Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Modelado y simulación del Skywalker X8

Autor: Alejandro Rísquez Ruiz
Tutor: Eduardo Fernández Camacho

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

Sevilla, 2017

Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Modelado y simulación del Skywalker X8

Autor:
Alejandro Rísquez Ruiz

Tutor:
Eduardo Fernández Camacho
Catedrático de Universidad

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Trabajo Fin de Grado: Modelado y simulación del Skywalker X8

Autor: Alejandro Rísquez Ruiz

Tutor: Eduardo Fernández Camacho

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

A Cecilia, Sergio y Manuel.

Agradecimientos

En primer lugar, me gustaría agradecer a mi tutor, Eduardo, por haberme concedido el privilegio de trabajar en este proyecto. Sin su orientación y paciencia, el camino hasta llegar aquí habría sido infinitamente más arduo.

En segundo lugar, quisiera dar las gracias a todos los profesores de los cuales he tenido la suerte de ser alumno a lo largo de mi vida. Pues su labor, imprescindible, ayuda a labrar el futuro de las personas, siendo una pieza fundamental del desarrollo humano.

Por último, pero no menos importante, mostrar mis más profundos agradecimientos a todas aquellas personas que han creído en mí y me han apoyado durante estos años.

Alejandro Rísquez Ruiz

Sevilla, 2017

Resumen

Este documento recoge el desarrollo de un modelo dinámico de un vehículo aéreo no tripulado de tipo ala volante, concretamente del Skywalker X8, así como su posterior implementación en un simulador de vuelo, donde se han construido los módulos de guiado, navegación y control típicos de una aeronave de estas características.

La documentación se divide en dos capítulos fundamentales. El primero, dedicado a un análisis en profundidad del modelo, donde se discute el trabajo de investigación previo llevado a cabo, los datos utilizados y el método seguido para la obtención de la ley de propulsión; el segundo, destinado a resumir el contenido de los bloques principales del simulador además de comentar algunos resultados obtenidos. No obstante, también se ha añadido un capítulo donde se exponen las principales limitaciones del sistema al igual que posibles mejoras futuras a implementar. Por último, se concluye con un anexo que contiene una guía de uso y puesta a punto del simulador, creado en Simulink; un tutorial de instalación de FlightGear, el cual ha servido de interfaz gráfica para el simulador; y el código de Matlab producido e implementado tanto en el apartado de modelado como en el de simulación.

Índice

Índice de tablas

Índice de figuras

Notación

a	Radio ecuatorial del modelo WGS84.
a_*	Aceleración lineal de la aeronave.
a_{ϕ_*}	Constante de la función de transferencia asociada con la dinámica del ángulo de balanceo.
a_{θ_*}	Constante de la función de transferencia asociada con la dinámica del ángulo de cabeceo.
a_{V_*}	Constante de la función de transferencia asociada con la dinámica de la velocidad aerodinámica.
AR	Ratio de aspecto.
α	Ángulo de ataque.
α_0	Ángulo de ataque de entrada en pérdida.
b	Envergadura.
β	Ángulo de deslizamiento.
c	Cuerda media del perfil aerodinámico del ala.
c	Radio polar del modelo WGS84.
C_{D_*}	Coeficiente de resistencia aerodinámica.
C_{l_*}	Coeficiente de momento aerodinámico alrededor del eje longitudinal de la aeronave.
C_{L_*}	Coeficiente de sustentación aerodinámica.
C_{m_*}	Coeficiente de momento aerodinámico alrededor del eje transversal de la aeronave.
C_{n_*}	Coeficiente de momento aerodinámico alrededor del eje vertical de la aeronave.
C_X	Coeficiente de fuerza aerodinámica a lo largo del eje longitudinal de la aeronave.
C_Y	Coeficiente de fuerza aerodinámica a lo largo del eje transversal de la aeronave.
C_Z	Coeficiente de fuerza aerodinámica a lo largo del eje vertical de la aeronave.
χ	Ángulo de curso.
χ_c	Ángulo de curso de referencia.
χ_d	Ángulo de curso deseado para la intersección del segmento de ruta.
χ_q	Ángulo de curso del segmento de ruta.
d_χ	Señal de perturbación asociada al modelo lineal del curso.
d_h	Señal de perturbación asociada al modelo lineal de la altitud.
d_{ϕ_*}	Señal de perturbación asociada al modelo lineal del ángulo de balanceo.
d_{θ_*}	Señal de perturbación asociada al modelo lineal del ángulo de cabeceo.
d_{V_*}	Señal de perturbación asociada al modelo lineal de la velocidad aerodinámica.

δ_a	Deflexión de los elevones actuando como alerones (en sentidos opuestos).
δ_e	Deflexión de los elevones actuando como elevadores (en el mismo sentido).
δ_t	Fuerza propulsiva generada por el motor de la aeronave.
e	Coeficiente de Oswald.
$\mathbf{e}_p$	Error de seguimiento de ruta.
e_{py}	Error lateral de seguimiento de ruta.
F_*	Fuerza ejercida sobre la aeronave.
g	Aceleración gravitatoria.
G	Constante de gravitación universal.
Γ_*	Productos de la matriz de inercia.
h	Altitud.
h_c	Altitud de referencia.
H_*	Función de transferencia del modelo Dryden de viento.
$\mathcal{H}(\mathbf{w},\mathbf{n})$	Semiplano definido por el punto de paso **w** y el vector normal **n**.
$I_{*(*)}$	Momento (producto) de inercia de la aeronave.
J_2	Factor de forma dinámica del modelo WGS84.
k_{d_*}	Ganancia derivativa del controlador PID.
k_{i_*}	Ganancia integral del controlador PID.
k_{p_*}	Ganancia proporcional del controlador PID.
k_{path}	Ganancia de control para el seguimiento del segmento de ruta.
$k_{\theta_{DC}}$	Ganancia DC de la función de transferencia de la posición del elevador al ángulo de cabeceo.
L	Momento de las fuerzas externas alrededor del eje longitudinal de la aeronave.
L_*	Longitud espacial de onda en el modelo Dryden de viento.
λ	Longitud en el sistema de referencia ECEF.
m	Masa del Skywalker X8.
M	Momento de las fuerzas externas alrededor del eje transversal de la aeronave.
M_e	Masa de la Tierra.
N	Momento de las fuerzas externas alrededor del eje vertical de la aeronave.
ω_e	Velocidad de rotación de la Tierra.
ω_{n_*}	Frecuencia natural.
ω_*	Velocidad angular.
P	Velocidad de rotación alrededor del eje longitudinal de la aeronave.
p_d	Componente vertical de la posición en el sistema de referencia NED.
p_e	Componente este de la posición en el sistema de referencia NED.
p_n	Componente norte de la posición en el sistema de referencia NED.
$\mathcal{P}_{line}(\mathbf{r},\mathbf{q})$	Segmento de ruta definido por el vector de referencia **r** y el vector normal **q**.
ϕ	Ángulo de balanceo.
φ	Latitud en el sistema de referencia ECEF.
ψ	Ángulo de guiñada.

Q	Velocidad de rotación alrededor del eje transversal de la aeronave.
r	Radio vector de posición.
R	Velocidad de rotación alrededor del eje vertical de la aeronave.
σ	Coeficiente aerodinámico sigma.
σ_*	Intensidad de turbulencia en el modelo Dryden de viento.
ρ	Densidad del aire.
$\boldsymbol{R_A^B}$	Matriz de rotación del sistema de referencia A al B.
S_{alar}	Superficie alar.
θ	Ángulo de cabeceo.
U	Componente longitudinal de la velocidad de la aeronave respecto a tierra.
U_a	Componente longitudinal de la velocidad aerodinámica de la aeronave.
U_∞	Velocidad aerodinámica total de la aeronave.
U_w	Componente longitudinal de la velocidad de la masa de aire respecto a tierra.
V	Componente transversal de la velocidad de la aeronave respecto a tierra.
V_a	Componente transversal de la velocidad aerodinámica de la aeronave.
V_w	Componente transversal de la velocidad de la masa de aire respecto a tierra.
W	Componente vertical de la velocidad de la aeronave respecto a tierra.
W_a	Componente vertical de la velocidad aerodinámica de la aeronave.
$\mathbf{w}_i$	Punto de paso i-ésimo.
W_w	Componente vertical de la velocidad de la masa de aire respecto a tierra.
ζ_*	Coeficiente de amortiguamiento.

1 INTRODUCCIÓN

En los últimos tiempos se ha evidenciado un aumento progresivo del uso de vehículos aéreos no tripulados, para labores muy diversas, por ejemplo: agricultura, defensa, medio ambiente o seguridad. Debido a la falta de consenso, han aparecido diferentes denominaciones para este tipo de vehículos que, aunque algunos expertos reseñan ligeras diferencias entre ellos, se utilizan indistintamente debido a que se refieren esencialmente a lo mismo. De este modo, han surgido términos como UAV (*Unmanned Aerial Vehicle*), UAS (*Unmanned Aerial System*), RPA (*Remotely Piloted Aircraft*) o RPAS (*Remotely Piloted Aerial System*). Sin embargo, el término genérico popularmente más aceptado es *drone*.

El avance de la tecnología de ámbito comercial de la cual se nutren la gran mayoría de estos vehículos junto con un abaratamiento de los costes de adquisición, ha fomentado su aparición no sólo en el sector público sino también privado, debido a una mayor accesibilidad por parte de empresas y particulares. Todo ello augura un creciente protagonismo de los mismos en las sociedades tecnológicamente más desarrolladas, donde es previsible su utilización en tareas cada vez más dispares.

El objetivo de este documento es la construcción de un modelo matemático que defina, con la mayor fidelidad posible, el comportamiento dinámico del Skywalker X8 además de su ulterior implementación en un simulador de vuelo. Con esta finalidad, se tendrá como único recurso los datos de telemetría obtenidos a lo largo de cinco vuelos de prueba, descritos en Fernández Rojas, B. (2014). *Desarrollo de un avión no tripulado: construcción, integración de sistemas y ensayos en vuelo*. Proyecto Fin de Carrera. Universidad de Sevilla. Sevilla, España.

Las herramientas utilizadas han sido exclusivamente: Mission Planner, Matlab, Simulink y FlightGear. Sin embargo, aparte de la telemetría recabada, no ha sido posible realizar un estudio experimental (aerodinámico y propulsivo) de la aeronave, por lo que se ha tenido que recurrir a investigaciones externas, donde sí se ha podido realizar estudios experimentales del Skywalker X8, con el fin de conseguir datos indispensables para un modelado preciso. Este ha sido el caso de los coeficientes aerodinámicos, los cuales se han obtenido de la investigacion previa descrita en Gryte, K. (2015). *High angle of attack landing of an unmanned aerial vehicle.* MSc Tesis. Norwegian University of Science and Technology. Trondheim, Norway.

Finalmente, añadir que el valor de este documento es meramente educacional pues, como se verá más adelante, un modelo fiel a la realidad requiere indispensablemente de un estudio experimental preliminar y de pruebas posteriores para su correcta afinación, así como de datos de telemetría completos y rigurosos.

2 MODELADO

Este primer capítulo se ha dedicado a la presentación del proyecto de referencia junto con los datos de partida así como a sentar las bases del modelo dinámico que se ha utilizado en el simulador, incluyendo todo el procedimiento seguido para el cálculo de la ley propulsiva a partir de los mismos.

2.1. Punto de partida

Este proyecto es una continuación de Fernández Rojas, B. (2014). *Desarrollo de un avión no tripulado: construcción, integración de sistemas y ensayos en vuelo*. Proyecto Fin de Carrera. Universidad de Sevilla. Sevilla, España. En él, se abarca todo el proceso desde la elección de la aeronave hasta su ensayo en vuelo.

Como se ha mencionado con anterioridad, el *drone* seleccionado fue el modelo Skywalker X8, de tipo ala volante, cuyas características físicas generales son las siguientes:

- Envergadura: 2120 mm.
- Longitud: 780 mm.
- Superficie alar: 74.28 dm^2.
- Máximo peso en vuelo: 3500 g.

Figura 2-1. Skywalker X8.

A la aeronave se le integró multitud de componentes electromecánicos, los cuales no son objeto de análisis en este proyecto debido a que su descripción en profundidad se encuentra en su proyecto original, pero sí de mención:

- Baterías LiPo de 2, 4 y 5 celdas con 2500, 4000 y 5800 mAh de capacidad respectivamente.
- Circuito de eliminación de batería o BEC (*Battery Elimination Circuit*).
- Servomotores (tanto para la deflexión de los elevones como de la rueda de morro).
- Controladores electrónicos de velocidad o ESC (*Electronic Speed Controller*).
- Motor de la hélice.
- Módulo de potencia del sensor o PSM (*Power Sensor Module*).
- GPS.
- Magnetómetro.
- Tubo de Pitot.
- Ardupilot.
- Raspberry Pi.
- Cámara.
- Antena.
- Receptor.
- Transmisor.

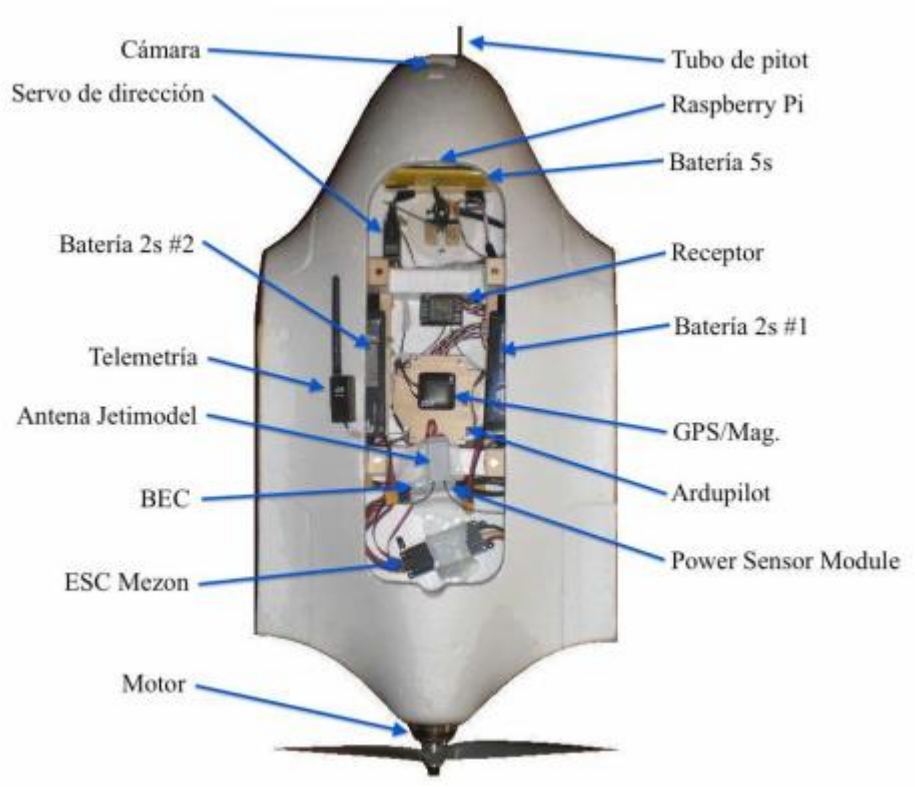

Figura 2-2. Distribución interior del Skywalker X8.

La integración de todos estos subsistemas junto a sus respectivos sensores hizo posible que se recabaran datos de telemetría de los cinco vuelos de prueba que se llevaron a cabo. Esta telemetría además de las características técnicas del UAV han sido la única fuente de información que se ha reciclado del proyecto original para construir el modelo dinámico del Skywalker X8, es decir, no se ha tenido contacto directo con el mismo en ningún momento a lo largo del trabajo.

Los vuelos se han podido simular utilizando Mission Planner, una aplicación de código abierto sólo disponible para Windows que se utiliza como estación de tierra. Esto ha permitido seguir el progreso en diferido de los distintos vuelos que se llevaron a cabo así como transformar los archivos de telemetría con extensión *.tlog* utilizados por esta aplicación a archivos de datos *.mat* utilizados por Matlab.

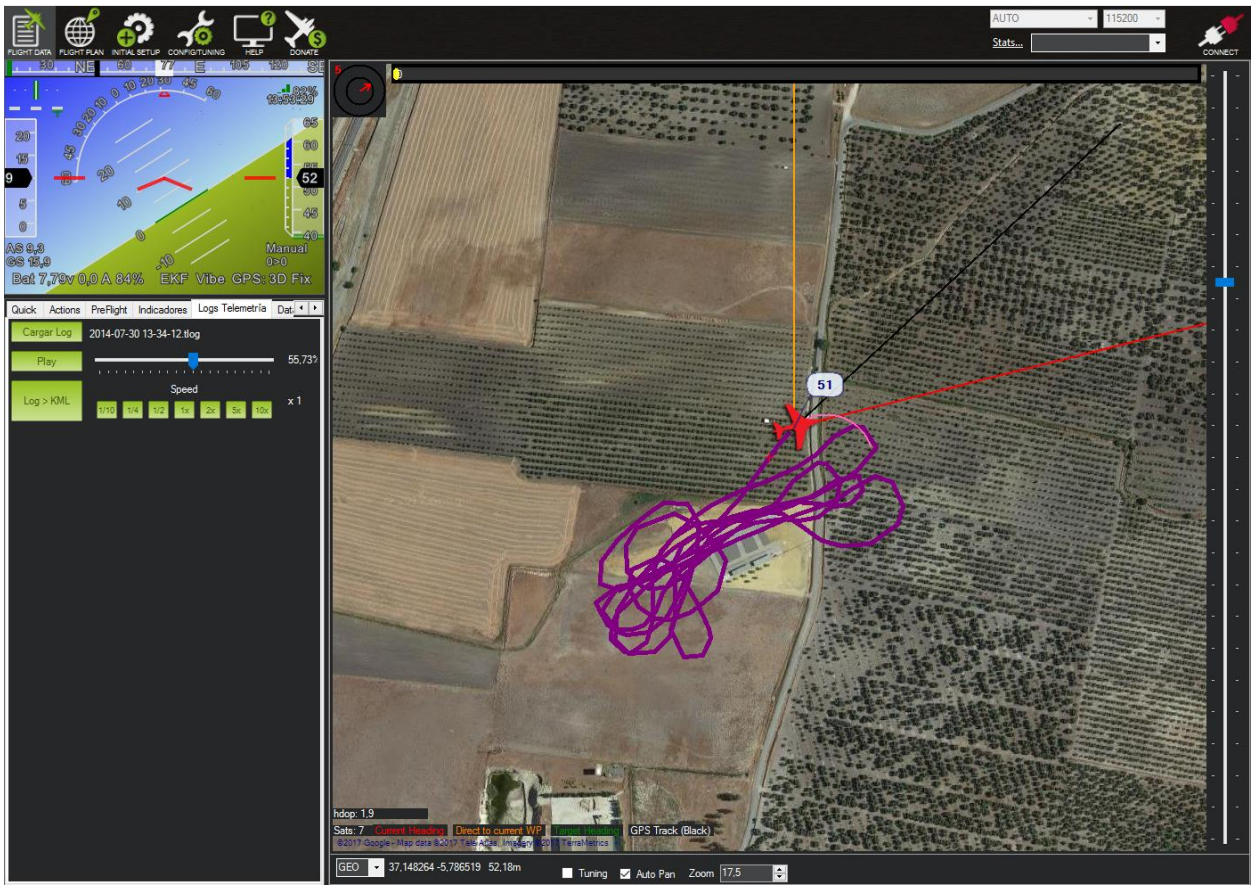

Figura 2-3. Simulación del quinto vuelo en Mission Planner.

A continuación, se muestra una serie de gráficas de la telemetría del quinto y último vuelo (el más largo) en las que se puede visualizar aquellos datos de interés que se han utilizado para la construcción del modelo:

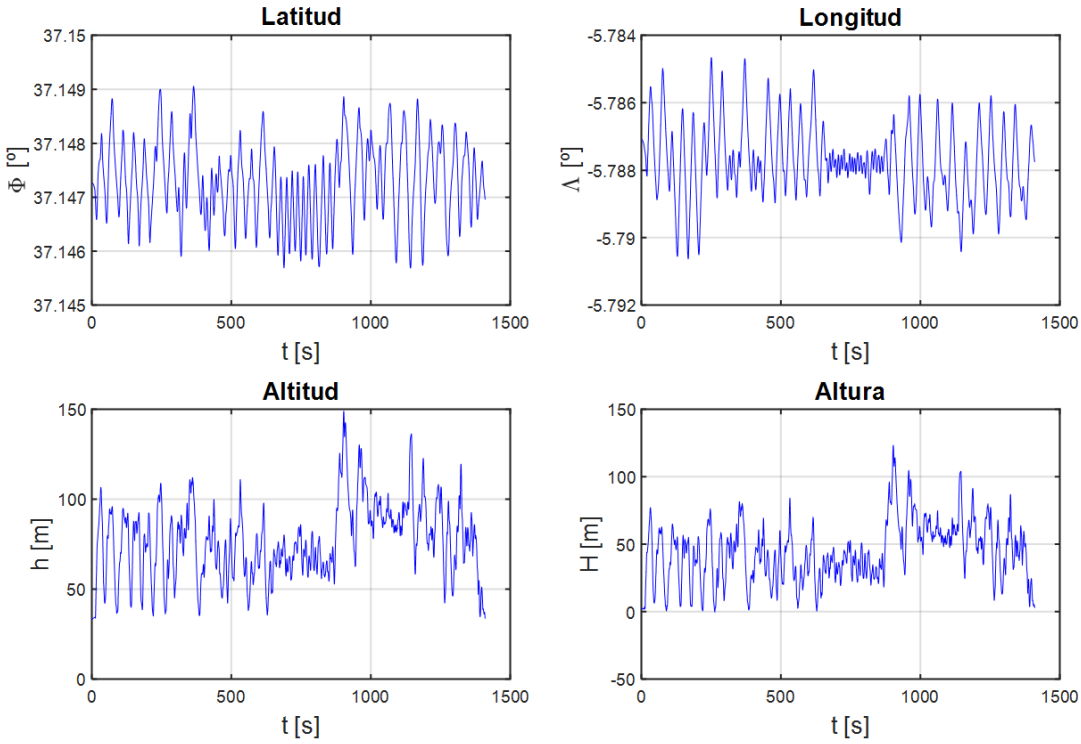

Figura 2-4. Latitud, longitud, altitud y altura.

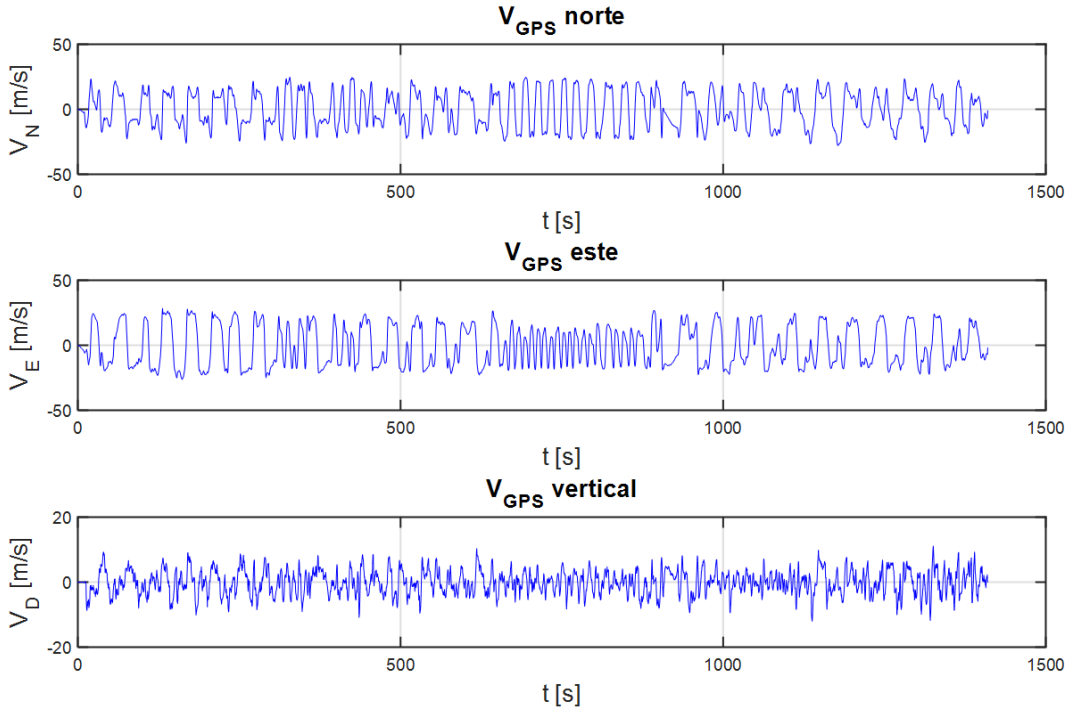

Figura 2-5. Componentes norte, este y vertical de la velocidad respecto a tierra en ejes de navegación.

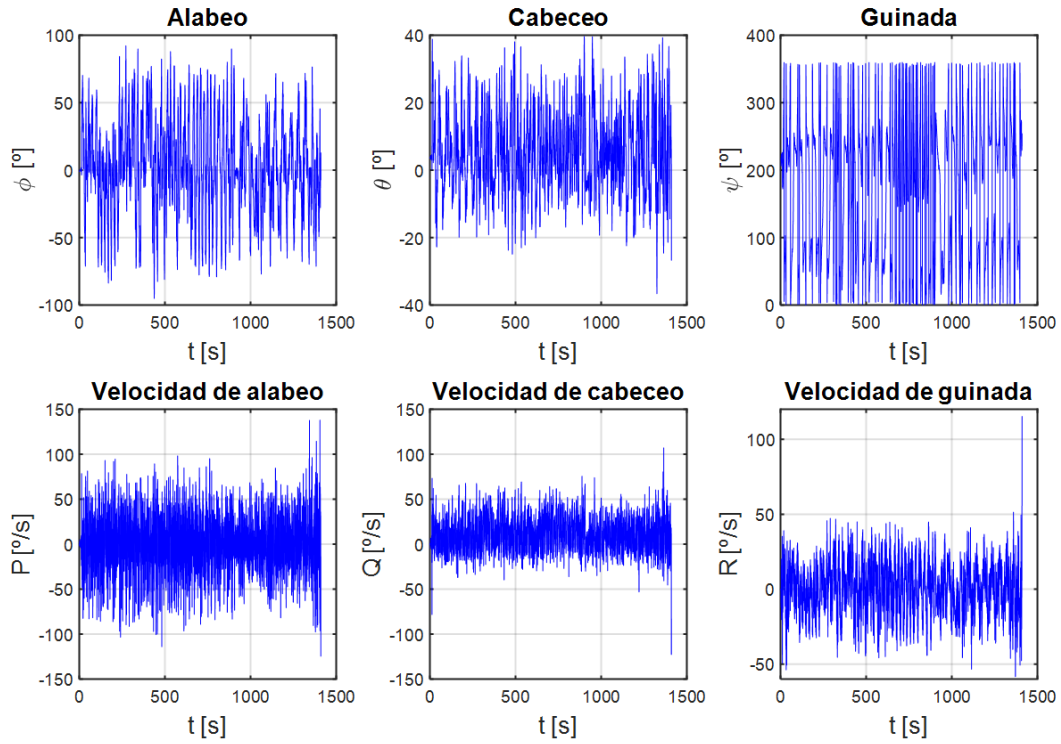

Figura 2-6. Ángulos de Euler y velocidades de rotación de los ejes cuerpo.

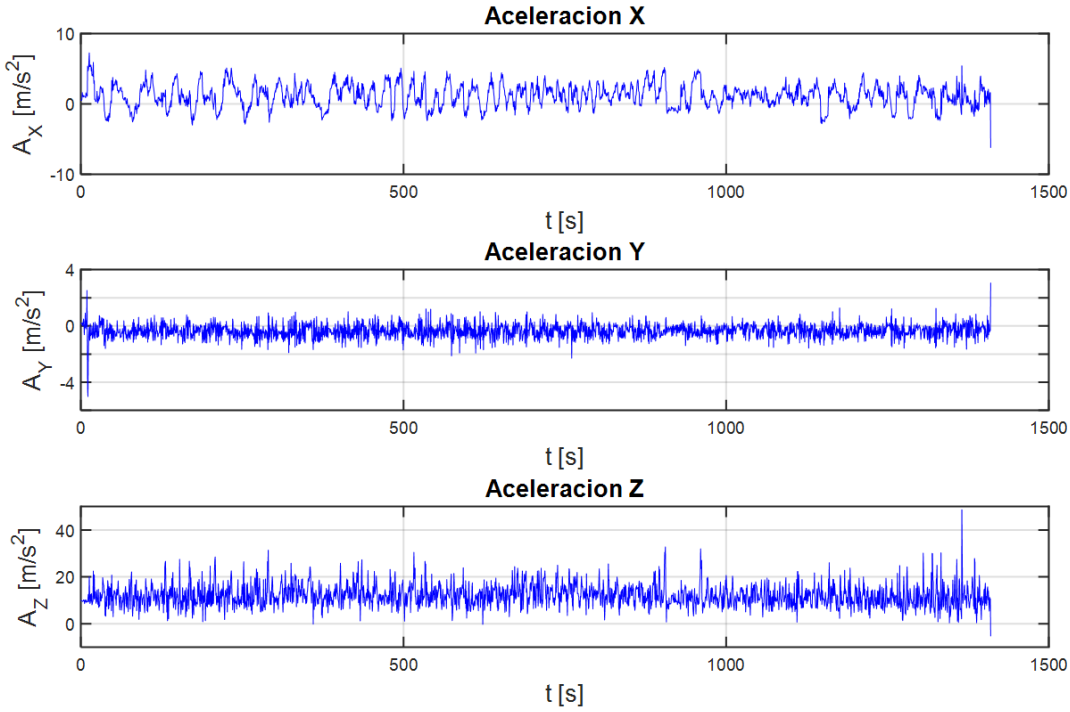

Figura 2-7. Componentes de la aceleración en ejes cuerpo.

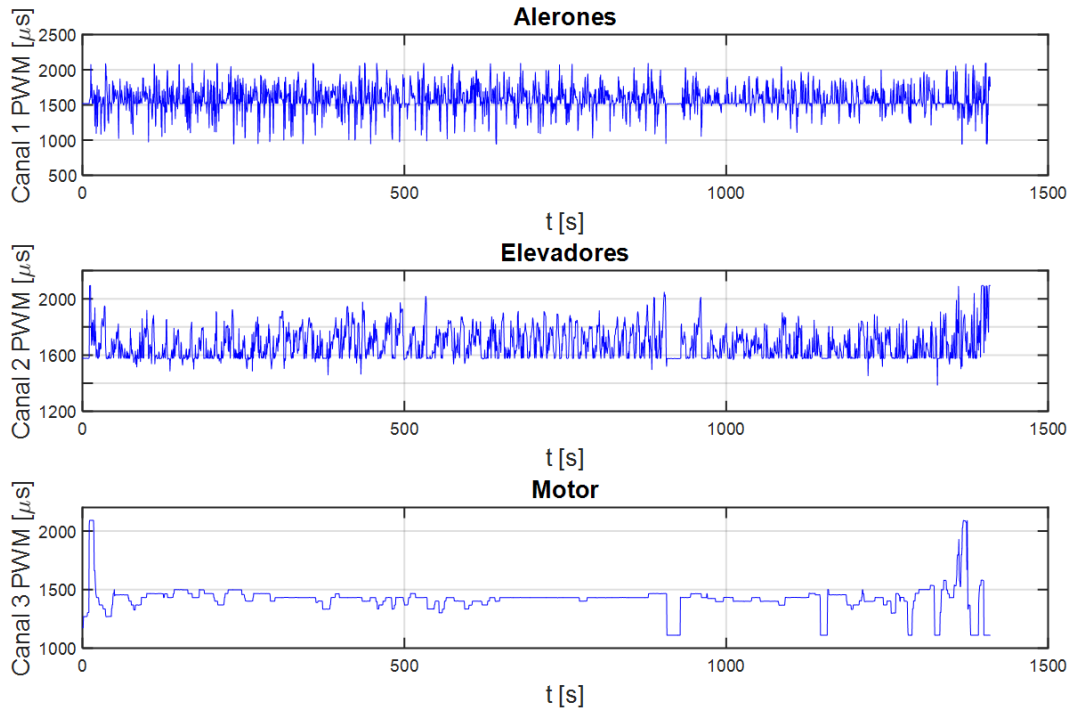

Figura 2-8. Señales PWM (*Pulse Width Modulation*) de cada actuador.

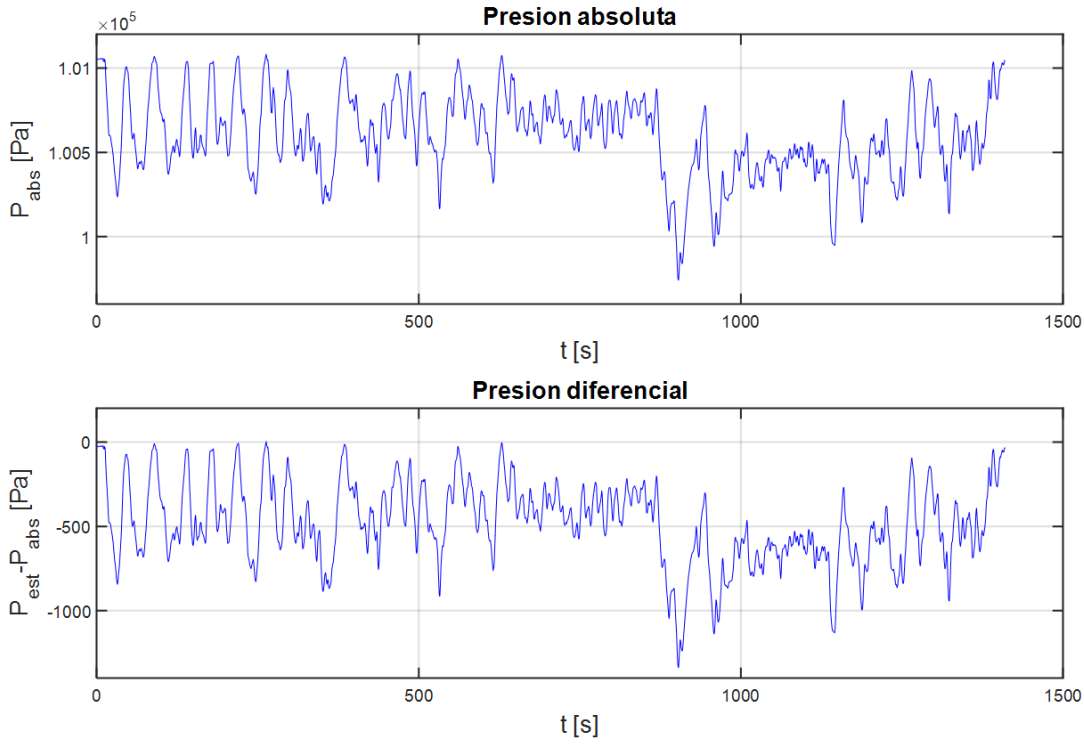

Figura 2-9. Presión absoluta y presión diferencial.

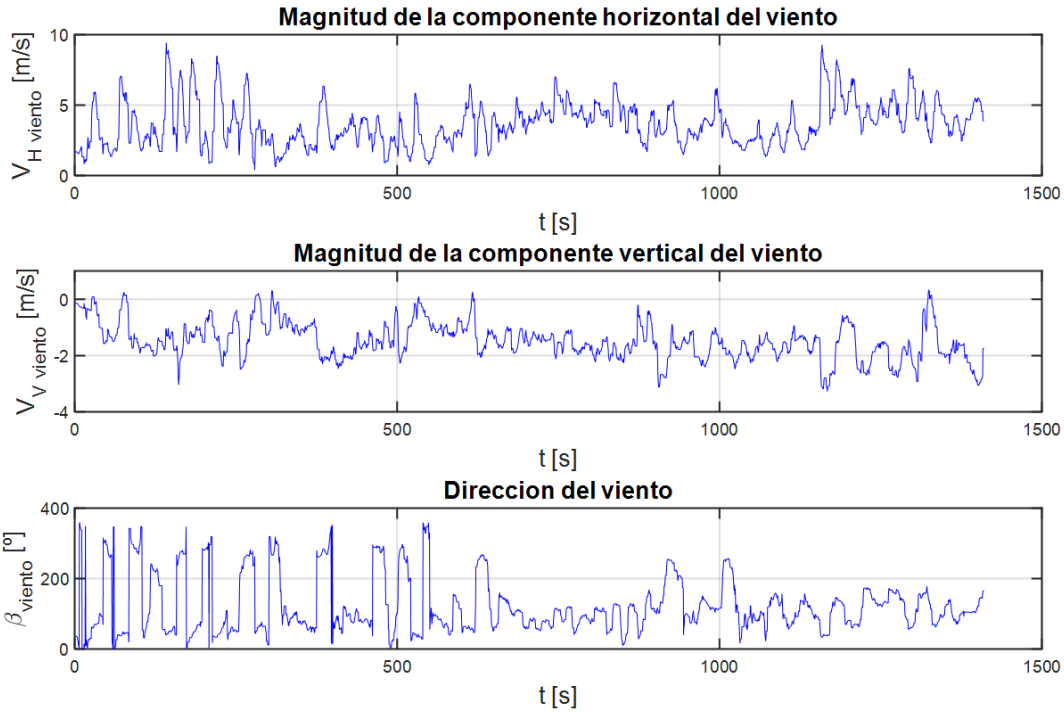

Figura 2-10. Componentes horizontal y vertical del viento y su dirección en ejes viento.

Las distintas longitudes de sus respectivos vectores de tiempo indican que los diferentes sensores no trabajaban a una única frecuencia de muestreo. Por ello, ha sido primordial un procesamiento de los datos para acondicionar las distintas señales y homogeneizar su distribución temporal; en otras palabras, las señales han sido modificadas con el objetivo de uniformizar los vectores de tiempo de tal manera que todos tengan la misma longitud. El motivo surge de la necesidad de trabajar en Matlab con vectores y matrices del mismo tamaño para poder realizar cálculos.

En consecuencia, se ha optado por un método de redimensionamiento de los vectores con cierta similitud a un ZOH (*Zero Order Hold*). Este consiste en rellenar los intervalos entre tiempos de los vectores de menor longitud con los tiempos del vector que trabaja a una mayor tasa de refresco, y por tanto su longitud es mayor, al mismo tiempo que se mantiene el valor anterior medido por el sensor.

Las siguientes tablas muestran este proceso mediante un sencillo ejemplo:

Tiempos (no procesado)	**Datos (no procesado)**
0	0
0.4010	0.1400
0.8010	-0.03000

Tabla 2-1. Matriz de datos no procesada.

Tiempos	Datos
0	-2.487
0.1210	-2.489
0.2400	-2.489
0.3610	-2.489
0.4800	-2.489
0.5990	-2.489
0.7220	-2.489
0.8410	-2.489

Tabla 2-2. Matriz de datos de mayor longitud.

Tiempos (procesado)	Datos (procesado)
0	0
0.1210	0
0.2400	0
0.4010	0.1400
0.4800	0.1400
0.5990	0.1400
0.8010	-0.03000
0.8410	-0.03000

Tabla 2-3. Matriz de datos procesada.

Las matrices no procesadas de mayor longitud, que contienen los datos de la unidad de medición inercial o IMU (*Inertial Measurement Unit*) debido a que trabaja a una mayor tasa de refresco, se dejan intactas durante el procesamiento.

Como resultado de aplicar este procedimiento para la telemetría de todos los vuelos, se obtienen vectores de la misma longitud que, esta vez sí, son aptos para su manejo en Matlab.

Desgraciadamente, los datos recabados por los sensores no son especialmente rigurosos ni precisos debido en gran medida al uso de componentes fácilmente asequibles y económicos. Ni siquiera el objetivo fundamental del proyecto original era el de recabar información precisa de los vuelos de prueba. Muestra de ello es el hecho de que no se implementara un sensor de medición de viento en el UAV, lo que implica que los datos de viento en este proyecto no sean más que meras estimaciones obtenidas por Mission Planner.

He aquí un ejemplo de error en los datos de telemetría correspondientes al cuarto vuelo:

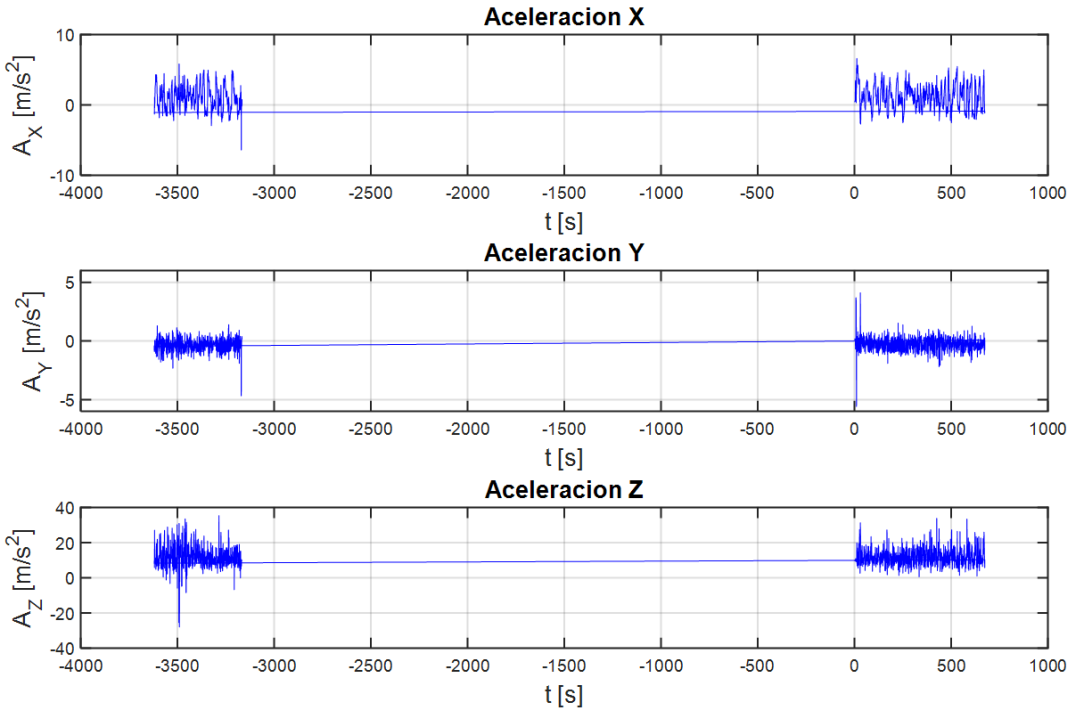

Figura 2-11. Componentes de la aceleración en ejes cuerpo correspondientes al cuarto vuelo (sin procesar).

Se observa una clara traslación en tiempo de los datos de telemetría, posiblemente debido a un reseteo del sensor en pleno vuelo. Es por esto que el simple hecho de no investigar suficientemente sobre la coherencia de los datos puede desembocar en la obtención de un modelo totalmente erróneo. Asimismo, la existencia de datos espúreos hace indispensable el procesamiento de los datos y su acondicionamiento en aras de poder ser utilizados en Matlab.

2.2. Modelo

En esta sección se presenta el modelo implementado en el simulador, incluyendo sus distintos componentes y ecuaciones, así como algunos de los resultados obtenidos a partir de los datos de telemetría, como el modelo propulsivo.

2.1.1 Sistemas de referencia

En aeronáutica se hace imprescindible el uso de distintos sistemas de referencia debido a la gran cantidad de elementos que afectan al comportamiento de la aeronave y, por tanto, variables que se requieren manejar. Por este motivo, se han utilizado:

- **Sistema de ejes Tierra o ECEF (*Earth Centered, Earth Fixed*):** útil para referenciar posiciones terrestres, pues gira con la Tierra, cuya forma es aquella del elipsoide WGS84 (utilizado por el sistema GPS). Los parámetros que se utilizan para marcar la posición en este sistema de referencia son los ángulos de longitud (λ), latitud (φ) y altitud (h).

- **Sistema de ejes horizonte local (o de navegación) o NED (*North, East, Down*):** sistema de referencia local centrado en un punto que puede o no encontrarse sobre la superficie terrestre. Se traslada al moverse el punto de referencia. Los parámetros que lo definen son las direcciones Norte (N), Este (E) y vertical hacia abajo (D).

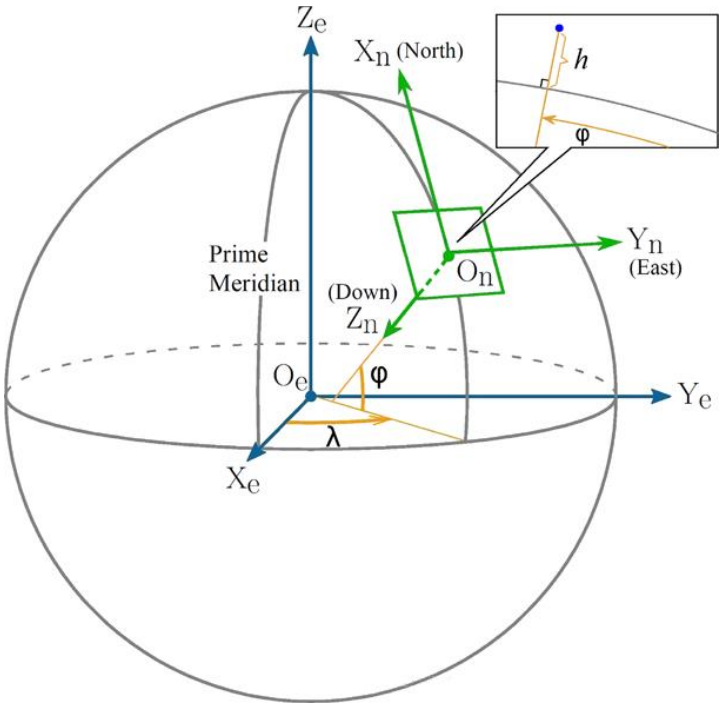

Figura 2-12. Sistemas de referencia de ejes Tierra y de navegación.

- **Sistema de ejes cuerpo o BFS (*Body Fixed System*):** utilizado para definir la actitud (orientación) de la aeronave respecto al sistema de ejes de navegación. Los parámetros que la determinan son los ángulos de Euler de alabeo (ϕ), cabeceo (θ) y guiñada (ψ).

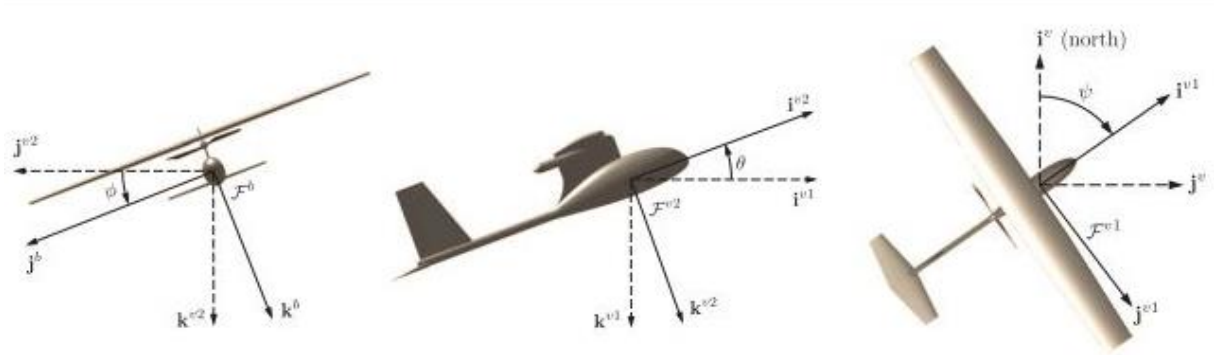

Figura 2-13. Sistema de referencia de ejes cuerpo.

- **Sistema de ejes viento o WF (*Wind Frame*):** sirve para orientar el viento respecto a la aeronave. Los parámetros que lo definen son la magnitud del mismo y dos ángulos: el ángulo de ataque (α) y el ángulo de deslizamiento (β).

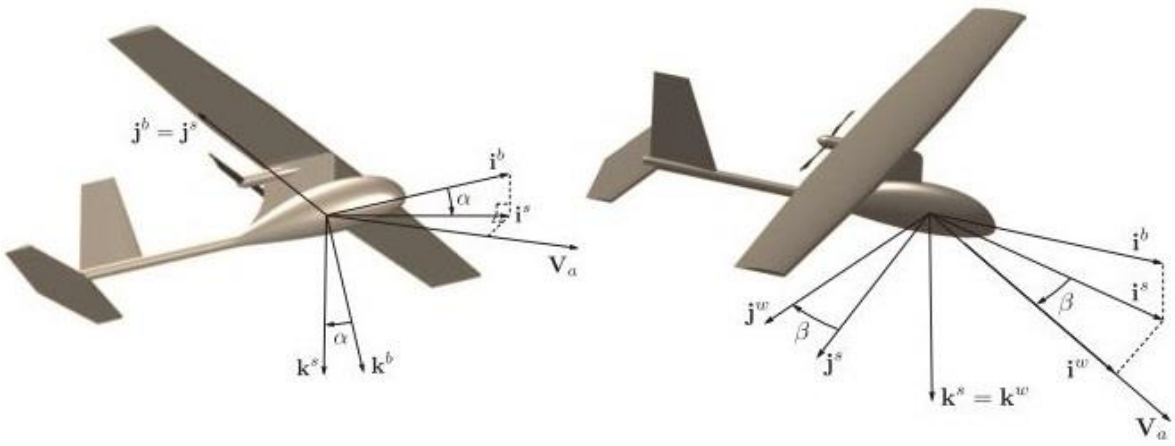

Figura 2-14. Sistema de referencia de ejes viento.

La importancia del uso de distintos sistemas de referencia radica en la capacidad de transformar vectores. Esto permite la representación de distintas variables bajo sistemas de referencia más acordes a su naturaleza, como es el caso de la posición en ejes Tierra o de navegación y el viento en ejes viento.

Para transformar vectores expresados en un sistema de referencia a otro se utilizan las matrices de rotación, también conocidas como matrices de cosenos directores o DCM (*Direction Cosine Matrix*). Así, la transformación de un vector *v* de un sistema de referencia A a otro B queda definida del siguiente modo:

$$v_B = \boldsymbol{R_A^B} v_A$$

Una característica importante de estas matrices de rotación es su ortogonalidad ($\boldsymbol{R_B^A} = (\boldsymbol{R_A^B})^T$), la cual permite transformar en el sentido contrario simplemente multiplicando el vector *v* por la traspuesta de dicha matriz:

$$v_A = \boldsymbol{R_B^A} v_B = (\boldsymbol{R_A^B})^T v_B$$

Para aplicar una transformación secuencial, esto es, de un sistema de referencia A a otro D, por ejemplo, de los cuales no se posee su matriz de rotación, pero sí de sistemas de referencia intermedios B y C, el prodimiento es el siguiente:

$$v_D = \boldsymbol{R_C^D R_B^C R_A^B} v_A$$

A continuación, se exponen las matrices de rotación que se han utilizado durante el proyecto. Nótese que, como se ha mencionado con anterioridad, sólo haría falta trasponer las matrices para obtener aquellas que definen la transformación opuesta.

- Matriz de rotación de ejes Tierra a ejes de navegación:

$$\boldsymbol{R_g^n} = \begin{pmatrix} -sen\varphi\ cos\lambda & -sen\varphi\ sen\lambda & cos\varphi \\ -sen\lambda & cos\lambda & 0 \\ -cos\varphi\ cos\lambda & -cos\varphi\ sen\lambda & -sen\varphi \end{pmatrix}$$

- Matriz de rotación de ejes de navegación a ejes cuerpo:

$$\boldsymbol{R_n^b} = \begin{pmatrix} c\theta c\psi & c\theta s\psi & -s\theta \\ -c\phi s\psi + s\phi s\theta c\psi & c\phi c\psi + s\phi s\theta s\psi & s\phi c\theta \\ s\phi s\psi + c\phi s\theta c\psi & -s\phi c\psi + c\phi s\theta s\psi & c\phi c\theta \end{pmatrix}$$

- Matriz de rotación de ejes viento a ejes cuerpo:

$$\boldsymbol{R_w^b} = \begin{pmatrix} cos\beta\ cos\alpha & -sen\beta\ cos\alpha & -sen\alpha \\ sen\beta & cos\beta & 0 \\ cos\beta\ sen\alpha & -sen\beta\ sen\alpha & cos\alpha \end{pmatrix}$$

2.1.2 Cinemática

Ahora, se exponen las ecuaciones mediante las cuales se obtiene la derivada de la posición en ejes de navegación del Skywalker X8 a partir de las componentes de la velocidad lineal y su orientación (ángulos de Euler). Seguidamente, se incluyen las ecuaciones para calcular la derivada de la actitud, en función de las componentes de la velocidad angular y la actitud.

- Ratio de cambio de la posición en ejes de navegación:

$$\begin{pmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{p}_d \end{pmatrix} = \begin{pmatrix} c_\theta c_\psi & s_\phi s_\theta c_\psi - c_\phi s_\psi & c_\phi s_\theta c_\psi + s_\phi s_\psi \\ c_\theta s_\psi & s_\phi s_\theta s_\psi + c_\phi c_\psi & c_\phi s_\theta s_\psi - s_\phi c_\psi \\ -s_\theta & s_\phi c_\theta & c_\phi c_\theta \end{pmatrix} \begin{pmatrix} U \\ V \\ W \end{pmatrix}$$

- Ratio de cambio de la actitud:

$$\begin{pmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} 1 & \text{sen}\phi\, tan\theta & cos\phi\, tan\theta \\ 0 & cos\phi & -sen\phi \\ 0 & sen\phi\, sec\theta & cos\phi\, sec\theta \end{pmatrix} \begin{pmatrix} P \\ Q \\ R \end{pmatrix}$$

Finalmente, estas derivadas se integran para obtener, junto a sus respectivos valores iniciales, la posición en ejes de navegación y la actitud en un instante concreto.

2.1.3 Dinámica

De igual modo, se presentan las ecuaciones de dinámica. Por un lado, permiten calcular la derivada de la velocidad lineal en función de las componentes de la velocidad lineal y angular, la masa de la aeronave y las componentes de la fuerza a la que se haya sometida expresadas en ejes cuerpo; por el otro, el cálculo de la derivada de la velocidad angular a partir de las misma, los momentos de las fuerzas experimentados por el Skywalker X8 en ejes cuerpo y una serie de parámetros mecánicos (Γ) que dependen exclusivamente de los momentos y productos de inercia.

- Ratio de cambio de la velocidad lineal:

$$\begin{pmatrix} \dot{U} \\ \dot{V} \\ \dot{W} \end{pmatrix} = \begin{pmatrix} RV - QW \\ PW - RU \\ QU - PV \end{pmatrix} + \frac{1}{m} \begin{pmatrix} F_X \\ F_Y \\ F_Z \end{pmatrix}$$

- Ratio de cambio de la velocidad angular:

$$\begin{pmatrix} \dot{P} \\ \dot{Q} \\ \dot{R} \end{pmatrix} = \begin{pmatrix} \Gamma_1 PQ - \Gamma_2 QR \\ \Gamma_5 PR - \Gamma_6 (P^2 - R^2) \\ \Gamma_7 \text{PQ} - \Gamma_1 QR \end{pmatrix} + \begin{pmatrix} \Gamma_3 L + \Gamma_4 N \\ \frac{1}{I_y} M \\ \Gamma_4 L + \Gamma_8 N \end{pmatrix}$$

Siendo:

$$\Gamma = I_x I_z - I_{xz}^2$$

$$\Gamma_1 = \frac{I_{xz}(I_x - I_y + I_z)}{\Gamma}$$

$$\Gamma_2 = \frac{I_z(I_z - I_y) + I_{xz}^2}{\Gamma}$$

$$\Gamma_3 = \frac{I_z}{\Gamma}$$

$$\Gamma_4 = \frac{I_{xz}}{\Gamma}$$

$$\Gamma_5 = \frac{I_z - I_x}{I_y}$$

$$\Gamma_6 = \frac{I_{xz}}{I_y}$$

$$\Gamma_7 = \frac{I_x(I_x - I_y) + I_{xz}^2}{\Gamma}$$

$$\Gamma_8 = \frac{I_x}{\Gamma}$$

Análogamente, para obtener la velocidad lineal y angular en un tiempo determinado es necesario la integración de sus derivadas a partir de sus valores iniciales.

2.1.4 Fuerzas y momentos

En este apartado se presentan las ecuaciones que permiten calcular las componentes de las fuerzas y momentos ejercidos sobre el Skywalker X8 expresados en ejes cuerpo. Las ecuaciones genéricas tienen la siguiente forma:

$$\sum \vec{F} = \vec{F}_{gravitatoria} + \vec{F}_{aerodinámica} + \vec{F}_{propulsiva} = m \cdot \vec{a} \qquad \sum \vec{M} = \sum(\vec{r} \times \vec{F})$$

Como se puede observar, el sumatorio de fuerzas se compone de tres grandes bloques: la fuerza gravitatoria, que se rige por el modelo de gravedad EGM96 (*Earth Gravitational Model '96*) incluido en el sistema de coordenadas WGS84 (*World Geodetic System '84*); la aerodinámica, cuyos coeficientes aerodinámicos se han obtenido de estudios experimentales externos; y la propulsiva, resultado del modelo desarrollado en este proyecto.

- **Sistema de coordenadas geodéticas WGS84:** estándar en navegación y utilizado por el sistema GPS (*Global Positioning System*). Formado por una superficie equipotencial, su geoide (EGM96), que define el nivel nominal del mar.

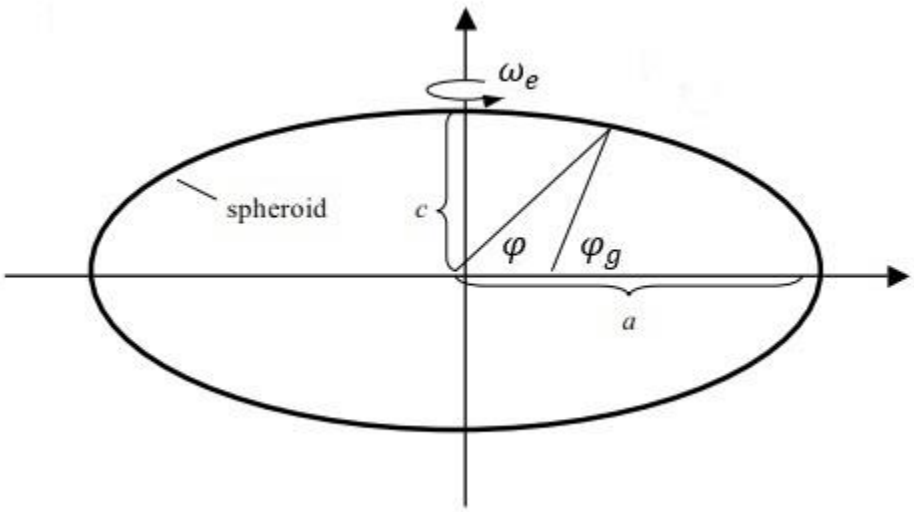

Figura 2-15. Esferoide del sistema WGS84.

Este modelo es implementado directamente a través de sendas funciones incluidas en Matlab y Simulink, las cuales calculan la aceleración de la gravedad a partir de la latitud y la altitud mediante la siguiente ecuación:

$$g(r,\varphi) = -\frac{GM_e}{r^2}\left[1 - \frac{3J_2a^2}{2r^2}(3sen^2\varphi - 1)\right] + {\omega_e}^2 r cos^2\varphi$$

- **Modelo aerodinámico:** como se mencionó previamente, debido a la imposibilidad de realizar ensayos experimentales de ningún tipo sobre la aeronave, se optó por llevar a cabo un trabajo de investigación previo el cual concluyó con la decisión de usar una serie de coeficientes tanto mecánicos como aerodinámicos obtenidos a partir de una serie de experimentos efectuados sobre un Skywalker X8 por parte de investigadores de la Universidad Noruega de Ciencia y Tecnología de Trondheim. Este documento se titula Gryte, K. (2015). *High angle of attack landing of an unmanned aerial vehicle.* MSc Tesis. Norwegian University of Science and Technology. Trondheim, Norway. Los parametros seleccionados no son mas que los momentos y productos de inercia (los cuales pueden sufrir ciertas variaciones con respecto a aquellos del Skywalker X8 cuya telemetría se ha usado en este proyecto debido a una diferente distribución de masa) y coeficientes aerodinámicos (adimensionales).

  Asimismo, se han reciclado parámetros dimensionales del proyecto original del cual se toman los datos de telemetría, por ejemplo: la masa de la aeronave, la envergadura, la cuerda media del perfil aerodinamico o la superficie alar.

  A continuación, se presenta una tabla completa con todos aquellos parámetros que definen a la aeronave y que han sido utilizados para los distintos cálculos, ya sea durante el modelado o la simulación.

Parámetro	Valor	Parámetro	Valor
m	3.7970	C_{Y_0}	3.2049e-18
b	2.1000	C_{Y_β}	-0.1949
c	0.3571	C_{Y_p}	-0.1172
S_{alar}	0.7500	C_{Y_r}	0.0959
AR	5.8800	$C_{Y_{\delta a}}$	-0.0696
I_x	1.2290	C_{l_0}	1.1518e-18
I_y	0.1702	C_{l_β}	-0.0765
I_z	0.8808	C_{l_p}	-0.4018
I_{xz}	0.9343	C_{l_r}	0.0250
e	0.9935	$C_{l_{\delta a}}$	0.2987
α_0	0.2670	C_{m_0}	0.0180
C_{L_0}	0.0254	C_{m_α}	-0.2524
C_{L_α}	4.0191	$C_{m_{fp}}$	-0.2168
C_{L_q}	3.8954	C_{m_q}	-1.3047
$C_{L_{\delta e}}$	0.5872	$C_{m_{\delta e}}$	-0.4857
C_{D_0}	0.0102	C_{n_0}	-2.2667e-7
$C_{D_{\beta 1}}$	-2.0864e-7	C_{n_β}	0.0403
$C_{D_{\beta 2}}$	0.0671	C_{n_p}	-0.0247
C_{D_q}	0	C_{n_r}	-0.1252
$C_{D_{\delta e}}$	0.8461	$C_{n_{\delta a}}$	0.0076

Tabla 2-4. Parámetros mecánicos y aerodinámicos del Skywalker X8.

Como se comprobará al término de este apartado, las ecuaciones de fuerzas y momentos incluyen un término sigma (σ) en sus componentes aerodinámicas. Este término no es más que la representación de una función sigmoide cuyo valor depende únicamente del ángulo de ataque (α) y que define la transición suave del comportamiento lineal del ala al de placa plana. Dicho comportamiento se puede apreciar en la siguiente figura:

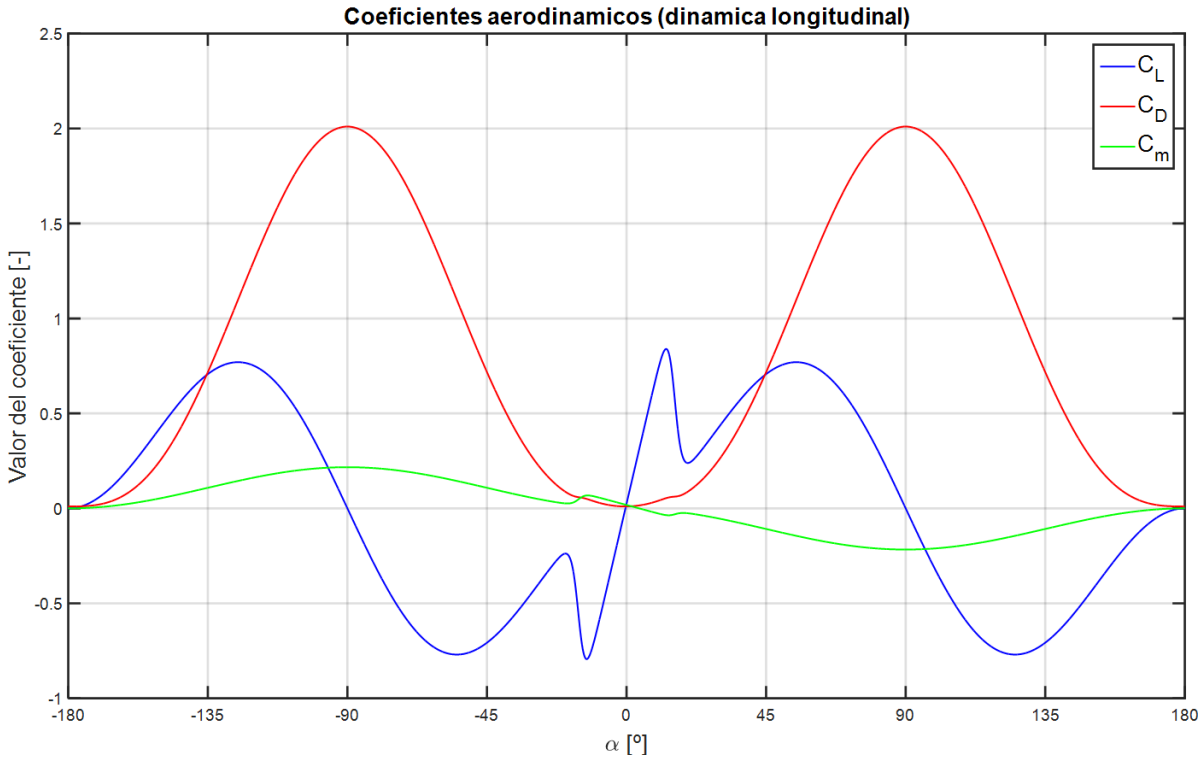

Figura 2-16. Coeficientes de sustentación, resistencia y momento de cabeceo.

- **Fuerza propulsiva:** esta ha jugado el papel de incógnita durante el modelado. Para obtenerla se ha despejado la fuerza propulsiva (δ_t) de la componente longitudinal de la fuerza resultante a la cual se hayaba sometido el UAV en cada instante de tiempo de todos los vuelos, a partir de la segunda ley de Newton. He aquí la ecuación mencionada:

$$\delta_t = m \cdot \left[a_X + \begin{pmatrix} 0 & -R & Q \end{pmatrix} \begin{pmatrix} U \\ V \\ W \end{pmatrix} \right] - F_{gravitatoria_X} - F_{aerodinámica_X}$$

Posteriormente, los resultados alcanzados se han ajustado mediante mínimos cuadrados con el objetivo de crear una función o ley que definiera el comportamiento de la fuerza propulsiva a partir de dos parámetros: la diferencia entre la señal PWM del motor respecto a la señal PWM del motor mínima y la velocidad aerodinámica en el eje longitudinal de la aeronave. Dicha función tiene la siguiente forma:

$$\delta_t = C_1 \cdot \Delta Motor_{PWM} + C_2 \cdot {U_a}^2$$

El propósito no es más que el hecho de poder implementar esta ley en el simulador, para lo cual la fuerza de propulsión debe estar caracterizada de alguna manera. Como se verá al término de este capítulo, el ajuste por mínimos cuadrados determina los valores de los coeficientes de tal manera que $C_1 > 0$ y $C_2 < 0$, tal y como se podía prever.

Finalmente, se exponen las ecuaciones generales que permiten el cálculo de las componentes de fuerzas y momentos constituidas por los puntos mencionados:

$$\begin{pmatrix} F_X \\ F_Y \\ F_Z \end{pmatrix} =$$

$$-\frac{1}{2}\rho {U_\infty}^2 S_{alar} \boldsymbol{R_w^b} \begin{pmatrix} C_{D_0} + \left(1-\sigma(\alpha)\right)\cdot \frac{\left(C_{L_0}+C_{L_\alpha}\alpha\right)^2}{\pi\, e\, AR} + \sigma(\alpha)\cdot(2\, signo(\alpha)\, sen^3\alpha) + C_{D_\beta}\beta + C_{D_{\beta^2}}\beta^2 + C_{D_q}\frac{c}{2U_\infty}Q + C_{D_{\delta_e}}\delta_e \\ 0 \\ \left(1-\sigma(\alpha)\right)\cdot\left(C_{L_0}+C_{L_\alpha}\alpha\right) + \sigma(\alpha)\cdot(2\, signo(\alpha)\, sen^2\alpha\cos\alpha) + C_{L_q}\frac{c}{2U_\infty}Q + C_{L_{\delta_e}}\delta_e \end{pmatrix}$$

$$+\frac{1}{2}\rho {U_\infty}^2 S_{alar} \begin{pmatrix} 0 \\ C_{Y_0} + C_{Y_\beta}\beta + C_{Y_p}\frac{b}{2U_\infty}P + C_{Y_r}\frac{b}{2U_\infty}R + C_{Y_{\delta_a}}\delta_a \\ 0 \end{pmatrix} + \boldsymbol{R_n^b}\begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} + \begin{pmatrix} \delta_t \\ 0 \\ 0 \end{pmatrix}$$

$$\begin{pmatrix} L \\ M \\ N \end{pmatrix} = \frac{1}{2}\rho {U_\infty}^2 S_{alar} \begin{pmatrix} b\cdot(C_{l_0} + C_{l_\beta}\beta + C_{l_p}\frac{b}{2U_\infty}P + C_{l_r}\frac{b}{2U_\infty}R + C_{l_{\delta_a}}\delta_a) \\ c\cdot\left(\left(1-\sigma(\alpha)\right)\cdot(C_{m_0}+C_{m_\alpha}\alpha) + \sigma(\alpha)\cdot\left(C_{m_{fp}}\, signo(\alpha)\, sen^2\alpha\right) + C_{m_q}\frac{c}{2U_\infty}Q + C_{m_{\delta_e}}\delta_e\right) \\ b\cdot(C_{n_0} + C_n\beta + C_{n_p}\frac{b}{2U_\infty}P + C_{n_r}\frac{b}{2U_\infty}R + C_{n_{\delta_a}}\delta_a) \end{pmatrix}$$

2.1.5 Perturbaciones atmosféricas

Por último, se han añadido perturbaciones atmosféricas basadas en el modelo Dryden de viento. Este modelo ha permitido simular de forma rigurosa turbulencia en la aeronave durante la parte del trabajo desarrollado en Simulink.

Según este modelo, la velocidad lineal de la masa de aire respecto a tierra se compone de dos sumandos. Por un lado, el perteneciente a un viento ambiental continuo, comúnmente expresado en ejes de navegación y, por otro lado, aquel producido por ráfagas de viento y otras perturbaciones atmosféricas, el cual es representado mediante un proceso estocástico y a menudo expresado en ejes cuerpo:

$$\vec{v}_{viento} = \vec{v}_{viento_{continuo}} + \vec{v}_{viento_{perturbaciones}}$$

Análogamente, el modelo implementado en el simulador introduce componentes de velocidad angular, lo que proporciona mayor realismo al inducir cierta vorticidad en la dinámica de la aeronave.

Los resultados experimentales indican que un buen modelo para esta componente no continua se obtiene haciendo pasar ruido blanco a través de un filtro lineal e invariante en el tiempo dado por el espectro de turbulencia de Von Karman. Desafortunadamente, dicho espectro no se haya definido por una función de transferencia racional. Las funciones de transferencia del modelo Dryden de viento, sin embargo, son una aproximación adecuada:

$$H_u(s) = \sigma_u \sqrt{\frac{2L_u}{\pi U_\infty}} \cdot \frac{1}{1 + \frac{L_u}{U_\infty} s} \qquad H_p(s) = \sigma_w \sqrt{\frac{0.8}{U_\infty}} \cdot \frac{\left(\frac{\pi}{4b}\right)^{\frac{1}{6}}}{L_w{}^{\frac{1}{3}} \left(1 + \left(\frac{4b}{\pi U_\infty}\right) s\right)}$$

$$H_v(s) = \sigma_v \sqrt{\frac{L_v}{\pi U_\infty}} \cdot \frac{1 + \frac{\sqrt{3}L_v}{U_\infty} s}{\left(1 + \frac{L_v}{U_\infty} s\right)^2} \qquad H_r(s) = \frac{\pm \frac{s}{U_\infty}}{\left(1 + \left(\frac{3b}{\pi U_\infty}\right) s\right)} \cdot H_v(s)$$

$$H_w(s) = \sigma_w \sqrt{\frac{L_w}{\pi U_\infty}} \cdot \frac{1 + \frac{\sqrt{3}L_w}{U_\infty} s}{\left(1 + \frac{L_w}{U_\infty} s\right)^2} \qquad H_q(s) = \frac{\pm \frac{s}{U_\infty}}{\left(1 + \left(\frac{4b}{\pi U_\infty}\right) s\right)} \cdot H_w(s)$$

Siendo σ_u, σ_v y σ_w las intensidades de turbulencia, y L_u, L_v y L_w la longitud espacial de las ondas que la definen. A continuación, se muestra una tabla extraída de [4] donde se presentan parámetros apropiados para las distintas condiciones de turbulencia deseadas.

gust description	*altitude (m)*	$L_u = L_v$ *(m)*	L_w *(m)*	$\sigma_u = \sigma_v$ *(m/s)*	σ_w *(m/s)*
low altitude, light turbulence	50	200	50	1.06	0.7
low altitude, moderate turbulence	50	200	50	2.12	1.4
medium altitude, light turbulence	600	533	533	1.5	1.5
medium altitude, moderate turbulence	600	533	533	3.0	3.0

Tabla 2-5. Parámetros del modelo Dryden de viento.

No obstante, los efectos del viento se han despreciado durante la etapa de modelado por lo que es previsible que los resultados que arroje el simulador no converjan con los datos de telemetría. El motivo ha sido la falta de rigurosidad en los datos de viento pues, como se ha mencionado con anterioridad, son únicamente estimaciones extraídas del Mission Planner, ya que no se embarcó ningún sensor de viento en el Skywalker X8.

2.3. Resultados

Para finalizar el capítulo, se exponen los resultados obtenidos durante el proceso de modelado, gracias al cual se obtuvo la ley propulsiva que define la fuerza generada por la planta motor en función de su señal PWM y la componente longitudinal de la velocidad aerodinámica.

En primera instancia, se presenta la gráfica resultado de despejar δ_t para cada instante de tiempo de cada uno de los cinco vuelos realizados, tal y como se ha explicado con anterioridad. En ella, se observa una nube de puntos en función de, aparte de las dos variables que definen la ley propulsiva, la altitud.

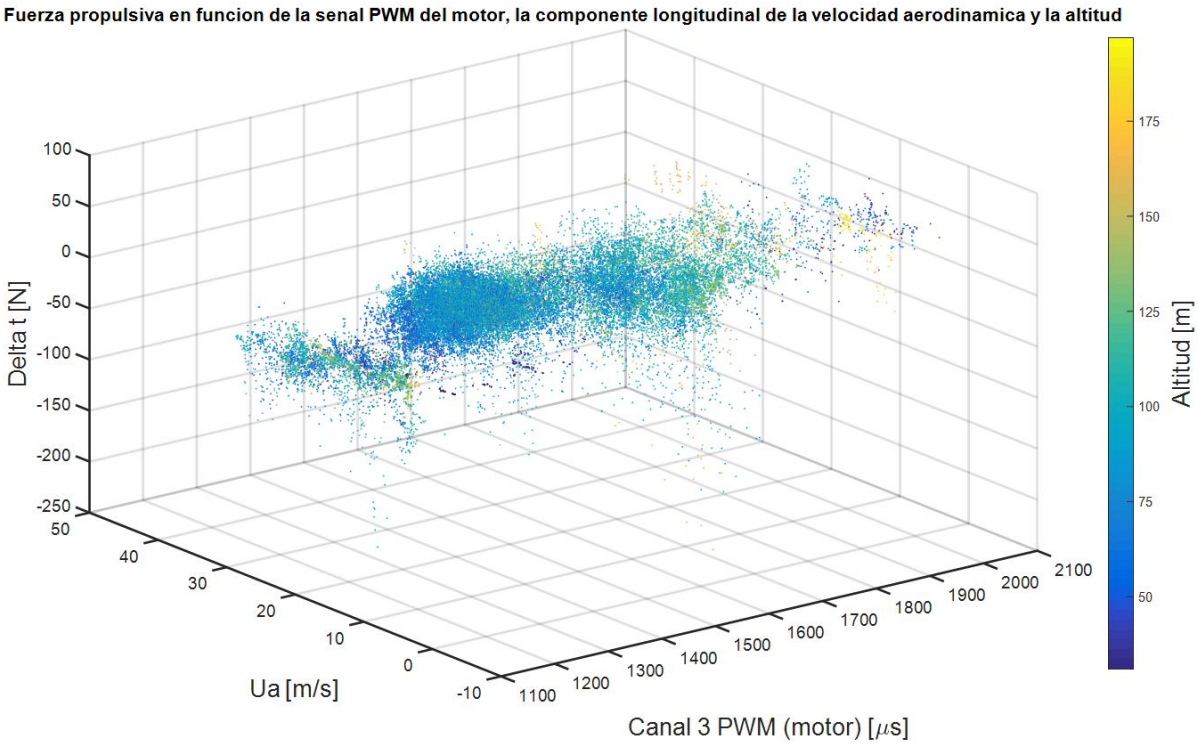

Figura 2-17. Resultados de la fuerza propulsiva obtenidos durante el modelado.

Puesto que la altitud máxima de vuelo alcanzó los escasos 200 metros, no se puede considerar dicha variable como un factor clave de la actuación de la planta propulsiva debido a que no se han recabado suficientes datos para que el ajuste de los resultados, incluyendo como incógnita a la altitud, sea satisfactorio. Al mismo tiempo, la aeronave bajo estudio no fue concebida para ser volada en capas altas, sino a bajas, por lo que la omisión de la misma cobra aún más sentido.

Lógicamente, el resultado esperado era aquel en el cual δ_t se hacía mayor conforme la potencia que generaba el motor, directamente relacionada con el valor de la señal PWM (1100 μs equivale a potencia nula mientras que 2100 μs equivale a potencia máxima), aumentaba y la componente longitudinal de la velocidad aerodinámica decrecía.

Al final, el ajuste por mínimos cuadrados del resultado mostrado lleva a la siguiente curva de actuación de la planta propulsiva:

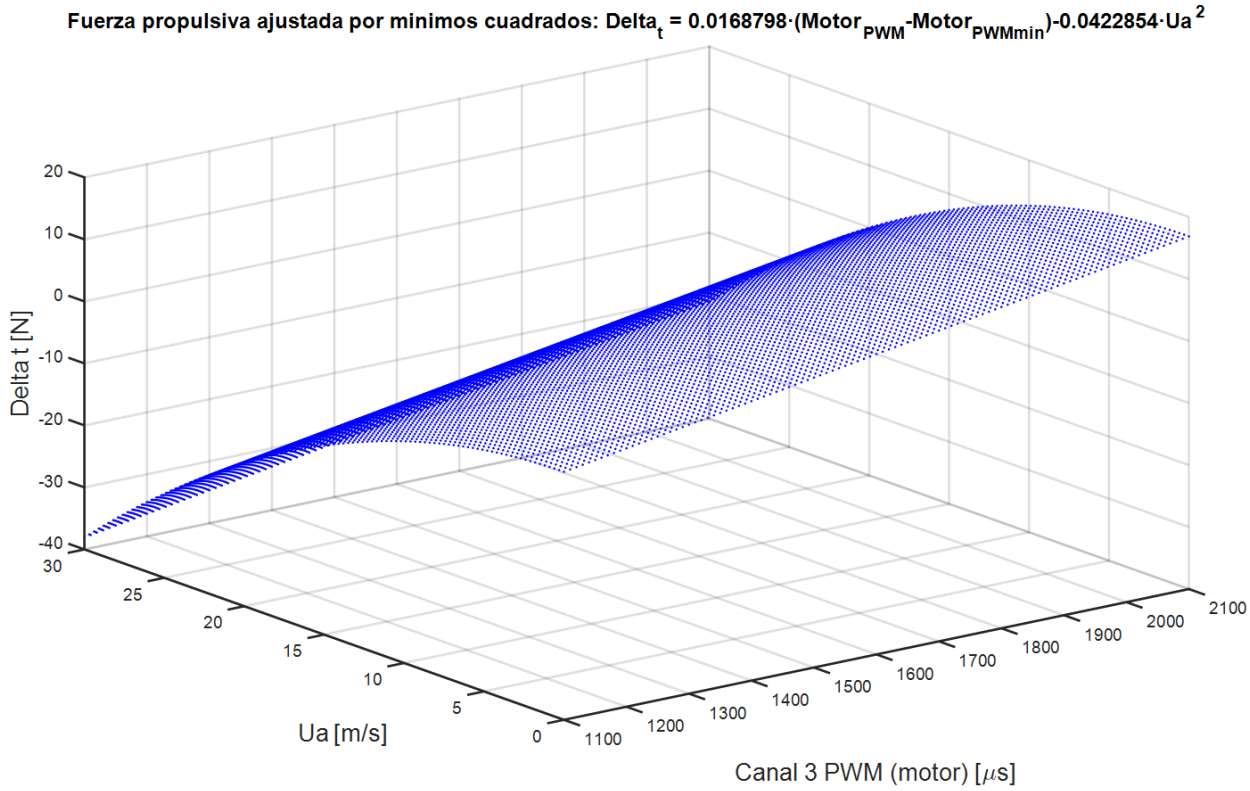

Figura 2-18. Resultado del ajuste por mínimos cuadrados de la fuerza propulsiva.

Se observa que, efectivamente, la fuerza de propulsión aumenta al dotar al motor de una mayor potencia eléctrica a la vez que decrece al aumentar la componente longitudinal de la velocidad aerodinámica, debido a que la resistencia aerodinámica se incrementa.

Finalmente, se consigue una ley que describe δ_t en función de $Motor_{PWM}$ y U_a, definida por las constantes $C_1 = 0.0168798$ y $C_2 = -0.0422854$. Esta ley, aunque simple, es suficiente para describir el comportamiento aproximado de la planta motor de la aeronave a partir de las dos variables que más repercuten en su actuación.

Por último, es digno de mención el hecho de que en ningún momento fue posible llevar a cabo pruebas experimentales de ningún tipo sobre la aeronave y sus componentes, inclusive el motor, lo que sin lugar a dudas habría facilitado en gran medida la obtención de un modelo propulsivo más preciso. Sin embargo, como se verá en el siguiente capítulo, esto no se convertirá en un impedimento para que la dinámica en simulación del Skywalker X8 sea lo suficientemente fiel a la realidad.

3 SIMULACIÓN

Este capítulo trata de describir a grandes rasgos el simulador y su funcionamiento. Para ello, se analiza su estructura y se comentan las principales características de los diferentes bloques que lo componen. Al final, se discuten los resultados obtenidos en él a partir de los datos de telemetría de los actuadores al mismo tiempo que se compara con la actuación de la aeronave durante los ensayos en vuelo.

3.1 Simulador

En esta sección se presenta la estructura general del simulador que, desde un principio, fue concebido para dotar al Skywalker X8 de los sistemas de GNC (*Guidance, Navigation and Control*) habituales en una aeronave de estas características. Por este motivo, incluye, entre muchos otros, un bloque de misión así como un autopiloto, a los cuales se prestará especial atención durante el desarrollo de este capítulo y que han supuesto la mayor parte del desarrollo.

En cuanto a su implementación, esta se llevo a cabo en Simulink; sin embargo, el simulador también se comunica con Matlab para el procesamiento de funciones contenidas en el mismo. Más aún, implementa una opción gracias a la cual se puede hacer uso de una segunda instancia de Matlab, aparte de la usada por el propio simulador para realizar sus cálculos, que juega el papel de receptor GNSS (*Global Navigation Satellite System*), como se explicará más adelante. Por último, el simulador contiene un bloque el cual permite enviar datos de posición y actitud a FlightGear, un software de simulación de vuelo de código abierto, de tal forma que este último sirva como interfaz gráfica del mismo.

Para configurar el simulador y entender su funcionamiento, en *Anexo 1* se explica en detalle su puesta a punto junto a una breve guía de uso para orientar a futuros usuarios sobre las distintas funcionalidades que posee. Por otro lado, *Anexo 2* contiene un tutorial para la instalación de FlightGear y sus componentes, como el modelo gráfico de la aeronave o los datos del terreno, en sistemas operativos Windows. Finalmente, *Anexo 3* recoge todo el código producido durante la etapa de simulación, aunque también contiene la parte del código correspondiente a la fase previa de modelado.

En las siguientes subsecciones se explica por encima la idea general de cada bloque, con especial énfasis en el sistema de misión, donde se expondrán los algoritmos de seguimiento y gestión de ruta, y el autopiloto, donde se presentan los distintos controladores utilizados tanto para dinámica lateral como longitudinal.

A continuación, se muestra una imagen del simulador implementado en Simulink, donde se pueden ver los bloques que lo componen:

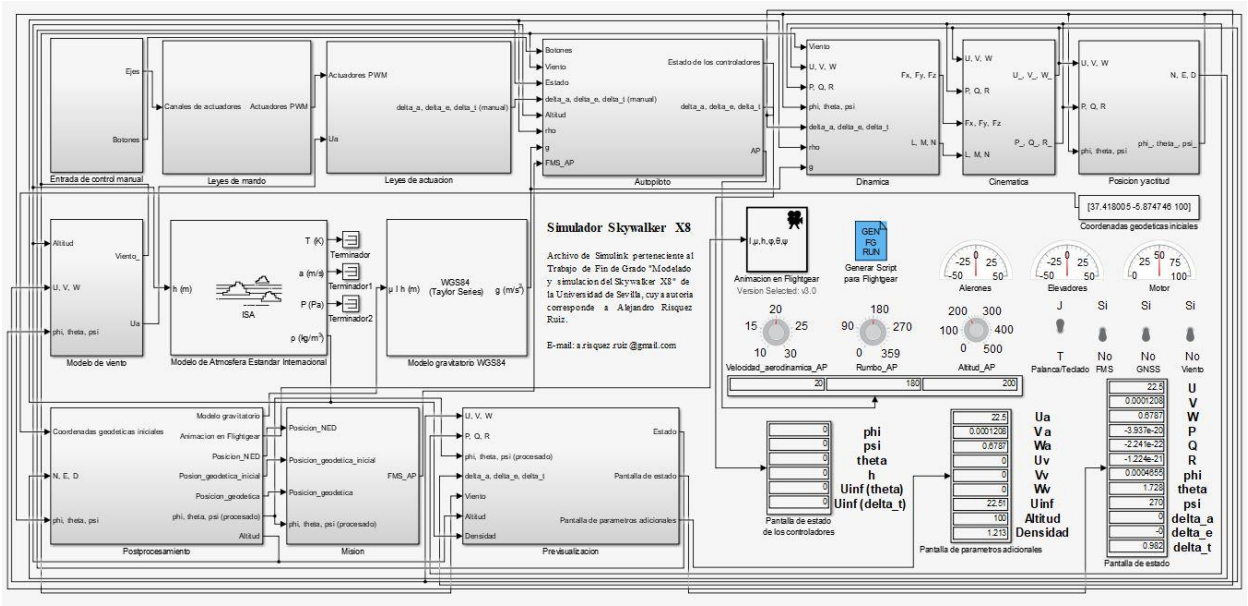

Figura 3-1. Simulador implementado en Simulink.

Este es el aspecto global del simulador, donde se pueden observar los diferentes bloques primarios. De izquierda a derecha y de arriba hacia abajo se encuentran los bloques de entrada de control manual, leyes de mando, leyes de actuación, autopiloto, dinámica, cinemática, posición y actitud, modelo de viento, modelo de Atmósfera Estándar Internacional o ISA (*International Standard Atmosphere*), modelo gravitatorio WGS84, postprocesamiento, misión y previsualización. Además, incluye una zona (en el cuadrante de abajo a la derecha) destinada a la presentación de las variables más importantes de la simulación como son el estado de los controladores, las variables de estado y otros parámetros de especial interés así como las ruedas de selección de las variables controladas por el autopiloto (velocidad aerodinámica, rumbo y altitud) y los interruptores que controlan difererentes funcionalidades del simulador (tipo de entrada manual, sistema de gestión de vuelo, sistema de navegación global por satélite y los efectos del viento).

3.1.1 Entrada de control manual

La función de este bloque es implementar la lógica necesaria para poder utilizar una palanca de mando (*joystick*) conectado al ordenador o el teclado como control manual de los actuadores de la aeronave. En la ventana principal del simulador existe un interruptor que permite seleccionar entre ambos controles. Desgraciadamente, el simulador remite un error si no encuentra un identificador correspondiente a una palanca de control conectada al ordenador. Por tanto, se ha creado un simulador idéntico, ejecutado desde un archivo diferente, con la única diferencia de no disponer de entrada de palanca para el caso en el que se esté corriendo sin tener una conectada.

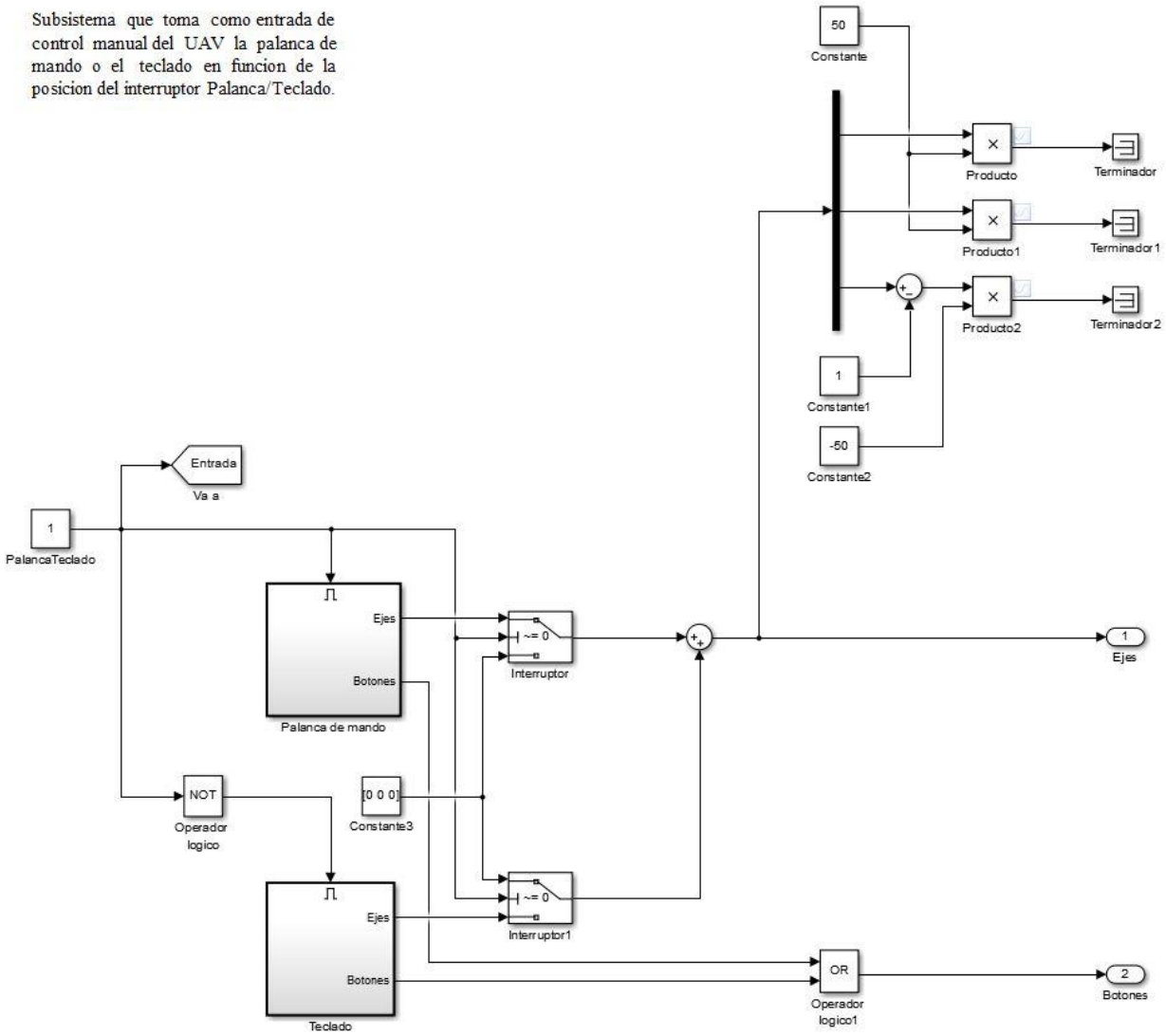

Figura 3-2. Bloque de entrada de control manual.

Como es apreciable, contiene dos subsistemas para sendas entradas, seleccionables a partir del valor del interruptor de entrada manual (*Palanca/Teclado*). Asimismo, manda información del estado de los actuadores (alerones, elevadores y motor) a sus respectivos indicadores en la pestaña principal.

3.1.2 Leyes de mando

El propósito de este sistema es simular una zona muerta entorno al punto de calibración de cada uno de los ejes de los actuadores para dotar de realismo a los mandos de vuelo. Es sobretodo útil a la hora de usar la palanca de mando, pero se aplica igualmente si se opta por usar el teclado.

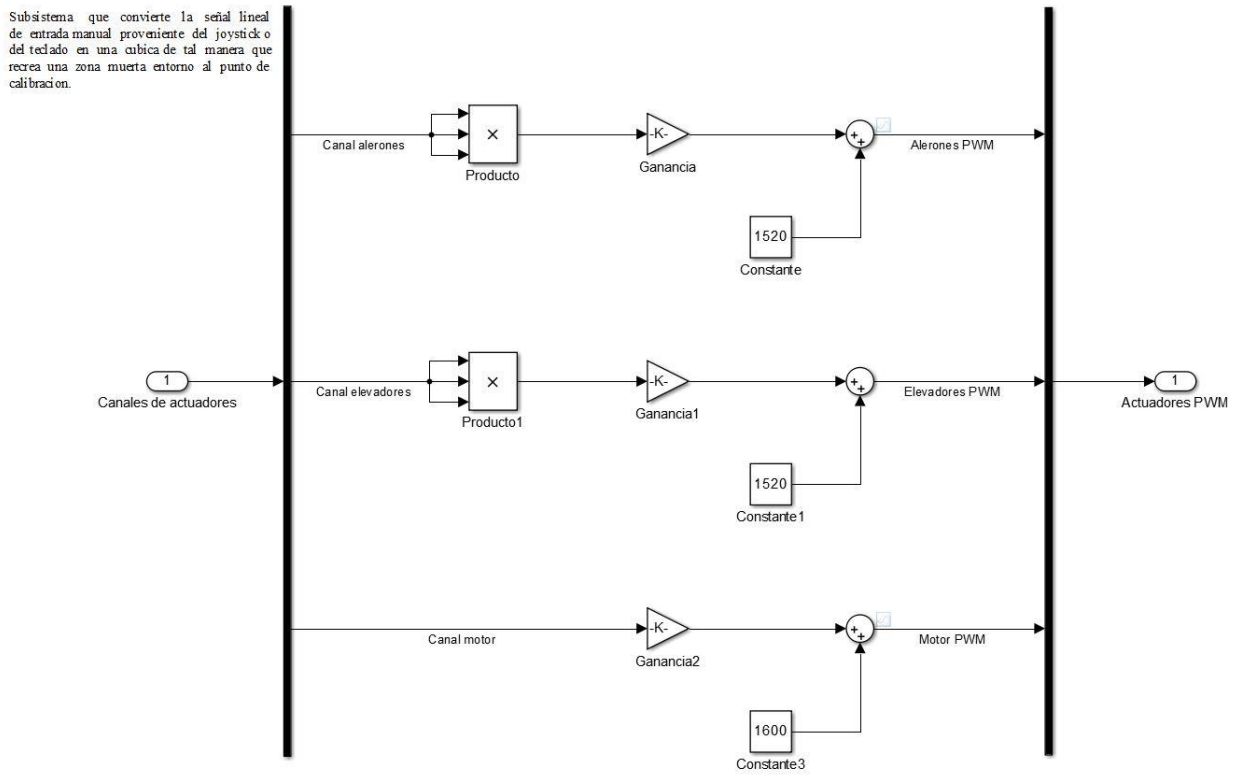

Figura 3-3. Bloque de leyes de mando.

Nótese que para reproducir esta zona muerta se ha implementado una función cúbica, con origen en el punto de calibración de cada uno de los ejes, de tal modo que suavice la derivada entorno al mismo. Esto provoca que cambios en las inmediaciones del origen no sean especialmente apreciables, tal y como ocurre con los mandos de vuelo que utilizan los pilotos en la vida real.

3.1.3 Leyes de actuación

Este sistema implementa las leyes que nos permiten transformar las respectivas señales PWM de los actuadores en magnitudes mecánicas. En este caso, los ángulos de deflexión de los elevones en su comportamiento como alerones (con sentidos opuestos) y elevadores (con el mismo sentido), y la fuerza ejercida por el grupo motor del Skywalker X8. Por un lado, las leyes de los alerones y elevadores no son más que una correspondencia lineal entre los ángulos de deflexión y la señal PWM. Por el otro lado, la ley de propulsión es una implementación en Simulink de aquella obtenida durante la etapa de modelado, la cual depende linealmente de la señal PWM del motor y cuadráticamente respecto a la componente longitudinal de la velocidad aerodinámica.

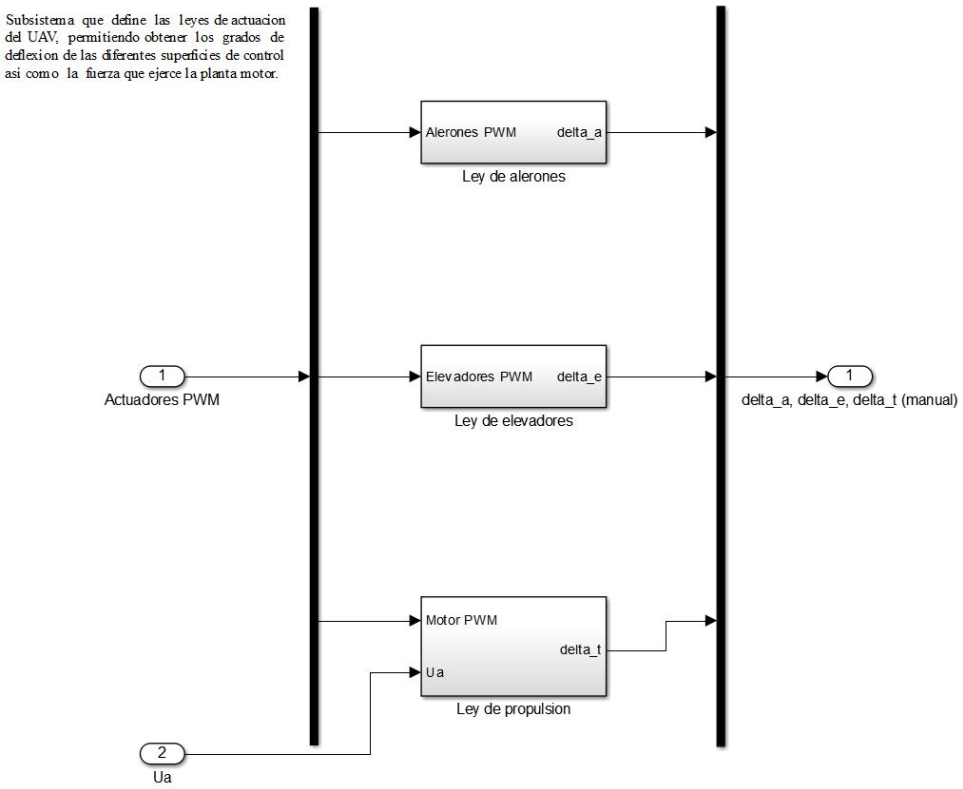

Figura 3-4. Bloque de leyes de actuación.

Se observa como las funciones se obtienen como resultado de subsistemas adicionales, cuyas salidas son finalmente guardadas en memoria para una futura representación de los mismos.

3.1.4 Autopiloto

Sin lugar a dudas, el autopiloto es el sistema que acarrea la mayor complejidad. Se compone de seis subsistemas fundamentales, uno por controlador implementado, los cuales permiten el control tanto de la dinámica lateral como longitudinal. En esencia el autopiloto recibe información de vuelo crítica proveniente de los bloques más importantes del simulador y trabaja con estos datos para garantizar una navegación automática. Los controladores desarrollados son los siguientes:

- Dinámica lateral:
    - Controlador de ángulo de balanceo (ϕ).
    - Controlador de rumbo (ψ).
- Dinámica longitudinal:
    - Controlador de ángulo de cabeceo (θ).
    - Controlador de altitud (h).
    - Controlador de velocidad aerodinámica (U_∞) mediante ángulo de cabeceo (θ).
    - Controlador de velocidad aerodinámica (U_∞) mediante fuerza propulsiva (δ_t).

El hecho de que el Skywalker X8 sea una aeronave de tipo ala volante, lo que implica la ausencia de un timón de cola, conlleva que no sea posible el control del ángulo de deslizamiento (β). Controlador que sí habría sido posible incluir si se hubiera optado por otro modelo de UAV.

Seguidamente, se muestra una captura del aspecto global del autopiloto que, aunque no permite visualizar los detalles de sus componentes, sirve para transmitir la idea de su complejidad:

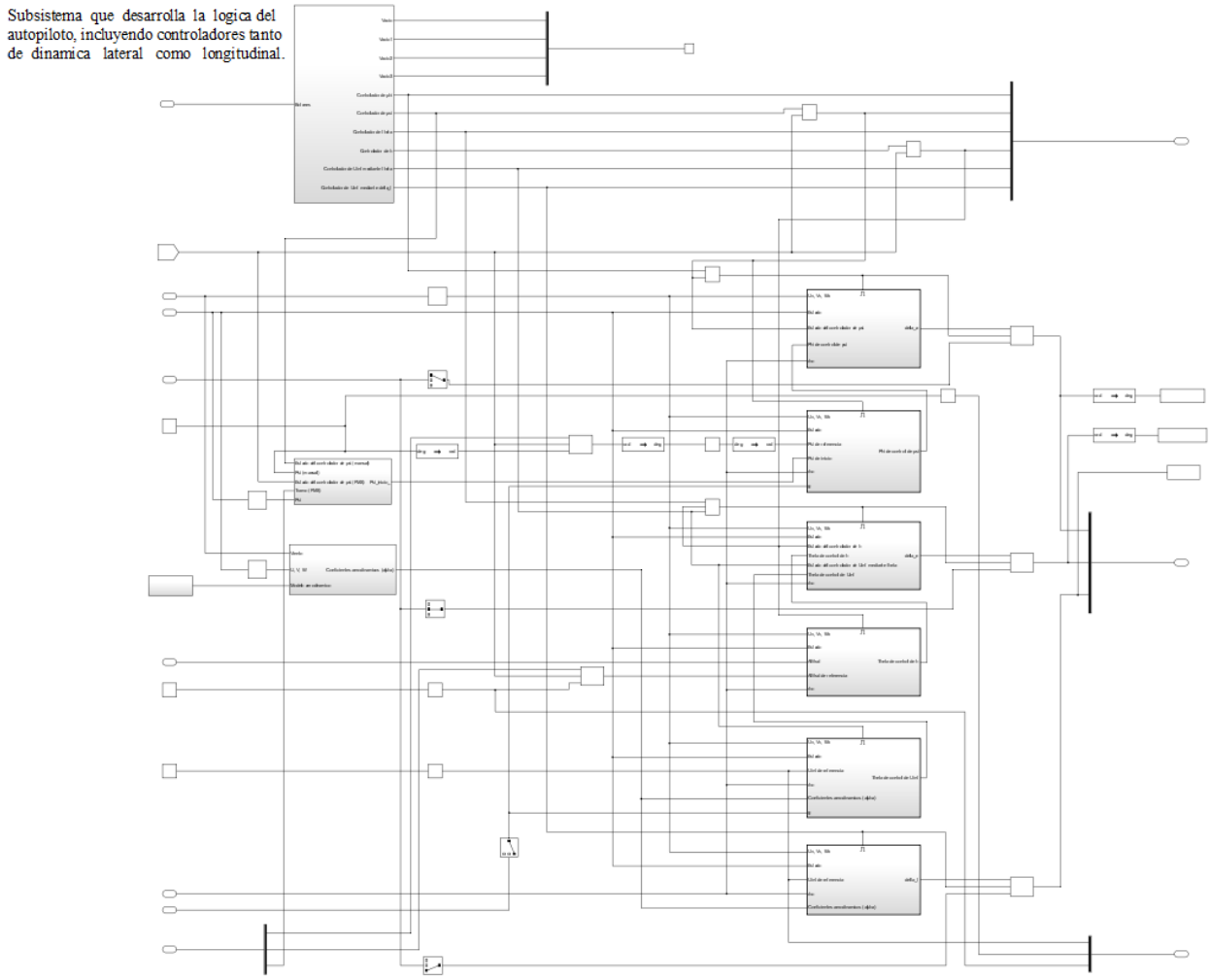

Figura 3-5. Bloque del autopiloto.

Los controladores se han desarrollado a semejanza de aquellos explicados en [3], con la única diferencia que la dinámica de cada uno no se ha calculado utilizando un modelo lineal entorno al punto de equilibrio de la aeronave, sino utilizando el modelo dinámico completo presentado en el capítulo anterior.

En los apartados siguientes se podrá apreciar como estos controladores se caracterizan por su sencillez, pues no son más que controladores de tipo PID, PI y PD. Debido a la gran cantidad de variables implicadas en cada uno de ellos, se ha optado por una afinación manual de los mismos que, pese a no ser óptima, es más que suficiente para satisfacer el comportamiento deseado. En consecuencia, aún existe margen de ajuste para la dinámica de los controladores y su optimización.

Las variables de referencia para los mismos se han calculado resolviendo el punto de equilibrio de la aeronave para un vuelo de crucero horizontal. Este equilibrio dinámico queda definido por los siguientes valores, expresados en el Sistema Internacional, de las variables de estado:

Variable de estado	Valor	Variable de estado	Valor
U_{ref}	14.9346	ϕ_{ref}	$3.34936 \cdot 10^{-6}$
V_{ref}	$8.04152 \cdot 10^{-5}$	θ_{ref}	0.0842084
W_{ref}	1.26060	ψ_{ref}	-
P_{ref}	$-1.57606 \cdot 10^{-20}$	$\delta_{a_{ref}}$	$1.37414 \cdot 10^{-6}$
Q_{ref}	$-1.67480 \cdot 10^{-17}$	$\delta_{e_{ref}}$	-0.00669962
R_{ref}	$-1.29450 \cdot 10^{-21}$	$\delta_{t_{ref}}$	1.21617

Tabla 3-1. Punto de equilibrio de vuelo de crucero horizontal.

A continuación, se exponen los distintos controladores junto a los diagramas que los definen extraídos de [3], donde la única diferencia respecto al implementado en este trabajo radica en el uso de un modelo dinámico lineal entorno al punto de equilibrio para el cálculo de las variables de entrada. No obstante, no se analizan las ecuaciones de los parámetros que definen la dinámica de los controladores. Simplemente se presentan sus respectivos diagramas de bloques y su implementación en Simulink.

3.1.4.1 Controlador de ángulo de balanceo (ϕ)

Este controlador PI permite estabilizar la aeronave siempre y cuando el ángulo de balanceo de referencia sea prácticamente nulo, como en este caso. Es útil a la hora de recuperar la aeronave tras, por ejemplo, una ráfaga de viento de gran intensidad. Sin embargo, también se activa al usar el controlador de rumbo, pues de él recibe la ϕ_{ref} que permite alcanzar la dirección deseada. Su diagrama de bloques es el siguiente:

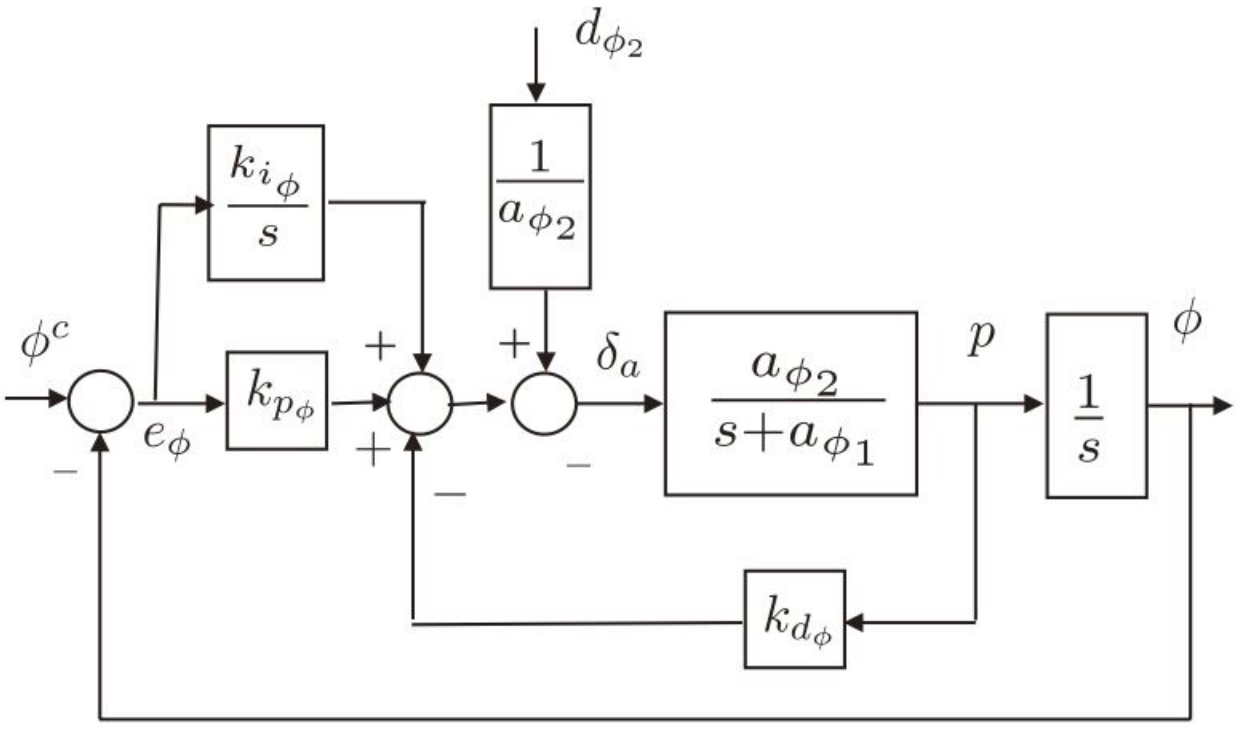

Figura 3-6. Diagrama de bloques del controlador de ángulo de balanceo.

Como es lógico, el control del ángulo de balanceo se efectúa mediante la actuación de los alerones, cuyos ángulos de deflexión (opuestos) se envían a los bloques de dinámica que simulan el comportamiento de la aeronave. El controlador ha sido implementado en Simulink de la siguiente manera:

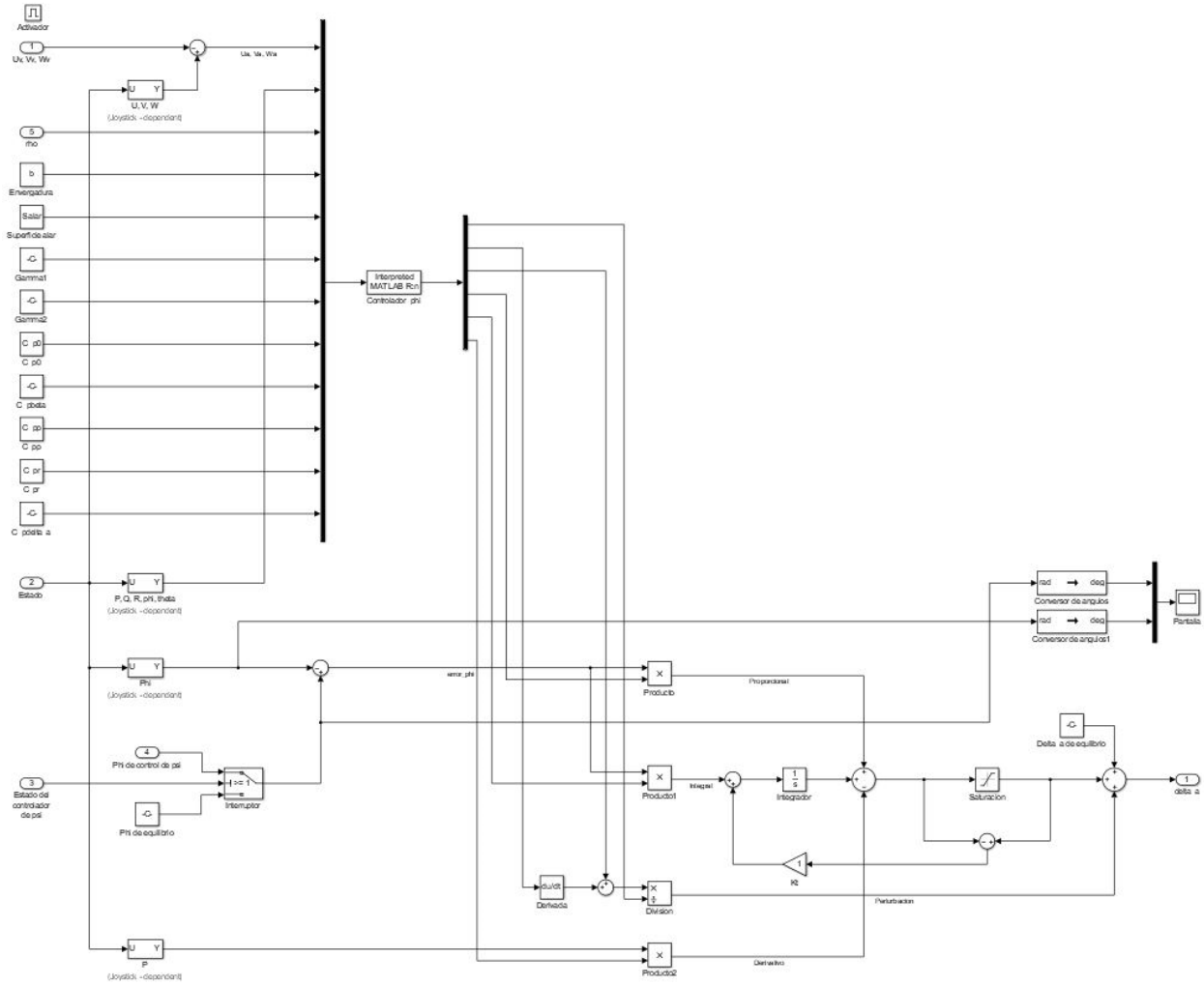

Figura 3-7. Controlador de ángulo de balanceo en Simulink.

3.1.4.2 Controlador de rumbo (ψ)

El controlador de rumbo es también de tipo PI y permite dirigir el Skywalker X8 hacia un rumbo seleccionado. Para ello, su salida, que es el ángulo de alabeo de referencia, se envia al controlador previamente presentado con el fin de conducir a la aeronave hacia dicha dirección. Es uno de los controladores más difíciles de ajustar puesto que la dinámica de la aeronave está acoplada con la del controlador del ángulo de alabeo. Este es su diagrama de bloques:

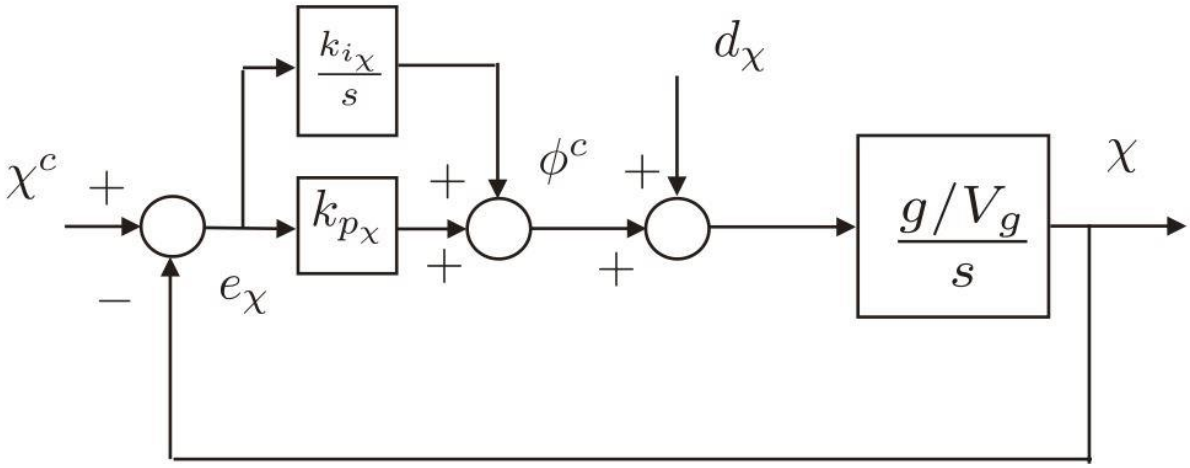

Figura 3-8. Diagrama de bloques del controlador de rumbo.

Al igual que sucedía con el controlador de ángulo de alabeo, y al estar íntimamente vinculado al mismo, su control es llevado a cabo gracias a los alerones. Esta ha sido su implementación en Simulink:

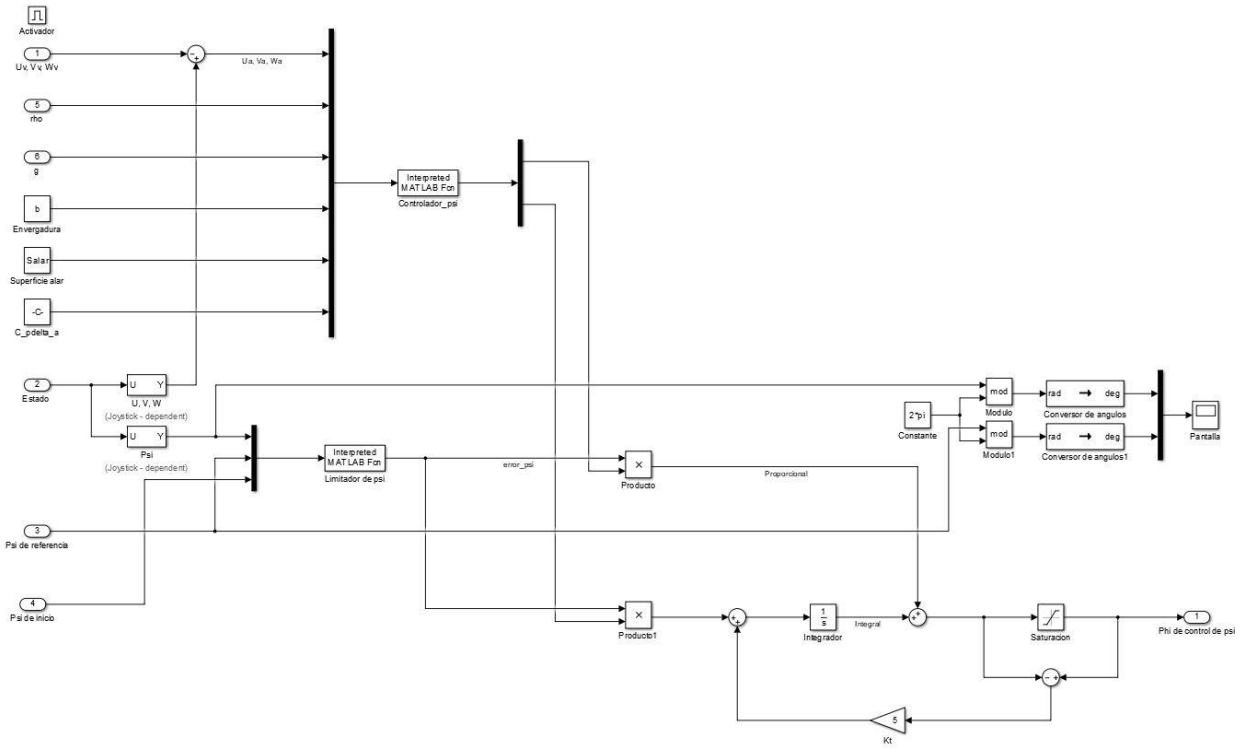

Figura 3-9. Controlador de rumbo en Simulink.

Se aprecia como, aparte de los componentes presentados en el diagrama de bloques, se ha incluido una ligera ganancia en retroalimentación para que actúe como *antiwindup*, lo que suavizará la dinámica del controlador al activarse cuando el módulo del error sea suficientemente grande.

3.1.4.3 Controlador de ángulo de cabeceo (θ)

Este es el único controlador de tipo PD implementado. Su misión es controlar el ángulo de cabeceo del Skywalker X8, pudiéndose utilizar, de manera análoga a como sucedía en el controlador de ángulo de balanceo, como estabilizador de la dinámica longitudinal de la aeronave en momentos determinados. Puesto que carece de control integral, siempre existirá un pequeño error estacionario que no será posible de suprimir; no obstante, esto posibilita una actuación más rápida por parte del controlador. He aquí su diagrama de bloques correspondiente:

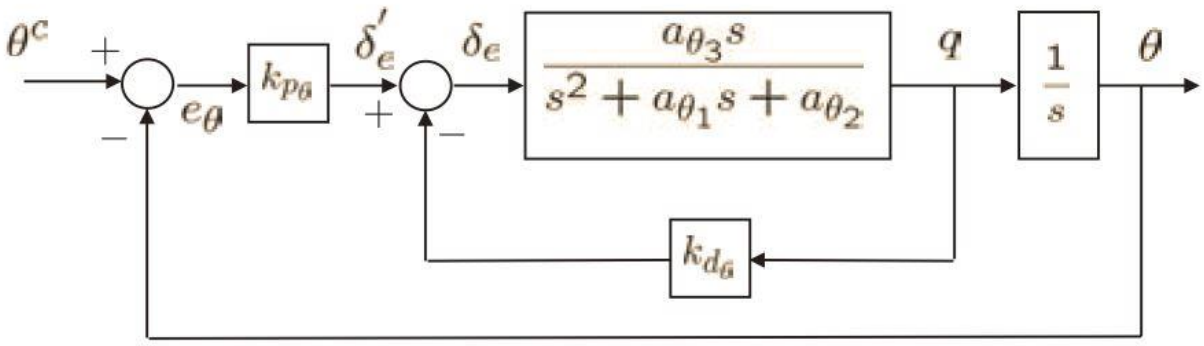

Figura 3-10. Diagrama de bloques del controlador de cabeceo.

Esta vez, al tratarse de un controlador de dinámica longitudinal, los actuadores involucrados no son otros que los elevadores, en este caso, los elevones deflectando hacia el mismo sentido ya que el Skywalker X8, al ser un ala volante, tiene acoplada la dinámica de los alerones y los elevadores. Estos ángulos de deflexión terminan formando parte de la entrada en el bloque de dinámica del simulador. Este es el aspecto que tiene el controlador tras su implementación en Simulink:

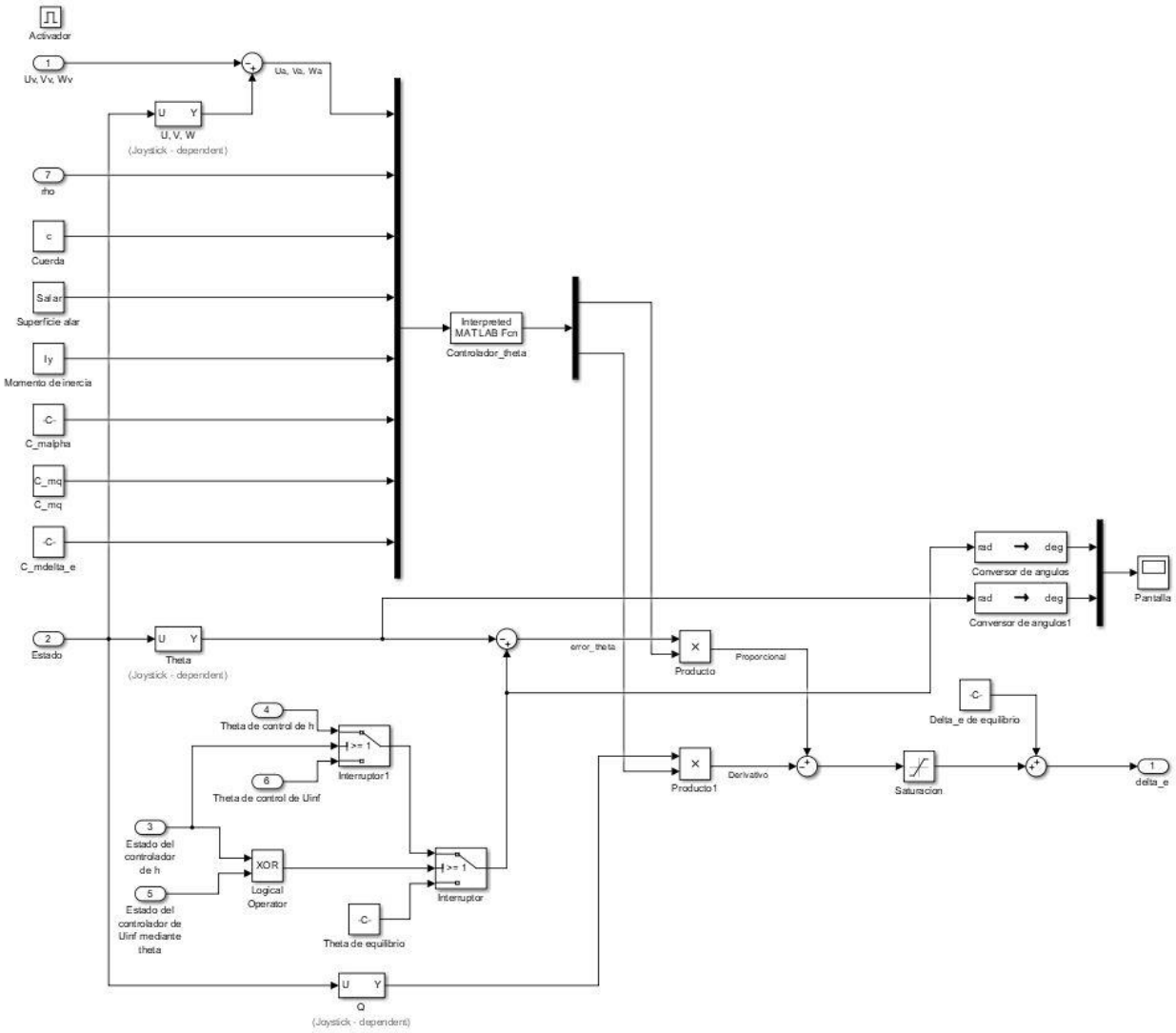

Figura 3-11. Controlador de ángulo de cabeceo en Simulink.

Como se comentará más adelante, este controlador también recibe θ_{ref} de los controladores de altitud (h) y de velocidad aerodinámica total (U_∞) tal que permita alcanzar los valores de referencia de estos últimos. Es decir, sus dinámicas estan acopladas, lo que provocará la activación automática del controlador de ángulo de cabeceo cuando el de altitud o el de velocidad aerodinámica total estén trabajando.

3.1.4.4 Controlador de altitud (h)

De forma análoga al controlador de rumbo, el de altitud se basa en enviar una θ_{ref} al controlador de ángulo de cabeceo tal que permita satisfacer la referencia de altitud en cada instante de tiempo. Este controlador, de tipo PI, está limitado a una altitud máxima de 500 metros. El motivo es que, pese a que en el simulador se logran alcanzar altitudes mayores, en los datos disponibles de telemetría el Skywalker apenas sobrepasa los 200 metros de altitud. Por ello, el modelo obtenido pierde fiabilidad fuera de ese rango, aunque se consideraría aceptable hasta esos 500 metros pues se hallan por debajo del techo de vuelo teórico. Ahora, se muestra el diagrama de bloques de el controlador de altitud:

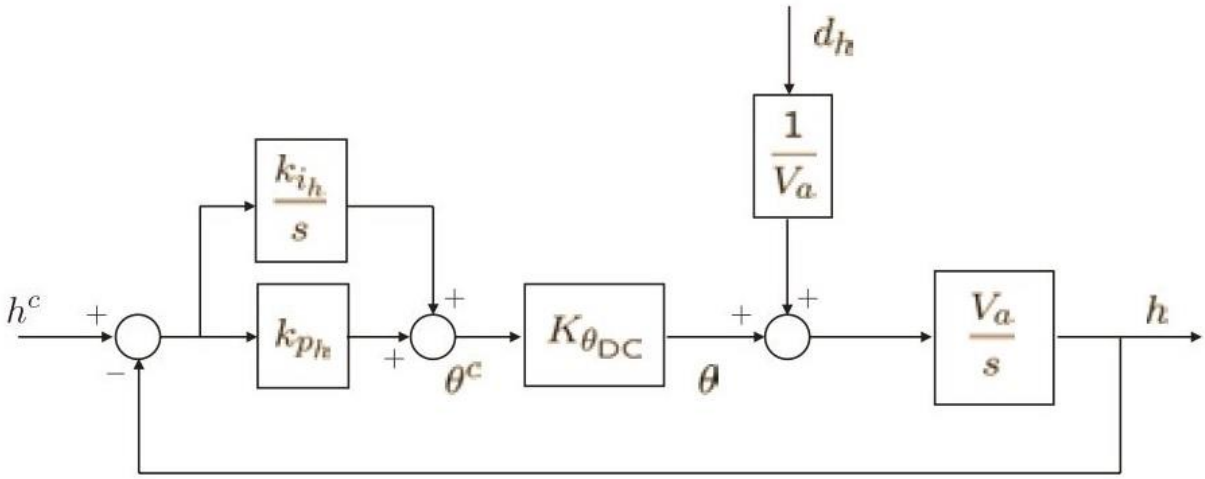

Figura 3-12. Diagrama de bloques del controlador de altitud.

Obviamente, la dinámica de este sistema es controlada mediante los elevadores, o lo que es lo mismo, los elevones deflectando hacia el mismo sentido. La única diferencia de este diagrama de bloques respecto a su aplicación en el simulador ha sido la adición de una componente de *antiwindup*. He aquí su implementación en Simulink:

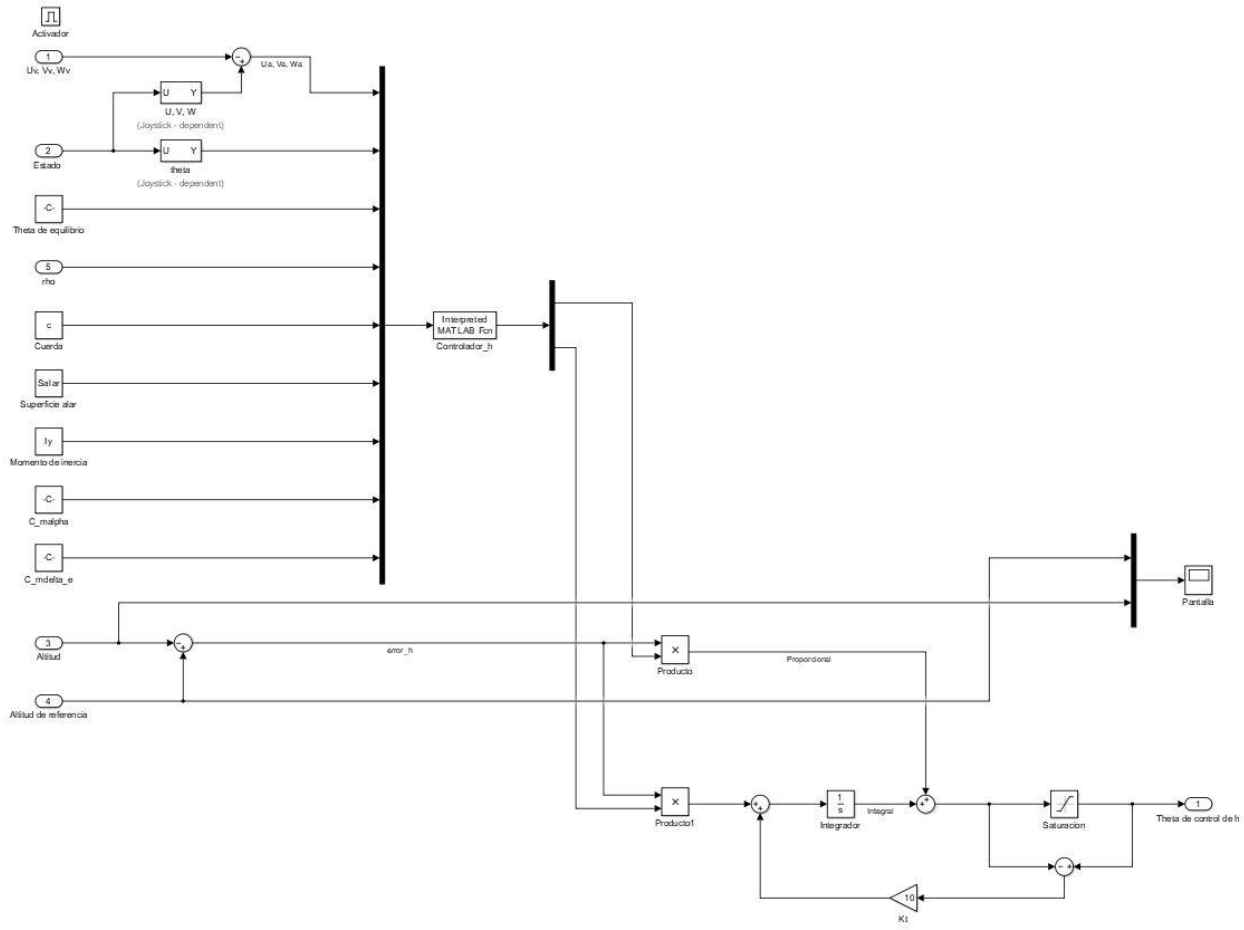

Figura 3-13. Controlador de altitud en Simulink.

3.1.4.5 Controlador de velocidad aerodinámica (U_∞) mediante ángulo de cabeceo (θ)

El objetivo de este controlador es alcanzar una velocidad aerodinámica de referencia, la cual es medida gracias al tubo de Pitot, a partir del control del ángulo de cabeceo. Se basa en el hecho de que la resistencia aerodinámica de la aeronave depende del ángulo de ataque al que se vuele. No obstante, existe la posibilidad de no alcanzar esta velocidad de referencia si el UAV se encuentra bajo un régimen de motor que no le permita alcanzar dicha velocidad. Consecuentemente, habrá que asegurarse de que el régimen de motor sea suficiente para cumplir con la referencia solicitada. El diagrama de bloques de este controlador PI tiene el siguiente aspecto:

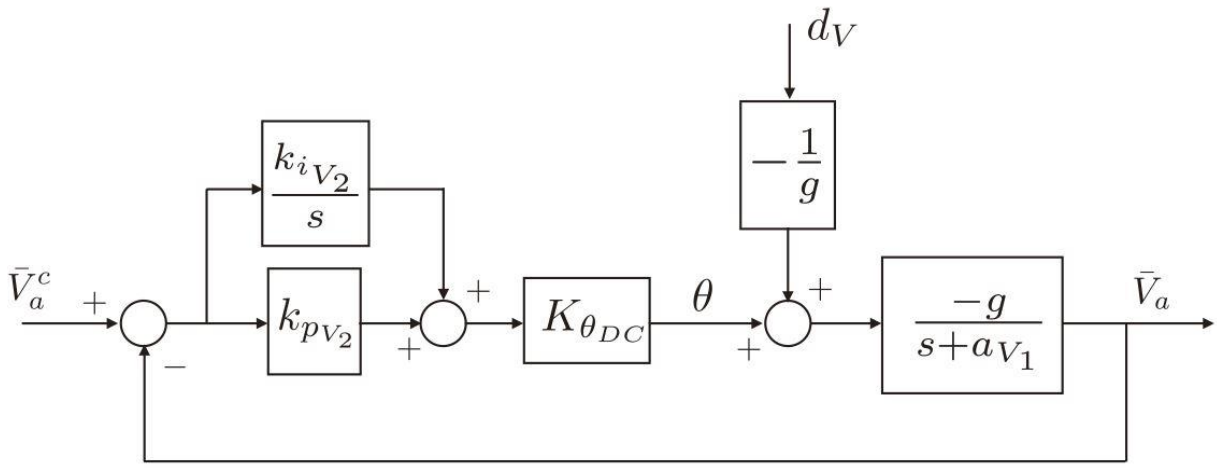

Figura 3-14. Diagrama de bloques del controlador de velocidad aerodinámica mediante el ángulo de cabeceo.

Sin lugar a dudas, su dinámica es controlada mediante los elevones actuando como elevadores. Además, como viene siendo ya habitual, se incluye una componente *antiwindup* que permite suavizarla, activándose cuando el error (diferencia entre el valor de la variable y su referencia) se encuentra en saturación. Su salida no es otra que el valor de referencia del ángulo de referencia de cabeceo, el cual es enviado a su controlador respectivo. Seguidamente, se presenta su desarrollo en Simulink:

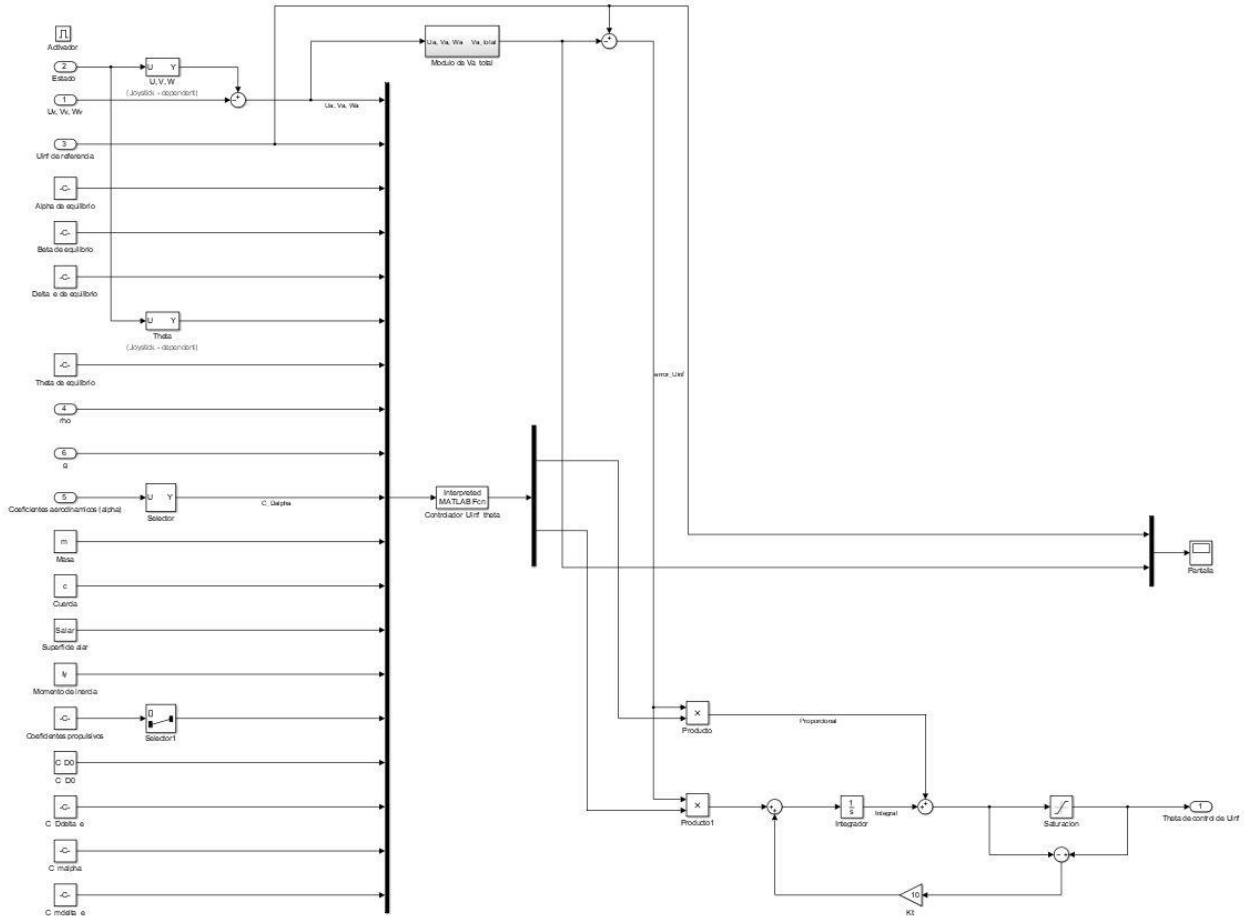

Figura 3-15. Controlador de velocidad aerodinámica mediante ángulo de cabeceo.

3.1.4.6 Controlador de velocidad aerodinámica (U_∞) mediante fuerza propulsiva (δ_t)

Este controlador de velocidad aerodinámica tiene la misma finalidad que el anterior, sólo que distinto medio. Es decir, alcanza el valor de referencia mediante la fuerza de propulsión que proporciona el motor. Su dinámica es más rápida que la del controlador presentado previamente y su rango de actuación mucho mayor ya que, dejando el resto de variables de estado intactas, el régimen de motor tiene mayor impacto sobre la velocidad aerodinámica que el ángulo de cabeceo. El diagrama de bloques que se ha seguido para su implementación es mostrado a continuación:

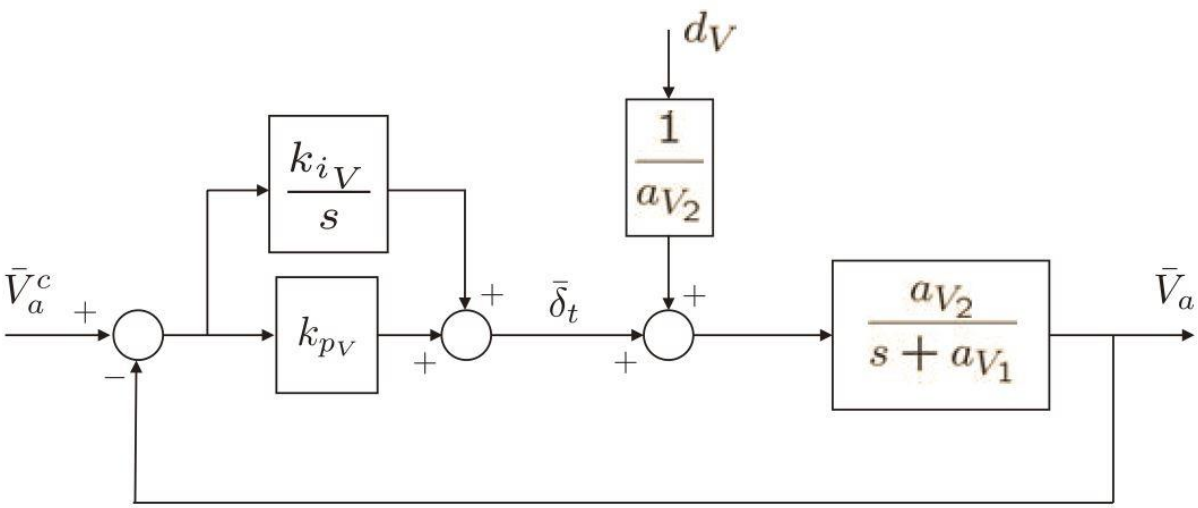

Figura 3-16. Diagrama de bloques del controlador de velocidad aerodinámica mediante fuerza propulsiva.

En este caso, la actuación del controlador se lleva a cabo mediante el régimen de motor de la aeronave. Para este controlador también se ha añadido retroalimentación en forma de *antiwindup*. Su implementación, como se puede apreciar, es similar a aquella del controlador previo:

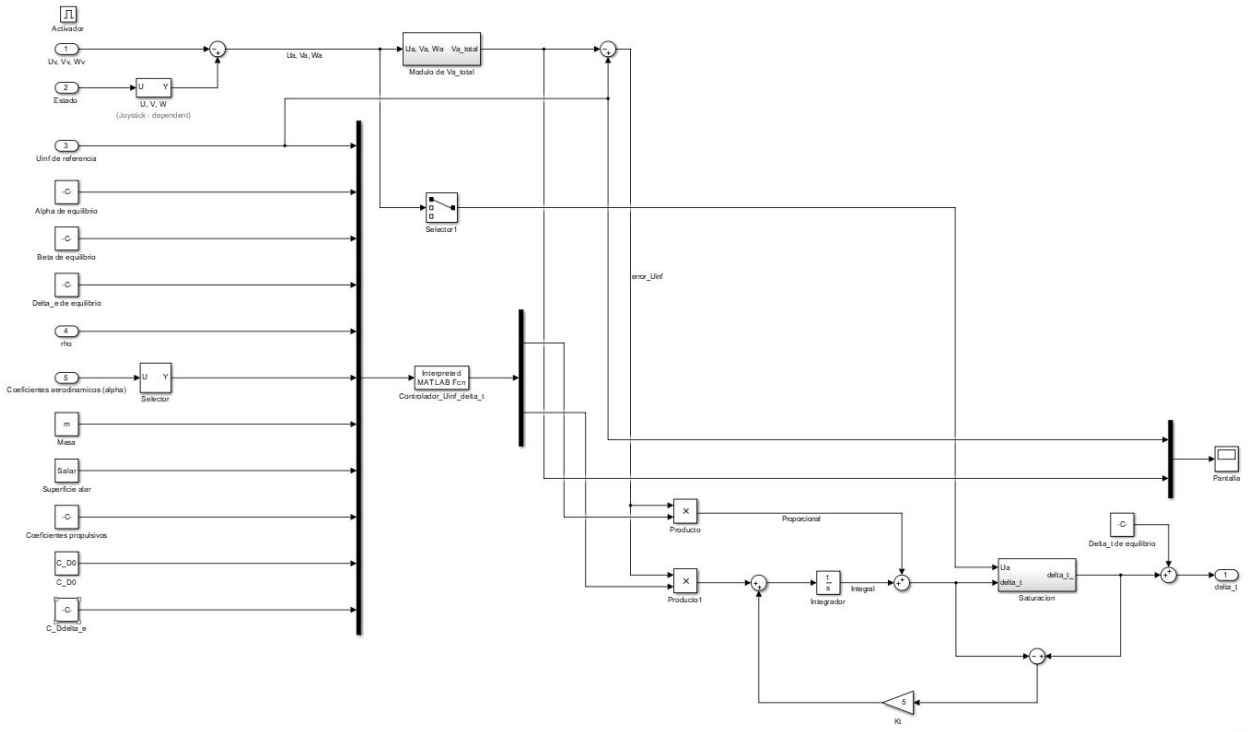

Figura 3-17. Controlador de velocidad aerodinámica mediante fuerza propulsiva.

3.1.5 Dinámica

La finalidad de este bloque no es otra que la de calcular las fuerzas y los momentos que experimenta la aeronave en cada instante de simulación. Para ello, es necesario tomar como entrada multitud de parámetros entre los que destacan las variables de estado en el instante de simulación previo, variables ambientales (como la densidad, el viento o la aceleración gravitatoria) y variables resultantes de la aplicación del propio modelo aerodinámico. Gracias a estos parámetros, se aplican las ecuaciones mostradas en la subsección de fuerzas y momentos del capítulo anterior.

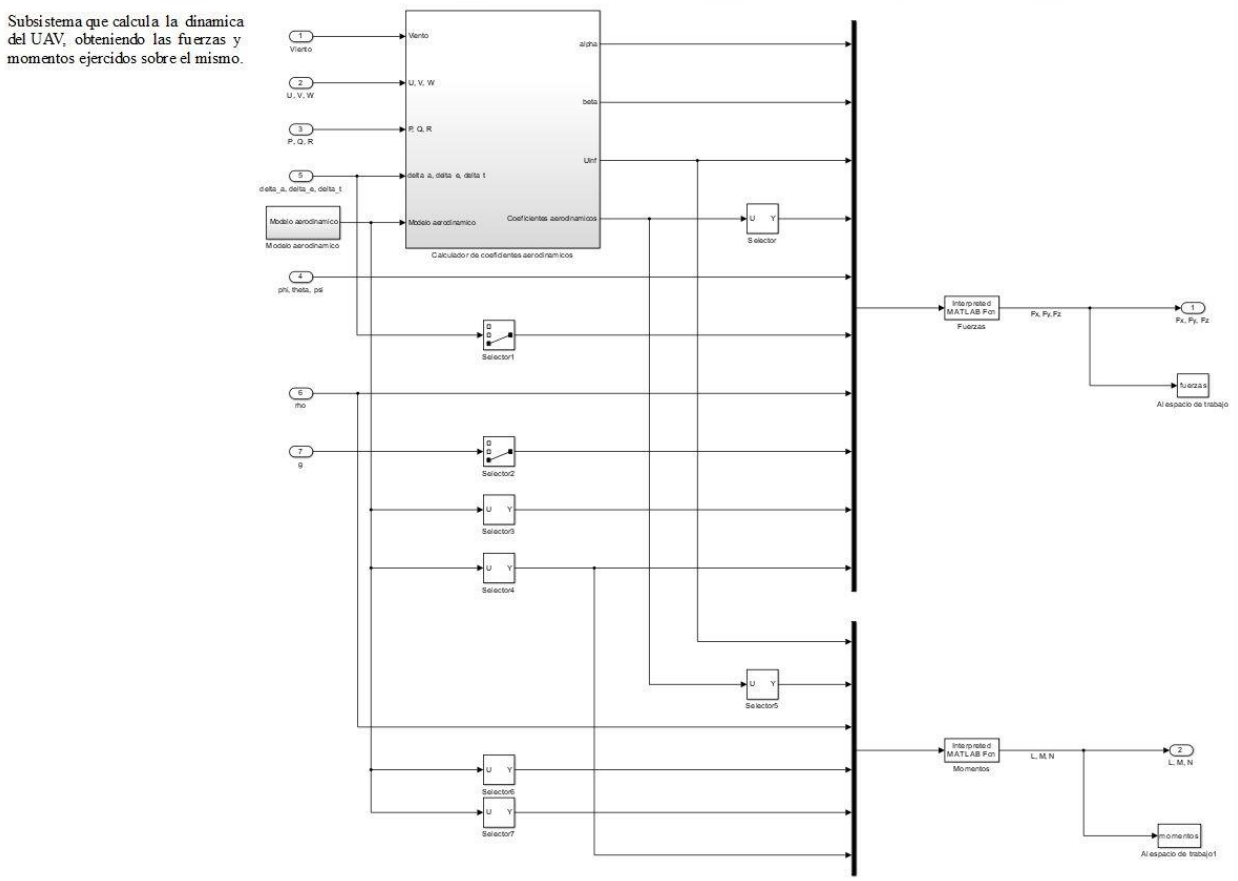

Figura 3-18. Bloque de dinámica.

El resultado emitido por este bloque forma parte de la entrada en el siguiente bloque de cinemática, tal y como se muestra a continuación. Finalmente, la salida es almacenada para su posterior representación.

3.1.6 Cinemática

Del mismo modo a como ocurre en el bloque de dinámica, este bloque está únicamente orientado al cálculo de las velocidades lineales y angulares de la aeronave, las cuales son expresadas en ejes cuerpo. Para el cálculo de las ecuaciones, las mismas que fueron mostradas en la subsección dedicada a la dinámica en el capítulo previo, es necesario tomar como entrada parte de las variables del bloque de dinámica, exceptuando las pertenecientes al modelo aerodinámico. Después, las aceleraciones resultantes son integradas para hallar los valores de las componentes de velocidad.

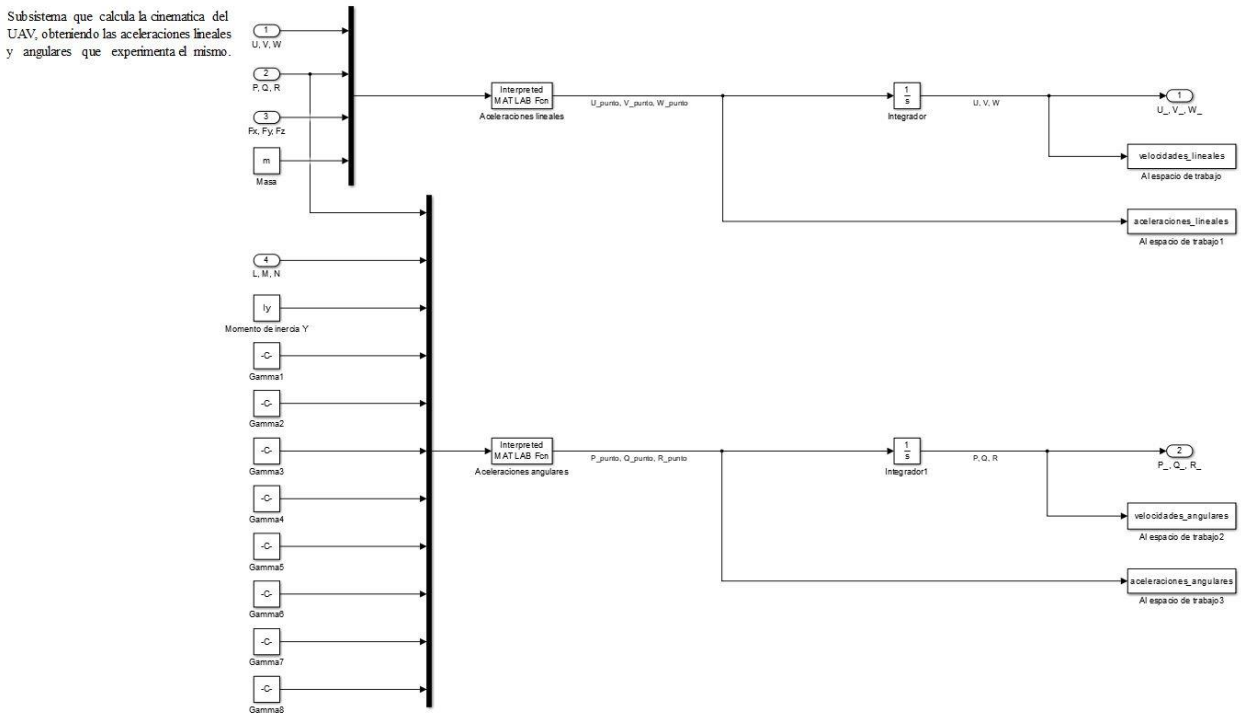

Figura 3-19. Bloque de cinemática.

Análogamente, la salida, esto es, las velocidades lineales y angulares, así como las aceleraciones lineales y angulares son guardadas para su ulterior representación, pues se consideran variables de estado de gran relevancia.

3.1.7 Posición y actitud

Finalmente, este bloque, que toma únicamente como entrada las variables de estado, implementa el cálculo presentado en la subsección de cinemática del capitulo anterior para obtener las derivadas de la posición, expresada en ejes de navegación, y la actitud. Lógicamente, estos resultados son integrados inmediatamente después, al mismo tiempo que ambas ternas de datos son guardadas tal y como se ha ido haciendo hasta ahora con aquellos datos de especial trascendencia.

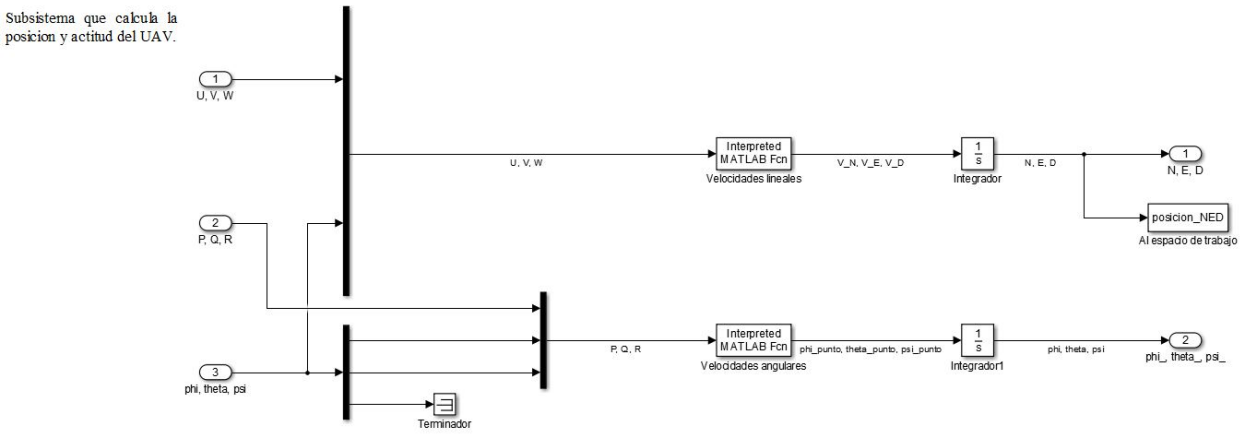

Figura 3-20. Bloque de posición y actitud.

3.1.8 Modelo de viento

El bloque de modelo de viento implementado no es otro que el modelo Dryden de viento, explicado en el capítulo anterior. A partir de la altitud, las tres componentes de velocidad y la orientación es capaz de obtener las componentes de velocidad de la masa de aire alrededor de la aeronave y el efecto rotacional del viento sobre la misma, expresado como una terna de velocidades angulares. La introducción de este modelo, no obstante, no implica que siempre esté activado, pues se ha incluido un interruptor que permite controlar su estado en caso de no querer incluir su efectos en la simulación.

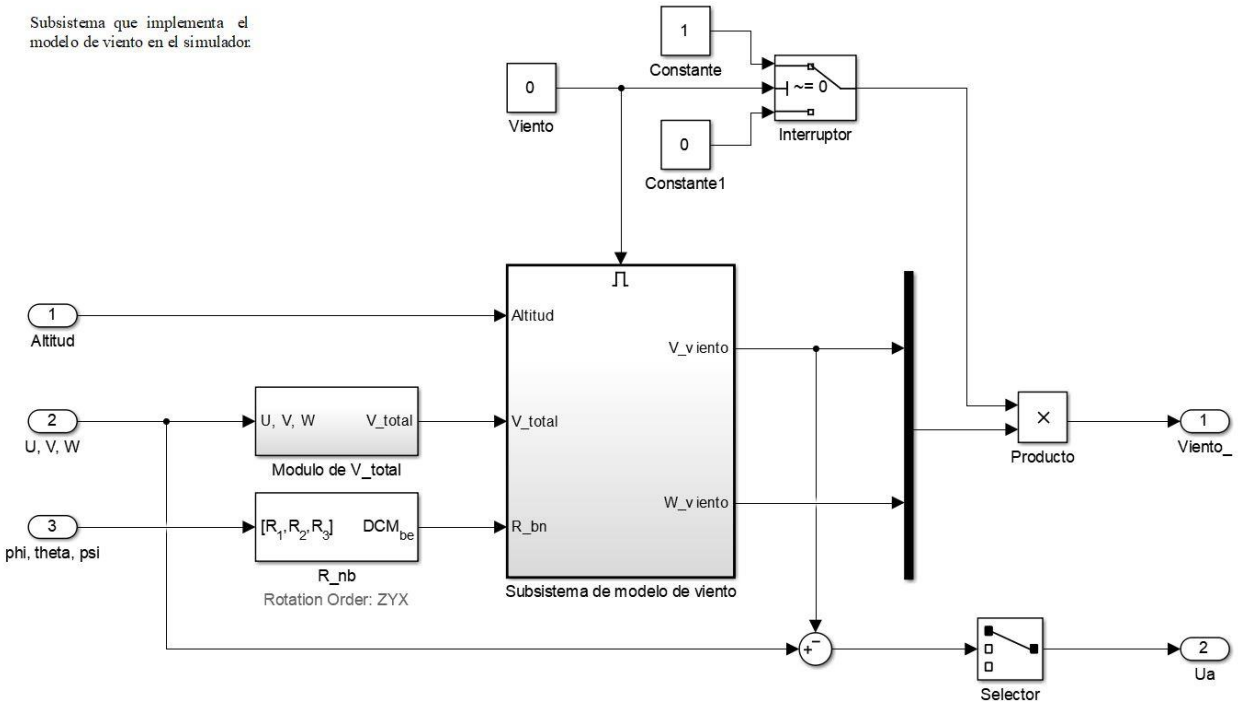

Figura 3-21. Bloque de modelo de viento.

3.1.9 Modelo de atmósfera

El modelo de atmósfera elegido es el modelo de Atmósfera Estándar Internacional o ISA (*International Standard Atmosphere*). El propósito de este bloque es únicamente el cálculo de la densidad, para lo cual toma como entrada el valor de la altitud, necesaria para la obtención de los diversos coeficientes aerodinámicos. Estas han sido las ecuaciones utilizadas:

$$T(h) = T_0 + a \cdot (h - h_0)$$

$$\rho(h) = \rho_0 \cdot \left(\frac{T(h)}{T_0}\right)^{-\frac{g}{a \cdot R} - 1}$$

Siendo R, la constante del aire $\left(R = 287\ \frac{m^2}{s^2 \cdot K}\right)$; a, el gradiente térmico en la troposfera $\left(a = -6.5\ \frac{K}{km}\right)$; g, la aceleración de la gravedad (dependiente de las coordenadas geodéticas del UAV, según el modelo WGS84); h_0, la altitud geopotencial base en la troposfera ($h_0 = 0\ m$); T_0, la temperatura base en la troposfera ($T_0 = 288.15\ K$); y ρ_0, la densidad base en la troposfera $\left(\rho_0 = 1.2252\ \frac{kg}{m^3}\right)$.

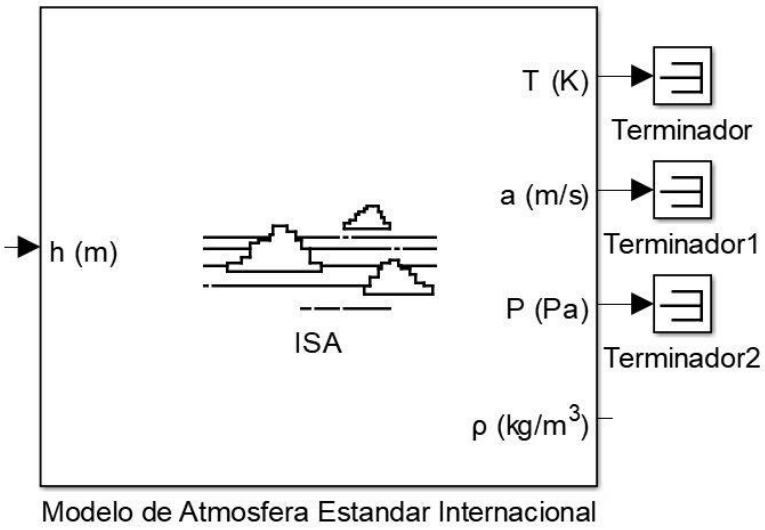

Figura 3-22. Bloque de modelo de atmósfera.

3.1.10 Modelo de gravedad

En este caso, se implementa el modelo gravitatorio WGS84 también explicado en el capítulo anterior. Su finalidad es hallar la aceleración gravitatoria en función de las coordenadas geodéticas del Skywalker X8, ya que es necesaria a la hora de calcular su dinámica. Su implementación, al igual que ocurre con el modelo atmosférico, se lleva a cabo mediante su correspondiente sistema de la "caja de herramientas aeroespacial" (*Aerospace Toolbox*) en Simulink, sin necesidad de elaborar uno dedicado.

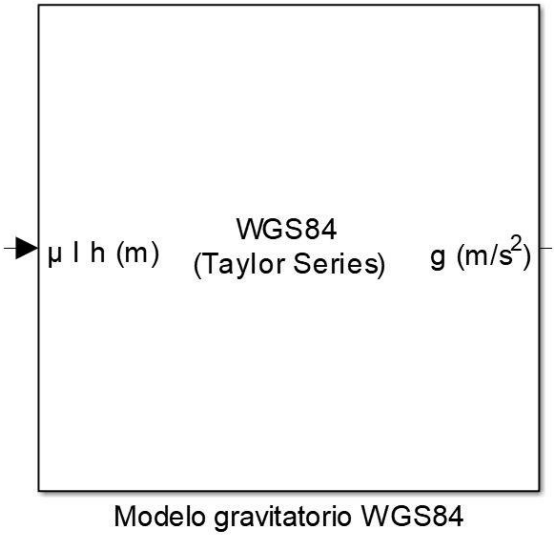

Figura 3-23. Bloque de modelo de gravedad.

3.1.11 Postprocesamiento

Como su nombre indica, el bloque de postprocesamiento está destinado a procesar variables de simulación obtenidas como salidas de determinados subsistemas las cuales se necesitan acondicionar (reordenarlas, realizar cálculos simples con ellas, cambiar su dominio, etc) para su entrada en otros bloques. Entre las variables a procesar se encuentran las coordenadas geodéticas, la posición expresada en ejes de navegación y la actitud de la aeronave.

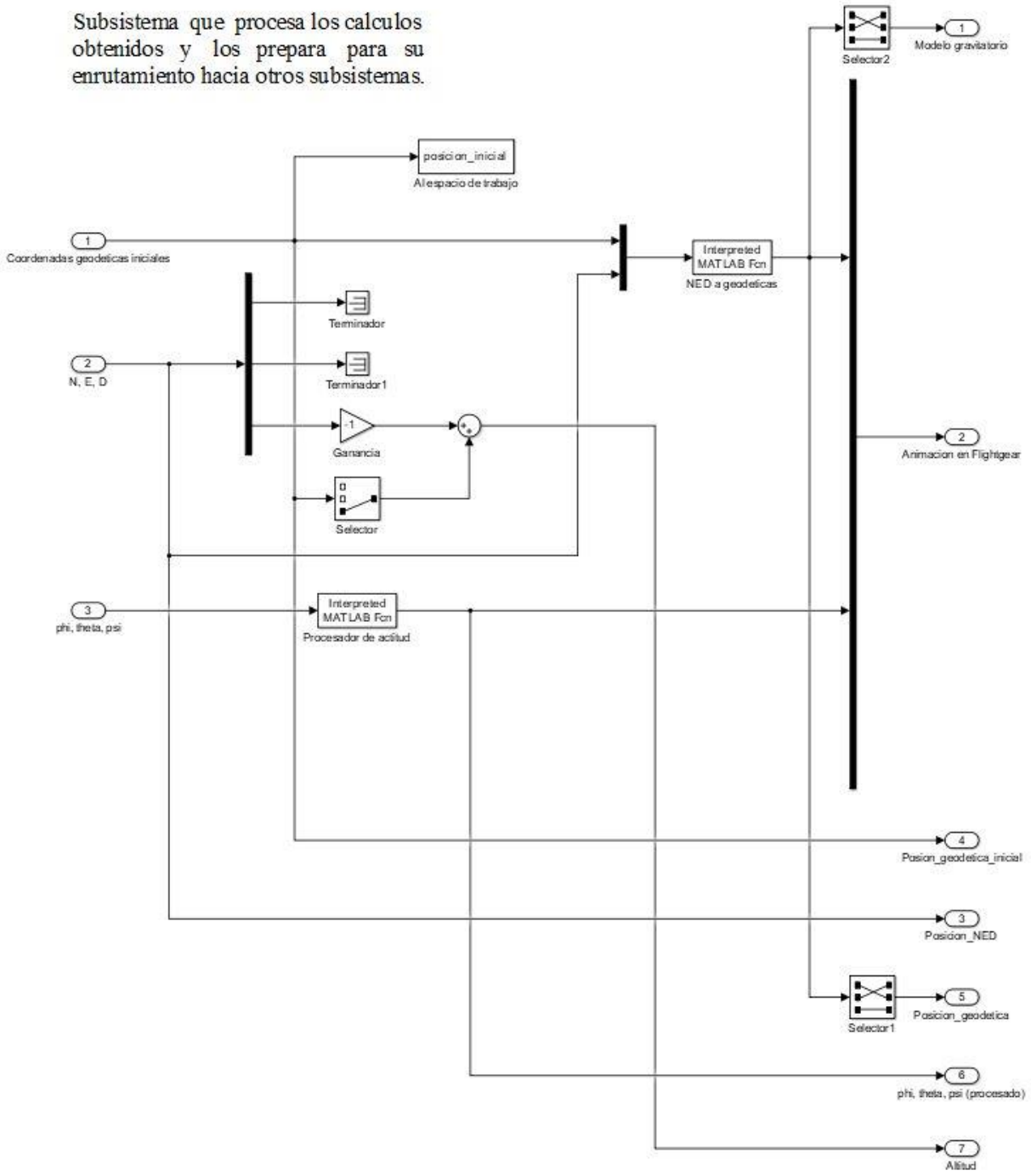

Figura 3-24. Bloque de postprocesamiento.

3.1.12 Misión

El bloque de misión es, junto con el de autopiloto, de aquellos que han requerido una mayor elaboración. Su papel es de gran relevancia dentro del simulador, pues permite el seguimiento de rutas de forma autónoma en su actuación conjunta con el autopiloto. Este tipo de sistemas se conoce comúnmente en el ámbito de la aeronáutica comercial como sistema de gestión de vuelo o FMS (*Flight Management System*). Asismismo, dispone de un subsistema destinado a enviar información de vuelo (posición y rumbo) a una segunda instancia abierta de Matlab mediante protocolo UDP (enviando paquetes de datos a través de la red local) de tal forma que se simule un sistema GNSS.

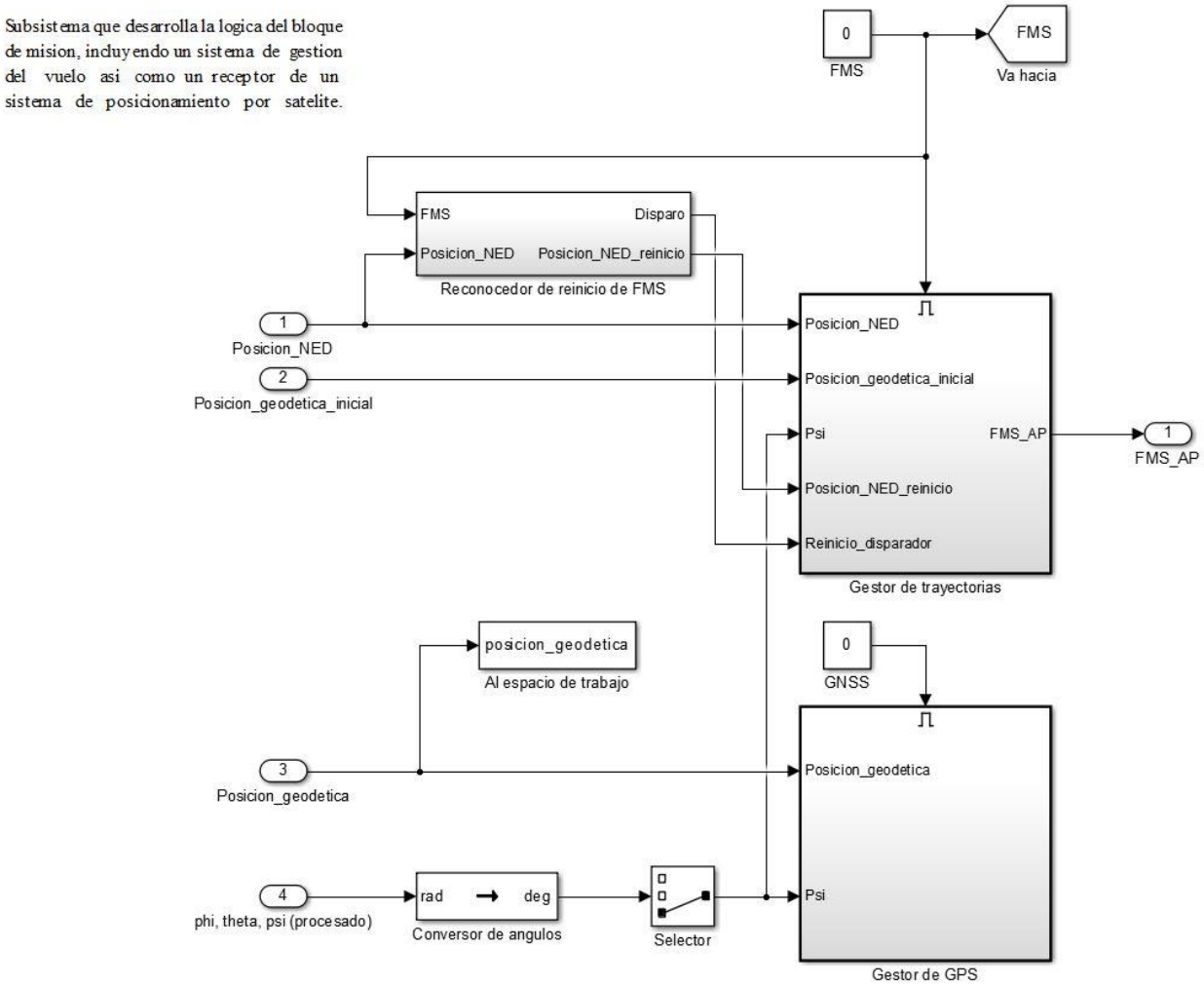

Figura 3-25. Bloque de misión.

El subsistema destinado al FMS implementa dos funciones: una cuyo objetivo es la gestión de la ruta, a partir de los puntos de paso o *waypoints* marcados por el usuario de la aeronave, y otra para el seguimiento de la ruta, siendo sus trayectorias lineas rectas entre *waypoints*.

La primera función mencionada aplica la lógica que permite determinar cuándo se ha alcanzado un *waypoint*, de manera que se le pueda asignar al Skywalker X8 una nueva senda de vuelo que lo dirija hacia el siguiente *waypoint* de la ruta. Para ello, se ha implementado una adaptación del siguiente algoritmo:

Algorithm 5 Follow Waypoints: $(\mathbf{r}, \mathbf{q}) = \text{followWpp}(\mathcal{W}, \mathbf{p})$

Input: Waypoint path $\mathcal{W} = \{\mathbf{w}_1, \ldots, \mathbf{w}_N\}$, MAV position $\mathbf{p} = (p_n, p_e, p_d)^\top$.

Require: $N \geq 3$

1: **if** New waypoint path $\mathcal{W}$ is received **then**
2: Initialize waypoint index: $i \leftarrow 2$
3: **end if**
4: $\mathbf{r} \leftarrow \mathbf{w}_{i-1}$
5: $\mathbf{q}_{i-1} \leftarrow \frac{\mathbf{w}_i - \mathbf{w}_{i-1}}{\|\mathbf{w}_i - \mathbf{w}_{i-1}\|}$
6: $\mathbf{q}_i \leftarrow \frac{\mathbf{w}_{i+1} - \mathbf{w}_i}{\|\mathbf{w}_{i+1} - \mathbf{w}_i\|}$
7: $\mathbf{n}_i \leftarrow \frac{\mathbf{q}_{i-1} + \mathbf{q}_i}{\|\mathbf{q}_{i-1} + \mathbf{q}_i\|}$
8: **if** $\mathbf{p} \in \mathcal{H}(\mathbf{w}_i, \mathbf{n}_i)$ **then**
9: Increment $i \leftarrow (i+1)$ until $i = N - 1$
10: **end if**
11: **return** $\mathbf{r}, \mathbf{q} = \mathbf{q}_{i-1}$ at each time step

Figura 3-26. Algoritmo de gestión de ruta.

El criterio de este algoritmo para actualizar la senda de vuelo y permitir a la aeronave continuar la ruta se basa en discernir si esta ha traspasado o no la línea imaginaria que separa el semiplano que contiene el *waypoint* al cual se dirige y el anterior, y el semiplano que contiene el *waypoint* al cual se dirige y el siguiente. En resumen, si ha pasado de un semiplano a otro, tal y como se muestra aquí:

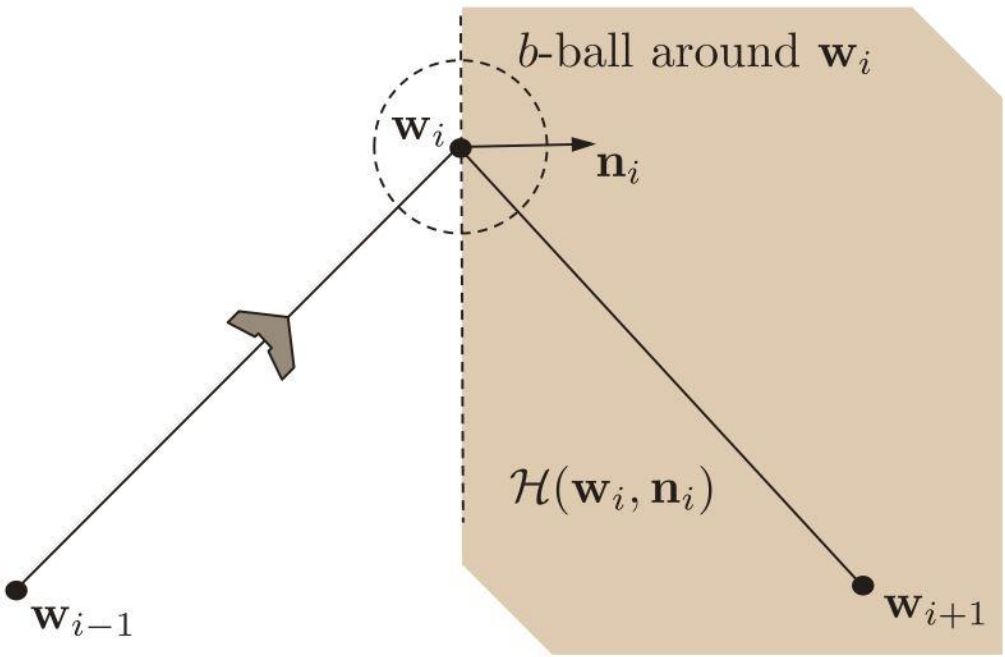

Figura 3-27. Criterio de actualización de segmento de ruta.

La actuación del mismo difiere de aquella incluida en aeronaves de tipo comercial en cuanto a que en este caso el Skywalker X8 llega a sobrevolar las coordenadas precisas del *waypoint*, debido a la necesidad de cumplir un objetivo concreto de la misión (fotografía, cartografía, búsqueda y rescate, vigilancia, etc). En cambio, la ruta seguida por un avión de pasajeros, por ejemplo, no tiene otra finalidad aparte del vuelo eficiente y ordenado en su camino desde el aeropuerto de origen hacia el de destino como un agente más del espacio aéreo. Se podría decir pues que, generalmente, en aviación comercial los *waypoints* juegan un papel más orientativo que en el caso de un vehículo no tripulado (prueba de ello es la reestructuración de rutas en pleno vuelo por orden de controladores aéreos). A continuación, se muestra un ejemplo en el que se observa el patrón de seguimiento de una ruta utilizando el algoritmo presentado:

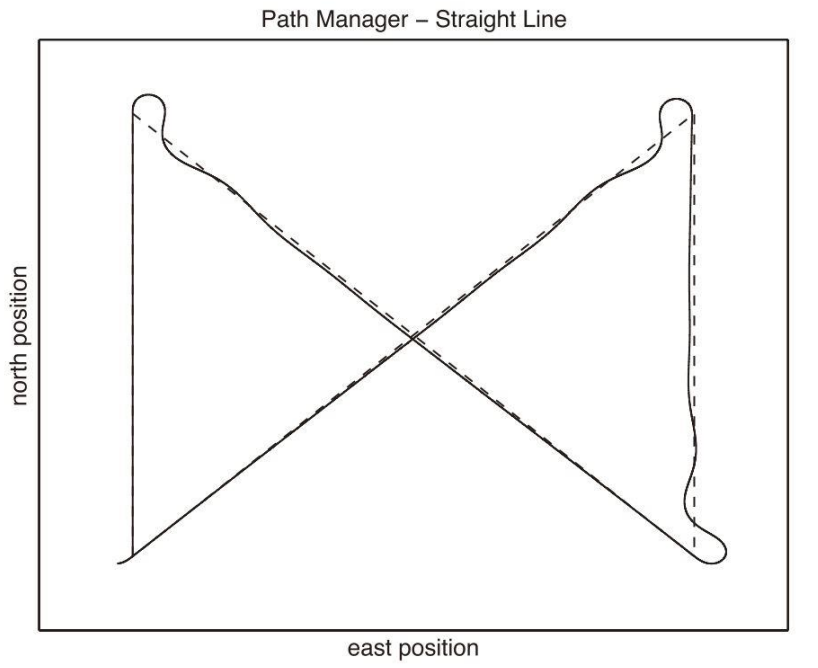

Figura 3-28. Patrón de seguimiento de ruta.

Respecto al seguimiento de la senda de vuelo entre *waypoints*, el FMC se encarga del cálculo continuo del rumbo y de la altitud, cuyos valores son función de la posición del UAV relativa a la ruta, necesarios para alcanzar la posición del siguiente *waypoint*. Por ello, ambas variables se enrutan hacia el autopiloto, donde todos los controladores exceptuando los de velocidad aerodinámica trabajan para llevar a la aeronave al siguiente *waypoint*, definido mediante sus coordenadas geodéticas. Consecuentemente, el sistema de gestión de vuelo implementado cubre tanto la navegación horizontal como la vertical.

Por un lado, la navegación lateral, o lo que es lo mismo: el comando de rumbo, se calcula a partir de un campo vectorial, el cual marca la dirección y el sentido a tomar tal que permita alcanzar el segmento que une ambos *waypoints*. He aquí la representación gráfica:

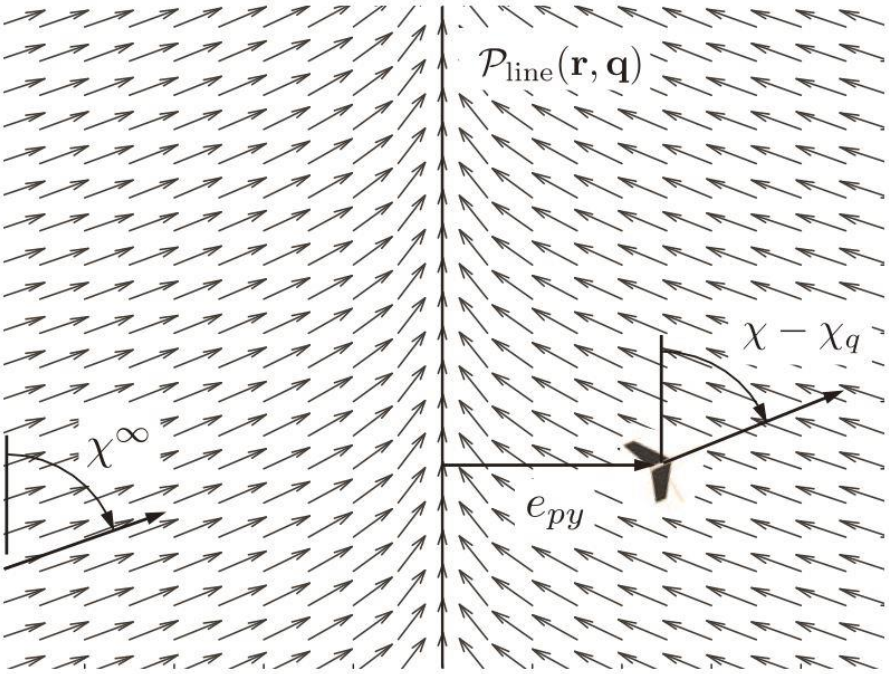

Figura 3-29. Navegación lateral de seguimiento de ruta.

Por otro lado, la navegación vertical, esto es, el comando de altitud, se basa en asignar aquel valor de altitud al cual debería estar volando la aeronave si esta se encontrase en el plano vertical que contiene a dicho segmento de ruta. Este concepto es mostrado gráficamente del siguiente modo:

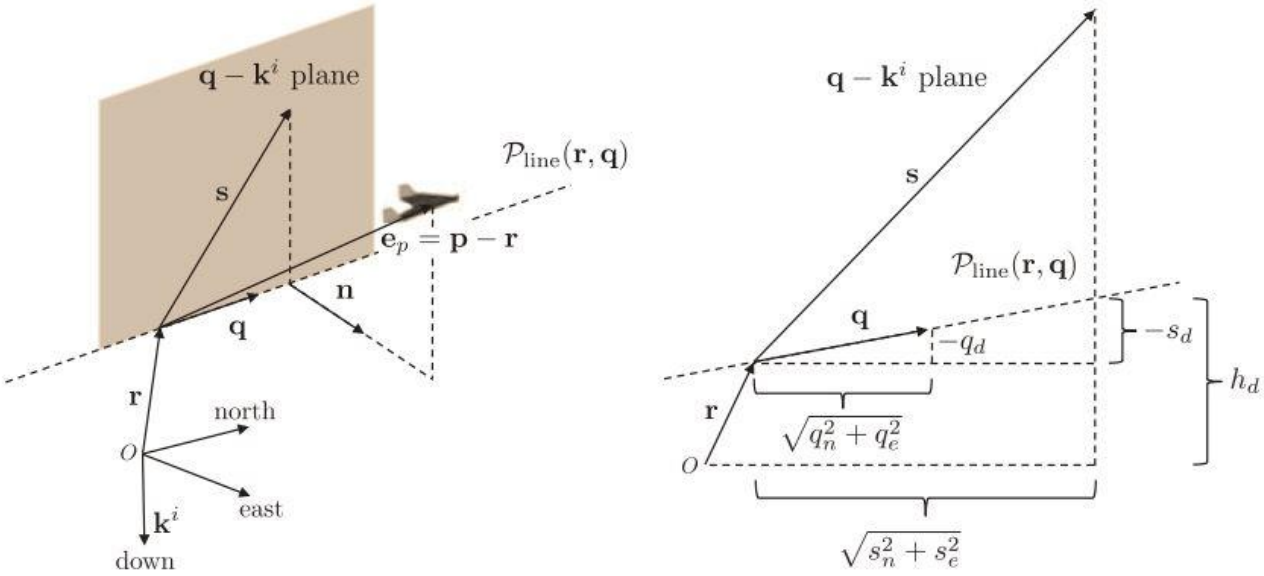

Figura 3-30. Navegación vertical de seguimiento de ruta.

Dado que las ecuaciones se pueden encontrar en [3] así como en el anexo dedicado al código, estas no serán objeto de análisis. Pese a ello, seguidamente se presenta el algoritmo utilizado, encargado de aunar el proceso que permite obtener el rumbo y la altitud a seguir por el vehículo aéreo no tripulado.

Algorithm 3 Straight-line Following: $[h^c, \chi^c] = \text{followStraightLine}(\mathbf{r}, \mathbf{q}, \mathbf{p}, \chi)$

Input: Path definition $\mathbf{r} = (r_n, r_e, r_d)^\top$ and $\mathbf{q} = (q_n, q_e, q_d)^\top$, MAV position $\mathbf{p} = (p_n, p_e, p_d)^\top$, course χ, gains $\chi_\infty, k_{\text{path}}$, sample rate T_s.

Compute commanded altitude using equation (10.5).
$\chi_q \leftarrow \texttt{atan2}(q_e, q_n)$
while $\chi_q - \chi < -\pi$ **do**
  $\chi_q \leftarrow \chi_q + 2\pi$
end while
while $\chi_q - \chi > \pi$ **do**
  $\chi_q \leftarrow \chi_q - 2\pi$
end while
$e_{py} \leftarrow -\sin \chi_q (p_n - r_n) + \cos \chi_q (p_e - r_e)$
Compute commanded course angle using equation (10.8).
return h^c, χ^c

Figura 3-31. Algoritmo de seguimiento de ruta.

Para finalizar, este es el aspecto general del receptor GNSS implementado, cuyos datos son enviados desde el bloque de misión a una segunda instancia de Matlab:

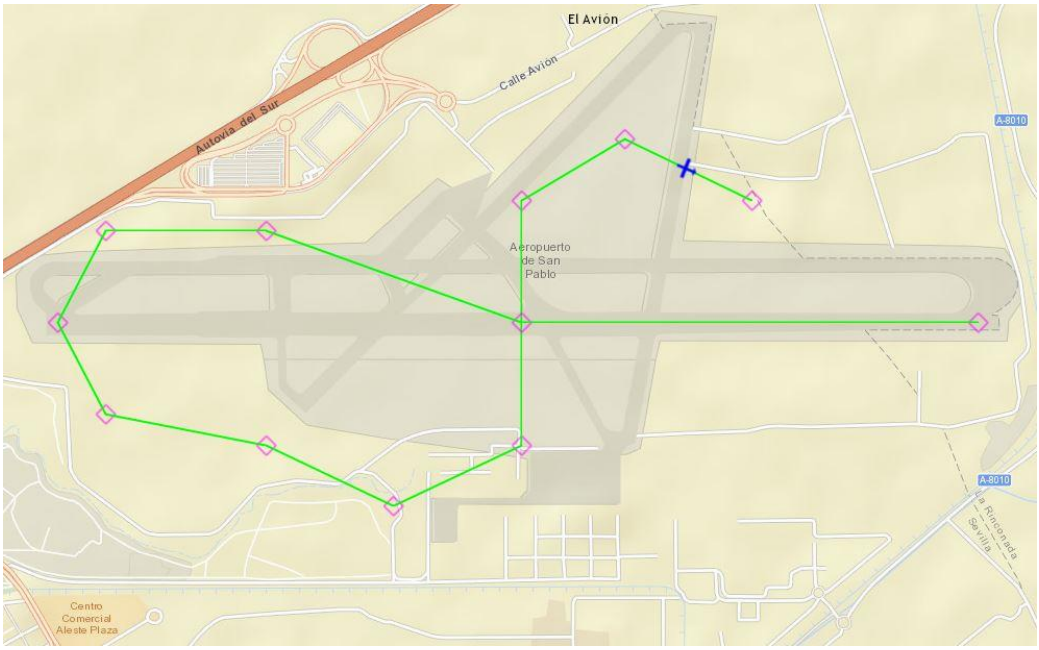

Figura 3-32. Receptor GNSS.

En este ejemplo se muestra la ruta que incluye por defecto el simulador, que no es sino un circuito alrededor del Aeropuerto de San Pablo (código OACI: LEZL). El receptor utiliza como capa base los datos cartográficos del lugar pertenecientes a World Street Map. No obstante, se ha añadido diferente iconografía para la representación de los puntos de paso o *waypoints*, los segmentos de ruta y el Skywalker X8, tal y como se puede apreciar.

La idea tras la implementación del receptor GNSS era la de disponer de un sistema de navegación funcional que permitiera al piloto conocer la localización exacta de la aeronave en todo momento. Por eso, este bloque se inspira en aquellos sistemas de presentación incluidos en aeronaves civiles de pasajeros, como la pantalla de navegación o ND (*Navigation Display*). Pese a ello, es importante recalcar que este no es suficiente para permitir una navegación por instrumentos tal y como sucedería con una aeronave de tipo comercial. En ese caso, sería necesario la integración de sistemas adicionales como el director de vuelo primario o PFD (*Primary Flight Director*) entre otros.

3.1.13 Previsualización

Finalmente, este último bloque tiene como única finalidad la ordenación y el acondicionamiento de diversas señales para su representación por pantalla en la zona habilitada a dicho propósito, la cual se encuentra en la parte inferior derecha de la pestaña global del simulador.

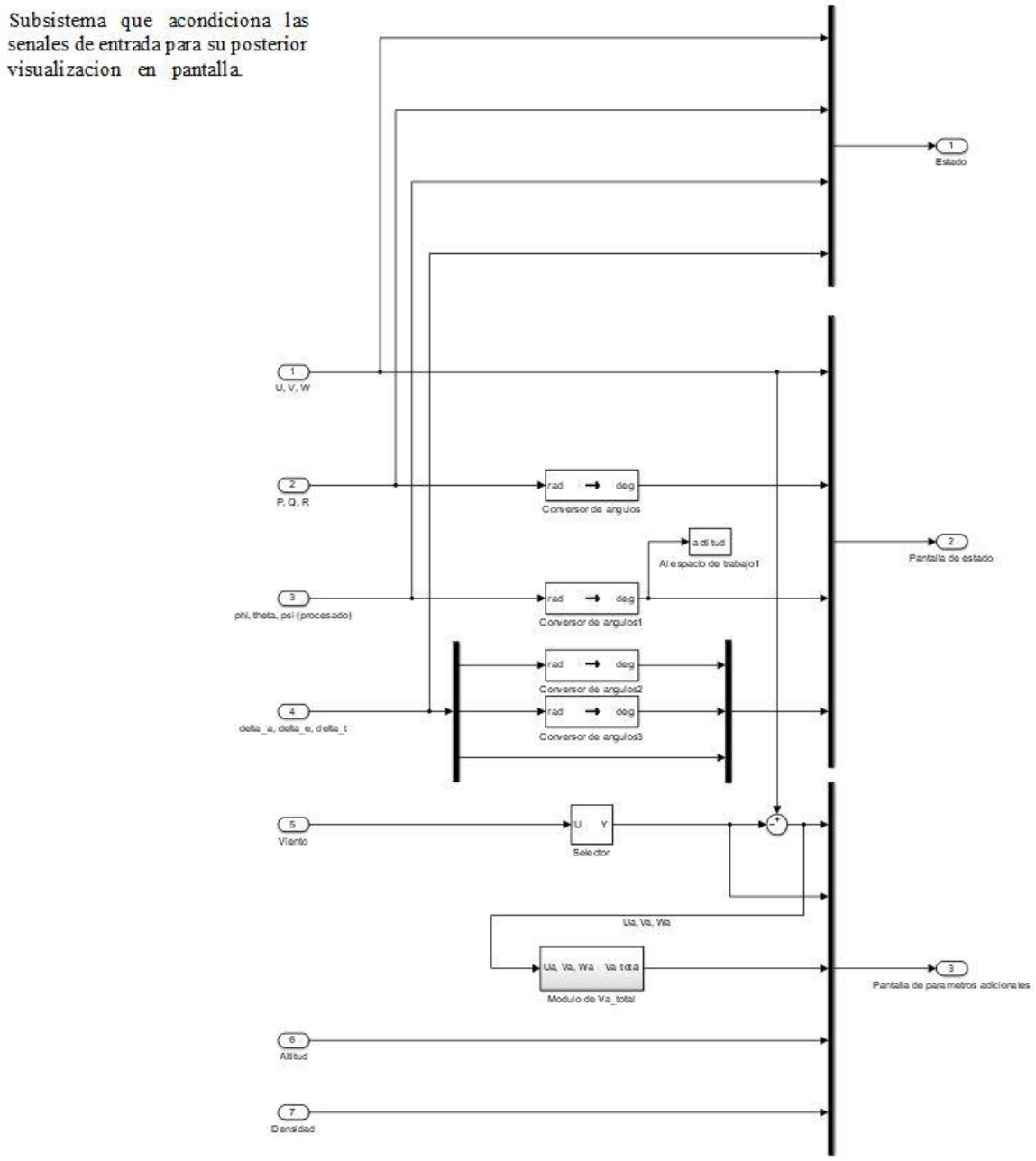

Figura 3-33. Bloque de previsualización.

3.2 Resultados

En esta sección se exponen brevemente los resultados recabados, los cuales nos permiten contrastar la fiabilidad del simulador con la telemetría. En esencia, el proceso llevado a cabo ha consistido en tomar los datos de telemetría de las señales PWM de los actuadores (alerones, elevadores y motor) referentes a un intervalo de tiempo de un determinado vuelo para incluirlos como entrada en el simulador. De esta manera, el resultado a obtener será el estado de la aeronave en todo momento al igual que aquellas variables de interés adicionales que más trascendencia tienen. El simulador original ha sido pues modificado de tal manera que incluya como entrada la dinámica de los actuadores en un espacio de tiempo dado.

Es muy importante mencionar que los efectos de viento utilizados en este proyecto no son más que meras estimaciones provenientes de Mission Planner, que sin lugar a dudas afectará en gran medida a los resultados de simulación. Nótese que el Skywalker X8 no es más que una aeronave de 4 kilogramos de masa, aproximadamente, y de tipo ala volante por lo que es muy susceptible de sufrir los efectos del viento. Pero, desgraciadamente, durante su construcción no se incluyó ningún sensor de viento que permitiera calcular la intensidad y dirección del mismo relativo a la aeronave. Esto se ha convertido en un grave impedimento tanto a la hora de modelar como de simular en este trabajo.

Pese a todos los inconvenientes, se optó por modificar el simulador original en aras de poder contrastar sus resultados con la realidad, reflejada en la telemetría. Los resultados se prevén ser de excasa fiabilidad, pero el simple hecho de haber construido el simulador proporciona una vía para probar futuros modelos que se hayan construido a través de un método más riguroso y en los cuales fuera posible un proceso iterativo de ensayos y afinación del mismo.

Al fin, se muestran algunos resultados obtenidos a la vez que se comparan con los datos de telemetría. La traza azul denota los datos extraídos del simulador mientras que la roja representa los datos de telemetría:

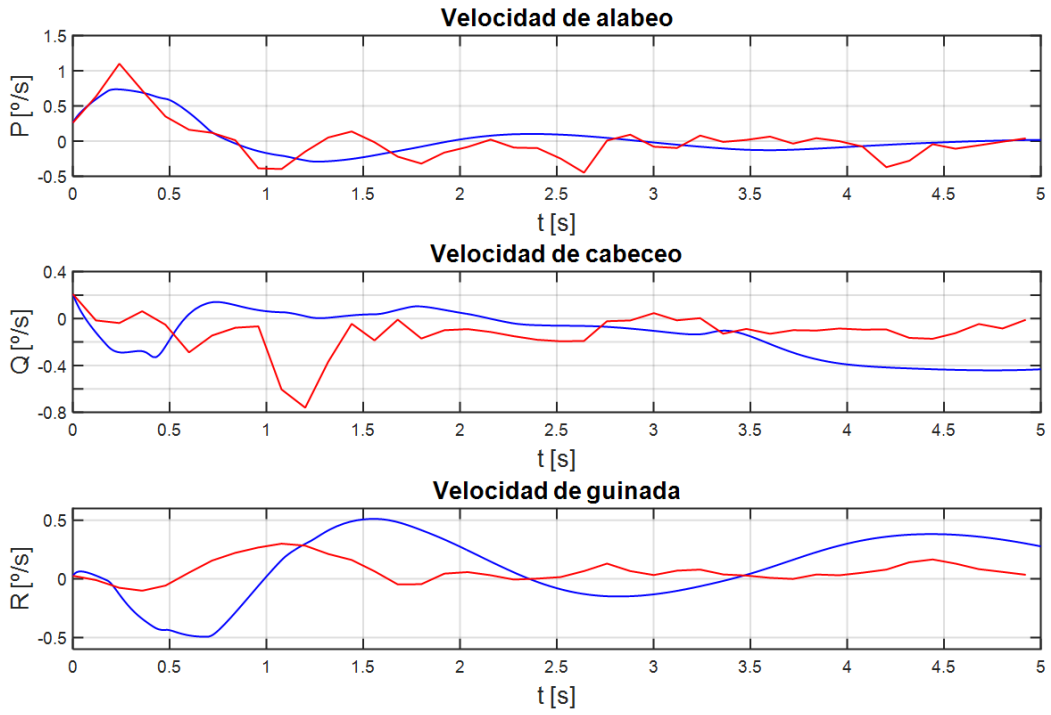

Figura 3-34. Resultados de velocidades angulares.

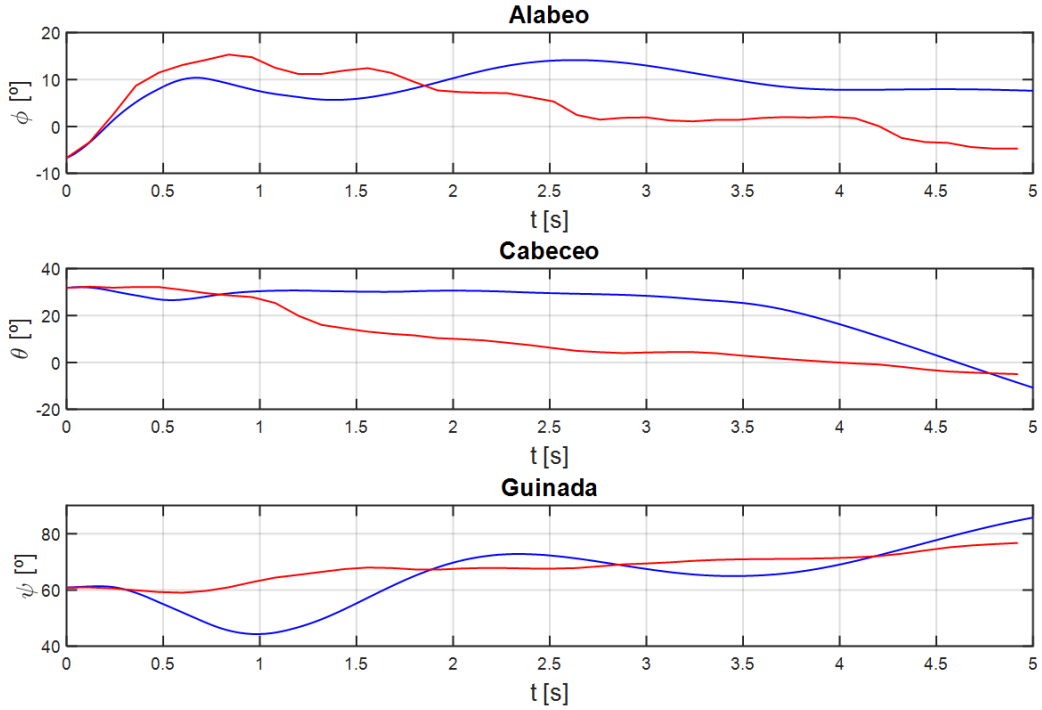

Figura 3-35. Resultados de actitud.

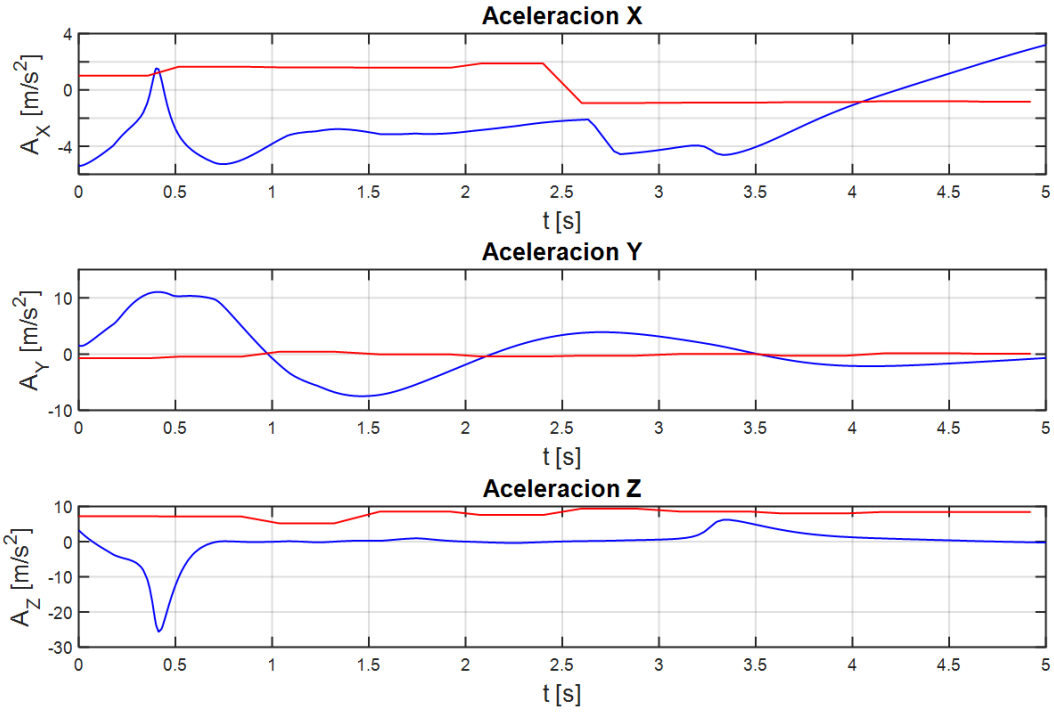

Figura 3-36. Resultados de aceleración lineal.

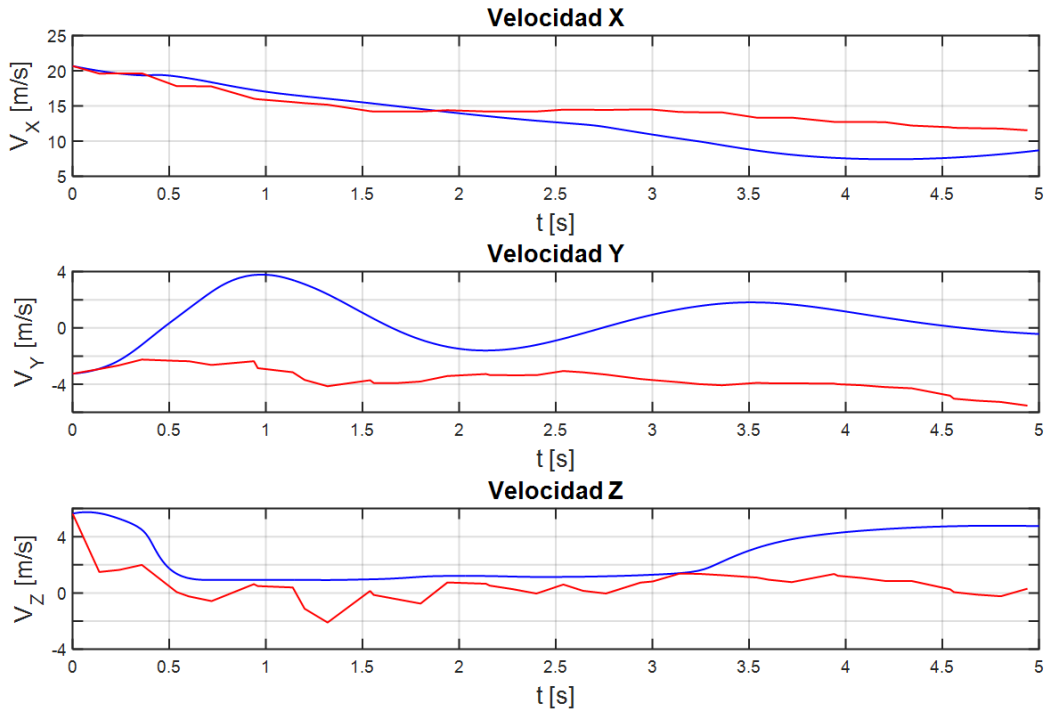

Figura 3-37. Resultados de velocidad lineal.

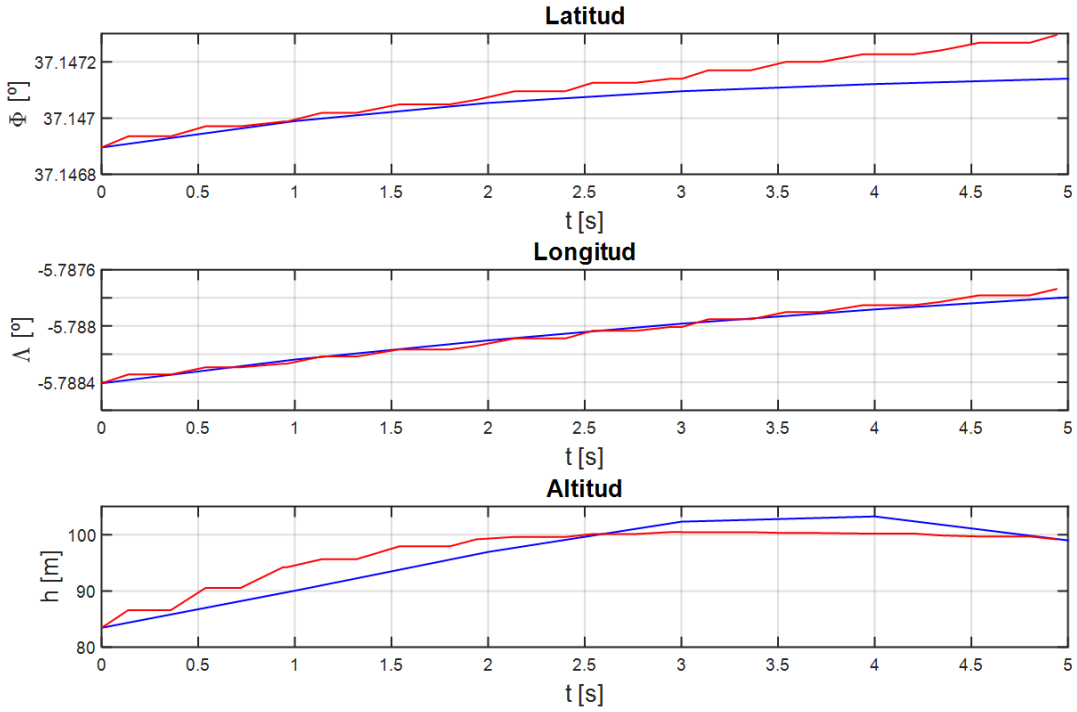

Figura 3-38. Resultados de coordenadas geodéticas.

4 LIMITACIONES Y FUTURAS MEJORAS

Este capítulo final sirve como recordatorio de aquellos factores limitantes que se pueden encontrar a la hora de simular el sistema al mismo tiempo que se menciona una serie de mejoras que podrían ser implementadas en este trabajo o en futuros proyectos relacionados.

4.1 Matrices de cosenos directores

Durante todo el trabajo se han realizado las transformaciones entre sistemas de referencia aplicando el álgebra de las matrices de cosenos directores tal y como se explicó en el segundo capítulo. Pese a que este método es intuitivo desde un punto de vista físico, debido a que estas matrices representan una rotación entre dos sistemas de coordenadas, conlleva un gran inconveniente a la hora de realizar cálculos con ellas. Esta desventaja no es más que la existencia de singularidades (depende de la transformación) para actuaciones determinadas de la aeronave. El hecho de que estas matrices, tal y como indica su nombre genérico, estén construidas a partir de cosenos o funciones trigonométricas en general, imposibilita el poder deshacerse de las singularidades inherentes a las mismas. Dos ejemplos de estas singularidades son las que acontecen cuando la aeronave alcanza un ángulo de cabeceo de $\frac{\pi}{2}$ y $-\frac{\pi}{2}$, donde el coseno de sendos ángulos no tiene solución (véase la matriz de rotación de ejes de navegación a ejes cuerpo presentada en el capítulo de modelado).

Este problema, no obstante, se puede enmendar implementando cuaterniones en vez de matrices de cosenos directores. Esta nueva representación tendría como gran ventaja la eliminación de toda singularidad ya que las funciones trigonométricas características de las DCM serían sustituidas por escalares. Al mismo tiempo, un cuaternión se define mediante cuatro parámetros en contra de los nueve de una matriz de cosenos directores (uno por cada término dentro de la matriz) dando lugar a una simplificación en las operaciones y a un consecuente ahorro de capacidad computacional. Pero no son todo ventajas, el sentido físico proporcionado por las DCM, o incluso por otras representaciones como el ángulo y eje de Euler, se pierde.

Como se mencionó en su momento, el propósito de este trabajo es meramente educativo, por tanto se optó desde el principio en implementar las matrices de cosenos directores. Asimismo, es improbable que un UAV como el seleccionado vuele a valores de ángulo de cabeceo mayores de $\frac{\pi}{4}$ a no ser que entre en un régimen de pérdida aerodinámica. Aún así, es posible experimentar fallos en el simulador por haber alcanzado una de estas singularidades, aunque no ocurre bajo actuaciones típicas de vuelo.

4.2 Rendimiento computacional

El simulador de vuelo requiere importantes recursos a nivel computacional, en otras palabras, de un ordenador capaz de realizar mientras más cálculos por segundo mejor. El hecho de no disponer de un ordenador con una unidad de procesamiento central o CPU (*Central Processing Unit*) de suficiente capacidad de cálculo se traduce en un simulador que no es capaz de generar datos a tiempo real, sino más despacio. Por ese motivo, si se busca una experiencia genuina de simulación es aconsejable disponer de un equipo acorde a las necesidades del programa.

La capacidad de cálculo depende de varios factores, pero uno de los que mayor repercusión tiene es la frecuencia de reloj de la CPU. Como consecuencia, en el caso de que no se pudiera disponer de una que fuera más potente, se aconseja realizar un "overclock" a este componente, aunque la mejora en rendimiento es marginal comparada con la que una CPU reciente pueda proporcionar.

En última instancia, puesto que el simulador utiliza FlightGear como interfaz gráfica, es recomendable que nuestro equipo posea una unidad de procesamiento gráfico o GPU (*Graphics Processing Unit*) acorde a sus requerimientos, los cuales se pueden encontrar en la página web oficial del simulador de vuelo (http://wiki.flightgear.org/Hardware_recommendations). Como en cualquier otro videojuego, mientras más potente sea la tarjeta gráfica más altas serán las opciones gráficas a las que se pueda ejecutar FlightGear de forma fluida.

Finalmente, se mencionan los componentes más importantes junto a algunas de sus características técnicas del equipo utilizado para ejecutar el simulador durante este proyecto:

- CPU: i5 6600
    - Núcleos: 4 (físicos).
    - Frecuencia de reloj base: 3.6 GHz.
    - Frecuencia de reloj turbo: 3.9 GHz.
    - Caché: 6 Mb (SmartCache).
    - Velocidad del bus: 8 GT/s DMI3.
    - Tamaño de memoria: 64 Gb (máximo) DDR4.
- GPU: Sapphire Nitro+ Radeon RX 480 8 Gb
    - Procesadores: 2304.
    - Frecuencia de reloj: 1342 MHz (máxima).
    - Bus de memoria: 256 bits GDDR5.
    - Frecuencia de reloj de memoria: 2 GHz.
    - Tamaño de memoria: 8 Gb.
- Memoria RAM: Avexir DDR4 2133 MHz 2x8 Gb
    - Velocidad: 2133 MHz DDR4.
    - Tamaño de memoria: 16 Gb.
    - Latencias: 15-15-15-35.

4.3 Optimización

Como último punto de este trabajo, es interesante mencionar que tanto el código implementado como la configuración de bloques dentro del simulador o los parámetros de los distintos controladores del autopiloto tienen margen de mejora a nivel de rendimiento, esto es, pueden optimizarse aún más para reducir el costo global de cálculo y, de este modo, aumentar su rapidez.

Dado que en ningún momento el objetivo de este trabajo era construir un código perfectamente optimizado, este proceso se ha llevado a cabo hasta cierto punto tal que se pudiera considerar aceptable para ser ejecutado en la mayoría de máquinas sin grandes impedimentos y a velocidades razonables.

Durante el proyecto, se probó a utilizar la caja de herramientas de computación paralela (*Parallel Computing Toolbox*) incluida en Matlab de tal forma que se amortizara el hecho de disponer de más de un núcleo de procesamiento operativo en la CPU, pues Matlab por defecto asigna un único núcleo por cada instancia abierta del programa. Sin embargo, y pese a resultar exitoso a nivel de código, se descartó la idea debido a que Simulink no permitía la paralelización de las tareas, siendo la simulación el proceso verdaderamente exigente a nivel computacional. Este no quiere decir que el código no se pueda adaptar a la configuración particular de un ordenador de tal forma que los procesos se distribuyan y ejecuten simultáneamente en los distintos núcleos de la CPU, sino que no resultaba especialmente rentable en este caso concreto.

Finalmente, los parámetros que definen los controladores tanto de dinámica lateral como longitudinal incluidos en el autopiloto se han obtenido mediante un procedimiento totalmente manual. Primero, se seleccionaba la ganancia del controlador proporcional hasta que la salida del lazo oscilase; luego, la del controlador integral hasta que el proceso se estabilizase en el tiempo requerido; y, por último, la del controlador derivativo si era necesario estabilizarlo en régimen transitorio. Por esta razón, sería interesante llegar a unos valores óptimos de dichos parámetros aplicando un método de afinación analítico (por ejemplo, el método de Ziegler-Nichols) y poder así comparar su dinámica con la aplicada en este trabajo.

ANEXO I. FLIGHTGEAR

Como se ha mencionado más de una vez a lo largo de este documento, el simulador hace uso de una interfaz gráfica que permite al usuario visualizar el comportamiento del UAV en tiempo real a partir de los resultados que se van extrayendo del mismo. Esta interfaz gráfica no es otra que el simulador de vuelo de código abierto FlightGear.

Este anexo está destinado a servir de guía de instalación para todos los archivos de FlightGear necesarios: primero, el videojuego en sí (FlightGear); luego, el modelo gráfico de la aeronave utilizada (Malolo1); y finalmente, el escenario (Sevilla).

El primer paso debe ser pues descargar los archivos. Debido a que en este proyecto se ha utilizado Matlab R2015a, la versión más reciente capaz de comunicarse con Simulink es la 3.0; no obstante, las actualizaciones de Matlab de los años sucesivos soportan versiones más recientes de FlightGear. Los archivos de instalación del simulador se pueden encontrar en la sección de descargas de su web oficial (http://www.flightgear.org/download/), dentro de la cual existe un acceso para la descarga de versiones más antiguas. En cuanto al modelo de la aeronave, de tipo ala volante, se denomina "Malolo1" y se puede bajar del siguiente enlace: http://ftp.igh.cnrs.fr/pub/flightgear/ftp/Aircraft-2.0.0/Malolo1_0.0.zip. Por último, los archivos de escenario se pueden obtener de http://ns334561.ip-5-196-65.eu/~fgscenery/WS2.0/scenery-2.0.1.html. Aquí habrá que seleccionar la cuadrícula de la malla que se desee descargar (en este caso, *"w010n30"*) en función de la zona geográfica donde va a transcurrir la simulación.

Los pasos de instalación de FlightGear aparecen al abrir el archivo ejecutable *"Setup FlightGear 3.0.0.exe"*, siendo estos los típicos de cualquier otro programa: directorio de instalación, características adicionales, software externo requerido, etc. Es recomendable asegurarse de que una vez ejecutado el archivo aparezca la ventana emergente para dotar al instalador con los permisos de administrador. Si este no fuera el caso se deben conceder dichos privilegios manualmente.

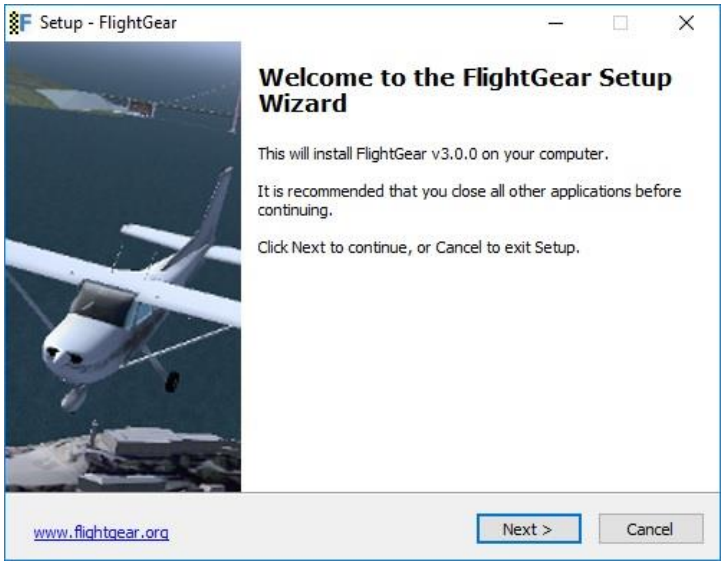

Figura I-1. Ventana inicial del instalador de FlightGear.

Posteriormente, los datos del modelo gráfico de la aeronave, contenidos en el archivo comprimido *"Malolo1_0.0.zip"*, deben ser descomprimidos y pegados en la carpeta que contiene los datos de las distintas aeronaves incluidas en el simulador. Esta carpeta se encuentra en el directorio base de FlightGear y por defecto es *"C:\Program Files\FlightGear\data\Aircraft"*.

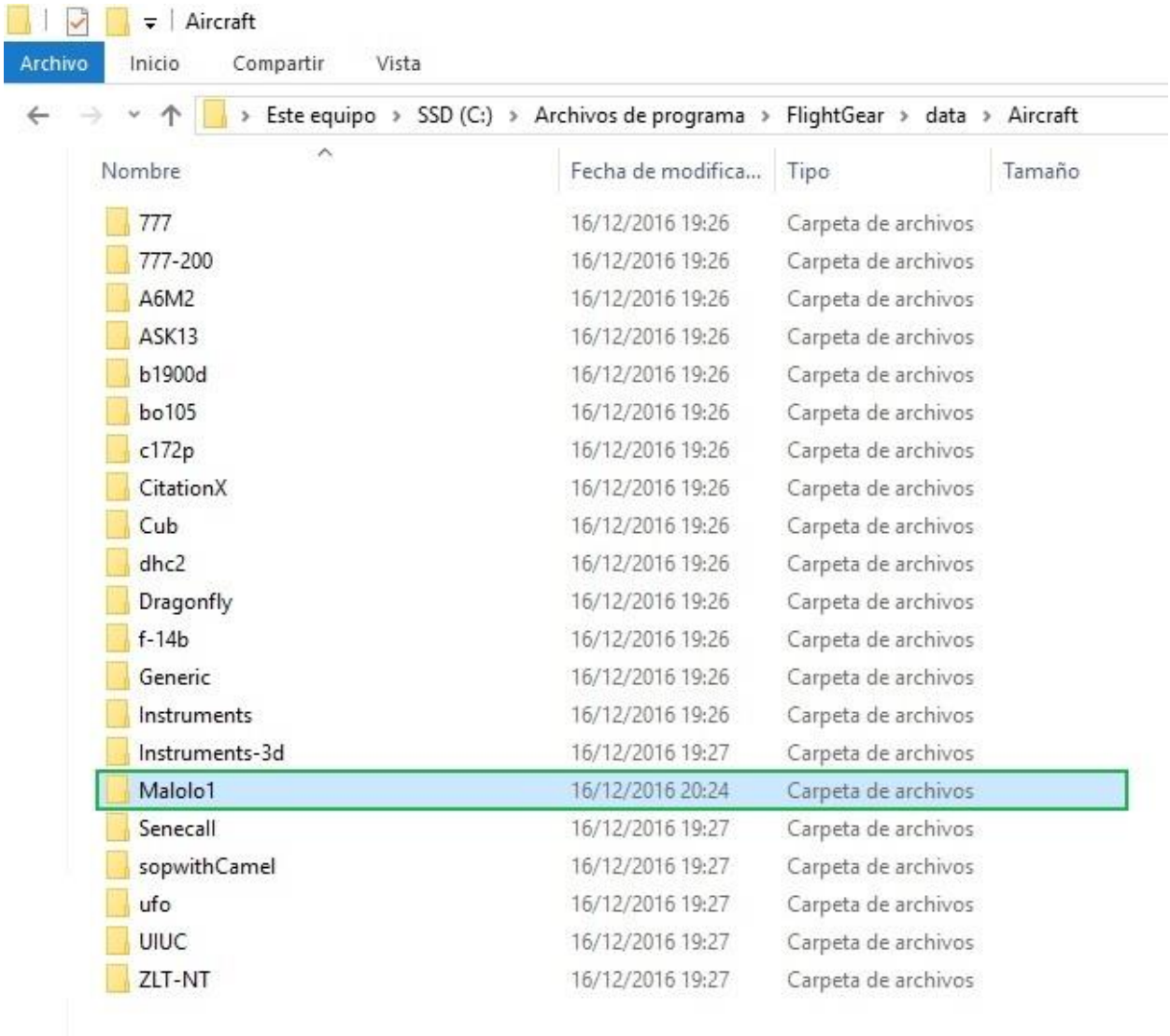

Figura I-2. Instalación del modelo gráfico de la aeronave.

Para terminar, se han de instalar los archivos de escenario. Esto se logra descomprimiendo *"w010n30.tar.gz"* y a su vez *"w010n30.tar"*. La carpeta restante, debe ser copiada y pegada *"C:\Users\(Nombre de usuario)\Documentos\FlightGear\TerraSync"*, más concretamente dentro de las carpetas *"Airports"*, *"Objects"* y *"Terrain"* respectivamente. Aunque este procedimiento debería ser suficiente, si por cualquier motivo el escenario no se cargara al abrir la GUI (evidente cuando el escenario es puramente océano), se sugiere pegar la carpeta *"w010n30"* en dentro de *"C:\Program Files\FlightGear\scenery"* y en las tres carpetas anteriormente mencionadas contenidas en *"C:\Program Files\FlightGear\data\Scenery"*.

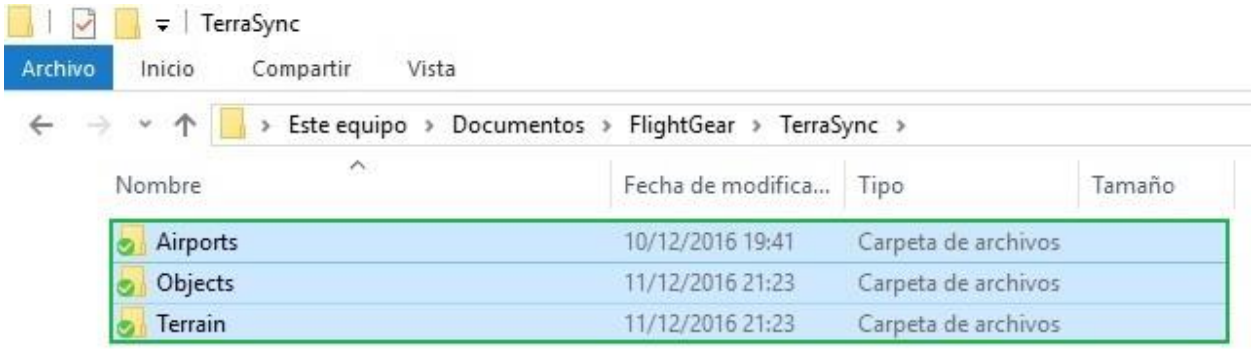

Figura I-3. Instalación del escenario.

Si bien en este anexo se ha presentado de forma concisa e ilustrativa los pasos de instalación, adicionalmente se adjunta un tutorial de instalación del programa, así como de aeronaves y escenarios, disponible en la siguiente página web: http://www.flightgear.org/Docs/getstart/getstartch2.html#x6-130002, donde trata esta disyuntiva detalladamente.

Al final, el aspecto de la interfaz gráfica previo a iniciar la simulación debe ser el siguiente:

Figura I-4. Aspecto de la interfaz gráfica del simulador en el instante inicial.

ANEXO II. SIMULADOR

Ahora se procede a presentar una guía de uso del simulador implementado en Simulink al mismo tiempo que se enseña a utilizar los *scripts* y funciones de Matlab relacionados con el mismo. En primer lugar, se empezará por la puesta a punto, dependiente del ordenador que se utilice y la red a la cual esté conectado. Todo ello bajo un sistema operativo Windows, concretamente Windows 10.

El simulador se ha distribuido en tres archivos distintos en función de las necesidades. El primero (*"Simulador.slx"*) contiene al simulador más completo y debe ser usado siempre y cuando se tenga un *joystick* conectado al ordenador; de lo contrario, saltará un error al ejecutarlo. Tiene implementada la lógica que permite manejar el Skywalker X8 mediante la palanca de control o el teclado. El segundo simulador (*"Simulador_Teclado.slx"*) se debe utilizar únicamente cuando no se dispone de un *joystick*, siendo la única diferencia respecto al anterior el hecho de no disponer de entrada para este periférico. De esta manera, se consigue que el error previamente comentado no aparezca. Por último, existe un tercer simulador (*"Simulador_Telemetria.slx"*) cuya función es la de comparar los resultados del modelo con los de la telemetría a partir de los datos de los actuadores y viento correspondientes a un intervalo determinado de tiempo de uno de los vuelos. Para ello, habrá que ejecutar con anterioridad en la ventana de comandos de Matlab el *script "carga_telemetria.m"*. A diferencia de los dos primeros, este último simulador es totalmente automático, por lo que no es posible manejar la actuación de la aeronave.

Para utilizar el simulador, lo primero es configurarlo. Por ello, se debe averiguar cual es la dirección IP del ordenador dentro de la red local así como su puerta de enlace predeterminada en caso de que sea necesario abrir puertos. La manera más fácil y rápida de lograrlo es mediante el atajo "Win+R", lo que dará lugar a la apertura de una ventana emergente titulada "Ejecutar", aunque alternativamente se puede clicar con el botón derecho del ratón sobre el icono de inicio de Windows y luego clicar en "Ejecutar", o directamente buscar "Ejecutar" en el buscador integrado del sistema operativo. Después, se debe abrir el intérprete de comandos o símbolo del sistema (*Command Prompt*) de Windows escribiendo "cmd" en el hueco que aparece.

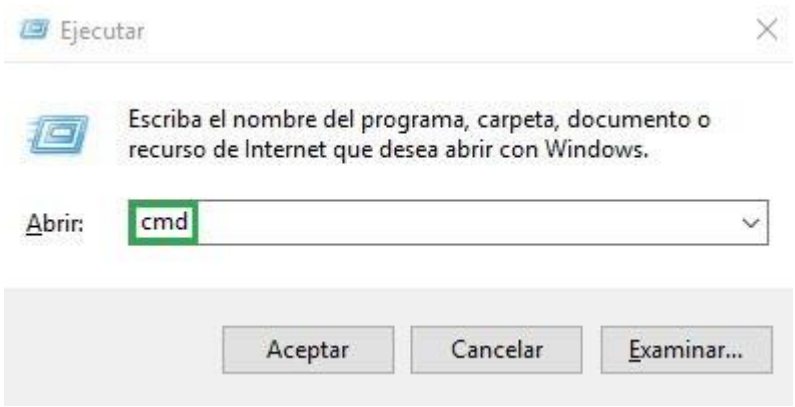

Figura II-1. Ventana emergente de "Ejecutar".

Una vez se haya abierto el intérprete de comandos, se procederá a ejecutar el comando "ipconfig" que permitirá visualizar la configuración básica de red del equipo, inclusive la dirección IP del sistema y la puerta de enlace predeterminada.

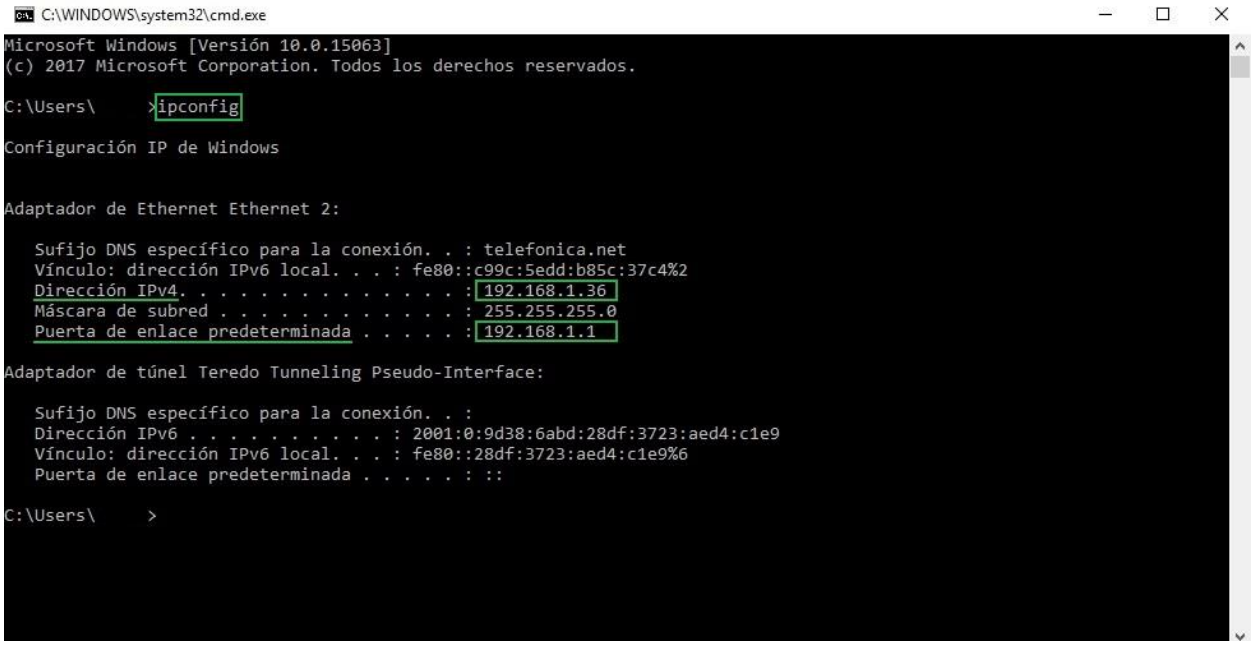

Figura II-2. Intérprete de comandos del sistema.

La dirección IPv4 es necesaria en dos bloques dentro del simulador: "Animacion en FlightGear" y "Generar Script para FlightGear". El primero se encarga de enviar la información de posición y actitud de la aeronave a FlightGear, de tal forma que se pueda observar su comportamiento en cada instante de simulación; el segundo, genera el *script* en formato *.bat* (*Batch*) que ejecuta la interfaz gráfica del simulador.

Ahora, se procede a configurar el bloque de animación de la interfaz gráfica. Lo primero es seleccionar la versión de FlightGear que se esté utilizando (en el caso de Matlab R2015a la más reciente es la versión 3.0). Luego, hay que introducir la dirección IP obtenida en el hueco titulado *"Destination IP address"* del primer bloque. Por último, se apuntará el puerto de destino (*"Destination port"*) a través del cual se mandan los datos a FlightGear en caso de que sea necesario abrirlo posteriormente. Se recomienda no modificar el tiempo de muestreo ("*Sample Time"*).

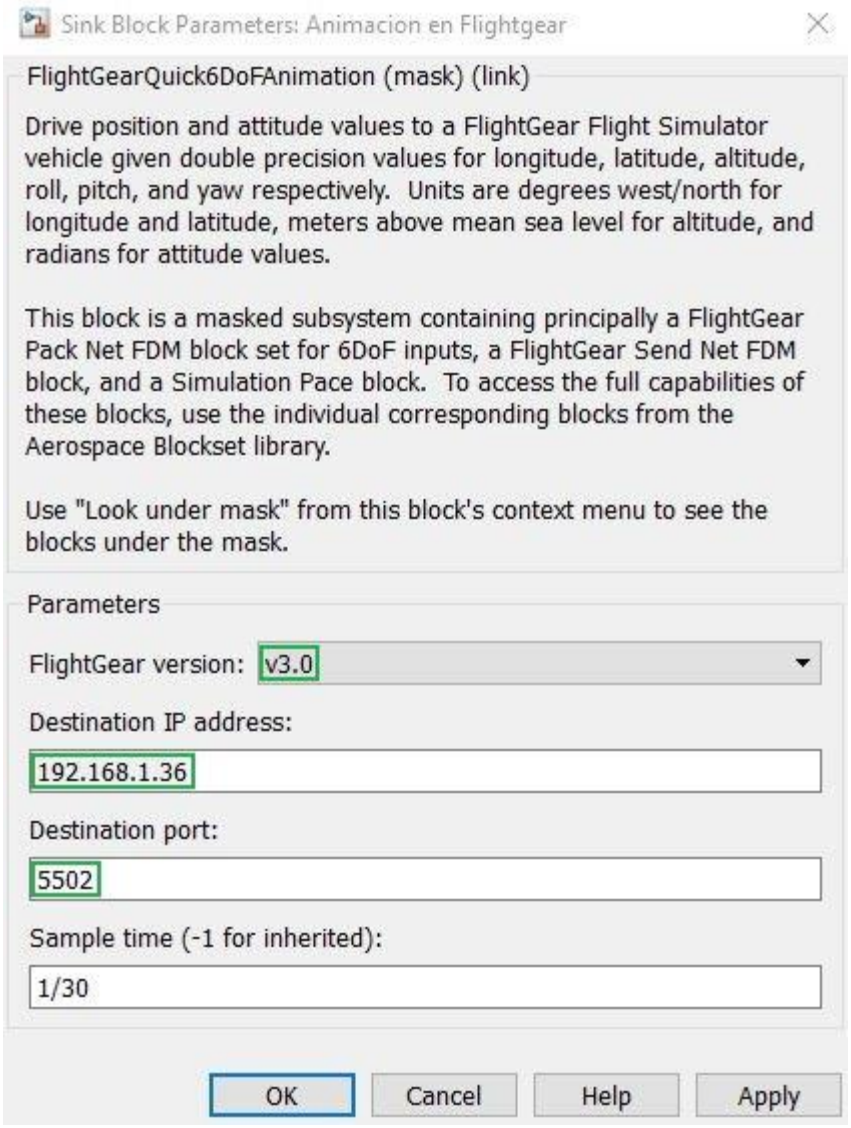

Figura II-3. Ventana de configuración del bloque de animación en FlightGear.

Análogamente, se procede a configurar el segundo bloque mencionado. En la pestaña *FlightGear*, la primera opción a configurar son los parámetros del bloque, donde se debe seleccionar la arquitectura del sistema operativo que se esté utilizando (32 bits o 64 bits) además de asignar el flujo de datos en modo de recepción (*"Receive"*), pues FlightGear sólamente va a recibir datos del simulador y no enviarlos. Después se ha de insertar "Malolo1" en el hueco del modelo de aeronave (*"FlightGear geometry model name"*) y configurar los valores iniciales del simulador en la interfaz gráfica como: el identificador del aeropuerto (*"Airport ID"*) y de la pista (*"Runway ID"*), la altitud (*"Initial altitude (ft)"*) y el rumbo iniciales (*"Initial heading (deg)"*), en pies y en grados respectivamente, y la compensación en distancia (*"Offset distance (miles)"*) y azimut (*"Offset azimuth (deg)"*), en millas y en grados.

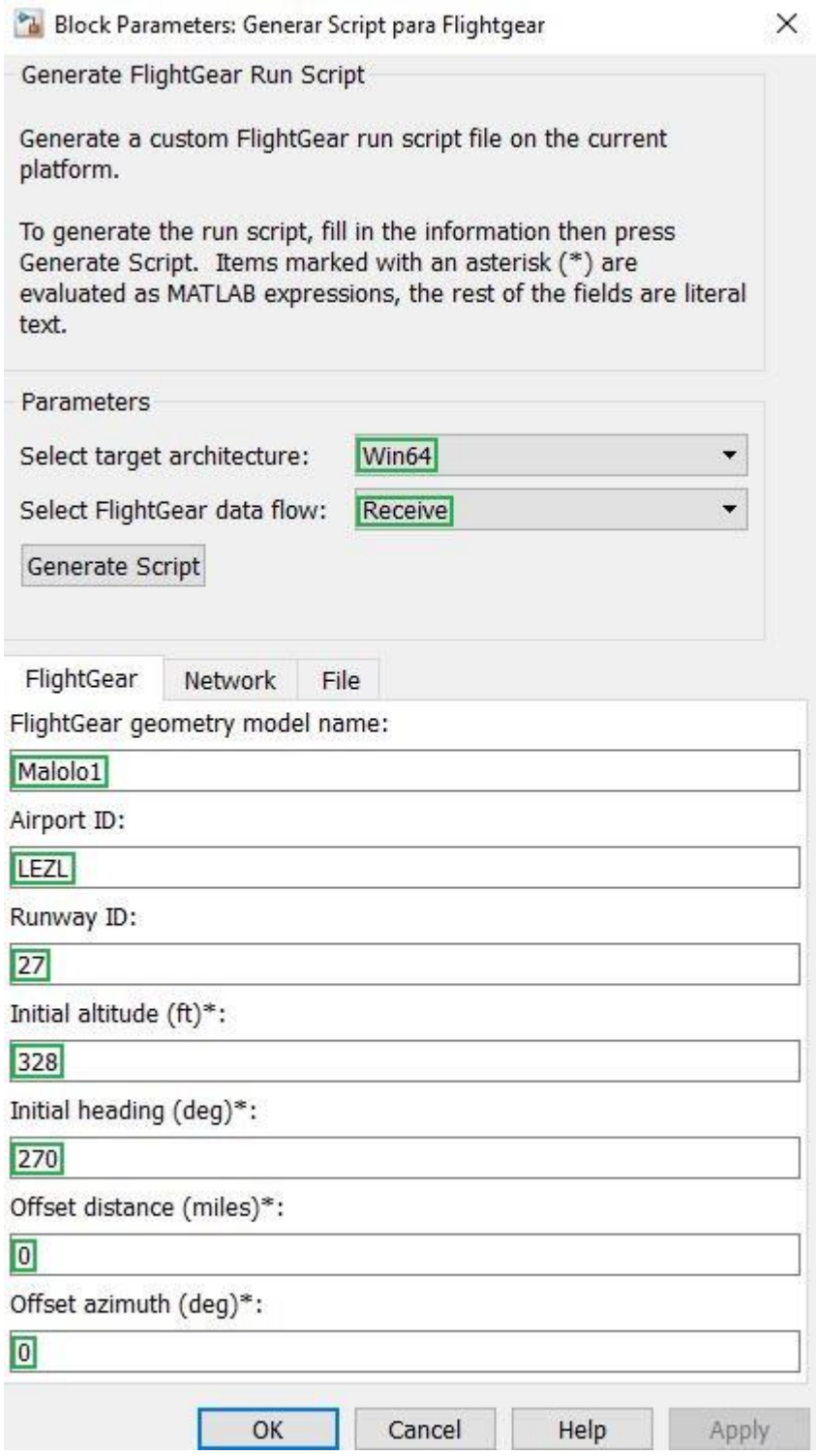

Figura II-4. Ventana de configuración del bloque de generación de scripts de FlightGear (I).

Seguidamente, en la pestaña de red (*Network*), se debe introducir la dirección IP de la máquina. También, es importante apuntar el puerto de origen (*"Origin port"*), pese a ser modificable, en caso de que se tengan que abrir los puertos.

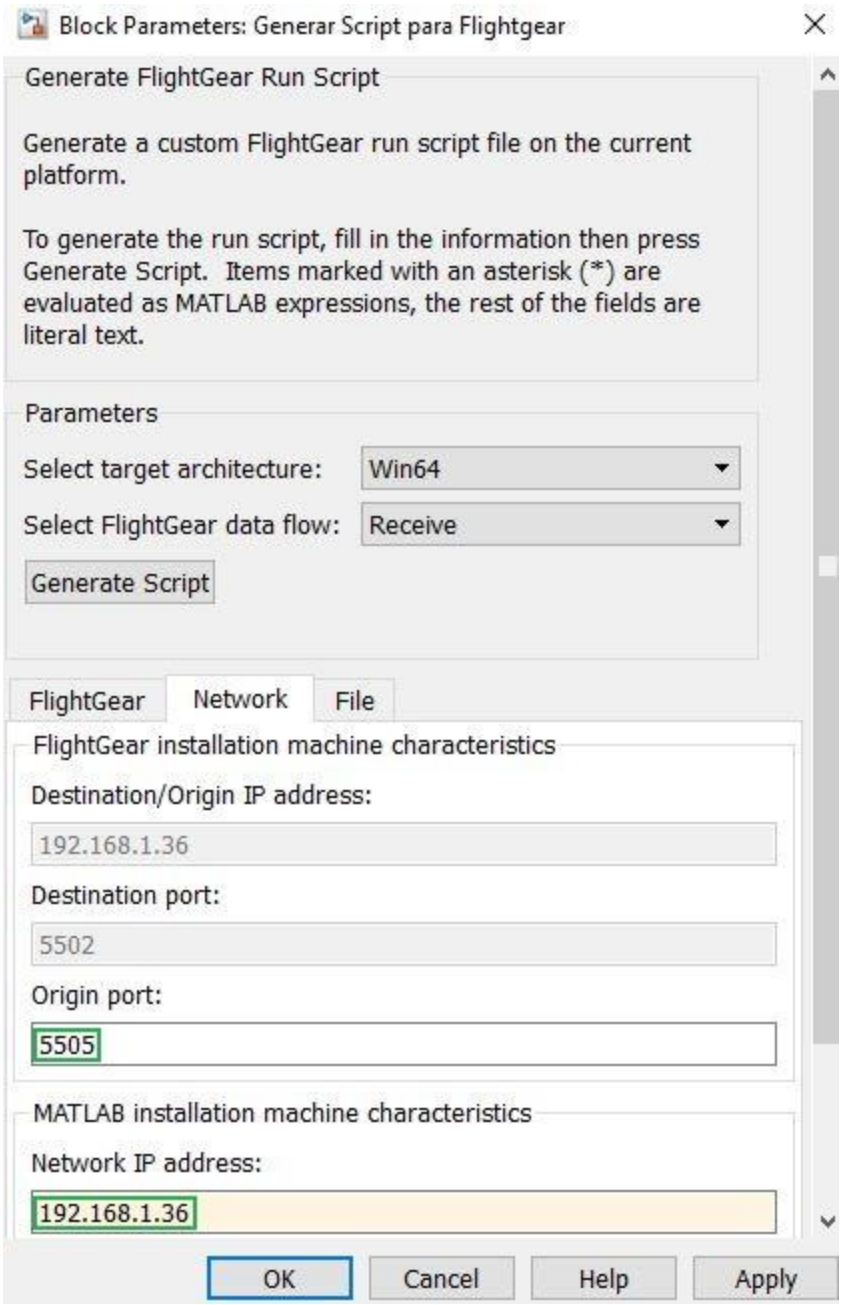

Figura II-5. Ventana de configuración del bloque de generación de scripts de FlightGear (II).

Para terminar con la configuración de este bloque, en la última pestaña (*File*), se debe elegir un nombre de archivo (*"Output file name"*) seguido de la extensión *.bat* y marcar el directorio base donde está instalado FlightGear (*"FlightGear base directory"*). Finalmente, y tras aplicar los cambios, se clicará sobre "Generate Script" para generarlo. Este archivo, encargado de conectar Simulink con FlightGear, se ha de ejecutar siempre antes de iniciar una simulación.

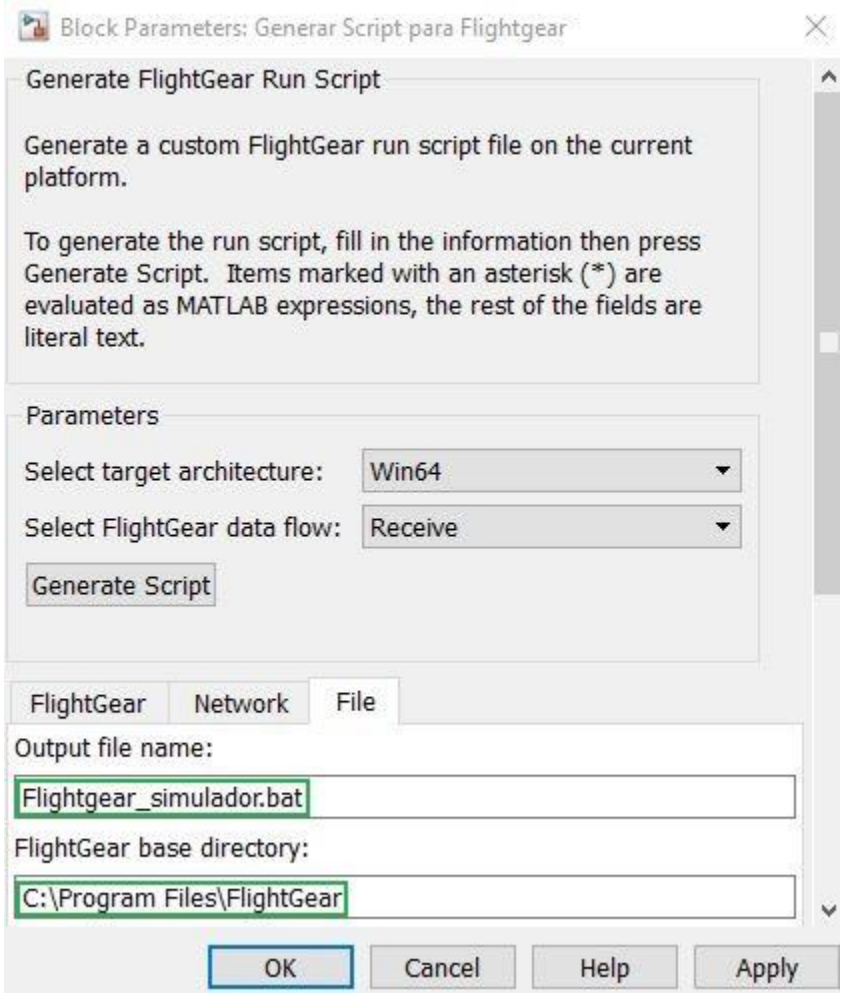

Figura II-6. Ventana de configuración del bloque de generación de scripts de FlightGear (III).

Posteriormente, es necesario asegurarse que la ID del *joystick* asignada por el ordenador a este periférico coincida con aquella del sistema de entrada de la palanca de mando, dentro del bloque de entrada de control manual.

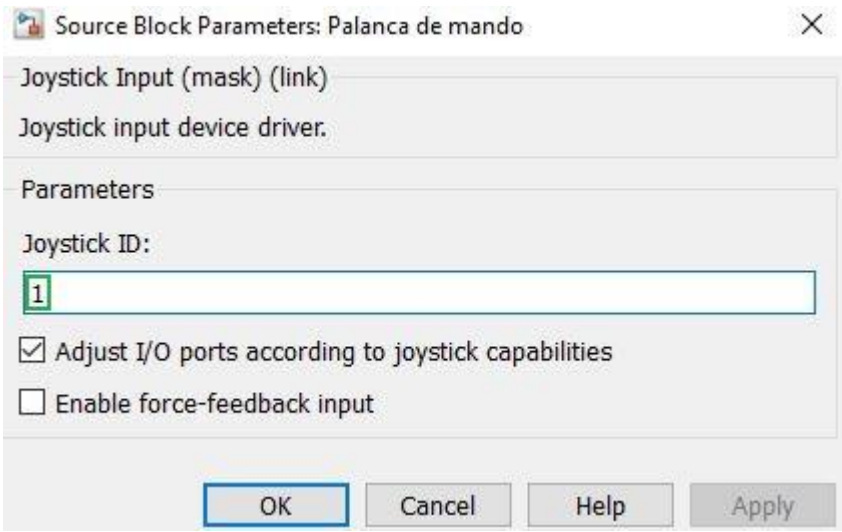

Figura II-7. Ventana de configuración del bloque de palanca de mando.

Las coordenadas geodéticas iniciales, o lo que es lo mismo, la posición de partida de la aeronave se configura como un vector fila en su respectivo bloque de entrada. Es muy importante no olvidar configurarlo.

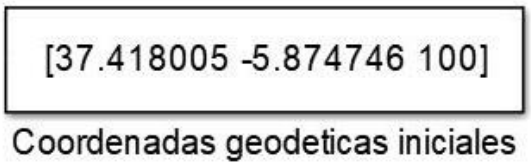

Figura II-8. Bloque de coordenadas geodéticas iniciales.

Por último, reseñar que el simulador tiene implementada una ruta predeterminada, la cual se puede modificar parcial o totalmente en el *script "plan_de_vuelo.m"*. Sólo se tienen que modificar las ternas que definen cada punto de paso (latitud, longitud y altitud) que conforma la trayectoria a seguir, pudiendo eliminar unos y añadir otros. La ruta debe estar formada al menos por un segmento de vuelo, es decir, dos *waypoints*, debiendo estar distribuidos de tal forma que segmentos sucesivos no formen ángulos mayores de 90 grados entre sí. Así se garantiza que el seguimiento de la ruta sea suficientemente suave.

Alcanzado este punto, el simulador debería estar completamente configurado. El primer paso para su ejecución es abrir FlightGear mediante el archivo *.bat* creado. Si por cualquier caso no se abriera, se sugiere dotarlo de permisos de administrador clicando con el botón derecho del ratón sobre el archivo y seleccionando "Ejecutar como administrador". La ejecución de este archivo dará lugar a que aparezca una nueva ventana con la interfaz gráfica del simulador, donde se puede visualizar el modelo de la aeronave, tal y como se ha mostrado en el anexo anterior.

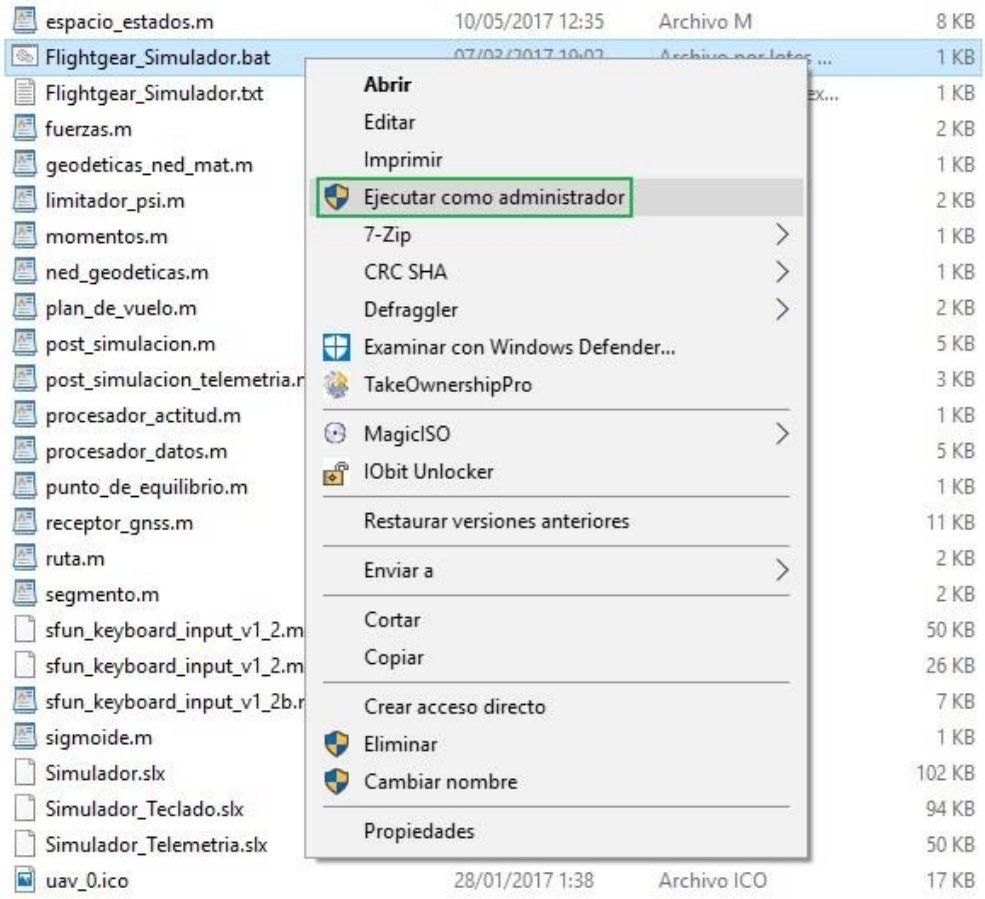

Figura II-9. Ejecución de la interfaz gráfica del simulador.

Seguidamente, se debe abrir una segunda instancia de Matlab, donde se ejecutará el *script "receptor_gnss.m"*. Esta segunda instancia abierta tendrá como única finalidad la recepción y representación de datos provenientes de la otra instancia de Matlab, destinada a correr el simulador. Aparte, se recomienda que, una vez terminada la simulación, y debido a que esta segunda instancia seguirá activa, parar este script pulsando "Ctrl+C" y ejecutar en la ventana de comandos de Matlab *"delete(GNSS)"* para borrar el objeto UDP que permite la conexión entre ambas instancias. Este objeto utiliza por defecto los puertos UDP 3333 y 3334, debiéndose tener en cuenta por si fuera necesaria su apertura.

Llegados a este punto no habría más que iniciar la simulación y comprobar si existe respuesta a variaciones en la entrada manual. Si es así, el simulador funciona correctamente.

Respecto a su manejo, existen cuatro interruptores a nivel de usuario que permiten controlar distintas funcionalidades dentro del simulador: *"Palanca/Teclado"*, seleccionador de tipo de entrada manual; *"FMS"*, interruptor del seguimiento automático de ruta; *"GNSS"*, interruptor del sistema de posicionamiento global (siempre y cuando *"receptor_gnss.m"* esté ejecutándose en una segunda instancia); y *"Viento"*, interruptor del modelo de viento.

Figura II-10. Interruptores del simulador.

Además, se ha incluido un pequeño panel de control del autopiloto que permite asignar la velocidad aerodinámica, el rumbo y la altitud de referencia mediante tres ruedas cuya posición se puede modificar manualmente. La idea era simular una unidad de control de vuelo o FCU (*Flight Control Unit*) como aquellas integradas en las cabinas de los aviones comerciales.

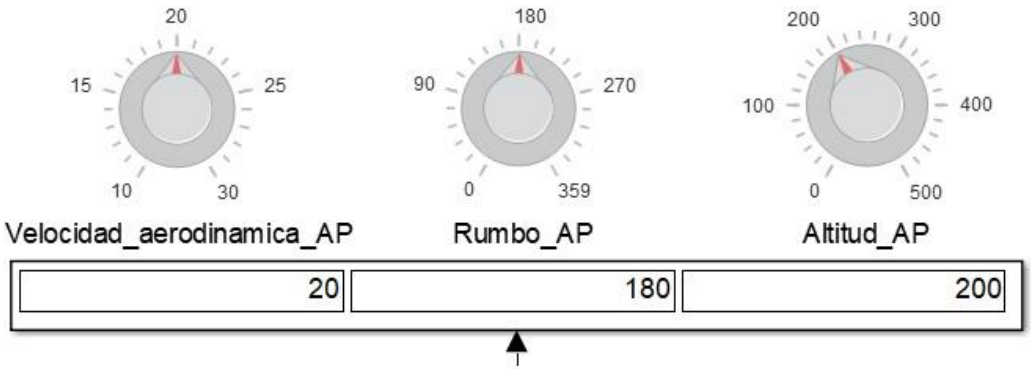

Figura II-11. Unidad de control de vuelo.

Ha de subrayarse que los controladores se deben activar con el fin de seguir las referencias marcadas. Por un lado, cuando la entrada manual está en la posición de palanca de mando se utilizan los botones del 6 al 11 como activadores respectivos de los seis controladores implementados y el botón 1 como reseteador de los mismos. Esta disposición se debe al hecho de haber utilizado un *Logitech Attack 3* como *joystick* durante todo el proyecto. Por el otro lado, si la entrada manual se encuentra en la posición de teclado, el estado de los controladores se controla con las teclas del 1 al 6, mientras que ‘ (la tecla a la derecha del 0) ejecuta su reinicio.

En cuanto a la configuración de la entrada manual, la palanca de control utiliza sus tres ejes para el control de los actuadores (alerones, elevadores y motor), mientras que el teclado hace uso de las teclas A y D para el control de aerodinámica lateral, W y S para la aerodinámica longitudinal y R y F en el caso del motor.

Finalmente, si por cualquier motivo la interfaz gráfica no fuera capaz de reflejar la actuación del Skywalker X8, se recomienda revisar el procedimiento completo. Igualmente, se deberían abrir los puertos mencionados y añadir Matlab y FlightGear como excepción en el cortafuegos o *firewall* de Windows.

Para abrir los puertos, se debe ingresar la puerta de enlace predeterminada, que habitualmente es 192.168.1.1, en la barra de direcciones del navegador. Una vez se ha accedido a esta dirección, aparecerá la página web de acceso a la configuración del *router*. Para acceder, se deberá conocer la clave de acceso al mismo que suele aparecer en una etiqueta adjunta físicamente a este dispositivo. No obstante, la contraseña predeterminada de cada modelo suele encontrarse tras una búsqueda rápida en internet.

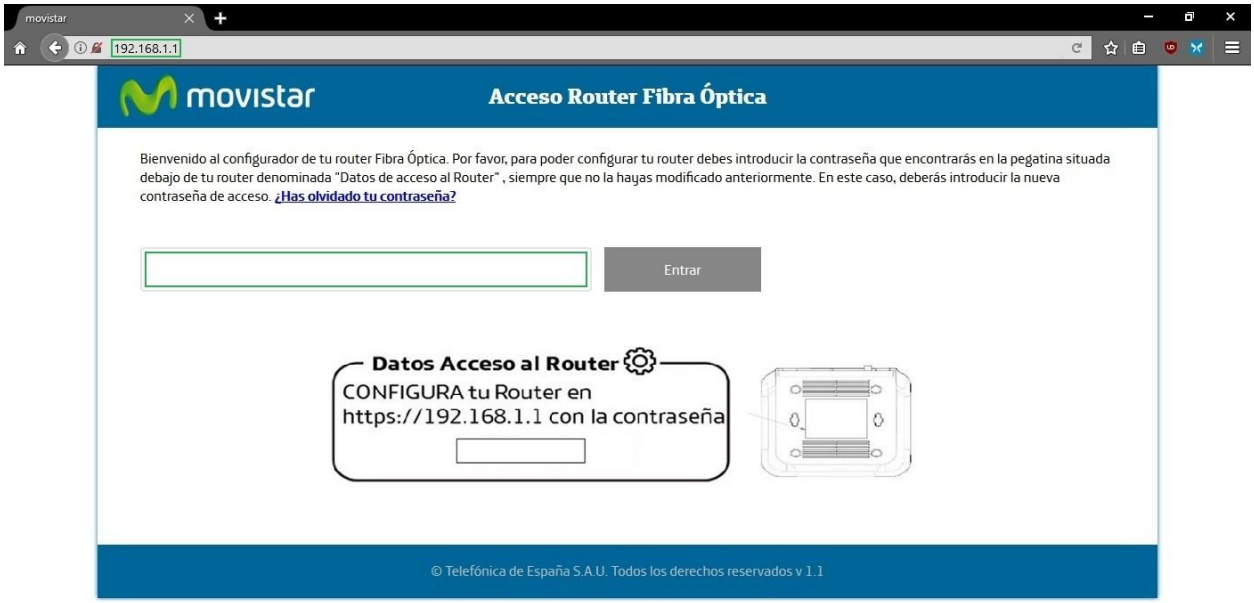

Figura II-12. Página web de acceso al router.

Ya dentro del panel de configuración del *router*, no es difícil encontrar una opción que permita la edición de puertos de entrada y de salida. Llegado a este punto, simplemente se ha de seleccionar la dirección IP local de la máquina que se esté usando y los puertos a abrir, mediante protocolo TCP o UDP (se recomienda ambos).

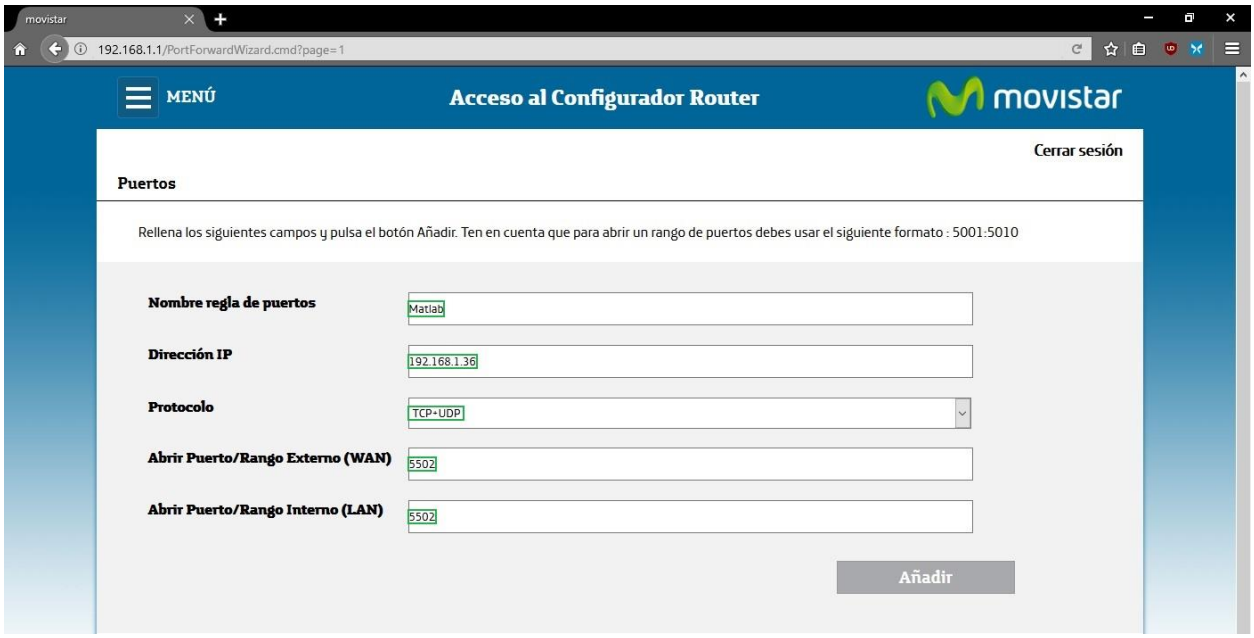

Figura II-13. Apertura de puertos.

Como última medida, es recomendable comprobar que tanto FlightGear como Matlab disponen de privilegios de administrador y se han añadido como excepciones al *firewall*, pese a que una vez abiertos correctamente los puertos que utilizan no debería existir problema alguno.

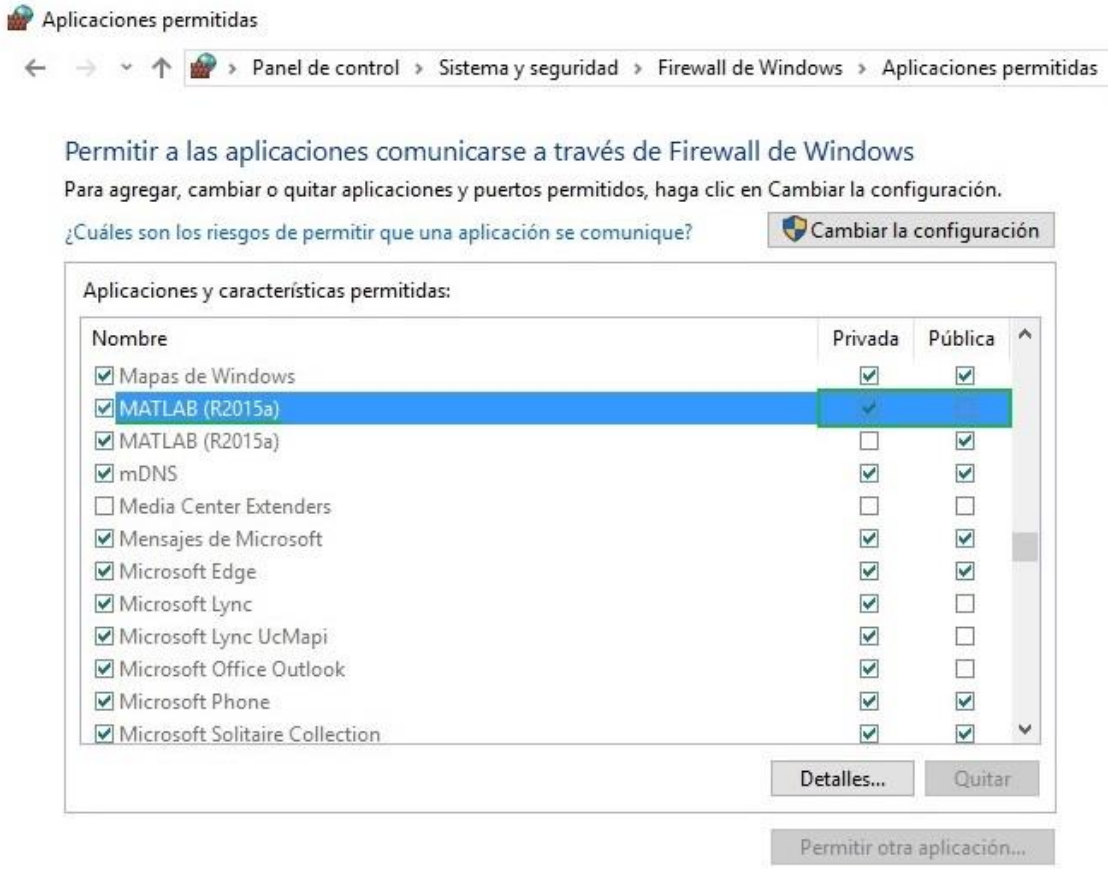

Figura II-14. Permiso a Matlab a través del cortafuegos.

ANEXO III. CÓDIGO

Por último, el siguiente anexo sirve como presentación del código implementado durante todo el proyecto. Todas y cada una de las líneas, a excepción de la función utilizada por el bloque de entrada del teclado en Simulink (véase [18]), han sido producidas por el autor de este documento.

Para facilitar la navegación a través del anexo, se ha dividido en dos secciones en función de si su implementación se ha realizado durante la etapa de modelado o simulación. Todos los *scripts* y funciones se introducen mediante el título de sus respectivos archivos. Asimismo, se ha omitido la repetición del código incluido en ambas partes a no ser que, aún teniendo el mismo título, se hayan producido modificaciones en el mismo.

III.1 Modelado

- *"datos_telemetria.m"*

```
function celdatos = datos_telemetria(vuelo)
% Funcion que carga, interpreta y acondiciona para su posterior
% procesamiento los datos de telemetria de un vuelo concreto del UAV.
%
% celdatos = datos_telemetria(vuelo)
%
% La entrada de la funcion ("vuelo") se define como un numero entre 1 y 5
% correspondiente a cada uno de los cinco vuelos llevados a cabo por el
% UAV.
%
% La salida de la funcion ("celdatos") es una estructura de celdas que
% almacena todos los datos de cada sensor como matrices en cada una de sus
% celdas en formato "[tiempo dato]", donde "tiempo" es el instante de
% tiempo en el cual el sensor realizo la medida con respecto al tiempo de
% referencia, esto es, el encendido de la camara de a bordo justo antes del
% despegue, y "dato" es la medida del sensor en dicho instante de tiempo.
%
% Ejemplo: celdatos = datos_telemetria(3)


%% CARGA DE DATOS, TIEMPO DE REFERENCIA INICIAL Y DEPURACION

% Se cargan los datos de telemetria y se establece el tiempo de referencia
% inicial (instante en el que la camara comienza a grabar) con "datenum".
% La fecha es un dia antes que la marcada en los archivos .tlog cargados.

switch vuelo
```

```
    case 1

        load 2014-06-27_10-38-46.tlog.mat
        tiempo_0_video = datenum('26-Jun-2014 10:44:56');
        ind_aux = [1114 3154 732 725 732 723 3 0];

    case 2

        load 2014-07-08_10-46-15.tlog.mat
        tiempo_0_video = datenum('07-Jul-2014 10:53:30');
        ind_aux = [2621 7423 1761 1710 1759 1704 0 3];

    case 3

        load 2014-07-08_12-02-59.tlog.mat
        tiempo_0_video = datenum('07-Jul-2014 12:06:19');
        ind_aux = [1032 2919 699 672 700 675 0 1];

    case 4

        load 2014-07-30_11-33-34.tlog.mat
        tiempo_0_video = datenum('29-Jul-2014 12:48:05');
        ind_aux = [3223 9125 2108 2111 2111 2095 3 0];

    case 5

        load 2014-07-30_13-34-12.tlog.mat
        tiempo_0_video = datenum('29-Jul-2014 13:43:21');
        ind_aux = [4087 11599 2673 2678 2678 2660 2 0];

end

% NOTA: Si se detectan errores de sincronismo, modificar el instante
% inicial de grabacion. De este modo se actualizaran todas las marcas de
% tiempo de los parametros de telemetria.


%% INFORMACION SOBRE LOS DATOS REGISTRADOS

% Cada parametro que recibe la estacion de tierra (mediante el protocolo
% MAVLink, ver "https://pixhawk.ethz.ch/mavlink/") se guarda en una matriz
% de dos columnas y tantas filas como medidas obtenidas de dicho parametro.
% La primera columna de dichas matrices se corresponde siempre con la fecha
% y hora de la estacion de tierra a la que llego el dato. Esta fecha puede
% reconstruirse con "datestr", teniendo en cuenta que la maxima precision
% del tiempo generado mediante este comando es de un segundo.
%
% Para obtener con mayor precision el momento en que se grabo cada dato,
% existen matrices que permiten relacionar la hora en la que llego cada
% dato con los milisegundos transcurridos desde el arranque de Ardupilot.
% Esto implica que en la practica se puede conocer el instante en el que se
% graba cada medida con una precision de milisegundos. Teniendo en cuenta
% que cada sensor trabaja a una frecuencia de muestreo distinta, se tienen
% cuatro matrices de este tipo:
%
%   - time_boot_ms_mavlink_global_position_int_t: para las medidas del GPS.
%   - time_boot_ms_mavlink_attitude_t: para las medidas de actitud.
```

```
%    - time_usec_mavlink_raw_imu_t: para las medidas de aceleracion.
%    - time_boot_ms_mavlink_rc_channels_raw_t: para las medidas de los
%      canales PWM de los mandos de vuelo.
%    - time_boot_ms_mavlink_scaled_pressure_t: para las medidas
%      barometricas.


%% REFERENCIA DE TIEMPOS

% Para cada tipo de medida, se busca el valor de tiempo en la columna 1
% asignado al instante del inicio del video y se crea una variable con
% dicho instante en ms, convirtiendola en la referencia temporal 0:

ind_t_gps    = find(time_boot_ms_mavlink_global_position_int_t(:,1)>=...
               tiempo_0_video,1);
tiempo_0_gps = time_boot_ms_mavlink_global_position_int_t(ind_t_gps,2);

ind_t_act    = find(time_boot_ms_mavlink_attitude_t(:,1)>=...
               tiempo_0_video,1);
tiempo_0_act = time_boot_ms_mavlink_attitude_t(ind_t_act,2);

ind_t_acel    = find(time_usec_mavlink_raw_imu_t(:,1)>=tiempo_0_video,1);
tiempo_0_acel = time_usec_mavlink_raw_imu_t(ind_t_acel,2);

ind_t_pwm    = find(time_boot_ms_mavlink_rc_channels_raw_t(:,1)>=...
               tiempo_0_video,1);
tiempo_0_pwm = time_boot_ms_mavlink_rc_channels_raw_t(ind_t_pwm,2);

ind_t_pres    = find(time_boot_ms_mavlink_scaled_pressure_t(:,1)...
                >=tiempo_0_video,1);
tiempo_0_pres = time_boot_ms_mavlink_scaled_pressure_t(ind_t_pres,2);

ind_t_vnt    = find(time_boot_ms_mavlink_system_time_t(:,1)...
               >=tiempo_0_video,1);
tiempo_0_vnt = time_boot_ms_mavlink_system_time_t(ind_t_vnt,2);

% Variables de tiempo en ms para todas las medidas. Se crean los vectores
% de tiempo en ms para cada tipo de medicion y se elimina la parte previa
% al inicio del video, es decir, anterior al despegue:

tiempo_gps = time_boot_ms_mavlink_global_position_int_t(:,2)-tiempo_0_gps;
gps_0_ms   = find(tiempo_gps>=0,1);
tiempo_gps = tiempo_gps(gps_0_ms:end);

tiempo_act = time_boot_ms_mavlink_attitude_t(:,2)-tiempo_0_act;
act_0_ms   = find(tiempo_act>=0,1);
tiempo_act = tiempo_act(act_0_ms:end);

tiempo_acel = (time_usec_mavlink_raw_imu_t(:,2)-tiempo_0_acel)/1000;
acel_0_ms   = find(tiempo_acel>=0,1);
tiempo_acel = tiempo_acel(acel_0_ms:end);

tiempo_pwm = time_boot_ms_mavlink_rc_channels_raw_t(:,2)-tiempo_0_pwm;
pwm_0_ms   = find(tiempo_pwm>=0,1);
tiempo_pwm = tiempo_pwm(pwm_0_ms:end);
```

```
tiempo_pres = time_boot_ms_mavlink_scaled_pressure_t(:,2)-tiempo_0_pres;
pres_0_ms   = find(tiempo_pres>=0,1);
tiempo_pres = tiempo_pres(pres_0_ms:end);

tiempo_vnt  = time_boot_ms_mavlink_system_time_t(:,2)-tiempo_0_vnt;
viento_0_ms = find(tiempo_vnt>=0,1);
tiempo_vnt  = tiempo_vnt(viento_0_ms:end);


%% EQUIVALENCIA HORA ESTACION EN TIERRA

% En base a esta referencia de tiempos, para cada parametro registrado, se
% va a construir una matriz que contiene tantas filas como numero de
% medidas de ese parametro, la cual tiene dos columnas que representan:
%
%   - Columna 1: tiempo en milisengudos desde la referencia temporal.
%   - Columna 2: valor del parametro medido (se especifican las unidades
%                en cada caso).

tiempo_0_gs = datestr(time_boot_ms_mavlink_global_position_int_t(1,1));


%% DATOS REGISTRADOS POR LA TELEMETRIA

% COORDENADAS GEOGRAFICAS:

latitud = [tiempo_gps ...
          lat_mavlink_global_position_int_t(gps_0_ms:end,2)/1e7];
latitud = latitud(1:ind_aux(1),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de latitud en grados.

longitud = [tiempo_gps ...
           lon_mavlink_global_position_int_t(gps_0_ms:end,2)/1e7];
longitud = longitud(1:ind_aux(1),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de longitud en grados.

altitud = [tiempo_gps ...
          alt_mavlink_global_position_int_t(gps_0_ms:end,2)/1000];
altitud = altitud(1:ind_aux(1),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de altitud (sobre el elipsoide WGS84) en metros.

altura = [tiempo_gps ...
         relative_alt_mavlink_global_position_int_t(gps_0_ms:end,2)/1000];
altura = altura(1:ind_aux(1),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de altura (sobre el suelo) en metros.

% VELOCIDADES RESPECTO A TIERRA EN EJES NED:

v_norte = [tiempo_gps ...
          vx_mavlink_global_position_int_t(gps_0_ms:end,2)/100];
v_norte = v_norte(1:ind_aux(1),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de la componente de velocidad norte en m/s.
```

```
v_este = [tiempo_gps ...
         vy_mavlink_global_position_int_t(gps_0_ms:end,2)/100];
v_este = v_este(1:ind_aux(1),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de la componente de velocidad este en m/s.

v_vert = [tiempo_gps ...
         -vz_mavlink_global_position_int_t(gps_0_ms:end,2)/100];
v_vert = v_vert(1:ind_aux(1),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de la componente de velocidad vertical en m/s.

% ANGULOS DE EULER:

alabeo = [tiempo_act ...
         roll_mavlink_attitude_t(act_0_ms:end,2)];
alabeo = alabeo(1:ind_aux(2),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de angulo de alabeo en radianes.

cabeceo = [tiempo_act ...
          pitch_mavlink_attitude_t(act_0_ms:end,2)];
cabeceo = cabeceo(1:ind_aux(2),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de angulo de cabeceo en radianes.

guinada = [tiempo_act ...
          yaw_mavlink_attitude_t(act_0_ms:end,2)];
guinada = guinada(1:ind_aux(2),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de angulo de guinada en radianes.

% VELOCIDADES ANGULARES EN EJES CUERPO:

w_alabeo = [tiempo_act ...
           rollspeed_mavlink_attitude_t(act_0_ms:end,2)];
w_alabeo = w_alabeo(1:ind_aux(2),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de velocidad angular de alabeo en radianes/s.

w_cabeceo = [tiempo_act ...
            pitchspeed_mavlink_attitude_t(act_0_ms:end,2)];
w_cabeceo = w_cabeceo(1:ind_aux(2),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de velocidad angular de cabeceo en radianes/s.

w_guinada = [tiempo_act ...
            yawspeed_mavlink_attitude_t(act_0_ms:end,2)];
w_guinada = w_guinada(1:ind_aux(2),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de velocidad angular de guinada en radianes/s.

% ACELERACIONES EN EJES CUERPO:

% Existe un error en las medidas de aceleracion del cuarto vuelo el cual
% queda solucionado con el siguiente codigo:
```

```
if vuelo == 4

    ind_error = 1258;
    error_t   = 4.2944462e+06; %cantidad desplazada en el tiempo

else

    ind_error = 1;
    error_t   = 0;

end

a_x = [tiempo_acel(1:ind_error) ...
      9.81e-3*xacc_mavlink_raw_imu_t(acel_0_ms:acel_0_ms+ind_error-1,2);
      tiempo_acel(ind_error+1:end)+error_t ...
      9.81e-3*xacc_mavlink_raw_imu_t(acel_0_ms+ind_error:end,2)];
a_x = a_x(1:ind_aux(3),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de la componente X de la aceleracion en m/s^2.

a_y = [tiempo_acel(1:ind_error) ...
      9.81e-3*yacc_mavlink_raw_imu_t(acel_0_ms:acel_0_ms+ind_error-1,2);
      tiempo_acel(ind_error+1:end)+error_t ...
      9.81e-3*yacc_mavlink_raw_imu_t(acel_0_ms+ind_error:end,2)];
a_y = a_y(1:ind_aux(3),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de la componente Y de la aceleracion en m/s^2.

a_z = [tiempo_acel(1:ind_error) ...
      -(9.81e-3*zacc_mavlink_raw_imu_t(acel_0_ms:acel_0_ms+ind_error-1,2));
      tiempo_acel(ind_error+1:end)+error_t ...
      -(9.81e-3*zacc_mavlink_raw_imu_t(acel_0_ms+ind_error:end,2))];
a_z = a_z(1:ind_aux(3),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de la componente Z de la aceleracion en m/s^2.

% SENALES PWM DE LOS ACTUADORES:

alerones_pwm = [tiempo_pwm ...
                chan1_raw_mavlink_rc_channels_raw_t(pwm_0_ms:end,2)];
alerones_pwm = alerones_pwm(1:ind_aux(4),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de senal del canal 1 (alerones) del PWM en us.

elevadores_pwm = [tiempo_pwm ...
                  chan2_raw_mavlink_rc_channels_raw_t(pwm_0_ms:end,2)];
elevadores_pwm = elevadores_pwm(1:ind_aux(4),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de senal del canal 2 (elevadores) del PWM en us.

motor_pwm = [tiempo_pwm ...
             chan3_raw_mavlink_rc_channels_raw_t(pwm_0_ms:end,2)];
motor_pwm = motor_pwm(1:ind_aux(4),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de senal del canal 3 (motor) del PWM en us.
```

```
% MEDIDAS BAROMETRICAS:

presion_abs = [tiempo_pres ...
               press_abs_mavlink_scaled_pressure_t(pres_0_ms:end,2)*100];
presion_abs = presion_abs(1:ind_aux(5),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de presion absoluta en pascales.

presion_dif = [tiempo_pres ...
               press_diff_mavlink_scaled_pressure_t(pres_0_ms:end,2)*100];
presion_dif = presion_dif(1:ind_aux(5),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de presion diferencial en pascales.

% MEDIDAS DE VIENTO

viento_mag_horz = [tiempo_vnt(1:end-ind_aux(7)) ...
                   speed_mavlink_wind_t(viento_0_ms:end-ind_aux(8),2)];
viento_mag_horz = viento_mag_horz(1:ind_aux(6),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de la componente horizontal de viento en m/s.

viento_mag_vert = [tiempo_vnt(1:end-ind_aux(7)) ...
                   -speed_z_mavlink_wind_t(viento_0_ms:end-ind_aux(8),2)];
viento_mag_vert = viento_mag_vert(1:ind_aux(6),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de la componente vertical de viento en m/s.

viento_dir = [tiempo_vnt(1:end-ind_aux(7)) ...
              mod(direction_mavlink_wind_t(viento_0_ms:end-ind_aux(8),2)...
              .*(pi/180)+pi,2*pi)];
viento_dir = viento_dir(1:ind_aux(6),:);
% 1ª columna: tiempo en s.
% 2ª columna: medida de direccion de viento en radianes.

%% CELDATOS

celdatos{1}  = latitud;
celdatos{2}  = longitud;
celdatos{3}  = altitud;
celdatos{4}  = altura;
celdatos{5}  = v_norte;
celdatos{6}  = v_este;
celdatos{7}  = v_vert;
celdatos{8}  = alabeo;
celdatos{9}  = cabeceo;
celdatos{10} = guinada;
celdatos{11} = w_alabeo;
celdatos{12} = w_cabeceo;
celdatos{13} = w_guinada;
celdatos{14} = a_x;
celdatos{15} = a_y;
celdatos{16} = a_z;
celdatos{17} = alerones_pwm;
celdatos{18} = elevadores_pwm;
celdatos{19} = motor_pwm;
celdatos{20} = presion_abs;
celdatos{21} = presion_dif;
celdatos{22} = viento_mag_horz;
```

```
celdatos{23} = viento_mag_vert;
celdatos{24} = viento_dir;
```

- *"fuerzasx.m"*

```
function delta_t = fuerzasx(entrada)
% Funcion que despeja la fuerza propulsiva de la ecuacion de la segunda ley
% de Newton en la componente longitudinal del UAV.
%
% salida = fuerzasx(entrada)
%
% Siendo:
%
% Ua              = entrada(1);
% Va              = entrada(2);
% Wa              = entrada(3);
% U               = entrada(4);
% V               = entrada(5);
% W               = entrada(6);
% a_x             = entrada(7);
% Q               = entrada(8);
% R               = entrada(9);
% theta           = entrada(10);
% elevadores_pwm  = entrada(11);
% rho             = entrada(12);
% altitud         = entrada(13);
% latitud         = entrada(14);
%
% delta_t         = salida;


%% ENTRADA

Ua              = entrada(1);
Va              = entrada(2);
Wa              = entrada(3);
U               = entrada(4);
V               = entrada(5);
W               = entrada(6);
a_x             = entrada(7);
Q               = entrada(8);
R               = entrada(9);
theta           = entrada(10);
elevadores_pwm  = entrada(11);
rho             = entrada(12);
altitud         = entrada(13);
latitud         = entrada(14);


%% DATOS

if exist('m')~=1

    x8;

end
```

```
%% COEFICIENTES

Uinf      = sqrt(Ua^2+Va^2+Wa^2);
alpha     = atan(Wa/(Ua+1e-16));
beta      = asin(Va/(Uinf+1e-16));
delta_e   = -((elevadores_pwm-1520)*(30*pi/180))/(2*(2100-1520));
[C_L,C_D] = x8_aerox(alpha,beta,Uinf,Q,delta_e);


%% MODELO GRAVITATORIO

g = gravitywgs84(altitud,latitud);


%% CALCULO DE GRAVEDAD

Gx = -m*g*sin(theta);


%% MATRIZ DE ROTACION DE EJES VIENTO A EJES CUERPO (EJE X)

R_wbx = [cos(alpha)*cos(beta) -cos(alpha)*sin(beta) -sin(alpha)];


%% CALCULO DE FUERZAS AERODINAMICAS

Fx_aero = -0.5*rho*Uinf^2*Salar*R_wbx*[C_D;0;C_L];


%% CALCULO DE FUERZAS TOTALES

delta_t = m*(a_x+[0 -R Q]*[U;V;W])-Gx-Fx_aero;
```

- *"funleyprop.m"*

```
function [delta_t,Ua,altitud,motor_pwm] = funleyprop(vuelo)
% Funcion utilizada a la hora de calcular la ley de propulsion con el
% objetivo de preprocesar distintas variables.


%% CALCULO

celdatos = datos_telemetria(vuelo);
procesador_datos;

Ua = [];

for k=1:longitud_max

    phi = alabeo(k,2); theta = cabeceo(k,2); psi = guinada(k,2);

    % Matriz de rotacion de ejes NED a ejes cuerpo:
```

```
    R_nb = [cos(theta)*cos(psi) cos(theta)*sin(psi) -sin(theta); ...
            sin(phi)*sin(theta)*cos(psi)-cos(phi)*sin(psi) sin(phi)*...
            sin(theta)*sin(psi)+cos(phi)*cos(psi) sin(phi)*cos(theta); ...
            cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi) cos(phi)*...
            sin(theta)*sin(psi)-sin(phi)*cos(psi) cos(phi)*cos(theta)];

    v_tierra_b = R_nb*[v_norte(k,2); v_este(k,2); v_vert(k,2)];

    w_total = sqrt(viento_mag_horz(k,2)^2+viento_mag_vert(k,2)^2);
    w_norte = -w_total*cos(viento_dir(k,2));
    w_este  = -w_total*sin(viento_dir(k,2));
    w_vert  = viento_mag_vert(k,2);

    v_viento_b = R_nb*[w_norte;w_este;w_vert];
    v_aero_b   = v_tierra_b-v_viento_b;

    Ua = [Ua; v_aero_b(1)];

    entrada = [v_aero_b(1) v_aero_b(2) v_aero_b(3) v_tierra_b(1) ...
               v_tierra_b(2) v_tierra_b(3) a_x(k,2) w_cabeceo(k,2) ...
               w_guinada(k,2) cabeceo(k,2) elevadores_pwm(k,2) ...
               densidad(k,2) altitud(k,2) latitud(k,2)];

   delta_t(k,1) = fuerzasx(entrada);

end
```

- *"graficas_telemetria.m"*

```
% Script que representa los datos de telemetria procesados o no procesados,
% en diversas graficas y correspondientes a un vuelo determinado.

close all, clear all

flag1 = 0;

while flag1==0

    texto1   = ['Pulse un numero entre 1 y 5 para analizar la ' ...
                'telemetria del vuelo correspondiente. A ' ...
                'continuacion, presione la tecla ENTER.\n'];
    entrada1 = input(texto1);

    if entrada1<1 || entrada1>5

        error1 = ['Debe pulsar una tecla entre 1 y 5 (1,2,3,4 o 5). ' ...
                  'Por favor, intentelo de nuevo.'];
        disp(error1);

    else

        flag1 = 1;

    end
```

```
end

flag2 = 0;

while flag2==0

    texto2   = ['Pulse 0 para visualizar los datos no procesados o 1 ' ...
                'para visualizar los datos procesados. A ' ...
                'continuacion, presione la tecla ENTER.\n'];
    entrada2 = input(texto2);

    if entrada2<0 || entrada2>1

        error2 = ['Debe pulsar la tecla 0 o 1. Por favor, intentelo ' ...
                  'de nuevo.'];
        disp(error2);

    else

        flag2 = 1;

    end

end

celdatos = datos_telemetria(entrada1);

if entrada2==1

    procesador_datos;

    figure;
    subplot(2,2,1);
    plot(celproces{1}(:,1)*1e-3,celproces{1}(:,2));
    grid; title('Latitud');
    xlabel('t [s]'); ylabel('\Phi [°]');
    subplot(2,2,2);
    plot(celproces{2}(:,1)*1e-3,celproces{2}(:,2));
    grid; title('Longitud');
    xlabel('t [s]'); ylabel('\Lambda [°]');
    subplot(2,2,3);
    plot(celproces{3}(:,1)*1e-3,celproces{3}(:,2));
    grid; title('Altitud');
    xlabel('t [s]'); ylabel('h [m]');
    subplot(2,2,4);
    plot(celproces{4}(:,1)*1e-3,celproces{4}(:,2));
    grid; title('Altura');
    xlabel('t [s]'); ylabel('H [m]');

    figure;
    subplot(3,1,1);
    plot(celproces{5}(:,1)*1e-3,celproces{5}(:,2));
    grid; title('V_G_P_S norte');
    xlabel('t [s]'); ylabel('V_N [m/s]');
    subplot(3,1,2);
    plot(celproces{6}(:,1)*1e-3,celproces{6}(:,2));
```

```
grid; title('V_G_P_S este');
xlabel('t [s]'); ylabel('V_E [m/s]');
subplot(3,1,3);
plot(celproces{7}(:,1)*1e-3,celproces{7}(:,2));
grid; title('V_G_P_S vertical');
xlabel('t [s]'); ylabel('V_D [m/s]');

figure;
subplot(2,3,1);
plot(celproces{8}(:,1)*1e-3,rem(celproces{8}(:,2).*(180/pi),180));
grid; title('Alabeo');
xlabel('t [s]'); ylabel('\phi [°]');
subplot(2,3,2);
plot(celproces{9}(:,1)*1e-3,rem(celproces{9}(:,2).*(180/pi),90));
grid; title('Cabeceo');
xlabel('t [s]'); ylabel('\theta [°]');
subplot(2,3,3);
plot(celproces{10}(:,1)*1e-3,mod(celproces{10}(:,2).*(180/pi),360));
grid; title('Guinada');
xlabel('t [s]'); ylabel('\psi [°]');
subplot(2,3,4);
plot(celproces{11}(:,1)*1e-3,celproces{11}(:,2).*(180/pi));
grid; title('Velocidad de alabeo');
xlabel('t [s]'); ylabel('P [°/s]');
subplot(2,3,5);
plot(celproces{12}(:,1)*1e-3,celproces{12}(:,2).*(180/pi));
grid; title('Velocidad de cabeceo');
xlabel('t [s]'); ylabel('Q [°/s]');
subplot(2,3,6);
plot(celproces{13}(:,1)*1e-3,celproces{13}(:,2).*(180/pi));
grid; title('Velocidad de guinada');
xlabel('t [s]'); ylabel('R [°/s]');

figure;
subplot(3,1,1);
plot(celproces{14}(:,1)*1e-3,celproces{14}(:,2));
grid; title('Aceleracion X');
xlabel('t [s]'); ylabel('A_X [m/s^2]');
subplot(3,1,2);
plot(celproces{15}(:,1)*1e-3,celproces{15}(:,2));
grid; title('Aceleracion Y');
xlabel('t [s]'); ylabel('A_Y [m/s^2]');
subplot(3,1,3);
plot(celproces{16}(:,1)*1e-3,celproces{16}(:,2));
grid; title('Aceleracion Z');
xlabel('t [s]'); ylabel('A_Z [m/s^2]');

figure;
subplot(3,1,1);
plot(celproces{17}(:,1)*1e-3,celproces{17}(:,2));
grid; title('Alerones');
xlabel('t [s]'); ylabel('Canal 1 PWM (alerones) [\mus]');
subplot(3,1,2);
plot(celproces{18}(:,1)*1e-3,celproces{18}(:,2));
grid; title('Elevadores');
xlabel('t [s]'); ylabel('Canal 2 PWM (elevadores) [\mus]');
subplot(3,1,3);
plot(celproces{19}(:,1)*1e-3,celproces{19}(:,2));
grid; title('Motor');
```

```
figure;
subplot(2,2,1);
plot(celproces{20}(:,1)*1e-3,celproces{20}(:,2));
grid; title('Presion absoluta');
xlabel('t [s]'); ylabel('P_a_b_s [Pa]');
subplot(2,2,2)
plot(celproces{25}(:,1)*1e-3,celproces{25}(:,2));
grid; title('Presion estatica');
xlabel('t [s]'); ylabel('P_e_s_t [Pa]');
subplot(2,2,3);
plot(celproces{21}(:,1)*1e-3,celproces{21}(:,2));
grid; title('Presion diferencial');
xlabel('t [s]'); ylabel('P_e_s_t-P_a_b_s [Pa]');
subplot(2,2,4);
plot(celproces{26}(:,1)*1e-3,celproces{26}(:,2));
grid; title('Densidad');
xlabel('t [s]'); ylabel('\rho [kg/m^3]');

figure;
subplot(3,1,1);
plot(celproces{22}(:,1)*1e-3,celproces{22}(:,2));
grid; title('Magnitud de la componente horizontal del viento');
xlabel('t [s]'); ylabel('V_H_ _v_i_e_n_t_o [m/s]');
subplot(3,1,2);
plot(celproces{23}(:,1)*1e-3,celproces{23}(:,2));
grid; title('Magnitud de la componente vertical del viento');
xlabel('t [s]'); ylabel('V_V_ _v_i_e_n_t_o [m/s]');
subplot(3,1,3);
plot(celproces{24}(:,1)*1e-3,celproces{24}(:,2).*(180/pi));
grid; title('Direccion del viento');
xlabel('t [s]'); ylabel('\beta_v_i_e_n_t_o [°]');

v_tierra_b = [];
v_viento_b = [];
v_aero_b   = [];

for k=1:longitud_max

    phi = alabeo(k,2); theta = cabeceo(k,2); psi = guinada(k,2);

    % Matriz de rotacion de ejes NED a ejes cuerpo:

    R_nb = [cos(theta)*cos(psi) cos(theta)*sin(psi) -sin(theta); ...
           sin(phi)*sin(theta)*cos(psi)-cos(phi)*sin(psi) sin(phi)*...
           sin(theta)*sin(psi)+cos(phi)*cos(psi) sin(phi)*...
           cos(theta); cos(phi)*sin(theta)*cos(psi)+sin(phi)*...
           sin(psi) cos(phi)*sin(theta)*sin(psi)-sin(phi)*cos(psi) ...
           cos(phi)*cos(theta)];

    v_tierra_b = [v_tierra_b; norm(R_nb*[v_norte(k,2); v_este(k,2);...
                  v_vert(k,2)])];

    w_total = sqrt(viento_mag_horz(k,2)^2+viento_mag_vert(k,2)^2);
    w_norte = -w_total*cos(viento_dir(k,2));
    w_este  = -w_total*sin(viento_dir(k,2));
    w_vert  = viento_mag_vert(k,2);
```

```
        v_viento_b = [v_viento_b; norm(R_nb*[w_norte;w_este;w_vert])];

        v_aero_b = [v_aero_b; v_tierra_b(k)-v_viento_b(k)];

    end

    figure;
    subplot(3,1,1);
    plot(v_norte(:,1)*1e-3,v_tierra_b);
    grid; title('V_U_A_V respecto a tierra');
    xlabel('t [s]'); ylabel('V_t_i_e_r_r_a [m/s]');
    subplot(3,1,2);
    plot(celproces{22}(:,1)*1e-3,v_viento_b);
    grid; title('V_v_i_e_n_t_o respecto a tierra');
    xlabel('t [s]'); ylabel('V_v_i_e_n_t_o [m/s]');
    subplot(3,1,3);
    plot(celproces{22}(:,1)*1e-3,v_aero_b);
    grid; title('V aerodinamica');
    xlabel('t [s]'); ylabel('V_a_e_r_o [m/s]');

elseif entrada2==0

    figure;
    subplot(2,2,1);
    plot(celdatos{1}(:,1)*1e-3,celdatos{1}(:,2));
    grid; title('Latitud');
    xlabel('t [s]'); ylabel('\Phi [°]');
    subplot(2,2,2);
    plot(celdatos{2}(:,1)*1e-3,celdatos{2}(:,2));
    grid; title('Longitud');
    xlabel('t [s]'); ylabel('\Lambda [°]');
    subplot(2,2,3);
    plot(celdatos{3}(:,1)*1e-3,celdatos{3}(:,2));
    grid; title('Altitud');
    xlabel('t [s]'); ylabel('h [m]');
    subplot(2,2,4);
    plot(celdatos{4}(:,1)*1e-3,celdatos{4}(:,2));
    grid; title('Altura');
    xlabel('t [s]'); ylabel('H [m]');

    figure;
    subplot(3,1,1);
    plot(celdatos{5}(:,1)*1e-3,celdatos{5}(:,2));
    grid; title('V_G_P_S norte');
    xlabel('t [s]'); ylabel('V_N [m/s]');
    subplot(3,1,2);
    plot(celdatos{6}(:,1)*1e-3,celdatos{6}(:,2));
    grid; title('V_G_P_S este');
    xlabel('t [s]'); ylabel('V_E [m/s]');
    subplot(3,1,3);
    plot(celdatos{7}(:,1)*1e-3,celdatos{7}(:,2));
    grid; title('V_G_P_S vertical');
    xlabel('t [s]'); ylabel('V_D [m/s]');

    figure;
    subplot(2,3,1);
    plot(celdatos{8}(:,1)*1e-3,rem(celdatos{8}(:,2).*(180/pi),180));
    grid; title('Alabeo');
    xlabel('t [s]'); ylabel('\phi [°]');
```

```
subplot(2,3,2);
plot(celdatos{9}(:,1)*1e-3,rem(celdatos{9}(:,2).*(180/pi),90));
grid; title('Cabeceo');
xlabel('t [s]'); ylabel('\theta [°]');
subplot(2,3,3);
plot(celdatos{10}(:,1)*1e-3,mod(celdatos{10}(:,2).*(180/pi),360));
grid; title('Guinada');
xlabel('t [s]'); ylabel('\psi [°]');
subplot(2,3,4);
plot(celdatos{11}(:,1)*1e-3,celdatos{11}(:,2).*(180/pi));
grid; title('Velocidad de alabeo');
xlabel('t [s]'); ylabel('P [°/s]');
subplot(2,3,5);
plot(celdatos{12}(:,1)*1e-3,celdatos{12}(:,2).*(180/pi));
grid; title('Velocidad de cabeceo');
xlabel('t [s]'); ylabel('Q [°/s]');
subplot(2,3,6);
plot(celdatos{13}(:,1)*1e-3,celdatos{13}(:,2).*(180/pi));
grid; title('Velocidad de guinada');
xlabel('t [s]'); ylabel('R [°/s]');

figure;
subplot(3,1,1);
plot(celdatos{14}(:,1)*1e-3,celdatos{14}(:,2));
grid; title('Aceleracion X');
xlabel('t [s]'); ylabel('A_X [m/s^2]');
subplot(3,1,2);
plot(celdatos{15}(:,1)*1e-3,celdatos{15}(:,2));
grid; title('Aceleracion Y');
xlabel('t [s]'); ylabel('A_Y [m/s^2]');
subplot(3,1,3);
plot(celdatos{16}(:,1)*1e-3,celdatos{16}(:,2));
grid; title('Aceleracion Z');
xlabel('t [s]'); ylabel('A_Z [m/s^2]');

figure;
subplot(3,1,1);
plot(celdatos{17}(:,1)*1e-3,celdatos{17}(:,2));
grid; title('Alerones');
xlabel('t [s]'); ylabel('Canal 1 PWM (alerones) [\mus]');
subplot(3,1,2);
plot(celdatos{18}(:,1)*1e-3,celdatos{18}(:,2));
grid; title('Elevadores');
xlabel('t [s]'); ylabel('Canal 2 PWM (elevadores) [\mus]');
subplot(3,1,3);
plot(celdatos{19}(:,1)*1e-3,celdatos{19}(:,2));
grid; title('Motor');
xlabel('t [s]'); ylabel('Canal 3 PWM (motor) [\mus]');

figure;
subplot(2,1,1);
plot(celdatos{20}(:,1)*1e-3,celdatos{20}(:,2));
grid; title('Presion absoluta');
xlabel('t [s]'); ylabel('P_a_b_s [Pa]');
subplot(2,1,2);
plot(celdatos{21}(:,1)*1e-3,celdatos{21}(:,2));
grid; title('Presion diferencial');
xlabel('t [s]'); ylabel('P_e_s_t-P_a_b_s [Pa]');
```

```
    figure;
    subplot(3,1,1);
    plot(celdatos{22}(:,1)*1e-3,celdatos{22}(:,2));
    grid; title('Magnitud de la componente horizontal del viento');
    xlabel('t [s]'); ylabel('V_H_ _v_i_e_n_t_o [m/s]');
    subplot(3,1,2);
    plot(celdatos{23}(:,1)*1e-3,celdatos{23}(:,2));
    grid; title('Magnitud de la componente vertical del viento');
    xlabel('t [s]'); ylabel('V_V_ _v_i_e_n_t_o [m/s]');
    subplot(3,1,3);
    plot(celdatos{24}(:,1)*1e-3,celdatos{24}(:,2).*(180/pi));
    grid; title('Direccion del viento');
    xlabel('t [s]'); ylabel('\beta_v_i_e_n_t_o [°]');

end
```

- *"ley_propulsiva.m"*

```
% Script que permite calcular la ley propulsiva. Calcula delta_t en Newton
% a partir de los datos de telemetria de todos los vuelos ademas de
% representar graficas de interes con los resultados.

close all, clear all

%% CALCULO

delta_t         = [];
Ua              = [];
altitud         = [];
motor_pwm       = [];

for vuelo=1:5

    [delta_t2,Ua2,altitud2,motor_pwm2] = funleyprop(vuelo);

    delta_t         = [delta_t;delta_t2];
    Ua              = [Ua;Ua2];
    altitud         = [altitud;altitud2];
    motor_pwm       = [motor_pwm;motor_pwm2];

end

% Se ajustan los resultados por minimos cuadrados:

A1   = motor_pwm(:,2)-1100;
A2   = Ua.^2;
b    = delta_t;
long = length(A1);
A    = [A1(1) A2(1)];

for k=2:long

    A = [A; A1(k) A2(k)];

end
```

```
coefprop = (A.'*A)\(A.'*b);

% Grafica ajustada:

paso            = 10;
motor_pwm_ajust = 1100:paso:2100;
Ua_ajust         = 0:0.03*paso:30;
long            = length(motor_pwm_ajust);

for k=1:long

    delta_t_ajust(:,k) = (coefprop.'*[(motor_pwm_ajust(k)-1100)...
                          *ones(1,long);Ua_ajust.^2]).';

end

% Segun establece el PFC de Benito Fernandez Rojas, la potencia electrica
% generada en el motor es funcion de delta_t:

P_electrica = 0.0415315*((motor_pwm-1100)./1000).^2;


%% FIGURAS

figure; plot3(motor_pwm(:,2),Ua,delta_t,'.'); grid;
titulo1 = ['Fuerza propulsiva en funcion de la senal PWM del motor y ' ...
           'la componente longitudinal de la velocidad aerodinamica'];
title(titulo1);
xlabel('Canal 3 PWM (motor) [\mus]');
ylabel('Ua [m/s]');
zlabel('Delta t [N]');

figure; grid;
grafica = scatter3(motor_pwm(:,2),Ua,delta_t,18,altitud(:,2),'.', ...
                   'MarkerFaceColor','auto');
grafica.Parent.Position = [0.1 0.1 0.76 0.82];
titulo2 = ['Fuerza propulsiva en funcion de la senal PWM del motor, ' ...
           'la componente longitudinal de la velocidad aerodinamica ' ...
           'y la altitud'];
title(titulo2);
xlabel('Canal 3 PWM (motor) [\mus]');
ylabel('Ua [m/s]');
zlabel('Delta t [N]');
barra_colores               = colorbar;
barra_colores.TickLabels    = {'0';'25';'50';'75';'100';'125';'150';...
                               '175';'200'};
barra_colores.Ticks         = [0:25:200];
barra_colores.Position      = [0.9 0.11 0.02 0.8];
barra_colores.Label.String = 'Altitud [m]';
barra_colores.FontSize      = 11;

figure;
plot3(ones(long,1)*motor_pwm_ajust,Ua_ajust.'*ones(1,long),...
      delta_t_ajust,'b.');
grid;
texto   = ['Fuerza propulsiva ajustada por minimos cuadrados: ' ...
           'Delta_t = %.6g·Motor_P_W_M%.6g·Ua^2'];
titulo3 = sprintf(texto,coefprop(1),coefprop(2));
```

```
title(titulo3);
xlabel('Canal 3 PWM (motor) [\mus]');
ylabel('Ua [m/s]');
zlabel('Delta t [N]');
```

- *"procesador_datos.m"*

```
% Script que procesa los datos contenidos en las celdas generadas por
% interprete_datos.m. El motivo es la necesidad de homogeneizar los datos
% provenientes de los distintos sensores de abordo, los cuales no trabajan
% a las mismas frecuencias de refresco, con el fin de poder utilizarlos
% para el calculo de la ley propulsiva, la cual relaciona la entrada de la
% senal PWM del motor con la fuerza generada por el mismo.
%
% El algoritmo aplicado se basa en un proceso similar al Zero-Order Hold
% (ZOH):
%
% 1.- Se elige la celda cuya matriz es mas larga (perteneciente al sensor
%     con la tasa de refresco mas alta) y se unifica esta misma longitud
%     para todas las demas celdas (dejando invariables a las de mayor
%     longitud).
%
% 2.- Se modifican las celdas cuyas matrices no tengan la longitud maxima
%     mediante un proceso ZOH, rellenandolas de este modo para los nuevos
%     valores de tiempo.


%% CALCULO

% Se miden las longitudes de las matrices y se dejan invariables aquellas
% que tengan la mayor longitud. Se generan vectores en los que se
% diferencian los parametros cuya longitud de matriz es maxima de los que
% no, caracterizados por su indice dentro de celdatos:

longitud  = [];
ind_datos = [1:size(celdatos,2)];

for k=1:ind_datos(end)

    longitud(k) = size(celdatos{k},1);

end

[longitud_max,ind_max] = max(longitud);
ind_max_repetidos      = find(longitud==longitud_max);

for k=ind_max_repetidos(1):ind_max_repetidos(end)

    celproces{k} = celdatos{k};

end

ind_no_max                    = ind_datos;
ind_no_max(ind_max_repetidos) = [];
```

```
% Se modifican las matrices de datos que no tengan una longitud maxima con
% tal de igualarlas a las de longitud maxima. Para ello se mantienen los
% datos hasta que se produce un cambio en el valor de tiempo (columna 1 de
% matriz de cada una de las celdas):

for i=ind_no_max

    celproces{i}(1,:) = celdatos{i}(1,:);

    if celdatos{i}(end,1)<celdatos{ind_max}(end,1)

        aux = 1;

    else

        aux = 0;

    end

    for j=2:longitud(i)

        ind_inicial = size(celproces{i},1);
        ind_final   = find(celproces{ind_max}(:,1)<=...
                      celdatos{i}(j,1),1,'last');

        celproces{i}(ind_inicial+1:ind_final,1) = ...
        celdatos{ind_max}(ind_inicial+1:ind_final,1);
        celproces{i}(ind_inicial+1:ind_final,2) = ...
        celproces{i}(ind_inicial,2);
        celproces{i}(ind_final,:) = celdatos{i}(j,:);

        if j~=longitud(i)

            if
celproces{i}(ind_final,1)~=celproces{ind_max}(ind_final+1,1)

                celproces{i}(ind_final,:) = celdatos{i}(j,:);

            else

                celproces{i}(ind_final,:) = celdatos{i}(ind_final+1,:);

            end

        else

            if aux==0

                celproces{i}(ind_final,:) = celdatos{i}(j,:);

            else

                celproces{i}(ind_final,:) = celdatos{i}(j,:);

                for k=ind_final+1:longitud_max
```

```
                    celproces{i}(k,1) = celdatos{ind_max}(k,1);
                    celproces{i}(k,2) = celdatos{i}(j,2);

                end

            end

        end

    end

end


%% CELPROCES

latitud         = celproces{1};
longitud        = celproces{2};
altitud         = celproces{3};
altura          = celproces{4};
v_norte         = celproces{5};
v_este          = celproces{6};
v_vert          = celproces{7};
alabeo          = celproces{8};
cabeceo         = celproces{9};
guinada         = celproces{10};
w_alabeo        = celproces{11};
w_cabeceo       = celproces{12};
w_guinada       = celproces{13};
a_x             = celproces{14};
a_y             = celproces{15};
a_z             = celproces{16};
alerones_pwm    = celproces{17};
elevadores_pwm  = celproces{18};
motor_pwm       = celproces{19};
presion_abs     = celproces{20};
presion_dif     = celproces{21};
viento_mag_horz = celproces{22};
viento_mag_vert = celproces{23};
viento_dir      = celproces{24};


%% CALCULO DE OTROS PARAMETROS

% Se anaden dos celdas nuevas (presion estatica y densidad), las
% cuales requerian procesamiento para ser creadas:

presion_est = [presion_abs(:,1) presion_abs(:,2)+presion_dif(:,2)];

rho0 = 1.225225;
T0   = 288.15;
a    = -6.5e-3;
g    = 9.81;
R    = 287;

densidad = [presion_abs(:,1) ...
            rho0*((T0+a.*altitud(:,2))/T0).^(-(g/(a*R))-1)];
% 1ª columna: tiempo en s.
```

```
% 2ª columna: calculo de densidad (ISA) en kg/m^3.

celproces{25} = presion_est;
celproces{26} = densidad;
```

- *"sigmoide.m"*

```
function sigma = sigmoide(alpha)
% Funcion que obtiene el valor de sigma (variable de las ecuaciones de
% dinamica del UAV) a partir de alpha.

M       = 50; % ratio de transicion (suaviza la transicion a placa plana)
alpha0  = 0.2670;
alpha   = abs(alpha);
sigma   = (1+exp(-M*(alpha-alpha0))+exp(M*(alpha+alpha0)))/((1+exp(-M*...
          (alpha-alpha0)))*(1+exp(M*(alpha+alpha0))));

if sigma>=1

    sigma = 1;

end

if sigma<=0

    sigma = 0;

end
```

- *"x8.m"*

```
% Script que contiene las caracteristicas tecnicas del Skywalker X8.

global m b c Salar AR Ix Iy Iz Ixz e alpha0 coefprop C_L0 C_Lalpha C_Lq ...
       C_Ldelta_e C_D0 C_Dbeta1 C_Dbeta2 C_Dq C_Ddelta_e C_Y0 C_Ybeta ...
       C_Yp C_Yr C_Ydelta_a C_l0 C_lbeta C_lp C_lr C_ldelta_a C_m0 ...
       C_malpha C_mfp C_mq C_mdelta_e C_n0 C_nbeta C_np C_nr C_ndelta_a ...
       gamma gamma1 gamma2 gamma3 gamma4 gamma5 gamma6 gamma7 gamma8

%% DATOS GENERALES DEL UAV

m      = 3.797;
b      = 2.1000;
c      = 0.3571;
Salar  = 0.7500;
AR     = b^2/Salar; % alargamiento (Aspect Ratio)


%% MOMENTOS DE INERCIA

Ix  = 1.2290;
Iy  = 0.1702;
```

```
Iz  = 0.8808;
Ixz = 0.9343;


%% DATOS AERODINAMICOS

e      = 0.9935; % coeficiente de Oswald
alpha0 = 0.2670; % alpha de entrada en perdida


%% LEY DE PROPULSION

coefprop = [0.0168798 -0.0422854]; % motor_pwm y U^2


%% COEFICIENTES DE ESTABILIDAD

C_L0       = 0.0254;
C_Lalpha   = 4.0191; % rad^-1
C_Lq       = 3.8954;
C_Ldelta_e = 0.5872;

C_D0       = 0.0102;
C_Dbeta1   = -2.0864e-7;
C_Dbeta2   = 0.0671;
C_Dq       = 0;
C_Ddelta_e = 0.8461;

C_Y0       = 3.2049e-18;
C_Ybeta    = -0.1949;
C_Yp       = -0.1172;
C_Yr       = 0.0959;
C_Ydelta_a = -0.0696;

C_l0       = 1.1518e-18;
C_lbeta    = -0.0765;
C_lp       = -0.4018;
C_lr       = 0.0250;
C_ldelta_a = 0.2987;

C_m0       = 0.0180;
C_malpha   = -0.2524;
C_mfp      = -0.2168;
C_mq       = -1.3047;
C_mdelta_e = -0.4857;

C_n0       = -2.2667e-7;
C_nbeta    = 0.0403;
C_np       = -0.0247;
C_nr       = -0.1252;
C_ndelta_a = 0.0076;


%% GAMMAS

gamma  = Ix*Iz-Ixz^2;
gamma1 = (Ixz*(Ix-Iy+Iz))/gamma;
```

```
gamma2 = (Iz*(Iz-Iy)+Ixz^2)/gamma;
gamma3 = Iz/gamma;
gamma4 = Ixz/gamma;
gamma5 = (Iz-Ix)/Iy;
gamma6 = Ixz/Iy;
gamma7 = (Ix*(Ix-Iy)+Ixz^2)/gamma;
gamma8 = Ix/gamma;
```

- *"x8_aerox.m"*

```
function [C_L,C_D] = x8_aerox(alpha,beta,Uinf,Q,delta_e)
% Funcion que calcula el valor de los coeficientes aerodinamicos C_L y C_D.
%
% [C_L,C_D] = x8_modelx(alpha,beta,Uinf,Q,delta_e)
%
% Siendo:
%
% alpha   = angulo de ataque [rad]
% beta    = angulo de deslizamiento [rad]
% Uinf    = velocidad aerodinamica en el infinito [m/s]
% Q       = momento de las fuerzas alrededor del eje Y (cuerpo) [N·m]
% delta_e = desflexion de los elevadores [rad]
%
% C_L     = coeficiente de sustentacion [-]
% C_D     = coeficiente de resistencia [-]


%% DATOS

if exist('m')~=1

    x8;

end

sigma = sigmoide(alpha);


%% CALCULO DE COEFICIENTES AERODINAMICOS

C_L = (1-sigma)*(C_L0+C_Lalpha*alpha)+sigma*(2*sign(alpha)*sin(alpha)^2*...
      cos(alpha))+0.5*C_Lq*c*Q/(Uinf+1e-16)+C_Ldelta_e*delta_e;
C_D = C_D0+(1-sigma)*((C_L0+C_Lalpha*alpha)^2/(pi*e*AR))+sigma*(2*...
      sign(alpha)*sin(alpha)^3)+0.5*C_Dq*c*Q/(Uinf+1e-16)+C_Dbeta1*beta+...
      C_Dbeta2*beta^2+C_Ddelta_e*delta_e;
```

III.2 Simulación

- *"aceleraciones_ang.m"*

```
function salida = aceleraciones_ang(entrada)
% Funcion que calcula las aceleraciones angulares experimentadas por el UAV
% en ejes cuerpo.
%
% salida = aceleraciones_ang(entrada)
%
% Siendo:
%
% P       = entrada(1);
% Q       = entrada(2);
% R       = entrada(3);
% L       = entrada(4);
% M       = entrada(5);
% N       = entrada(6);
% Iy      = entrada(7);
% gamma1  = entrada(8);
% gamma2  = entrada(9);
% gamma3  = entrada(10);
% gamma4  = entrada(11);
% gamma5  = entrada(12);
% gamma6  = entrada(13);
% gamma7  = entrada(14);
% gamma8  = entrada(15);
%
% P_punto = salida(1);
% Q_punto = salida(2);
% R_punto = salida(3);


%% ENTRADA

P      = entrada(1);
Q      = entrada(2);
R      = entrada(3);
L      = entrada(4);
M      = entrada(5);
N      = entrada(6);
Iy     = entrada(7);
gamma1 = entrada(8);
gamma2 = entrada(9);
gamma3 = entrada(10);
gamma4 = entrada(11);
gamma5 = entrada(12);
gamma6 = entrada(13);
gamma7 = entrada(14);
gamma8 = entrada(15);


%% ACELERACIONES ANGULARES

salida = [gamma1*P*Q-gamma2*Q*R; gamma5*P*R-gamma6*(P^2-R^2); ...
          gamma7*P*Q-gamma1*Q*R]+[gamma3*L+gamma4*N; M/Iy; ...
          gamma4*L+gamma8*N];
```

- *"aceleraciones_lin.m"*

```
function salida = aceleraciones_lin(entrada)
% Funcion que calcula las aceleraciones lineales experimentadas por el UAV
% en ejes cuerpo.
%
% salida = aceleraciones_lin(entrada)
%
% Siendo:
%
% U       = entrada(1);
% V       = entrada(2);
% W       = entrada(3);
% P       = entrada(4);
% Q       = entrada(5);
% R       = entrada(6);
% Fx      = entrada(7);
% Fy      = entrada(8);
% Fz      = entrada(9);
% m       = entrada(10);
%
% U_punto = salida(1);
% V_punto = salida(2);
% W_punto = salida(3);


%% ENTRADA

U  = entrada(1);
V  = entrada(2);
W  = entrada(3);
P  = entrada(4);
Q  = entrada(5);
R  = entrada(6);
Fx = entrada(7);
Fy = entrada(8);
Fz = entrada(9);
m  = entrada(10);


%% SALIDA

salida = [R*V-Q*W; P*W-R*U; Q*U-P*V]+[Fx;Fy;Fz]/m;
```

- *"carga_telemetria.m"*

```
% Script que permite seleccionar y cargar un intervalo concreto de la
% telemetria de uno de los vuelos para ser utilizado como entrada en
% "Simulador_Telemetria".

clear all, close all
```

```
flag1 = 0;
flag2 = 0;
flag3 = 0;

while flag1==0

    texto1   = ['Pulse un numero entre 1 y 5 para seleccionar la ' ...
                'telemetria del vuelo correspondiente. A ' ...
                'continuacion, presione la tecla ENTER.\n'];
    entrada1 = input(texto1);

    if entrada1<1 || entrada1>5

        error1 = ['Debe pulsar una tecla entre 1 y 5 (1,2,3,4 o 5). ' ...
                  'Por favor, intentelo de nuevo.'];
        disp(error1);

    else

        flag1 = 1;

    end

end

while flag2==0

    texto2   = ['Inserte el numero de segundos que define la longitud ' ...
                'del tramo. A continuacion, presione la tecla ENTER.\n'];
    entrada2 = input(texto2);

    if entrada2<=0

        error2 = ['Debe insertar un numero mayor que 0. Por favor, ' ...
                  'intentelo de nuevo.'];
        disp(error2);

    else

        flag2 = 1;

    end

end

while flag3==0

    texto3   = ['Inserte el percentil que define el inicio del tramo ' ...
                'de telemetria a simular. A continuacion, presione la ' ...
                'tecla ENTER.\n'];
    entrada3 = input(texto3);

    if entrada3<0 || entrada3>=100

        error3 = ['Debe insertar un numero mayor o igual que 0 y ' ...
                  'menor que 100. Por favor, intentelo de nuevo.'];
        disp(error3);
```

```
    else

        flag3 = 1;

    end

end

celdatos = datos_telemetria(entrada1);
procesador_datos;

aux_gnss       = find(latitud(:,1)<=round(entrada3*latitud(end,1)...
                 /100,1),1,'last');
intervalo_gnss = [aux_gnss:find(latitud(:,1)*1e-3<=latitud(aux_gnss,1)...
                  *1e-3+entrada2,1,'last')];

aux_velocidades_ned       = find(v_norte(:,1)<=round(entrada3*...
                            v_norte(end,1)/100,1),1,'last');
intervalo_velocidades_ned = [aux_velocidades_ned:find(v_norte(:,1)*1e-3...
                             <=v_norte(aux_velocidades_ned,1)*1e-3+...
                             entrada2,1,'last')];

aux_actitud       = find(alabeo(:,1)<=round(entrada3*alabeo(end,1)...
                    /100,1),1,'last');
intervalo_actitud = [aux_actitud:find(alabeo(:,1)*1e-3<=alabeo...
                     (aux_actitud ,1)*1e-3+entrada2,1,'last')];

aux_velocidades_ang       = find(w_alabeo(:,1)<=round(entrada3*...
                            w_alabeo(end,1)/100,1),1,'last');
intervalo_velocidades_ang = [aux_velocidades_ang:find(w_alabeo(:,1)*...
                             1e-3<=w_alabeo(aux_velocidades_ang,1)*1e-3...
                             +entrada2,1,'last')];

aux_aceleraciones_lin       = find(a_x(:,1)<=round(entrada3*a_x(end,1)...
                              /100,1),1,'last');
intervalo_aceleraciones_lin = [aux_aceleraciones_lin:find(a_x(:,1)*1e-3...
                               <=a_x(aux_aceleraciones_lin ,1)*1e-3...
                               +entrada2,1,'last')];

aux_actuadores       = find(alerones_pwm(:,1)<=round(entrada3*...
                       alerones_pwm(end,1)/100,1),1,'last');
intervalo_actuadores = [aux_actuadores:find(alerones_pwm(:,1)*1e-3<=...
                        alerones_pwm(aux_actuadores,1)*1e-3+entrada2,...
                        1,'last')];

aux_viento       = find(viento_dir(:,1)<=round(entrada3*...
                   viento_dir(end,1)/100,1),1,'last');
intervalo_viento = [aux_viento:find(viento_dir(:,1)*1e-3<=viento_dir...
                    (aux_viento,1)*1e-3+entrada2,1,'last')];

% GNSS:

latitud  = timeseries(latitud(intervalo_gnss,2),...
           (latitud(intervalo_gnss,1)-latitud...
           (aux_gnss,1))*1e-3);
```

```
longitud = timeseries(longitud(intervalo_gnss,2),...
           (longitud(intervalo_gnss,1)-longitud...
           (aux_gnss,1))*1e-3);

altitud  = timeseries(altitud(intervalo_gnss,2),...
           (altitud(intervalo_gnss,1)-altitud...
           (aux_gnss,1))*1e-3);

% Actitud:

alabeo   = timeseries(alabeo(intervalo_actitud,2),(alabeo...
           (intervalo_actitud,1)-alabeo(aux_actitud,1))*1e-3);

cabeceo = timeseries(cabeceo(intervalo_actitud,2),(cabeceo...
           (intervalo_actitud,1)-cabeceo(aux_actitud,1))*1e-3);

guinada = timeseries(guinada(intervalo_actitud,2),(guinada...
           (intervalo_actitud,1)-guinada(aux_actitud,1))*1e-3);

% Velocidades NED:

v_norte = timeseries(v_norte(intervalo_velocidades_ned,2),...
           (v_norte(intervalo_velocidades_ned,1)-v_norte...
           (aux_velocidades_ned,1))*1e-3);

v_este  = timeseries(v_este(intervalo_velocidades_ned,2),...
           (v_este(intervalo_velocidades_ned,1)-v_este...
           (aux_velocidades_ned,1))*1e-3);

v_vert  = timeseries(v_vert(intervalo_velocidades_ned,2),...
           (v_vert(intervalo_velocidades_ned,1)-v_vert...
           (aux_velocidades_ned,1))*1e-3);

% Velocidades lineales:

v_x = [];
v_y = [];
v_z = [];

for k=1:size(intervalo_velocidades_ned,2)

    phi   = alabeo.Data(k);
    theta = cabeceo.Data(k);
    psi   = guinada.Data(k);

    % Matriz de rotacion de ejes NED a ejes cuerpo:

    R_nb  = [cos(theta)*cos(psi) cos(theta)*sin(psi) -sin(theta); ...
             sin(phi)*sin(theta)*cos(psi)-cos(phi)*sin(psi) sin(phi)*...
             sin(theta)*sin(psi)+cos(phi)*cos(psi) sin(phi)*cos(theta); ...
             cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi) cos(phi)*...
             sin(theta)*sin(psi)-sin(phi)*cos(psi) cos(phi)*cos(theta)];

    v_lin = R_nb*[v_norte.Data(k);v_este.Data(k);v_vert.Data(k)];

    v_x = [v_x;v_lin(1)];
```

```
    v_y = [v_y;v_lin(2)];
    v_z = [v_z;v_lin(3)];

end

v_x = timeseries(v_x,v_norte.Time);

v_y = timeseries(v_y,v_este.Time);

v_z = timeseries(v_z,v_vert.Time);

% Velocidades angulares:

w_alabeo  = timeseries(w_alabeo(intervalo_velocidades_ang,2),...
            (w_alabeo(intervalo_velocidades_ang,1)-...
            w_alabeo(aux_velocidades_ang,1))*1e-3);

w_cabeceo = timeseries(w_cabeceo(intervalo_velocidades_ang,2),...
            (w_cabeceo(intervalo_velocidades_ang,1)-...
            w_cabeceo(aux_velocidades_ang,1))*1e-3);

w_guinada = timeseries(w_guinada(intervalo_velocidades_ang,2),...
            (w_guinada(intervalo_velocidades_ang,1)-...
            w_guinada(aux_velocidades_ang,1))*1e-3);

% Aceleraciones lineales:

a_x = timeseries(a_x(intervalo_aceleraciones_lin,2),...
      (a_x(intervalo_aceleraciones_lin,1)-...
      a_x(aux_aceleraciones_lin,1))*1e-3);

a_y = timeseries(a_y(intervalo_aceleraciones_lin,2),...
      (a_y(intervalo_aceleraciones_lin,1)-...
      a_y(aux_aceleraciones_lin,1))*1e-3);

a_z = timeseries(a_z(intervalo_aceleraciones_lin,2),...
      (a_z(intervalo_aceleraciones_lin,1)-...
      a_z(aux_aceleraciones_lin,1))*1e-3);

% Actuadores:

alerones_pwm   = timeseries(alerones_pwm(intervalo_actuadores,2),...
                 (alerones_pwm(intervalo_actuadores,1)-alerones_pwm...
                 (aux_actuadores,1))*1e-3);

elevadores_pwm = timeseries(elevadores_pwm(intervalo_actuadores,2),...
                 (elevadores_pwm(intervalo_actuadores,1)-...
                 elevadores_pwm(aux_actuadores,1))*1e-3);

motor_pwm      = timeseries(motor_pwm(intervalo_actuadores,2),...
                 (motor_pwm(intervalo_actuadores,1)-...
                 motor_pwm(aux_actuadores,1))*1e-3);

% Viento

viento_mag_horz = timeseries(viento_mag_horz(intervalo_viento,2),...
                  (viento_mag_horz(intervalo_viento,1)-...
                  viento_mag_horz(aux_viento,1))*1e-3);
```

```
viento_mag_vert = timeseries(viento_mag_vert(intervalo_viento,2),...
                  (viento_mag_vert(intervalo_viento,1)-...
                  viento_mag_vert(aux_viento,1))*1e-3);

viento_dir      = timeseries(viento_dir(intervalo_viento,2),...
                  (viento_dir(intervalo_viento,1)-...
                   viento_dir(aux_viento,1))*1e-3);

viento_x = [];
viento_y = [];
viento_z = [];

indice = min([size(intervalo_actitud,2) size(intervalo_viento,2)]);

for k=1:size(indice,2)

    phi   = alabeo.Data(k);
    theta = cabeceo.Data(k);
    psi   = guinada.Data(k);

    % Matriz de rotacion de ejes NED a ejes cuerpo:

    R_nb  = [cos(theta)*cos(psi) cos(theta)*sin(psi) -sin(theta); ...
             sin(phi)*sin(theta)*cos(psi)-cos(phi)*sin(psi) sin(phi)*...
             sin(theta)*sin(psi)+cos(phi)*cos(psi) sin(phi)*cos(theta); ...
             cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi) cos(phi)*...
             sin(theta)*sin(psi)-sin(phi)*cos(psi) cos(phi)*cos(theta)];

    viento_NE = R_nb*([-viento_mag_horz.Data(k)*...
                      [cos(viento_dir.Data(k))*cos(psi);...
                       cos(viento_dir.Data(k))*sin(psi)];...
                       viento_mag_vert.Data(k)]);

    viento_x  = [viento_x; viento_NE(1)];
    viento_y  = [viento_y; viento_NE(2)];
    viento_z  = [viento_z; viento_NE(3)];


end

viento_x = timeseries(viento_x,viento_dir.Time);
viento_y = timeseries(viento_y,viento_dir.Time);
viento_z = timeseries(viento_z,viento_dir.Time);

% Valores iniciales:

coordenadas_geodeticas_iniciales = [0 latitud.Data(1) ...
                                    longitud.Data(1) ...
                                    altitud.Data(1)];

posicion_NED_inicial = [0 0 0 0];

actitud_inicial      = [0 alabeo.Data(1) cabeceo.Data(1) guinada.Data(1)];

phi   = alabeo.Data(1);
theta = cabeceo.Data(1);
psi   = guinada.Data(1);
```

```
R_nb  = [cos(theta)*cos(psi) cos(theta)*sin(psi) -sin(theta); ...
         sin(phi)*sin(theta)*cos(psi)-cos(phi)*sin(psi) sin(phi)*...
         sin(theta)*sin(psi)+cos(phi)*cos(psi) sin(phi)*cos(theta); ...
         cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi) cos(phi)*...
         sin(theta)*sin(psi)-sin(phi)*cos(psi) cos(phi)*cos(theta)];

velocidades_lin_inicial    =  [0 (R_nb*[v_norte.Data(1);...
                               v_este.Data(1);v_vert.Data(1)]).'];

velocidades_ang_inicial    =  [0 w_alabeo.Data(1) ...
                               w_cabeceo.Data(1) w_guinada.Data(1)];

aceleraciones_lin_inicial  =  [0 a_x.Data(1) a_y.Data(1) a_z.Data(1)];
```

- *"controlador_h.m"*

```
function salida = controlador_h(entrada)
% Funcion que calcula los parametros del controlador de h.
%
% salida = controlador_h(entrada)
%
% Siendo:
%
% Ua          = entrada(1);
% Va          = entrada(2);
% Wa          = entrada(3);
% theta       = entrada(4);
% theta_ref   = entrada(5);
% rho         = entrada(6);
% c           = entrada(7);
% Salar       = entrada(8);
% Iy          = entrada(9);
% C_malpha    = entrada(10);
% C_mdelta_e  = entrada(11);
%
% Kp_h        = salida(1);
% Ki_h        = salida(2);


%% ENTRADA

Ua          = entrada(1);
Va          = entrada(2);
Wa          = entrada(3);
theta       = entrada(4);
theta_ref   = entrada(5);
rho         = entrada(6);
c           = entrada(7);
Salar       = entrada(8);
Iy          = entrada(9);
C_malpha    = entrada(10);
C_mdelta_e  = entrada(11);
```

```
%% COEFICIENTES

Uinf       = sqrt(Ua^2+Va^2+Wa^2);
delta_emax = 5*pi/180;
e_thetamax = 10*pi/180;
dseta_h    = 1;
s          = 40;

% Singularidad en theta nulo:

if abs(theta)>1

    Ktheta_dc = theta/theta_ref;

else

    Ktheta_dc = 1;

end


%% CALCULO

a_theta2 = -0.5*rho*Uinf^2*c*Salar*C_malpha/Iy;
a_theta3 = 0.5*rho*Uinf^2*c*Salar*C_mdelta_e/Iy;
wn_h     = (sqrt(a_theta2+(delta_emax*abs(a_theta3)/e_thetamax)))/s;
Kp_h     = 2*dseta_h*wn_h/(Ktheta_dc*Uinf+1e-16);
Ki_h     = wn_h^2/(Ktheta_dc*Uinf+1e-16);


%% SALIDA

salida = [Kp_h;Ki_h];
```

- *"controlador_phi.m"*

```
function salida = controlador_phi(entrada)
% Funcion que calcula los parametros del controlador de phi.
%
% salida = controlador_phi(entrada)
%
% Siendo:
%
% Ua         = entrada(1);
% Va         = entrada(2);
% Wa         = entrada(3);
% P          = entrada(4);
% Q          = entrada(5);
% R          = entrada(6);
% phi        = entrada(7);
% theta      = entrada(8);
% rho        = entrada(9);
% b          = entrada(10);
% Salar      = entrada(11);
```

```
% gamma1     = entrada(12);
% gamma2     = entrada(13);
% C_p0       = entrada(14);
% C_pbeta    = entrada(15);
% C_pp       = entrada(16);
% C_pr       = entrada(17);
% C_pdelta_a = entrada(18);
%
% a_phi2     = salida(1);
% d_phi1     = salida(2);
% d_phi2_    = salida(3);
% Kp_phi     = salida(4);
% Ki_phi     = salida(5);
% Kd_phi     = salida(6);


%% ENTRADA

Ua         = entrada(1);
Va         = entrada(2);
Wa         = entrada(3);
P          = entrada(4);
Q          = entrada(5);
R          = entrada(6);
phi        = entrada(7);
theta      = entrada(8);
rho        = entrada(9);
b          = entrada(10);
Salar      = entrada(11);
gamma1     = entrada(12);
gamma2     = entrada(13);
C_p0       = entrada(14);
C_pbeta    = entrada(15);
C_pp       = entrada(16);
C_pr       = entrada(17);
C_pdelta_a = entrada(18);


%% COEFICIENTES

Uinf       = sqrt(Ua^2+Va^2+Wa^2);
beta       = asin(Va/(Uinf+1e-16));
delta_amax = 5*pi/180;
e_phimax   = 10*pi/180;
dseta_phi  = 1;


%% CALCULO

a_phi1  = -0.25*rho*Uinf*Salar*b^2*C_pp;
a_phi2  = 0.5*rho*Uinf^2*Salar*b*C_pdelta_a;
d_phi1  = Q*sin(phi)*tan(theta)+R*cos(phi)*tan(theta);
d_phi2_ = gamma1*P*Q-gamma2*Q*R+0.5*rho*Uinf^2*Salar*b*(C_p0+C_pbeta*...
          beta-0.5*C_pp*b*d_phi1/(Uinf+1e-16)+0.5*C_pr*b*R/(Uinf+1e-16));
Kp_phi  = delta_amax*sign(a_phi2)/e_phimax;
Ki_phi  = 0.5;
wn_phi  = sqrt(Kp_phi*a_phi2);
Kd_phi  = (2*dseta_phi*wn_phi-a_phi1)/a_phi2;
```

```
%% SALIDA

salida = [a_phi2;d_phi1;d_phi2_;Kp_phi;Ki_phi;Kd_phi];
```

- *"controlador_psi.m"*

```
function salida = controlador_psi(entrada)
% Funcion que calcula los parametros del controlador de psi.
%
% salida = controlador_psi(entrada)
%
% Siendo:
%
% Ua         = entrada(1);
% Va         = entrada(2);
% Wa         = entrada(3);
% rho        = entrada(4);
% g          = entrada(5);
% b          = entrada(6);
% Salar      = entrada(7);
% C_pdelta_a = entrada(8);
%
% Kp_psi     = salida(1);
% Ki_psi     = salida(2);


%% ENTRADA

Ua         = entrada(1);
Va         = entrada(2);
Wa         = entrada(3);
rho        = entrada(4);
g          = entrada(5);
b          = entrada(6);
Salar      = entrada(7);
C_pdelta_a = entrada(8);


%% COEFICIENTES

Uinf       = sqrt(Ua^2+Va^2+Wa^2);
delta_amax = 5*pi/180;
e_phimax   = 10*pi/180;
dseta_psi  = 1;
s          = 40;


%% CALCULO

a_phi2 = 0.5*rho*Uinf^2*Salar*b*C_pdelta_a;
Kp_phi = delta_amax*sign(a_phi2)/e_phimax;
wn_psi = sqrt(Kp_phi*a_phi2)/s;
Kp_psi = 2*dseta_psi*wn_psi*Uinf/g;
Ki_psi = wn_psi^2*Uinf/g;
```

```
%% SALIDA

salida = [Kp_psi;Ki_psi];
```

- *"controlador_theta.m"*

```
function salida = controlador_theta(entrada)
% Funcion que calcula los parametros del controlador de theta.
%
% salida = controlador_theta(entrada)
%
% Siendo:
%
% Ua         = entrada(1);
% Va         = entrada(2);
% Wa         = entrada(3);
% rho        = entrada(4);
% c          = entrada(5);
% Salar      = entrada(6);
% Iy         = entrada(7);
% C_malpha   = entrada(8);
% C_mq       = entrada(9);
% C_mdelta_e = entrada(10);
%
% Kp_theta   = salida(1);
% Kd_theta   = salida(2);


%% ENTRADA

Ua         = entrada(1);
Va         = entrada(2);
Wa         = entrada(3);
rho        = entrada(4);
c          = entrada(5);
Salar      = entrada(6);
Iy         = entrada(7);
C_malpha   = entrada(8);
C_mq       = entrada(9);
C_mdelta_e = entrada(10);


%% COEFICIENTES

Uinf        = sqrt(Ua^2+Va^2+Wa^2);
delta_emax  = 5*pi/180;
e_thetamax  = 10*pi/180;
dseta_theta = 1;


%% CALCULO

a_theta1 = -0.25*rho*Uinf*c^2*Salar*C_mq/Iy;
```

```
a_theta2 = -0.5*rho*Uinf^2*c*Salar*C_malpha/Iy;
a_theta3 = 0.5*rho*Uinf^2*c*Salar*C_mdelta_e/Iy;
Kp_theta = delta_emax*sign(a_theta3)/e_thetamax;
wn_theta = sqrt(a_theta2+(delta_emax*abs(a_theta3)/e_thetamax));
Kd_theta = (2*dseta_theta*wn_theta-a_theta1)/a_theta3;


%% SALIDA

salida = [Kp_theta;Kd_theta];
```

- *"controlador_uinf_delta_t.m"*

```
function salida = controlador_uinf_delta_t(entrada)
% Funcion que calcula los parametros del controlador de Uinf (delta t).
%
% salida = controlador_uinf_delta_t(entrada)
%
% Siendo:
%
% Uinf_ref       = entrada(1);
% alpha_trim     = entrada(2);
% beta_trim      = entrada(3);
% delta_e_trim   = entrada(4);
% rho            = entrada(5);
% C_Dalpha       = entrada(6);
% m              = entrada(7);
% Salar          = entrada(8);
% coefprop_motor = entrada(9);
% coefprop_U     = entrada(10);
% C_D0           = entrada(11);
% C_Ddelta_e     = entrada(12);
%
% Kp_v           = salida(1);
% Ki_v           = salida(2);


%% ENTRADA

Uinf_ref       = entrada(1);
alpha_trim     = entrada(2);
beta_trim      = entrada(3);
delta_e_trim   = entrada(4);
rho            = entrada(5);
C_Dalpha       = entrada(6);
m              = entrada(7);
Salar          = entrada(8);
coefprop_motor = entrada(9);
coefprop_U     = entrada(10);
C_D0           = entrada(11);
C_Ddelta_e     = entrada(12);
```

```
%% COEFICIENTES

dseta_v = 1;
wn_v    = 1;


%% CALCULO

a_v1 = rho*Uinf_ref*Salar*(C_D0+(C_Dalpha-C_D0)*alpha_trim+C_Ddelta_e*...
       delta_e_trim)/m-2*coefprop_U*cos(alpha_trim)*cos(beta_trim)*...
       Uinf_ref/m;
a_v2 = coefprop_motor;
Kp_v = (2*dseta_v*wn_v-a_v1)/a_v2;
Ki_v = wn_v^2/a_v2;


%% SALIDA

salida = [Kp_v;Ki_v];
```

- *"controlador_uinf_theta.m"*

```
function salida = controlador_uinf_theta(entrada)
% Funcion que calcula los parametros del controlador de Uinf (theta).
%
% salida = controlador_uinf_theta(entrada)
%
% Siendo:
%
% Ua           = entrada(1);
% Va           = entrada(2);
% Wa           = entrada(3);
% Uinf_ref     = entrada(4);
% alpha_trim   = entrada(5);
% beta_trim    = entrada(6);
% delta_e_trim = entrada(7);
% theta        = entrada(8);
% theta_ref    = entrada(9);
% rho          = entrada(10);
% g            = entrada(11);
% C_Dalpha     = entrada(12);
% m            = entrada(13);
% c            = entrada(14);
% Salar        = entrada(15);
% Iy           = entrada(16);
% coefprop_U   = entrada(17);
% C_D0         = entrada(18);
% C_Ddelta_e   = entrada(19);
% C_malpha     = entrada(20);
% C_mdelta_e   = entrada(21);
%
% Kp_v2        = salida(1);
% Ki_v2        = salida(2);
```

```
%% ENTRADA

Ua           = entrada(1);
Va           = entrada(2);
Wa           = entrada(3);
Uinf_ref     = entrada(4);
alpha_trim   = entrada(5);
beta_trim    = entrada(6);
delta_e_trim = entrada(7);
theta        = entrada(8);
theta_ref    = entrada(9);
rho          = entrada(10);
g            = entrada(11);
C_Dalpha     = entrada(12);
m            = entrada(13);
c            = entrada(14);
Salar        = entrada(15);
Iy           = entrada(16);
coefprop_U   = entrada(17);
C_D0         = entrada(18);
C_Ddelta_e   = entrada(19);
C_malpha     = entrada(20);
C_mdelta_e   = entrada(21);


%% COEFICIENTES

Uinf       = sqrt(Ua^2+Va^2+Wa^2);
delta_emax = 30*pi/180;
e_thetamax = 30*pi/180;
dseta_v2   = 1;
s          = 20;

% Singularidad en theta nulo:

if abs(theta)>1

    Ktheta_dc = theta/theta_ref;

else

    Ktheta_dc = 1;

end


%% CALCULO

a_v1     = rho*Uinf_ref*Salar*(C_D0+(C_Dalpha-C_D0)*alpha_trim+...
           C_Ddelta_e*delta_e_trim)/m-2*coefprop_U*cos(alpha_trim)*...
           cos(beta_trim)*Uinf_ref/m;
a_theta2 = -0.5*rho*Uinf^2*c*Salar*C_malpha/Iy;
a_theta3 = 0.5*rho*Uinf^2*c*Salar*C_mdelta_e/Iy;
wn_theta = sqrt(a_theta2+(delta_emax*abs(a_theta3)/e_thetamax));
wn_v2    = wn_theta/s;
Kp_v2    = (a_v1-2*dseta_v2*wn_v2)/(Ktheta_dc*g);
Ki_v2    = -wn_v2^2/(Ktheta_dc*g);
```

```
%% SALIDA

salida = [Kp_v2;Ki_v2];
```

- *"eom.m"*

```
function salida = eom(entrada)
% Funcion que calcula las ecuaciones del movimiento del UAV, utilizada para
% hallar el punto de equilibrio del mismo.


%% ENTRADA

U       = entrada(1);
V       = entrada(2);
W       = entrada(3);
P       = entrada(4);
Q       = entrada(5);
R       = entrada(6);
phi     = entrada(7);
theta   = entrada(8);
psi     = entrada(9);
delta_a = entrada(10);
delta_e = entrada(11);
delta_t = entrada(12);


%% DATOS

x8;
rho = 1.225;
g   = 9.807;


%% CALCULO

Uinf  = sqrt(U^2+V^2+W^2);
alpha = atan(W/(U+1e-16));
beta  = asin(V/(Uinf+1e-16));

sigma = sigmoide(alpha);

entrada_x8_aero  = [alpha beta Uinf P Q R delta_a delta_e sigma m b c ...
                    Salar AR e C_L0 C_Lalpha C_Lq C_Ldelta_e C_D0 ...
                    C_Dbeta1 C_Dbeta2 C_Dq C_Ddelta_e C_Y0 C_Ybeta C_Yp ...
                    C_Yr C_Ydelta_a C_l0 C_lbeta C_lp C_lr C_ldelta_a ...
                    C_m0 C_malpha C_mp C_mq C_mdelta_e C_n0 C_nbeta ...
                    C_np C_nr C_ndelta_a];

salida_x8_aero = x8_aero(entrada_x8_aero);
```

```
C_L = salida_x8_aero(1);
C_D = salida_x8_aero(2);
C_Y = salida_x8_aero(3);
C_l = salida_x8_aero(4);
C_m = salida_x8_aero(5);
C_n = salida_x8_aero(6);

entrada_fuerzas  = [alpha beta Uinf C_L C_D C_Y phi theta psi delta_t ...
                    rho g m Salar];
entrada_momentos = [Uinf C_l C_m C_n rho b c Salar];
Fuerzas          = fuerzas(entrada_fuerzas);
Momentos         = momentos(entrada_momentos);
FuerzasX         = Fuerzas(1);
FuerzasY         = Fuerzas(2);
FuerzasZ         = Fuerzas(3);
MomentosX        = Momentos(1);
MomentosY        = Momentos(2);
MomentosZ        = Momentos(3);

entrada_aceleraciones_lin = [U V W P Q R FuerzasX FuerzasY FuerzasZ m];
entrada_aceleraciones_ang = [P Q R MomentosX MomentosY MomentosZ Iy ...
                             gamma1 gamma2 gamma3 gamma4 gamma5 gamma6 ...
                             gamma7 gamma8];
Aceleraciones_lin         = aceleraciones_lin(entrada_aceleraciones_lin);
Aceleraciones_ang         = aceleraciones_ang(entrada_aceleraciones_ang);

entrada_velocidades_lin = [U V W phi theta psi];
entrada_velocidades_ang = [P Q R phi theta];
Velocidades_lin         = velocidades_lin(entrada_velocidades_lin);
Velocidades_ang         = velocidades_ang(entrada_velocidades_ang);

posd_punto  = Velocidades_lin(3);
U_punto     = Aceleraciones_lin(1);
V_punto     = Aceleraciones_lin(2);
W_punto     = Aceleraciones_lin(3);
phi_punto   = Velocidades_ang(1);
theta_punto = Velocidades_ang(2);
psi_punto   = Velocidades_ang(3);
P_punto     = Aceleraciones_ang(1);
Q_punto     = Aceleraciones_ang(2);
R_punto     = Aceleraciones_ang(3);


%% SALIDA

salida = [posd_punto;FuerzasX;FuerzasY;FuerzasZ;MomentosX;MomentosY;...
          MomentosZ;U_punto;V_punto;W_punto;phi_punto;theta_punto;...
          psi_punto^2];
```

- *"espacio_estados.m"*

```
% Script que construye los espacios de estados lateral y longitudinal del
% UAV para un punto de equilibrio en vuelo de crucero horizontal.
```

```
%% DATOS:

x8;
rho = 1.225;
g   = 9.807;


%% PUNTO DE EQUILIBRIO (CRUZERO HORIZONTAL):

U_equilibrio       = 14.9346;
V_equilibrio       = 8.04152e-5;
W_equilibrio       = 1.26060;
P_equilibrio       = -1.57606e-20;
Q_equilibrio       = -1.67480e-17;
R_equilibrio       = -1.29450e-21;
phi_equilibrio     = 3.34936e-6;
theta_equilibrio   = 0.0842084;
psi_equilibrio     = 0;
delta_a_equilibrio = 1.37414e-6;
delta_e_equilibrio = -0.00669962;
delta_t_equilibrio = 1.21617;

Uinf_equilibrio  = sqrt(U_equilibrio^2+V_equilibrio^2+W_equilibrio^2);
alpha_equilibrio = atan(W_equilibrio/U_equilibrio);
beta_equilibrio  = asin(V_equilibrio/Uinf_equilibrio);


%% COEFICIENTES AERODINAMICOS CRUZADOS:

% Matriz de rotacion de ejes viento a ejes cuerpo:

R_wb_equilibrio = [cos(theta_equilibrio)   0   -sin(theta_equilibrio); ...
                                0          1              0          ; ...
                   sin(theta_equilibrio)   0    cos(theta_equilibrio)];

sigma = sigmoide(alpha_equilibrio);

entrada_x8_aero  = [0 Uinf_equilibrio 0 0 0 0 0 ...
                    sigma m b c Salar AR e C_L0 C_Lalpha C_Lq ...
                    C_Ldelta_e C_D0 C_Dbeta1 C_Dbeta2 C_Dq C_Ddelta_e ...
                    C_Y0 C_Ybeta C_Yp C_Yr C_Ydelta_a C_l0 C_lbeta C_lp
...
                    C_lr C_ldelta_a C_m0 C_malpha C_mp C_mq C_mdelta_e ...
                    C_n0 C_nbeta C_np C_nr C_ndelta_a];

salida1 = x8_aero([0 entrada_x8_aero]);
C_F0    = -R_wb_equilibrio*[salida1(2);0;salida1(1)];
C_X0    = C_F0(1);
C_Z0    = C_F0(3);
C_m0ss  = salida1(5);

salida2    = x8_aero([alpha_equilibrio entrada_x8_aero]);
C_Falpha   = -R_wb_equilibrio*[salida2(2);0;salida2(1)];
C_Xalpha   = C_Falpha(1);
C_Zalpha   = C_Falpha(3);
C_malphass = salida2(5);
```

```
C_Fq = -R_wb_equilibrio*[C_Dq;0;C_Lq];
C_Xq = C_Fq(1);
C_Zq = C_Fq(3);

C_Fdelta_e = -R_wb_equilibrio*[C_Ddelta_e;0;C_Ldelta_e];
C_Xdelta_e = C_Fdelta_e(1);
C_Zdelta_e = C_Fdelta_e(3);


%% COEFICIENTES DEL ESPACIO DE ESTADOS LATERAL:

Yv       = rho*Salar*b*V_equilibrio*(C_Yp*P_equilibrio+C_Yr*...
           R_equilibrio)/4*m*Uinf_equilibrio+rho*Salar*V_equilibrio*...
           (C_Y0+C_Ybeta*beta_equilibrio+C_Ydelta_a*delta_a_equilibrio)...
           /m+rho*Salar*C_Ybeta*sqrt(U_equilibrio^2+W_equilibrio^2)/2*m;
Yp       = W_equilibrio+rho*Uinf_equilibrio*Salar*b*C_Yp/4*m;
Yr       = -U_equilibrio+rho*Uinf_equilibrio*Salar*b*C_Yr/4*m;
Ydelta_a = rho*Uinf_equilibrio^2*Salar*C_Ydelta_a/2*m;

Lv       = rho*Salar*b^2*V_equilibrio*(C_pp*P_equilibrio+C_pr*...
           R_equilibrio)/4*Uinf_equilibrio+rho*Salar*b*V_equilibrio*...
           (C_p0+C_pbeta*beta_equilibrio+C_pdelta_a*delta_a_equilibrio)...
           +rho*Salar*b*C_pbeta*sqrt(U_equilibrio^2+W_equilibrio^2)/2;
Lp       = gamma1*Q_equilibrio+rho*Uinf_equilibrio*Salar*b^2*C_pp/4;
Lr       = -gamma2*Q_equilibrio+rho*Uinf_equilibrio*Salar*b^2*C_pr/4;
Ldelta_a = rho*Uinf_equilibrio^2*Salar*b*C_pdelta_a/2;

Nv       = rho*Salar*b^2*V_equilibrio*(C_rp*P_equilibrio+C_rr*...
           R_equilibrio)/4*Uinf_equilibrio+rho*Salar*b*V_equilibrio*...
           (C_r0+C_rbeta*beta_equilibrio+C_rdelta_a*delta_a_equilibrio)...
           +rho*Salar*b*C_rbeta*sqrt(U_equilibrio^2+W_equilibrio^2)/2;
Np       = gamma7*Q_equilibrio+rho*Uinf_equilibrio*Salar*b^2*C_rp/4;
Nr       = -gamma1*Q_equilibrio+rho*Uinf_equilibrio*Salar*b^2*C_rr/4;
Ndelta_a = rho*Uinf_equilibrio^2*Salar*b*C_rdelta_a/2;


%% COEFICIENTES DEL ESPACIO DE ESTADOS LONGITUDINAL:

Xu       = U_equilibrio*rho*Salar*(C_X0+C_Xalpha*alpha_equilibrio+...
           C_Xdelta_e*delta_e_equilibrio)/m-rho*Salar*W_equilibrio*...
           C_Xalpha/2*m+rho*Salar*c*C_Xq*U_equilibrio*Q_equilibrio/...
           4*m*Uinf_equilibrio+2*coefprop(2)*U_equilibrio/m;
Xw       = -Q_equilibrio+W_equilibrio*rho*Salar*(C_X0+C_Xalpha*...
           alpha_equilibrio+C_Xdelta_e*delta_e_equilibrio)/m+rho*Salar*...
           c*C_Xq*W_equilibrio*Q_equilibrio/4*m*Uinf_equilibrio+rho*...
           Salar*U_equilibrio*C_Xalpha/2*m;
Xq       = -W_equilibrio+rho*Uinf_equilibrio*Salar*C_Xq*c/4*m;
Xdelta_e = rho*Uinf_equilibrio^2*Salar*C_Xdelta_e/2*m;
Xdelta_t = 1/m;

Zu       = Q_equilibrio+U_equilibrio*rho*Salar*(C_Z0+C_Zalpha*...
           alpha_equilibrio+C_Zdelta_e*delta_e_equilibrio)/m-rho*Salar*...
           C_Zalpha*W_equilibrio/2*m+U_equilibrio*rho*Salar*C_Zq*c*...
           Q_equilibrio/4*m*Uinf_equilibrio;
Zw       = W_equilibrio*rho*Salar*(C_Z0+C_Zalpha*alpha_equilibrio+...
           C_Zdelta_e*delta_e_equilibrio)/m+rho*Salar*C_Zalpha*...
           U_equilibrio/2*m+rho*W_equilibrio*Salar*c*C_Zq*Q_equilibrio/...
           4*m*Uinf_equilibrio;
```

```
Zq       = U_equilibrio+rho*Uinf_equilibrio*Salar*C_Zq*c/4*m;
Zdelta_e = rho*Uinf_equilibrio^2*Salar*C_Zdelta_e/2*m;

Mu       = U_equilibrio*rho*Salar*c*(C_m0ss+C_malphass*...
           alpha_equilibrio+C_mdelta_e*delta_e_equilibrio)/Iy-rho*Salar*...
           c*C_malphass*W_equilibrio/2*Iy+rho*Salar*c^2*C_mq*...
           Q_equilibrio*U_equilibrio/4*Iy*Uinf_equilibrio;
Mw       = W_equilibrio*rho*Salar*c*(C_m0ss+C_malphass*...
           alpha_equilibrio+C_mdelta_e*delta_e_equilibrio)/Iy+rho*Salar*...
           c*C_malphass*U_equilibrio/2*Iy+rho*Salar*c^2*C_mq*...
           Q_equilibrio*W_equilibrio/4*Iy*Uinf_equilibrio;
Mq       = rho*Uinf_equilibrio*Salar*c^2*C_mq/4*Iy;
Mdelta_e = rho*Uinf_equilibrio^2*Salar*c*C_mdelta_e/2*Iy;


%% ESPACIO DE ESTADOS LATERAL:

A_lat = [Yv Yp Yr g*cos(theta_equilibrio) 0; Lv Lp Lr 0 0; Nv Np Nr 0 ...
         0; 0 1 cos(phi_equilibrio)*tan(theta_equilibrio) Q_equilibrio*...
         cos(phi_equilibrio)*tan(theta_equilibrio)-R_equilibrio*...
         sin(phi_equilibrio)*tan(theta_equilibrio) 0; 0 0 ...
         cos(phi_equilibrio)*sec(theta_equilibrio) P_equilibrio*...
         cos(phi_equilibrio)*sec(theta_equilibrio)-R_equilibrio*...
         sin(phi_equilibrio)*sec(theta_equilibrio) 0];
B_lat = [Ydelta_a; Ldelta_a; Ndelta_a; 0; 0];


%% ESPACIO DE ESTADOS LONGITUDINAL:

A_long = [Xu Xw Xq -g*cos(theta_equilibrio) 0; Zu Zw Zq ...
          -g*sin(theta_equilibrio) 0; Mu Mw Mq 0 0; 0 0 1 0 0; ...
          sin(theta_equilibrio) -cos(theta_equilibrio) 0 U_equilibrio*...
          cos(theta_equilibrio)+W_equilibrio*sin(theta_equilibrio) 0];
B_long = [Xdelta_e Xdelta_t; Zdelta_e 0; Mdelta_e 0; 0 0; 0 0];


%% ESPACIO DE ESTADOS:

estado_lat    = {'V' 'P' 'R' 'phi' 'psi'};
actuador_lat  = {'delta_a'};
respuesta_lat = {'V punto' 'P punto' 'R punto' 'phi punto' 'psi punto'};
sys_lat       = ss(A_lat,B_lat,eye(5),zeros(5,1),'statename',estado_lat,...
                   'inputname',actuador_lat,'outputname',respuesta_lat);

Polos    = eig(A_long);
Polos(3) = -Polos(3);
Polos(4) = -Polos(4);
K        = place(A_long,B_long,Polos);

estado_long    = {'U' 'W' 'Q' 'theta' 'h'};
actuador_long  = {'delta_e' 'delta_t'};
respuesta_long = {'U punto' 'W punto' 'Q punto' 'theta punto' 'h punto'};
sys_long       = ss(A_long-B_long*K,B_long,eye(5),zeros(5,2),...
                    'statename',estado_long,'inputname',actuador_long,...
                    'outputname',respuesta_long);
```

- *"fuerzas.m"*

```
function salida = fuerzas(entrada)
% Funcion que calcula las fuerzas ejercidas sobre el UAV en ejes cuerpo.
%
% salida = fuerzas(entrada)
%
% Siendo:
%
% alpha   = entrada(1);
% beta    = entrada(2);
% Uinf    = entrada(3);
% C_L     = entrada(4);
% C_D     = entrada(5);
% C_Y     = entrada(6);
% phi     = entrada(7);
% theta   = entrada(8);
% psi     = entrada(9);
% delta_t = entrada(10);
% rho     = entrada(11);
% g       = entrada(12);
% m       = entrada(13);
% Salar   = entrada(14);
%
% Fx      = salida(1);
% Fy      = salida(2);
% Fz      = salida(3);


%% ENTRADA

alpha   = entrada(1);
beta    = entrada(2);
Uinf    = entrada(3);
C_L     = entrada(4);
C_D     = entrada(5);
C_Y     = entrada(6);
phi     = entrada(7);
theta   = entrada(8);
psi     = entrada(9);
delta_t = entrada(10);
rho     = entrada(11);
g       = entrada(12);
m       = entrada(13);
Salar   = entrada(14);


%% CALCULO

% Matriz de rotacion de ejes viento a ejes cuerpo:

R_wb = [cos(alpha)*cos(beta)  -cos(alpha)*sin(beta)  -sin(alpha) ; ...
            sin(beta)                cos(beta)           0       ; ...
        sin(alpha)*cos(beta)  -sin(alpha)*sin(beta)   cos(alpha)];
```

```
% Matriz de rotacion de ejes NED a ejes cuerpo:

R_nb = [cos(theta)*cos(psi) cos(theta)*sin(psi) -sin(theta); ...
        sin(phi)*sin(theta)*cos(psi)-cos(phi)*sin(psi) sin(phi)*...
        sin(theta)*sin(psi)+cos(phi)*cos(psi) sin(phi)*cos(theta); ...
        cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi) cos(phi)*...
        sin(theta)*sin(psi)-sin(phi)*cos(psi) cos(phi)*cos(theta)];


%% FUERZAS

salida = -0.5*rho*Uinf^2*Salar*R_wb*[C_D;0;C_L]+0.5*rho*Uinf^2*Salar*...
         [0;C_Y;0]+R_nb*[0;0;m*g]+[delta_t;0;0];
```

- *"geodeticas_ned_mat.m"*

```
function salida = geodeticas_ned_mat(entrada)
% Funcion que transforma coordenadas geodeticas a NED.
%
% salida = geodeticas_ned_mat(entrada)
%
% Siendo:
%
% [latitud0, longitud0, altitud0, coordenadas] = entrada;
%
% donde coordenadas es un vector formado por una sucesion de coordenadas
% geodeticas en el formato lineal [latitud, longitud, altitud].
%
% N = salida(1);
% E = salida(2);
% D = salida(3);


%% ENTRADA

latitud0  = entrada(1);
longitud0 = entrada(2);
altitud0  = entrada(3);

latitud  = [];
longitud = [];
altitud  = [];

for k=4:3:length(entrada)-2

    latitud   = [latitud; entrada(k)];
    longitud  = [longitud; entrada(k+1)];
    altitud   = [altitud; entrada(k+2)];

end
```

```
%% CALCULO

WGS84   = referenceEllipsoid('wgs84');
[N,E,D] = geodetic2ned(latitud,longitud,altitud,latitud0,longitud0,...
          altitud0,WGS84);


%% SALIDA

salida = reshape([N,E,D].',[],1);
```

- *"limitador_psi.m"*

```
function salida = limitador_psi(entrada)
% Funcion que procesa el calculo del error en el controlador de psi.
%
% salida = limitador_psi(entrada)
%
% Siendo:
%
% psi        = entrada(1);
% psi_ref    = entrada(2);
% psi_inicio = entrada(3);
%
% error      = salida;


%% ENTRADA

psi        = entrada(1);
psi_ref    = entrada(2);
psi_inicio = entrada(3);


%% CALCULO

vueltas_psi        = floor(psi/(2*pi));
vueltas_psi_ref    = floor(psi_ref/(2*pi));
vueltas_psi_inicio = floor(psi_inicio/(2*pi));
psi                = psi-2*pi*vueltas_psi;
psi_ref            = psi_ref-2*pi*vueltas_psi_ref;
psi_inicio         = psi_inicio-2*pi*vueltas_psi_inicio;
error              = psi_ref-psi;
error_inicio       = psi_ref-psi_inicio;
signo              = sign(error);
signo_inicio       = sign(error_inicio);
error_abs          = abs(error);
error_inicio_abs   = abs(error_inicio);

if error_inicio_abs>pi

    if error_abs>pi

        salida = -signo_inicio*abs(error-signo_inicio*2*pi);
        return;
```

```
    else

        salida = error;
        return;

    end

else

    if error_abs>=1.5*pi

        salida = -signo*abs(error-signo*2*pi);
        return;

    else

        salida = error;
        return;

    end

end
```

- *"momentos.m"*

```
function salida = momentos(entrada)
% Funcion que calcula los momentos ejercidos sobre el UAV en ejes cuerpo.
%
% salida = momentos(entrada)
%
% Siendo:
%
% Uinf    = entrada(1);
% C_l     = entrada(2);
% C_m     = entrada(3);
% C_n     = entrada(4);
% rho     = entrada(5);
% b       = entrada(6);
% c       = entrada(7);
% Salar   = entrada(8);
%
% L       = salida(1);
% M       = salida(2);
% N       = salida(3);


%% ENTRADA

Uinf    = entrada(1);
C_l     = entrada(2);
C_m     = entrada(3);
C_n     = entrada(4);
```

```
rho     = entrada(5);
b       = entrada(6);
c       = entrada(7);
Salar   = entrada(8);


%% MOMENTOS

salida = 0.5*rho*Uinf^2*Salar*[b*C_l;c*C_m;b*C_n];
```

- *"ned_geodeticas.m"*

```
function salida = ned_geodeticas(entrada)
% Funcion que transforma coordenadas NED a geodeticas.
%
% salida = ned_geodeticas(entrada)
%
% Siendo:
%
% latitud0  = entrada(1);
% longitud0 = entrada(2);
% altitud0  = entrada(3);
% X         = entrada(4);
% Y         = entrada(5);
% Z         = entrada(6);
%
% longitud  = salida(1);
% latitud   = salida(2);
% altitud   = salida(3);


%% ENTRADA

latitud0  = entrada(1);
longitud0 = entrada(2);
altitud0  = entrada(3);
X         = entrada(4);
Y         = entrada(5);
Z         = entrada(6);


%% CALCULO

WGS84 = referenceEllipsoid('wgs84');
[latitud,longitud,altitud] = ned2geodetic(X,Y,Z,latitud0,longitud0,...
                             altitud0,WGS84);

salida = [longitud;latitud;altitud];
```

- *"plan_de_vuelo.m"*

```
% Plan de vuelo. Seleccion de waypoints en formato:
%
% waypoint = [latitud longitud altitud]
%
% Con el objetivo de facilitar la busqueda de coordenadas de los puntos de
% paso, ejecutar en la ventana de comandos de Matlab "webmap". En la
% esquina superior derecha del mapa se representan las coordenadas
% geodeticas en funcion de la posicion actual del raton.
%
% NOTA: Es recomendable seleccionar los waypoints de tal manera que la
ruta
% sea lo mas sueva posible, por tanto se deberian escoger waypoints
% sucesivos de tal forma que segmentos de ruta contiguos no formen mas de
% 90 grados entre si con el fin de evitar giros abruptos. Del mismo modo,
% se sugiere la activacion del controlador de Uinf (delta_t) para recorrer
% la ruta a una velocidad constante. Por ultimo, es importante tener en
% cuenta que el correcto seguimiento de la misma queda a expensas de la
% actuacion del UAV.

waypoints_mat = [37.42200 -5.88350 150;
                 37.42400 -5.88875 175;
                 37.42200 -5.89300 200;
                 37.41400 -5.89300 200;
                 37.41200 -5.89825 175;
                 37.41400 -5.90350 150;
                 37.41500 -5.91010 125;
                 37.41800 -5.91210 100;
                 37.42100 -5.91010 100;
                 37.42100 -5.90350 125;
                 37.41800 -5.89300 150;
                 37.41800 -5.87420 150];

waypoints = reshape(waypoints_mat.',[],1);
```

- *"postsimulacion.m"*

```
% Script que representa en distintas graficas datos de interes obtenidos
% a lo largo de la simulacion.

close all

figure;
subplot(3,1,1);
plot(posicion_geodetica.Time,posicion_geodetica.Data(:,2));
grid; title('Latitud');
xlabel('t [s]'); ylabel('\Phi [°]');
subplot(3,1,2);
plot(posicion_geodetica.Time,posicion_geodetica.Data(:,1));
grid; title('Longitud');
xlabel('t [s]'); ylabel('\Lambda [°]');
subplot(3,1,3);
plot(posicion_geodetica.Time,posicion_geodetica.Data(:,3));
grid; title('Altitud');
xlabel('t [s]'); ylabel('h [m]');
```

```
figure;
subplot(2,3,1);
plot(fuerzas.Time,fuerzas.Data(:,1));
grid; title('Fuerza X');
xlabel('t [s]'); ylabel('F_X [N]');
subplot(2,3,2);
plot(fuerzas.Time,fuerzas.Data(:,2));
grid; title('Fuerza Y');
xlabel('t [s]'); ylabel('F_Y [N]');
subplot(2,3,3);
plot(fuerzas.Time,fuerzas.Data(:,3));
grid; title('Fuerza Z');
xlabel('t [s]'); ylabel('F_Z [N]');
subplot(2,3,4);
plot(momentos.Time,momentos.Data(:,1));
grid; title('Momento alrededor de X');
xlabel('t [s]'); ylabel('L [N·m]');
subplot(2,3,5);
plot(momentos.Time,momentos.Data(:,2));
grid; title('Momento alrededor de Y');
xlabel('t [s]'); ylabel('M [N·m]');
subplot(2,3,6);
plot(momentos.Time,momentos.Data(:,3));
grid; title('Momento alrededor de Z');
xlabel('t [s]'); ylabel('N [N·m]');

figure;
subplot(3,3,1);
plot(posicion_NED.Time,posicion_NED.Data(:,1));
grid; title('Posicion Norte');
xlabel('t [s]'); ylabel('N [m]');
subplot(3,3,2);
plot(posicion_NED.Time,posicion_NED.Data(:,2));
grid; title('Posicion Este');
xlabel('t [s]'); ylabel('E [m]');
subplot(3,3,3);
plot(posicion_NED.Time,posicion_NED.Data(:,3));
grid; title('Posicion Vertical');
xlabel('t [s]'); ylabel('D [m]');
subplot(3,3,4);
plot(velocidades_lineales.Time,velocidades_lineales.Data(:,1));
grid; title('Velocidad X');
xlabel('t [s]'); ylabel('V_X [m/s]');
subplot(3,3,5);
plot(velocidades_lineales.Time,velocidades_lineales.Data(:,2));
grid; title('Velocidad Y');
xlabel('t [s]'); ylabel('V_Y [m/s]');
subplot(3,3,6);
plot(velocidades_lineales.Time,velocidades_lineales.Data(:,3));
grid; title('Velocidad Z');
xlabel('t [s]'); ylabel('V_Z [m/s]');
subplot(3,3,7);
plot(aceleraciones_lineales.Time,aceleraciones_lineales.Data(:,1));
grid; title('Aceleracion X');
xlabel('t [s]'); ylabel('A_X [m/s^2]');
subplot(3,3,8);
plot(aceleraciones_lineales.Time,aceleraciones_lineales.Data(:,2));
grid; title('Aceleracion Y');
xlabel('t [s]'); ylabel('A_Y [m/s^2]');
```

```
subplot(3,3,9);
plot(aceleraciones_lineales.Time,aceleraciones_lineales.Data(:,3));
grid; title('Aceleracion Z');
xlabel('t [s]'); ylabel('A_Z [m/s^2]');

figure;
subplot(3,3,1);
plot(actitud.Time,actitud.Data(:,1));
grid; title('Alabeo');
xlabel('t [s]'); ylabel('\phi [°]');
subplot(3,3,2);
plot(actitud.Time,actitud.Data(:,2));
grid; title('Cabeceo');
xlabel('t [s]'); ylabel('\theta [°]');
subplot(3,3,3);
plot(actitud.Time,actitud.Data(:,3));
grid; title('Guinada');
xlabel('t [s]'); ylabel('\psi [°]');
subplot(3,3,4);
plot(velocidades_angulares.Time,velocidades_angulares.Data(:,1));
grid; title('Velocidad de alabeo');
xlabel('t [s]'); ylabel('P [°/s]');
subplot(3,3,5);
plot(velocidades_angulares.Time,velocidades_angulares.Data(:,2));
grid; title('Velocidad de cabeceo');
xlabel('t [s]'); ylabel('Q [°/s]');
subplot(3,3,6);
plot(velocidades_angulares.Time,velocidades_angulares.Data(:,3));
grid; title('Velocidad de guinada');
xlabel('t [s]'); ylabel('R [°/s]');
subplot(3,3,7);
plot(aceleraciones_angulares.Time,aceleraciones_angulares.Data(:,1));
grid; title('Aceleracion de alabeo');
xlabel('t [s]'); ylabel('P punto [°/s^2]');
subplot(3,3,8);
plot(aceleraciones_angulares.Time,aceleraciones_angulares.Data(:,2));
grid; title('Aceleracion de cabeceo');
xlabel('t [s]'); ylabel('Q punto [°/s^2]');
subplot(3,3,9);
plot(aceleraciones_angulares.Time,aceleraciones_angulares.Data(:,3));
grid; title('Aceleracion de guinada');
xlabel('t [s]'); ylabel('R punto [°/s^2]');

figure;
subplot(3,1,1);
plot(deflexion_alerones.Time,deflexion_alerones.Data);
grid; title('Alerones');
xlabel('t [s]'); ylabel('Deflexion de los alerones [°]');
subplot(3,1,2);
plot(deflexion_elevadores.Time,deflexion_elevadores.Data);
grid; title('Elevadores');
xlabel('t [s]'); ylabel('Deflexion de los elevadores [°]');
subplot(3,1,3);
plot(fuerza_motor.Time,fuerza_motor.Data);
grid; title('Motor');
xlabel('t [s]'); ylabel('Fuerza del motor [N]');
```

```
figure;
plot3(posicion_NED.Data(:,1),posicion_NED.Data(:,2),...
      posicion_inicial.Data(1,3)-posicion_NED.Data(:,3));
grid; title('Trayectoria');
xlabel('Norte [m]'); ylabel('Este [m]'); zlabel('Altitud [m]');
```

- *"postsimulacion_telemetria.m"*

```
% Script que representa en distintas graficas datos de interes obtenidos
% a lo largo de la simulacion de la telemetria.

close all

figure;
subplot(3,1,1);
plot(posicion_geodetica.Time,posicion_geodetica.Data(:,1),...
     latitud.Time,latitud.Data,'r');
grid; title('Latitud');
xlabel('t [s]'); ylabel('\Phi [°]');
subplot(3,1,2);
plot(posicion_geodetica.Time,posicion_geodetica.Data(:,2),...
     longitud.Time,longitud.Data,'r');
grid; title('Longitud');
xlabel('t [s]'); ylabel('\Lambda [°]');
subplot(3,1,3);
plot(posicion_geodetica.Time,posicion_geodetica.Data(:,3),...
     altitud.Time,altitud.Data,'r');
grid; title('Altitud');
xlabel('t [s]'); ylabel('h [m]');

figure;
subplot(3,1,1)
plot(velocidades_lineales.Time,velocidades_lineales.Data(:,1),...
     v_x.Time,v_x.Data);
grid; title('Velocidad X');
xlabel('t [s]'); ylabel('V_X [m/s]');
subplot(3,1,2);
plot(velocidades_lineales.Time,velocidades_lineales.Data(:,2),...
     v_y.Time,v_y.Data);
grid; title('Velocidad Y');
xlabel('t [s]'); ylabel('V_Y [m/s]');
subplot(3,1,3);
plot(velocidades_lineales.Time,velocidades_lineales.Data(:,3),...
     v_z.Time,v_z.Data);
grid; title('Velocidad Z');
xlabel('t [s]'); ylabel('V_Z [m/s]');

figure;
subplot(3,1,1)
plot(aceleraciones_lineales.Time,aceleraciones_lineales.Data(:,1),...
     a_x.Time,a_x.Data);
grid; title('Aceleracion X');
xlabel('t [s]'); ylabel('A_X [m/s^2]');
```

```
subplot(3,1,2);
plot(aceleraciones_lineales.Time,aceleraciones_lineales.Data(:,2),...
     a_y.Time,a_y.Data);
grid; title('Aceleracion Y');
xlabel('t [s]'); ylabel('A_Y [m/s^2]');
subplot(3,1,3);
plot(aceleraciones_lineales.Time,aceleraciones_lineales.Data(:,3),...
     a_z.Time,a_z.Data);
grid; title('Aceleracion Z');
xlabel('t [s]'); ylabel('A_Z [m/s^2]');

figure;
subplot(3,1,1);
plot(actitud.Time,actitud.Data(:,1),alabeo.Time,alabeo.Data*180/pi);
grid; title('Alabeo');
xlabel('t [s]'); ylabel('\phi [°]');
subplot(3,1,2);
plot(actitud.Time,actitud.Data(:,2),cabeceo.Time,cabeceo.Data*180/pi);
grid; title('Cabeceo');
xlabel('t [s]'); ylabel('\theta [°]');
subplot(3,1,3);
plot(actitud.Time,actitud.Data(:,3),guinada.Time,...
     mod(guinada.Data*180/pi,360));
grid; title('Guinada');
xlabel('t [s]'); ylabel('\psi [°]');

figure;
subplot(3,1,1);
plot(velocidades_angulares.Time,velocidades_angulares.Data(:,1),...
     w_alabeo.Time,w_alabeo.Data);
grid; title('Velocidad de alabeo');
xlabel('t [s]'); ylabel('P [°/s]');
subplot(3,1,2);
plot(velocidades_angulares.Time,velocidades_angulares.Data(:,2),...
     w_cabeceo.Time,w_cabeceo.Data);
grid; title('Velocidad de cabeceo');
xlabel('t [s]'); ylabel('Q [°/s]');
subplot(3,1,3);
plot(velocidades_angulares.Time,velocidades_angulares.Data(:,3),...
     w_guinada.Time,w_guinada.Data);
grid; title('Velocidad de guinada');
xlabel('t [s]'); ylabel('R [°/s]');
```

- *"procesador_actitud.m"*

```
function salida = procesador_actitud(entrada)
% Funcion que procesa la actitud del UAV con el objetivo de limitar los
% rangos de definicion de los angulos de Euler: phi, theta y psi. De esta
% manera, dichos angulos abarcan [-180°, 180°), [-90°, 90°) y [0°, 359°)
% respectivamente.
%
% salida = procesador_actitud(entrada)
%
% Siendo:
%
% phi    = entrada(1);
% theta  = entrada(2);
```

```
% psi    = entrada(3);
%
% phi_   = salida(1);
% theta_ = salida(2);
% psi_   = salida(3);


%% ENTRADA

phi   = entrada(1);
theta = entrada(2);
psi   = entrada(3);


%% CALCULO

phi_   = 2*pi*(phi/(2*pi)-floor(0.5+phi/(2*pi)));

theta_ = pi*(theta/pi-floor(0.5+theta/pi));

psi_   = mod(psi,2*pi);


%% SALIDA

salida = [phi_;theta_;psi_];
```

- *"punto_de_equilibrio.m"*

```
% Script que calcula el punto de equilibrio del UAV.

close all, clear all

options           = optimset('TolFun',1e-16,'TolX',1e-16,'TolCon',1e-16,...
                    'MaxIter',Inf,'MaxFunEvals',Inf);
x0                = [15 0 1 0 0 0 0.01 0.1 0 0 0 1];
lb                = [0 0 0 0 0 0 -pi/2 -pi/2 0 944 1056 1100];
ub                = [30 0 2 0 0 0 pi/2 pi/2 0 2100 2100 2100];
estado_equilibrio = fsolve(@eom,x0,options);
salida_eom        = eom(estado_equilibrio)

U_equilibrio       = estado_equilibrio(1);
V_equilibrio       = estado_equilibrio(2);
W_equilibrio       = estado_equilibrio(3);
P_equilibrio       = estado_equilibrio(4);
Q_equilibrio       = estado_equilibrio(5);
R_equilibrio       = estado_equilibrio(6);
phi_equilibrio     = estado_equilibrio(7);
theta_equilibrio   = estado_equilibrio(8);
psi_equilibrio     = estado_equilibrio(9);
delta_a_equilibrio = estado_equilibrio(10);
delta_e_equilibrio = estado_equilibrio(11);
delta_t_equilibrio = estado_equilibrio(12);
```

- *"receptor_gnss.m"*

```
% Script a ejecutar en una segunda instancia de Matlab. Su objetivo es
% representar la localizacion del UAV y de la ruta sobre un mapa. Para ello
% recibe datos de la instancia principal, la cual debera ejecutar el
% simulador.
%
% NOTA: para reiniciar el script, presionar CTRL+C y luego ejecutar en la
% ventana de comandos "delete(GNSS)".

plan_de_vuelo;

waypoints_num = size(waypoints_mat,1);

waypoint_icono = fullfile(pwd,'waypoint.ico');
wmmarker(waypoints_mat(1,1),waypoints_mat(1,2),'Icon',waypoint_icono,...
'FeatureName',sprintf('Waypoint %d',1),'Description',...
sprintf('%d metros',waypoints_mat(1,3)),'Autofit',true);

for k=2:waypoints_num

    waypoint_icono = fullfile(pwd,'waypoint.ico');
    wmmarker(waypoints_mat(k,1),waypoints_mat(k,2),'Icon',...
    waypoint_icono,'FeatureName',sprintf('Waypoint %d',k),...
    'Description',sprintf('%d metros',waypoints_mat(k,3)),'Autofit',true);

    wmline([waypoints_mat(k-1,1) waypoints_mat(k,1)],...
    [waypoints_mat(k-1,2) waypoints_mat(k,2)],'Color','green',...
    'FeatureName',sprintf('Segmento %d',k-1),'Description',...
    sprintf('Waypoint %d a %d ',k-1,k),'Autofit',true);

end

GNSS       = udp('localhost', 'RemotePort', 3334, 'LocalPort', 3333);
fopen(GNSS);
GNSS.ReadAsyncMode = 'continuous';

datos = [];

while 1

    if GNSS.BytesAvailable>0

    datos = [datos; fread(GNSS,[4 1],'double').'];

    latitud  = datos(end,1);
    longitud = datos(end,2);
    psi      = datos(end,3);
    tiempo   = datos(end,4);

    if size(datos,1)>1

        wmremove;

    end
```

```
if psi>174.125 && psi<=354.125

  if psi>264.125

    if psi>309.125

      if psi>331.625

        if psi>342.875

          uav_icono = fullfile(pwd,'uav_348_75.ico');
          wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
          continue;

        else

          uav_icono = fullfile(pwd,'uav_337_5.ico');
          wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
          continue;

        end

      else

        if psi>320.375

          uav_icono = fullfile(pwd,'uav_326_25.ico');
          wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
          continue;
        else

          uav_icono = fullfile(pwd,'uav_315.ico');
          wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
          continue;

        end

      end

    else

      if psi>286.625

        if psi>297.875

          uav_icono = fullfile(pwd,'uav_303_75.ico');
          wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
          continue;

        else

          uav_icono = fullfile(pwd,'uav_292_5.ico');
          wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
          continue;

        end
```

```
        else

          if psi>275.375

            uav_icono = fullfile(pwd,'uav_281_25.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;

          else

            uav_icono = fullfile(pwd,'uav_270.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;

          end

        end

      end

    else

      if psi>219.125

        if psi>241.625

          if psi>252.875

            uav_icono = fullfile(pwd,'uav_258_75.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;

          else

            uav_icono = fullfile(pwd,'uav_247_5.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;

          end

        else

          if psi>230.375

            uav_icono = fullfile(pwd,'uav_236_25.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;

          else

            uav_icono = fullfile(pwd,'uav_225.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;

          end
```

```
        end

      else

        if psi>196.625

          if psi>207.875

            uav_icono = fullfile(pwd,'uav_213_75.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;

          else

            uav_icono = fullfile(pwd,'uav_202_5.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;

          end

        else

          if psi>185.375

            uav_icono = fullfile(pwd,'uav_191_25.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;

          else

            uav_icono = fullfile(pwd,'uav_180.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;

          end

        end

      end

    end

  else

    if psi>84.125

      if psi>129.125

        if psi>151.625

          if psi>354.125

            uav_icono = fullfile(pwd,'uav_0.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;
```

```
        elseif psi>162.875

          uav_icono = fullfile(pwd,'uav_168_75.ico');
          wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
          continue;

        else

          uav_icono = fullfile(pwd,'uav_157_5.ico');
          wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
          continue;

        end

      else

        if psi>140.375

          uav_icono = fullfile(pwd,'uav_146_25.ico');
          wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
          continue;

        else

          uav_icono = fullfile(pwd,'uav_135.ico');
          wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
          continue;

        end

      end

    else

      if psi>106.625

        if psi>117.875

          uav_icono = fullfile(pwd,'uav_123_75.ico');
          wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
          continue;

        else

          uav_icono = fullfile(pwd,'uav_112_5.ico');
          wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
          continue;

        end

      else

        if psi>95.375
```

```
            uav_icono = fullfile(pwd,'uav_101_25.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;

          else

            uav_icono = fullfile(pwd,'uav_90.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;

          end

        end

      end

    else

      if psi>39.125

        if psi>61.625

          if psi>72.875

            uav_icono = fullfile(pwd,'uav_78_75.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;

          else

            uav_icono = fullfile(pwd,'uav_67_5.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;

          end

        else

          if psi>50.375

            uav_icono = fullfile(pwd,'uav_56_25.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;

          else

            uav_icono = fullfile(pwd,'uav_45.ico');
            wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
            continue;

          end

        end

      else
```

```
            if psi>16.625

              if psi>27.875

                uav_icono = fullfile(pwd,'uav_33_75.ico');
                wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
                continue;

              else

                uav_icono = fullfile(pwd,'uav_22_5.ico');
                wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
                continue;

              end

            else

              if psi>5.375

                uav_icono = fullfile(pwd,'uav_11_25.ico');
                wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
                continue;

              else

                uav_icono = fullfile(pwd,'uav_0.ico');
                wmmarker(latitud,longitud,'Icon',uav_icono,'Autofit',true);
                continue;

              end

            end

          end

        end

      end

      end

end
```

- *"ruta.m"*

```
function salida = ruta(entrada)
% Funcion que implementa la ruta a seguir por el sistema de gestion de
% vuelo.
%
% salida = ruta(entrada)
%
% Siendo:
%
```

```
% disparo_reinicio = entrada(1);
% p_reinicio       = entrada(2:4);
% p_ned            = entrada(5:7);
% tramo            = entrada(8);
% waypoints_ned    = entrada(9:end);
%
% r1               = salida(1);
% q1               = salida(2);
% tramo            = salida(3);
%
% donde r1 es el vector unitario que marca la posicion del UAV respecto al
% origen de coordenadas NED y q1 es el vector unitario que une el ultimo
% waypoint con el siguiente.


%% ENTRADA

disparo_reinicio = entrada(1);
p_reinicio       = entrada(2:4);
p_ned            = entrada(5:7);
tramo            = entrada(8);
waypoints_ned    = entrada(9:end);


%% CALCULO

k             = 3*tramo;
waypoints_num = size(waypoints_ned,1)/3;

if disparo_reinicio==1 || tramo==0

    tramo = 0;
    r1    = p_reinicio;
    r2    = waypoints_ned(1:3);

elseif disparo_reinicio==2

    r1 = p_reinicio;
    r2 = waypoints_ned(k-2:k);
    r3 = waypoints_ned(k+1:k+3);

else

    if k==3

        r1 = p_reinicio;
        r2 = waypoints_ned(k-2:k);
        r3 = waypoints_ned(k+1:k+3);

    else

        r1 = waypoints_ned(k-5:k-3);
        r2 = waypoints_ned(k-2:k);

        if tramo<waypoints_num

            r3 = waypoints_ned(k+1:k+3);
```

```
        end

    end

end

q1  = (r2-r1)/norm(r2-r1);

if tramo<waypoints_num

    if tramo==0

        q2 = q1;

    else

        q2 = (r3-r2)/norm(r3-r2);

    end

    n  = (q1+q2)/norm(q1+q2);

    if (p_ned-r2).'*n>=0

        tramo  = tramo+1;
        q1 = q2;

    end

end


%% SALIDA

salida = [r1;q1;tramo];
```

- *"segmento.m"*

```
function salida = segmento(entrada)
% Funcion que implementa el calculo de psi y altitud instantanea del
% sistema de gestion de vuelo.
%
% salida = segmento(entrada)
%
% Siendo:
%
% p_n   = entrada(1);
% p_e   = entrada(2);
% p_d   = entrada(3);
% r_n   = entrada(4);
% r_e   = entrada(5);
% r_d   = entrada(6);
% q_n   = entrada(7);
```

```
% q_e   = entrada(8);
% q_d   = entrada(9);
% psi   = entrada(10);
%
% psi_c = salida(1);
% h_c   = salida(2);


%% ENTRADA

p_n = entrada(1);
p_e = entrada(2);
p_d = entrada(3);
r_n = entrada(4);
r_e = entrada(5);
r_d = entrada(6);
q_n = entrada(7);
q_e = entrada(8);
q_d = entrada(9);
psi = entrada(10);


%% CALCULO

r = [r_n;r_e;r_d];
p = [p_n;p_e;p_d];

psi_inf = pi/4; % entre 0 y pi/2
k_ruta  = 1e-2; % suavidad del campo vectorial alrededor del segmento
psi_q = atan2(q_e,q_n); % rumbo de la senda

e_p = p-r;

n_aux = [0 -q_d q_e; q_d 0 -q_n; -q_e q_n 0]*[0;0;1];
n     = n_aux/norm(n_aux);

s       = e_p-(e_p.'*n)*n;
s_n     = s(1);
s_e     = s(2);

h_c = -r_d-sqrt(s_n^2+s_e^2)*(q_d/sqrt(q_n^2+q_e^2));

if (psi_q-psi)>pi

    psi_q = psi_q-2*pi;

elseif (psi_q-psi)<-pi

    psi_q = psi_q+2*pi;

end

e_py = -sin(psi_q)*(p_n-r_n)+cos(psi_q)*(p_e-r_e);

psi_c = psi_q-2*psi_inf*atan(k_ruta*e_py)/pi;
```

```
%% SALIDA

salida = [psi_c;h_c];
```

- *"sfun_keyboard_input_v1_2b.m"*

```
function sfun_keyboard_input_v1_2(block)
% modified from sfun_keyboard_input_v1_01.m of Marc Compere by Emanuele
Ruffaldi
% created : 17 June 2003
% modified: 20 June 2003
% created: 19 May 2009 => Level 2 and terminate
%

% Level-2 M file S-Function with keyboard
%   Copyright 1990-2004 The MathWorks, Inc.
%   $Revision: 1.1.6.1 $

  setup(block);

%endfunction

function setup(block)

  block.NumDialogPrms  = 0;
  block.NumInputPorts  = 0;
  block.NumOutputPorts = 2;

  block.OutputPort(1).Dimensions       = 1;
  block.OutputPort(1).SamplingMode = 'Sample';
  block.OutputPort(1).DatatypeID  = 0;
  block.OutputPort(2).Dimensions       = 1;
  block.OutputPort(2).SamplingMode = 'Sample';
  block.OutputPort(2).DatatypeID  = 0;
  block.SampleTimes = [-1, 0];
%end

  %% Register methods
%  block.RegBlockMethod('CheckParameters', @CheckPrms);
  block.RegBlockMethod('PostPropagationSetup',    @DoPostPropSetup);
  block.RegBlockMethod('InitializeConditions',    @InitConditions);
  block.RegBlockMethod('Outputs',                 @Output);
  block.RegBlockMethod('Update',                  @Update);
  block.RegBlockMethod('Terminate',                 @Terminate);
%endfunction

function DoPostPropSetup(block)

  %% Setup Dwork
  block.NumDworks = 3;
  block.Dwork(1).Name = 'key';
  block.Dwork(1).Dimensions      = 1;
  block.Dwork(1).DatatypeID      = 0;
  block.Dwork(1).Complexity      = 'Real';
  block.Dwork(1).UsedAsDiscState = true;
```

```
  block.Dwork(2).Name = 'fig';
  block.Dwork(2).Dimensions      = 1;
  block.Dwork(2).DatatypeID      = 0;
  block.Dwork(2).Complexity      = 'Real';
  block.Dwork(2).UsedAsDiscState = false;

  block.Dwork(3).Name = 'new';
  block.Dwork(3).Dimensions      = 1;
  block.Dwork(3).DatatypeID      = 0;
  block.Dwork(3).Complexity      = 'Real';
  block.Dwork(3).UsedAsDiscState = true;
%endfunction

function InitConditions(block)

  %% Initialize Dwork
  block.Dwork(1).Data = 0;
   handle.figure=findobj('type','figure','Tag','keyboard input figure');

   if isempty(handle.figure)
      % 'position' args -> [left, bottom, width, height]
      handle.figure=figure('position',[100 100 400 200],...
                           'WindowStyle','Modal',...
                           'Name','Keyboard Input',...

'Color',get(0,'DefaultUicontrolBackgroundColor')); %,...
                          %'HandleVisibility','callback');
      %handle.figure=figure('position',[800 620 400 300]);
      %handle.figure=figure('position',[800 620 400
300],'WindowButtonDownFcn',@myCallback)
      %handle.figure=figure('position',[800 620 400
300],'WindowButtonMoveFcn',@myCallback_move,'WindowButtonDownFcn',@myCallba
ck_clickdown)
      set(handle.figure,'Tag','keyboard input figure');

      % Make the OFF button (position args->[left bottom width height])
      handle.offbutton = uicontrol(handle.figure,...
          'Style','pushbutton',...
          'Units','characters',...
          'Position',[5 5 46 2],...
          'String','Disable exclusive figure-keyboard input',...
          'Callback',{@turn_modal_off,handle});

      % Make the ON button (position args->[left bottom width height])
      handle.onbutton = uicontrol(handle.figure,...
          'Style','pushbutton',...
          'Units','characters',...
          'Position',[5 1 46 2],...
          'String','Re-enable exclusive figure-keyboard input',...
          'Callback',{@turn_modal_on,handle});

   else, % reset the figure to 'modal' to continue accepting keyboard input
      set(handle.figure,'WindowStyle','Modal')
   end
   if strcmp(class(handle.figure),'double')
       block.Dwork(2).Data = handle.figure;
   else
          block.Dwork(2).Data = handle.figure.Number;
   end
```

```
%endfunction

function Output(block)

  block.OutputPort(1).Data = block.Dwork(1).Data;
  block.OutputPort(2).Data = block.Dwork(3).Data;

%endfunction

function Update(block)

   handle.figure = block.Dwork(2).Data;
   %handle = get(handle.figure,'userdata');

   current_char=get(handle.figure,'CurrentCharacter'); % a single
character, like 'b'

   % update the grahics object
   %set(handle.point,'Xdata',[x(1) x(1)],'Ydata',[x(2) x(2)],'Zdata',[-1
+1]);


   % conditionally update the (numeric) state
   if ~isempty(current_char)
      block.Dwork(1).Data =double(current_char);
      block.Dwork(3).Data = 1;
       % reset 'CurrentCharacter' so if user lifts up from key, this is
noticed
       set(handle.figure,'CurrentCharacter',char(0)) % the plus key is the
only key that may be
   else,
      block.Dwork(3).Data = 0;
   end

                                    % pressed, but when the user stops, is
not noticed

   % notes:
   %    - use sprintf() to convert string -> number contained in a string
(or character) variable
   %    - use char() to convert floating point number -> ascii character
   %    - use str2num() to convert the string-number into a (double)
floating point number
   %    - use str2num() with char() to convert a number-string into a
char-string
   %         char(97) == char(str2num('97')) --> 'a'
   %    - use num2str() to convert a (double) number into the same number
but contained
   %      in a string variable
   % For example:
   %    tmp='a';                          % assign a string
   %    tmp_num=sprintf('%i',tmp)         % convert that string into a
number contained in a string variable, tmp_num
   %        (tmp_num is the string containing the characters '97')
   %    tmp_char=char(str2num(tmp_num)) % convert that string variable
back into a number, then into the original string
```

```
%endfunction



% Callback for turning Modal OFF
function turn_modal_off(obj,eventdata,handle)
%disp('turn_modal_off:')
%handle
set(handle.figure,'WindowStyle','Normal')
%end


% Callback for turning Modal ON
function turn_modal_on(obj,eventdata,handle)
%disp('turn_modal_on:')
%handle
set(handle.figure,'WindowStyle','Modal')
%end


% % Callback for 'WindowButtonMoveFcn' in figure
% function myCallback_move(obj,eventdata)
% str=sprintf('\tWindowButtonMoveFcn callback executing');disp(str)
% end
%
% % Callback for 'WindowButtonDownFcn' in figure
% function myCallback_clickdown(obj,eventdata)
% str=sprintf('\t\tWindowButtonDownFcn callback executing');disp(str)
% end



function Terminate(block)
   handle.figure = block.Dwork(2).Data;
   if handle.figure ~= 0
       close (handle.figure);
   end
```

- *"velocidades_ang.m"*

```
function salida = velocidades_ang(entrada)
% Funcion que calcula las velocidades angulares experimentadas por el UAV
% en ejes de navegacion.
%
% salida = velocidades_ang(entrada)
%
% Siendo:
%
% P     = entrada(1);
% Q     = entrada(2);
% R     = entrada(3);
% phi   = entrada(4);
% theta = entrada(5);
```

```
%
% W_N   = salida(1);
% W_E   = salida(2);
% W_D   = salida(3);


%% ENTRADA

P     = entrada(1);
Q     = entrada(2);
R     = entrada(3);
phi   = entrada(4);
theta = entrada(5);


%% SALIDA

salida = [1   sin(phi)*tan(theta)   cos(phi)*tan(theta);
          0        cos(phi)             -sin(phi)      ;
          0   sin(phi)/cos(theta)   cos(phi)/cos(theta)]*[P;Q;R];
```

- *"velocidades_lin.m"*

```
function salida = velocidades_lin(entrada)
% Funcion que calcula las velocidades lineales experimentadas por el UAV
% en ejes de navegacion.
%
% salida = velocidades_lin(entrada)
%
% Siendo:
%
% U     = entrada(1);
% V     = entrada(2);
% W     = entrada(3);
% phi   = entrada(4);
% theta = entrada(5);
% psi   = entrada(6);
%
% V_N   = salida(1);
% V_E   = salida(2);
% V_D   = salida(3);


%% ENTRADA

U     = entrada(1);
V     = entrada(2);
W     = entrada(3);
phi   = entrada(4);
theta = entrada(5);
psi   = entrada(6);
```

```
%% CALCULO

% Matriz de rotacion de ejes cuerpo a ejes NED:

R_bn = [cos(theta)*cos(psi) sin(phi)*sin(theta)*cos(psi)-cos(phi)*...
        sin(psi) cos(phi)*sin(theta)*cos(psi)+sin(phi)*sin(psi);
        cos(theta)*sin(psi) sin(phi)*sin(theta)*sin(psi)+cos(phi)*...
        cos(psi) cos(phi)*sin(theta)*sin(psi)-sin(phi)*cos(psi);
        -sin(theta) sin(phi)*cos(theta) cos(phi)*cos(theta)];


%% SALIDA

salida = R_bn*[U;V;W];
```

- *"x8.m"*

```
% Script que contiene las caracteristicas tecnicas del Skywalker X8.

global m b c Salar AR Ix Iy Iz Ixz e alpha0 coefprop C_L0 C_Lalpha C_Lq ...
       C_Ldelta_e C_D0 C_Dbeta1 C_Dbeta2 C_Dq C_Ddelta_e C_Y0 C_Ybeta ...
       C_Yp C_Yr C_Ydelta_a C_l0 C_lbeta C_lp C_lr C_ldelta_a C_m0 ...
       C_malpha C_mp C_mq C_mdelta_e C_n0 C_nbeta C_np C_nr C_ndelta_a ...
       gamma gamma1 gamma2 gamma3 gamma4 gamma5 gamma6 gamma7 gamma8 ...
       C_p0 C_pbeta C_pp C_pr C_pdelta_a C_r0 C_rbeta C_rp C_rr C_rdelta_a

%% DATOS GENERALES DEL UAV

m      = 3.797;
b      = 2.1000;
c      = 0.3571;
Salar  = 0.7500;
AR     = b^2/Salar; % alargamiento (Aspect Ratio)


%% MOMENTOS DE INERCIA

Ix  = 1.2290;
Iy  = 0.1702;
Iz  = 0.8808;
Ixz = 0.9343;


%% DATOS AERODINAMICOS

e      = 0.9935; % coeficiente de Oswald
alpha0 = 0.2670; % alpha de entrada en perdida


%% LEY DE PROPULSION

coefprop = [0.0168798 -0.0422854]; % motor_pwm y U^2
```

```
%% COEFICIENTES DE ESTABILIDAD

C_L0       = 0.0254;
C_Lalpha   = 4.0191; % rad^-1
C_Lq       = 3.8954;
C_Ldelta_e = 0.5872;

C_D0       = 0.0102;
C_Dbeta1   = -2.0864e-7;
C_Dbeta2   = 0.0671;
C_Dq       = 0;
C_Ddelta_e = 0.8461;

C_Y0       = 3.2049e-18;
C_Ybeta    = -0.1949;
C_Yp       = -0.1172;
C_Yr       = 0.0959;
C_Ydelta_a = -0.0696;

C_l0       = 1.1518e-18;
C_lbeta    = -0.0765;
C_lp       = -0.4018;
C_lr       = 0.0250;
C_ldelta_a = 0.2987;

C_m0       = 0.0180;
C_malpha   = -0.2524;
C_mp       = -0.2168;
C_mq       = -1.3047;
C_mdelta_e = -0.4857;

C_n0       = -2.2667e-7;
C_nbeta    = 0.0403;
C_np       = -0.0247;
C_nr       = -0.1252;
C_ndelta_a = 0.0076;


%% GAMMAS

gamma  = Ix*Iz-Ixz^2;
gamma1 = (Ixz*(Ix-Iy+Iz))/gamma;
gamma2 = (Iz*(Iz-Iy)+Ixz^2)/gamma;
gamma3 = Iz/gamma;
gamma4 = Ixz/gamma;
gamma5 = (Iz-Ix)/Iy;
gamma6 = Ixz/Iy;
gamma7 = (Ix*(Ix-Iy)+Ixz^2)/gamma;
gamma8 = Ix/gamma;


%% COEFICIENTES CRUZADOS DE VELOCIDADES ANGULARES

C_p0       = gamma3*C_l0+gamma4*C_n0;
C_pbeta    = gamma3*C_lbeta+gamma4*C_nbeta;
C_pp       = gamma3*C_lp+gamma4*C_np;
C_pr       = gamma3*C_lr+gamma4*C_nr;
C_pdelta_a = gamma3*C_ldelta_a+gamma4*C_ndelta_a;
```

```
C_r0       = gamma4*C_l0+gamma8*C_n0;
C_rbeta    = gamma4*C_lbeta+gamma8*C_nbeta;
C_rp       = gamma4*C_lp+gamma8*C_np;
C_rr       = gamma4*C_lr+gamma8*C_nr;
C_rdelta_a = gamma4*C_ldelta_a+gamma8*C_ndelta_a;
```

- *"x8_aero.m"*

```
function salida = x8_aero(entrada)
% Funcion que calcula los coeficientes aerodinamicos: C_L, C_D, C_Y, C_l,
% C_m y C_n.
%
% salida = x8_aero(entrada)
%
% Siendo:
%
% alpha      = entrada(1);
% beta       = entrada(2);
% Uinf       = entrada(3);
% P          = entrada(4);
% Q          = entrada(5);
% R          = entrada(6);
% delta_a    = entrada(7);
% delta_e    = entrada(8);
% sigma      = entrada(9);
% m          = entrada(10);
% b          = entrada(11);
% c          = entrada(12);
% Salar      = entrada(13);
% AR         = entrada(14);
% C_L0       = entrada(15);
% C_Lalpha   = entrada(16);
% C_Lq       = entrada(17);
% C_Ldelta_e = entrada(18);
% C_D0       = entrada(19);
% C_Dbeta1   = entrada(20);
% C_Dbeta2   = entrada(21);
% C_Dq       = entrada(22);
% C_Ddelta_e = entrada(23);
% C_Y0       = entrada(24);
% C_Ybeta    = entrada(25);
% C_Yp       = entrada(26);
% C_Yr       = entrada(27);
% C_Ydelta_a = entrada(28);
% C_l0       = entrada(29);
% C_lbeta    = entrada(30);
% C_lp       = entrada(31);
% C_lr       = entrada(32);
% C_ldelta_a = entrada(33);
% C_m0       = entrada(34);
% C_malpha   = entrada(35);
% C_mp       = entrada(36);
% C_mq       = entrada(37);
% C_mdelta_e = entrada(38);
% C_n0       = entrada(39);
% C_nbeta    = entrada(40);
% C_np       = entrada(41);
```

```
% C_nr       = entrada(42);
% C_ndelta_a = entrada(43);
%
% C_L        = salida(1);
% C_D        = salida(2);
% C_Y        = salida(3);
% C_l        = salida(4);
% C_m        = salida(5);
% C_n        = salida(6);


%% ENTRADA

alpha      = entrada(1);
beta       = entrada(2);
Uinf       = entrada(3);
P          = entrada(4);
Q          = entrada(5);
R          = entrada(6);
delta_a    = entrada(7);
delta_e    = entrada(8);
sigma      = entrada(9);
m          = entrada(10);
b          = entrada(11);
c          = entrada(12);
Salar      = entrada(13);
AR         = entrada(14);
e          = entrada(15);
C_L0       = entrada(16);
C_Lalpha   = entrada(17);
C_Lq       = entrada(18);
C_Ldelta_e = entrada(19);
C_D0       = entrada(20);
C_Dbeta1   = entrada(21);
C_Dbeta2   = entrada(22);
C_Dq       = entrada(23);
C_Ddelta_e = entrada(24);
C_Y0       = entrada(25);
C_Ybeta    = entrada(26);
C_Yp       = entrada(27);
C_Yr       = entrada(28);
C_Ydelta_a = entrada(29);
C_l0       = entrada(30);
C_lbeta    = entrada(31);
C_lp       = entrada(32);
C_lr       = entrada(33);
C_ldelta_a = entrada(34);
C_m0       = entrada(35);
C_malpha   = entrada(36);
C_mp       = entrada(37);
C_mq       = entrada(38);
C_mdelta_e = entrada(39);
C_n0       = entrada(40);
C_nbeta    = entrada(41);
C_np       = entrada(42);
C_nr       = entrada(43);
C_ndelta_a = entrada(44);
```

```
%% CALCULO

C_L = (1-sigma)*(C_L0+C_Lalpha*alpha)+sigma*(2*sign(alpha)*...
      sin(alpha)^2*cos(alpha))+0.5*C_Lq*c*Q/(Uinf+1e-16)+...
      C_Ldelta_e*delta_e;
C_D = C_D0+(1-sigma)*((C_L0+C_Lalpha*alpha)^2/(pi*e*AR))+sigma*(2*...
      sign(alpha)*sin(alpha)^3)+0.5*C_Dq*c*Q/(Uinf+1e-16)+C_Dbeta1*beta+...
      C_Dbeta2*beta^2+C_Ddelta_e*delta_e;
C_Y = C_Y0+C_Ybeta*beta+0.5*C_Yp*b*P/(Uinf+1e-16)+0.5*C_Yr*b*R/(Uinf+...
      1e-16)+C_Ydelta_a*delta_a;
C_l = C_l0+C_lbeta*beta+0.5*C_lp*b*P/(Uinf+1e-16)+0.5*C_lr*b*R/(Uinf+...
      1e-16)+C_ldelta_a*delta_a;
C_m = (1-sigma)*(C_m0+C_malpha*alpha)+sigma*(C_mp*sign(alpha)*...
      sin(alpha)^2)+0.5*C_mq*b*Q/(Uinf+1e-16)+C_mdelta_e*delta_e;
C_n = C_n0+C_nbeta*beta+0.5*C_np*b*P/(Uinf+1e-16)+0.5*C_nr*b*R/(Uinf+...
      1e-16)+C_ndelta_a*delta_a;


%% SALIDA

salida = [C_L;C_D;C_Y;C_l;C_m;C_n];
```

Referencias

[1] Fernández Rojas, B. (2014). *Desarrollo de un avión no tripulado: construcción, integración de sistemas y ensayos en vuelo*. Proyecto Fin de Carrera. Universidad de Sevilla. Sevilla, España.

[2] Gryte, K. (2015). *High angle of attack landing of an unmanned aerial vehicle. MSc Tesis. Norwegian University of Science and Technology*. Trondheim, Norway.

[3] R. W. Beard, T. W. McLain. (2012). *Small Unmanned Aircraft Theory and Practice*. Princeton University Press. Princeton, New Jersey, EE.UU.

[4] J. W. Langelaan, N. Alley, J. Niedhoefer. (2010). *Wind field estimation for small unmanned aerial vehicles*. AIAA Guidance, Navigation and Control Conference (Toronto, Canada). AIAA. Reston (VA), EEUU.

[5] Ogata, K. (2010). *Ingeniería de control moderna*. Pearson Educación S.A. Madrid, España.

[6] K. J. Åström, T. Hägglund. (2009). *Control PID avanzado*. Pearson Educación S.A. Madrid. España.

[7] Vázquez Valenzuela, R. (2015). Apuntes de la asignatura "Fundamentos de navegación aérea". Universidad de Sevilla. Sevilla, España.

[8] Franco Espín, A. (2014). Apuntes de la asignatura "Sistemas de propulsión". Universidad de Sevilla. Sevilla, España.

[9] Heredia Benot, J.G. (2013). Apuntes de la asignatura "Control automático". Universidad de Sevilla. Sevilla, España.

[10] Fernández Camacho, E. (2016). Apuntes de la asignatura "Sistemas de control y guiado". Universidad de Sevilla. Sevilla, España.

[11] Descripción de las variables del protocolo Mavlink. http://mavlink.org/messages/ardupilotmega.

[12] Página oficial de Mission Planner. http://ardupilot.org/planner/.

[13] Página oficial de FlightGear. http://www.flightgear.org.

[14] Documentación de FlightGear. http://wiki.flightgear.org/Main_Page.

[15] Enlace de descarga del modelo gráfico de la aeronave para FlightGear. http://ftp.igh.cnrs.fr/pub/flightgear/ftp/Aircraft-2.0.0/Malolo1_0.0.zip.

[16] Enlace de descarga del escenario utilizado en FlightGear. http://ns334561.ip-5-196-65.eu/~fgscenery/WS2.0/scenery-2.0.1.html.

[17] Tutorial de instalación de FlightGear. http://www.flightgear.org/Docs/getstart/getstartch2.html#x6-130002.

[18] Ruffaldi. E. (2016). *Simulink Keyboard Input v2*. Bloque de Simulink. https://es.mathworks.com/matlabcentral/fileexchange/24216-simulink-keyboard-input-v2.

[19] Documentación de la caja de herramientas aeroespacial (*Aerospace Toolbox*) de Matlab. https://es.mathworks.com/help/aerotbx/index.html.

[20] Documentación del servicio de mapa web o WMS (*Web Map Service*) de Matlab. https://es.mathworks.com/help/map/web-map-service.html.

Glosario

UAV: *Unmanned Aerial Vehicle* 1
UAS: *Unmanned Aerial System* 1
RPA: *Remotely Piloted Aircraft* 1
RPAS: *Remotely Piloted Aerial System* 1
BEC: *Battery Elimination Circuit* 3
ESC: *Electronic Speed Controller* 3
PSM: *Power Sensor Module* 3
PWM: *Pulse Width Modulation* 7
ZOH: *Zero Order Hold* 8
IMU: *Inertial Measurement Unit* 9
ECEF: *Earth Centered, Earth Fixed* 11
NED: *North, East, Down* 11
BFS: *Body Fixed System* 12
WF: *Wind Frame* 12
DCM: *Direction Cosine Matrix* 13
EGM96: *Earth Gravitational Model '96* 15
WGS84: *World Geodetic System '84* 15
GPS: *Global Positioning System* 16
GNC: *Guidance, Navigation and Control* 23
GNSS: *Global Navigation Satellite System* 23
ISA: *International Standard Atmosphere* 24
FMS: *Flight Management System* 43
UDP: *User Datagram Protocol* 43
ND: *Navigation Display* 48
PFD: *Primary Flight Director* 48
CPU: *Central Processing Unit* 52
GPU: *Graphics Processing Unit* 53
GUI: *Graphical User Interface* 56
FCU: *Flight Control Unit* 65
TCP: *Transmission Control Protocol* 66