerciales de la aeronave).

El software que se ha seleccionado para llevar a cabo esta tarea es el software de diseño 3D CATIA V5 [7]. Este software desarrollado por la empresa francesa Dassault Systèmes ofrece

soluciones para múltiples etapas de desarrollo de productos, incluyendo conceptualización, diseño (CAD), ingeniería (CAE) y fabricación (CAM).

A continuación se muestra una sucesión de figuras que representan la evolución en el diseño de la geometría para obtener la geometría final de la aeronave (Figura 4 y Figura 5). La aeronave X tendrá una envergadura de 4 metros.

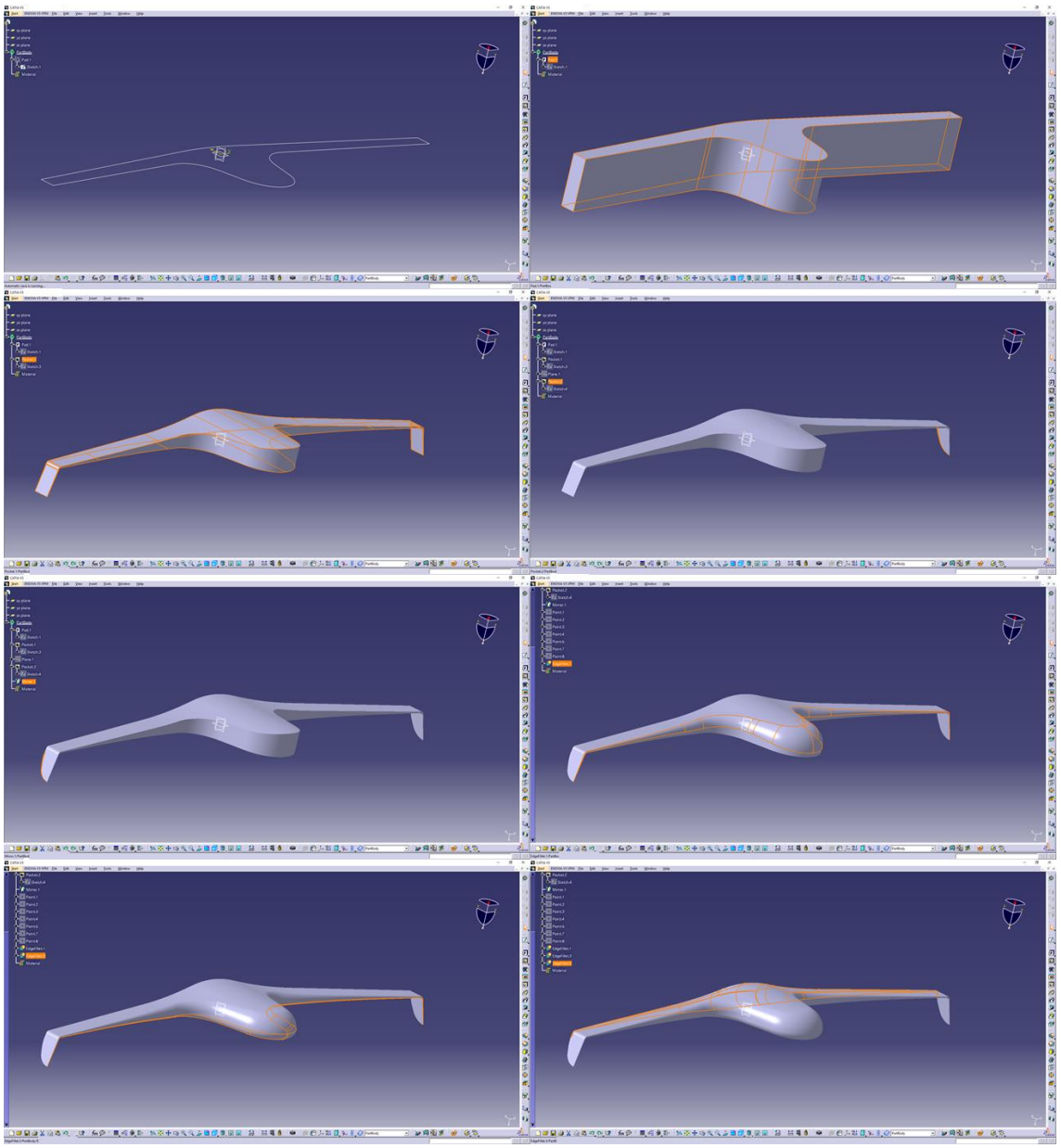

Figura 4: Secuencia del proceso de modelado de la geometría de la Aeronave X (primera parte)

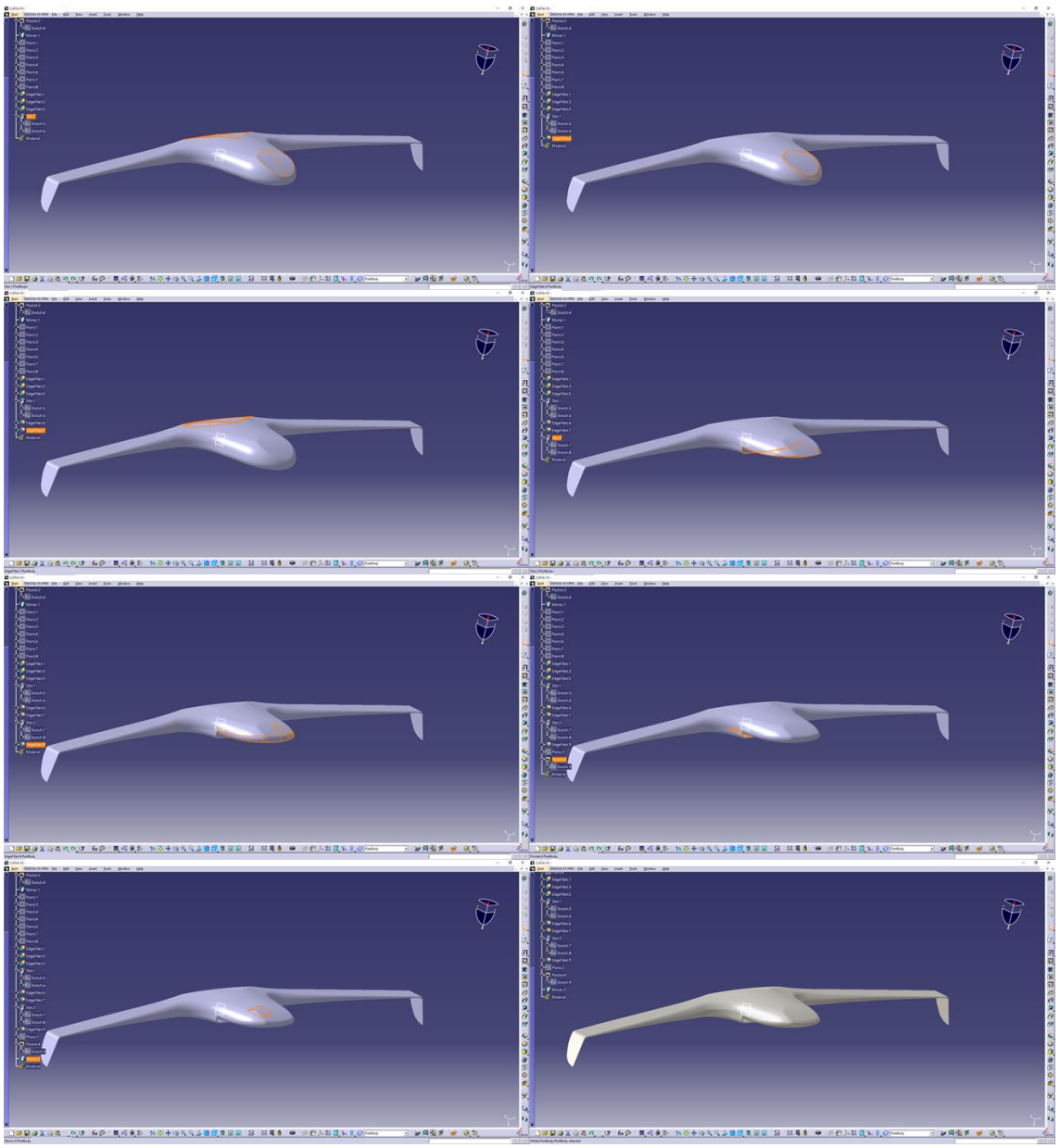

Figura 5: Secuencia del proceso de modelado de la geometría de la Aeronave X (segunda parte)

A continuación se muestra individualmente la imagen de la geometría final (Figura 6).

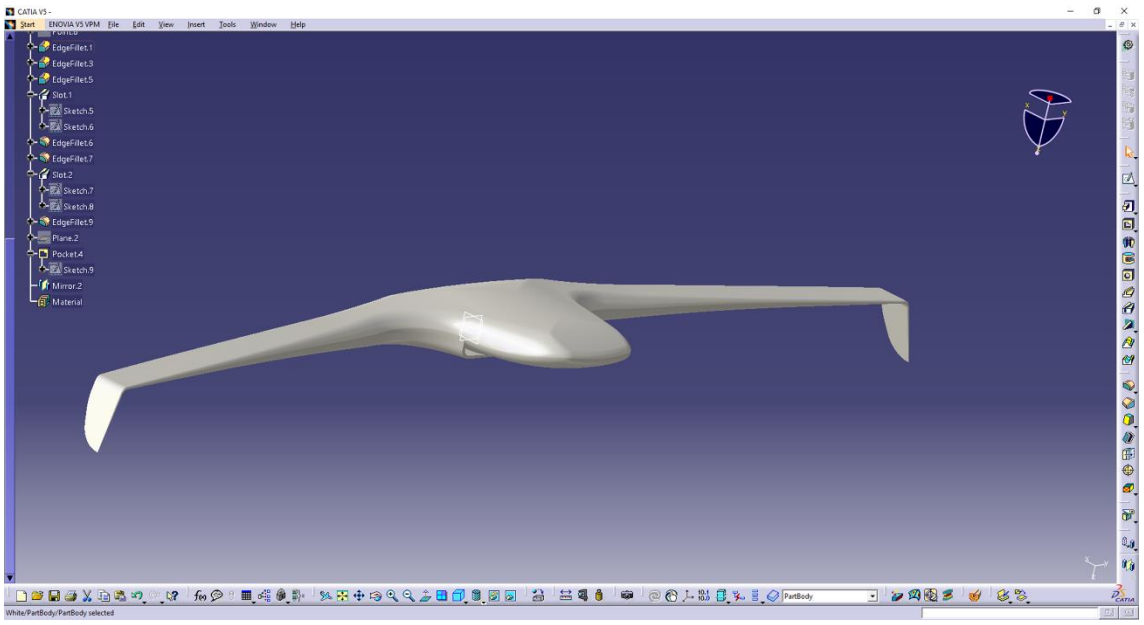

Figura 6: Geometría final de la Aeronave X (Vista trasera)

Una vez que el diseño de la geometría de la aeronave X se ha dado por finalizado se puede proceder al análisis másico de la geometría. Para simplificar el proceso, se van a estimar las propiedades inerciales de la aeronave suponiendo una masa total y una densidad uniforme a lo largo de todo el volumen de la geometría. Se hallarán estas propiedades para dos condiciones másicas diferentes debido a las diferencias en la matriz de inercia que puede ocasionar la cantidad de combustible que posea la aeronave.

(1) Tanque lleno: Masa total de 31 kg (Figura 7).
(2) Tanque casi vacío: Masa total de 22 kg (Figura 8).

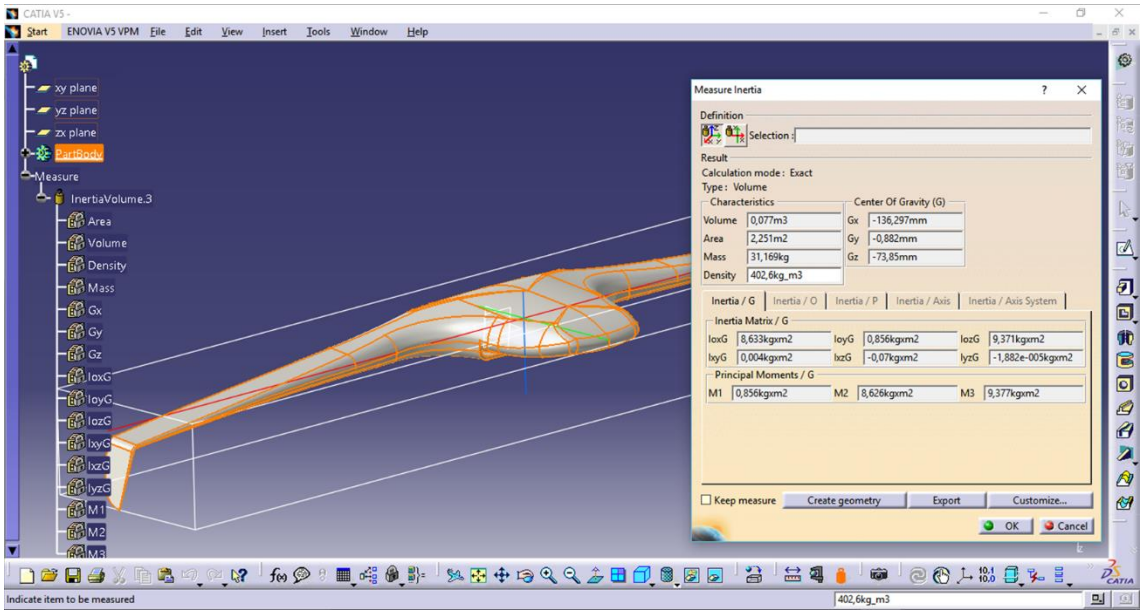

Figura 7: Cálculo inercial de la geometría (tanque lleno)

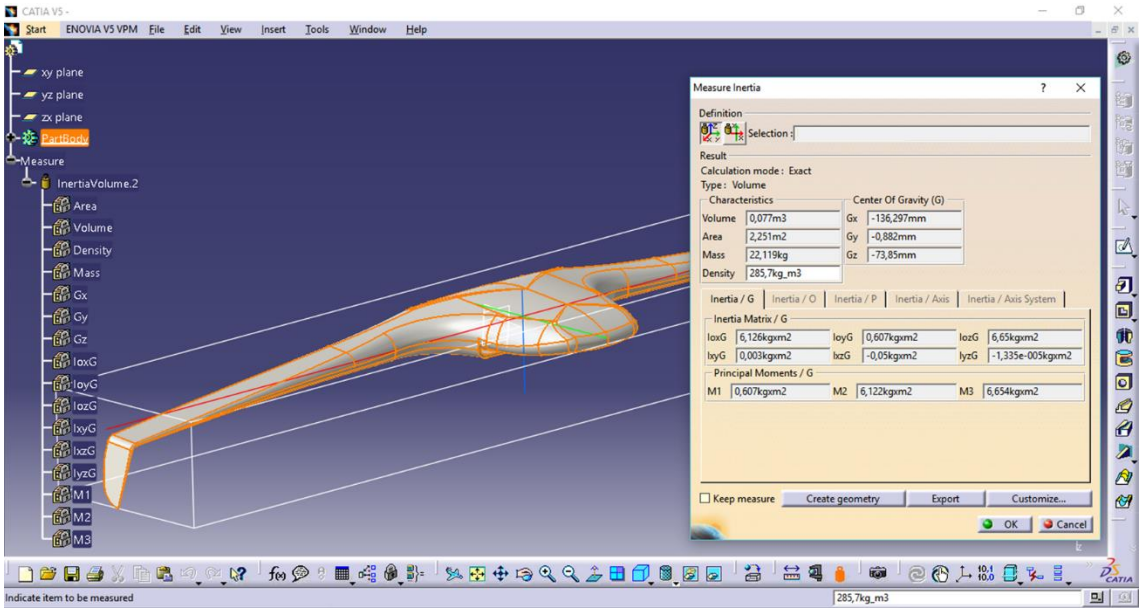

Figura 8: Cálculo inercial de la geometría (tanque vacío)

Debido a la usual simetría dorsal de los aviones, los momentos de inercia I_{XY} e I_{YZ} son pequeños, por lo que no se tendrán en cuenta en el desarrollo de la plataforma. Por tanto los resultados de las características másicas son los siguientes.

Condición másica 1:	Condición másica 2:
$m = 31$ kg	$m = 22$ kg
$I_X = 8.6$ kg·m^2	$I_X = 6.1$ kg·m^2
$I_Y = 0.85$ kg·m^2	$I_Y = 0.61$ kg·m^2
$I_Z = 9.4$ kg·m^2	$I_Z = 6.7$ kg·m^2
$I_{XZ} = -0.05$ kg·m^2	$I_{XZ} = -0.05$ kg·m^2

2.2 Características aerodinámicas, propulsivas y parámetros de control de la aeronave X

Las características aerodinámicas, propulsivas y los parámetros de control de la aeronave X se presentan a continuación. En el capítulo 3 se expone el contexto en el que se van a utilizar dichos parámetros.

Características propulsivas:	Características geométricas:
$T_{max} = 70$ N	$S_w = 1.068$ m^2
$V_{max} = 41.7$ m/s	$b = 4$ m

Características aerodinámicas:

$C_{L_0} = 0.57$ $C_{D_0} = 0.0289$ $C_{Y_\beta} = -0.316$

$C_{L_\alpha} = 6.19$ $C_{D_{\alpha^2}} = 0.2$ $C_{Y_{\delta r}} = 0$

$C_{L_q} = 0$

$C_{l_\beta} = -0.0413$ $C_{m_0} = 0.095$ $C_{n_\beta} = 0.046$

$C_{l_{\delta a}} = -0.13$ $C_{m_\alpha} = -0.95$ $C_{n_{\delta a}} = 0$

$C_{l_{\delta r}} = 0.03$ $C_{m_{\delta e}} = -0.725$ $C_{n_{\delta r}} = -0.05$

$C_{l_p} = -0.71$ $C_{m_q} = -8.88$ $C_{n_p} = -0.084$

$C_{l_r} = 0.116$ $C_{m_{\dot{\alpha}}} = -10$ $C_{n_r} = -0.021$

Capítulo 3

Desarrollo de un simulador de vuelo

3.1 Estructura general del simulador

Una aeronave es un sistema muy complejo. Para simularla es necesario considerar al Sistema como una unión de subsistemas interconectados. Generalmente, se suelen identificar 5 subsistemas diferentes: Comandos, Sistema de Control de Vuelo, Modelo de aeronave, Ambiente externo y Sensores. Además se añade un bloque de Visualización que facilita la interpretación de los resultados del simulador. En la Figura 9 se esquematiza la estructura general del simulador.

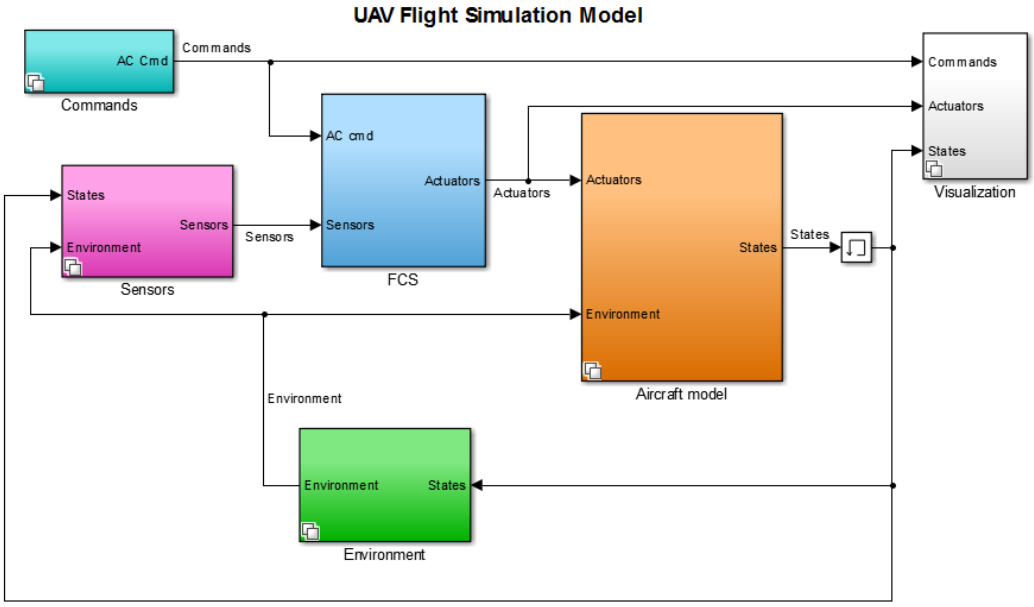

Figura 9: Estructura del simulador

El simulador se ha desarrollado utilizando la herramienta de software y entorno de desarrollo Simulink®, un entorno de programación gráfico para modelado, simulado y análisis de sistemas dinámicos. El subsistema de comandos transmite las señales obtenidas del joystick al sistema de control de vuelo. El sistema de control de vuelo utiliza algoritmos de control para calcular los movimientos de los actuadores de las superficies de control utilizando como input las señales de comandos. Las señales de los actuadores viajan al subsistema del modelo dinámico de la aeronave, que simula las propiedades dinámicas de esta y calcula la evolución de sus estados, utilizando como datos los movimientos de los actuadores y también las propiedades del ambiente externo de la aeronave, provenientes de otro subsistema llamado “Environment”. Dicho subsistema, además de proveer datos de entrada al modelo dinámico, toma como inputs los estados de la aeronave en el ciclo de simulación anterior. El retraso a un sólo paso de simulación anterior se consigue utilizando el bloque “Memory” [8] (Figura 10).

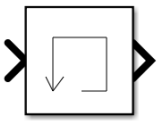

Figura 10: Bloque "Memory"

Los estados del ambiente también se envían a un subsistema que simula los errores de medición de los sensores de la aeronave, que además toma como entrada los propios estados de ésta. Las señales que da como salida este subsistema corresponden por tanto a las medidas (erróneas) de los estados de la aeronave que tomarían los sensores en la aeronave real. Estas señales se envían a su vez al sistema de control de vuelo, que necesita la medida de los estados para su algoritmo de control. Por último, se envían las señales de comandos, actuadores y estados al subsistema de visualización para posibilitar su posterior representación gráfica.

La frecuencia de ejecución del simulador [9] se ha establecido en 200 Hz, es decir, cada 0.005 segundos se ejecuta todo el modelo. Cada bloque de Simulink® se ejecuta solamente una vez que haya recibido todos sus inputs correspondientes.

Como se aprecia en la figura 10 algunos de los bloques presentan un símbolo (dos cuadrados solapados) en su margen inferior izquierdo. Este símbolo indica que se ha utilizado la función de “subsistema variante” (Variant Subsystem) que permite múltiples implementaciones para un subsistema donde únicamente una de las implementaciones está activa durante la simulación [10]. Es posible cambiar la implementación activa mediante código MATLAB®. Esta función es muy útil porque permite unificar en un mismo modelo distintos subsistemas sin necesidad de que se ejecuten simultáneamente.

El principal cometido del simulador es actuar como una planta cuyos inputs son los comandos introducidos mediante el joystick, entre otros dispositivos, y cuyos outputs corresponden a los estados de vuelo del avión en cada instante. Estos estados se visualizan mediante el motor gráfico FlightGear. Además, como se indicó en la introducción, se ha desarrollado una interfaz gráfica que tiene la capacidad de interactuar con el simulador. Esta interfaz se explicará en detalle en el capítulo 5.

La estructura principal del simulador se ha obtenido a partir de un ejemplo incorporado en el paquete de software MATLAB® [11]. Prácticamente todos los bloques han sido modificados para adaptarlos a este trabajo aunque existen algunos en los que se ha mantenido la estructura original.

Para desarrollar este simulador, se han utilizado muchos bloques pertenecientes al "Aerospace Blockset" de la librería de bloques de Simulink® [12]. Este Blockset (o grupo de bloques), contiene bloques prediseñados para simulación y modelado de aeronaves, naves espaciales y sistemas de propulsión.

3.2 Desarrollo de los subsistemas del simulador

En este apartado, se detalla cada subsistema del simulador siguiendo el orden en que los encontraría una señal virtual que partiera desde el primero (Commands) hasta el último (Visualization). En todos los casos se representará gráficamente el bloque al que nos estamos refiriendo.

3.2.1 Comandos

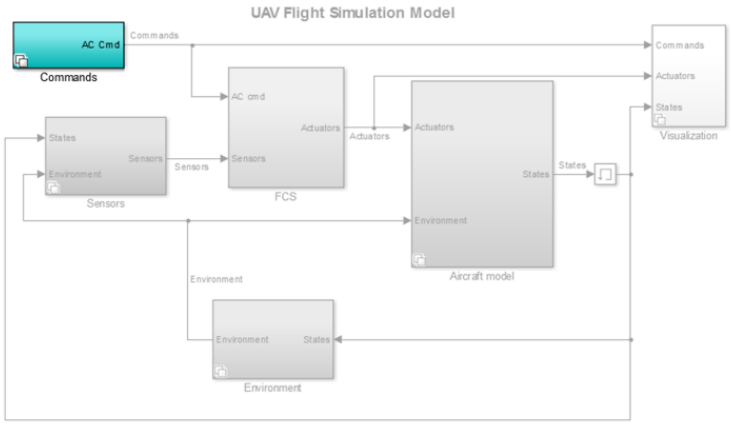

Figura 11: Ubicación del subsistema de comandos en el simulador

El subsistema "commands" es el primero que se ejecuta en el simulador. Utiliza la función de Subsistemas Variantes que como ya se ha indicado permite incorporar en un mismo bloque varios subsistemas manteniendo activo únicamente el seleccionado. En este caso se han incorporado cuatro subsistemas: "Joystick", "Signal Builder", "Steering Wheel" y "Data". De todos ellos el más importante es el subsistema "Joystick" quedando los demás como alternativas si se carece de Joystick o si se dispone de datos de entrada medidos en un vuelo real.

3.2.1.1 Subsistema "Joystick"

El bloque Pilot Joystick pertenece al Aerospace Blockset de la librería de bloques de Simulink®. Este bloque provee una interfaz rápida y simple para operar un joystick conectado al ordenador con el objetivo de controlar el avión en tiempo real introduciendo comandos en la simulación

[13]. Este bloque está preconfigurado para funcionar correctamente con muchos joysticks comerciales, por lo que no suele ser necesario realizar ningún ajuste adicional. En el simulador este bloque está configurado con la opción de 4 outputs, uno para cada eje del joystick. Estos ejes pueden controlar los siguientes grados de libertad del avión: Roll (balance), Pitch (cabeceo) y Yaw (guiñada), además de la posición del Throttle (variador de empuje). Los outputs de este bloque se muestran en la Tabla 2:

Tabla 2: Señales obtenidas del Joystick

	Movim. Joystick	Output	Movim. Joystick	Output
Roll	Despl. izquierda	−1	Despl. derecha	1
Pitch	Despl. arriba	−1	Despl. abajo	1
Yaw	Giro antihorario	−1	Giro horario	1
Throttle	Despl. arriba	1	Despl. abajo	0

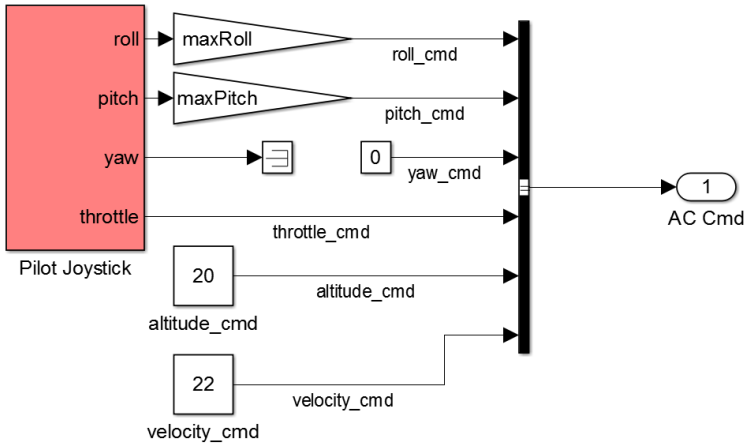

Figura 12: Subsistema "Joystick"

Para limitar el pitch y el roll comandados, se han añadido bloques de ganancias en sus señales. Estas ganancias, maxRoll y maxPitch, se definen en el script externo “StartVars.m” que es necesario ejecutar antes de ejecutar el simulador para llevar a cabo el proceso de inicialización.

Como se aprecia en el esquema de la Figura 12 se ha anulado la señal de yaw (siempre da un valor nulo). Esto se debe a que la aeronave seleccionada, al tratarse de un ala volante, no dispone de superficies de control específicamente diseñadas para controlar este grado de libertad (timón de dirección), es decir, en el caso de esta aeronave el ángulo de guiñada sólo puede controlarse mediante los alerones.

También es apreciable en el esquema de la Figura 12 que los bloques de altitude_cmd y velocity_cmd son constantes. Sin embargo, los valores de estas constantes se pueden modificar utilizando la interfaz gráfica interactiva con el simulador.

Las seis señales se juntan en un bus mediante el bloque “Bus Creator” [14]. Esto permite extraer del bus la señal requerida más adelante manteniendo el modelo libre de un número excesivo de rutas de señales.

3.2.1.2 Subsistema “Signal Builder”

Este subsistema existe como alternativa al subsistema “Joystick”. Permite diseñar mediante una interfaz sencilla la evolución temporal de los valores de los cuatro ejes que se definieron en el apartado anterior además de la altitud y velocidad comandadas . Finalmente las seis señales se unen en un bus de manera similar al subsistema explicado en el apartado anterior (Figura 13).

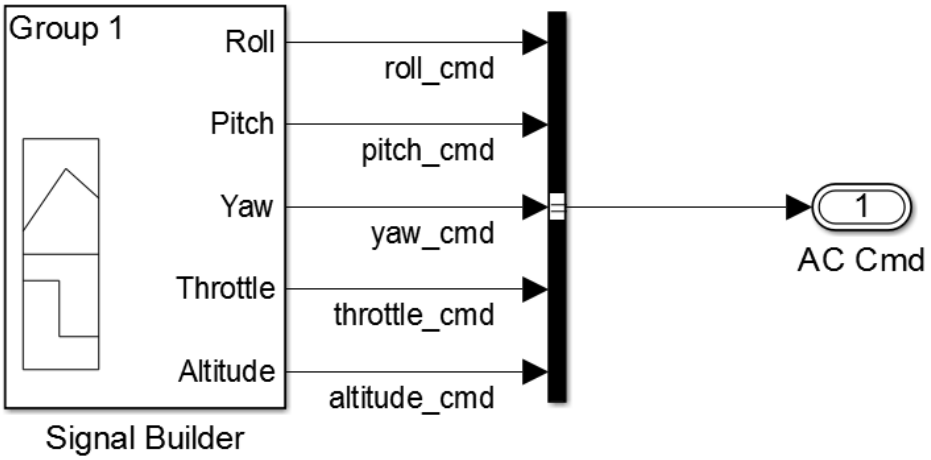

Figura 13: Subsistema "Signal Builder"

Dicha interfaz de construcción de señales se muestra a continuación en la Figura 14.

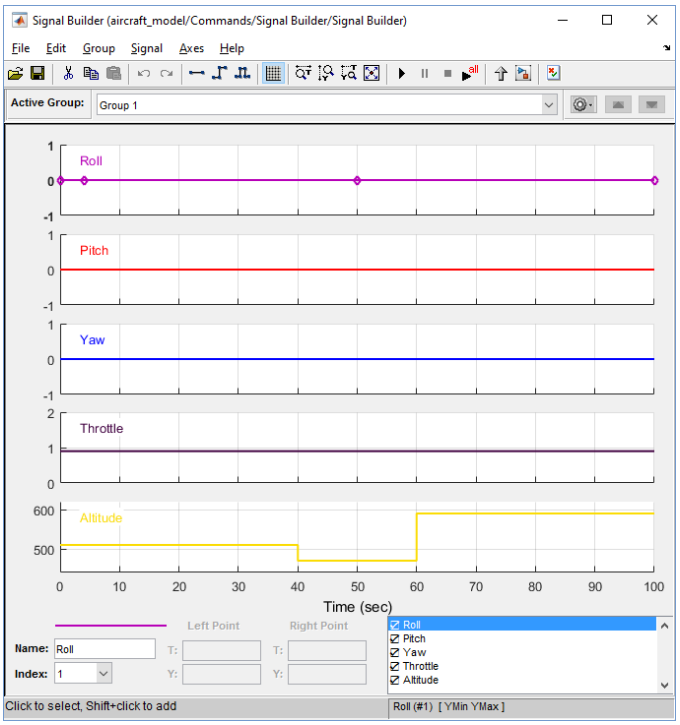

Figura 14: Interfaz de creación de señales "Signal Builder"

3.2.1.3 Subsistema "Data"

Este subsistema permite introducir los comandos de entrada a partir de un archivo externo. El procedimiento a seguir es definir las señales en MATLAB® con formato "timeseries" o "matrix". A continuación guardar el "Base Workspace" y enlazar el archivo creado con el bloque "From File".

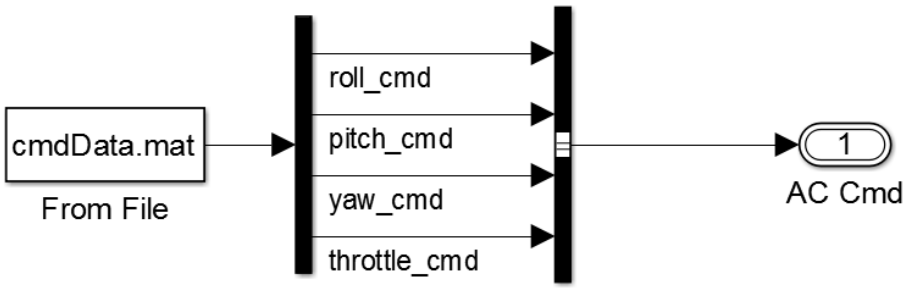

Figura 15: Subsistema "Data"

3.2.2 Sistema de Control de Vuelo

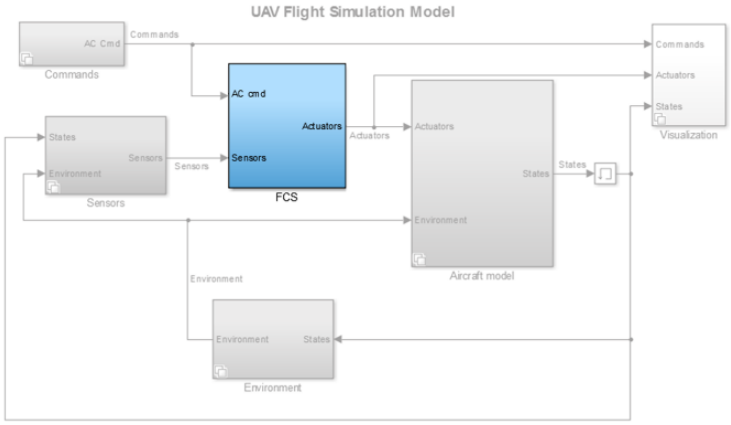

Figura 16: Ubicación del subsistema "FCS" en el simulador

El bloque FCS (“Flight Control System”) está compuesto en un primer nivel por los elementos que se muestran en la Figura 17. El controlador de la aeronave se encuentra dentro del subsistema “Controller”. El resto de los subsistemas están destinados a albergar filtros o funciones de transferencia que modelen interferencias en el sistema de control en caso de que se produzcan en la aeronave real. En el simulador de este trabajo no se han incluido filtros en ninguno de estos bloques, pero se mantienen por si fuera necesario incluirlos.

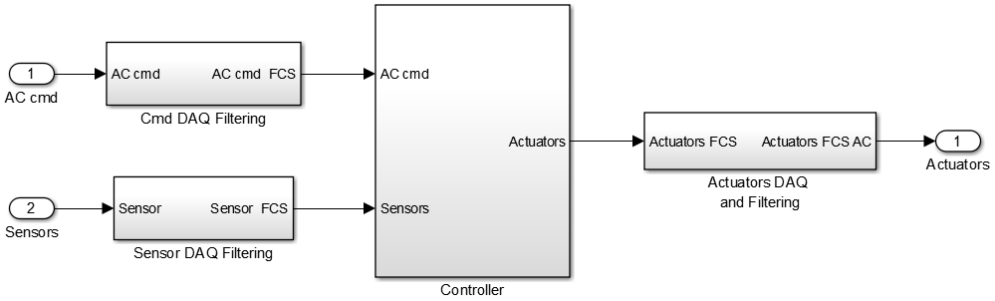

Figura 17: Estructura del subsistema "FCS"

A continuación se muestra en la Figura 18 la estructura del controlador de vuelo del simulador, el subsistema “Controller”. Se puede comprobar, como veíamos en el nivel anterior, que el controlador posee dos inputs para dar un output. Los inputs son (1) “Commands” ; (2) “States”. Como se ha indicado anteriormente, el Sistema de control de Vuelo intenta llevar a la aeronave a los estados comandados teniendo en cuenta en cada instante los estados actuales de la aeronave [15][16]. Mediante leyes de control que serán explicadas a continuación, el controlador dará como resultado outputs a los actuadores de las superficies de control de la aeronave, que a su vez constituirán los inputs del bloque siguiente (“Aircraft Model”).

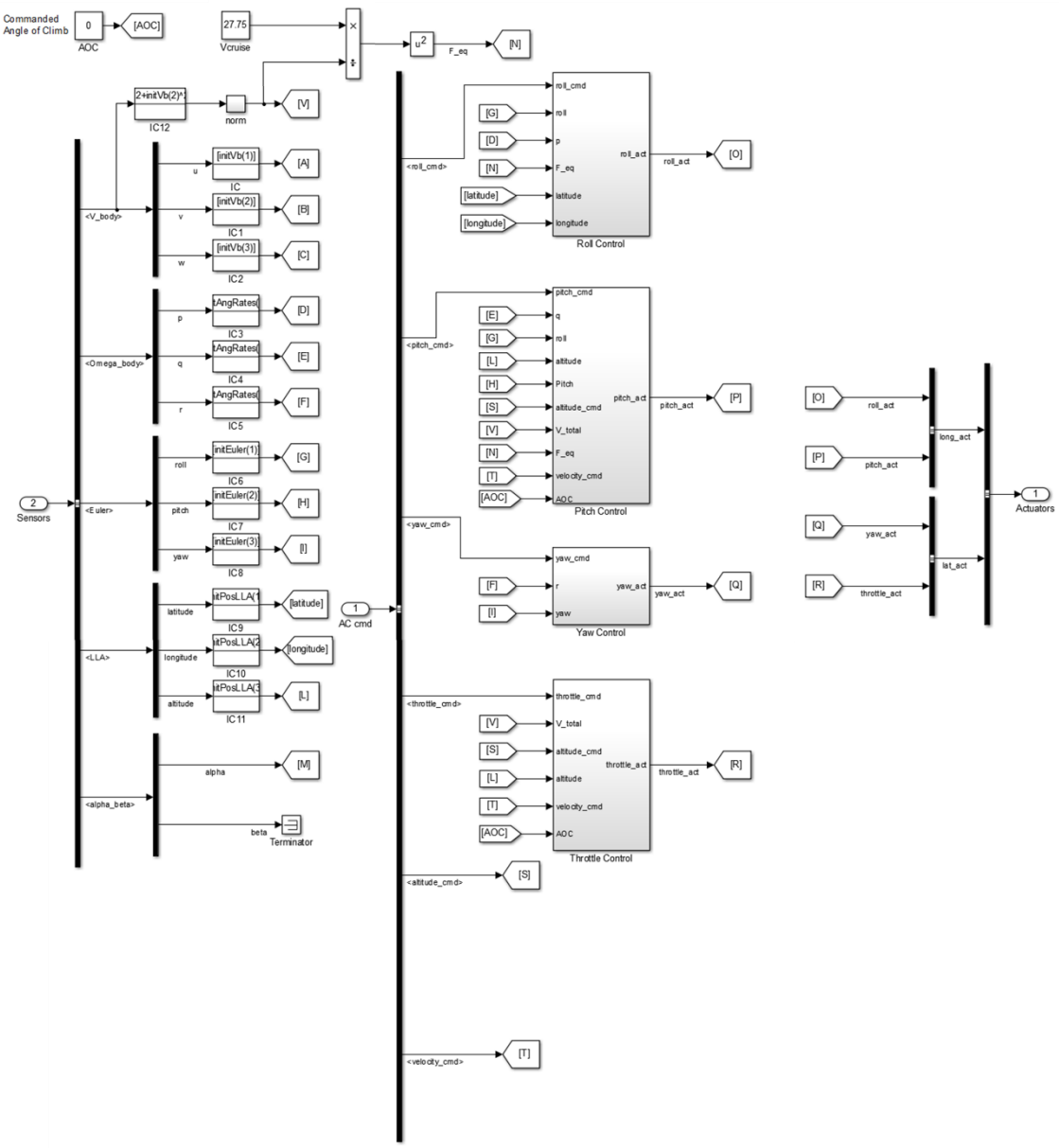

Figura 18: Estructura del subsistema "Controller"

En el diagrama de la Figura 18 se incluyen bloques que se aplican a cada uno de los estados con el objetivo de establecer valores iniciales para éstos durante la primera ejecución del simulador [17]. Son los bloques que llevan la denominación IC, IC1, IC2, … Los valores iniciales se definen en el script de inicialización “StartVars.m”.

En este diagrama se introducen un tipo de bloques que ha sido muy utilizado en todo el simulador. Se trata de los bloques “GoTo” y “From”. Como se muestra a continuación el bloque “GoTo” es un pentágono que apunta a la izquierda mientras que el “From” apunta a la derecha. Estos bloques sirven para conectar dos puntos de un modelo sin necesidad de que haya una ruta de señal entre ellos y de esta manera economizar el espacio disponible. Las señales que entran en el bloque “GoTo” se almacenan en una memoria virtual y prosiguen su camino en el caso de que exista un bloque “From” con el mismo nombre que el “GoTo” donde entró esa misma señal.

En el diagrama de la Figura 18, estos bloques se han utilizado para redireccionar las señales de los estados de la aeronave hacia los bloques de control que también se alimentan de los comandos de entrada.

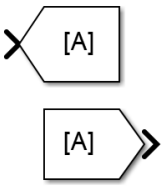

Figura 19: Bloques "GoTo" y "From"

En la parte superior izquierda de la Figura 18 se incluyen una serie de bloques que calculan un factor a partir de la velocidad de la aeronave. Este factor se utilizará más adelante en los controles de cada grado de libertad, y por tanto se explicará con más detalle.

A continuación se mostrará cómo se ha modelado el control de cada grado de libertad de la aeronave. Como es habitual, se han dividido los controles en cuatro grupos: Roll, Pitch, Yaw y Throttle.

3.2.2.1 Roll Control

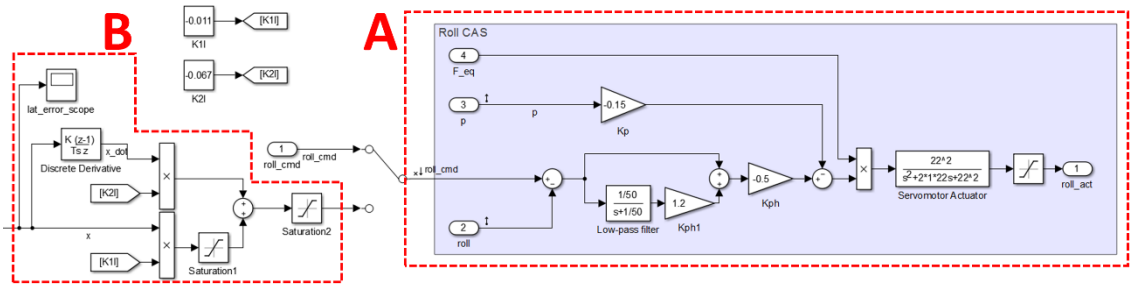

Figura 20: Estructura del subsistema "Roll Control"

En la Figura 20 se puede observar el subsistema del control de Roll. Se han señalado dos áreas del subsistema que realizan distintas funciones.

La **zona A** corresponde al "Control Augmentation System" (CAS) de roll. Este sistema está siempre activo. Se encarga de que la aeronave responda de forma adecuada a los comandos de roll introducidos por el piloto. Se pueden distinguir varios componentes dentro de este sistema. El núcleo del CAS lo constituye el llamado "error de roll", es decir, la diferencia existente entre el roll comandado y el roll real de la aeronave. Este error introduce una señal en el actuador de los alerones que tiende a disminuir el propio error. Al error de roll se le aplica un filtro paso bajo ("Low-Pass Filter"). Este tipo de filtros permite el paso de las frecuencias más bajas y atenúa las frecuencias más altas. Después se multiplica por una ganancia (Kph1). A la señal resultante se le suma el propio error de roll y seguidamente se le vuelve a aplicar otra ganancia (Kph).

La estructura que se acaba de describir constituye el núcleo del sistema de control del simulador y será muy utilizado a lo largo de su desarrollo en todos los bucles (Figura 21).

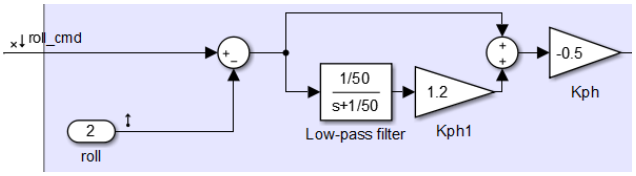

Figura 21: Detalle de la estructura del subsistema "Roll Control"

A la señal resultante de esta estructura se le suma posteriormente la velocidad angular del avión respecto al eje x (p) multiplicada por una ganancia negativa (Kp). Esto consigue un efecto de amortiguamiento deseable en la respuesta de roll. La efectividad de las superficies de control es proporcional a la presión dinámica, que a su vez es proporcional al cuadrado de la velocidad de la aeronave. Para compensar este efecto se utiliza se multiplica la señal por un factor de ecualización que viene de un nivel anterior (bloque "controller"). Este factor se calcula comparando la velocidad total de la aeronave con una velocidad de referencia:

$$F_{eq} = \left(\frac{V_{referencia}}{V} \right) \tag{1}$$

En este caso, se tomará como velocidad de referencia la velocidad media entre la velocidad de entrada en pérdida (17,5 m/s) y la velocidad que nunca debe excederse (38 m/s), lo que da una velocidad de 27,75 m/s. Este factor se aplica tanto en roll como en pitch, como se verá más adelante.

Para simular los alerones, la señal pasa por dos bloques. El **primero** de ellos, "Servomotor actuator", corresponde al modelo dinámico de los servoactuadores que mueven los alerones de la aeronave. En este caso se trata de un modelo de segundo orden, con una frecuencia natural de 22 rad/s y un factor de amortiguamiento de 1, que proporciona por tanto respuestas críticamente amortiguadas (2). Se muestra la respuesta del bloque a un escalón de entrada de magnitud uno en la Figura 22.

$$\frac{\omega_n^{\,2}}{s^2 + 2 \cdot \zeta \cdot \omega_n + \omega_n^{\,2}} \qquad \omega_n = 22 \text{ rad/s} \qquad \zeta = 1 \tag{2}$$

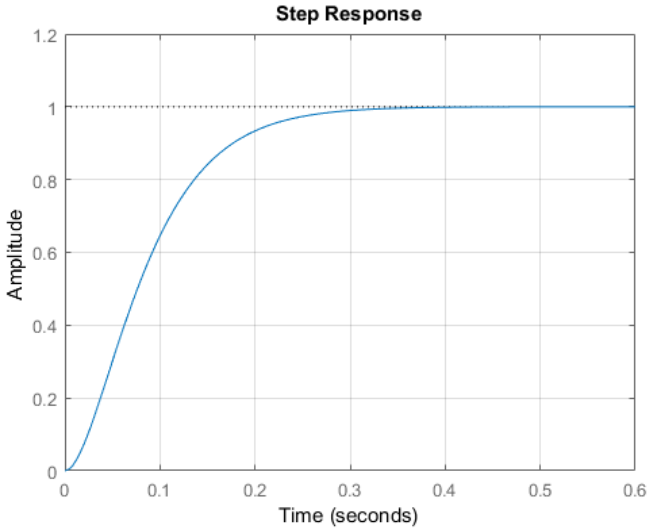

Figura 22: Diagrama de respuesta a escalón del bloque "Servomotor Actuator"

Se puede observar que el actuador tiene un tiempo de respuesta corto, de unos 0,3 segundos, requisito necesario para conseguir una aeronave controlable y ágil. Como se verá más adelante, los servoactuadores del elevador poseen el mismo modelo dinámico que los de los alerones. El **segundo** de los bloques que compone el modelo dinámico de los alerones corresponde a un bloque de "Saturation", que limita la señal a los valores angulares máximos y mínimos que es posible aplicar a los alerones, en este caso +/- 0,55 radianes. Este valor puede modificarse rápidamente a través de la interfaz gráfica que interactúa con el simulador (capítulo 5).

La **zona B** corresponde al sistema de guiado de la aeronave. El input de esta estructura (señal denominada "x") corresponde al error lateral en posición con respecto a una línea de referencia fijada a la superficie terrestre. Este error se mide calculando la mínima distancia entre el punto subavión y la línea de referencia. Tras explicar el funcionamiento del sistema de guiado se explicará cómo se calcula este error lateral y cómo se establece la línea de referencia deseada en cada momento.

El sistema de guiado funciona de la siguiente manera. El objetivo del sistema es enviar un determinado comando de roll como input al CAS (zona A) en función del error lateral que exista entre la aeronave y la línea de referencia.

El error lateral se multiplica por una ganancia negativa, lo que consigue que la tendencia sea siempre a acercarse hacia la línea. Además se aplica un bloque de "Saturation" para limitar esta señal, ya que no tiene sentido comandar un roll excesivamente grande cuando el error lateral es muy alto

Sin embargo, para limitar el máximo ángulo que la aeronave forma con la línea de referencia, también se tiene en cuenta la derivada del error lateral, señal que se obtiene aplicando un bloque de "Discrete derivative". La derivada del error lateral también se multiplica por una ganancia negativa. El resultado de aplicar esta señal se opone al resultado obtenido con el error lateral sin derivar, por lo que se consigue que el sistema se equilibre y llegue a un estado

estacionario y por tanto a un ángulo fijo con respecto a la línea. Conviene que este ángulo no sea demasiado grande, lo que provocaría una respuesta muy lenta que podría incluso llegar a ser inestable.

De esta manera cuando el sistema de guiado esté activado la aeronave tendrá capacidad para mantenerse automáticamente en la trayectoria definida por la línea de referencia. Como se verá en el capítulo 5, el sistema de guiado puede activarse a través de la interfaz gráfica.

A continuación se explicará cómo se calcula el error lateral y cómo se establece la línea de referencia deseada en cada momento. El bloque encargado de esta tarea (Figura 23) está unido al sistema de guiado, y da como output la señal "x", el error lateral, que es enviado como input a dicho sistema.

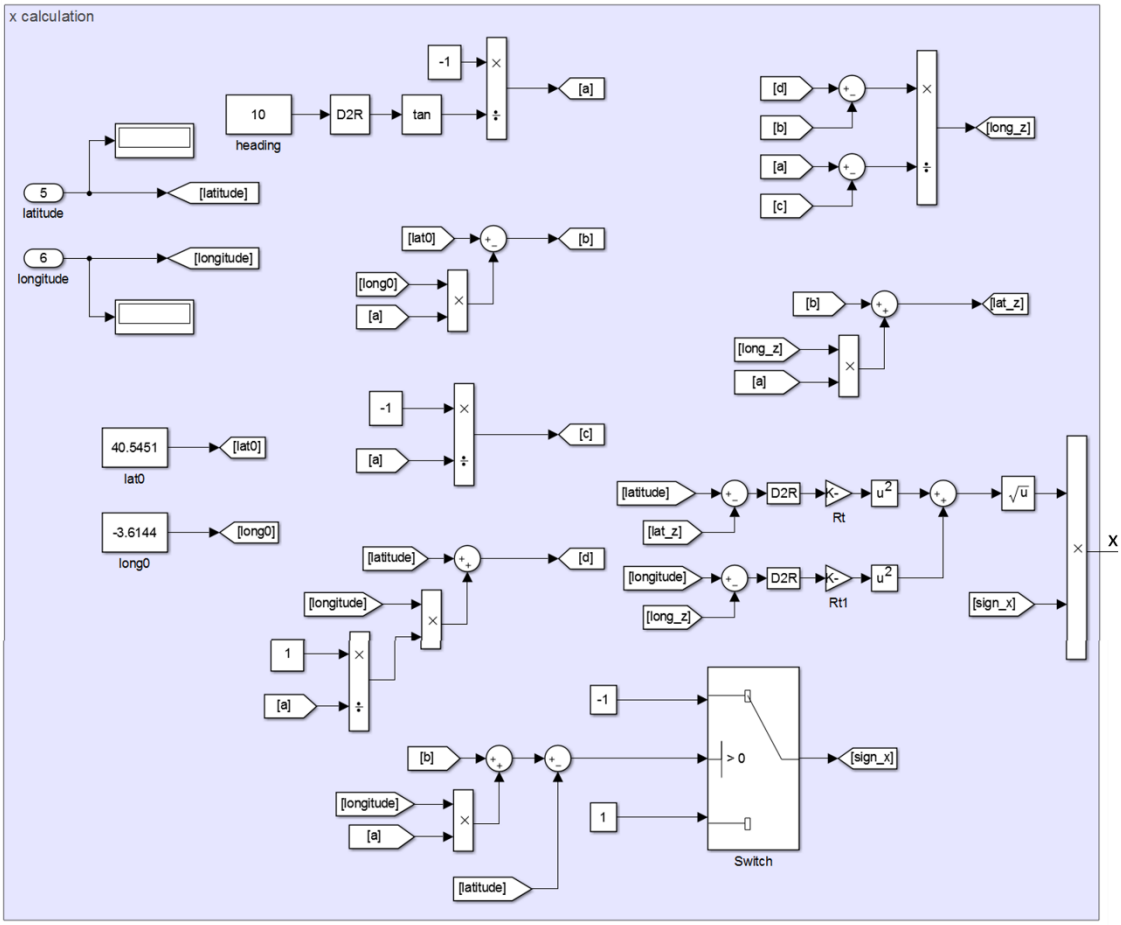

Figura 23: Estructura del cálculo del error lateral

Antes de proceder al cálculo del error lateral, es necesario definir una línea de referencia. Esta definición se realiza con ayuda de la interfaz gráfica. En el instante en que el usuario activa el sistema de guiado, la interfaz actualiza los valores de los bloques de constantes "lat0" (con la latitud terrestre del avión en ese instante), "long0" (con la longitud terrestre del avión en ese instante) y "heading" (con un valor previamente definido en la interfaz). La línea de referencia queda entonces definida como la línea contenida en la superficie terrestre que pasa por el punto

de coordenadas latitud “lat0”, longitud “lon0” y altitud nula y apunta en la dirección que forma un ángulo de valor “heading” en grados con el norte terrestre medidos hacia el este (Figura 24).

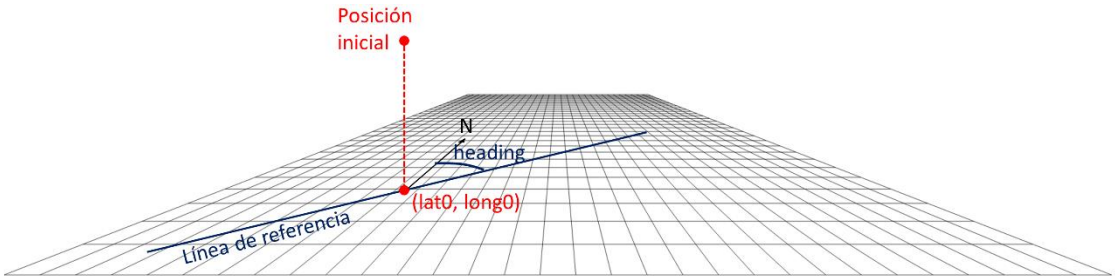

Figura 24: Procedimiento de creación de la línea de referencia

Una vez definida la línea de referencia se procede al cálculo del error lateral de la siguiente manera. Como se puede apreciar en la Figura 24 este método se ha desarrollado utilizando la hipótesis de tierra plana, ya que la curvatura terrestre es despreciable si se considera el alcance típico de las aeronaves de estas características. Por tanto la línea de referencia corresponde a una recta.

Recta de referencia: $lat = a \cdot long + b$

$$\boxed{a = \frac{1}{tg(heading)}} \tag{3}$$

$$\boxed{b = lat_0 - long_0 \cdot a} \tag{4}$$

Recta perpendicular: $lat = c \cdot long + d$

$$\boxed{c = -\frac{1}{a}} \tag{5}$$

$$lat = -\frac{1}{a} \cdot long + d$$

$$\boxed{d = lat + \frac{1}{a} \cdot long} \tag{6}$$

$$\begin{cases} lat_Z = a \cdot long_Z + b \\ lat_Z = c \cdot long_Z + d \end{cases}$$

$$\boxed{long_Z = \frac{d - b}{a - c}} \tag{7}$$

$$lat_Z = a \cdot \frac{d-b}{a-c} + b \tag{8}$$

$$x = \sqrt{(lat - lat_Z)^2 + (long - long_Z)^2} \tag{9}$$

Cuando se refiere a la latitud y longitud, el subíndice Z representa el punto de intersección entre la recta perpendicular a la recta de referencia que pasa por el punto subavión en ese instante y la propia recta de referencia. Cuando no tienen subíndice se refiere a la latitud y longitud de la aeronave en ese instante, señales que entran al sistema por realimentación de los estados.

3.2.2.2 Pitch Control

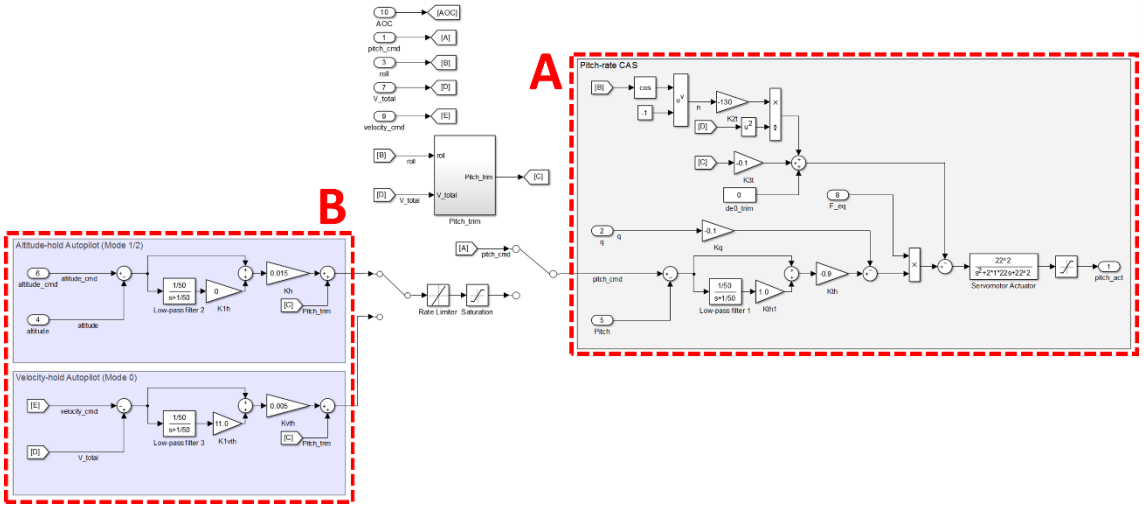

Figura 25: Estructura del subsistema "Pitch Control"

El sistema de control de pitch contiene dos estructuras principales. Para una mejor visualización se han incluido imágenes individuales de cada zona (Figura 26 y Figura 28).

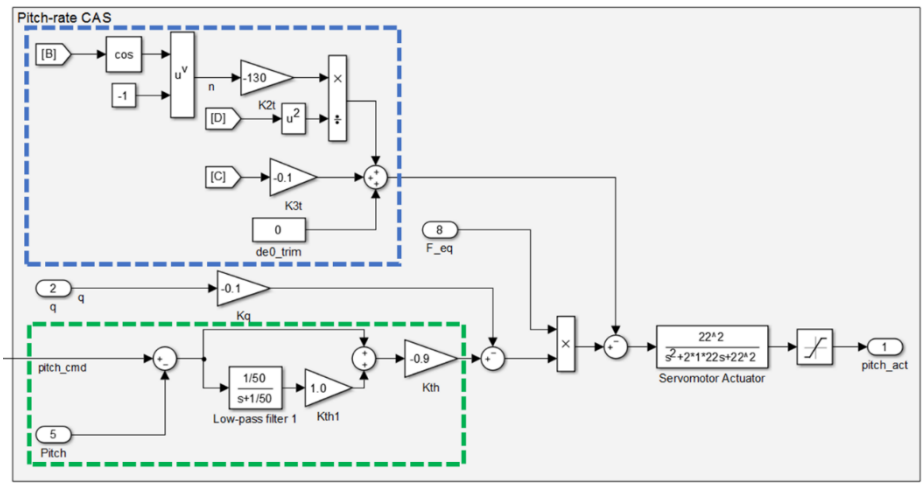

Figura 26: Zona A del subsistema "Pitch Control"

La **zona A** corresponde al "Control Augmentation System" (CAS) del control de pitch. El núcleo de este sistema es idéntico al que se utilizó en el CAS de roll (señalado en verde).

Además, similarmente al CAS de roll, se ha utilizado la velocidad angular (en este caso q) para amortiguar la respuesta en pitch y se ha aplicado el factor de ecualización que compensa el efecto del cambio de efectividad por la presión dinámica.

En este sistema de control se ha incluido una estructura que corresponde al trim de elevador (señalada en azul). Esta estructura es necesaria para aportar al elevador el ángulo necesario para que la aeronave mantenga una condición de vuelo estable. Para ello, se tienen en cuenta dos factores proporcionalmente: La velocidad elevada al cuadrado y el factor de carga ocasionado por el ángulo de roll. Pertenece también a esta estructura la señal que entra a través del bloque "From" con etiqueta "C" proveniente del subsistema "Pitch Trim", que se expone a continuación (Figura 27).

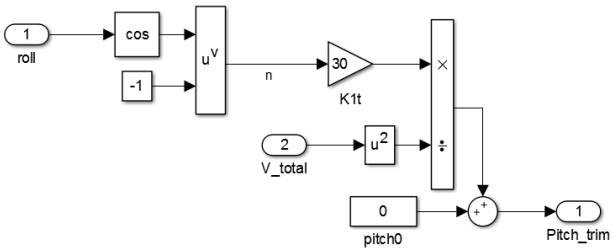

Figura 27: Subsistema "Pitch Trim"

El elevador se ha modelado de manera similar a los alerones en el CAS de roll. El bloque "Servomotor Actuator" corresponde al modelo dinámico de los actuadores del elevador, idéntico al de los actuadores de los alerones. También se ha aplicado un bloque "Saturation" que limita los ángulos posibles en el elevador.

A continuación se explicará el funcionamiento de la **zona B**. Esta zona forma parte del autopiloto de la aeronave (Figura 28).

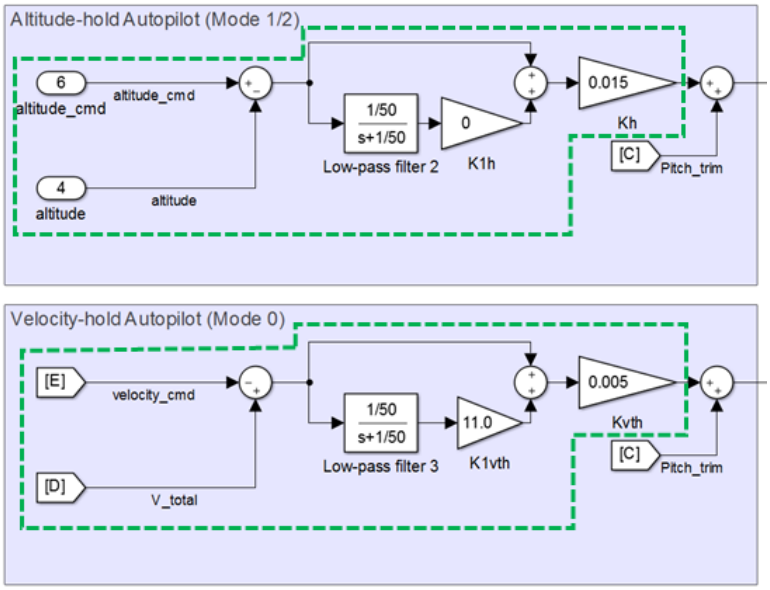

Figura 28: Zona B del subsistema "Pitch Control"

El autopiloto de esta aeronave puede funcionar en dos modos de vuelo distintos.

- **Modo 1**: Cuando el modo 1 está activado, la altitud se controla mediante el pitch (elevador) mientras que la velocidad se controla regulando la potencia de salida del motor (throttle). El modo 1 proporciona una respuesta rápida y precisa a los comandos de altitud y velocidad. Sin embargo presenta un inconveniente cuando el avión vuela a baja velocidad, próxima a la velocidad de entrada en pérdida, ya que si el proceso de entrada en pérdida llegara a suceder la altitud sería incontrolable con el elevador. Por tanto en situaciones como despegue y aterrizaje es necesario activar otro modo de vuelo.

- **Modo 0**: Este es el modo alternativo que resuelve el problema planteado. En este modo la altitud se controla regulando la potencia de salida del motor (throttle) mientras que la velocidad se controla mediante el pitch. Este modo es más lento, menos estable y menos preciso que el modo 1 pero es necesario durante el despegue y el aterrizaje.

El núcleo de la estructura de los autopilotos (señalado en verde) es una vez más la estructura básica de control que se mencionó en el apartado 3.2.2.1. Para compensar efectos de la presión dinámica y del factor de carga se suma a los comandos de los autopilotos la señal obtenida del bloque “Pitch Trim”, que también fue utilizada en el trim de elevador.

Antes de introducir el valor de pitch comandado por el autopiloto (modo 0 ó 1) al CAS de pitch, la señal pasa por un bloque “Rate Limiter” y un “Saturation” para asegurar que el autopiloto comande valores de pitch adecuados para el CAS en todo momento. El bloque “Rate Limiter” se asegura de que la derivada de la señal entrante no salga de una franja preestablecida de valores en todo momento, y si lo hace obliga a que tome el valor umbral. El bloque “saturation” limita la señal entrante con unos valores preestablecidos.

Se explicará más detalladamente cómo se activa/desactiva el autopiloto de la aeronave y como se comanda una altitud a través de la interfaz gráfica en el apartado 5.2.1.

3.2.2.3 Yaw Control

El control de yaw no se ha desarrollado en este simulador ya que, como se explicó en el apartado 3.2.1, la aeronave X no dispone de superficies de control específicamente diseñadas para controlar este grado de libertad, Es decir, el ángulo de guiñada sólo puede controlarse mediante los alerones.

3.2.2.4 Throttle Control

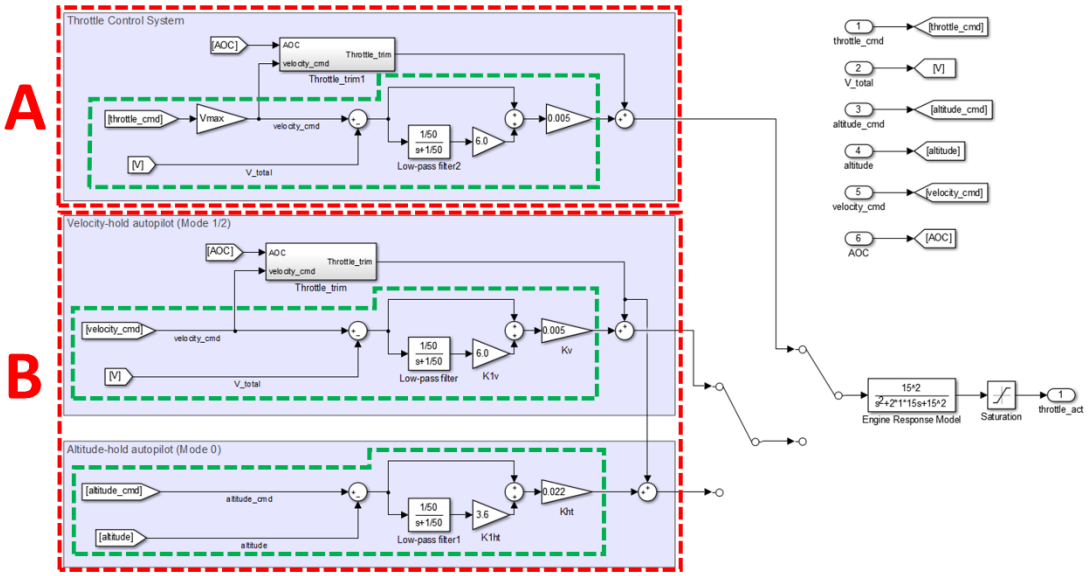

Figura 29: Estructura del subsistema "Throttle Control"

En la Figura 29 se puede observar la estructura general del subsistema de control de throttle. Se puede dividir la estructura en dos partes.

La **parte A** corresponde al sistema de control de throttle desde los comandos. Este sistema se utiliza para obtener una respuesta adecuada de la aeronave al control de throttle desde el joystick. El sistema se basa en asignar a cada posición de la palanca de throttle una velocidad comandada, definida por la variable “Vmax” en un bloque de ganancia. Esta variable se define en el script de inicialización “startVars”. De esta manera es posible comandar una velocidad desde el joystick. La respuesta del throttle va a ser en todo momento aquella que consiga que la aeronave alcance la velocidad comandada. Por ejemplo, cuando la aeronave adquiera un ángulo de subida sin mover la palanca de throttle, la respuesta del throttle va a aumentar para compensar el efecto de resistencia de la componente de gravedad que ha aparecido.

Esta estructura nuevamente está basada en la estructura básica de control que se mencionó en el apartado 3.2.2.1 (señalada en verde). Además, se suma un término que corresponde al “Throttle trim” y se puede ver a continuación (Figura 30).

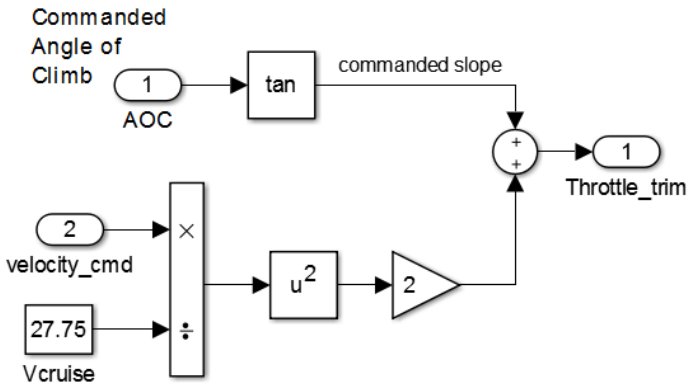

Figura 30: Estructura del subsistema "Throttle Trim"

El trim de throttle se incorpora para compensar los efectos que generan en la resistencia la velocidad de la aeronave y la pendiente comandada.

La **zona B** corresponde, junto con la zona B del control de pitch, al autopiloto de la aeronave (Figura 29). Tal y como se explicó en el apartado 3.2.2.2, el autopiloto se puede activar en dos modos de vuelo (0 ó 1). Los autopilotos tienen la misma estructura que el sistema de control de throttle desde los comandos (zona A), siendo la única diferencia que la altitud y velocidad comandadas no son señales provenientes del joystick, sino que son constantes que se pueden actualizar mediante la interfaz gráfica.

Contrariamente a los autopilotos ubicados en el control de pitch, estos autopilotos no alimentan al sistema de control de throttle predeterminado (zona A) sino que dan como output la propia respuesta del motor, no sin antes aplicar dos últimos bloques para terminar el modelado: “Engine response model” y “Saturation”. Este último limita el rango de empuje que el motor es capaz de proveer. El bloque “Engine response model” modela el retardo en la respuesta del motor debido a la inercia de los componentes rotatorios del mismo. Se ha modelado este bloque de manera similar a los modelos de respuesta de los actuadores de pitch y de roll, una función de transferencia de segundo orden. La respuesta de este bloque a un input escalón de magnitud unidad se presenta a continuación.

$$\frac{\omega_n^{\,2}}{s^2 + 2\cdot\zeta\cdot\omega_n + \omega_n^{\,2}} \qquad \omega_n = 15 \text{ rad/s} \qquad \zeta = 1 \tag{10}$$

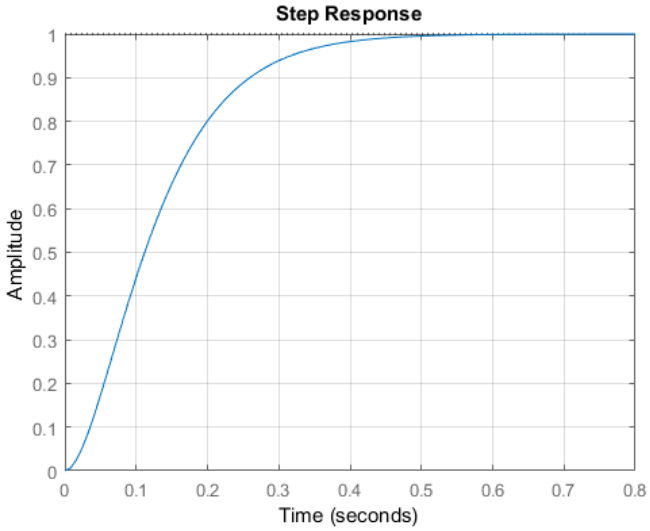

Figura 31: Diagrama de respuesta a escalón del bloque "Engine response model"

3.2.3 Modelo dinámico de la aeronave

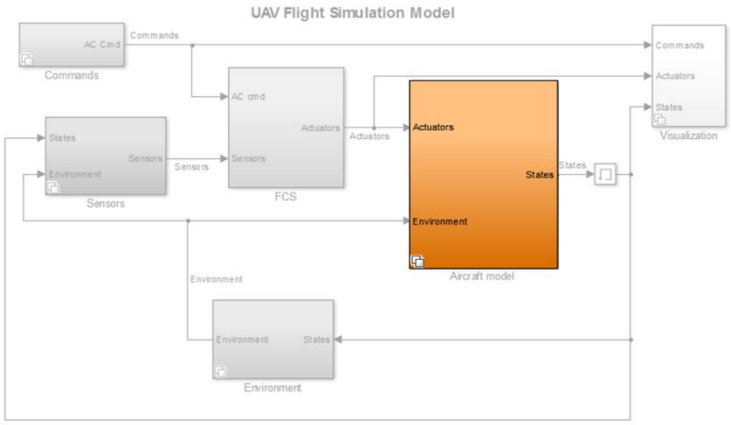

Figura 32: Ubicación del subsistema "Aircraft Model" en el simulador

Construir un modelo dinámico es un proceso fundamental. En el proceso de diseño se suelen utilizar modelos físicos para obtener información aerodinámica en ensayos en túnel de viento y posteriormente incorporar los datos experimentales en un modelo matemático. La fase de obtención de las características aerodinámicas también puede hacerse mediante simulaciones de Dinámica de Fluidos Computacional (CFD en inglés). El proceso de modelado suele ser iterativo.

De hecho, debido al alto coste de construir un avión y realizar ensayos de vuelo, la importancia de los modelos matemáticos va mucho más allá del diseño. El modelo matemático se utiliza, junto a simulaciones computacionales, para evaluar las actuaciones de las aeronaves prototipo, ayudando de esta manera a mejorar el diseño. También se pueden utilizar como núcleo de simuladores de entrenamiento, para reproducir las condiciones de vuelo en casos de accidente,

o para estudiar los efectos de modificaciones en el diseño. En consecuencia, los modelos matemáticos se utilizan en todas las fases del diseño de aeronaves.

En un primer nivel, se puede observar que en el subsistema "Aircraft Model" se ha utilizado nuevamente la función de subsistema variante. Esta vez existe la posibilidad de activar el modelo no lineal de la aeronave (por defecto) o el modelo lineal, que se explicarán en detalle en los siguientes apartados.

3.2.3.1 Subsistema "Nonlinear Model"

A continuación se expondrá el desarrollo que se ha seguido para construir el modelo dinámico no lineal de la plataforma [1][18].

A) Definición de sistemas de coordenadas y variables angulares

Las fuerzas y momentos aerodinámicos en una aeronave se producen por su movimiento relativo con respecto al aire, y dependen de la orientación de esta con respecto al flujo de aire. Si el flujo es uniforme, las fuerzas y los momentos aerodinámicos no cambian al rotar la aeronave alrededor de su vector velocidad. Por tanto sólo serían necesarios dos ángulos (con respecto al viento relativo) para que las fuerzas y momentos aerodinámicos queden determinados. Los ángulos que se utilizan son el *ángulo de ataque* (α) y el *ángulo de resbalamiento* (β). Se conocen como los *ángulos aerodinámicos* y a continuación serán definidos formalmente para el caso de una aeronave. Nótese que las fuerzas y momentos aerodinámicos también dependen de las velocidades angulares, pero de momento sólo se centrará la atención en la orientación.

La Figura 33 muestra la aeronave X con con el viento relativo en su lado derecho. Se han representado tres sistemas de coordenadas a derechas y con un origen común en el centro de masas del avión, además de los ángulos α y β.

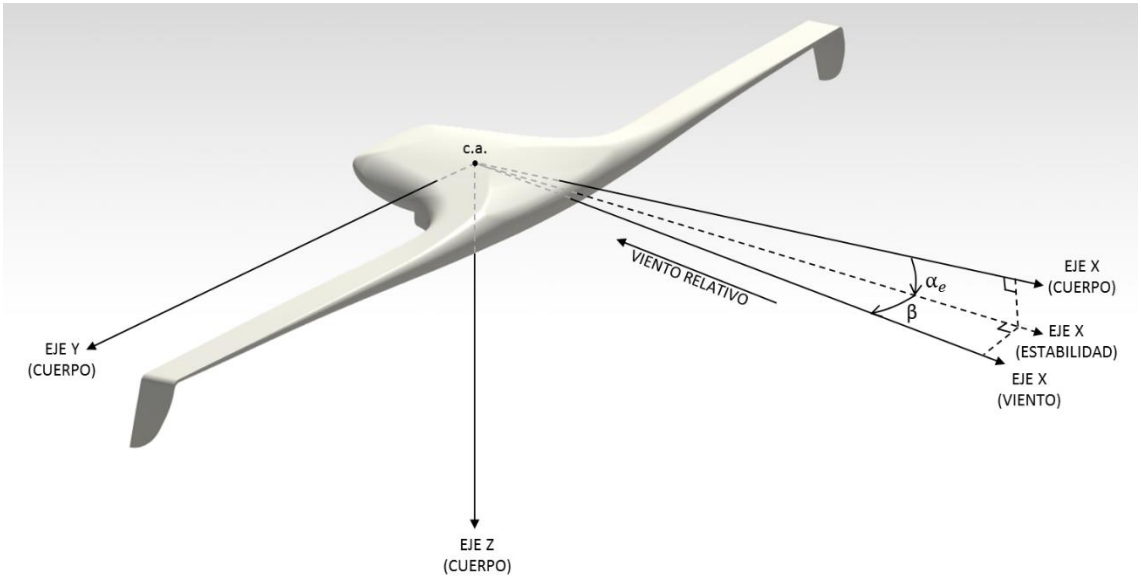

Figura 33: Sistemas de coordenadas

El **sistema de ejes cuerpo** tiene el eje x paralelo a la línea de referencia del fuselaje y su eje z está contenido en el plano de simetría de la aeronave. Se denotará por α_{frl} al ángulo de ataque medido desde la línea de referencia del fuselaje hasta la proyección del viento relativo sobre el plano x-z. Es positivo cuando el viento relativo entra por la parte inferior de la aeronave. El ángulo de resbalamiento se mide desde dicha proyección hasta el vector velocidad relativa. Será positivo cuando el viento relativo entre por la parte derecha de la aeronave.

El ángulo de ataque también se puede denotar por α_0 cuando se mida referenciado a la línea de sustentación nula de la aeronave. A lo largo de este desarrollo se usará la denominación α para referirse a α_{frl}. Para una aeronave en condición de vuelo estacionaria, el ángulo de ataque de equilibrio se denotará por α_e, mientras que el ángulo de resbalamiento de equilibrio suele ser cero.

En la Figura 33 se define también la orientación del **sistema de ejes estabilidad**, que se usa para analizar el efecto de las perturbaciones desde el estado de vuelo estacionario. Como se observa en la figura, se obtiene a partir del sistema de ejes cuerpo realizando una rotación a izquierdas, el ángulo α_e, alrededor del eje y (cuerpo).

El **sistema de ejes viento** se obtiene a partir del sistema de ejes estabilidad realizando una rotación alrededor del eje z (cuerpo) que consiga alinear el eje x (viento) con el vector de viento relativo.

En el pasado (Pope, 1954) se ha utilizado un sistema de ejes viento a izquierdas, alineado hacia atrás, arriba e izquierda relativos al avión para referenciar datos medidos en túnel de viento. Las componentes *sustentación*, L, *resistencia*, D y *fuerza lateral*, C, fueron definidas en estos ejes como las componentes de fuerza aerodinámica en el sentido positivo de los mismos.

Durante el desarrollo, los subíndices se referirán a los ejes cuerpo como b (body), a los ejes viento como w (wind) y a los ejes estabilidad como s (stability). Utilizando las reglas para hallar matrices de rotación, las matrices de rotación de ejes cuerpo a estabilidad y de ejes estabilidad a viento son las siguientes.

$$C_{s/b} = \begin{bmatrix} \cos\alpha_e & 0 & \sin\alpha_e \\ 0 & 1 & 0 \\ -\sin\alpha_e & 0 & \cos\alpha_e \end{bmatrix} \tag{11}$$

$$C_{w/s} = \begin{bmatrix} \cos\beta & \sin\beta & 0 \\ -\sin\beta & \cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{12}$$

Por lo que la matriz de rotación de ejes cuerpo a ejes viento quedaría

$$C_{b/w} = \begin{bmatrix} \cos\alpha_e \cdot \cos\beta & \sin\beta & \sin\alpha_e \cdot \cos\beta \\ -\cos\alpha_e \cdot \sin\beta & \cos\beta & -\sin\alpha_e \cdot \sin\beta \\ -\sin\alpha_e & 0 & \cos\alpha_e \end{bmatrix} \tag{13}$$

B) Fuerzas gravitatorias

Para calcular las fuerzas ocasionadas por el campo gravitatorio terrestre es necesario definir otro sistema de referencia. Este corresponde al sistema de ejes horizonte local.(subíndice n). Este sistema tiene origen común con los otros tres sistemas definidos en el apartado anterior (cuerpo, estabilidad y viento), es decir, el centro de masas de la aeronave. Su eje z está dirigido hacia el centro de la Tierra, por tanto los ejes x e y están contenidos en un plano horizontal paralelo al plano tangente a la superficie terrestre en el punto subavión, y apuntan a una dirección fija de dicho plano. El eje x suele apuntar al Norte terrestre, y por tanto el eje y suele apuntar al Este.

$$C_{n/b} = \begin{bmatrix} \cos\theta\cos\psi & \cos\theta\sin\psi & -\sin\theta \\ \sin\phi\sin\theta\cos\psi - \cos\theta\sin\psi & \sin\phi\sin\theta\sin\psi + \cos\theta\cos\psi & \sin\phi\cos\theta \\ \cos\phi\sin\theta\cos\psi + \sin\theta\sin\psi & \cos\phi\sin\theta\sin\psi - \sin\theta\cos\psi & \cos\phi\cos\theta \end{bmatrix} \quad (14)$$

Donde los ángulos θ, ϕ y ψ corresponden a los ángulos de cabeceo (pitch, θ), balance (roll, ϕ) y guiñada (yaw, ψ), denominados en conjunto *ángulos de Euler*. Estos ángulos surgen de aplicar el método de las rotaciones de Euler para obtener la orientación entre dos sistemas de referencia con origen común. Este método consiste en definir tres rotaciones sucesivas finitas dadas en un orden especificado. En primer lugar se rota el sistema de ejes horizonte local alrededor de su eje z un ángulo ψ y se obtiene un primer sistema intermedio. Después se rota este sistema intermedio alrededor de su eje y un ángulo θ y se obtiene un segundo sistema intermedio. Por último se rota este segundo sistema intermedio alrededor de su eje x un ángulo ϕ para obtener el sistema final, coincidente con el sistema de ejes cuerpo. Este orden de aplicación de rotaciones (ejes z,y,x)suele denominarse *Convención zxy*, *Convención 321*, o *Convención de Tait-Bryan*, y es universalmente utilizado para orientar aeronaves y vehículos espaciales [19].

La fuerza gravitatoria puede expresarse fácilmente en el sistema de ejes horizonte local mediante

$$\mathbf{F}_g^n = \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} \quad (15)$$

Donde g es la aceleración de la gravedad, que se suele considerar constante para el rango de vuelo de las aeronaves, de valor $g = g_0 = 9.81\ m/s^2$.

Para proyectar la fuerza gravitatoria en el sistema de ejes cuerpo, se utiliza la matriz de rotación $C_{n/b}$ (14):

$$\mathbf{F}_g^b = C_{n/b}\mathbf{F}_g^n = \begin{pmatrix} -mg\sin\theta \\ mg\cos\theta\sin\phi \\ mg\cos\theta\cos\phi \end{pmatrix} \quad (16)$$

El sistema de ejes Tierra, tiene todos los ejes paralelos a los del sistema de ejes horizonte local, pero su origen corresponde al punto subavión. Este sistema de ejes se utilizará más delante.

C) Coeficientes de fuerzas y momentos

Las fuerzas y momentos que actúan en la aeronave completa se definen en función de coeficientes aerodinámicos adimensionales. Los coeficientes dependen de los dos ángulos aerodinámicos, además de los números de Mach y de Reynolds. Además, las aeronaves tiene estructuras flexibles, cuya forma cambia con la influencia de presiones dinámicas altas, lo que provoca a su vez cambios en los coeficientes aerodinámicos. Si se especifican el Mach y la altitud, junto con una temperatura y un modelo de densidad de la atmósfera, tanto el número de Reynolds como la presión dinámica se pueden determinar. Por tanto, los coeficientes aerodinámicos se suelen especificar como funciones de los ángulos aerodinámicos, el número de Mach y la altitud (en la atmósfera estándar). Adicionalmente, las deflexiones de las superficies de control y efectos del sistema de propulsión pueden causar cambios en los coeficientes. La deflexión de una superficie de control (δ_s) puede cambiar la curvatura del ala, lo que afecta a su vez a la sustentación, la resistencia y el momento aerodinámicos. En consecuencia, se escribe la dependencia de un coeficiente aerodinámico como

$$C_{(\,)} = C_{(\,)}(\alpha, \beta, M, h, \delta_S, T_C) \tag{17}$$

Donde T_C es un *coeficiente de empuje*. Otros factores que cambian los coeficientes son los cambios de configuración de la aeronave (por ejemplo tren de aterrizaje, tanques externos, etc.) y efectos por proximidad del suelo. A continuación se presenta la nomenclatura utilizada para fuerzas, momentos y velocidades de la aeronave.

Fuerzas aerodinámicas:

$$\mathbf{F}_A^w = \begin{bmatrix} -D \\ -C \\ -L \end{bmatrix} = C_{w/b} \begin{bmatrix} X_A \\ Y_A \\ Z_A \end{bmatrix} = C_{w/b}\mathbf{F}_A^b \tag{18}$$

Coeficientes de fuerza adimensionales:

	x	y	z
Viento:	C_D	C_C	C_L
Estabilidad:	$-$	C_Y	C_L
Cuerpo:	C_X	C_Y	$C_Z(-C_N)$

(19)

Momentos aerodinámicos:

$$\mathbf{M}_A^b = \begin{bmatrix} l \\ m \\ n \end{bmatrix} \quad \mathbf{M}_A^s = \begin{bmatrix} l_s \\ m \\ n_s \end{bmatrix} \quad \mathbf{M}_A^w = \begin{bmatrix} l_w \\ m_w \\ n_w \end{bmatrix} \tag{20}$$

Coeficientes de momento adimensionales:

$$C_l \ , C_m \ , C_n$$

(misma notación en todos los sistemas)

Fuerza y momento propulsivos:

$$\mathbf{F}_T^b = \begin{bmatrix} X_T \\ Y_T \\ Z_T \end{bmatrix} \quad \mathbf{M}_T^b = \begin{bmatrix} m_{x,T} \\ m_{y,T} \\ m_{z,T} \end{bmatrix} \tag{21}$$

Componentes de velocidad relativa:

$$\mathbf{v}_{rel}^b = \begin{bmatrix} U' \\ V' \\ W' \end{bmatrix} = C_{b/w}\mathbf{v}_{rel}^w = C_{b/w}\begin{bmatrix} V_T \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} V_T \cos\alpha\cos\beta \\ V_T \sin\beta \\ V_T \sin\alpha\cos\beta \end{bmatrix} \tag{22}$$

Componentes de velocidad absoluta:

$$\mathbf{v}_{CM/e}^b = \begin{bmatrix} U \\ V \\ W \end{bmatrix} \tag{23}$$

Componentes de velocidad angular (subíndice r denota cualquier sistema de referencia):

$$\omega_{b/r}^b = \begin{bmatrix} P \\ Q \\ R \end{bmatrix} \quad \omega_{b/r}^s = \begin{bmatrix} P_s \\ Q \\ R_s \end{bmatrix} \quad \omega_{b/r}^w = \begin{bmatrix} P_w \\ Q_w \\ R_w \end{bmatrix} (R_w = R_s) \tag{24}$$

Deflexiones de las superficies de control:

Elevador: δ_e Alerones: δ_a Timón de dirección: δ_r Posición de throttle: δ_t

Vector de control: $\bar{U} = \begin{bmatrix} \delta_t & \delta_e & \delta_a & \delta_r & \ldots \end{bmatrix}^T$

Ángulos aerodinámicos:

$$\tan\alpha = \frac{W'}{U}; \quad \sin\beta = \frac{V'}{V_T}; \quad V_T = \left|\mathbf{v}_{rel}\right| \tag{25}$$

Utilizando como referencia el sistema de ejes viento, los coeficientes quedan

$$\text{resistencia, } D = \bar{q} \cdot S \cdot C_D \tag{26}$$

$$\text{sustentación, } L = \bar{q} \cdot S \cdot C_L \tag{27}$$

$$\text{fuerza lateral, } \mathrm{C} = \bar{q} \cdot S \cdot C_C \tag{28}$$

$$\text{momento de balance, } l_w = \bar{q} \cdot S \cdot b \cdot C_l \tag{29}$$

$$\text{momento de cabeceo, } m_w = \bar{q} \cdot S \cdot \bar{c} \cdot C_m \tag{30}$$

$$\text{momento de guiñada, } n_w = \bar{q} \cdot S \cdot b \cdot C_n \tag{31}$$

Se han realizado definiciones equivalentes referenciadas en ejes cuerpo y estabilidad, utilizando los símbolos correspondientes a cada caso.

En la mayoría de los casos los *coeficientes longitudinales* (sustentación, resistencia y momento de cabeceo) dependen principalmente del ángulo de ataque, mientras que los *coeficientes lateral-direccionales* (momento de balance, momento de guiñada y fuerza lateral) tienen una dependencia de orden similar con ambos ángulos aerodinámicos.

La ecuación (17) implica una compleja dependencia que tendría que ser modelada como una tabla de búsqueda en un ordenador. La gran mayoría de los aviones tienen envolventes de vuelo restringidas a ángulos de ataque pequeños y/o números de Mach bajos. Para estas aeronaves, la dependencia funcional será más simple y algunos coeficientes podrán ser descompuestos a una suma de términos más simples, algunos de ellos lineales.

Los coeficientes que se consideran hasta este punto son *coeficientes estáticos*, es decir, se pueden obtener a partir de medidas de un modelo estacionario en un túnel de viento (más adelante se considerarán otros métodos). También es necesario modelar los efectos aerodinámicos presentes cuando una aeronave está maniobrando. Como norma general, esto requiere un modelo de ecuaciones diferenciales de la fuerza y momento aerodinámicos. Para determinar si este nivel de complejidad es necesario, se procederá a examinar el vuelo en maniobras con más detalle, en dos categorías. Primeramente se considerarán maniobras lo suficientemente lentas como para que el flujo de aire alrededor del avión sea capaz de adaptarse a la propia maniobra, y que la velocidad de translación de los puntos de la aeronave causada por la maniobra induzca cambios en los ángulos aerodinámicos locales que permanezcan en el régimen lineal. Las fuerzas o momentos aerodinámicos se pueden modelar entonces como linealmente proporcionales a la velocidad angular que los produjo. El proceso de linealización normalmente se asocia a establecer una derivada parcial, y en este caso el coeficiente de proporcionalidad se denomina *derivada aerodinámica*. Las derivadas aerodinámicas se describirán en el siguiente apartado.

En la segunda categoría se encuentran las maniobras en las que una aeronave puede cambiar su orientación significantemente en un intervalo de tiempo comparable al correspondiente al que necesita el flujo de aire que rodea al avión para reajustarse. Estos *efectos aerodinámicos transitorios* se traducen en coeficientes aerodinámicos variables con el tiempo y modelos matemáticos mucho más complejos. Por ejemplo, cuando se introduce una velocidad angular de cabeceo alta en un avión muy maniobrable, y el ángulo de ataque alcanza un valor cercano al de entrada en pérdida, la sustentación generada por el ala puede exceder brevemente durante un periodo corto de tiempo la que predice la curva de sustentación estática. Esta *sustentación dinámica* ocurre porque la separación de flujo tarda un tiempo finito en progresar desde el borde de salida hasta el borde de ataque del ala. El efecto se puede modelar haciendo que el coeficiente de sustentación satisfaga una ecuación diferencial de primer orden dependiente de la velocidad angular del ángulo de ataque, $\dot{\alpha}$ (Goman and Khrabrov, 1994).

D) Las derivadas aerodinámicas

Las derivadas aerodinámicas se pueden dividir en dos categorías. Por un lado, cuando el sistema de ejes cuerpo tiene un vector de velocidad angular constante, todos los puntos del avión tienen diferentes velocidades de translación en un sistema geográfico y, tomando las componentes en ejes cuerpo, los ángulos aerodinámicos se pueden calcular en cualquier instante. Por ejemplo, una velocidad angular de balance P generaría componentes de velocidad de translación $\pm Pb/2$ en las puntas de las alas. Cuando $P > 0$ el ángulo de ataque se reduciría aproximadamente $Pb/(2V_T)$ en la punta de la semiala izquierda, y se incrementaría el mismo valor en la punta de la semiala derecha. Esto produciría una variación antisimétrica de la sustentación a lo largo de la envergadura y, asumiendo que el ala no haya entrado en pérdida en la mayor parte de la envergadura, produciría un momento de balance negativo. Debido a que el momento se opone a la velocidad angular P, el coeficiente que relaciona el momento de balance con la velocidad angular de balance se denomina *derivada de amortiguamiento*.

A la cantidad $Pb/(2V_T)$ se le asigna el símbolo $\hat{p}$ y se utiliza como una velocidad angular de balance adimensional. Con una velocidad de balance constante y el centro de masas del avión moviéndose en una línea recta, las puntas de ala se mueven a lo largo de una trayectoria helicoidal, y $Pb/(2V_T)$ corresponde al *ángulo de la hélice*. El modelo matemático para la fuerza de amortiguamiento adimensional, o momento ΔC, es de la forma:

$$\Delta C_{()} = C_{()}(\alpha, \beta, M, h, \delta_s, T_c) \times \frac{k}{2V_T} \times \text{velocidad angular} \tag{32}$$

Donde la constante k representa la envergadura (para velocidades angulares de balance y guiñada) o la cuerda media aerodinámica (para la velocidad angular de cabeceo). El coeficiente $C_{()}$ es una de las siguientes derivadas de p, q ó r.

$$\begin{matrix} C_{l_p} & C_{m_q} & C_{n_r} \\ & C_{l_r} \quad C_{n_p} & \\ C_{L_q} & C_{Y_p} & C_{Y_r} \end{matrix} \tag{33}$$

Estas derivadas relacionan los incrementos en los momentos o fuerzas con las velocidades angulares de balance, cabeceo y guiñada. Las fuerzas y momentos adimensionales se convierten a fuerzas y momentos de la manera que indican las ecuaciones (26)-(31). Algunas posibles derivadas se han omitido, por ejemplo, el efecto de la velocidad angular de cabeceo en la resistencia es normalmente despreciable. Las derivadas de momentos son el núcleo de los efectos de amortiguamiento presentes en los modos naturales de la aeronave.

La segunda categoría de derivadas aerodinámicas son las *derivadas de aceleración*. Cuando la aeronave tiene aceleración de translación, los ángulos aerodinámicos tienen derivadas no nulas, que se pueden obtener diferenciando las ecuaciones (25). De esta manera, quedaría

$$\dot{\alpha} = \frac{U'\dot{W}' - W'\dot{U}'}{(U')^2 + (W')^2} \tag{34}$$

$$\dot{\beta} = \frac{\dot{V}'V_T - V'\dot{V}_T}{V_T\left[\left(U'\right)^2 + \left(W'\right)^2\right]^{1/2}} \tag{35}$$

$$\dot{V}_T = \frac{U'\dot{U}' + V'\dot{V}' + W'\dot{W}'}{V_T} \tag{36}$$

El mayor efecto de los ángulos aerodinámicos cambiantes es que, a medida que el flujo de aire alrededor de las alas y el fuselaje cambia, hay un pequeño retraso dependiente de la velocidad del aire hasta que los cambios se perciben en la cola del avión. Una aproximación de primer orden para modelar estos efectos consiste en hacer los incrementos de fuerza y momento resultantes directamente dependientes de las velocidades angulares de los ángulos aerodinámicos. Por tanto, frecuentemente se usan las siguientes derivadas de aceleración.

$$C_{L_{\dot{\alpha}}} \quad C_{m_{\dot{\alpha}}}$$

Estas derivadas se utilizan en una ecuación de la misma forma que la ecuación (32). Las derivadas de la velocidad angular del ángulo de resbalamiento ($\dot{\beta}$) no son tan usadas.

E) Medida y estimación de los coeficientes aerodinámicos

Los coeficientes aerodinámicos estáticos se pueden medir en un túnel de viento utilizando un modelo a escala de la aeronave montado en un apoyo rígido al que se aplican extensómetros. Un túnel de viento más antiguo puede utilizar un método de balance del modelo, en lugar de usar extensómetros. El montado rígido del modelo permite calcular los *coeficientes en desequilibrio*, esto es, se pueden medir los momentos aerodinámicos parásitos mientras que se cambian los ángulos aerodinámicos o se mueven las superficies de control.

Algunos túneles de viento especialmente adaptados permiten aplicar al modelo un movimiento oscilatorio (Queijo, 1971) para medir las derivadas de amortiguamiento y las de aceleración. Desafortunadamente, como cabría esperar, los resultados dependen de la frecuencia aplicada. Se han formulado criterios empíricos para determinar los límites de frecuencia por debajo de los cuales se puede asumir una condición de flujo cuasi-estacionaria (Duncan, 1952).

El segundo método existente para medir coeficientes aerodinámicos es mediante pruebas en vuelo. En este caso los *coeficientes en equilibrio* se miden utilizando las superficies de control para inducir perturbaciones desde el estado estacionario de equilibrio (Maine and Iliffe, 1980). Los resultados típicos son curvas de un coeficiente graficadas en función del Mach, con la altitud como parámetro, para una determinada masa de la aeronave y posición de su centro de masas. La dependencia con la altitud se debe a la variación del ángulo de ataque con la altitud para un Mach dado, a los efectos aeroelásticos cambiantes con la presión dinámica, y a efectos del número de Reynolds. Para convertir a coeficientes en desequilibrio, que son funciones de los ángulos aerodinámicos, el Mach y la altitud, el ángulo de ataque de equilibrio también debe ser medido de la misma manera. Los coeficientes en desequilibrio son requeridos para construir un modelo de aeronave destinado a funcionar en un rango grande de condiciones de vuelo. En cambio, los coeficientes en equilibrio se utilizan para construir modelos de pequeñas

perturbaciones para diseño de sistemas de control (objetivo principal del desarrollo de esta plataforma de simulación) o estudios de cualidades de vuelo.

Otra manera de determinar los coeficientes aerodinámicos consiste en utilizar códigos de computación CFD (Computational Fluid Dynamics) o incluir una combinación de datos empíricos y teoría en programas de software como "Datcom" (Hoak et al., 1970). Con los datos de entrada es necesario incluir un modelo geométrico de la aeronave. Existen también fórmulas simples basadas en hipótesis de linealidad que se pueden utilizar para estimar las derivadas aerodinámicas.

F) Implementación en simulador

Se procede a continuación a mostrar cómo se ha implementado el modelo de cálculo de fuerzas y momentos descrito en los apartados previos en el simulador no lineal de la plataforma.

El bloque "Nonlinear Model" se muestra a continuación (Figura 34).

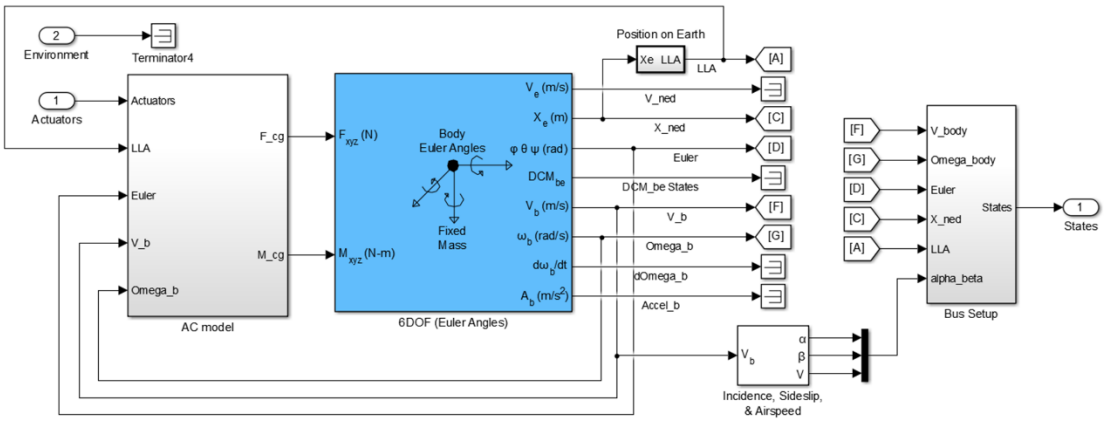

Figura 34: estructura del subsistema "Nonlinear Model"

El modelo dinámico tiene dos bloques principales: (1) El bloque "AC model" (aircraft model) contiene el modelo de cálculo de fuerzas y momentos de la aeronave. Toma como inputs los movimientos de las superficies de control (actuators) y los propios estados de la aeronave, que vienen realimentados de la salida del siguiente bloque. (2) El bloque "6DOF (Euler Angles)" contiene el modelo de cálculo de estados de la aeronave.

F.1) Cálculo de fuerzas y momentos

El bloque "AC model" se muestra en la Figura 35.

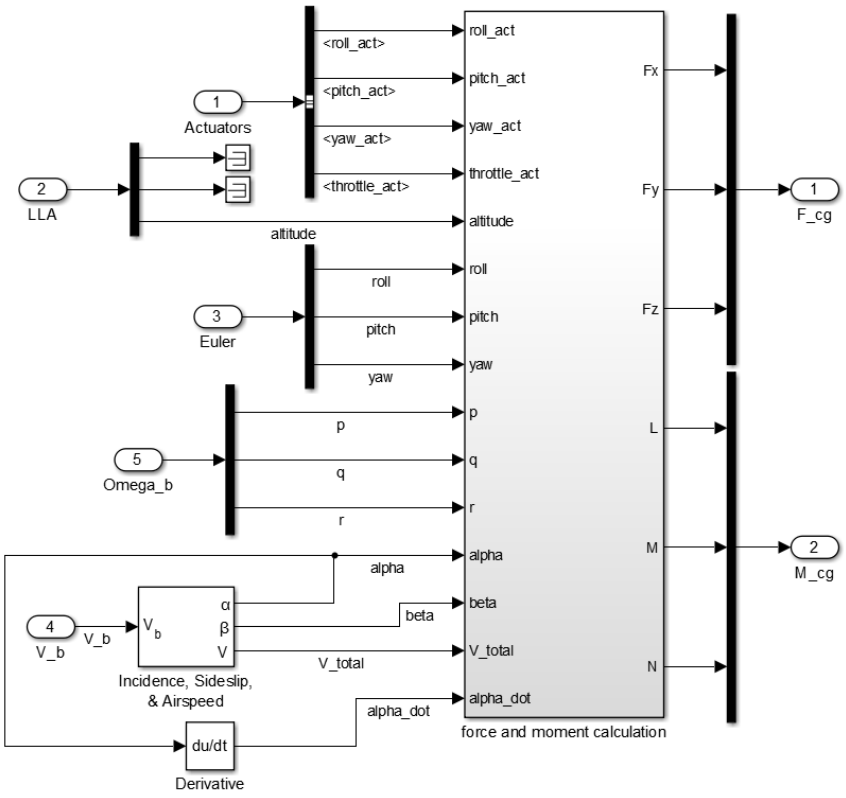

Figura 35: Estructura del bloque "AC Model"

La parte izquierda del diagrama de la Figura 35 está formada por la estructura de adaptación de los comandos y estados necesarios para alimentar el bloque de cálculo de fuerzas y momentos. Se puede apreciar que se ha utilizado un bloque denominado “Incidence, Sideslip & Airspeed”. Este bloque pertenece al “Aerospace Blockset” incorporado en Simulink® [20]. Calcula los ángulos aerodinámicos y la velocidad total de la siguiente manera.

$$\alpha = \arctan\left(\frac{w}{u}\right) \tag{37}$$

$$\beta = \arcsin\left(\frac{v}{V}\right) \tag{38}$$

$$V = \sqrt{u^2 + v^2 + w^2} \tag{39}$$

A continuación se muestra el bloque “force and moment calculation” (Figura 36).

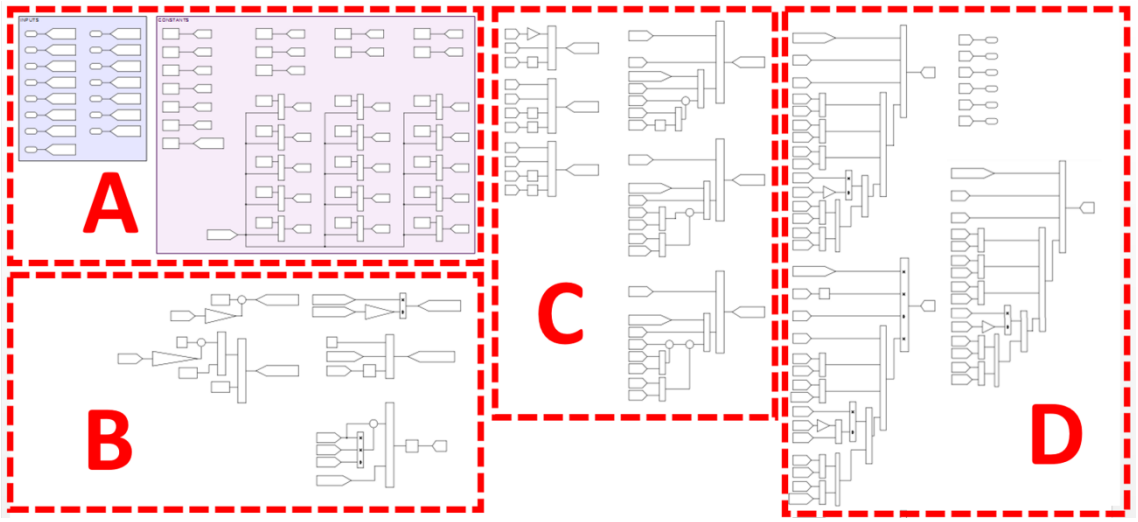

Figura 36: Estructura del bloque "Force and moment calculation"

Dado el tamaño de la estructura se ha dividido ésta en cuatro secciones para su correcta visualización. La sección A se muestra a continuación (Figura 37).

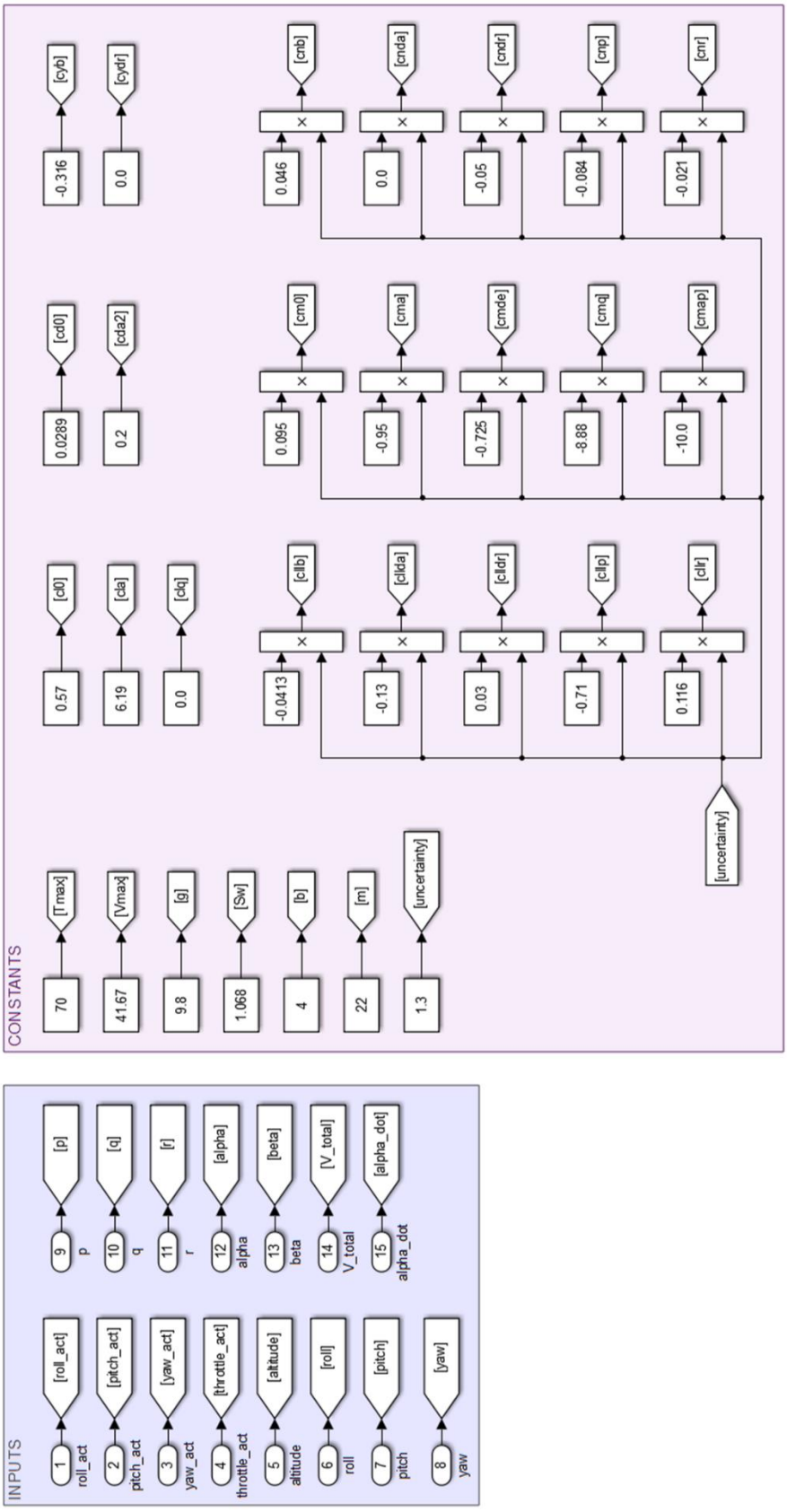

Figura 37: Sección A del bloque "Force and moment calculation"

La **sección A** (Figura 37) envía las señales de entrada a bloques “GoTo” para después ser utilizadas en las siguientes secciones. Adicionalmente, se definen las propiedades geométricas, másicas y propulsivas, así como las derivadas aerodinámicas (todas definidas en el capítulo 2) de la aeronave. También se ha incorporado una constante denominada “uncertainty” que puede modificarse y corresponde a un factor que multiplica a todas las derivadas aerodinámicas. Se puede utilizar para realizar un estudio del comportamiento de la aeronave suponiendo incertidumbre en las derivadas aerodinámicas.

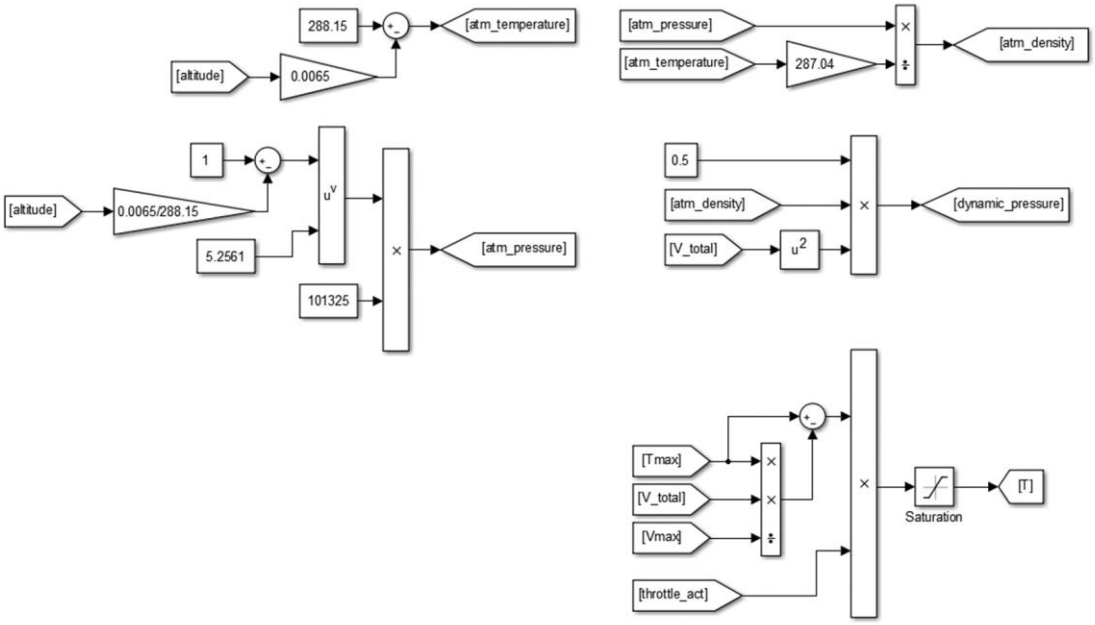

Figura 38: Sección B del bloque "Force and moment calculation"

La **sección B** (Figura 38) calcula la densidad y la presión dinámicas a partir de la altitud de la aeronave. Se utiliza el modelo de atmósfera estándar internacional. Además se calcula el empuje del motor a partir de un modelo muy simple basado en un empuje lineal con la velocidad de valor T_{max} a velocidad nula y de valor nulo a V_{max}. Las constantes T_{max} y V_{max} se definieron en la sección A.

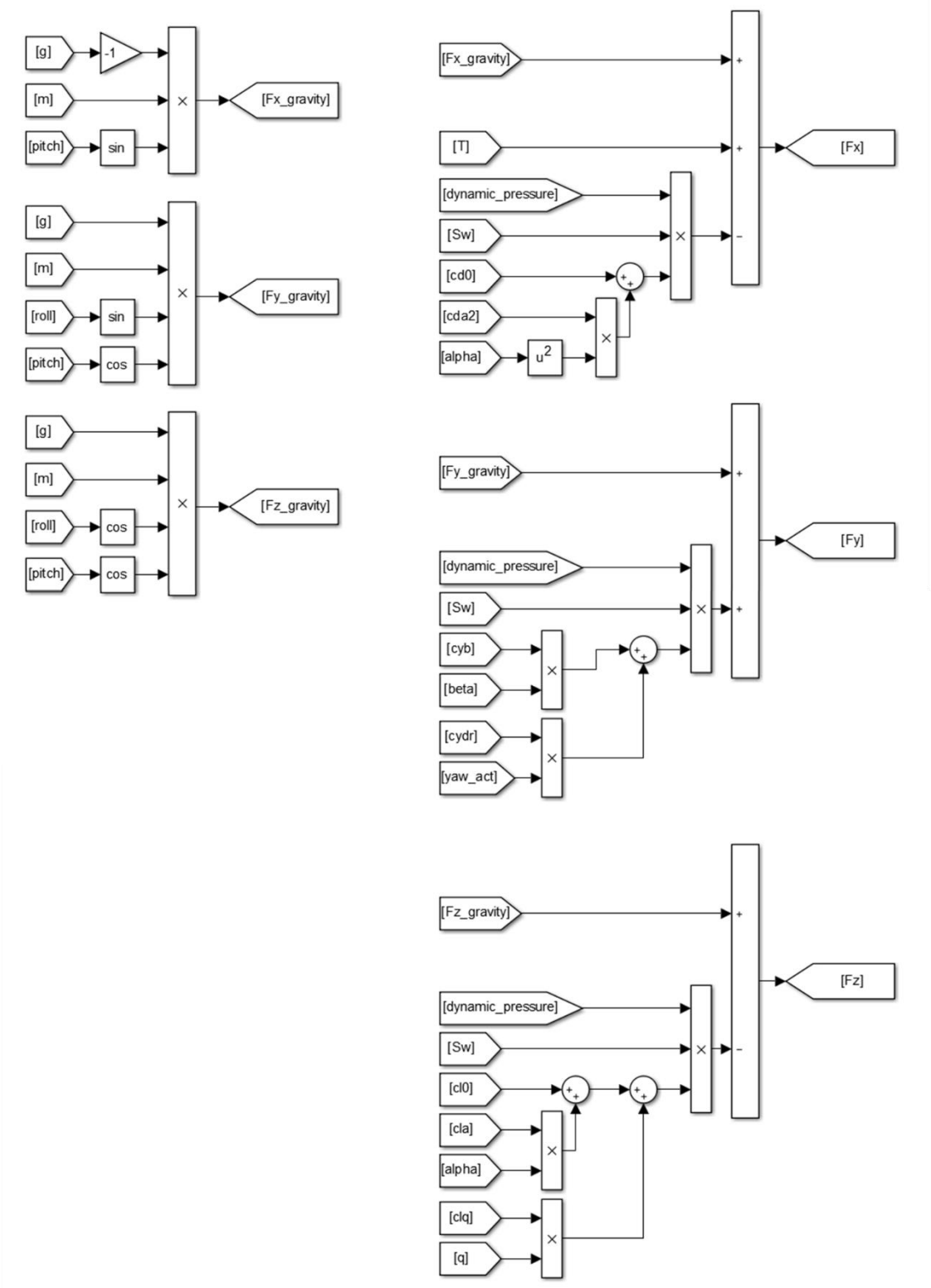

Figura 39: Sección C del bloque "Force and moment calculation"

La **sección C** contiene el cálculo de fuerzas gravitatorias y aerodinámicas y propulsivas.

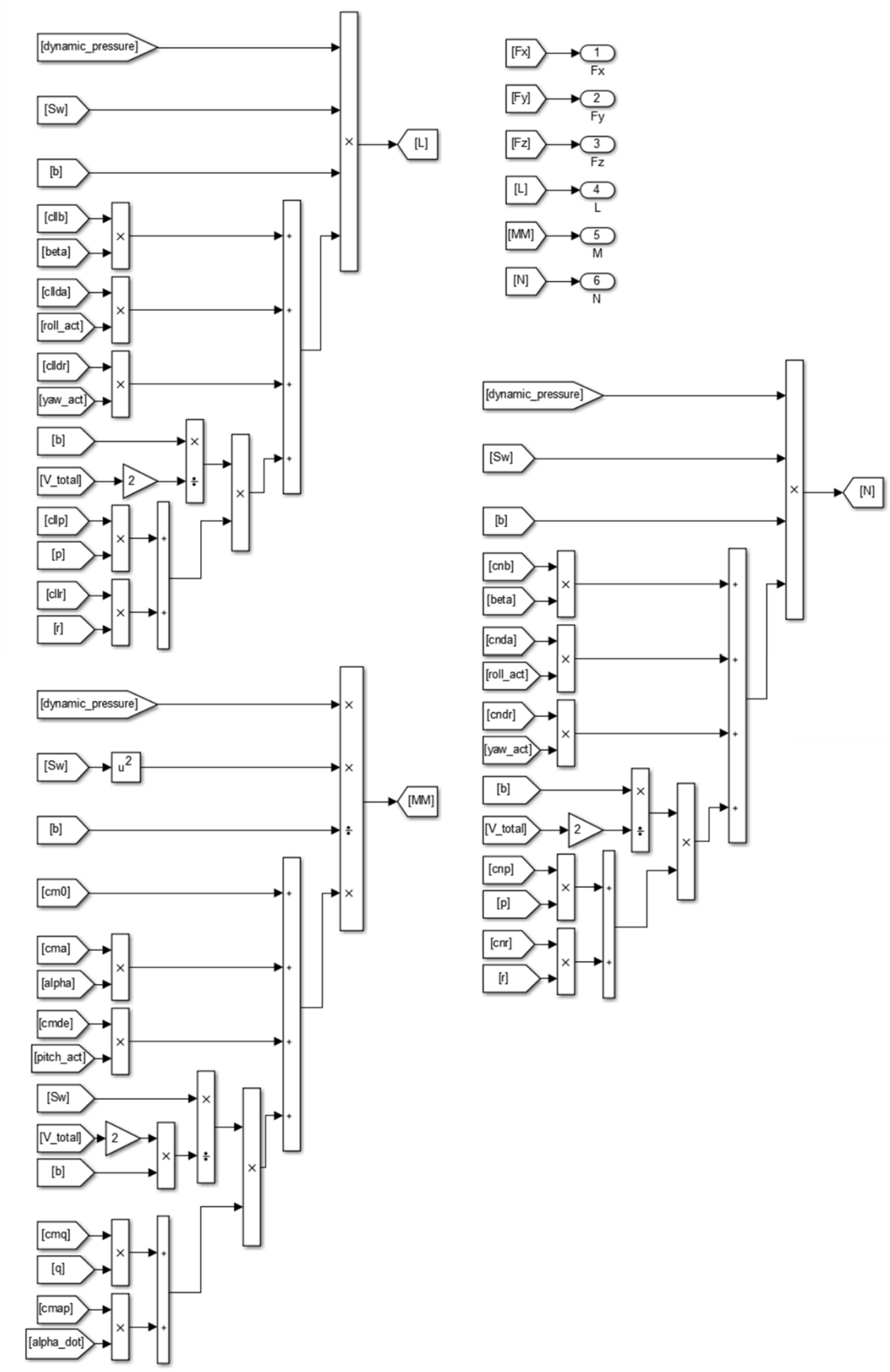

Figura 40: Sección D del bloque "Force and moment calculation"

Por último, la **sección D** contiene el cálculo de momentos aerodinámicos y el envío de las señales resultantes hacia los outputs mediante los bloques "From".

F.2) Cálculo de estados

Para realizar el proceso de cálculo de estados a partir de las fuerzas y los momentos aplicados a la aeronave se va a utilizar un bloque perteneciente a la librería de bloques de Simulink®, el bloque "6DOF (Euler Angles)" [21].

Este bloque considera la rotación de un sistema de ejes fijados a la aeronave (sistema de ejes cuerpo, cuyos ejes se denominan en este desarrollo X_b, Y_b, Z_b) alrededor del sistema de ejes Tierra (X_e, Y_e, Z_e). El origen del sistema de ejes cuerpo es el centro de gravedad de la aeronave, y ésta se supone rígida, hipótesis que elimina la necesidad de considerar las fuerzas que actúan entre elementos de masa individuales. El sistema de ejes Tierra se considera inercial, una excelente aproximación que permite depreciar las fuerzas debidas al movimiento de la Tierra con respecto a las "estrellas fijas".

El movimiento de translación del sistema de ejes cuerpo se muestra en las ecuaciones (41), donde las fuerzas aplicadas

$$\begin{bmatrix} F_x & F_y & F_z \end{bmatrix}^T \tag{40}$$

se dan en el mismo sistema de referencia, y la masa m de la aeronave se asume constante.

$$\begin{aligned} \bar{F}_b &= \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m\left(\dot{\bar{V}}_b + \bar{\omega} \times \bar{V}_b\right) \\ \bar{V}_b &= \begin{bmatrix} u_b \\ v_b \\ w_b \end{bmatrix}; \; \bar{\omega} = \begin{bmatrix} p \\ q \\ r \end{bmatrix} \end{aligned} \tag{41}$$

La dinámica rotacional del sistema de ejes cuerpo se muestra en las ecuaciones (43), donde los momentos aplicados son

$$\begin{bmatrix} L & M & N \end{bmatrix}^T \tag{42}$$

y el tensor de inercia I está referido al origen de los ejes (centro de masas de la aeronave).

$$\begin{aligned} \bar{M}_B &= \begin{bmatrix} L \\ M \\ N \end{bmatrix} = I\dot{\bar{\omega}} + \bar{\omega} \times (I\bar{\omega}) \\ I &= \begin{bmatrix} I_{xx} & -I_{xy} & -I_{xz} \\ -I_{yx} & I_{yy} & -I_{yz} \\ -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix} \end{aligned} \tag{43}$$

La relación entre el vector velocidad angular en el sistema de ejes cuerpo,

$$\begin{bmatrix} p & q & r \end{bmatrix}^T \tag{44}$$

Y la derivada con respecto al tiempo de los ángulos de Euler,

$$\begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T \tag{45}$$

Se pueden determinar resolviendo las derivadas de los ángulos de Euler en el sistema de ejes cuerpo:

$$\begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\phi & \sin\phi \\ 0 & -\sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \equiv J^{-1} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{46}$$

Invirtiendo J se consigue la relación requerida para determinar el vector de derivadas de los ángulos de Euler.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = J \begin{bmatrix} p \\ q \\ r \end{bmatrix} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \dfrac{\sin\phi}{\cos\theta} & \dfrac{\cos\phi}{\cos\theta} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{47}$$

3.2.3.2 Subsistema "Linear Model"

Aunque es recomendable utilizar un modelo no lineal de la aeronave a simular como el descrito en el apartado 3.2.3.1, también se ha desarrollado una estructura de modelo lineal, que se puede utilizar en caso de que no se disponga de otra opción [22][23].

A) La representación en espacio de estados

El modelo que se ha desarrollado utiliza la representación en espacio de estados, que permite establecer derivadas parciales de la evolución de los estados debida a entradas en los propios estados y a entradas en las superficies de control, en forma de coeficientes. Este planteamiento es similar al concepto de las derivadas aerodinámicas, siendo la diferencia que estas últimas se refieren a las fuerzas y los momentos, que han de ser resueltos mediante ecuaciones para obtener la dinámica de estados, mientras que los coeficientes del espacio de estados se refieren a los propios estados finales.

Consiste en un modelo matemático de un sistema físico compuesto por variables de inputs, outputs y estados que están relacionadas mediante ecuaciones diferenciales de primer orden. Las variables se representan como vectores. Adicionalmente, si la dinámica del sistema es lineal, invariante con el tiempo y dimensionalmente finita, entonces las ecuaciones diferenciales se pueden expresar en notación matricial.

La representación en espacio de estados general de un sistema lineal con p inputs, q outputs y n variables de estado se escribe de la siguiente manera.

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \mathbf{A}(t)\cdot\mathbf{x}(t)+\mathbf{B}(t)\cdot\mathbf{u}(t)\\ \mathbf{y}(t) &= \mathbf{C}(t)\cdot\mathbf{x}(t)+\mathbf{D}(t)\cdot\mathbf{u}(t)\end{aligned} \tag{48}$$

Donde

$\mathbf{x}$ es el vector de estados ($\mathbf{x}(t)\in\mathbb{R}^n$);

$\mathbf{y}$ es el vector de outputs ($\mathbf{y}(t)\in\mathbb{R}^q$);

$\mathbf{u}$ es el vector de inputs ó vector de control ($\mathbf{u}(t)\in\mathbb{R}^p$);

$\mathbf{A}$ es la matriz de estados ($\dim(\mathbf{A})=n\times n$);

$\mathbf{B}$ es la matriz de inputs ($\dim(\mathbf{B})=n\times p$);

$\mathbf{C}$ es la matriz de outputs ($\dim(\mathbf{C})=q\times n$);

$\mathbf{D}$ es la matriz de "feedthrough" (o "feedforward"), para casos en los que es necesario realimentar directamente a las entradas con las salidas. Normalmente se define como una matriz nula ($\dim(\mathbf{D})=q\times p$).

B) Implementación en el simulador

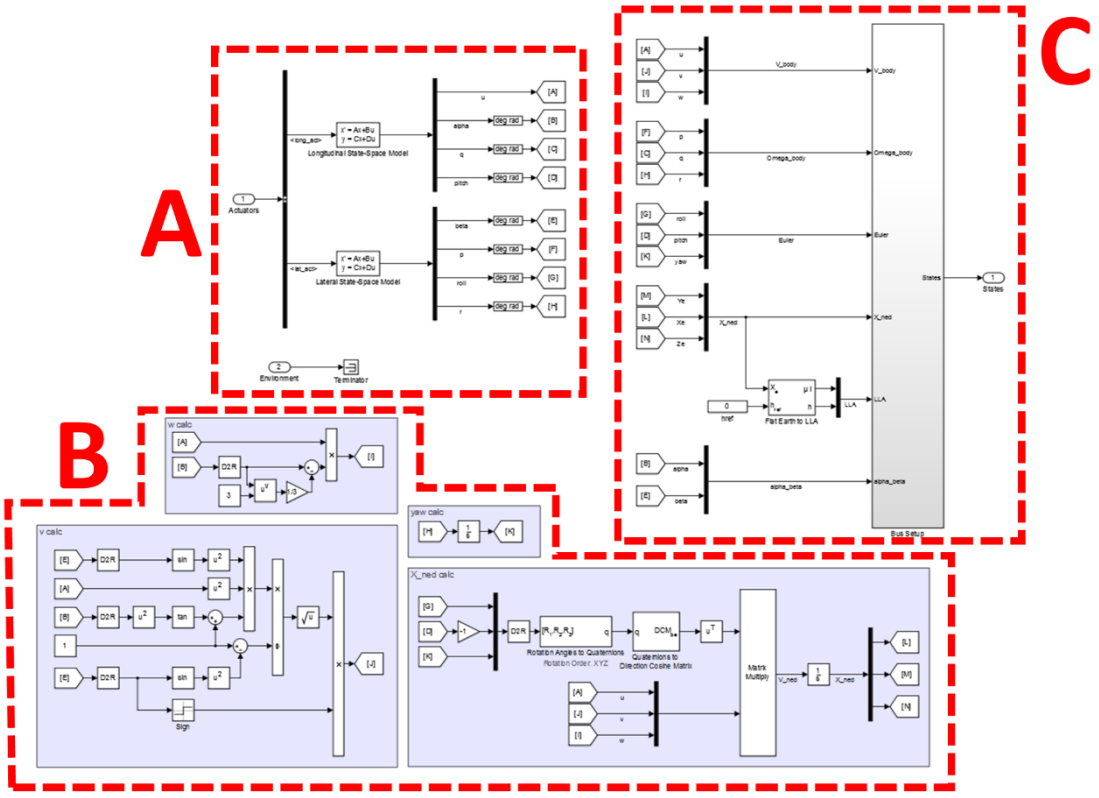

Figura 41: Estructura del bloque "Linear Model"

En la Figura 41 se puede observar el modelo dinámico lineal basado en la representación en espacio de estados implementado en el simulador. Se ha dividido la estructura en tres secciones para una correcta visualización.

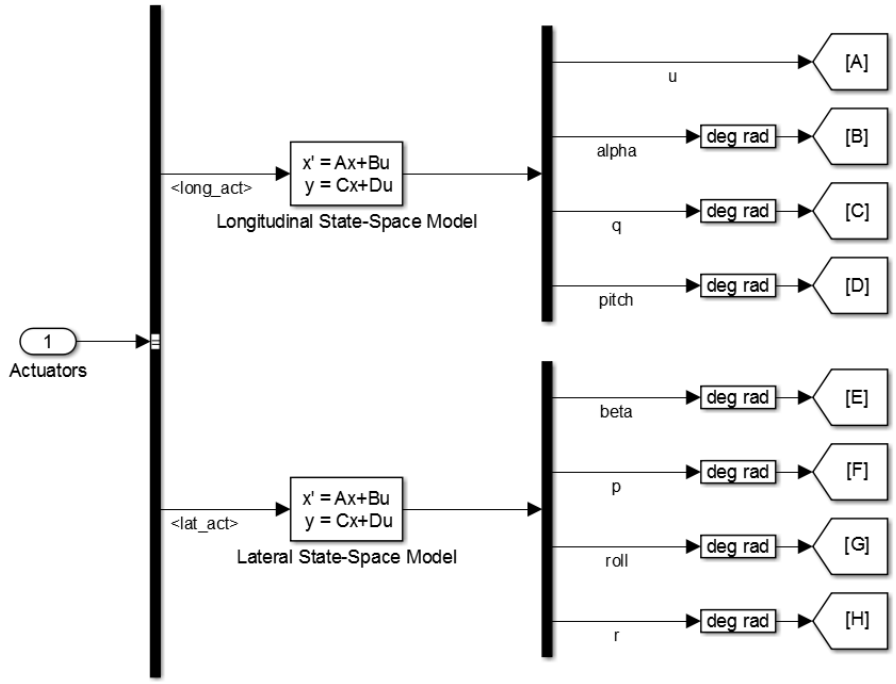

Figura 42: Sección A del bloque "Linear Model"

La **sección A** (Figura 42) contiene el modelo de espacio de estados. Se han utilizado los bloques "State-Space" pertenecientes a la librería de bloques de Simulink® [24]. Se ha dividido el modelo dinámico en dos modelos independientes, el longitudinal y el lateral-direccional. En los bloques "Stace-Space" se establecen las matrices A, B, C y D y las condiciones iniciales de los estados de la aeronave (Figura 43), todas ellas definidas en el script "startVars.m".

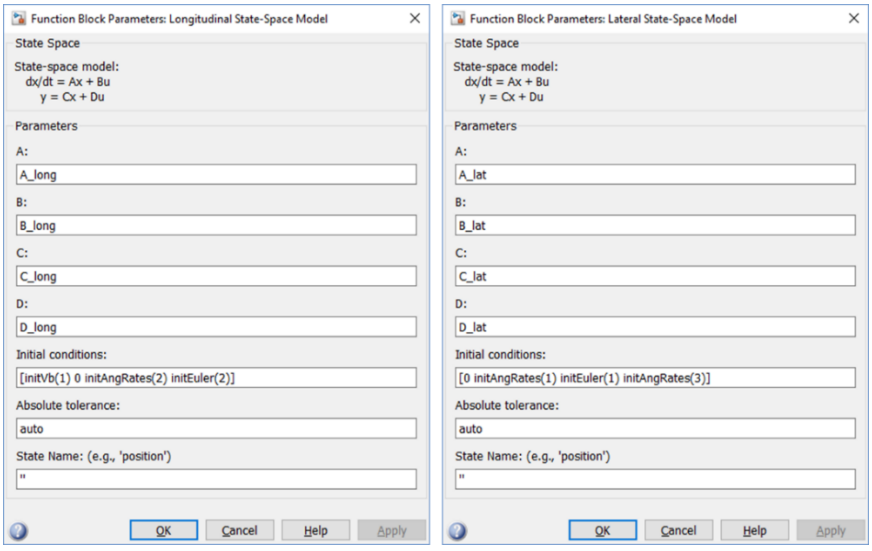

Figura 43: Opciones de los bloques "State-space"

La **sección B** se encarga del cálculo de estados que no se obtienen directamente a partir del modelo de espacio de estados que se ha utilizado en este caso, a partir de los que sí se obtienen directamente. Se muestra a continuación (Figura 44).

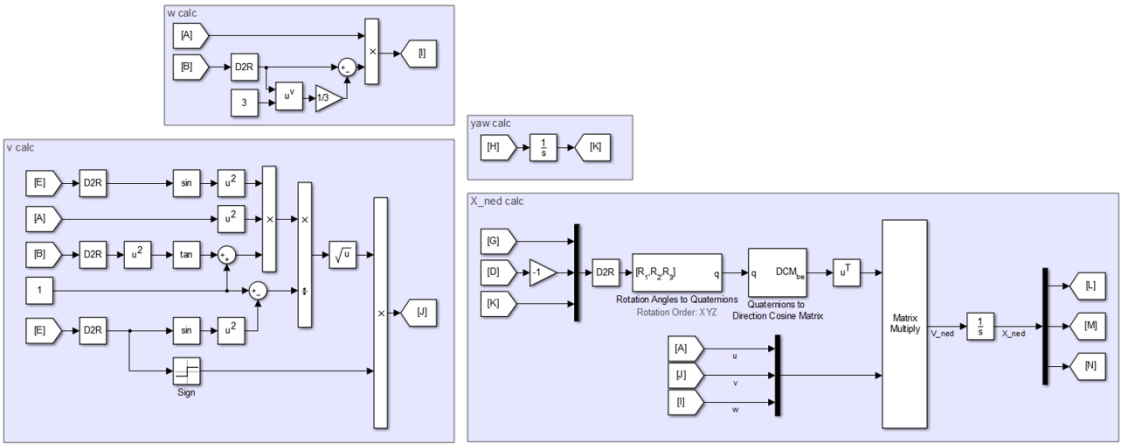

Figura 44: Sección B del bloque "Linear Model"

Por último, la **sección C** contiene la estructura de agrupación de los outputs de estados en un Bus individual.

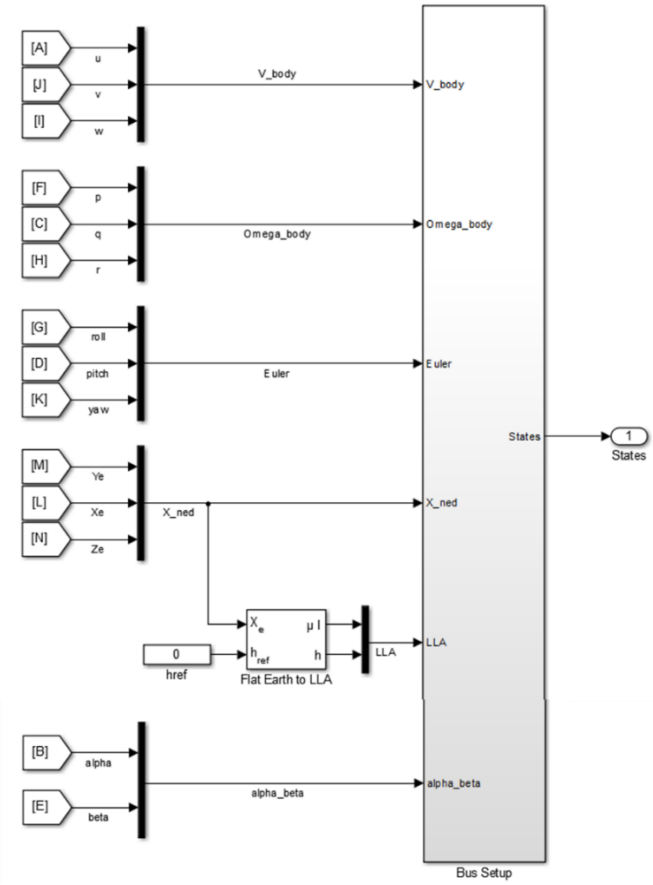

Figura 45: Sección C del bloque "Linear Model"

Se puede apreciar que se ha utilizado el bloque “Flat Earth to LLA”. Este bloque convierte el vector tridimensional de la posición de la aeronave referenciada en ejes Tierra a un vector que contiene la posición mediante las componentes latitud, longitud y altitud geodésicas [25].

3.2.4 Ambiente externo

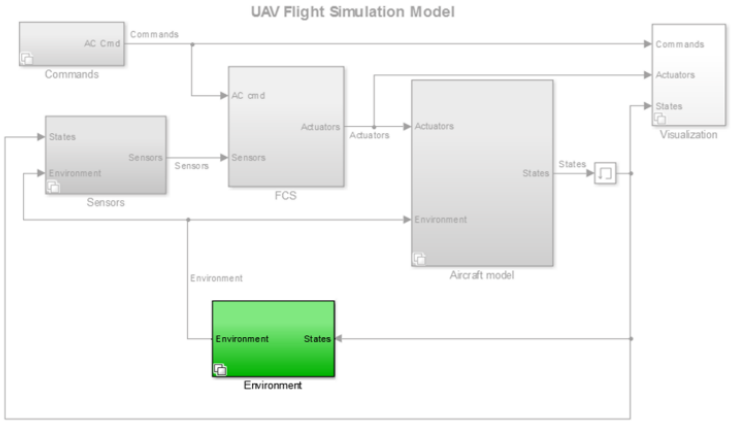

Figura 46: Ubicación del subsistema "Environment" en el simulador

El bloque “Environment” se encarga de modelar las condiciones físicas de la atmósfera terrestre. Utilizando su único input, los estados del avión, devuelve como output las condiciones físicas atmosféricas en el punto concreto donde se encuentra la aeronave.

En el siguiente nivel de este bloque se han incluido dos subsistemas distintos que utilizan la función de “Variant Subsystem”, para tener uno u otro activo pero nunca ambos simultáneamente. Uno de ellos está diseñado para proveer unas condiciones físicas dependientes de la posición en la atmósfera (“Environment -Variable-”) mientras que el otro provee condiciones físicas constantes (“Environment -Constant-”). Este último subsistema está indicado para disminuir la carga computacional en simulaciones que van a transcurrir dentro de una franja de altitudes donde la densidad no varía apreciablemente. Además, es necesario recordar que, como se indicó anteriormente, el modelo no lineal de la aeronave tiene en cuenta los cambios de densidad con la altitud, por tanto se debe usar el subsistema de ambiente constante cuando esté activado el modelo no lineal de la aeronave.

3.2.4.1 Subsistema “Environment (Variable)”

A continuación se presenta un esquema del subsistema de ambiente variable (Figura 47).

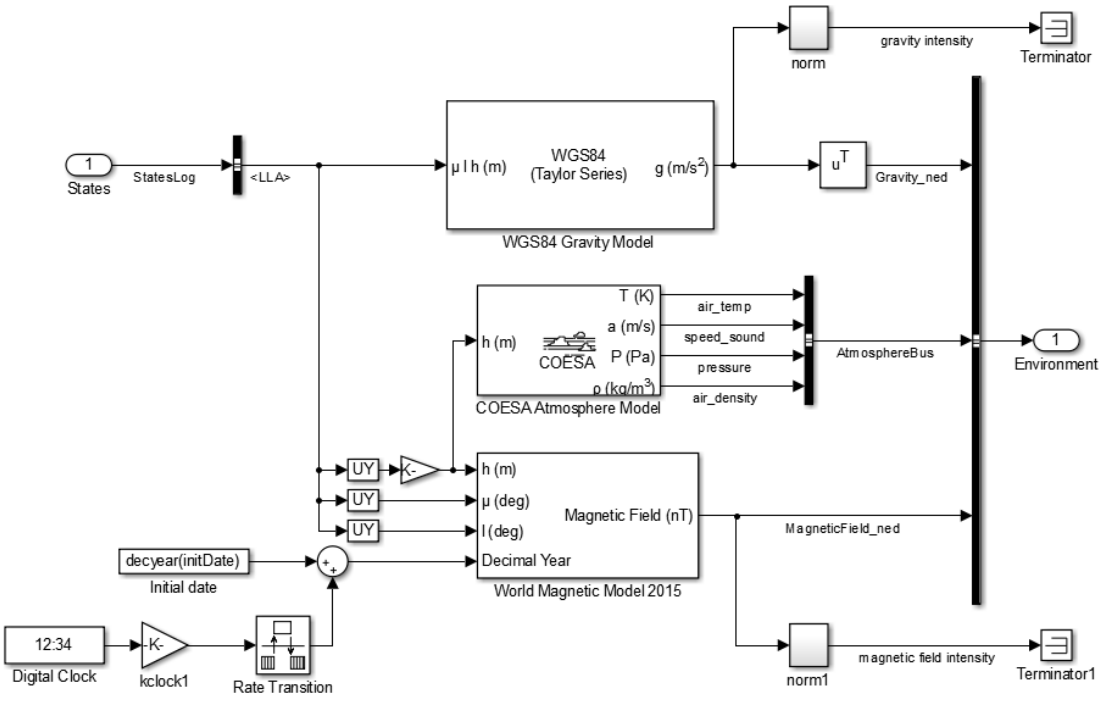

Figura 47: Estructura del subsistema "Environment (Variable)"

Se aprecian tres unidades principales dentro de este subsistema:

- En la parte superior se aprecia el modelo gravitatorio terrestre (bloque "WGS84 Gravity Model") [26]. Este bloque calcula la gravedad terrestre en una localización específica utilizando el "World Geodetic System" (WGS 84). El modelo WGS 84 se define como un elipsoide geocéntrico equipotencial. Este modelo se puede encontrar en NIMA TR8350.2, "Department of Defense World Geodetic System 1984, Its Definition and Relationship with Local Geodetic Systems." La altitud se introduce en el bloque en las mismas unidades que las seleccionadas para la gravedad, en este caso metros, la latitud y longitud geodéticas se introducen en grados sexagesimales. Este bloque por tanto tiene como entrada la latitud, longitud y altitud de la aeronave, proveniente del input "States" y devuelve como output un vector de tres dimensiones que corresponde al vector gravedad en el sistema de coordenadas Tierra geodética, o NED (North-East-Down).

- En la parte central encontramos el modelo atmosférico terrestre. El bloque que se ocupa de esta tarea es el "COESA Atmosphere Model" [27]. Este bloque funciona utilizando el modelo "1976 COESA-extended U.S. Standard Atmosphere". Dada una altitud geopotencial, calcula la temperatura absoluta, la presión y la densidad utilizando fórmulas de interpolación estándares. El modelo COESA extrapola la temperatura linealmente y la presión y la densidad logarítmicamente a partir del rango de altitudes entre 0 y 84852 metros geopotenciales. Se calcula la densidad y la presión utilizando una relación de gas perfecto. Se ha utilizado el sistema métrico en este bloque. Por tanto, el bloque se alimenta de la altitud geopotencial de la aeronave y da como outputs

la temperatura absoluta, la velocidad de propagación del sonido, la presión y la densidad del aire en ese punto.

- Por último, se puede apreciar el modelo magnético terrestre. El bloque encargado de llevar a cabo esta tarea es el "World Magnetic Model 2015" [28]. Este bloque calcula el campo magnético terrestre en una localización y tiempo específicos utilizando el "World Magnetic Model (WMM)". Este modelo sólo es válido en el intervalo de tiempo entre el año 2015 y 2020. El WMM-2015 se puede encontrar en la web en goo.gl/boJaxn y en "NOAA Technical Report: The US/UK World Magnetic Model for 2015-2020". La altitud se introduce en unidades del sistema métrico mientras que la latitud y la longitud se introduce en grados sexagesimales. El input temporal se consigue mediante los subsistemas "Initial Date" y "Digital Clock". El subsistema "Initial Date" devuelve el año de la fecha inicial declarada en el script de inicialización "startVars.m". El subsistema "Digital Clock" da como output el tiempo transcurrido de simulación en unidad años, teniendo en cuenta si el año es bisiesto. Este bloque se ejecuta cada cinco segundos, de ahí que sea necesario añadir el bloque "Rate Transition" para adaptar esta frecuencia de ejecución a la del resto del simulador.

Aunque este subsistema provee datos gravitatorios, atmosféricos y magnéticos terrestres, los modelos de aeronave utilizados en este simulador solamente recogen datos de densidad atmosférica y de gravedad. Aun así, se mantienen los modelos del resto de parámetros por si se quisiera mejorar la precisión del modelo en un futuro.

3.2.4.2 Subsistema "Environment (Constant)"

A continuación se presenta un esquema del subsistema de ambiente constante.

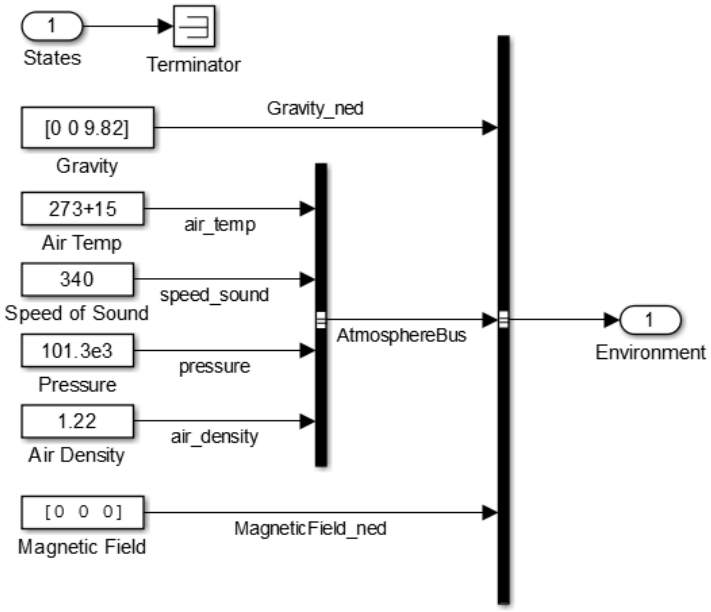

Figura 48: Estructura del subsistema "Environment (Constant)"

Como apreciamos en el diagrama, este subsistema no contiene ningún modelo atmosférico, en su lugar podemos definir las variables físicas atmosféricas deseadas como constantes.

Como se mencionó anteriormente, este subsistema es especialmente útil para disminuir la carga computacional en simulaciones que van a transcurrir dentro de una franja de altitudes donde la densidad no varía apreciablemente. Además, es recomendable activar este subsistema si se encuentra activo el modelo no lineal de la aeronave, ya que éste tiene en cuenta la variabilidad de la densidad atmosférica con la altitud.

3.2.5 Sensores

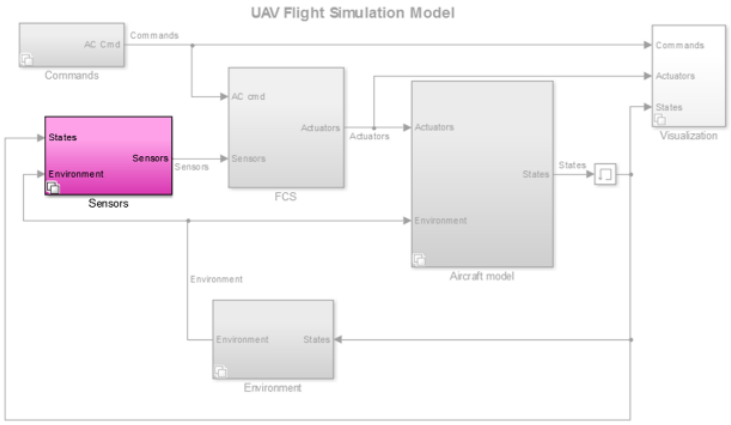

Figura 49: Ubicación del subsistema "Sensors" en el simulador

Los sensores de una aeronave se encargan de proveer información que se puede utilizar para conocer los estados de la aeronave en cada momento (por ejemplo posición y orientación). Entre otras técnicas, se utiliza la *navegación inercial* [29].

El funcionamiento de los sistemas de navegación inercial se basa en las leyes de la mecánica clásica tal y como las formuló Newton [30]. Las leyes de Newton dicen que el movimiento de un cuerpo será uniforme y en una línea recta a no ser que una fuerza externa lo perturbe o que el cuerpo se hallara en reposo. Las leyes también dicen que esta fuerza producirá una aceleración proporcional en el cuerpo. Dada la habilidad de medir esa aceleración, sería posible calcular el cambio en la velocidad y la posición mediante sucesivas integraciones matemáticas de la aceleración con respecto al tiempo. La aceleración se puede medir utilizando un dispositivo denominado *acelerómetro*. Un sistema de navegación inercial habitualmente contiene tres de estos dispositivos, siendo cada uno de ellos capaz de medir la aceleración en una sola dirección. Es habitual que los acelerómetros se instalen con sus ejes principales perpendiculares entre sí.

Con el fin de navegar con respecto a un sistema inercial de referencia, es necesario saber en todo momento la dirección en la que apuntan los acelerómetros. El movimiento rotacional del cuerpo con respecto al sistema inercial de referencia se puede medir utilizando *giróscopos* y de esta manera asegurar que se conoce la orientación de los acelerómetros en todo momento. Dada esta información, es posible resolver las aceleraciones en el sistema de referencia antes de que se lleve a cabo el proceso de integración.

Por tanto, la navegación inercial es el proceso mediante el cual las medidas tomadas por giróscopos y acelerómetros se utilizan para determinar la posición del vehículo en el que se encuentran instalados.

Sin embargo los giróscopos y acelerómetros no toman medidas perfectas. Existen diversas fuentes de error que deben ser tenidas en cuenta. Algunos errores pueden ser acumulativos a lo largo del proceso de integración, por lo que en muchos casos es necesario utilizar técnicas destinadas a disminuir dichos errores.

De vuelta al simulador en desarrollo, se observa en un primer nivel que el subsistema "Sensors" utiliza nuevamente la función de subsistema variable. Esta vez existen dos bloques que se pueden activar individualmente.

3.2.5.1 Subsistema "Sensors (Feedthrough)"

Este subsistema modela una IMU (Inertial Measurement Unit) ideal que no proporciona error ninguno en los estados medidos. Es decir, los estados reales pasan directamente a ser los estados medidos, por lo que la estructura es extremadamente simple (Figura 50).

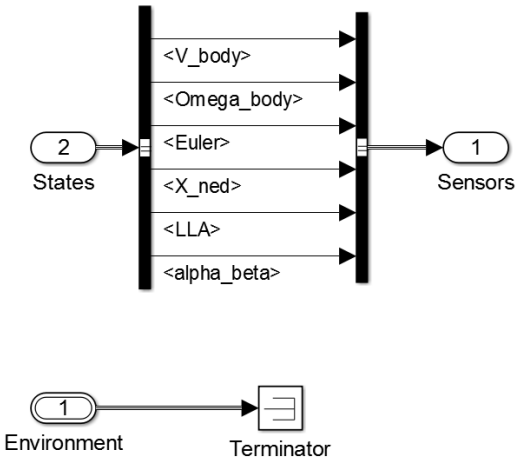

Figura 50: Estructura del subsistema "Sensors (Feedthrough)"

Este subsistema se puede utilizar para casos en los que los errores de medida se consideren despreciables.

3.2.5.2 Subsistema "Sensors (Dynamics)"

Este subsistema introduce en las señales de los estados reales los errores correspondientes a una IMU real, por lo que los estados medidos y los reales son distintos. A continuación se muestra la estructura que se ha utilizado para modelar la unidad de medida inercial.

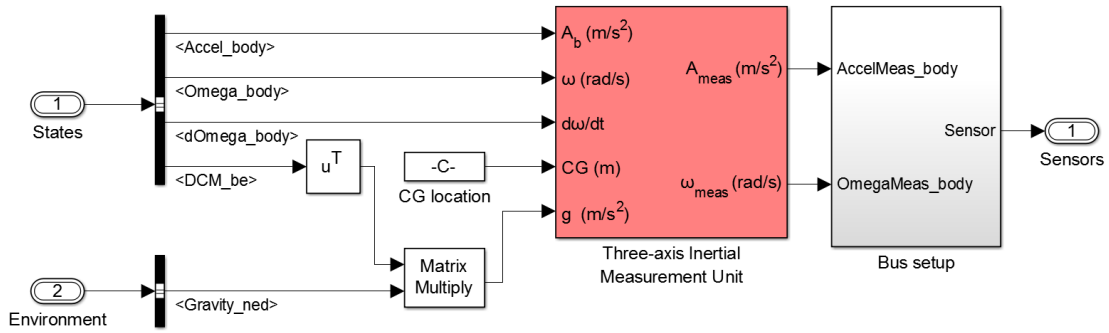

Figura 51: Estructura del subsistema "Sensors (Dynamics)"

El bloque "Three-axis Inertial Measurement Unit" pertenece a la librería de Simulink® [31]. Este bloque modela los errores introducidos por tres acelerómetros [32] y tres giróscopos [33].

3.2.6 Visualización

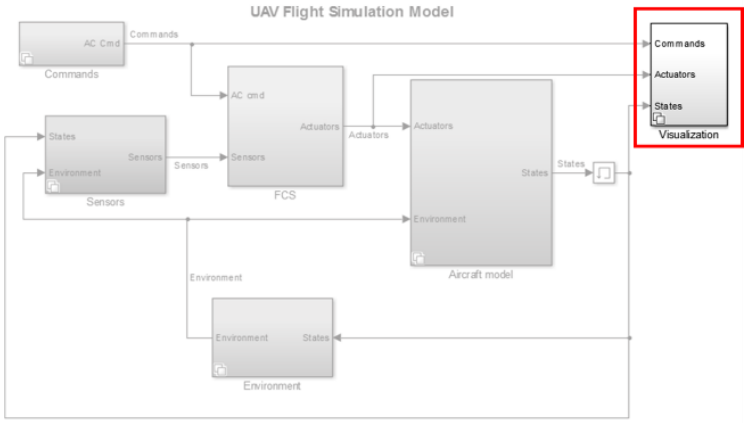

Figura 52: Ubicación del subsistema "Visualization" en el simulador

El bloque de visualización se ha desarrollado con el objetivo de proporcionar una adecuada interfaz gráfica que facilite la visualización de los resultados [34]. Nuevamente se ha utilizado en este bloque la función de "Variant Subsystem". En este bloque encontramos tres subsistemas que se pueden activar de manera individual y excluyente. Estos subsistemas son (1) FlightGear (2) Scopes (3) Workspace.

3.2.6.1 Subsistema "FlightGear"

Este subsistema es el que tiene mayor importancia de los subsistemas del bloque "Visualization".

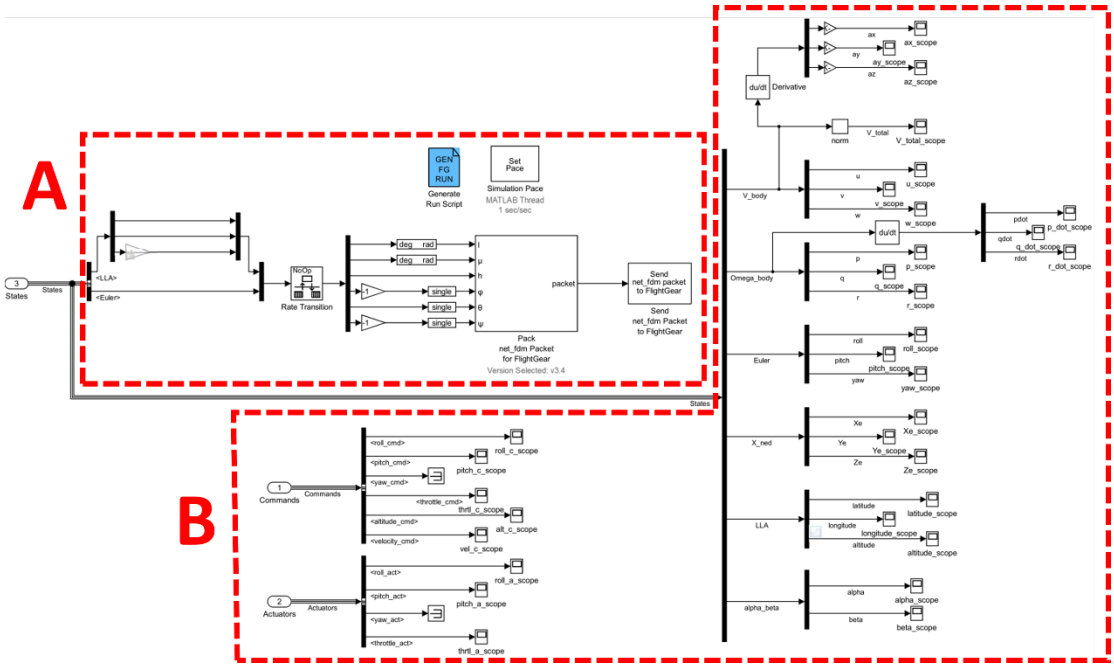

Figura 53: Estructura del subsistema "FlightGear"

La estructura del subsistema ha sido dividida en dos zonas o secciones con propósitos distintos, que se detallarán a continuación.

La **zona A** (Figura 54) tiene la función de crear y enviar paquetes de datos de los estados del avión al software externo FlightGear, que será utilizado como motor gráfico.

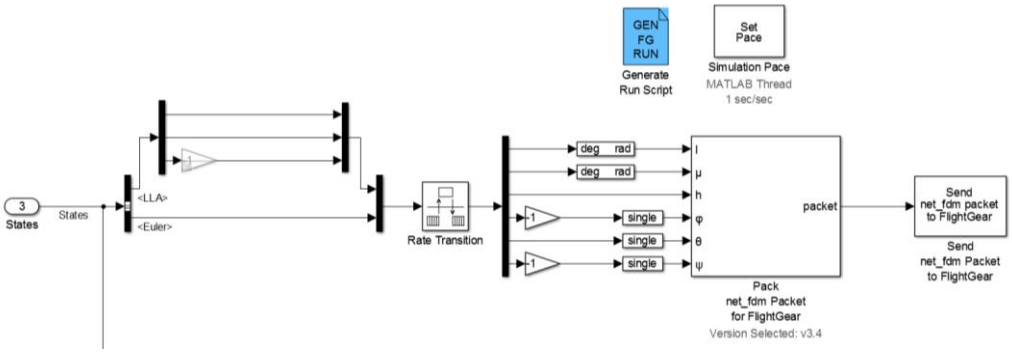

Figura 54: Zona A del subsistema "FlightGear"

El núcleo principal de la estructura lo componen dos bloques:

El bloque "**Pack net_fdm Packet for FlightGear**" prepara el paquete de datos que va a ser enviado a FlightGear [35]. El paquete será denominado *net_fdm*. Este bloque admite varias combinaciones de inputs según el modo operativo que se elija. Por defecto, los inputs corresponden a la posición y la actitud del avión (latitud, longitud, altitud, ángulo de roll, ángulo de pitch, ángulo de yaw). Sin embargo se pueden establecer otros grupos de inputs para el

empaquetamiento de datos. Se pueden ver las distintas opciones de grupos de inputs para este bloque en la Figura 55.

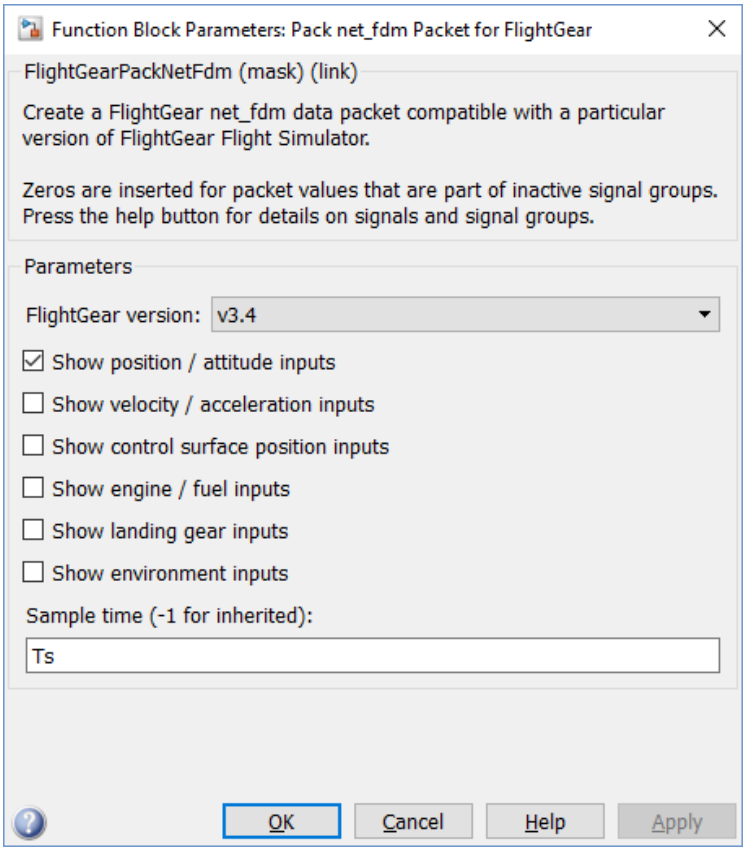

Figura 55: Opciones del bloque "Pack net_fdm Packet for FlightGear"

Cuantos más inputs se empaqueten para enviar a FlightGear, más detallada y completa será la posterior experiencia de visualización. Un grupo de inputs muy útil para enviar a FlightGear es el grupo de posición de las superficies de control, que ayuda a visualizar a tiempo real el movimiento de las superficies de control durante la simulación. Este grupo de inputs no se ha activado en la plataforma porque el diseño geométrico que se ha realizado de la aeronave X no dispone de partes móviles, característica necesaria para poder aprovechar esta funcionalidad.

El bloque "Pack net_fdm Packet for FlightGear" también permite establecer la versión de FlightGear a la que los datos van a ser enviados. De esta manera se asegura que la comunicación entre los dos programas de software (Simulink® y FlightGear) va a ser la adecuada.

Se ha elegido como tiempo de muestra el mismo al que se ejecuta el resto del simulador (0.005 s), definido por la variable "Ts", definida en el script de inicialización "startVars.m".

Seguidamente, el bloque "**Send net_fdm Packet to FlightGear**" se encarga de transmitir el paquete *net_fdm* previamente creado por el bloque "Pack net_fdm Packet for FlightGear" a FlightGear en el ordenador de trabajo, o un ordenador remoto conectado a través de una red al de trabajo [36]. El bloque requiere la definición de los siguientes campos (Figura 56).

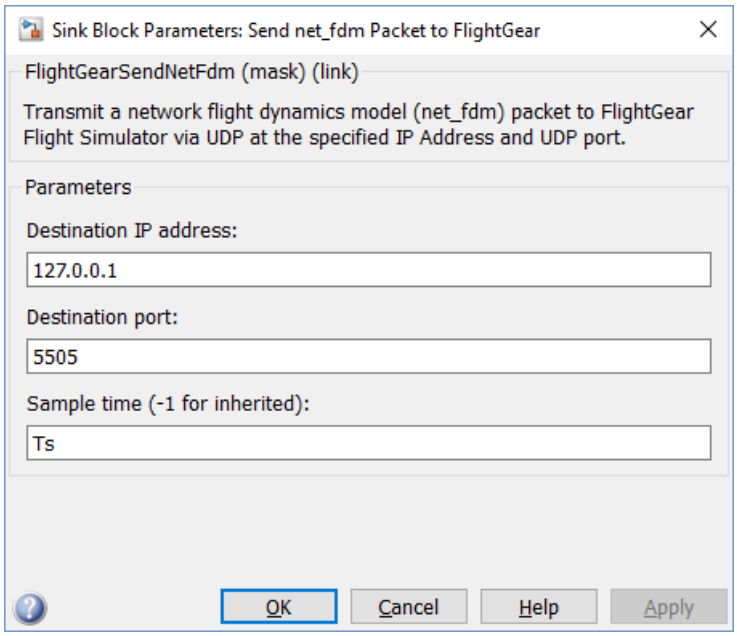

Figura 56: Opciones del bloque "Send net_fdm Packet to FlightGear"

La dirección IP de destino corresponde a la dirección donde se van a enviar los paquetes. Se puede usar la dirección 127.0.0.1 para apuntar al propio ordenador de trabajo. La posibilidad de poder enviar los paquetes a otra estación de trabajo es muy útil para los casos en los que el propio ordenador de trabajo no disponga de la potencia de procesamiento suficiente para ejecutar el simulador en Simulink® y el motor gráfico FlightGear simultáneamente. El puerto de destino debe ser un puerto que no se esté utilizando y que FlightGear puede usar si se ejecuta utilizando la línea de código

```
--fdm=network,localhost,5501,5502,5503
```

Donde el segundo puerto en la lista, 5502, es el puerto de modelo dinámico de vuelo ("flight dynamics network" ó fdm) de la red. Mediante la utilización de un bloque que se explicará más adelante, "Generate Run Script", no es necesario que el usuario se preocupe por inicializar FlightGear mediante esta línea de código porque el propio bloque genera el script de ejecución requerido. Por último, el bloque requiere la definición de un tiempo de muestra o de ejecución. Si se define como -1 tomará el mismo tiempo de muestra que su señal de input.

Existe también, aunque no se han utilizado en el desarrollo de la plataforma, dos bloques relacionados con los que han sido expuestos y que realizan las tareas recíprocas a estos. Son los bloques "**Receive net_ctrl Packet from FlightGear**" y "**Unpack net_ctrl Packet from FlightGear**". El primero recibe un paquete de datos proveniente de FlightGear mientras que el segundo desempaqueta los datos que contiene y los convierte en variables usables en el entorno de Simulink®.

Además de los dos bloques que se han explicado, "Pack net_fdm Packet for FlightGear" y "Send net_fdm Packet to FlightGear", encontramos el bloque "**Generate Run Script**". Este bloque es independiente, es decir, no recibe ni da señales. El bloque se utiliza para generar un Script de ejecución de FlightGear que se puede utilizar para inicializar FlightGear de manera customizada

para que funcione adecuadamente con la plataforma [37]. El bloque requiere definir los siguientes campos (Figura 57, Figura 58 y Figura 59).

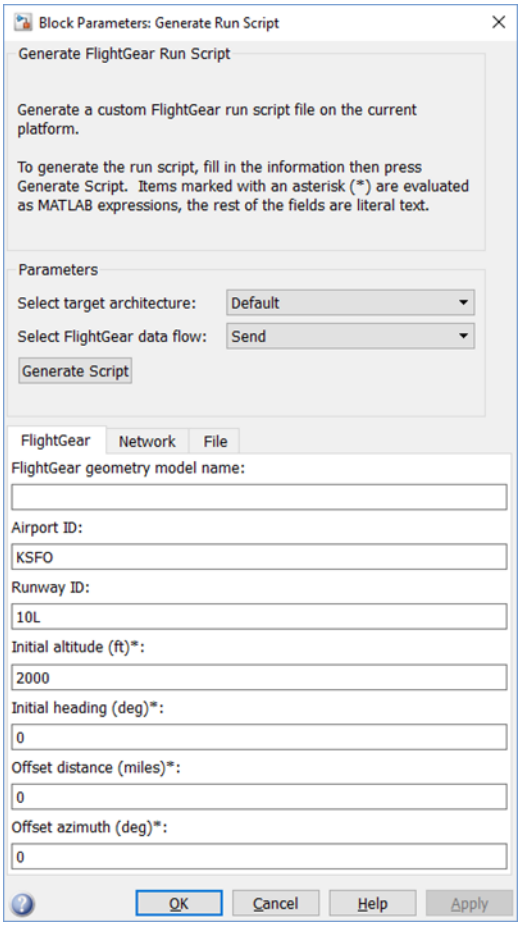

Figura 57: Opciones del bloque "Generate Run Script", pestaña FlightGear

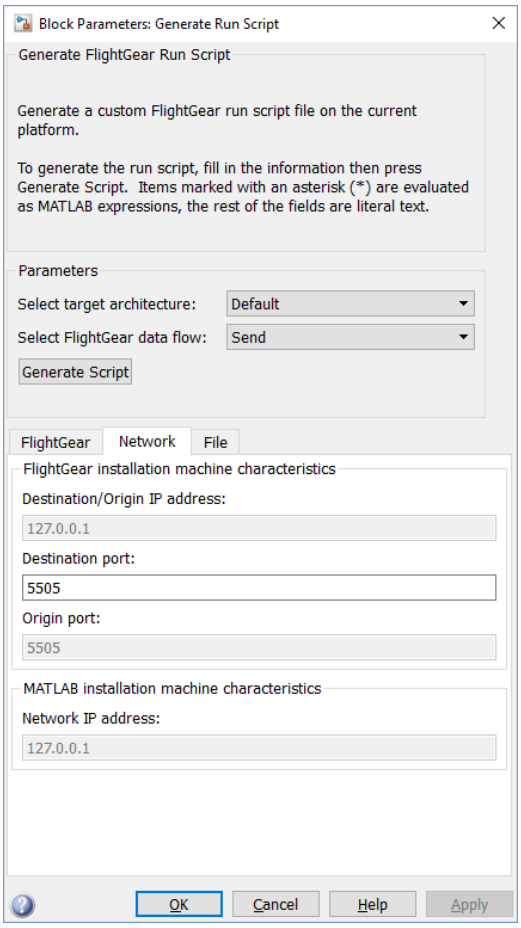

Figura 58: Opciones del bloque "Generate Run Script", pestaña Network

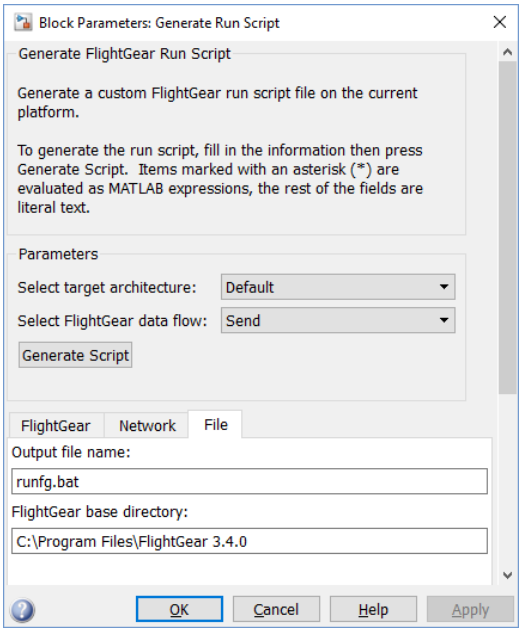

Figura 59: Opciones del bloque "Generate Run Script", pestaña File

En la pestaña *FlightGear* se define la geometría de la aeronave que se quiere utilizar (este aspecto se explicará con más detalle en el capítulo 4), así como la posición inicial de la aeronave en la visualización.

En la pestaña *Network* se definen la dirección IP de origen/destino y los puertos de origen y de destino de la estación donde se ha instalado FlightGear así como la dirección IP de la estación donde se haya instalado MATLAB®. En el caso de que sólo se utilice una estación de trabajo el único campo que estará activo para editar es el del puerto de destino, que debe definirse con el mismo valor del puerto de origen (definido también en "Send net_fdm Packet to FlightGear").

Por último, en la pestaña *File* se definen tanto el nombre que tendrá el script generado (formato .bat) como el directorio de instalación de FlightGear en el ordenador.

Además, es posible definir la arquitectura del ordenador de destino (Windows 32 o 64 bits, Linux o Mac) en "Select target architecture" y el modo de flujo de datos requerido (enviar datos, recibir datos o enviar y recibir datos) en "Select FlightGear data flow". Es importante establecer el modo de flujo de datos correctamente si se quisiera utilizar los bloques destinados a recibir y desempaquetar datos que han sido descritos previamente.

Una vez rellenados todos los campos se puede generar el script de ejecución pulsando el botón "Generate Script". El script se generará instantáneamente en el directorio actual de MATLAB®.

Existe también en esta zona un bloque denominado "Simulation Pace" [38]. Este bloque permite que la simulación en su totalidad se ejecute al ritmo especificado para que las animaciones conectadas sean adecuadas estéticamente (Figura 60).

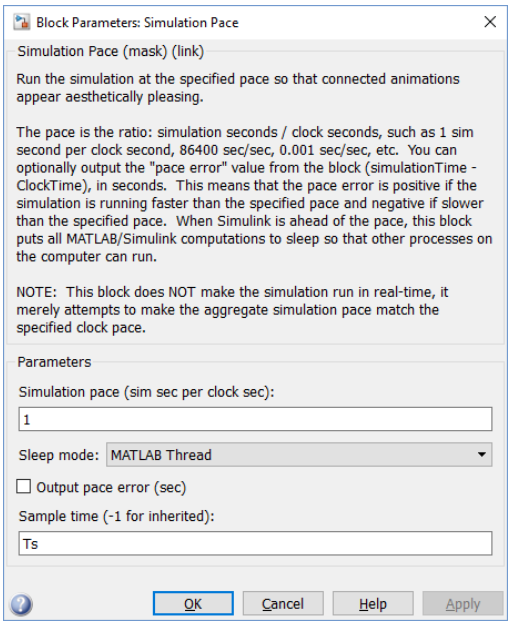

Figura 60: Opciones del bloque "Simulation Pace"

Este bloque no hace que la simulación se ejecute en tiempo real, simplemente hace tender el ritmo de ejecución del simulador al de un reloj virtual. En el caso de la plataforma, se ha

especificado un ritmo de 1 segundo de simulación por 1 segundo de reloj para que sea lo más parecido posible a una simulación en tiempo real.

Existen algunos bloques cuya función es adaptar las señales de estado para que entren en el bloque “Pack net_fdm Packet for FlightGear”. Entre ellos se encuentra el bloque “Rate transition”, que puede modificar la frecuencia de ejecución de la señal de salida si es necesario que sea distinta que la de entrada. Además, se observan bloques de cambio de unidades angulares para que las unidades de entrada sean las requeridas (grados Oeste/Norte para la longitud y latitud, metros sobre el nivel medio del mar para la altitud y radianes para valores de actitud) así como bloques de conversión de tipo de señal (los señales de actitud deben ser de tipo *single*).

En la **zona B** se individualiza cada señal de los grupos de señales (buses) *Estados*, *Comandos* y *Actuadores* para enviar cada una de ellas a un bloque “Scope”. Los bloques “Scope” tienen como objetivo proporcionar una visualización de los resultados del simulador en forma gráfica y a tiempo real. Estos bloques se pueden abrir individualmente a través de la interfaz gráfica.

3.2.6.2 Subsistema “Scopes”

El subsistema “Scopes” tiene como objetivo proporcionar una visualización de los resultados del simulador en forma gráfica y a tiempo real. Tiene una funcionalidad muy similar al subsistema “FlightGear” siendo la única diferencia que no envía paquetes de datos a este programa. A continuación se presenta un esquema del subsistema “Scopes” (Figura 61).

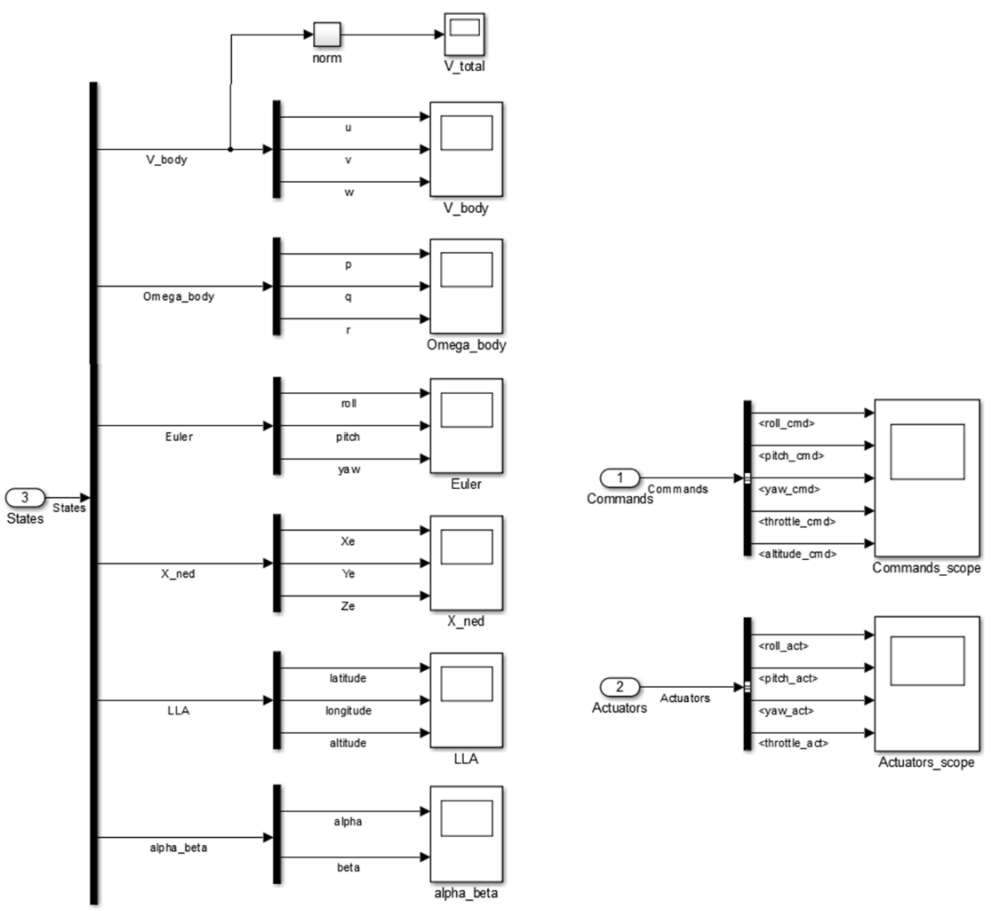

Figura 61: Estructura del subsistema "Scopes"

En la siguiente figura se muestra un ejemplo de la interfaz del bloque “Scope” de este subsistema (Figura 62), concretamente del bloque scope llamado “Euler” que muestra los ángulos de Euler.

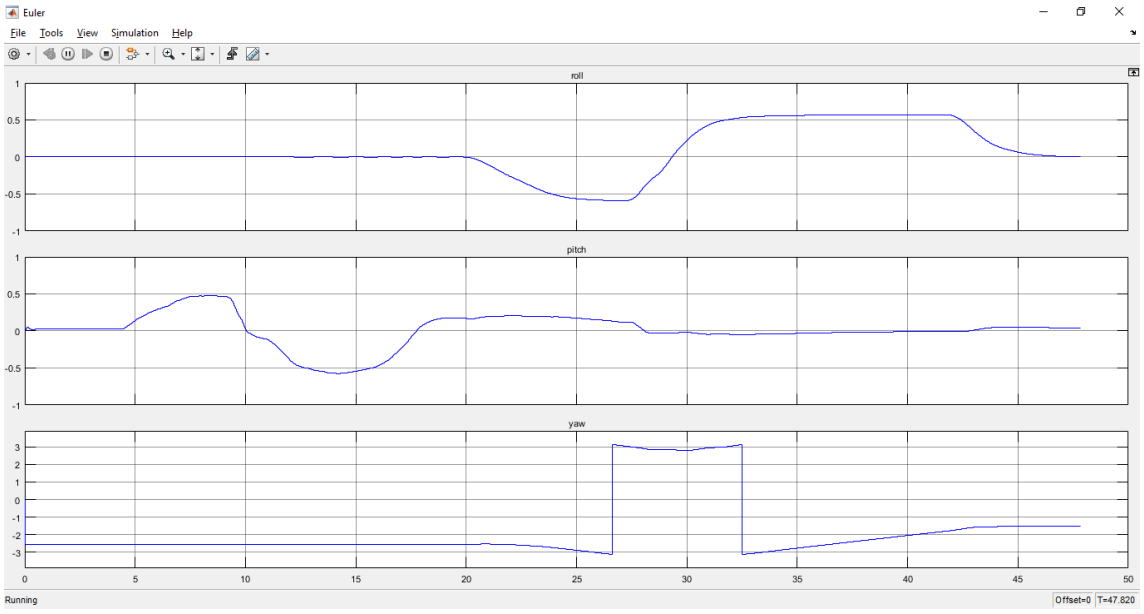

Figura 62: Interfaz del bloque "Scope"

3.2.6.3 Subsistema "Workspace"

El subsistema "Workspace" está destinado a la tarea de enviar los datos de salida del simulador a la "Base Workspace" de MATLAB®. Este concepto se refiere a un espacio de memoria reservado para albergar las variables que se crean durante una sesión de MATLAB®. Externalizar los datos de salida del simulador a MATLAB® permite realizar un análisis más detallado de éstos utilizando las herramientas y funciones que MATLAB® tiene incorporadas. En la siguiente figura se muestra un esquema del subsistema "Workspace" (Figura 63).

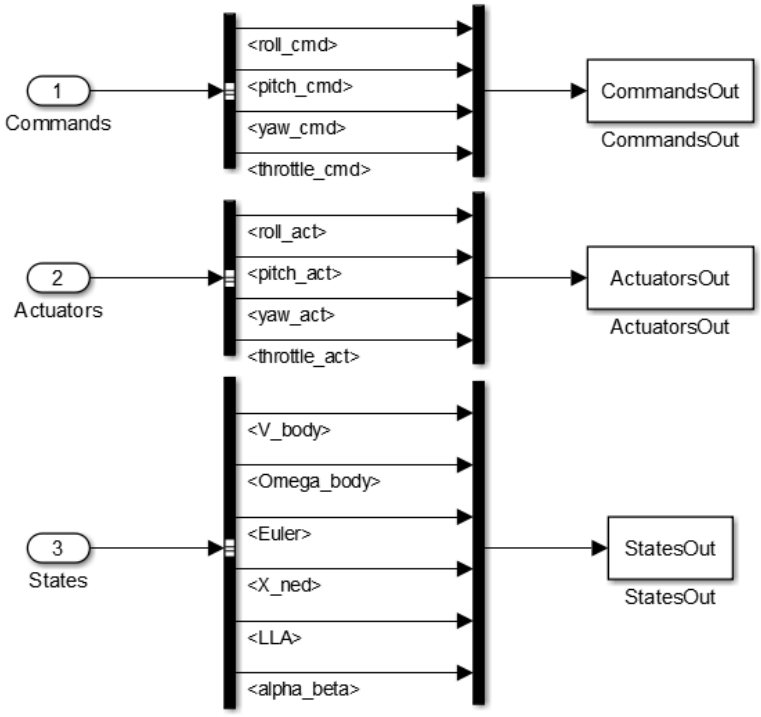

Figura 63: Estructura del subsistema "Workspace"

Los bloques "CommandsOut", "ActuatorsOut" y "StatesOut" son bloques "To Workspace". Estos bloques envían los datos que reciben como inputs al "Base Workspace" en formato de estructura con tiempo ("Structure with Time").

3.3 Scripts de inicialización

El simulador que se ha desarrollado requiere para su funcionamiento la inicialización de diversos parámetros y variables. Para facilitar este proceso, se han desarrollado varios scripts de inicialización que se detallarán a continuación.

3.3.1 Script "startVars.m"

Este script corresponde al script principal de inicialización de la plataforma. El código se muestra a continuación

```
% startVars.m - Initialize variables
% This script initializes variables required for the model to
```

```
% work.

clearvars

% Variants Conditions
Variants_Command = 1;       % 0: Signal builder, 1: Joystick, 2: Pre-
saved data
Variants_Sensors = 0;       % 0: Feedthrough, 1: Dynamics
Variants_Environment = 0;   % 0: Constant, 1: Variable
Variants_Vehicle = 1;       % 0: Linear dynamics, 1: Nonlinear
dynamics
Variants_Visualization = 2; % 0: Scopes, 1: Send values to workspace,
2: FlightGear.

% Add enum structure for the Variants
Simulink®.defineIntEnumType('Variants',{'Command','Vehicle','Environme
nt','Sensors','Visualization'},[0;0;0;0;0]);

% Bus definitions
asbBusDefinitionCommand;
asbBusDefinitionSensors;
asbBusDefinitionEnvironment;
asbBusDefinitionStates;

% Sampling rate
Ts= 0.005;

% Linear Model properties (space-state)
 A_long = [-0.107 1.04 -0.69 -19.31
          -0.015 -14.97 0.34 -0.028
          0.255 23.21 -11.65 0.32
          0.003 -7.56 0.52 -0.053];
 B_long = [-6.68 0.579
          -7.78 -0.018
          -138.68 0.087
          -9.76 -0.024];
 C_long = eye(4);
 D_long = zeros(4,2);

 A_lat = [-10.02 0.068 0.532 -0.91
            0.02 -8.94 -0.278 1.86
           -0.0065 0.403 -0.011 0.1
            0.189 -0.78 0.602 -1.82];
 B_lat = [-0.201 1.18
          -27.717 3.81
           -1.88 0.353
           4.194 -22.89];
 C_lat = eye(4);
 D_lat = zeros(4,2);

% Initial conditions
initDate = [2016 1 1 0 0 0];
initPosLLA = [40.548510 -3.612251 20];
initPosNED = [57 95 20];
initVb = [30 0 0];
initEuler = [0 0 10];
initAngRates = [0 0 0];

% MASS AND MOMENT OF INERTIA. Assumed pricipal axis of inertia
mass_config = 0;
```

```
if (mass_config == 0)
    m = 31.0; % kg
    Ix = 8.6; % kg*m^2
    Iy = 0.85; % kg*m^2
    Iz = 9.4; % kg*m^2
    Ixz = -0.05; % kg*m^3
elseif (mass_config == 1)
    m = 22.0; % kg
    Ix = 6.1; % kg*m^2
    Iy = 0.61; % kg*m^2
    Iz = 6.7; % kg*m^2
    Ixz = -0.05; % kg*m^3
end
inertia = [Ix 0 -Ixz;0 Iy 0;-Ixz 0 Iz];

% Engine Vars
Tmax = 70;
Vmax = 38;

% OTHERS
k1 = 2;
k2 = 2;
maxPitch = 0.7;
maxRoll = 0.54;
```

A continuación se detallará el funcionamiento de cada segmento de código del script en el orden en que se ejecuta.

(1) Eliminación de las variables que se ubican en el "Base Workspace" de MATLAB®.
(2) Definición de las condiciones de *Variants*. El valor que se asigne a cada una de estas condiciones determinará el subsistema que quedará activo en el simulador en los bloques que tienen incorporada esta funcionalidad. La leyenda de valores de asignamiento se encuentra comentada a la derecha de cada condición.
(3) Definición de objetos de bus llamando a otros scripts que se detallarán en el apartado 3.3.2.
(4) Definición de tiempo de muestreo
(5) Definición de coeficientes del modelo lineal de la aeronave representado en espacio de estados.
(6) Definición de condiciones iniciales de las variables de estado
(7) Definición de propiedades másicas de la aeronave
(8) Definición de parámetros del modelo de motor
(9) Otras variables

3.3.2 Otros Scripts

Una señal de Simulink® que se transmite por un bus puede ser virtual o no virtual. Cuando es virtual, la ruta de la señal que se visualiza en el modelo es sólo una conveniencia gráfica que no tiene ningún efecto funcional. En cambio si la señal es no virtual significa que ésta ocupa su propio espacio de memoria. Los buses de la plataforma son virtuales. Sin embargo, los siguientes scripts permiten crear objetos de bus en el caso de que se quisiera utilizar buses no virtuales en el modelo. Para más información acerca de los buses virtuales y no virtuales se pueden consultar las referencias [39] y [40].

- “asbBusDefinitionCommand.m”
- “asbBusDefinitionEnvironment.m”
- “asbBusDefinitionSensors.m”
- “asbBusDefinitionStates.m”

Capítulo 4

Implementación del motor gráfico FlightGear

Como se explicó en el apartado 1.4, uno de los objetivos del presente trabajo es conseguir una adecuada visualización de los estados de la aeronave en tiempo real. Para ello se ha preparado la interfaz para que los estados se envíen en forma de paquetes a un software externo que actúa como motor gráfico, FlightGear.

FG es un simulador de vuelo de código abierto multiplataforma que está en continuo desarrollo desde el año 1997. Se ha elegido utilizar este software como motor gráfico debido a que Simulink® tiene incorporadas diversas funcionalidades que facilitan el desarrollo de la infraestructura necesaria para el proceso de comunicación entre los dos programas. Ya se ha desarrollado la infraestructura que envía los paquetes de datos de los estados a FG (apartado 3.2.6.1). El presente capítulo se centra en los diversos procesos que han sido realizados para conseguir la correcta visualización en el motor gráfico.

En este trabajo se ha utilizado la versión de FG 3.4, ya que correspondía a la versión más reciente del mismo en el momento en que se desarrolló la plataforma de simulación.

En el apartado 3.2.6.1 se explicó cómo generar un script de inicialización de FG utilizando el bloque “Generate Run Script”. Este script va a llamar al programa FG con unos parámetros previamente customizados. Entre esos parámetros se encuentra el modelo de avión que se desea utilizar y los estados iniciales de la aeronave.

De forma predeterminada, la instalación de FG tiene varios modelos de aeronaves reales. Estos modelos incluyen modelos dinámicos y geométricos. El usuario de la plataforma tiene la opción de utilizar uno de los modelos geométricos que tiene FG por defecto, o en cambio diseñar la geometría de la aeronave deseada para utilizarla en FG. Una tercera opción consiste en descargar un modelo de aeronave existente a través de la librería de modelos de FG. Esta librería se puede consultar visitando el enlace “http://goo.gl/G6FyK3”. Para cualquier modelo que se utilice, es importante indicar a FG que no se va a utilizar el modelo dinámico, sino que la

dinámica del avión vendrá impuesta por los paquetes de datos que llegan desde Simulink®. Es decir, tan sólo se utiliza el modelo geométrico. Más adelante se detallará como llevar a cabo dicha indicación.

En el presente trabajo se ha utilizado como geometría la que se diseñó utilizando el software CATIA V5 (apartado 2.1). Sin embargo, conviene indicar que el formato recomendado para diseñar gráficos 3D en FG corresponde al formato "**.ac**". Este es el formato nativo de gráficos 3D de la aplicación de diseño *Invis AC3D* (aplicación de pago con versión de prueba gratuita). El software CATIA V5 no tiene capacidad para guardar geometrías en dicho formato. Por este motivo se ha utilizado el software de diseño 3D *Blender* que ha actuado como intermediario para poder transformar la geometría diseñada al formato ac utilizando el formato ".stl". Conviene indicar que, aunque este ha sido el procedimiento que se ha llevado a cabo en el desarrollo de este trabajo, existen otras opciones para conseguir transformar una geometría a formato ac.

Una vez que se dispone de una geometría de aeronave en formato ac, hay que definir los ejes cuerpo del mismo. Para ello, se desplaza y rota la geometría en el programa *Inivis AC3D* para establecer el sistema de coordenadas con centro en el centro aerodinámico de la aeronave y los ejes en la dirección requerida (Figura 64).

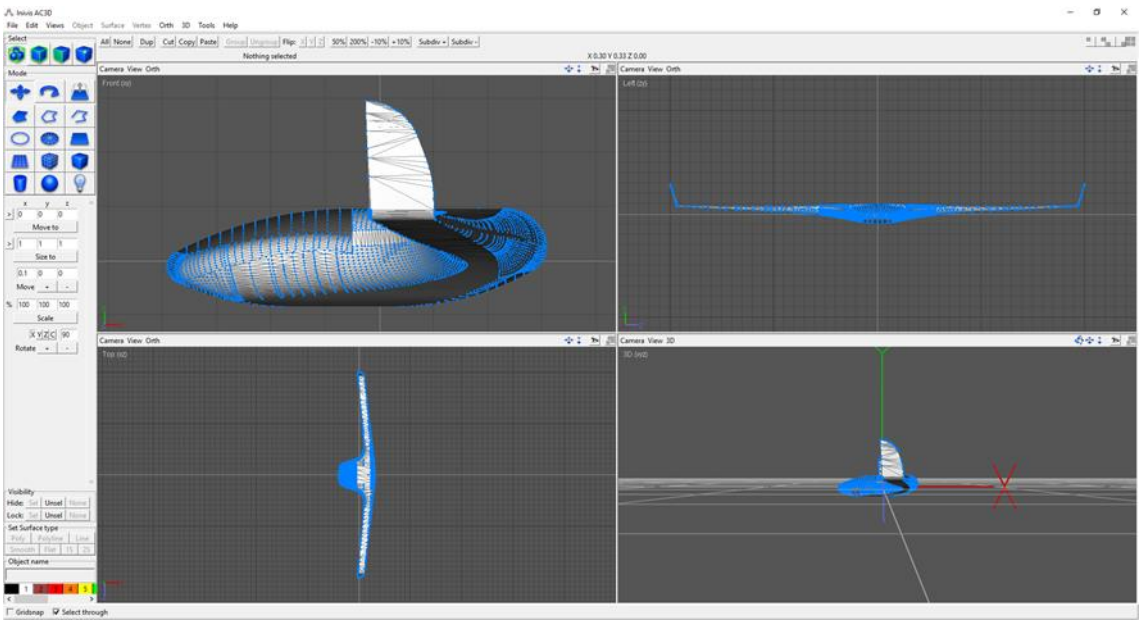

Figura 64: Interfaz del programa Inivis AC3D donde se muestra el sistema de coordenadas elegido

De esta manera, los estados del avión provenientes de Simulink®, que están referenciados a los ejes cuerpo, conseguirán que el movimiento en la visualización sea el adecuado.

Una vez que se dispone de la geometría que se desea utilizar en formato ac y referenciada adecuadamente se puede proceder a introducir dicha geometría en los directorios de instalación de FG. La manera más fácil de realizar este proceso es utilizar el modelo de otra aeronave y sustituir en él la geometría diseñada. Para ello, es necesario introducir el archivo ac dentro del directorio

```
"directorio de instalación"\FlightGear
3.4.0\data\Aircraft\nombre_aeronave\Models
```

Donde "nombre_aeronave" corresponde al nombre que se haya elegido para la aeronave. Este nombre deberá ser el definido mediante el bloque "Generate Run Script" (apartado 3.2.6.1). El directorio de instalación de FG por defecto es `C:\Program Files`

A continuación, se debe indicar que el modelo dinámico que se va a utilizar es el proveniente de Simulink® a través de la red. Para ello hay que modificar una sección de código ubicada en el archivo

```
"Directorio de instalación"\FlightGear
3.4.0\data\Aircraft\nombre_aeronave\nombre_aeronave-set.xml
```

En la sección de código donde se define el modo de modelo dinámico, es decir, la sección de código entre los términos `<flight-model>` y `</flight-model>`, hay que introducir la palabra `network`. La línea quedará así

```
<flight-model>network</flight-model>
```

Por último, es necesario configurar dos aspectos. Primeramente, la posición relativa de la geometría en el entorno 3D de FG. Para ello se pueden modificar algunos parámetros en el código ubicado en el archivo

```
"Directorio de instalación"\FlightGear
3.4.0\data\Aircraft\nombre_aeronave\Models\nombre_aeronave.xml
```

La sección de código a la que se refiere se muestra a continuación.

```
<offsets>
   <x-m> 0.0</x-m>
   <y-m> 0.0</y-m>
   <z-m> 0.0</z-m>
   <pitch-deg>180.0</pitch-deg>
   <roll-deg>0.0</roll-deg>
   <heading-deg>0.0</heading-deg>
</offsets>
```

Por último, se deben definir las vistas de la aeronave. Las vistas definen la posición de la cámara de renderizado y en qué objeto se centra ésta durante la simulación. Es posible crear y configurar vistas en el siguiente archivo de configuración XML.

```
"Directorio de instalación"\FlightGear
3.4.0\data\Aircraft\nombre_aeronave\nombre_aeronave-set.xml
```

Además, FG tiene vistas predeterminadas que, aunque no recomendable, se pueden configurar en el siguiente archivo de configuración.

```
"Directorio de instalación"\FlightGear 3.4.0\data\preferences.xml
```

Para información más detallada acerca de los tipos de vista y su definición de posiciones y distancias se puede visitar el siguiente enlace

http://goo.gl/2RHvsL

En la plataforma se han desarrollado dos vistas. La vista de persecución está activada por defecto y permite ver la geometría del avión y rotar alrededor de su centro aerodinámico. Además siempre está alineada horizontalmente con el horizonte aunque la aeronave rote (Figura 65). La segunda vista es la de cabina, que está fija al avión en todo momento y no permite ver ninguna geometría, ya que geometría de la cabina no se diseñó. Al estar fija a la aeronave, es una buena opción para visualizar con más precisión la dinámica del avión (Figura 66). Se puede cambiar de la vista en persecución a la vista en cabina pulsando el gatillo trasero del joystick que se esté utilizando, en el caso de que disponga de él.

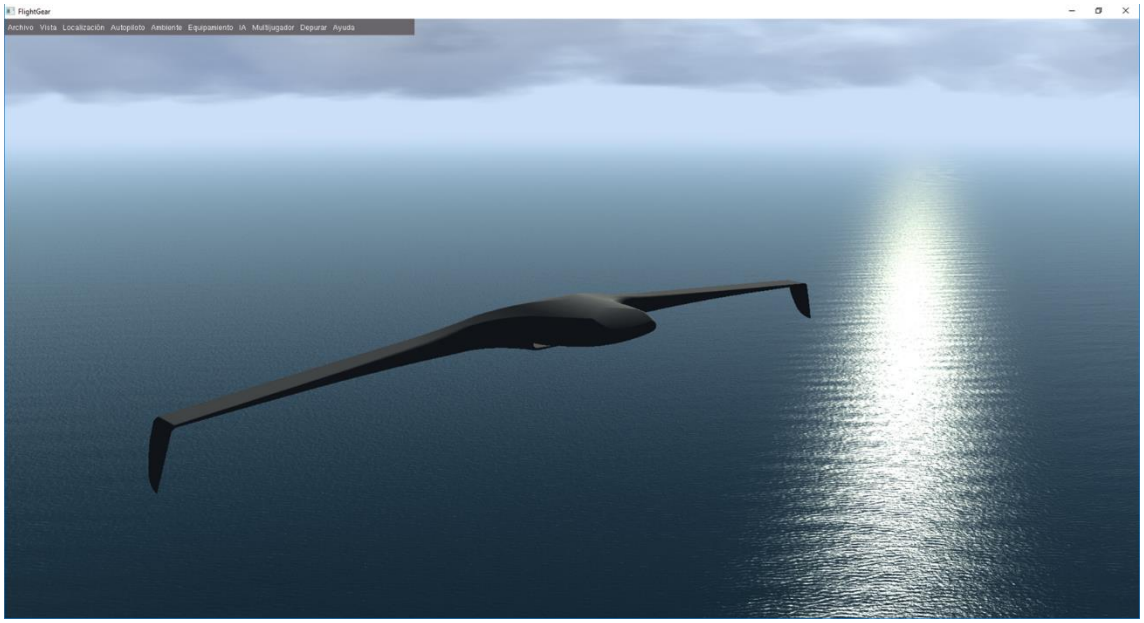

Figura 65: Motor gráfico FlightGear con vista de persecución

Figura 66: Motor gráfico FlightGear con vista de cabina

Con el fin de hacer más accesible el proceso de ejecución del script de inicialización creado mediante el bloque "Generate Run Script" se ha creado un acceso directo en el entorno de MATLAB®. Se detallará la configuración de este acceso directo en el Anexo B.

Capítulo 5

Interfaz gráfica mediante GUIDE de MATLAB

5.1 Estructura general de la interfaz

Con el objetivo de facilitar el manejo del simulador se ha desarrollado una interfaz gráfica que tiene la capacidad de interactuar con el simulador de diversas maneras. En la Figura 67 se muestra una vista general de la interfaz cuando se inicializa. Se ha desarrollado un comando que se encarga de que la interfaz gráfica se abra automáticamente al abrir el simulador, el procedimiento que se ha utilizado para esta tarea será detallado en el Anexo B.

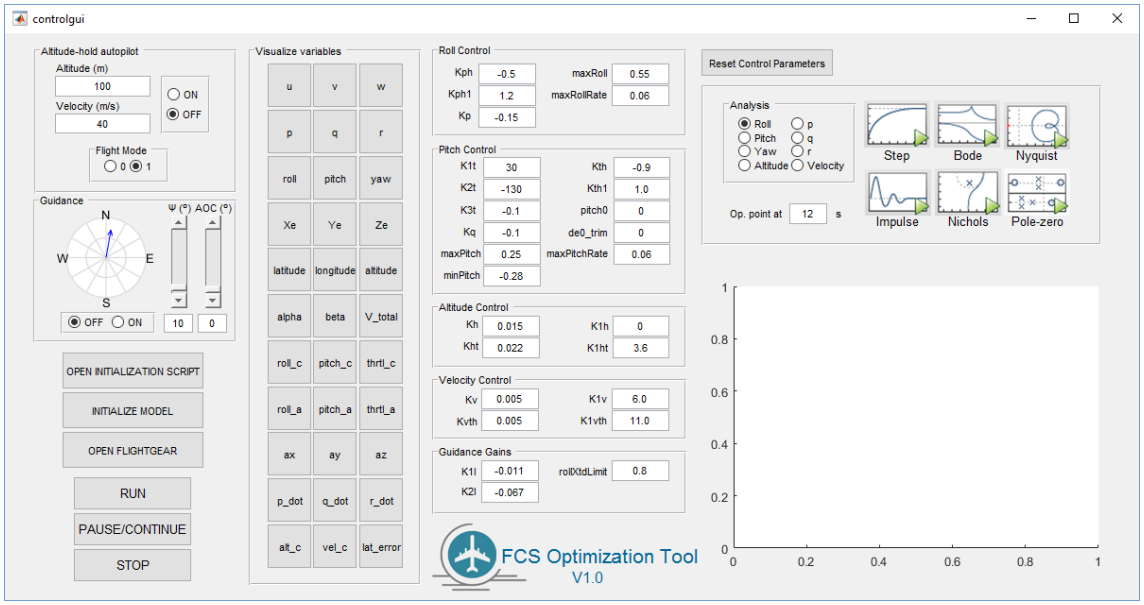

Figura 67: Estructura de la interfaz gráfica

Esta interfaz se ha desarrollado utilizando la aplicación GUIDE (entorno de desarrollo de interfaces gráficas de usuario) de MATLAB®. Esta aplicación proporciona herramientas para diseñar interfaces de usuario personalizadas. Para más información sobre GUIDE se puede consultar la referencia [41].

Las interfaces gráficas que se desarrollan con GUIDE utilizan un script que contiene funciones. Al interactuar con la interfaz, el programa ejecuta directamente la función del script correspondiente. La estructura del script junto con las funciones se generan automáticamente a medida que se va construyendo la interfaz.

En esta interfaz se han utilizado funciones "Callback", que son ejecutadas tras hacer click en el elemento correspondiente. Otras funciones posibles (ButtonDownFcn, KeyPressFcn, …) no se han utilizado en este caso.

Para facilitar el proceso de inicialización al usuario, se ha implementado una función en el simulador que permite que al abrir este último se abra automáticamente la interfaz gráfica. Esto se consigue mediante una línea de código, que se ha introducido en el menú "Model Properties" del simulador (Figura 68). Todo el código incluido en "PostLoadFcn" se ejecuta justo después del proceso de carga al abrirlo.

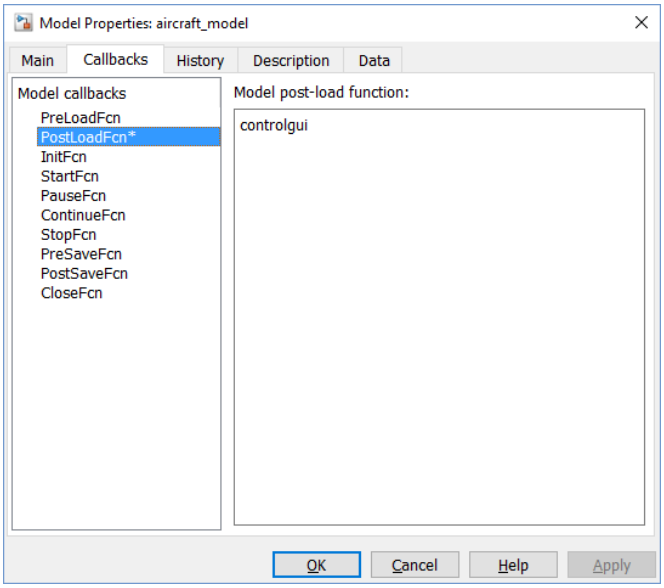

Figura 68: Menú de opciones "Model Properties"

Para exponer en detalle todas las características de la interfaz gráfica se ha dividido ésta en 6 secciones. Las distintas secciones se muestran en la figura siguiente (Figura 69).

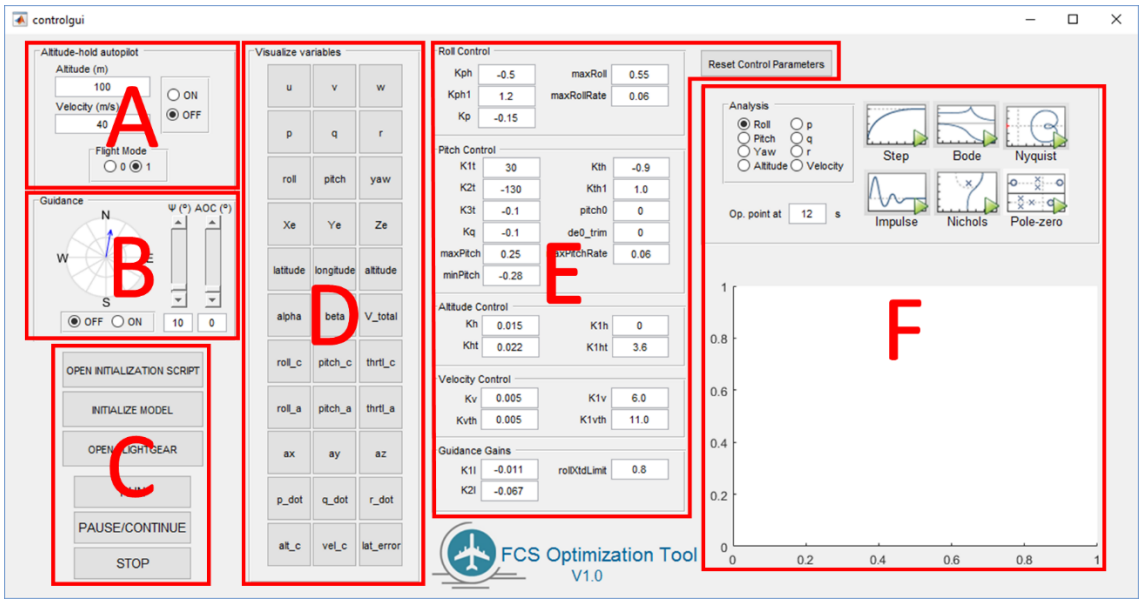

Figura 69: Secciones de la interfaz gráfica

5.2 Desarrollo de las secciones de la interfaz

A continuación se exponen en detalle las distintas secciones de la interfaz gráfica.

5.2.1 Sección A: Autopiloto de altitud y velocidad

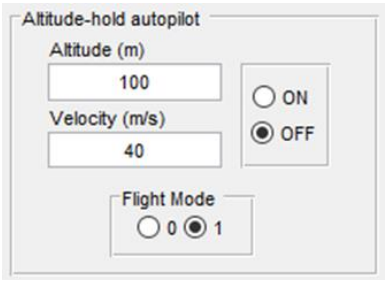

Figura 70: Control del autopiloto de altitud y velocidad

Esta sección de la interfaz controla el autopiloto de altitud y velocidad que forma parte del sistema de control de vuelo. Como se mencionó en el apartado 3.2.2.2 dicho autopiloto puede funcionar en dos modos de vuelo distintos (modos 0 y 1).

La activación del autopiloto y el cambio de modo deben ser instantáneos en el simulador y deben permitir que la simulación sea continua. La función “Variant Subsystem” que se explicó en el apartado 3.1 no es adecuada para esta tarea ya que el cambio de un subsistema activo a otro requiere una parada de la simulación. Por tanto, para cambiar la estructura de una sección del modelo durante la simulación se ha recurrido a los bloques “Manual Switch”. Estos bloques permiten cambiar la estructura del subsistema durante la simulación de forma continua. Además pueden activarse mediante código. Como inconveniente, conviene mencionar que

utilizar estos bloques conlleva que durante la simulación se esté ejecutando toda la estructura, sin importar que una parte de ella no se utilice debido a la posición del interruptor.

En la Figura 71 y Figura 72 se presentan de nuevo los subsistemas “Pitch Control” y “Throttle Control” que ya fueron presentados en la Figura 25 y Figura 29. En este caso se señalan utilizando rectángulos de color rojo o azul los bloques “Manual Switch” que se han utilizado en la estructura para realizar la activación de los autopilotos y los modos de vuelo.

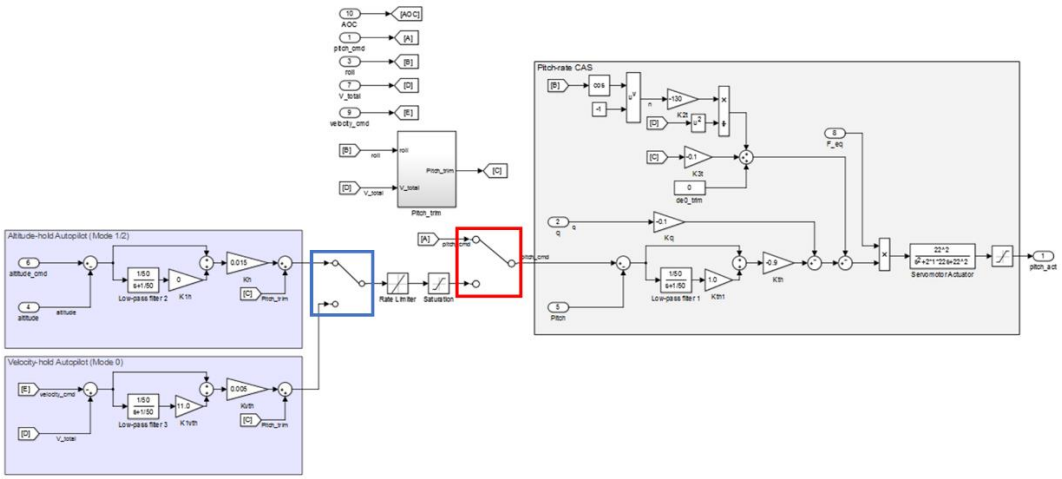

Figura 71: Subsistema "Pitch Control"

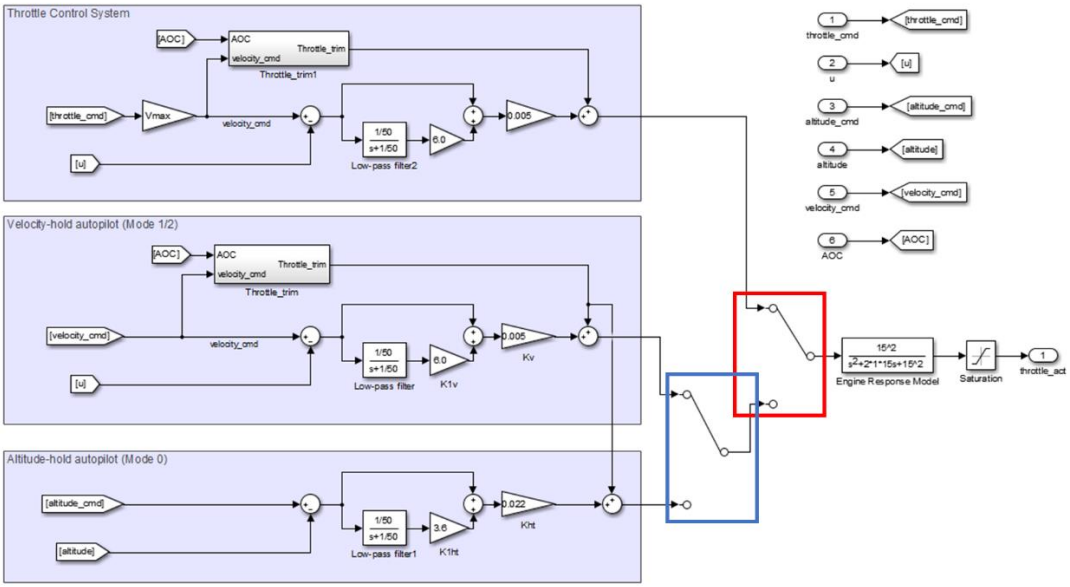

Figura 72: Subsistema "Throttle Control"

Los bloques “Manual Switch” señalados con los rectángulos rojos permiten activar o desactivar el autopiloto, mientras que los señalados con los rectángulos azules permiten cambiar de modo de vuelo (Modos 0 ó 1). Al actuar sobre la interfaz, el cambio ocurre simultáneamente en los dos subsistemas (Pitch Control y Throttle Control).

Para comandar la altitud y velocidad deseadas, se introducen los valores elegidos en los cuadros de edición (Edit Box) de la sección A de la interfaz. Al modificar estos valores automáticamente se modifican los valores de los bloques "Constant" llamados "altitude_cmd" y "velocity_cmd" del subsistema Joystick perteneciente al bloque "Commands" (Figura 12).

Como se ha comentado previamente el código utilizado se ha desarrollado en lenguaje MATLAB®. En la sección A se ha utilizado código para dos tareas distintas:

(1) En primer lugar para activar los "Manual Switches"

```
set_param([bdroot '/FCS/Controller/Pitch Control/Manual
Switch'],'sw','0')
```

Esta línea de código contiene la ruta del bloque "Manual Switch" y permite el cambio del interruptor a la posición especificada (0 ó 1).

(2) Para actualizar los valores de los bloques constantes. El código tiene dos partes: En la primera se obtiene el valor que se ha introducido en el "Edit Box".

```
value = get(hObject,'String');
```

Y en la segunda parte se actualiza el valor del bloque "Constant" con el valor obtenido en la primera parte.

```
set_param([bdroot
'/Commands/Joystick/altitude_cmd'],'Value',value)
```

Es recomendable añadir el siguiente código al final de las funciones que afectan a la estructura del modelo mientras que se está ejecutando. Sirve para actualizar el modelo si se está ejecutando.

```
status = get_param(bdroot,'simulationstatus');
if strcmp(status,'running')
    set_param(bdroot, 'SimulationCommand', 'Update')
end
```

5.2.2 Sección B: Sistema de guiado

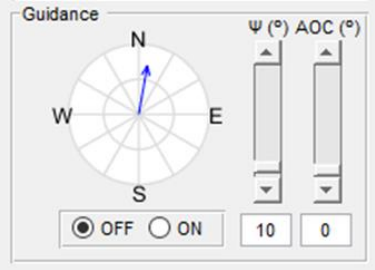

Figura 73: Control del sistema de guiado

Esta sección de la interfaz controla el sistema de guiado que se ha desarrollado en el bloque FCS (Sistema de Control de Vuelo). Esta sección permite controlar los comandos de ángulo de guiñada y de ángulo de subida (Angle of Climb) de la aeronave. Los botones ON y OFF permiten activar o desactivar el sistema de guiado y funcionan controlando un "Manual Switch" que se encuentra en el subsistema "Roll Control" (Figura 20), de manera similar a lo explicado en los botones de la sección A. Las cajas de edición (Edit Box) controlan los valores de bloques "Constant", en el caso del ángulo de guiñada el bloque llamado "heading" dentro del subsistema "Roll Control". Nuevamente esta función es similar a la de los bloques "Edit Box" de la sección anterior. En esta sección se ha desarrollado además una brújula (utilizando la función compass [42]) y dos deslizadores (sliders) interactivos que muestran los valores introducidos en los "Edit Box". No está todavía operativo el control del ángulo de subida del sistema de guiado.

No se presentan explícitamente las líneas de código de esta sección porque son muy similares a las de la sección anterior, a excepción del código de creación de la brújula, que se expone a continuación.

```
angle = 10;
a = sqrt(2)*sin(deg2rad(angle));
b = sqrt(2)*cos(deg2rad(angle));
axes(handles.axes2); %set the current axes to axes2
compass(a,b,'-b')
set(findall(handles.axes2, 'String', '0'),'String', 'E'); % Alter the
angular labels
set(findall(handles.axes2, 'String', '30'),'String', ' ');
set(findall(handles.axes2, 'String', '60'),'String', ' ');
set(findall(handles.axes2, 'String', '90'),'String', 'N');
set(findall(handles.axes2, 'String', '120'),'String', ' ');
set(findall(handles.axes2, 'String', '150'),'String', ' ');
set(findall(handles.axes2, 'String', '180'),'String', 'W');
set(findall(handles.axes2, 'String', '210'),'String', ' ');
set(findall(handles.axes2, 'String', '240'),'String', ' ');
set(findall(handles.axes2, 'String', '270'),'String', 'S');
set(findall(handles.axes2, 'String', '300'),'String', ' ');
set(findall(handles.axes2, 'String', '330'),'String', ' ');
set(findall(handles.axes2, 'String', '  0.5'),'String', ' '); % Alter
the radial labels
set(findall(handles.axes2, 'String', '  1'),'String', ' ');
set(findall(handles.axes2, 'String', '  1.5'),'String', ' ');
set(findall(handles.axes2, 'String', '  2'),'String', ' ');
```

Para actualizar la dirección de la flecha direccional de la brújula es suficiente con definir la variable "angle" con el valor en grados que se desee y volver a ejecutar el código.

5.2.3 Sección C: Controles de simulación

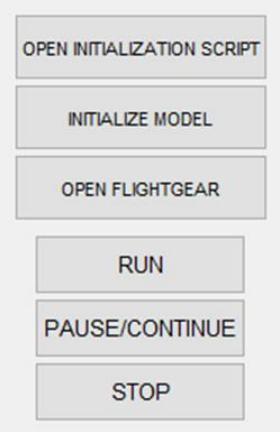

Figura 74: Controles de simulación

En esta sección se encuentran los controles de la simulación, además de algunas funciones que se han introducido con el fin de facilitar el proceso de inicialización de la plataforma de simulación.

El botón "Open initialization script" tiene la función de abrir el script de inicialización "startVars.m". Este script se utiliza de forma muy frecuente durante el uso normal de la plataforma por lo que el botón supone un ahorro de tiempo. El botón "Initialize model" ejecuta el script "startVars.m" para inicializar el simulador. "Open FlightGear" es un botón que permite abrir el motor gráfico FlightGear inicializado con las características y condiciones requeridas (geometría de aeronave, estados iniciales…).

A continuación los tres botones de la zona inferior corresponden a los controles de estado de la simulación: El botón "Run" permite el inicio de la simulación, "Pause/continue" permite pausar o continuar la simulación y "Stop" permite detener la simulación.

Los códigos de los seis botones citados anteriormente se presentan a continuación.

(1)
```
open('startVars.m')
```

(2)
```
startVars
```

(3)
```
system('runfg &')
```

(4)
```
set_param('aircraft_model','SimulationCommand','start');
```

(5)
```
if strcmp(status,'paused')
  set_param('aircraft_model','SimulationCommand','continue');
else
  set_param('aircraft_model','SimulationCommand','pause');
end
```

(6)
```
set_param('aircraft_model','SimulationCommand','stop');
```

5.2.4 Sección D: Visualización gráfica de estados

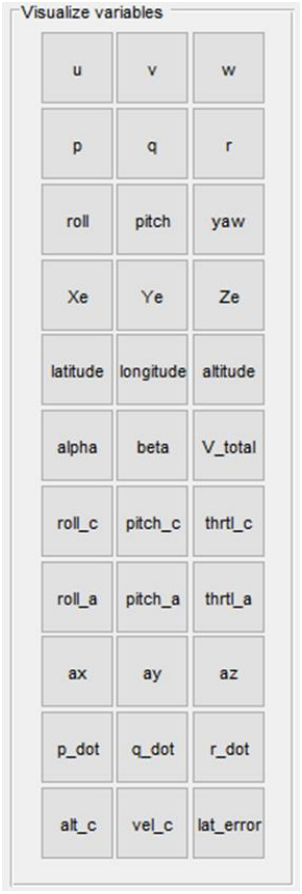

Figura 75: Control de visualización de variables

Esta sección permite abrir la visualización gráfica del simulador (Scopes) de las variables de estado que se desee. Para ello contiene numerosos botones que representan a las variables. Internamente, el código de todos los botones es similar. Los bloques “Scope” se encuentran en el subsistema “FlightGear”, dentro del bloque “Visualization”.

El código que se muestra a continuación como ejemplo corresponde al del botón u.

```
open_system('aircraft_model/Visualization/FlightGear/u_scope')
```

5.2.5 Sección E: Parámetros de control de vuelo

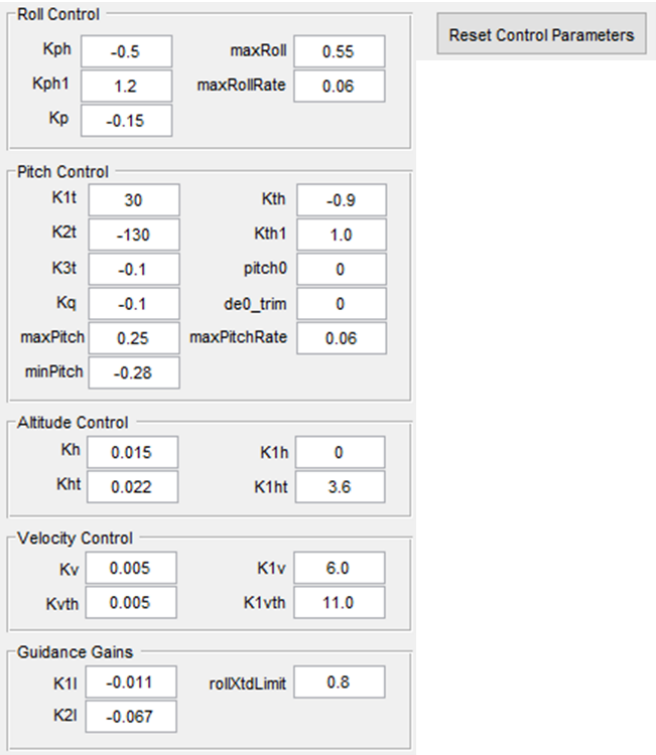

Figura 76: Control de los parámetros del sistema FCS

La sección E de la interfaz gráfica se ocupa del control de las ganancias del sistema de control de la aeronave. El método que utiliza consiste en actualizar mediante código los valores de bloques de ganancia (gain). Estos bloques se encuentran dentro de los distintos sistemas de control (Roll Control, Pitch Control, Yaw Control, Throttle Control).

El botón "Reset control parameters" restablece los valores de todas las casillas de edición al valor predeterminado. Esta función es útil en el caso de que el usuario olvide el valor predeterminado de las ganancias durante el proceso de optimización. Para cambiar los valores predeterminados es necesario establecerlo en el código.

Para almacenar la información introducida en la casilla de edición se utiliza el siguiente código.

```
value11 = get(handles.roll_1,'String');
```

El texto subrayado en verde corresponde al nombre de la casilla de edición correspondiente en la interfaz gráfica. Para actualizar el valor del bloque de ganancia requerido se utiliza el siguiente código.

```
set_param([bdroot '/FCS/Controller/Roll
Control/Kph'],'Gain',value11)
```

En el caso de los valores limitadores de los ángulos y velocidades angulares del avión se puede realizar el mismo procedimiento pero aplicando los valores a otras propiedades de los bloques

"Saturation" y "Rate Limiter". Estos bloques sirven para limitar los ángulos y velocidades angulares de sus inputs. El código sería entonces el siguiente.

```
set_param([bdroot '/FCS/Controller/Roll
Control/Saturation2'],'UpperLimit',value11)
set_param([bdroot '/FCS/Controller/Roll
Control/Saturation2'],'LowerLimit',value11_negative)
set_param([bdroot '/FCS/Controller/Roll Control/Rate
Limiter'],'risingSlewLimit',value11)
set_param([bdroot '/FCS/Controller/Roll Control/Rate
Limiter'],'fallingSlewLimit',value11_negative)
```

5.2.6 Sección F: Análisis del sistema

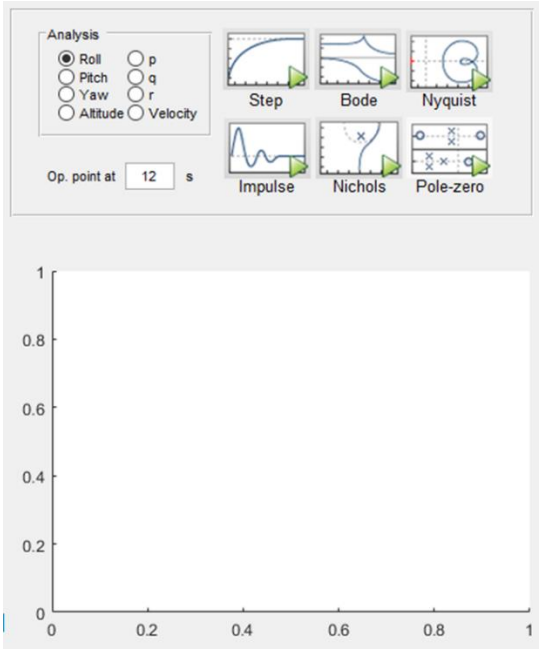

Figura 77: Sección de control de las capacidades analíticas

Esta sección de la interfaz es la que se encarga de llevar a cabo las funciones de análisis de sistemas dinámicos de la plataforma de simulación. El método que utiliza la plataforma para llevar a cabo el análisis consiste en realizar un proceso de linealización del sistema no lineal en una condición de vuelo determinada. La sección F se puede dividir en cuatro subsecciones (Figura 78).

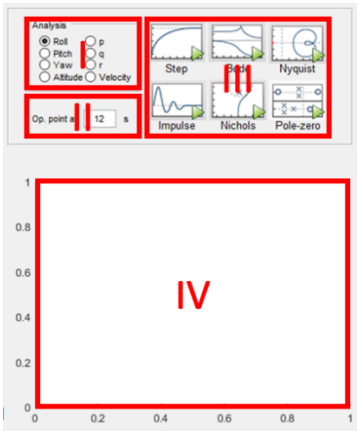

Figura 78: Secciones de análisis

La **subsección I** contiene una serie de botones de opción que permiten seleccionar la estructura de control que se desea analizar (roll, pitch, yaw, p, q, r, altitud o velocidad). Más adelante se explicará en detalle el método que se ha utilizado para tratar los bucles individualmente durante el análisis.

La **subsección II** permite establecer un punto operativo para la linealización. En este caso, el parámetro que se introduce en la casilla de edición corresponde al tiempo en segundos que es necesario simular el sistema a partir de su condición inicial para obtener la condición de vuelo en la que se realizará la linealización. Es recomendable establecer un tiempo que permita que el sistema alcance una condición aproximadamente estacionaria para llevar a cabo un proceso de linealización fiable. Para ello es necesario simular el sistema previamente para observar de qué orden es este tiempo con las condiciones del sistema presentes.

En la **subsección III** se encuentran los botones que permiten llevar a cabo el proceso de linealización y análisis del modelo lineal obtenido. Tras presionar un botón de esta sección se ejecutan sucesivamente tres procesos:

(1) **Simulación** del modelo dinámico desde su condición inicial hasta el tiempo establecido en la subsección II, para obtener la condición de vuelo en la que se realizará la linearización.
(2) **Linealización** del sistema dinámico en la condición de vuelo obtenida en el paso 1.
(3) Cálculo de un **diagrama** (el tipo de diagrama cambia en función del botón que haya sido presionado) a partir del modelo lineal obtenido en el paso 2. La plataforma permite obtener seis tipos de diagrama diferentes:
    a. Diagrama de respuesta a escalón.
    b. Diagrama respuesta a impulso.
    c. Diagrama de Bode.
    d. Diagrama de Nichols.
    e. Diagrama de Nyquist.
    f. Diagrama de polos y zeros.

La **subsección IV** consiste en un espacio de "Axis" (ejes). Este espacio está destinado a la representación de gráficas, diagramas, imágenes… En este caso corresponde a la zona de la interfaz gráfica donde aparecerán los diagramas creados al pulsar los botones de la subsección III.

A continuación, se explicará brevemente en qué consisten los tipos de diagramas que se mencionaron en el desarrollo de la subsección III. Todos ellos son herramientas muy útiles para el análisis de sistemas (estabilidad, robustez,…). Este trabajo no abarcará la manera en que se analizan los distintos tipos de diagramas, para más información consúltese la referencia [43].

- Diagrama de **respuesta a escalón** (Step): Este diagrama muestra la respuesta del sistema a una señal de entrada (input) de tipo escalón de Heaviside. La función escalón de Heaviside es una función discontinua cuyo valor es cero para argumentos (en este caso tiempos) negativos y uno para argumentos positivos. En ingeniería electrónica y teoría de control, la respuesta a escalón representa el comportamiento de los outputs de un sistema cuando sus inputs cambian de cero a uno en un periodo de tiempo muy corto.

- Diagrama de **respuesta a impulso** (Impulse): Este diagrama muestra la respuesta del sistema a una señal de entrada (input) de tipo Delta de Dirac, si el sistema está modelado en tiempo continuo, o Delta de Kronecker, si el sistema está modelado en tiempo discreto. La función Delta de Dirac representa el caso límite de un pulso dado en un intervalo muy corto de tiempo manteniendo su área o integral (aun teniendo una amplitud máxima infinita). La función Delta de Kronecker no tiene una amplitud máxima infinita, sino que vale uno durante una unidad del intervalo de tiempo de discretización, y cero en el resto. Al igual que en el caso de la función de Heaviside, la función Delta de Dirac no se puede dar en un sistema real, sin embargo supone una idealización muy útil. En la teoría de análisis de Fourier un impulso como el de esta función está compuesto por porciones idénticas de todas las frecuencias de excitación posibles.

- Diagrama de **Bode**: Un diagrama de Bode representa la respuesta en frecuencia de un sistema. Normalmente, es una combinación de un diagrama de amplitud, donde se representa la amplitud (en decibelios) de la respuesta en frecuencia, y un diagrama de fase, en el que se representa el desfase. Ambas magnitudes se grafican en función del logaritmo de la frecuencia.

- Diagrama de **Nichols**: Dada una función de transferencia $G(s) = Y(s)/X(s)$ la función de transferencia en lazo cerrado se define como $M(s) = G(s)/(1 + G(s))$. El diagrama de Nichols es una representación de $20\log_{10}(|G(s)|)$ en función de $\arg(G(s))$, donde este último término se refiere a la fase de la respuesta del sistema.

- Diagrama de **Nyquist**: La representación se realiza en los ejes cardinales. La parte real de la función de transferencia se representa en el eje de abscisas y la parte imaginaria en el eje de ordenadas. La frecuencia se recorre como un parámetro, por lo que a cada frecuencia le corresponde un punto de la gráfica. Alternativamente, en coordenadas polares, la ganancia de la función de transferencia se representa en la coordenada

radial, mientras que la fase de la función de transferencia se representa en la coordenada angular.

- Diagrama de **polos y zeros:** Este diagrama representa en un sistema de coordenadas cartesianas siendo el eje de abscisas el eje real y el de ordenadas el eje imaginario los polos y ceros del sistema. Existe otro tipo de gráfica que surge al obtener el lugar geométrico de los polos y ceros a medida que se varía una ganancia del sistema en un determinado intervalo (lugar de las raíces).

A continuación se explicará más detalladamente el método que se ha utilizado para tratar los bucles individualmente durante el análisis.

La función que se ha utilizado para linealizar el sistema, y por tanto, el núcleo de esta sección es la función de MATLAB® "linearize". Esta función ejecuta una aproximación lineal de un modelo o bloque de Simulink®. La función permite ser ejecutada con diversas combinaciones de inputs. En este caso, se ha elegido utilizar la siguiente estructura.

```
linsys = linearize(sys,op,io)
```

Esta línea de código realiza la linealización del modelo no lineal de Simulink® de nombre el string definido en "sys" delimitado por los puntos input/output de linealización definidos en "io". La linealización se realiza en la condición impuesta por "op", en nuestro caso un valor de tiempo en segundos que, como se explicó en el desarrollo de la subsección II, corresponde al tiempo que es necesario simular el sistema a partir de su condición inicial para obtener la condición de vuelo en la que se realizará la linealización [44].

En cuanto a los puntos de input/output para el análisis en los distintos bucles, se ha seguido el siguiente procedimiento.

(1) Inicialmente se han definido todos los puntos de entrada y salida de la linealización dentro del simulador (Figura 79, Figura 80 y Figura 81).
(2) A continuación se desactivan todos los puntos de entrada y salida. En estado inactivo los puntos siguen estando ahí, pero no se tienen en cuenta en el proceso de linealización a no ser que se vuelvan a activar.
(3) Por último, en función de qué bucle se ha elegido analizar en la interfaz gráfica, se activa la combinación de puntos de entrada/salida deseados.

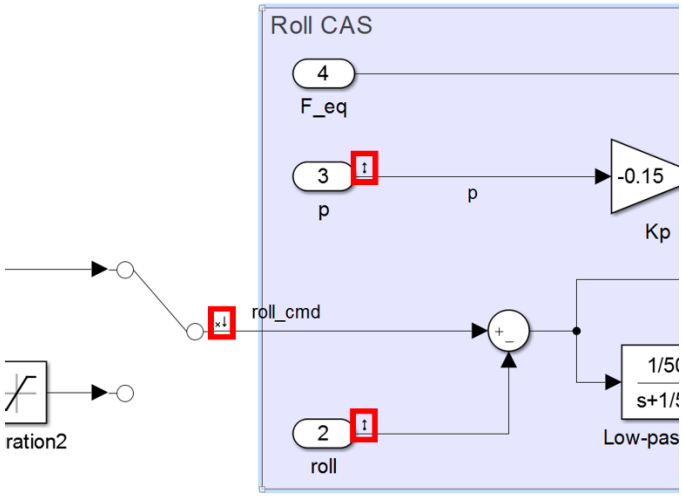

Figura 79: Inputs/Outputs en "Roll Control"

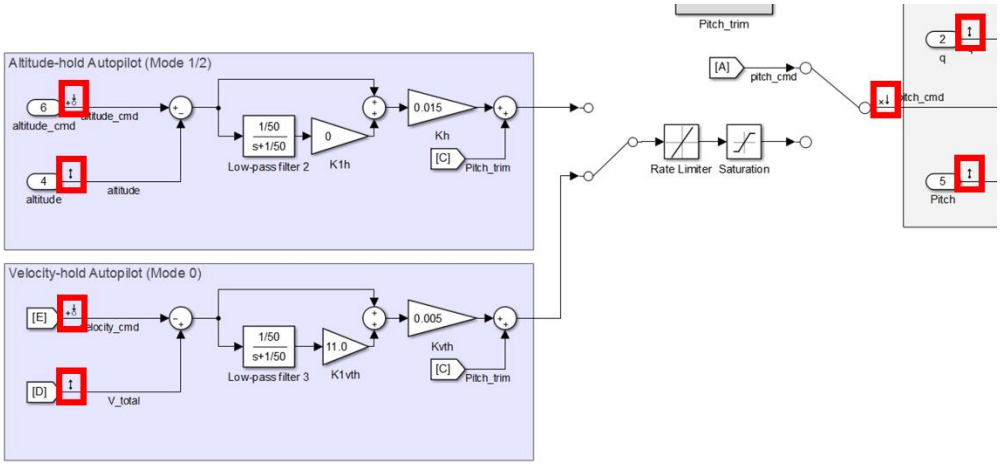

Figura 80: Inputs/Outputs en "Pitch Control"

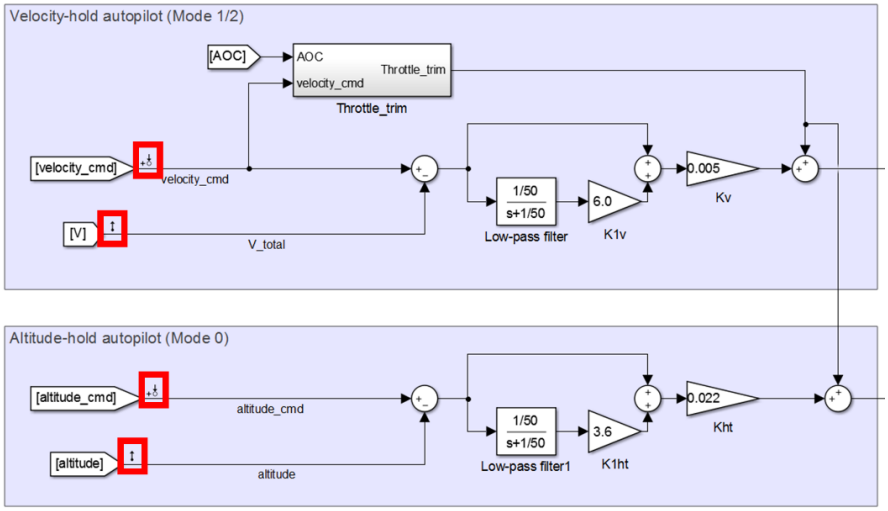

Figura 81: Inputs/Outputs en "Throttle Control"

Se han utilizado tres tipos distintos de puntos de entrada/salida.

(1) "Open-loop Input" . Este tipo de entrada no permite que la señal que le precede siga enviando información. Es decir, "corta" el circuito por ese punto. Se ha seleccionado este tipo de input para los comandos de pitch y roll porque en el proceso de linealización se llega a la condición a linealizar utilizando los autopilotos de velocidad y altitud del sistema de control de vuelo con el objetivo de que la aeronave se encuentre en una condición estacionaria. Sin embargo, el análisis de esos bucles es necesario realizarlo sin la función de los autopilotos que preceden al punto de entrada. Como no se pueden desactivar los autopilotos durante la simulación del proceso de linearización, se ha elegido cortar el circuito en ese punto impidiendo por tanto que los autopilotos actúen de ninguna manera.

(2) "Input perturbation". Este tipo de entrada no corta el circuito. Solamente introduce la perturbación necesaria para linealizar sumándola a la señal que pasa por ese punto. Se ha elegido utilizar este input en los comandos de los autopiolotos para que éstos no dejen de funcionar durante el proceso de linealización.

(3) "Output measurement" . Este tipo de salida no introduce ninguna perturbación ni corta el circuito. Su única función es medir la señal en ese punto. Ha sido la elección para medir las variables de estado de salida.

El inicio del proceso de linealización tiene como requerimiento que el modelo se encuentre parado, no pausado ni en estado de ejecución. Para asegurar que este requerimiento siempre se cumpla se han programado una serie de "warnings" o mensajes de alerta que aparecen en la pantalla si el usuario intenta realizar un análisis mientras que el modelo se esté ejecutando o esté pausado (Figura 82).

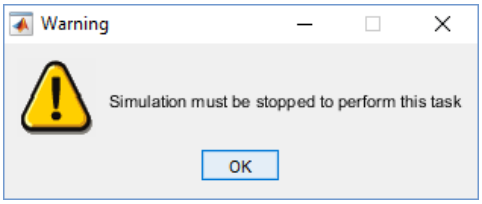

Figura 82: Ventana warning

También se ha programado un mensaje de alerta que aparece si el usuario intenta activar el análisis en el bucle de yaw ó r ya que, como se indicó en el apartado 1.5, la aeronave seleccionada (aeronave X) no posee superficies de control específicamente diseñadas para controlar el yaw (Figura 83). Su ángulo de guiñada se controla únicamente utilizando los alerones.

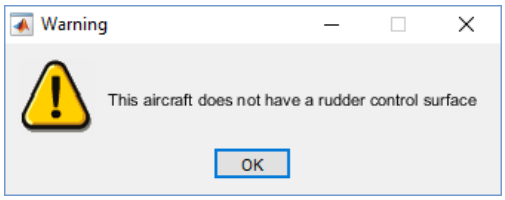

Figura 83: Ventana warning

El proceso completo de análisis no es instantáneo. La principal causa es que es necesario simular el sistema hasta la condición del tiempo objetivo. Para reducir este tiempo, antes de llevar a cabo el proceso de simulación se desactiva el bloque "Simulation Pace", del que se habló en el apartado 2.2.6.1, cuya función es controlar el ritmo de la simulación para adaptar el ritmo de avance de tiempo de la simulación al real. De esta manera, la simulación hasta el tiempo objetivo es más rápida y por tanto el proceso completo. Una vez que finaliza el proceso de análisis se vuelve a activar dicho bloque.

Además, para indicar al usuario que el proceso se está llevando a cabo, se ha programado una ruleta dinámica de carga que se muestra durante el tiempo de ejecución (Figura 84).

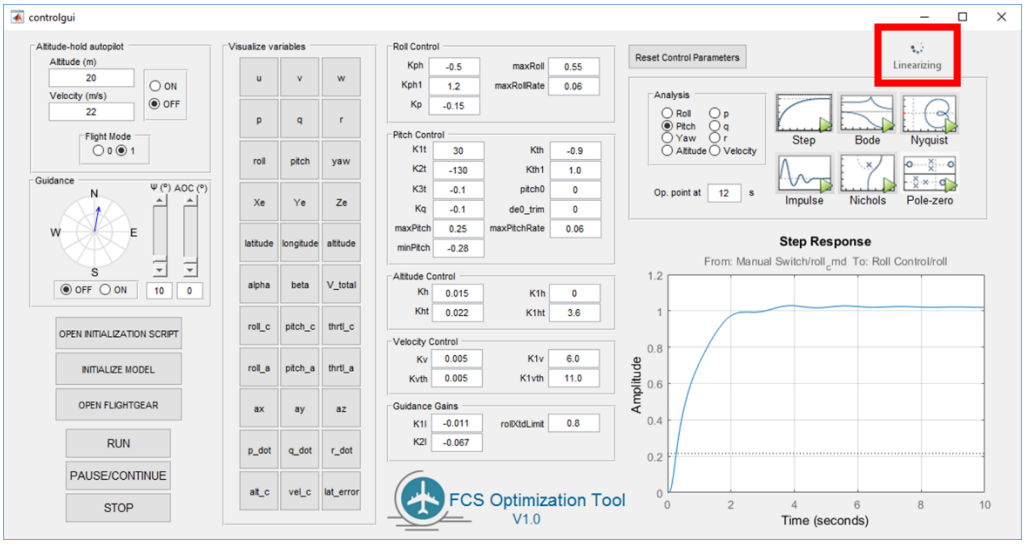

Figura 84: Ubicación de la ruleta dinámica de carga

A continuación se muestra el código que se ha utilizado en las funciones correspondientes a los botones de análisis. El código posee una estructura común que se usa en todos los botones, además de una estructura particular característica en cada uno de ellos. Se ha señalado en color amarillo la zona donde hay que insertar el código específico requerido dentro del código común. Nótese que al no haber suficiente espacio algunas líneas de código se muestran en dos líneas en este documento.

Estructura del código **común** para el proceso de análisis:

```
axes(handles.axes); %set the current axes to axes
status = get_param(bdroot,'simulationstatus');
if strcmp(status,'stopped')
```

```
    iconsClassName =
'com.mathworks.widgets.BusyAffordance$AffordanceSize';
    iconsSizeEnums = javaMethod('values',iconsClassName);
    SIZE_32x32 = iconsSizeEnums(2);  % (1) = 16x16,  (2) = 32x32
    jObj = com.mathworks.widgets.BusyAffordance(SIZE_32x32,
'Linearizing');  % icon, label
    jObj.setPaintsWhenStopped(false);  % default = false
    jObj.useWhiteDots(false);         % default = false (true is good
for dark backgrounds)
    javacomponent(jObj.getComponent, [1050,550,60,60], gcf);
    jObj.start;
    warning('off','all') % Turn warnings off
    blockName = 'aircraft_model/Visualization/FlightGear/Simulation
Pace'; % Root of "Set Pace" block
    set_param(blockName,'Commented','on') % Deactivate "Set Pace" block
to make linearization process faster
    mode_0_on = get(handles.mode_0_button,'Value');
    ah_on = get(handles.ah_on,'Value');
    roll_radiobutton = get(handles.roll_an,'Value');
    pitch_radiobutton = get(handles.pitch_an,'Value');
    yaw_radiobutton = get(handles.yaw_an,'Value');
    altitude_radiobutton = get(handles.alt_an,'Value');
    velocity_radiobutton = get(handles.velocity_an, 'Value');
    p_radiobutton = get(handles.p_an,'Value');
    q_radiobutton = get(handles.q_an,'Value');
    r_radiobutton = get(handles.r_an,'Value');
    op = str2num(get(handles.op_point,'String')); % Operating point time
(s)
    io = getlinio(bdroot); %get all I\O defined in the model
    set(io,'Active','off'); %deactivate all I/O
    if roll_radiobutton == 1
        set(io(4),'Active','on'); %activate I/O
        set(io(12),'Active','on'); %activate I/O
    elseif pitch_radiobutton == 1
        set(io(2),'Active','on'); %activate I/O
        set(io(8),'Active','on'); %activate I/O
    elseif yaw_radiobutton == 1
        warningstring = 'This aircraft does not have a rudder control
surface';
        dlgname = 'Warning';
        warndlg(warningstring,dlgname)
        jObj.stop;
    elseif altitude_radiobutton == 1
        if mode_0_on == 1 && ah_on == 1
            set(io(6),'Active','on'); %activate I/O
            set(io(14),'Active','on'); %activate I/O
        elseif mode_0_on == 0 && ah_on == 1
            set(io(3),'Active','on'); %activate I/O
            set(io(9),'Active','on'); %activate I/O
        else
            warningstring = 'Altitude-hold autopilot must be activated
to perform this task';
            dlgname = 'Warning';
            warndlg(warningstring,dlgname)
            jObj.stop;
        end
    elseif velocity_radiobutton == 1
        if mode_0_on == 1 && ah_on == 1
            set(io(1),'Active','on'); %activate I/O
            set(io(7),'Active','on'); %activate I/O
        elseif mode_0_on == 0 && ah_on == 1
            set(io(5),'Active','on'); %activate I/O
            set(io(13),'Active','on'); %activate I/O
        else
            warningstring = 'Altitude-hold autopilot must be activated
to perform this task';
            dlgname = 'Warning';
            warndlg(warningstring,dlgname)
```

```
                jObj.stop;
            end
        elseif p_radiobutton == 1
            set(io(4),'Active','on'); %activate I/O
            set(io(11),'Active','on'); %activate I/O
        elseif q_radiobutton == 1
            set(io(2),'Active','on'); %activate I/O
            set(io(10),'Active','on'); %activate I/O
        elseif r_radiobutton == 1
            warningstring = 'This aircraft does not have a rudder control
surface';
            dlgname = 'Warning';
            warndlg(warningstring,dlgname)
            jObj.stop;
        end
        OL = linearize(bdroot,io,op);
        ESTRUCTURA ESPECÍFICA DE CADA DIAGRAMA
else
        warningstring = 'Simulation must be stopped to perform this task';
        dlgname = 'Warning';
        warndlg(warningstring,dlgname)
end
set_param(blockName,'Commented','off') % Activate "Set Pace" block
warning('on','all') % Turn warnings on
jObj.stop;
jObj.setBusyText('Completed');
```

Estructura del código **específica** de cada diagrama para el proceso de análisis:

1. Diagrama de **respuesta a escalón**:

```
step(OL,10)
grid on
```

2. Diagrama de **respuesta a impulso**:

```
impulse(OL,10)
grid on
```

3. Diagrama de **Bode**:

```
bodeplot(OL, {0.1, 50}, 'b');
grid on
[GmOL,PmOL,WcgOL,WcpOL] = margin(OL);
fbOL = bandwidth(OL);
title(sprintf('Bode Diagram  Gm: %.2f dB  Pm: %.2f °  BW: %.2f
rad/s',GmOL,PmOL,fbOL));
```

4. Diagrama de **Nichols**:

```
nichols(OL)
grid on
```

5. Diagrama de **Nyquist**:

```
nyquist(OL)
grid on
```

6. Diagrama de **polos y ceros**:

```
pzmap(OL)
grid on
```

Capítulo 6

Resultados

En este capítulo se muestra el comportamiento dinámico que se ha obtenido con la plataforma de simulación aplicada a la aeronave X. En un primer apartado se presentan los resultados del modo de vuelo manual de la aeronave. Seguidamente, se muestran los resultados obtenidos utilizando los autopilotos de altitud y velocidad. A continuación se realiza lo mismo con el sistema de guiado. Por último se demuestra la capacidad de análisis de la plataforma de simulación.

6.1 Vuelo manual

Con el fin de mostrar el comportamiento de la aeronave en modo manual se han realizado dos maniobras. Ambas comienzan en una condición de vuelo estacionario. En la primera de las maniobras se ha comandado un “doblete” de roll (comando de roll positivo seguido de comando de roll negativo o viceversa), mientras que en la segunda se ha comandado un doblete de pitch. Los resultados se muestran a continuación (Figura 85 para el roll y Figura 86 para el pitch).

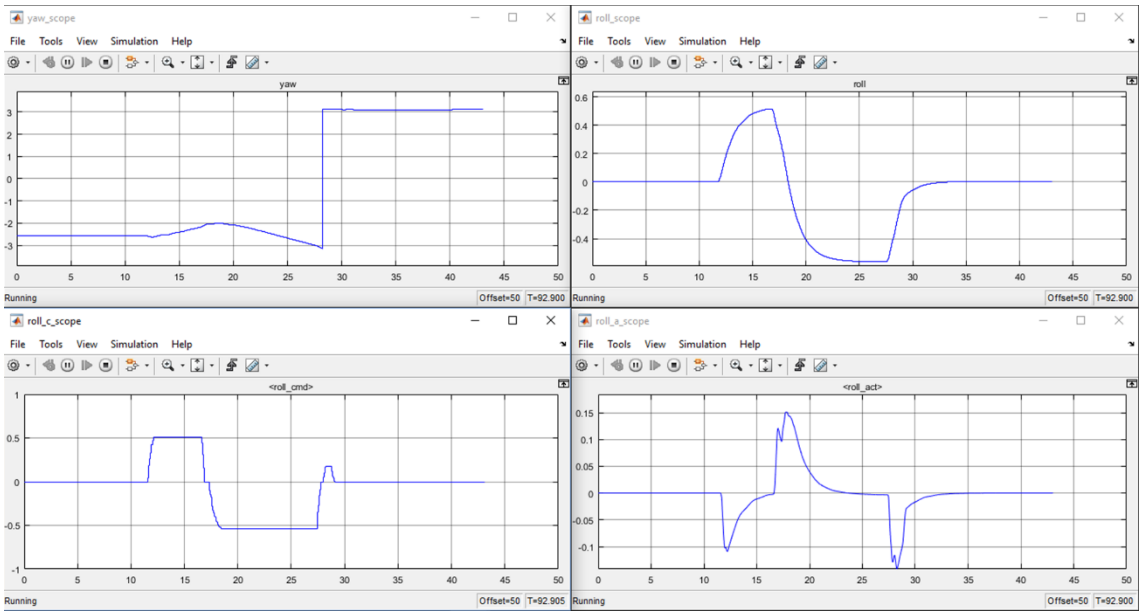

Figura 85: Maniobra manual en roll

Se recuerda que en todos los bloques "scope" el eje de abscisas corresponde al tiempo de simulación en segundos. La monitorización de estados durante el doblete de roll se ha llevado a cabo mediante los "scopes" correspondientes a ángulo de yaw (gráfica superior izquierda, en radianes), ángulo de roll (gráfica superior derecha, en radianes), roll comandado (gráfica inferior izquierda, en una escala de -1 a 1) y ángulo de los alerones (gráfica inferior derecha, en radianes).

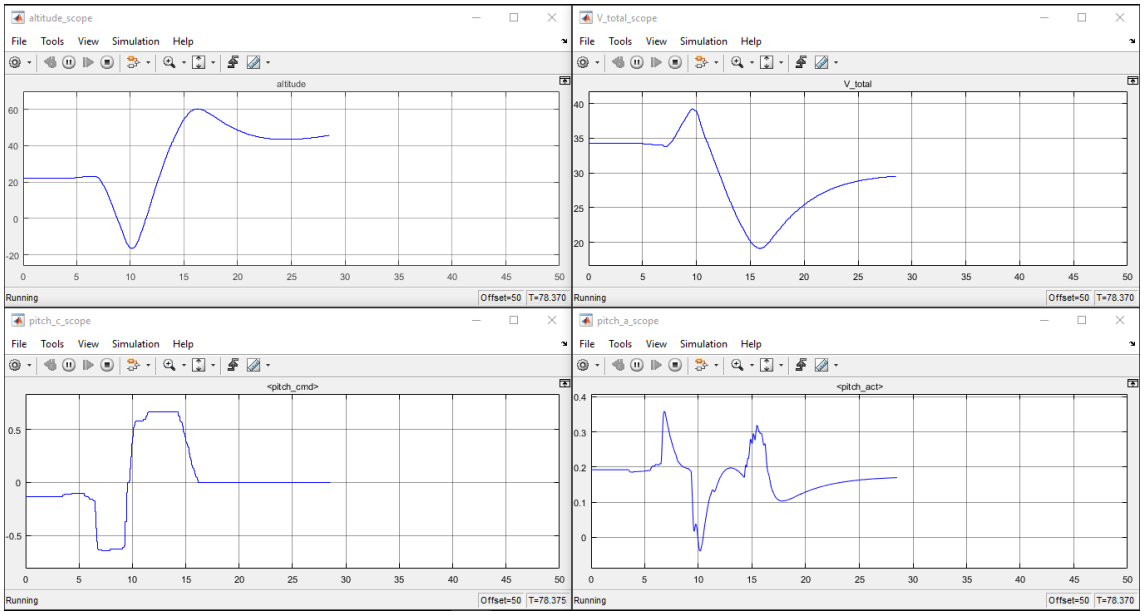

Figura 86: Maniobra manual en pitch

Durante el doblete de pitch, la monitorización de estados se ha llevado a cabo mediante los "scopes" correspondientes a altitud (gráfica superior izquierda, en metros), velocidad total

(gráfica superior derecha, en metros por segundo), pitch comandado (gráfica inferior izquierda, en una escala de -1 a 1) y ángulo del elevador (gráfica inferior derecha, en radianes).

En este grupo de gráficas se puede observar que cuando la altitud comienza a bajar, la velocidad comienza a subir y viceversa.

6.2 Autopilotos de altitud y velocidad

A continuación se muestran los resultados obtenidos al comandar una señal escalón en altitud de amplitud 5 metros con el autopiloto de control de altitud y velocidad constantes activado (Figura 87).

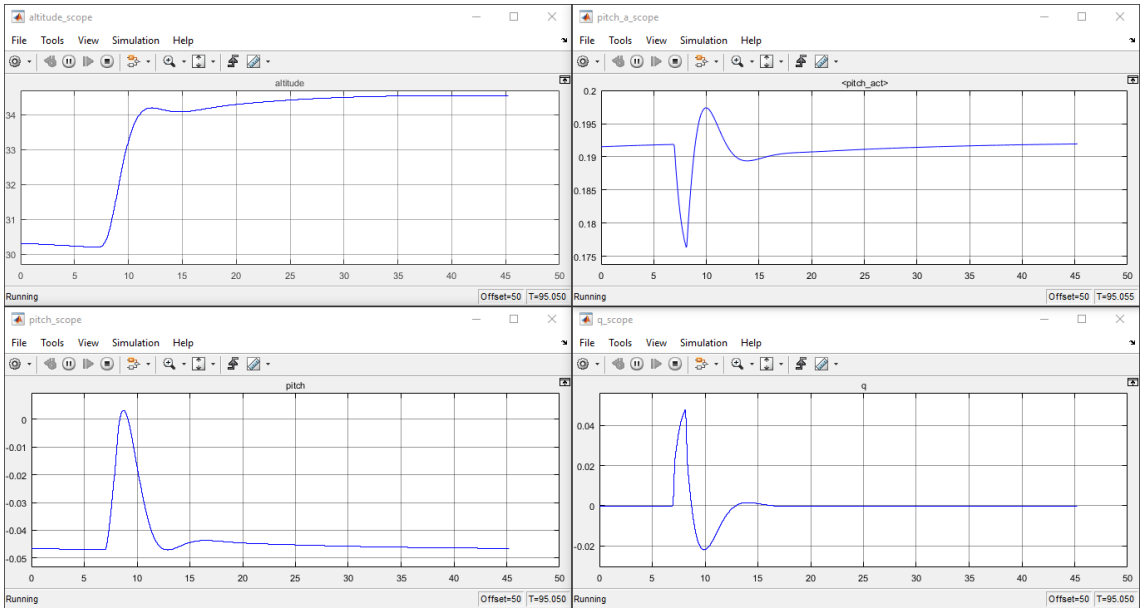

Figura 87: Comando de altitud al autopiloto (modo 1)

El modo de funcionamiento del autopiloto es el modo 1. La monitorización de estados se ha llevado a cabo mediante los “scopes” correspondientes a altitud (gráfica superior izquierda, en metros), ángulo del elevador (gráfica superior derecha, en radianes), ángulo de pitch (gráfica inferior izquierda, en radianes) y velocidad angular de cabeceo (gráfica inferior derecha, en radianes por segundo).

A continuación se muestran los resultados de una maniobra similar a la anterior pero con un comando de señal escalón en velocidad total de amplitud 2 m/s (Figura 88). Además, el modo de vuelo es ahora el modo 0.

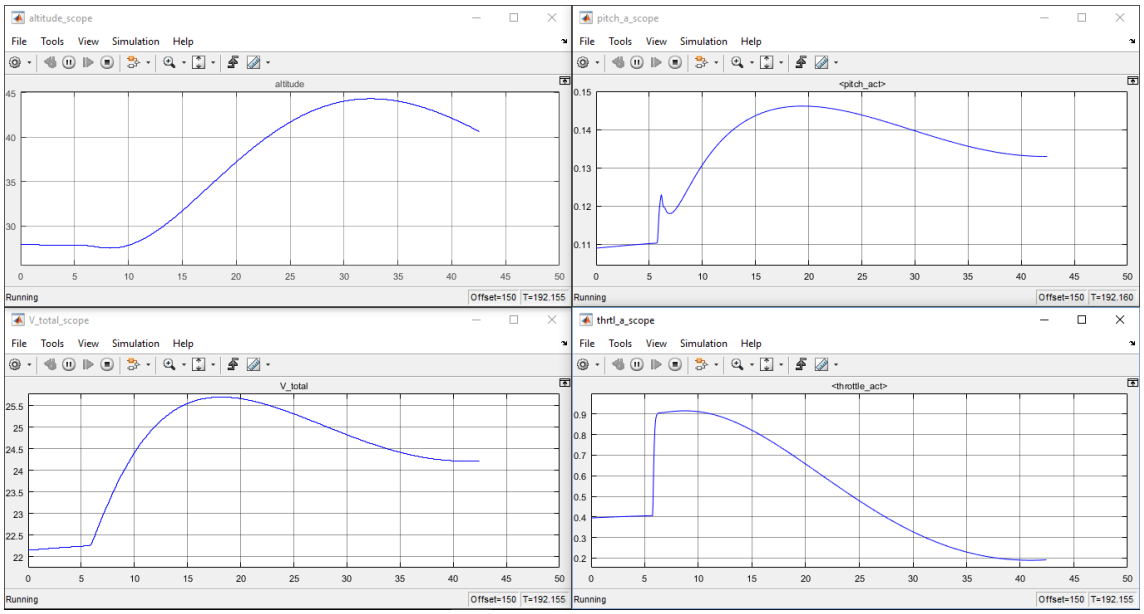

Figura 88: comando de velocidad al autopiloto (modo 0)

La monitorización de estados se ha llevado a cabo mediante los "scopes" correspondientes a altitud (gráfica superior izquierda, en metros), ángulo del elevador (gráfica superior derecha, en radianes), velocidad total (gráfica inferior izquierda, en metros por segundo) y factor de regulación de empuje (gráfica inferior derecha, en una escala de 0 a 1).

Se puede observar que la respuesta en velocidad es más lenta que la respuesta en altitud. Esto se debe, entre otros factores, a que el motor tarda más en reaccionar a los comandos que los actuadores del elevador. Además conviene aclarar que este autopiloto controla dos variables que están acopladas entre sí (cambio de altitud y velocidad), por lo que las respuestas suelen componerse de una secuencia iterativa de condiciones de equilibrado temporal de cada una de las variables individualmente hasta que finalmente se encuentra la condición de equilibrio común para las dos.

6.3 Sistema de guiado

El procedimiento seguido para mostrar el comportamiento del sistema de guiado ha sido el siguiente. Se ha activado el sistema de guiado con un ángulo de yaw objetivo para la recta de referencia de valor 20 grados sexagesimales superior al ángulo de yaw que poseía la aeronave en el momento de la activación (Figura 89).

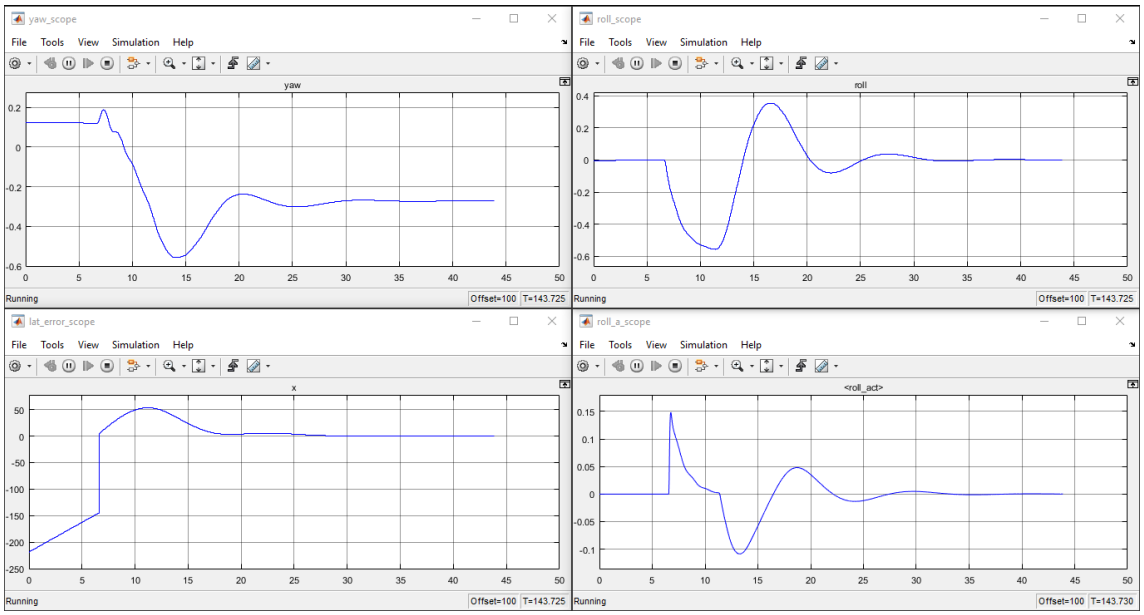

Figura 89: Activación del sistema de guiado

Se puede apreciar que en el momento de la activación, el error lateral cambia súbitamente a 0 y comienza a crecer debido a que la aeronave no se mueve en la dirección de la recta de referencia. Por lo que el sistema de guiado comienza a comandar un roll que a su vez consigue corregir el yaw, hasta que la aeronave alcanza la posición y dirección de la recta de referencia y el error lateral se anula.

6.4 Capacidad de análisis

Con el fin de mostrar la capacidad de análisis de la plataforma, se ha realizado una secuencia de procesos de análisis del sistema desarrollado. Primeramente, se han representado todos los tipos posibles de diagramas para el modelo lineal resultante al linealizar la respuesta en ángulo de roll a un comando de alerones. Seguidamente se ha representado la respuesta a una entrada escalón para todas las combinaciones de inputs/outputs posibles del sistema (roll, pitch, p, q, altitud y velocidad). Los resultados se muestran a continuación.

6.4.1 Respuesta del ángulo de roll a un comando de alerones

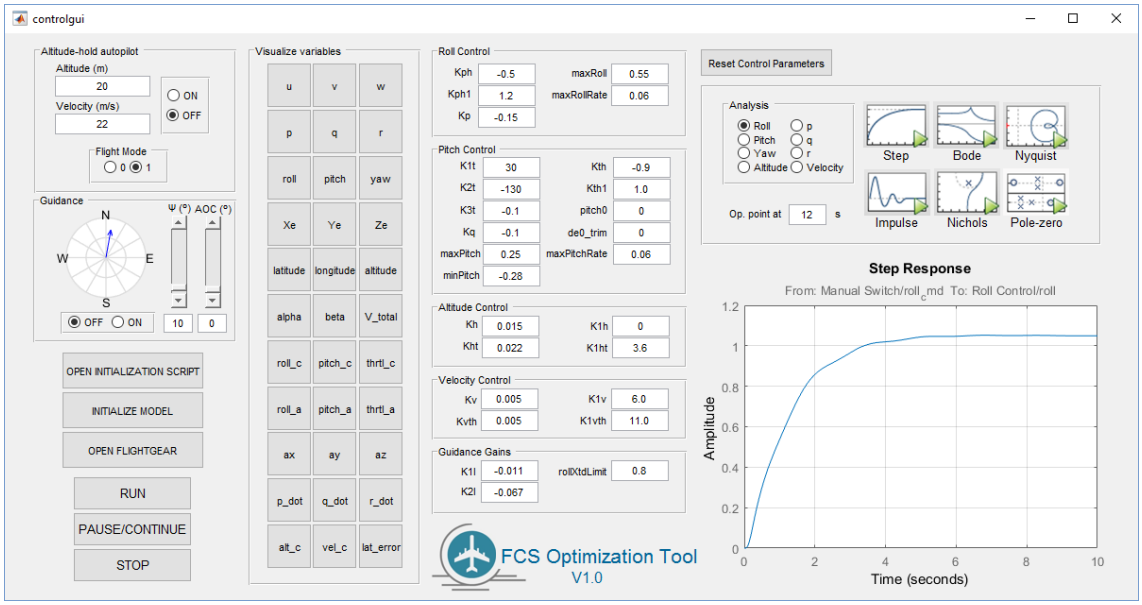

Figura 90: diagrama de respuesta a escalón (Roll_cmd VS roll)

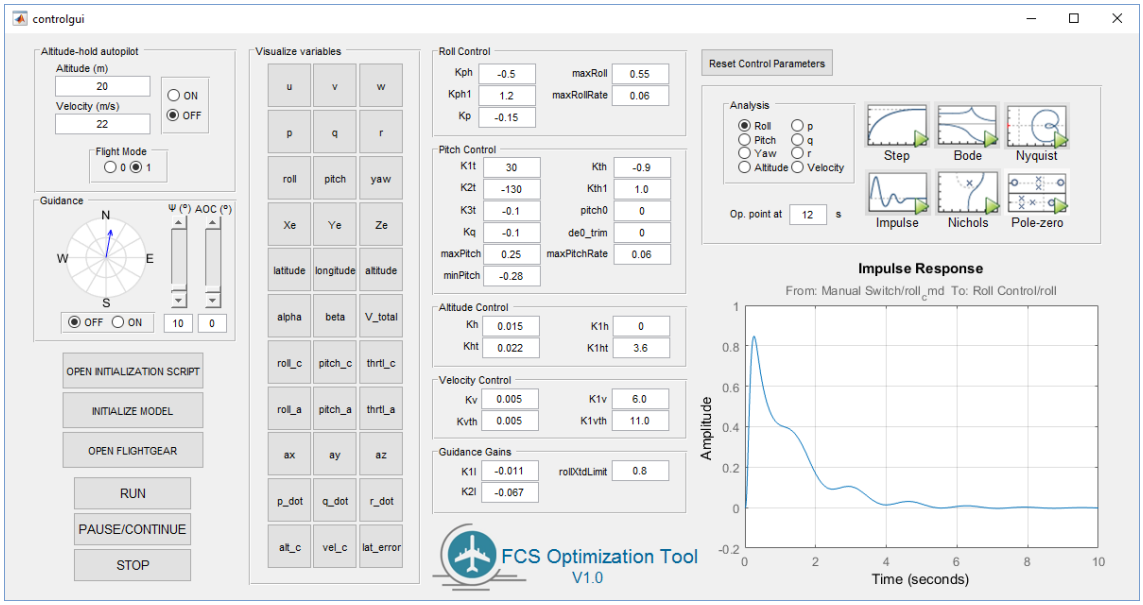

Figura 91: diagrama de respuesta a impulso (Roll_cmd VS roll)

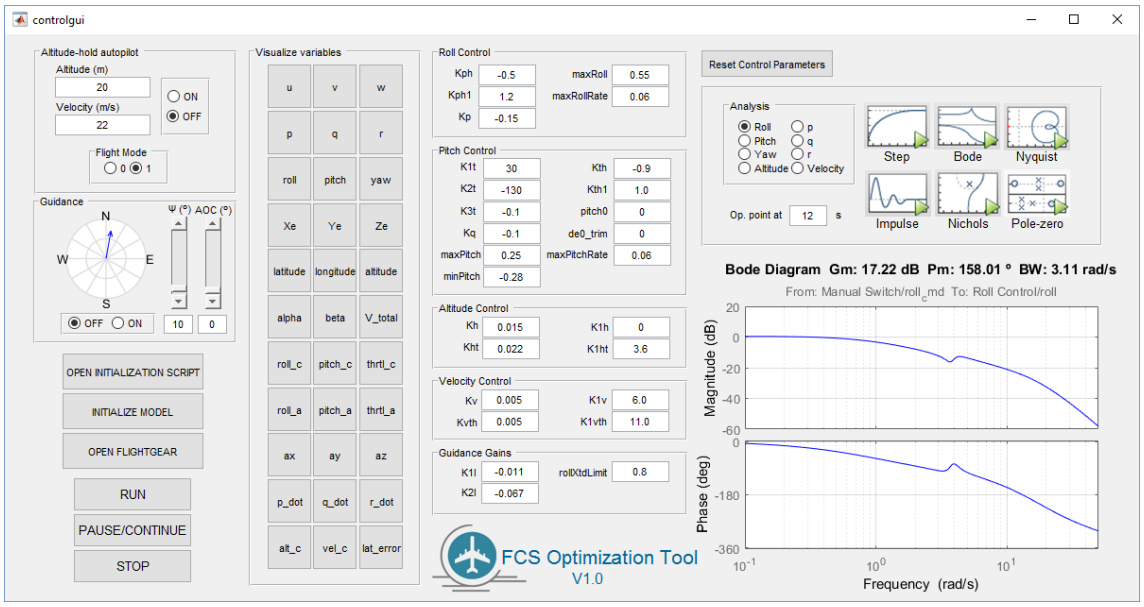

Figura 92: diagrama de Bode (Roll_cmd VS roll)

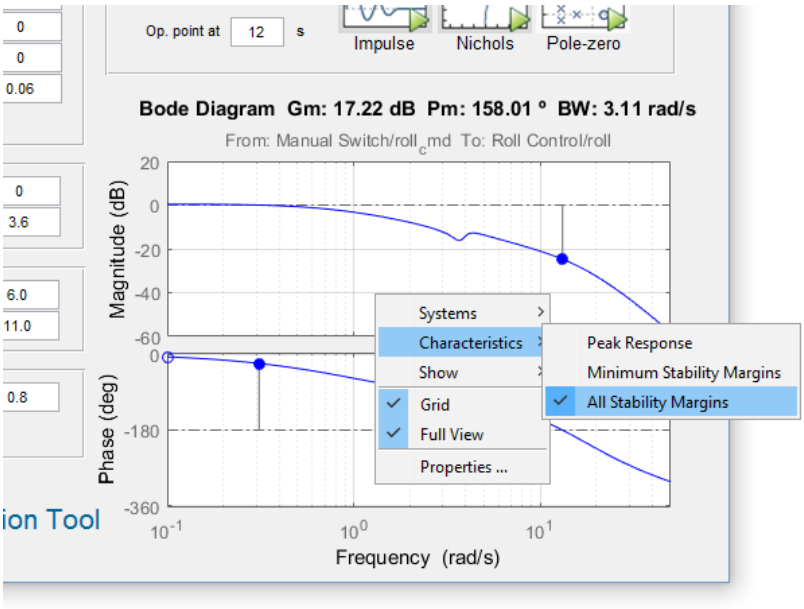

Figura 93: diagrama de Bode mostrando márgenes de estabilidad (Roll_cmd VS roll)

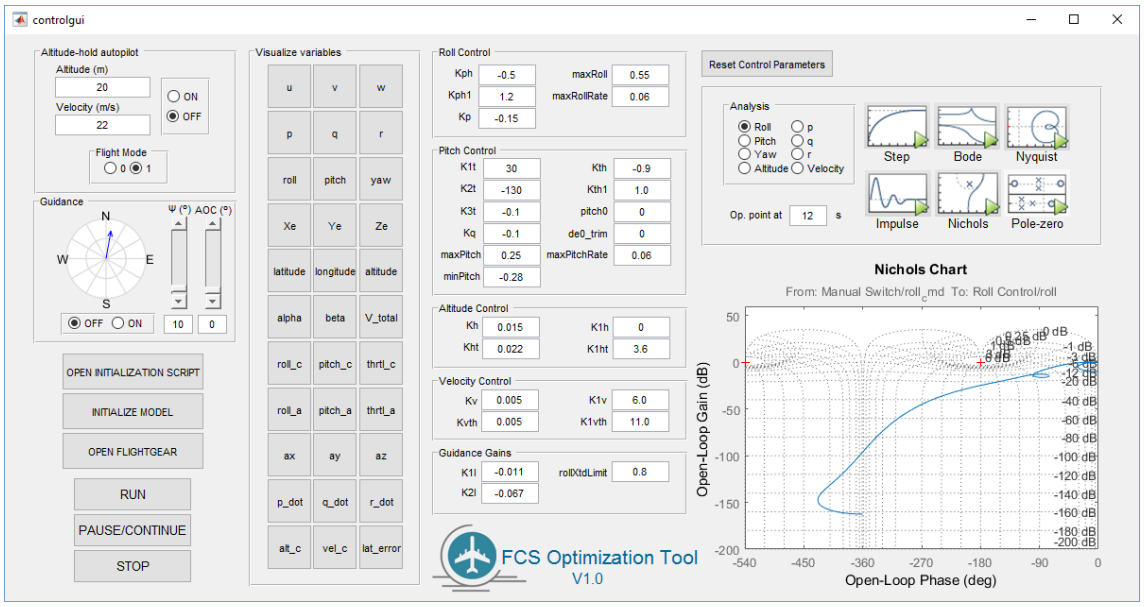

Figura 94: diagrama de Nichols (Roll_cmd VS roll)

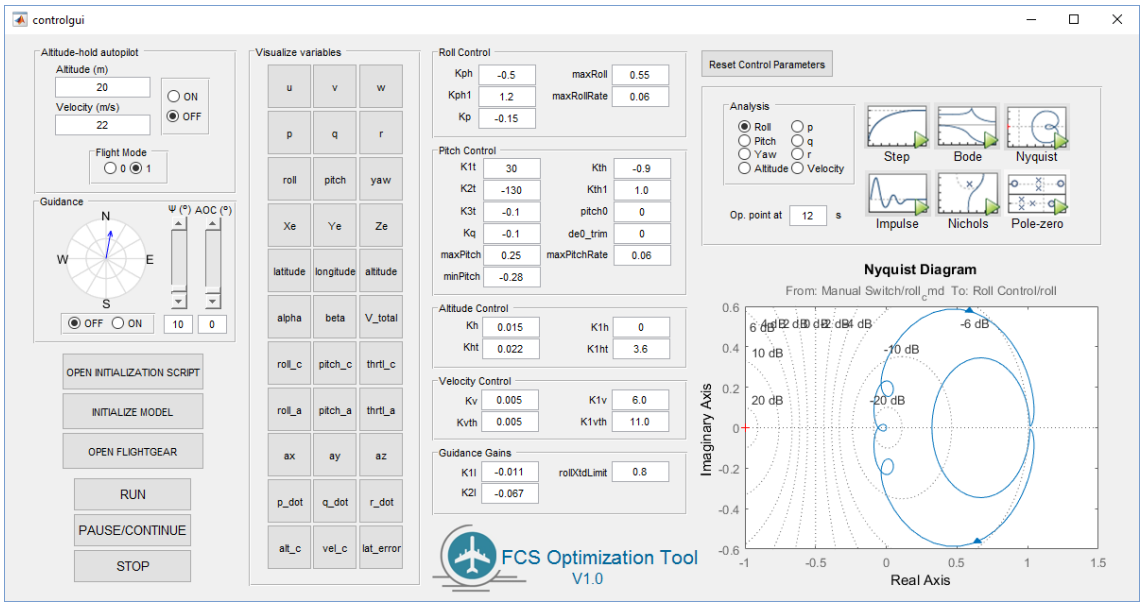

Figura 95: diagrama de Nyquist (Roll_cmd VS roll)

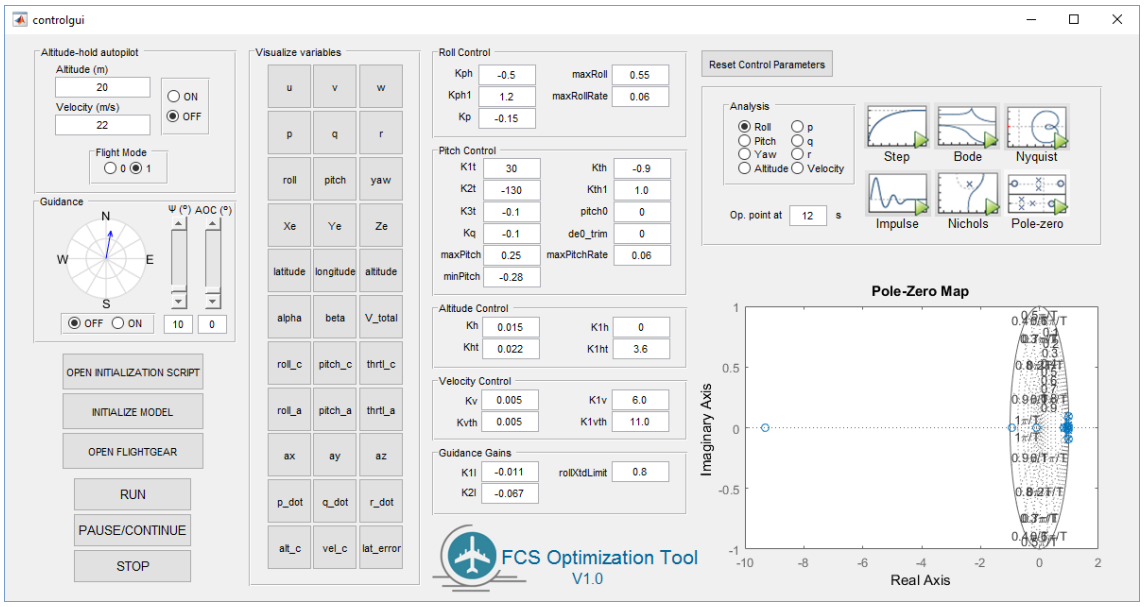

Figura 96: Diagrama de polos y ceros (Roll_cmd VS roll)

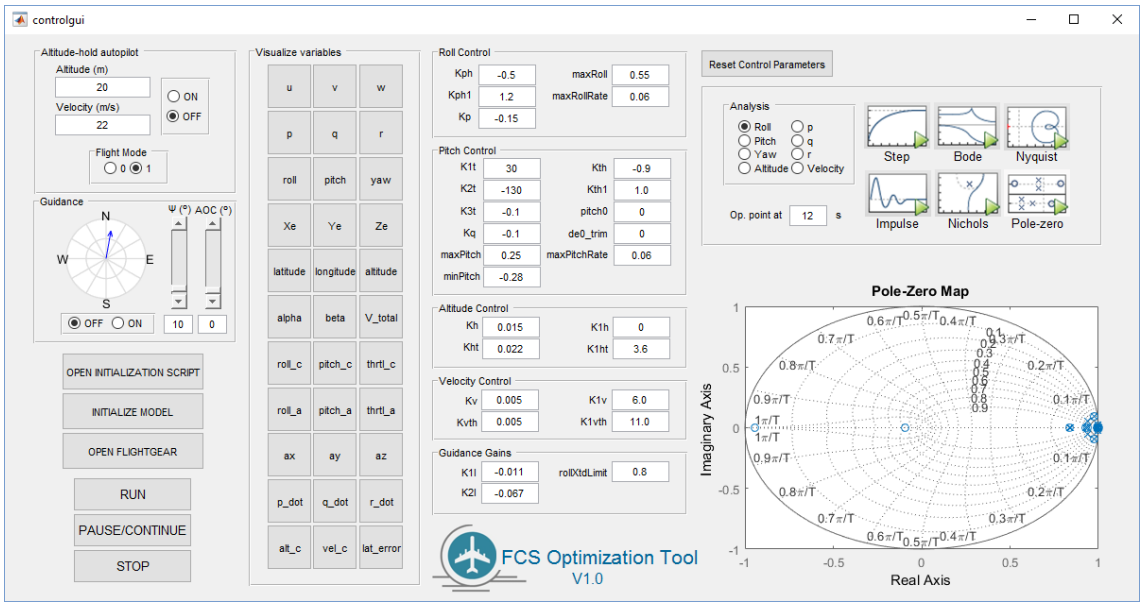

Figura 97: Diagrama de polos y ceros aumentado (Roll_cmd VS roll)

6.4.2 Respuesta a entrada escalón de todos los bucles del sistema

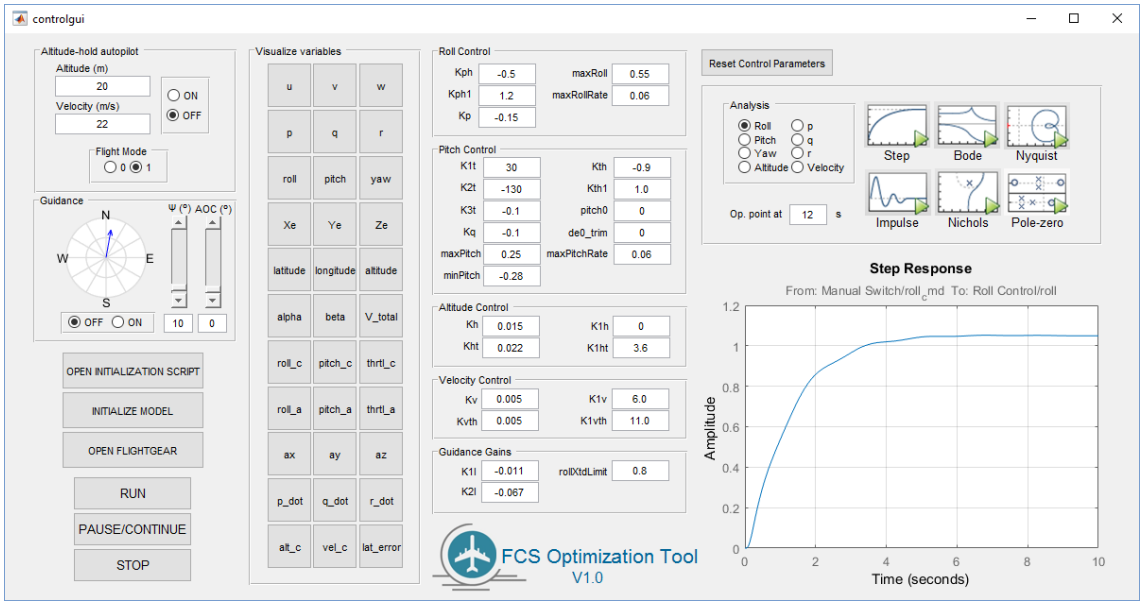

Figura 98: Diagrama de respuesta a escalón (Roll_cmd VS roll)

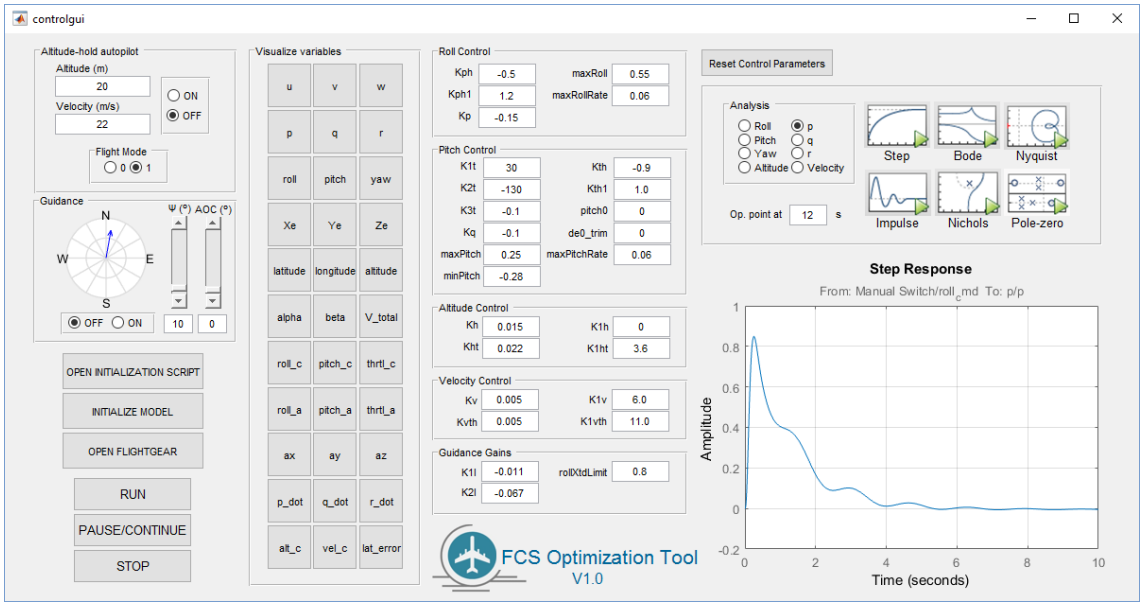

Figura 99: Diagrama de respuesta a escalón (Roll_cmd VS p)

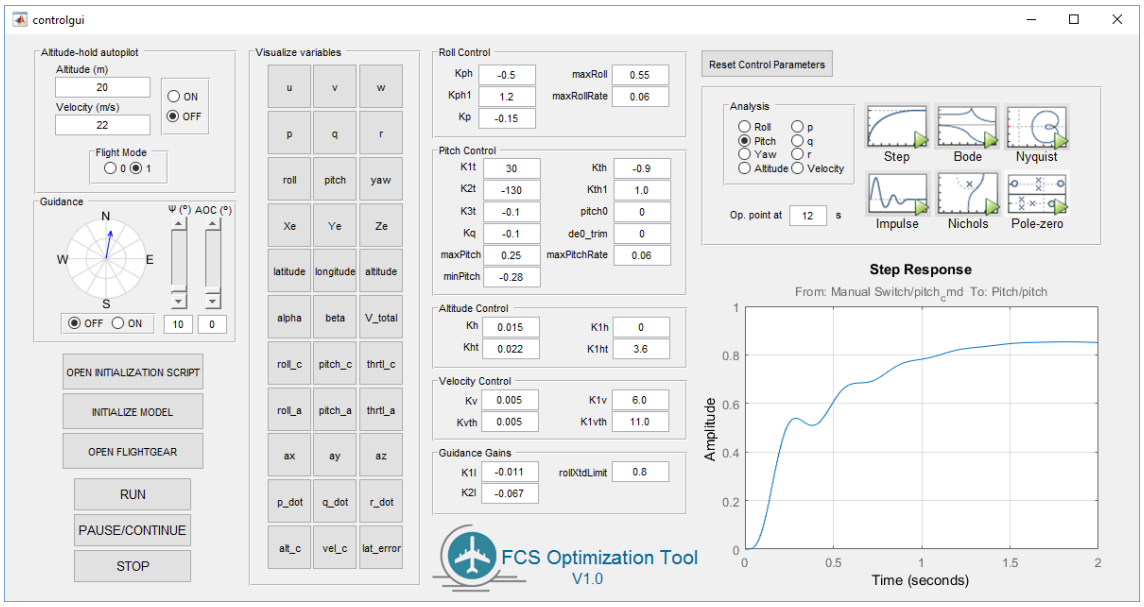

Figura 100: Diagrama de respuesta a escalón (Pitch_cmd VS Pitch)

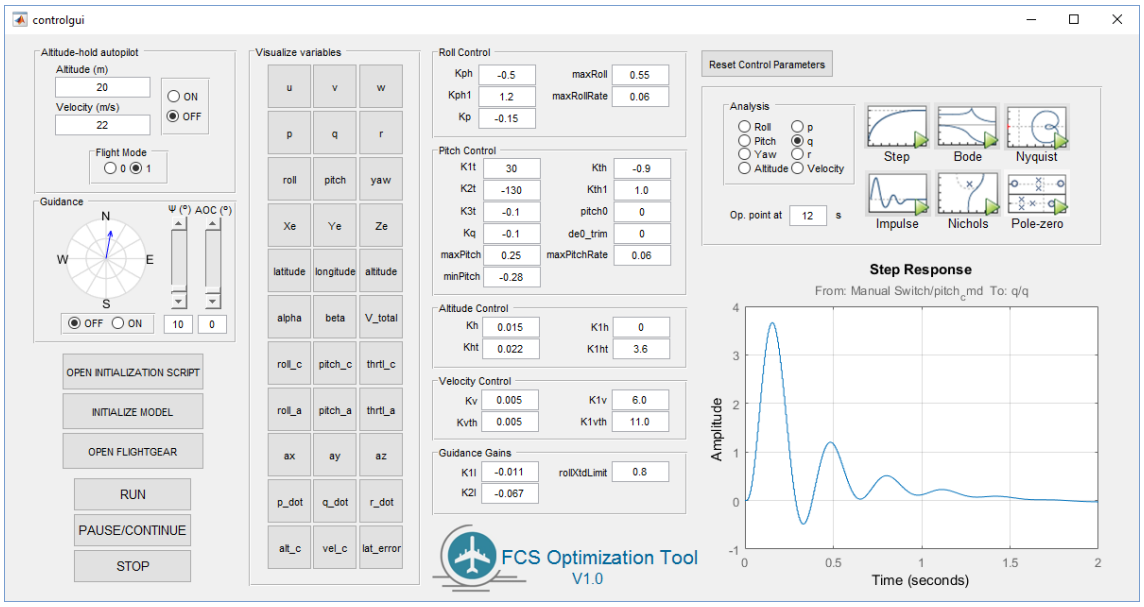

Figura 101: Diagrama de respuesta a escalón (Pitch_cmd VS q)

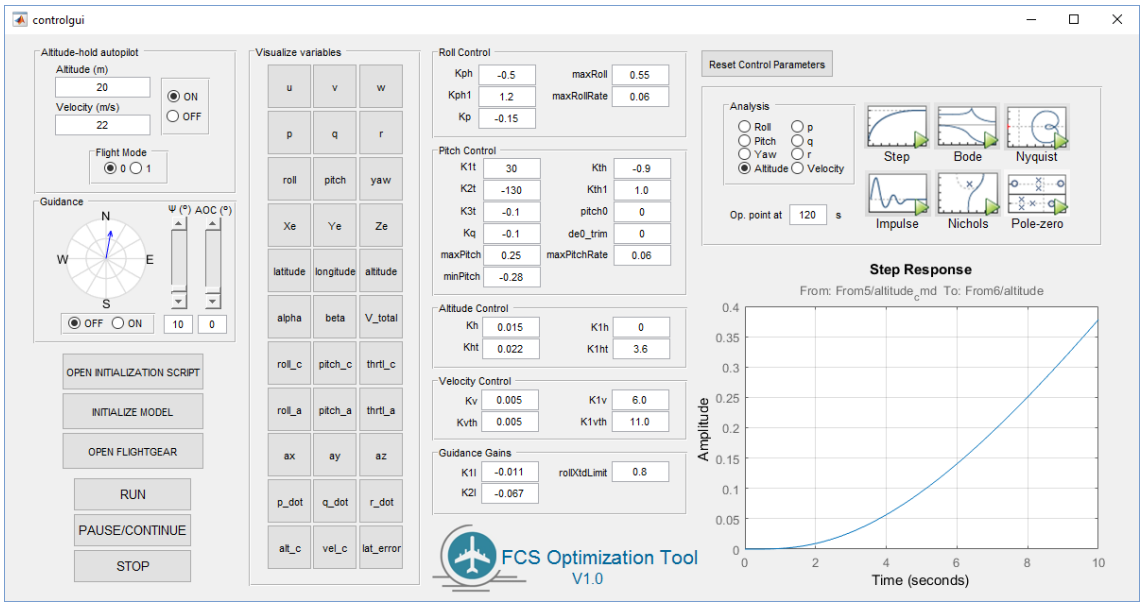

Figura 102: Diagrama de respuesta a escalón (altitude_cmd VS altitude)

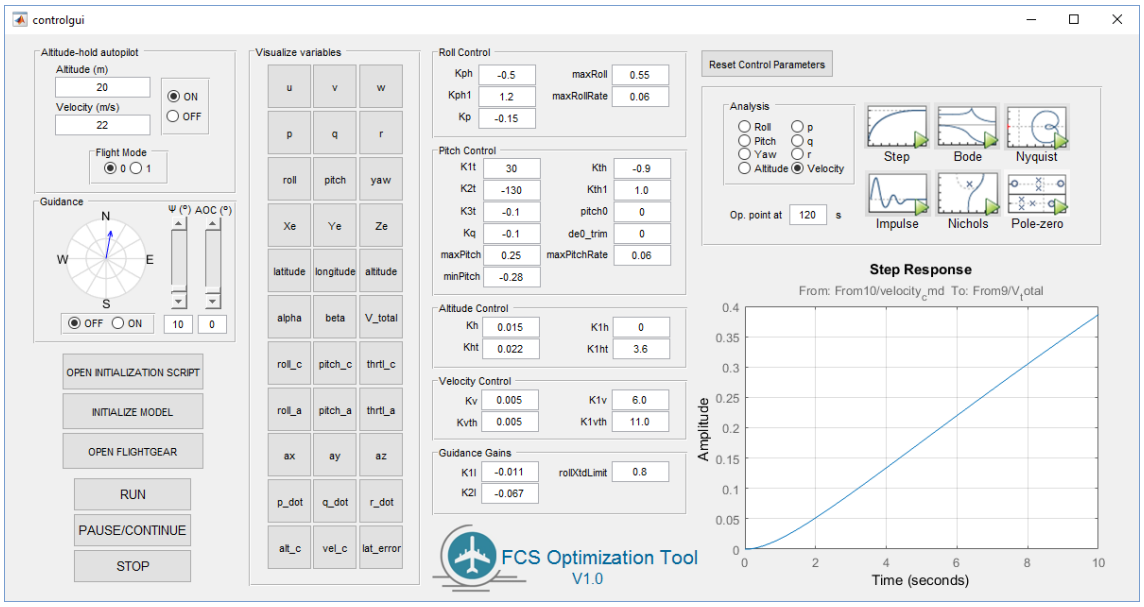

Figura 103: Diagrama de respuesta a escalón (velocity_cmd VS velocity)

Capítulo 7

Conclusiones y proyección futura

7.1 Conclusiones

Los objetivos que se planteaban en este trabajo se han cumplido satisfactoriamente. En efecto, a través del uso de distintas herramientas de software incluyendo "MATLAB®", "Simulink®", "FlightGear", "CATIA" y, para el anexo A, "Qt", se ha conseguido desarrollar una herramienta de software que permite optimizar sistemas de control de vuelo de aeronaves de manera eficiente. En este caso un sistema adaptado a una aeronave no tripulada ficticia, aunque la plataforma es perfectamente adaptable a otras aeronaves e incluso a otros sistemas dinámicos.

La plataforma de simulación es **versátil** y **funcional**, y constituye una herramienta muy útil para la optimización de sistemas de control de vuelo.

El **simulador** admite el proceso de modelado de cualquier sistema dinámico de una forma ágil gracias al entorno visual Simulink®. Cada módulo puede expandirse incluyendo nuevos elementos.

La **interfaz gráfica** añade mucha flexibilidad, facilitando la ejecución de distintos procesos. Al igual que el simulador, la interfaz está completamente abierta a desarrollos futuros.

El **visualizador** FlightGear otorga a la plataforma un valor añadido adicional al posibilitar la visualización tridimensional de la aeronave en tiempo real.

7.2 Proyección futura

Esta plataforma está abierta a nuevos desarrollos futuros entre los que podrían citarse:

- Implementación de los movimientos de las superficies de control en tiempo real en el visualizador FlightGear. Para ello, es necesario que la geometría de la aeronave esté compuesta por varias secciones y que se hayan definido ejes de rotación de unas sobre otras.

- Mejora del modelo de motor del simulador para obtener un modelo más complejo.
- Implementación de otras funcionalidades en la interfaz gráfica que interactúa con el simulador. Se citan algunos ejemplos de funcionalidades que podrían ser factibles:
    a) Capacidad interactiva con el modelo de la aeronave (tipo de modelo, derivadas aerodinámicas, coeficientes de matrices de la representación en espacio de estados, etc.)
    b) Capacidad de introducir automáticamente una nueva geometría en FG y realizar la configuración correspondiente. Implica además crear un nuevo script de inicialización de FG.
    c) Capacidad de definición de las variables de inicialización en la propia interfaz, en lugar de en el script externo “startVars.m”.
    d) Capacidad de control del tiempo de muestreo de la simulación, tipo de algoritmo de simulación, entre otros.

Figura 104: Plataforma de simulación

Bibliografía

[1] B. L. Stevens and F. L. Lewis, *Aircraft Control and Simulation*, Second Edi. Wiley, 2003.

[2] "Simulink User's Guide R2015b." The Mathworks Inc., 2015.

[3] S. C. Chapra, *Applied Numerical Methods with MATLAB.*, Second Edi. McGraw-Hill Higher Education, 2004.

[4] A. Knight, *Basics of MATLAB and Beyond*. Taylor & Francis, 1999.

[5] H. Moore, *MATLAB para ingenieros*. Pearson Education, 2007.

[6] M. Kalechman, *Practical MATLAB Applications for engineers*. CRC Press, 2008.

[7] "Documentación de CATIA V5." [Online]. Available: goo.gl/XTUiaJ.

[8] "Artículo de documentación de Mathworks: Memory." [Online]. Available: http://goo.gl/pnlc3u.

[9] "Artículo de documentación de Mathworks: What is Sample Time?" [Online]. Available: http://goo.gl/9GdMY7.

[10] "Artículo de documentación de Mathworks: Variant Subsystem." [Online]. Available: http://goo.gl/HN3Cbl.

[11] "Artículo de documentación de Mathworks: Quadcopter Project." [Online]. Available: http://goo.gl/RFbzzs.

[12] "Artículo de documentación de Mathworks: Aerospace Blockset." [Online]. Available: http://goo.gl/O9Kocd.

[13] "Artículo de documentación de Mathworks: Pilot Joystick." [Online]. Available: http://goo.gl/TdVRc0.

[14] "Artículo de documentación de Mathworks: Bus Creator." [Online]. Available: http://goo.gl/Z6cvAp.

[15] M. J. Abzug and E. E. Larrabee, *Airplane stability and control: A history of the technologies that made aviation possible*, Second Edi. Cambridge Aerospace Series, 2002.

[16] A. Farré Gabernet, "Controllers for Systems with Bounded Actuators: Modeling and Control of an F-16 aircraft.," University of California, Irvine, 2007.

[17] "Artículo de documentación de Mathworks: IC." [Online]. Available: http://goo.gl/iQGN0W.

[18] M. V. Cook, *Flight Dynamics Principle. A Linear Systems Approach to Aircraft Stability and Control.*, Third Edit. Elsevier, 2013.

[19] M. Á. Gómez Tierno, M. Pérez Cortés, and C. Puentes Márquez, *Mecánica del Vuelo*. Garceta, 2012.

[20] "Artículo de documentación de Mathworks: Incidence, Sideslip & Airspeed." [Online]. Available: http://goo.gl/fW4Dzs.

[21] "Artículo de documentación de Mathworks: 6DOF (Euler Angles)." [Online]. Available: http://goo.gl/5zua41.

[22] A. E. Ahmed, O. A. N. Hafez A., H. E. Hussein Ahmed, and H. M. Abd-Elkader, "Modelling of a Small Unmanned Aerial Vehicle," *Int. J. Mech. Aerospace, Ind. Mechatron. Manuf. Eng.*, vol. 9, no. 3, 2015.

[23] T. I. Fossen, "Mathematical Models for Control of Aircraft and Satellites," *Dep. Eng. Cybern.*, no. January, 2011.

[24] "Artículo de documentación de Mathworks: State-space." [Online]. Available: http://goo.gl/DXQOVP.

[25] "Artículo de documentación de Mathworks: Flat Earth to LLA." [Online]. Available: http://goo.gl/3DKYQ7.

[26] "Artículo de documentación de Mathworks: WGS84 Gravity Model." [Online]. Available: http://goo.gl/kZBeIK.

[27] "Artículo de documentación de Mathworks: COESA Atmosphere Model." [Online]. Available: http://goo.gl/3TZTkW.

[28] "Artículo de documentación de Mathworks: World Magnetic Model 2015." [Online]. Available: http://goo.gl/Zu1hxj.

[29] O. J. Woodman, "An introduction to inertial navigation," University of Cambridge, Computer Laboratory, 2007.

[30] D. H. Titterton and J. L. Weston, *Strapdown inertial navigation technology*. IET, 2004.

[31] "Artículo de doumentación de Mathworks: Three-Axis Inertial Measurement Unit." [Online]. Available: http://goo.gl/xm6G40.

[32] "Artículo de documentación de Mathworks: Three-Axis Accelerometer." [Online]. Available: http://goo.gl/c8oJ85.

[33] "Artículo de documentación de Mathworks: Three-Axis Gyroscope." [Online]. Available: http://goo.gl/s8jPKZ.

[34] N. Khaled, *Virtual Reality and Animation for MATLAB® and Simulink® Users. Visualization of Dynamic Models and Control Simulations.* Springer, 2012.

[35] "Artículo de documentación de Mathworks: Pack net_fdm Packet for FlightGear." [Online]. Available: http://goo.gl/16mHuh.

[36] "Artículo de documentación de Mathworks: Send net_fdm Packet to FlightGear." [Online]. Available: http://goo.gl/YBtBbR.

[37] "Artículo de documentación de Mathworks: Generate Run Script." [Online]. Available: http://goo.gl/bvOggA.

[38] "Artículo de documentación de Mathworks: Simulation Pace." [Online]. Available: http://goo.gl/wLZ8kF.

[39] "Artículo de documentación de Mathworks: Virtual and Nonvirtual Buses." [Online]. Available: http://goo.gl/X9mNMD.

[40] "Artículo de documentación de Mathworks: Bus Objects." [Online]. Available: http://goo.gl/rPDrj8.

[41] "Artículo de documentación de Mathworks: GUI de Matlab." [Online]. Available: goo.gl/YgYQSj.

[42] "Artículo de documentación de Mathworks: Compass." [Online]. Available: http://goo.gl/Me5iql.

[43] R. C. Dorf and R. H. Bishop, *Modern Control Systems*, 13th Editi. Pearson, 2016.

[44] "Artículo de documentación de Mathworks: Linearize." [Online]. Available: http://goo.gl/BzFWHS.

[45] "Página oficial de la librería QCustomPlot." [Online]. Available: goo.gl/RR47WW.

Anexo A

Simulador C++ en Qt

Es posible que el usuario de esta plataforma se encuentre con la necesidad de desarrollar un buen modelo dinámico de una aeronave que ya haya realizado algún vuelo de prueba. Esto es un caso común en el área de diseño y desarrollo de aeronaves no tripuladas. Si se dispone de datos medidos durante vuelos de prueba se pueden utilizar éstos para desarrollar un modelo matemático que represente de forma aproximada la dinámica del avión. Para ello, se replican los movimientos reales de las superficies de control que se midieron durante las pruebas en vuelo en el simulador y se compara la respuesta del propio simulador con la respuesta de los vuelos reales. De esta manera se puede realizar un proceso de ajuste de parámetros manual que ayuda a conseguir un modelo dinámico fidedigno.

Para llevar a cabo esta tarea se ha desarrollado un simulador totalmente independiente de la plataforma de simulación objeto del presente trabajo. Este simulador se ha desarrollado en lenguaje C++, y para ello se ha utilizado el marco de trabajo de aplicaciones multiplataforma Qt. Este marco de trabajo se utiliza ampliamente para desarrollar aplicaciones de software que puedan ejecutarse en diversas plataformas (tanto de hardware como de software) con pocas o ningunas modificaciones en el código base y manteniendo la misma capacidad y velocidad que la aplicación nativa.

El simulador que se ha desarrollado no dispone de muchas de las funcionalidades presentes en la plataforma de simulación centrada en Simulink®. Por ejemplo, no se ha incorporado la capacidad de introducir inputs desde un Joystick, o de visualizar los estados de la aeronave en una aplicación que actúe como motor gráfico en tres dimensiones.

La estructura del simulador en Qt es idéntica al de la plataforma de simulación. Por ello, no se entrará en tanto detalle al explicar la matemática y la física que hay detrás del funcionamiento del simulador, dado que ya se ha explicado ésta detalladamente a lo largo de este documento.

Tal y como se hizo con la plataforma de simulación centrada en Simulink®, se ha desarrollado también junto con este simulador una interfaz gráfica que interactúa con el simulador. Para crear esta interfaz gráfica se han utilizado las funcionalidades para creación de interfaces que tiene incorporadas el software Qt.

A.1 Código principal

El código del simulador se ubica en varios archivos (tanto archivos fuente como archivos de encabezamiento). Dado que el desarrollo de este simulador no es el objeto principal del presente trabajo, sólo se van a mostrar en este anexo las partes esenciales del código del simulador, útiles para comprender su funcionamiento. La parte más importante del código se ubica en el archivo fuente "mainwindow.cpp", por lo que todo el código que se va a mostrar en este anexo pertenece a ese archivo.

A continuación se muestra la sección de código principal del simulador.

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include "math_utils.h"
#include <QByteArray>
#include <QPushButton>
#include <math.h>    /* ceil */
#include <iostream>  /* cout */
#include <fstream>   /* ofstream */
#include <QFile>
#include <QCoreApplication>
#include <QTextStream>

MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    socket = new QUdpSocket(this);
    socket->bind(QHostAddress::LocalHost,5500);
    ui->setupUi(this);
    ms = 5;      // Time step in miliseconds
    InitSim();  // Initialize simulator (constant parameter
     definitions, variables initial states and read data from .txt)
    InitPlot(); // Initialize plots
    dtime = ms * 0.001F;
    connect(&dataTimer, SIGNAL(timeout()), this, SLOT(runSim()));
    // Runs simulator in each time step
    connect(&dataTimer, SIGNAL(timeout()), this, SLOT(updateplot()));
    // Updates plots in each time step
    dataTimer.start(ms);
}

MainWindow::~MainWindow()
{
    delete ui;
}
```

La función principal del código es la función "MainWindow". La función realiza las siguientes tareas secuencialmente:

(1) Se define la variable "ms" que representa el tiempo de muestra al que se va a ejecutar el simulador en milisegundos.
(2) Se ejecuta la función "InitSim". Esta función se encarga de inicializar el simulador.

(3) Se ejecuta la función “InitPlot”. Esta función se encarga de inicializar las gráficas de visualización que se encuentran en la interfaz gráfica.
(4) Se establece una conexión en la estructura “dataTimer” (o reloj) que ejecutará la función “runSim” a la frecuencia que se especifique cuando se ejecute el comando de inicio del mismo. La función “runSim” contiene el simulador de la aeronave.
(5) Se establece otra conexión en la estructura “dataTimer” que ejecutará la función “updateplot” a la frecuencia que se especifique cuando se ejecute el comando de inicio del mismo. La función “updateplot” actualiza las gráficas contenidas en la interfaz con los valores de los estados de la aeronave obtenidos de “runSim”.
(6) Se ejecuta el comando de inicio de la estructura “dataTimer” para que comience la simulación.

Las funciones “InitSim”, “InitPlot”, “runSim” y “updateplot” se detallarán a continuación.

A.1.1 Función “InitSim”

```
void MainWindow::InitSim()
{
    ////////////////////////////////////////////
    ////// CONSTANT PARAMETER DEFINITIONS //////
    ////////////////////////////////////////////

    // SIMULATION PARAMETERS
    time_step = 0.001F; // s

    // ENVIRONMENT
    g = 9.8F; // gravity (m/s^2)
    atm_density = 1.225F; // kg/m^3

    // GEOMETRY
    Sw = 1.068F; // m^2
    c = 0.2704F; // m
    b = 4.0F; // m

    // STABILITY DERIVATIVES
    cl0 = 0.57F;
    cla = 6.19F;
    clq = 0.0F;

    cd0 = 0.0289F;
    cda2 = 0.2F;

    cyb =  -0.316F;
    cydr =  0.00F; // no rudder

    float uncertainty = 0.7F;

    cllb =  -0.0413F*uncertainty;
    cllda =  -0.13F*uncertainty;
    clldr = 0.03F*uncertainty;
    cllp = -0.71F*uncertainty;
    cllr = 0.116F*uncertainty;

    cm0 = 0.095F*uncertainty;
    cma = -0.95F*uncertainty;
    cmde = -0.725F*uncertainty;
    cmq = -8.88F*uncertainty;
```

```
cmap = -10.0F*uncertainty;

cnb =  0.046F*uncertainty;
cnda = 0.0F*uncertainty;
cndr =  -0.05F*uncertainty;
cnp =  -0.084F*uncertainty;
cnr =   -0.021F*uncertainty;

// SIMPLE ENGINE MODEL. Thrust decreases with airspeed. At Vmax
 thrust is zero.
Tmax= 70.0F; // N
Vmax= 150.0F/3.6F; // m/s

// MASS AND MOMENT OF INERTIA. Assumed pricipal axis of inertia
mass_config = 1;
if (mass_config == 0)
{
    m = 31.0F; // kg
    Ix = 8.6F; // kg*m^2
    Iy = 0.85F; // kg*m^2
    Iz = 9.4F; // kg*m^2
    Ixz = -0.05F; // kg*m^3
}
else if (mass_config == 1)
{
    m = 22.0F; // kg
    Ix = 6.1F; // kg*m^2
    Iy = 0.61F; // kg*m^2
    Iz = 6.7F; // kg*m^2
    Ixz = -0.05F; // kg*m^3
}

// CONTROL PARAMETERS
Kth = -0.37F;
Kth1 = 1.5F;
Kq = -0.01F;
pitch0 = 0.0F;
K1t = 60.0F;
K2t = 80.0F;
K3t = 1.0F;
de0_trim = 0.0F;
Kph = -0.7F;
Kph1 = 0.5F;
Kp = -0.01F;
Kh = -0.15F;//-0.015F;
K1h = 0.0F;
samples = 1.0F/time_step; // for LPF
V_cruise = 38.6F;         // for equalization factor

// Flat Earth to LLA (Latitude, longitude, altitude)
f = 0.0033528F; // Flattening parameter of Earth (WGS84)
R = 6378.1F; // Equatorial radius of Earth

////////////////////////////////////
////// VARIABLES INITIAL STATES //////
////////////////////////////////////

if (mass_config == 0) {u = 24.7F;} // m/s
else if (mass_config == 1) {u = 38.6F;} // m/s
v = 0.0F; // m/s
w = 0.0F; // m/s
```

```
pitch = 0.0F; // rad
roll = 0.0F; // rad
yaw = 0.0F; // rad
p = 0.0F; // rad/s
q = 0.0F; // rad/s
r = 0.0F; // rad/s
posN = 0.0F; // m
posE = 0.0F; // m
h = 50.0F; // m
latitude = 0.0F; // m
longitude = 0.0F; // m
altitude = 50.0F; // m
pdot = 0.0F; // rad/s^2
qdot = 0.0F; // rad/s^2
rdot = 0.0F; // rad/s^2
alpha_dot = 0.0F; // rad/s^2
pitch_rate = 0.0F; // rad/s^2
roll_rate = 0.0F; // rad/s^2
yaw_rate = 0.0F; // rad/s^2
udot = 0.0F; // m/s^2
vdot = 0.0F; // m/s^2
wdot = 0.0F; // m/s^2
delta_e = 0.0F;
delta_t = 0.0F;
delta_a = 0.0F;
delta_r = 0.0F;
delta_e_cmd_lpf = 0.0F;
delta_t_cmd_lpf = 0.0F;
delta_a_cmd_lpf = 0.0F;
delta_r_cmd_lpf = 0.0F;
h_cmd_lpf = 0.0F;

V_total = sqrt(u*u+v*v+w*w); // these states depend upon the
 previous ones
alpha = atan(w/u);
beta = asin(v/V_total);
dynamic_pressure = 0.5*atm_density*V_total*V_total;

//////////////////////////////////////////////////////////////
////// READ FLIGHT DATA FROM .txt AND STORE IT IN ARRAYS //////
//////////////////////////////////////////////////////////////

QFile inputFile("txt_root_directory/file.txt");
if (inputFile.open(QIODevice::ReadOnly))
{
    QTextStream in(&inputFile);
    while (!in.atEnd())
    {
        static int i=0;
        QString line = in.readLine();
        QStringList lstLine = line.split(" ");
        time_txt[i] = lstLine.at(0).toFloat();
        ailerons_txt[i] = lstLine.at(1).toFloat();
        elevator_txt[i] = lstLine.at(2).toFloat();
        rudder_txt[i] = lstLine.at(3).toFloat();
        throttle_txt[i] = lstLine.at(4).toFloat();
        roll_txt[i] = lstLine.at(5).toFloat();
        pitch_txt[i] = lstLine.at(6).toFloat();
        yaw_txt[i] = lstLine.at(7).toFloat();
        altitude_txt[i] = lstLine.at(8).toFloat();
        p_txt[i] = lstLine.at(9).toFloat();
```

```
            q_txt[i] = lstLine.at(10).toFloat();
            r_txt[i] = lstLine.at(11).toFloat();
            ktas_txt[i] = lstLine.at(12).toFloat();
            latstck_txt[i] = lstLine.at(13).toFloat();
            lonstck_txt[i] = lstLine.at(14).toFloat();
            pedals_txt[i] = lstLine.at(15).toFloat();
            alpha_txt[i] = lstLine.at(16).toFloat();
            beta_txt[i] = lstLine.at(17).toFloat();
            i++;
        }
        inputFile.close();
    }
}
```

La función que se acaba de mostrar está compuesta por tres áreas principales. Inicialmente se inicializan y definen todos los parámetros constantes que se utilizarán en el simulador (geométricos, másicos, derivadas de estabilidad y de control…). A continuación se procede a inicializar y definir los valores iniciales de las variables de estado de la aeronave. Por último se incluye una sección de código que tiene la capacidad de leer datos externos ubicados en un archivo txt y convertirlos en "arrays" para utilizarlos posteriormente en el graficado con el fin de hacer la comparación con el avión real.

A.1.2 Función "InitPlot"

La función "InitPlot" se encarga de inicializar los espacios de visualización gráfica disponibles en la interfaz. El código necesario para utilizar esta funcionalidad no se ha desarrollado en el presente trabajo. Se ha utilizado una librería de código abierto llamada QCustomPlot para graficado y visualizado de datos. Para más información acerca de esta librería gratuita se puede consultar la referencia [45]. La función "InitPlot" llama a otras funciones más pequeñas que se encargan de inicializar cada gráfica individualmente.

```
void MainWindow::InitPlot()
{
    InitPlot_V_body();
    InitPlot_Omega_body();
    InitPlot_Euler();
    InitPlot_Position();
    InitPlot_LLA();
    InitPlot_Sideslip();
    InitPlot_Commands_Actuators();
    InitPlot_Accel_body();
    InitPlot_Omegadot_body();
}
```

Como ejemplo representativo de la estructura de cada una de las funciones "InitPlot_X" se muestra a continuación la función "InitPlot_Euler", que inicializa la gráfica que muestra los ángulos de Euler (Pitch, Roll y Yaw) de la aeronave.

```
void MainWindow::InitPlot_Euler()
{
    ui->roll_chart->legend->setVisible(true);
    ui->roll_chart->addGraph();
    ui->roll_chart->graph(0)->setName("roll (°)");
    ui->roll_chart->graph(0)->setPen(QPen(Qt::red));
    ui->roll_chart->addGraph();
```

```
    ui->roll_chart->graph(1)->setName("roll_act (°)");
    ui->roll_chart->graph(1)->setPen(QPen(Qt::green));
    ui->roll_chart->addGraph();
    ui->roll_chart->graph(2)->setName("roll_flight_data (°)");
    ui->roll_chart->graph(2)->setPen(QPen(Qt::blue));
    ui->roll_chart->yAxis->setRange(-40.0F, 40.0F);
    ui->roll_chart->axisRect()->setupFullAxesBox();
    ui->roll_chart->setInteractions(QCP::iRangeDrag |
   QCP::iRangeZoom);

    ui->pitch_chart->legend->setVisible(true);
    ui->pitch_chart->addGraph();
    ui->pitch_chart->graph(0)->setName("pitch (°)");
    ui->pitch_chart->graph(0)->setPen(QPen(Qt::red));
    ui->pitch_chart->addGraph();
    ui->pitch_chart->graph(1)->setName("pitch_act (°)");
    ui->pitch_chart->graph(1)->setPen(QPen(Qt::green));
    ui->pitch_chart->addGraph();
    ui->pitch_chart->graph(2)->setName("pitch_flight_data (°)");
    ui->pitch_chart->graph(2)->setPen(QPen(Qt::blue));
    ui->pitch_chart->yAxis->setRange(-40.0F, 40.0F);
    ui->pitch_chart->axisRect()->setupFullAxesBox();
    ui->pitch_chart->setInteractions(QCP::iRangeDrag |
   QCP::iRangeZoom);

    ui->yaw_chart->legend->setVisible(true);
    ui->yaw_chart->addGraph();
    ui->yaw_chart->graph(0)->setName("yaw (°)");
    ui->yaw_chart->graph(0)->setPen(QPen(Qt::red));
    ui->yaw_chart->addGraph();
    ui->yaw_chart->graph(1)->setName("yaw_act (°)");
    ui->yaw_chart->graph(1)->setPen(QPen(Qt::green));
    ui->yaw_chart->addGraph();
    ui->yaw_chart->graph(2)->setName("yaw_flight_data (°)");
    ui->yaw_chart->graph(2)->setPen(QPen(Qt::blue));
    ui->yaw_chart->yAxis->setRange(-40.0F, 40.0F);
    ui->yaw_chart->axisRect()->setupFullAxesBox();
    ui->yaw_chart->setInteractions(QCP::iRangeDrag | QCP::iRangeZoom);
}
```

En la función que se acaba de mostrar se definen los nombres de las variables a mostrar para que aparezcan en la leyenda, los colores que tendrá cada variable en la gráfica, los valores límites de los ejes de representación y el modo de arrastre y zoom mediante ratón deseados.

A.1.3 Función "runSim"

Esta función contiene todo el núcleo del simulador y se presenta a continuación. Conviene recordar que, junto con la función "updateplot", esta función se ejecuta cada 5 milisegundos.

```
void MainWindow::runSim()
{

    if (ui->START_checkBox->checkState())
    {
        dtime = 0.001F * ms;
        time += dtime;

        ////////////////////
        ////// INPUTS //////
```

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

raw_ailerons = 0.01F * ui->daStick->value() - 0.5F; // from -
0.5 to 0.5
raw_elevator = 0.01F * ui->deStick->value() - 0.5F; // from -
0.5 to 0.5
raw_rudder = 0.01F * ui->drStick->value() - 0.5F; // from -0.5
to 0.5
raw_throttle = 0.01F * ui->dtStick->value(); // from 0 to 1

delta_a_max = 20.0F * (float)DEG_2_RAD;
delta_a_min = -20.0F * (float)DEG_2_RAD;
delta_e_max = 20.0F * (float)DEG_2_RAD;
delta_e_min = -20.0F * (float)DEG_2_RAD;
delta_r_max = 20.0F * (float)DEG_2_RAD;
delta_r_min = -20.0F * (float)DEG_2_RAD;

delta_a_cmd = (delta_a_max-
delta_a_min)*raw_ailerons+(delta_a_min+delta_a_max)/2.0F;
delta_e_cmd = (delta_e_max-
delta_e_min)*raw_elevator+(delta_e_min+delta_e_max)/2.0F;
delta_r_cmd = (delta_r_max-
delta_r_min)*raw_rudder+(delta_r_min+delta_r_max)/2.0F;
delta_t_cmd = raw_throttle;

h_cmd = 50.0F;//ui->hStick->value(); // commanded altitude in
m

//////////////////////
////// CONTROL //////
//////////////////////

if (ui->import_from_txt->isChecked())
{
    static int counter = 0;
    static int vecpos = 0;
    counter++;
    delta_e = elevator_txt[vecpos]/180*3.1415926538F;
    delta_a = ailerons_txt[vecpos]/180*3.1415926538F;
    delta_r = rudder_txt[vecpos]/180*3.1415926538F;
    delta_t = throttle_txt[vecpos];
    if (counter > 2)
    {
        vecpos += 5;
        counter = 0;
    }
}
else if (ui->STAILER_checkBox->isChecked())
{
    static float u_before = u;
    static float alpha_before = alpha;
    static float q_before = q;
    static float pitch_before = pitch;
    static float beta_before = beta;
    static float p_before = p;
    static float roll_before = roll;
    static float r_before = r;
    static float u_middle = u;
    static float alpha_middle = alpha;
    static float q_middle = q;
    static float pitch_middle = pitch;
```

```
        static float beta_middle = beta;
        static float p_middle = p;
        static float roll_middle = roll;
        static float r_middle = r;
        static int lin_counter = 1;
        static int lin_switch = 0;
        static int number = 100; // defines number of iterations
        before linearizing
        static float lin_magnitude = 5.0F*DEG_2_RAD; // defines
        magnitude of step for linearization
        if (lin_counter == number & lin_switch == 0)
        {
            u_middle = u;
            alpha_middle = alpha;
            q_middle = q;
            pitch_middle = pitch;
            beta_middle = beta;
            p_middle = p;
            roll_middle = roll;
            r_middle = r;
            delta_a += lin_magnitude;
        }
        if (lin_counter == 2*number & lin_switch == 0)
        {
            float B_beta = (beta-
             2*beta_middle+beta_before)/(dtime*(number)); float B_p
             = (p-2*p_middle+p_before)/(dtime*(number));
            float B_roll = (roll-
             2*roll_middle+roll_before)/(dtime*(number)); float B_r
             = (r-2*r_middle+r_before)/(dtime*(number));
            lin_switch = 1;
            std::cout << B_beta*(1/lin_magnitude) << ' ' <<
             B_p*(1/lin_magnitude) << ' ' <<
             B_roll*(1/lin_magnitude) << ' ' <<
             B_r*(1/lin_magnitude) << std::endl;
        }
        if (lin_switch == 0)
        {
            lin_counter++;
        }
    }
    Omitted code
    else
    {
        eq_factor = (V_cruise/V_total)*(V_cruise/V_total);
        static int altitude_hold = 0;

        if (altitude_hold == 0) // Altitude-hold off
        {
            // Elevator
            pitch_trim = pitch0 + K1t/(cos(roll)*V_total*V_total);
            de_trim = K2t/(cos(roll)*V_total*V_total) +
             K3t*pitch_trim + de0_trim;
            delta_e_cmd_lpf = (samples-
             1.0F)/samples*delta_e_cmd_lpf+1.0F/samples*(delta_e_cm
             d-pitch);
            delta_e = Kth*((delta_e_cmd-
             pitch)+Kth1*delta_e_cmd_lpf)+Kq*(0.0F-q)+de_trim;
            delta_e = delta_e * eq_factor;
        }
        else if (altitude_hold == 1) // Altitude-hold on
```

```
    {
        // Elevator
        pitch_trim = pitch0 + K1t/(cos(roll)*V_total*V_total);
        de_trim = K2t/(cos(roll)*V_total*V_total) +
         K3t*pitch_trim + de0_trim;
        h_cmd_lpf = (samples-
         1.0F)/samples*h_cmd_lpf+1.0F/samples*(h_cmd-h);
        delta_e = Kh*((h_cmd-h)+K1h*h_cmd_lpf)+pitch_trim;
    }
    else if (altitude_hold == 2)
    {
        static float x = 0.0F;
        static float y = 0.0F;
        float x_prev = x;
        x = (h_cmd-h)*3.3F;
        float y_prev = y;
        y = ((3/dtime)*y_prev+(1/dtime+0.3F)*x-
        (1/dtime)*x_prev)/((3/dtime)+7.2F);
        delta_e = -(0.3*(y-pitch)-0.3F*q);
    }

    // Ailerons
    delta_a_cmd_lpf = (samples-
    1.0F)/samples*delta_a_cmd_lpf+1.0F/samples*(delta_a_cmd-
    roll);
    delta_a = Kph*((delta_a_cmd-
    roll)+Kph1*delta_a_cmd_lpf)+Kp*(0.0F-p);
    delta_a = delta_a * eq_factor;

    // Throttle
    delta_t = delta_t_cmd;

    // Rudder
    delta_r = delta_r_cmd;
}

//////////////////////////////////////////
////// FORCES AND MOMENTS COMPUTATION //////
//////////////////////////////////////////

T = limit(delta_t*(Tmax-V_total*Tmax/Vmax),Tmax,0.0F); //
Engine thrust
Fx_gravity = -g*m*sin(pitch);
Fy_gravity = g*m*sin(roll)*cos(pitch);
Fz_gravity = g*m*cos(roll)*cos(pitch);
Fx = -dynamic_pressure*Sw*(cd0+cda2*alpha*alpha)+T+Fx_gravity;
Fy = dynamic_pressure*Sw*(cyb*beta+cydr*delta_r)+Fy_gravity;
Fz = -dynamic_pressure*Sw*(cl0+cla*alpha+clq*q)+Fz_gravity;
L =
dynamic_pressure*Sw*b*(cllb*beta+cllda*delta_a+clldr*delta_r+(
b/(2*V_total))*(cllp*p+cllr*r));
M =
dynamic_pressure*Sw*Sw/b*(cm0+cma*alpha+cmde*delta_e+(Sw/(2*b*
V_total))*(cmq*q+cmap*alpha_dot));
N =
dynamic_pressure*Sw*b*(cnb*beta+cnda*delta_a+cndr*delta_r+(b/(
2*V_total))*(cnp*p+cnr*r));

//////////////////////////////////
////// STATES COMPUTATION //////
//////////////////////////////////
```

```
pdot = (Ixz*(Ix-Iy+Iz)*p*q-(Iz*(Iz-
Iy)+Ixz*Ixz)*q*r+Iz*L+Ixz*N)/(Ix*Iz-Ixz*Ixz);
qdot = ((Iz-Ix)*p*r-Ixz*(p*p-r*r)+M)/Iy;
rdot = (((Ix-Iz)*Ix+Ixz*Ixz)*p*q-Ixz*(Ix-
Iy+Iz)*q*r+Ixz*L+Ix*N)/(Ix*Iz-Ixz*Ixz);
////////////////
p = p+dtime*pdot;
q = q+dtime*qdot;
r = r+dtime*rdot;
////////////////
roll_rate =
p+q*(sin(roll)*tan(pitch))+r*(cos(roll)*tan(pitch));
pitch_rate = q*cos(roll)-r*sin(roll);
yaw_rate = q*(sin(roll)/cos(pitch))+r*(cos(roll)/cos(pitch));
////////////////
pitch = pitch+dtime*pitch_rate;
roll = roll+dtime*roll_rate;
yaw = yaw+dtime*yaw_rate;
////////////////
udot = r*v-q*w-g*sin(pitch)+Fx/m;
vdot = -r*u+p*w+g*sin(roll)*cos(pitch)+Fy/m;
wdot = q*u-p*v+g*cos(roll)*cos(pitch)+Fz/m;
////////////////
u = u+dtime*udot;
v = v+dtime*vdot;
w = w+dtime*wdot;
////////////////
posNdot = u*cos(pitch)*cos(yaw)+v*(-
cos(roll)*sin(yaw)+sin(roll)*sin(pitch)*cos(yaw))+w*(sin(roll)
*sin(yaw)+cos(roll)*sin(pitch)*cos(yaw));
posEdot =
u*cos(pitch)*sin(yaw)+v*(cos(roll)*cos(yaw)+sin(roll)*sin(pitc
h)*sin(yaw))+w*(-
sin(roll)*cos(yaw)+cos(roll)*sin(pitch)*sin(yaw));
hdot = u*sin(pitch)-v*sin(roll)*cos(pitch)-
w*cos(roll)*cos(pitch);
////////////////
posN_old = posN;
posE_old = posE;
h_old = h;
////////////////
posN = posN+dtime*posNdot;
posE = posE+dtime*posEdot;
h = h+dtime*hdot;
////////////////
// MATLAB® documentation "Flat Earth to LLA" (from North, East
coordinates)
RN = R/sqrt(1.0F-(2.0F*f-f*f)*sin(latitude)*sin(latitude));
RM = RN*(1.0F-(2.0F*f-f*f))/(1-(2.0F*f-
f*f)*sin(latitude)*sin(latitude));
latitude = latitude + atan(1.0F/RM)*(posN-posN_old);
longitude = longitude + atan(1.0F/(RN*cos(latitude)))*(posE-
posE_old);
altitude = h;
////////////////
V_total = sqrt(u*u+v*v+w*w);
alpha_old = alpha;
alpha = atan(w/u);
alpha_dot = (alpha_old-alpha)/dtime;
beta = asin(v/V_total);
```

```
            atm_temperature = 288.15F-(6.5F*altitude/1000.0F);
            atm_pressure = 101325.0F*pow(1.0F-
             (0.0065F*altitude/288.15F),5.2561F);
            atm_density = atm_pressure/(atm_temperature*287.04F);
            dynamic_pressure = 0.5*atm_density*V_total*V_total;
        }
}
```

A continuación se explicará a grandes rasgos el funcionamiento de cada una de las secciones de esta función en el orden en que se ejecutan.

Para empezar se ha programado una estructura "if" que permite comenzar y parar la simulación a través del botón "START SIM" que se encuentra en la interfaz. Acto seguido, se procede al cálculo de la variable "time" que contiene el tiempo transcurrido de simulación al sumar su valor al tiempo de muestreo una vez cada ejecución.

A continuación se leen los comandos de input, que se introducen en la interfaz a través de deslizadores. Estos valores de input se transforman para obtener valores comandados en unidades de radianes de las superficies de control.

Seguidamente se procede a la sección del control de la aeronave. Existen tres modos de funcionamiento distintos para esta sección.

(1) Declaración "**if**": En primer lugar se ha desarrollado el código que introduce comandos en las superficies de control correspondientes a los medidos en las superficies reales durante las pruebas de vuelo. Estos datos han sido almacenados previamente en "arrays" a partir del archivo txt al ejecutar la función "initSim". Hay que tener en cuenta que si la frecuencia de ejecución del proceso de medida difiere de la del simulador es necesario corregir este desfase. Esta secuencia de código se ejecuta si el botón "IMPORT INPUTS FROM TXT" de la interfaz se encuentra activo.

(2) Declaraciones "**else if**": A continuación existe una sucesión de estructuras que se encargan de introducir cambios bruscos (señales escalón) en los comandos y en los estados. Observar la respuesta de la aeronave a una entrada escalón en alguna de sus superficies de control o en alguno de sus estados es un proceso muy útil para realizar una linealización del modelo en una condición de vuelo determinada. Para realizar este proceso, es recomendable que la aeronave se encuentre en una condición de vuelo estacionaria.

    El código tiene además capacidad de realizar dicha linearización calculando en qué cantidad cambian los estados de la aeronave tras realizar el cambio brusco. De esta manera se puede obtener un modelo lineal mediante la representación de espacio de estados. Dado que todas las estructuras de "step" son similares, se han omitido y se ha mantenido una de ellas para disminuir el espacio ocupado por el código en el informe. Estas secuencias de código se ejecutan si el botón "Step _" correspondiente se encuentra activo en la interfaz.

(3) Declaración "**else**": Esta estructura contiene el sistema de control normal de la aeronave, incluyendo sus autopilotos y sus modos nominales. Está activa por defecto si ningún botón de la interfaz está activado (excepto el botón "START SIM", que inicia el simulador, y el botón "SHOW GRAPH", que permite la visualización de estados en forma gráfica, como se verá en el apartado B.1.4).

A continuación se procede al cálculo de fuerzas aerodinámicas y gravitatorias y momentos aerodinámicos de la aeronave. Este constituye el núcleo del modelo dinámico del avión. Seguidamente se calculan los estados de la aeronave tras aplicar las fuerzas y momentos resultantes.

A.1.4 Función "updateplot"

La función "updateplot" se encarga de actualizar las gráficas de la interfaz con los valores obtenidos de la función "runSim". Dado que esta función se ejecuta a una frecuencia alta, al igual que la función "runSim", se consigue un efecto de movimiento en tiempo real de las gráficas (como se verá más adelante las gráficas no se actualizan cada 5 milisegundos sino cada 45 milisegundos). Además, el usuario dispone de herramientas de zoom y deslizamiento en el eje vertical incluso durante la simulación, lo que supone una característica muy útil para el visualizado.

```
void MainWindow::updateplot()
{
    if (ui->START_checkBox->checkState())
    {
        static int j = 0;
        j++;
        if (j>8)
        {
            j = 0;
            if (ui->SHOW_checkBox->checkState())
            {
                // Only update the tab selected (Makes the computation
                 faster)
                if (ui->tabWidget->currentIndex()==0)
                {
                    updateplot_V_body();
                }
                if (ui->tabWidget->currentIndex()==1)
                {
                    updateplot_Omega_body();
                }
                if (ui->tabWidget->currentIndex()==2)
                {
                    updateplot_Euler();
                }
                if (ui->tabWidget->currentIndex()==3)
                {
                    updateplot_Position();
                }
                if (ui->tabWidget->currentIndex()==4)
                {
                    updateplot_LLA();
                }
                if (ui->tabWidget->currentIndex()==5)
```

```
                {
                    updateplot_Sideslip();
                }
                if (ui->tabWidget->currentIndex()==6)
                {
                    updateplot_Commands_Actuators();
                }
                if (ui->tabWidget->currentIndex()==7)
                {
                    updateplot_Accel_body();
                }
                if (ui->tabWidget->currentIndex()==8)
                {
                    updateplot_Omegadot_body();
                }
            }
        }
    }
}
```

Al igual que la función "initSim", esta función llama a otras funciones más pequeñas que actualizan cada gráfica individualmente.

Con el fin de disminuir la carga computacional correspondiente a esta función se han tenido en cuenta dos factores:

(1) Sólo se actualizan las gráficas que actualmente están activas en la interfaz. Es decir, las correspondientes a la pestaña seleccionada actual en la interfaz.
(2) Las gráficas se actualizan una vez cada nueve ejecuciones de las funciones "runSim" y "updateplot", es decir cada 45 centésimas de segundo. Esta frecuencia de actualización es suficiente para proveer el efecto de movimiento en tiempo real de las gráficas y disminuye intensamente la carga computacional.

Dado que la estructura de las funciones "updateplot_X" es muy similar, se muestra una de ellas como ejemplo ("updateplot_Euler").

```
void MainWindow::updateplot_Euler()
{
    static int counter2 = 0;
    static int vecpos2 = 0;
    counter2++;

    ui->roll_chart->graph(0)->addData(time, roll * (float)RAD_2_DEG);
    ui->roll_chart->graph(1)->addData(time, -delta_a *
    (float)RAD_2_DEG);
    ui->roll_chart->graph(2)->addData(time, -roll_txt[vecpos2]);
    ui->roll_chart->xAxis->setRange(time+1.5, 8, Qt::AlignRight);
    ui->roll_chart->replot();

    ui->pitch_chart->graph(0)->addData(time, pitch *
    (float)RAD_2_DEG);
    ui->pitch_chart->graph(1)->addData(time, -delta_e *
    (float)RAD_2_DEG);
    ui->pitch_chart->graph(2)->addData(time, -pitch_txt[vecpos2]);
    ui->pitch_chart->xAxis->setRange(time+1.5, 8, Qt::AlignRight);
    ui->pitch_chart->replot();
```

```
    ui->yaw_chart->graph(0)->addData(time, yaw * (float)RAD_2_DEG);
    ui->yaw_chart->graph(1)->addData(time, -delta_r *
    (float)RAD_2_DEG);
    ui->yaw_chart->graph(2)->addData(time, -yaw_txt[vecpos2]);
    ui->yaw_chart->xAxis->setRange(time+1.5, 8, Qt::AlignRight);
    ui->yaw_chart->replot();

    if (ui->import_from_txt->isChecked())
    {
        if (counter2 > 2)
        {
            vecpos2 += 5;
            counter2 = 0;
        }
    }
}
```

En esta función se establecen los datos a representar, los valores límites de los ejes (al ser función del tiempo se consigue que la gráfica se actualice siempre en una línea vertical fija) y además se corrige la frecuencia de actualización de los estados reales medidos de los datos de vuelo para hacerlos compatibles en tiempo con las salidas del simulador.

A.2 Interfaz gráfica

La interfaz gráfica que se ha diseñado para el simulador en Qt es muy simple. A continuación se muestra una imagen de la interfaz tras inicializarse (Figura 105).

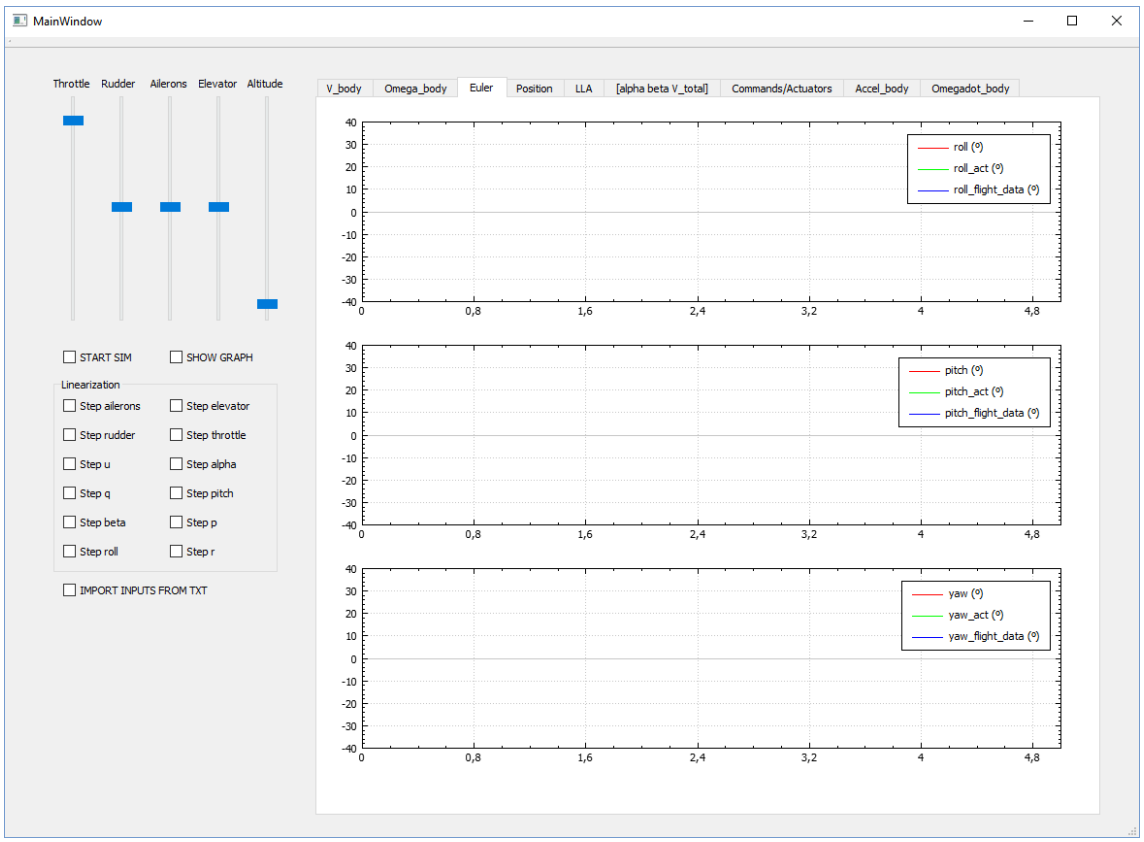

Figura 105: Interfaz gráfica del simulador en Qt

Se ha dividido la interfaz en varias secciones (Figura 106).

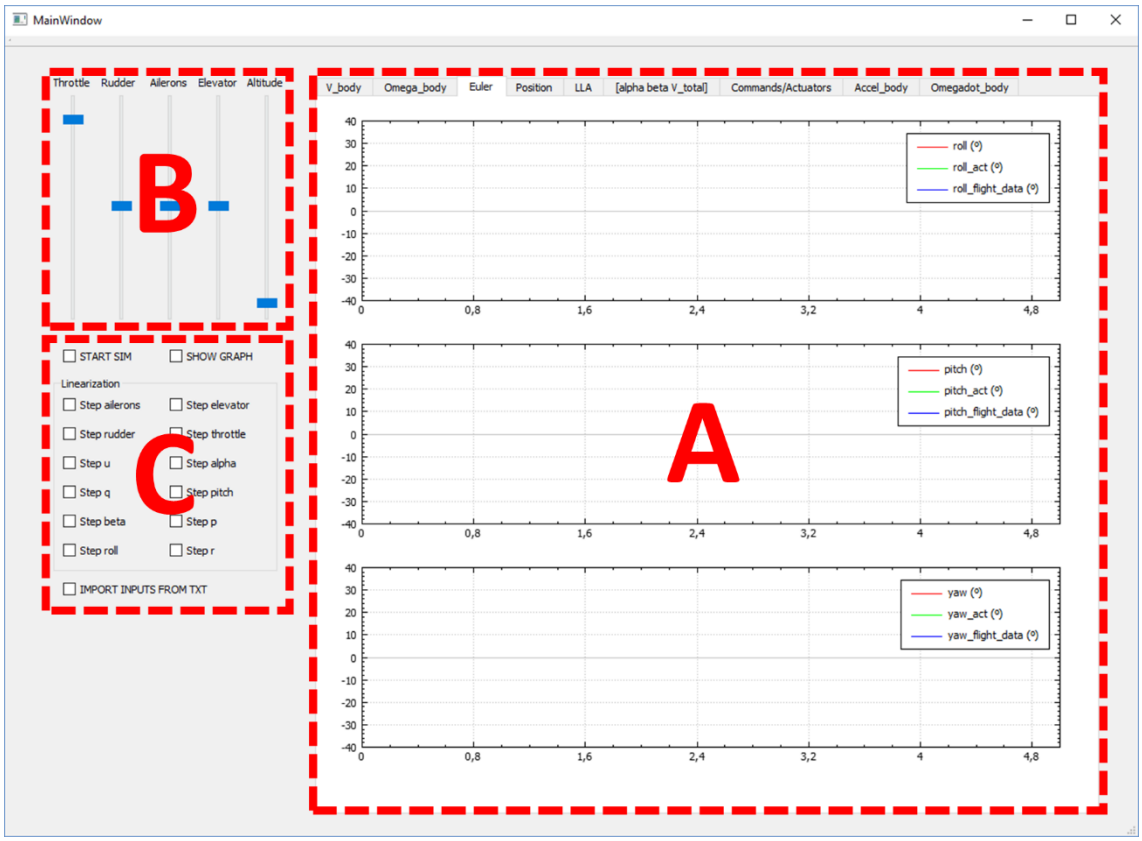

Figura 106: Secciones de la interfaz gráfica

La **sección A** corresponde al espacio donde se representan las gráficas de cada variable de estado. Los estados están agrupados en conjuntos de tres o cuatro y mediante pestañas se cambia de un grupo a otro.

En la **sección B** se pueden encontrar los deslizadores que se pueden utilizar para comandar señales de entrada de manera manual.

La **sección C** contiene los botones "START SIM" y "SHOW GRAPH" que consiguen iniciar/parar el simulador y activar/desactivar las gráficas respectivamente. Además se encuentran en esta sección los botones que permiten linealizar el sistema a base de introducir entradas de input en los comandos o en las variables de estado. Por último se puede observar el botón "IMPORT INPUTS FROM TXT", que permite realizar el ajuste del modelo dinámico, objetivo del presente simulador en Qt.

A continuación se muestra una imagen de la interfaz pausada tras haber ejecutado el simulador (Figura 107). En este caso se ha utilizado la función "IMPORT INPUTS FROM TXT" para realizar el ajuste del modelo dinámico.

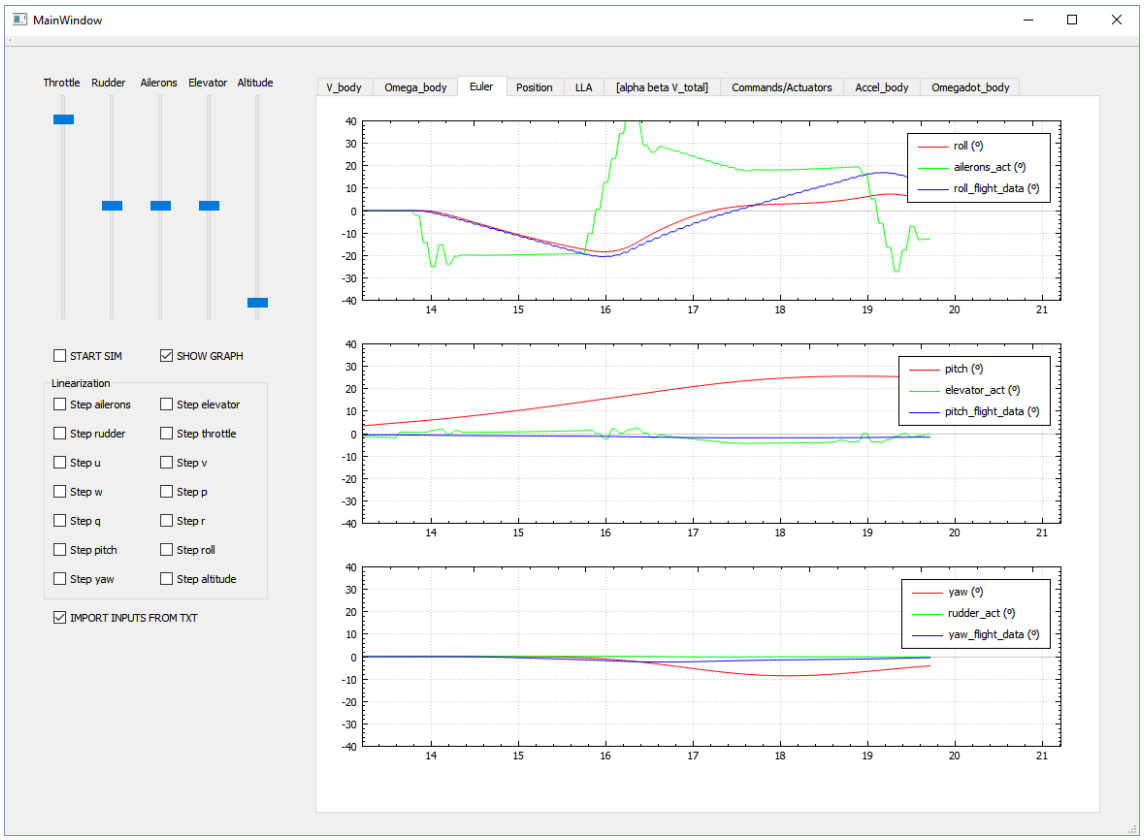

Figura 107: Interfaz del simulador en ejecución

En la figura se puede apreciar que la respuesta en roll del avión que se ha modelado en el simulador (en este caso un Boeing 767-300 que se ha utilizado como demostración) y la de su contraparte real son muy parecidas durante los primeros tres segundos. Debido a la dificultad de obtener datos de vuelo reales de este avión se han considerado como datos de vuelo reales los obtenidos a partir de una simulación profesional de un modelo previamente desarrollado utilizando otro software.

Anexo B

Manual de instalación y puesta a punto de la plataforma

En el directorio que contiene los archivos necesarios para la instalación de la plataforma se encuentran los siguientes archivos:

I. Archivo "**aircraft_model.slx**": Este archivo contiene el simulador de la plataforma (Simulink®).
II. Archivo "**asbBusDefinitionCommand.m**": Este script de MATLAB® se puede utilizar para crear un objeto de bus aplicado al bus de comandos (apartado 3.3.2). El uso de este archivo es OPCIONAL.
III. Archivo "**asbBusDefinitionEnvironment.m**": Este script de MATLAB® se puede utilizar para crear un objeto de bus aplicado al bus de ambiente (apartado 3.3.2). El uso de este archivo es OPCIONAL.
IV. Archivo "**asbBusDefinitionSensors.m**": Este script de MATLAB® se puede utilizar para crear un objeto de bus aplicado al bus de sensores (apartado 3.3.2). El uso de este archivo es OPCIONAL.
V. Archivo "**asbBusDefinitionStates.m**": Este script de MATLAB® se puede utilizar para crear un objeto de bus aplicado al bus de estados (apartado 3.3.2). El uso de este archivo es OPCIONAL.
VI. Archivo "**controlgui.fig**": Este archivo contiene información de los objetos visuales de la interfaz gráfica (MATLAB® GUIDE).
VII. Archivo "**controlgui.m**": Este script de MATLAB® contiene el código de la interfaz gráfica.
VIII. Archivo "**runfg.bat**": Este archivo corresponde al script de inicialización de FlightGear que se crea utilizando el bloque de Simulink® "Generate Run Script" (apartado 3.2.6.1).
IX. Archivo "**startVars.m**": Este script de MATLAB® es el script de inicialización del simulador (apartado 3.3.1).
X. Directorio "**Aeronave_X**": Esta carpeta contiene el modelo geométrico de la aeronave X preparado para ejecutarse en FlightGear.

A continuación se detallan los pasos que es necesario seguir para llevar a cabo la instalación y puesta a punto de la plataforma de simulación. La plataforma se ha desarrollado para funcionar en el sistema operativo Microsoft Windows.

B.1 Primer paso: Instalación de MATLAB y Simulink

El primer paso consiste en descargar e instalar la herramienta de software MATLAB® y Simulink®. El simulador se ha desarrollado para funcionar en la versión 8.6 (R2015b).

B.2 Segundo paso: Instalación de FlightGear

Seguidamente es necesario descargar e instalar el software FlightGear versión 3.4. En caso de que esta versión no esté disponible en la página oficial de FG porque exista otra versión más reciente, se puede utilizar esta última versión. Sin embargo sería necesario generar de nuevo el script de inicialización de FlightGear ("runfg.bat") estableciendo la versión de FG elegida en el bloque "Generate Run Script".

B.3 Tercer paso: Configuración de FlightGear

A continuación se procede a introducir la geometría de la aeronave X en los directorios de instalación de FlightGear. Para ello es necesario introducir el directorio "Aeronave_X" en el interior de la carpeta

```
"directorio de instalación"\FlightGear 3.4.0\data\Aircraft
```

B.4 Cuarto paso: Configuración de MATLAB

Por último se procede a crear una herramienta para facilitar la ejecución del script de inicialización de FlightGear en caso de que la interfaz gráfica no se encuentre abierta. En el entorno de MATLAB®, dentro de la pestaña "HOME", hay que pulsar en "New" y seguidamente "Command Shortcut" (Figura 108).

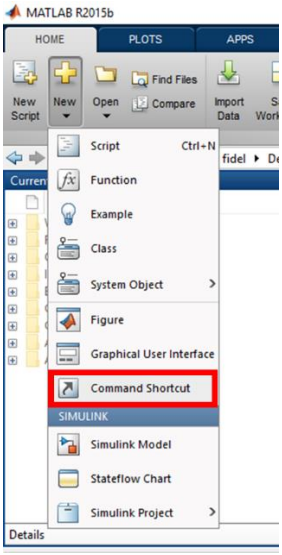

Figura 108: Ubicación del editor de accesos directos

Seguidamente se introduce la siguiente información (Figura 109).

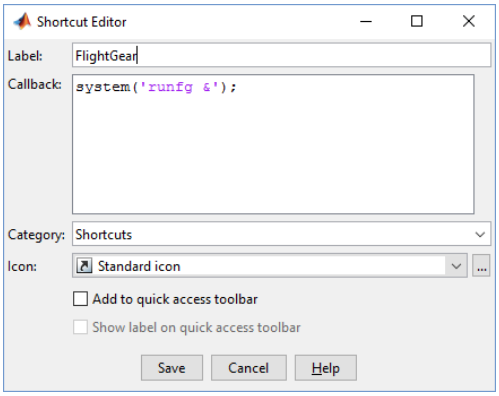

Figura 109: Editor de accesos directos

De esta manera se crea un acceso directo en el entorno de MATLAB® (dentro de la pestaña "SHORTCUTS") que si es pulsado ejecutará el script de inicialización "runfg.bat" e iniciará FlightGear.

De la misma manera, se crea un segundo acceso directo, esta vez introduciendo la siguiente información (Figura 110).

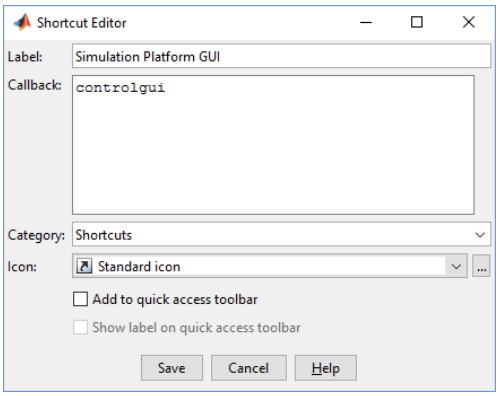

Figura 110: Editor de accesos directos

Al pulsar en este acceso directo se iniciaría la interfaz gráfica de la plataforma en caso de que no se encontrara abierta.

B.5 Quinto paso: Verificación

Finalmente se debe asegurar que los archivos I-IX (los nueve primeros) se encuentran ubicados en el mismo directorio y que se dispone de un Joystick conectado al equipo (generalmente no es necesario ningún proceso de instalación). Por último se puede proceder a abrir el modelo "aircraft_model.slx" mediante Simulink®. Automáticamente se abre el modelo del simulador y la interfaz gráfica, desde la cual se puede inicializar el simulador pulsando el botón "INITIALIZE MODEL", abrir FG pulsando el botón "OPEN FLIGHTGEAR" e inicializar el simulador pulsando el botón "RUN". En este punto la plataforma de simulación comienza a ejecutarse.