UNIVERSITAT POLITÈCNICA DE VALÈNCIA

ETSI Aeroespacial y Diseño Industrial

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Aeroespacial y Diseño Industrial

Desarrollo de una interfaz gráfica dinámica para un simulador de vuelo docente en Matlab

Trabajo Fin de Grado

Grado en Ingeniería Aeroespacial

AUTOR/A: Monteagudo Sáez, Jorge

Tutor/a: Martí Gómez-Aldaraví, Pedro

CURSO ACADÉMICO: 2023/2024

Agradecimientos

La elaboración de este trabajo no habría sido posible si no fuese por el apoyo de aquellas personas que nunca han dudado de mí y siempre han tenido un momento para animarme a seguir.

Me gustaría darle las gracias a mi familia, especialmente a mis padres por darme la oportunidad de elegir mi camino, y a mi tía Encarnita, que me ha acompañado durante horas para que la redacción de este documento fuese la mejor posible.

Gracias también a mis amigas y amigos, los cuales han sabido motivarme siempre que lo he necesitado y me han ayudado a ver el final del camino. Os quiero.

Por supuesto, también le doy las gracias a mi tutor, Pedro Martí, por la ayuda y los consejos que ha me han guiado ante cualquier obstáculo.

Resumen

El principal objetivo de este trabajo consistirá en la programación y elaboración de una GUI dinámica con la herramienta *AppDesigner* de MATLAB a partir del código de un simulador de vuelo académico. Esta aplicación facilitará la visualización en tiempo real de la respuesta de una aeronave con configuración convencional ante una variación en los controles de la misma. Se ha realizado un estudio comparativo de diferentes métodos numéricos con el fin de encontrar una alternativa mejor al *ode45* que utilizaba el programa original. Para ello se ha evaluado el tiempo de cálculo y el error cometido, dando como resultado que el *ode113* puede ser una alternativa mejor. Posteriormente, se validará la implementación del código del simulador en la aplicación y se expondrán los resultados obtenidos mediante la interfaz dinámica a través de secuencias temporales. Para finalizar, se estudiará el fenómeno del *gimbal lock* desde diferentes sistemas de referencia y se presentarán los datos de una serie de aeronaves que podrán ser estudiadas con la aplicación.

Una vez terminada la aplicación, el estudiantado podrá analizar la dinámica de una aeronave bajo la actuación sobre sus controles en tiempo real gracias a la interfaz dinámica que muestra la respuesta temporal en cada instante.

Palabras clave: Interfaz dinámica, Simulador, Matlab, Mecánica de vuelo
Paraules clau: Interfície dinàmica, Simulador, Matlab, Mecànica de vol

Abstract

The main objective of this work will consist in the programming and development of a dynamic GUI with MATLAB's *AppDesigner* tool from the code of an academic flight simulator. This application will facilitate the real-time visualization of the response of an aircraft with conventional configuration to a variation in its controls. A comparative study of different numerical methods has been carried out in order to find a better alternative to the *ode45* used by the original program. For this purpose, the calculation time and the error committed have been evaluated, resulting in the *ode113* being a better alternative. Subsequently, the implementation of the simulator code in the application will be validated and the results obtained by means of the dynamic interface will be presented through time sequences. Finally, the *gimbal lock* phenomenon will be studied from different reference systems and data from a number of aircraft that can be studied with the application will be presented.

Once the application is finished, the students will be able to analyze the dynamics of an aircraft under the actuation of its controls in real time thanks to the dynamic interface that shows the temporal response at each instant.

Keywords: Dynamic interface, Simulator, Matlab, Flight mechanics

Índice general

Índice de figuras

Índice de tablas

Lista de Símbolos

Geometría

b_w Envergadura alar

c_w Cuerda media aerodinámica

S_w Superficie alar

L Longitud de la aeronave

x_{CoG} Posición del Centro de Gravedad

m Masa de la aeronave

W Peso de la aeronave

I_{xx}, I_{yy}, I_{zz} Momentos de inercia

I_{xz}, I_{yx}, I_{zy} Productos de inercia cruzados

Ángulos

α Ángulo de ataque

$\dot{\alpha}$ Derivada temporal del ángulo de ataque

β Ángulo de derrape

$\dot{\beta}$ Derivada temporal del ángulo de derrape

ϕ Ángulo de asiento lateral o alabeo

θ Ángulo de asiento longitudinal o cabeceo

ψ Ángulo de guiñada o rumbo

$\dot{\phi}, \dot{\theta}, \dot{\psi}$ Derivadas temporales de los Ángulos de Euler en Ejes Cuerpo

μ, γ, χ Ángulos de asiento lateral, pendiente y rumbo en Ejes Viento

$\dot{\mu}, \dot{\gamma}, \dot{\chi}$ Derivadas temporales del asiento lateral, pendiente y rumbo en Ejes Viento

δ_E Deflexión de la superficie de control del timón de profundidad (elevadores)

δ_A Deflexión de la superficie de control de los alerones

δ_R Deflexión de la superficie de control del timón de dirección (rudder)

Ecuaciones de Bryan

V Velocidad del viento

$\dot{V}$ Derivada temporal de la velocidad del viento

u, v, w Componentes lineales de la velocidad

$\dot{u}, \dot{v}, \dot{w}$ Aceleraciones lineales

p, q, r Velocidades angulares en Ejes Cuerpo

$\dot{p}, \dot{q}, \dot{r}$ Aceleraciones angulares en Ejes Cuerpo

p_w, q_w, r_w Velocidades angulares en Ejes Viento

x, y, z Posición de la aeronave en Ejes Horizonte Local

$\dot{x}, \dot{y}, \dot{x}$ Derivadas temporales de la posición en Ejes Horizonte Local

Aerodinámica

D Fuerza de resistencia

Y Fuerza lateral

L Fuerza de sustentación

L Momento de alabeo

M Fuerza de cabeceo

N Fuerza de guiñada

X_A Componente longitudinal de la fuerza aerodinámica en Ejes Cuerpo

Y_A Componente lateral de la fuerza aerodinámica en Ejes Cuerpo

Z_A Componente vertical de la fuerza aerodinámica en Ejes Cuerpo

$X_{A,w}$ Componente longitudinal de la fuerza aerodinámica en Ejes Viento

$Y_{A,w}$ Componente lateral de la fuerza aerodinámica en Ejes Viento

$Z_{A,w}$ Componente vertical de la fuerza aerodinámica en Ejes Viento

$C_{X,w}, C_{Y,w}, C_{z,w}$ Coeficientes de fuerzas aerodinámicas en Ejes Viento

C_D Coeficiente de resistencia

C_{D0} Coeficiente de resistencia parásita

K Constante de resistencia inducida

C_L Coeficiente de sustentación

C_{L0} Coeficiente de sustentación a ángulo de ataque nulo

$C_{L\alpha}$ Pendiente de la curva de sustentación

$C_{L\delta E}$ Variación del coeficiente de sustentación con deflexión de elevadores

C_{Lq} Variación del coeficiente de sustentación con q

$C_{L\dot{\alpha}}$ Variación del coeficiente de sustentación con $\dot{\alpha}$

C_M Coeficiente de momento de cabeceo

C_{M0} Coeficiente de momento de cabeceo a ángulo de ataque nulo

$C_{M\alpha}$ Variación del coeficiente de momento de cabeceo con α

$C_{M\delta E}$ Variación del coeficiente de momento de cabeceo con deflexión de elevadores

C_{Mq} Variación del coeficiente de momento de cabeceo con q

$C_{M\dot{\alpha}}$ Variación del coeficiente de momento de cabeceo con $\dot{\alpha}$

C_Y Coeficiente de fuerza lateral

$C_{Y\beta}$ Variación del coeficiente de fuerza lateral con β

$C_{Y\dot{\beta}}$ Variación del coeficiente de fuerza lateral con $\dot{\beta}$

C_{Yp} Variación del coeficiente de fuerza lateral con p

C_{Yr} Variación del coeficiente de fuerza lateral con r

$C_{Y\delta A}$ Variación del coeficiente de fuerza lateral con deflexión de alerones

$C_{Y\delta R}$ Variación del coeficiente de fuerza lateral con deflexión de rudder

C_l Coeficiente de momento de alabeo

$C_{l\beta}$ Variación del coeficiente de momento de alabeo con β

$C_{l\dot{\beta}}$ Variación del coeficiente de momento de alabeo con $\dot{\beta}$

C_{lp} Variación del coeficiente de momento de alabeo con p

C_{lr} Variación del coeficiente de momento de alabeo con r

$C_{l\delta A}$ Variación del coeficiente de momento de alabeo con deflexión de alerones

$C_{l\delta R}$ Variación del coeficiente de momento de alabeo con deflexión de rudder

C_N Coeficiente de momento de guiñada

$C_{N\beta}$ Variación del coeficiente de momento de guiñada con β

$C_{N\dot{\beta}}$ Variación del coeficiente de momento de guiñada con $\dot{\beta}$

C_{Np} Variación del coeficiente de momento de guiñada con p

C_{Nr} Variación del coeficiente de momento de guiñada con r

$C_{N\delta A}$ Variación del coeficiente de momento de guiñada con deflexión de alerones

$C_{N\delta R}$ Variación del coeficiente de momento de guiñada con deflexión de rudder

Modelo de empuje

T Empuje

T_0 Empuje a nivel del mar

T_x, T_y, T_z Componentes del empuje en Ejes Cuerpo

$T_{x,w}, T_{y,w}, T_{z,w}$ Componentes del empuje en Ejes Viento

δ_P Posición de la palanca de gases

ε Inclinación del empuje con la velocidad

C_e Consumo específico de combustible

BPR Tasa de derivación

N_e Número de motores

m_f Masa de combustible consumida

$\dot{m}_f$ Flujo másico de combustible

Otros símbolos

g Aceleración debida a la gravedad

M Número de Mach

$\rho(z)$ Densidad

$\sigma(z)$ Densidad relativa

n_x, n_y, n_z Factores de carga definidos en Ejes Cuerpo

$n_{x,w}, n_{y,w}, n_{z,w}$ Factores de carga definidos en Ejes Viento

h Paso de integración de los métodos numéricos

Ω Ratio de giro

Capítulo 1

Planteamiento del trabajo

1.1. Introducción

Hoy en día la tecnología forma parte de la vida cotidiana de las personas como una extensión de ellas mismas con el propósito de facilitar las tareas diarias y avanzar en una infinidad de aplicaciones científicas. La aparición de los simuladores fue uno de estos avances y que constituyen una herramienta fundamental a día de hoy, ya que sirven para analizar procesos que experimentalmente pueden ser muy costosos o lentos y para adquirir nuevas habilidades.

Más concretamente, los simuladores de vuelo suponen un gran adelanto tanto en la parte de diseño de aeronaves como en el aprendizaje del pilotaje de las mismas. Con estas herramientas es posible prever el comportamiento de una aeronave sin necesidad de poner un modelo en el aire y, además, permiten posprocesar los datos recopilados para realizar tantos estudios como se deseen. Existen una gran diversidad de simuladores en función de la complejidad de los mismos, desde videojuegos que se encuentran al alcance de cualquiera, hasta cabinas enteras de grandes aviones comerciales que permiten el movimiento de dicha cabina en función de la actitud de la aeronave. Como el de la Figura 1.1.1 del Centro de Formación de Airbus en Miami.

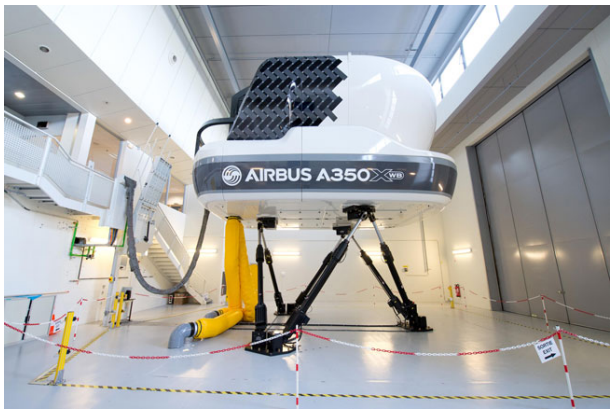


Figura 1.1.1: Simulador del A350 XWB. [1]

La gran ventaja que tienen los simuladores de vuelo es que permiten interactuar con ellos en tiempo real y ver cómo se modifican los parámetros que se estudian en la Mecánica de Vuelo al actuar sobre los controles de la aeronave. La representación gráfica de estas variables en tiempo real se realiza mediante interfaces gráficas dinámicas, las cuales muestran la evolución en cada instante de aquellos parámetros que se quieran estudiar y permiten ver los efectos inmediatos al actuar sobre el simulador.

1.2. Motivación

Como se ha comentado, los simuladores de vuelo son una valiosa herramienta a la hora de estudiar la dinámica de las aeronaves ante diversas condiciones de vuelo y actuaciones de los controles. Por ello, se cree necesario contar con un simulador académico lo más completo posible que le sirva al alumnado para mejorar su comprensión de los conceptos estudiados en las asignaturas de *Mecánica del Vuelo* y *Ampliación de Mecánica del Vuelo*.

Este fue el objetivo del trabajo de L. Villanueva [2] en el curso anterior. En él se creó, mediante programación MATLAB, un simulador de vuelo con el que es posible estudiar la dinámica de un avión en configuración convencional para diferentes maniobras en los dos sistemas de referencia habituales estudiados en las asignaturas mencionadas anteriormente: Ejes Cuerpo y Ejes Viento.

La razón de la creación del simulador es intentar cubrir la falta de disponibilidad de herramientas como esta por parte del estudiantado de la UPV, ofreciendo la posibilidad de darle un enfoque más práctico e ilustrativo a la teoría impartida en clase.

1.3. Objetivos

Para continuar con su propósito, y conseguir un simulador aún más completo, con este trabajo se pretende desarrollar una interfaz dinámica que implemente el código elaborado por Villanueva y que a su vez permita mostrar la variación temporal de las diferentes variables y su comportamiento bajo la actuación de los controles en tiempo real.

Los objetivos parciales que se pretenden alcanzar con la elaboración de este trabajo son los siguientes:

- Desarrollo de una aplicación en MATLAB que implemente el código del simulador de vuelo académico.
- Crear una interfaz gráfica dinámica que muestre la evolución temporal de las variables en tiempo real, consiguiéndose tener un cierto control sobre la simulación.

- Establecer una estrategia de representación para resolver las ecuaciones en cada instante y mostrar los valores de las variables en ese instante.
- Implementar de una forma sencilla para el usuario la actuación sobre los controles y que se puedan modificar durante toda la simulación.
- Estudiar si se consiguen mejores resultados y menor tiempo de cálculo resolviendo las ecuaciones con otro *solver*.
- Plantear la aplicación de tal manera que sea sencilla de manejar por el usuario.
- Crear una biblioteca de diversas aeronaves para su posible estudio con el simulador.

Capítulo 2

Fundamentos del simulador

2.1. Inicios y evolución de las *GUIs*

Durante la década de los años 30 empezaron a aparecer las primeras ideas sobre dispositivos de visualización gráfica que perseguían favorecer una mejor interacción entre humanos y máquinas. Sin embargo, como en muchas otras ocasiones en los avances de la computación, no existía una tecnología lo suficientemente desarrollada para poder llevar a cabo dichas ideas, es más, ni siquiera se había inventado el ordenador digital.

Estos conceptos quedaron relegados a un segundo plano hasta la llegada de Douglas Engelbart, un ingeniero eléctrico que trabajaba en el *Instituto NACA*, y que en 1957 consiguió un puesto en el *Standford Research Institute*, para desarrollar sus ideas sobre nuevas máquinas que permitiesen aumentar el intelecto humano.

Concretamente, el trabajo de Engelbert venía motivado por el artículo *"As We May Think"* (1945), escrito por Vannevar Bush y fuertemente marcado por los acontecimientos sucedidos durante la Segunda Guerra Mundial [3]. Años de trabajo llevaron a Engelbart a culminar su investigación en 1968 con la demostración de un sistema capaz de mostrar líneas sólidas y texto en la misma pantalla mediante tecnología de gráficos vectoriales. Este sistema, conocido como *NLS (oN-Line System)* (Figura 2.1.1), contaba, además, con un teclado y una caja rectangular con botones: era la primera versión del ratón.

Figura 2.1.1: El visor de oN-Line System, teclado y ratón. [4]

En 1973 la empresa de fotocopiadoras Xerox PARC creó su propio ordenador, el *Xerox Alto*, ya que hasta la fecha no existía ninguna máquina con las prestaciones suficientes para poder manejar su última invención: las impresoras láser. La primera versión del software del *Alto* era bastante tosca y los investigadores de PARC se dieron cuenta de que era necesaria una interfaz gráfica para sacarle el máximo partido y poder desarrollar nuevas funcionalidades. Así es como nació SmallTalk (Figura 2.1.2).

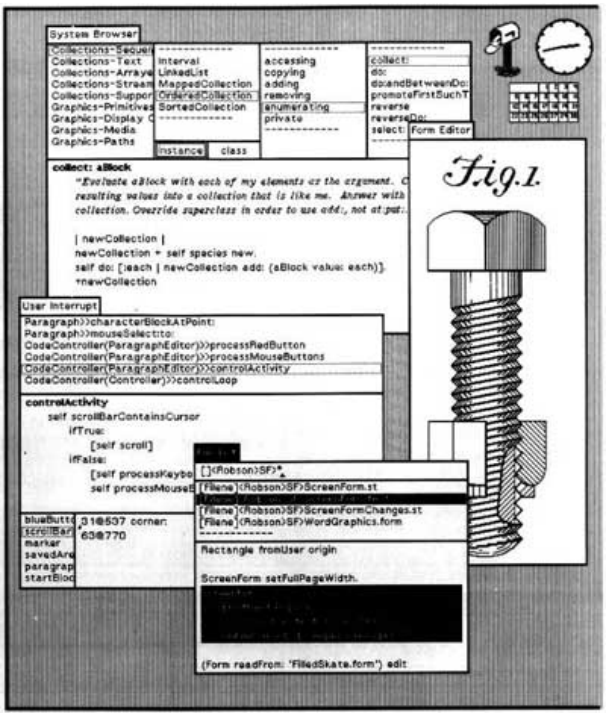


Figura 2.1.2: Entorno GUI de SmallTalk. [4]

Algunos consideran a SmallTalk como la primera GUI moderna, siendo concebida como un entorno gráfico que resultara sencillo de manejar para cualquier persona. Este entorno empezó a dar sus primeros pasos en 1974 y se siguió actualizando y mejorando continuamente.

Como se puede ver en la figura adjunta, SmallTalk contaba con una serie de ventanas, definidas por bordes, superpuestas sobre un fondo gris. Encima de estas ventanas se encontraba una pequeña barra que permitía moverlas por el espacio de visualización y llevar al frente aquellas con las que interesase trabajar.

También es necesario destacar la contribución de otro grupo de pioneros de las GUIs, que acabó fundando una de las empresas más conocidas y exitosas del panorama tecnológico actual: Steve Jobs y Steve Wozniak con la creación de su startup *Apple Computer* en 1976.

A día de hoy los lugares donde se pueden encontrar implementadas las GUIs no dejan de crecer: sistemas operativos en multitud de dispositivos, pantallas táctiles, videojuegos e incluso en la tecnología de realidad aumentada.

En el ámbito académico también se han realizado estudios para fomentar el uso de las GUIs en las clases, siendo el caso del artículo *"Graphic User Interfaces in an Engineering Educational Enviroment"* [5]. En él se muestran varias GUIs basadas en diferentes lenguajes de programación, así como los diversos beneficios que tiene cada uno y cómo los autores las han usado para demostrar el valor que tienen este tipo de herramientas a la hora de manejar información y posprocesar los resultados de una manera flexible.

Adicionalmente, en el artículo se llega a concluir que, si el alumnado quiere desarrollar una interfaz con la colaboración de los profesores, la mejor opción es MATLAB, pues los estudiantes suelen estar familiarizados con el programa y les permite hacer las tareas necesarias sin mayor dificultad.

2.2. Base teórica

En este capítulo se pretende mostrar la base teoría que se ha utilizado para la programación del simulador y los modelos matemáticos que este incluye. Las Secciones 2.2.1, 2.2.2 y 2.2.3, en las que se exponen las ecuaciones del movimiento, los modelos de empuje y el modelo aerodinámico usado, respectivamente, fueron desarrolladas por L. Villanueva [2] en su trabajo, por lo que se realizará un resumen de su contenido junto con la exposición de algún ejemplo que sea de interés para el trabajo actual.

2.2.1. Ecuaciones de Bryan

Para conocer la dinámica de vuelo de un vehículo rígido es necesario establecer una serie de ecuaciones, que permitan conocer cómo evolucionará el movimiento del avión ante las fuerzas y momentos que surgen por la acción sobre los elementos de control. Para llegar a dichas ecuaciones se debe partir de la Segunda Ley de Newton en términos relativos a un sistema de coordenadas no inercial y fijo al avión.

Inicialmente, estas ecuaciones fueron estudiadas por Zhukowsky (1897), Bryan (1903) y Lanchester (1908), y posteriormente, en un entorno tridimensional por Bothezat (1911) y Bryan (1911) [6].

El modelo implementado en el simulador es el que se ha estudiado en la asignatura *Ampliación de Mecánica del Vuelo*, caracterizado por tener seis grados de libertad (tres de rotación y tres de traslación) y a cuya obtención se llega a partir de una serie de hipótesis:

1. Se considera que el avión es un cuerpo rígido, por lo que se desprecian deformaciones y desplazamientos de componentes. Esto no tiene por qué ser cierto ya que en ocasiones las deformaciones pueden ser significativas y afectar a la sustentación.

2. El origen de los sistemas de coordenadas se encuentra en el Centro de Gravedad de la aeronave. De esta manera las fuerzas gravitatorias no generan momentos y, además, las ecuaciones del movimiento de rotación son independientes de las de traslación.

3. Se ignora la influencia del giro de la Tierra. La aceleración centrípeta asociada a la rotación de la Tierra y la aceleración de Coriolis se pueden considerar despreciables siempre que no se esté volando a altas velocidades.

4. No se tiene en cuenta el movimiento de la atmósfera con respecto a la tierra (viento nulo).

5. Atmósfera sin turbulencia.

6. Se supone que los aviones cuentan con plano de simetría, por lo que los productos de inercia cruzados $I_{yx} = I_{yz} = 0$.

7. No se incluyen los efectos de la inercia por la rotación de motores y hélices.

8. La masa del avión permanece constante durante una determinada maniobra. Esta hipótesis puede ser aceptable en maniobras de corta duración o que impliquen un bajo consumo de combustible, pero en los casos donde no se cumplan estas condiciones se puede llegar a resultados imprecisos.

Con esta información ya es posible establecer las doce ecuaciones en forma Cartesiana, utilizando el sistema de referencia de Ejes Cuerpo (ejes fijos con respecto a la geometría de la aeronave):

$$\begin{aligned} m(\dot{u} + qw - vr) &= T_x + X_A - mg\sin\theta \\ m(\dot{v} + ru - pw) &= T_y + Y_A + mg\cos\theta\sin\phi \\ m(\dot{w} + pv - qu) &= T_z + Z_A + mg\cos\theta\cos\phi \end{aligned} \tag{2.1}$$

De la misma manera que se han establecido tres ecuaciones de la dinámica traslacional (Ecuación 2.1) es posible establecer otras tres sobre la dinámica rotacional (Ecuación 2.2). Adicionalmente, será necesario determinar la actitud del avión en cada instante, y para ello se ha hecho uso de los Ángulos de Euler y la relación de estos con las velocidades angulares $\{p, q, r\}$ (Ecuación 2.3). Para finalizar, se obtienen las ecuaciones cinemáticas (Ecuación 2.5) con las que se completa el estudio del movimiento de la aeronave.

$$\begin{aligned} \dot{p} &= \frac{I_{zz}}{A}L + \frac{I_{xz}}{A}N + \left(\frac{I_{xz}(I_{xx} - I_{yy} + I_{zz})}{A}\right)pq + \left(\frac{I_{zz}(I_{yy} - I_{zz}) - I_{xz}^2}{A}\right)rq \\ \dot{q} &= \frac{M}{I_{yy}} + \frac{I_{zz} - I_{xx}}{I_{yy}}\,pr + \frac{I_{xz}}{I_{yy}}\left(r^2 - p^2\right) \\ \dot{r} &= \frac{I_{xx}}{A}N + \frac{I_{xz}}{A}L + \left(\frac{I_{xx}(I_{xx} - I_{yy}) + I_{xz}^2}{A}\right)pq + \left(\frac{I_{xz}(I_{yy} - I_{xx} - I_{zz})}{A}\right)rq \end{aligned} \tag{2.2}$$

con $A = I_{xx}I_{zz} - I_{xz}^2$.

$$\begin{aligned} p &= \dot{\phi} - \dot{\psi}\sin\theta \\ q &= \dot{\theta}\cos\phi + \dot{\psi}\cos\theta\sin\phi \\ r &= \dot{\psi}\cos\theta\cos\phi - \dot{\theta}\sin\phi \end{aligned} \tag{2.3}$$

También es posible despejar las derivadas de los Ángulos de Euler en función de las velocidades angulares a partir de la Ecuación 2.3. Esta transformación será útil para visualizar el fenómeno del *gimbal lock* en el Capítulo 4.

$$\begin{aligned}\dot{\phi} &= p + q\frac{\sin\phi\sin\theta}{\cos\theta} + r\frac{\cos\phi\sin\theta}{\cos\theta} \\ \dot{\theta} &= q\cos\phi - r\sin\phi \\ \dot{\psi} &= \frac{q\sin\phi + r\cos\phi}{\cos\theta}\end{aligned} \tag{2.4}$$

$$\begin{aligned}\dot{x} &= u\cos\psi\cos\theta + v(\cos\psi\sin\theta\sin\phi - \cos\phi\sin\psi) - w(\sin\theta\cos\phi\cos\psi + \sin\phi\sin\psi) \\ \dot{y} &= u\cos\theta\sin\psi + v(\cos\phi\cos\psi + \sin\theta\sin\phi\sin\psi) + w(-\cos\psi\sin\phi + \cos\phi\sin\theta\sin\psi) \\ \dot{z} &= -u\sin\theta + v\cos\theta\sin\phi + w\cos\theta\cos\phi\end{aligned} \tag{2.5}$$

Siguiendo un procedimiento similar, es posible expresar dichas ecuaciones en Ejes Viento a partir de las transformaciones pertinentes [7]. Este sistema de referencia está ligado a la velocidad aerodinámica instantánea del avión, por lo que solo es fijo en el Centro de Gravedad de la aeronave. Los ángulos aerodinámicos α y β son los que establecen la relación entre los Ejes Viento y los Ejes Cuerpo.

$$\begin{aligned}\alpha &= \arctan\frac{w}{u} \\ \beta &= \arcsin\frac{v}{\sqrt{u^2+v^2+w^2}}\end{aligned} \tag{2.6}$$

Conociendo sus definiciones, se puede realizar la transformación de las ecuaciones.

$$\begin{aligned}m\dot{V} &= T\cos\beta\cos\alpha + X_{A,w} - mg\sin\gamma \\ mVr_w &= -T\sin\beta\cos\alpha + Y_{A,w} + mg\cos\gamma\sin\mu \\ -mVq_w &= -T\sin\alpha + Z_{A,w} + mg\cos\gamma\cos\mu\end{aligned} \tag{2.7}$$

$$\begin{aligned}p_w &= \dot{\mu} - \dot{\chi}\sin\gamma \\ q_w &= \dot{\gamma}\cos\mu + \dot{\chi}\cos\gamma\sin\mu \\ r_w &= \dot{\chi}\cos\gamma\cos\mu - \dot{\gamma}\sin\mu\end{aligned} \tag{2.8}$$

$$\begin{aligned} \dot{x} &= V\cos\chi\cos\gamma \\ \dot{y} &= V\cos\gamma\sin\chi \\ \dot{z} &= -V\sin\gamma \end{aligned} \tag{2.9}$$

La utilización de este sistema de referencia presenta una particularidad con respecto a las ecuaciones de la dinámica rotacional, ya que, si se quisiera realizar la transformación a Ejes Viento, el valor de las inercias (definidas en Ejes Cuerpo) pasaría a depender del ángulo de ataque. Esta dependencia implicaría incluir las derivadas temporales de las inercias en las ecuaciones, complicando el sistema.

Es por eso por lo que se utilizan las ecuaciones de momentos para Ejes Cuerpo definidas en la Ecuación 2.2, quedando, por tanto, una cuestión pendiente: establecer una relación entre las ecuaciones en Ejes Cuerpo y Ejes Viento. Para ello se incluyen un conjunto de tres ecuaciones adicionales (Ecuación 2.10) que relacionan las velocidades angulares en Ejes Cuerpo y en Ejes Viento.

$$\begin{aligned} p_w &= \cos\beta(p\cos\alpha + r\sin\alpha) + \sin\beta(q - \dot{\alpha}) \\ q_w &= -\sin\beta(p\cos\alpha + r\sin\alpha) + \cos\beta(q - \dot{\alpha}) \\ r_w &= r\cos\alpha - p\sin\alpha + \dot{\beta} \end{aligned} \tag{2.10}$$

2.2.2. Modelo de empuje

Para el cálculo del empuje se establecen una serie de modelos matemáticos, a partir de los que se deberá evaluar cuál es el que mejor se ajusta al tipo de aeronave que se esté analizando. Al tratarse de modelos es habitual encontrase con limitaciones en su uso, como, por ejemplo, que no se tienen en cuenta los efectos del posquemador. Algunos de estos modelos implican una dependencia de parámetros en función del tipo de sistema propulsivo o de la zona de la atmósfera en la que se esté volando. En la Tabla 2.2.1 aparecen recogidas dichas dependencias para los modelos incluidos en el simulador y una breve descripción de los mismos.

Para la validación del código de la interfaz gráfica en el Capítulo 4 se va a comparar lo obtenido mediante la GUI con los resultados de L. Villanueva, por lo que se utilizará el mismo avión para la simulación.

Modelo	Dependencia del modelo	Descripción
J-C Wanner	$T = T(T_0, z, V, k_f, \lambda_f, \delta_P)$	Utilizado en turboprops, motores a reacción de flujo simple y ramjets. Varía en función del tipo de aeronave.
Aérospatiale	$T = T(T_0, z, M, \delta_P)$	Utilizado en motores con tasa de derivación entre 5 y 8.
J. Mattingly	$T = T(T_0, z, M, \delta_P)$	Utilizado en motores con tasa de derivación entre 5 y 8 para velocidades inferiores a Mach 0.9.
D. Hull	$T = T(T_0, z, a, \delta_P)$	Aplicable a turbojets y turbofans de baja tasa de derivación. El empuje depende de la capa de la atmósfera a la que se vuele.
Howe	$T = T(T_0, z, M, BPR, \delta_P)$	Usado en turbofans para un amplio rango de la tasa de derivación.
Motores turbofan	$T = T(T_0, z, M, BPR, n, k_1, k_2, k_3, k_4\delta_P)$	Modelo genérico para los motores turbofan. Los parámetros son función de altitud y número de Mach.

Tabla 2.2.1: Modelos matemáticos de empuje del simulador.

A continuación se desarrolla el modelo de D. Hull, pues es el utilizado por la autora para calcular el empuje en el avión de estudio. En la Ecuación 2.11 se muestra la expresión del empuje para este modelo.

$$T = T_0[\sigma(z)]^a \delta_P \tag{2.11}$$

Como se ha comentado anteriormente, la expresión de este modelo depende de la capa de la atmósfera en la que se vuele (a través de la densidad relativa y del parámetro a) y del tipo de motor, siendo la equivalencia la recogida en la Tabla 2.2.2.

Turbojet	a
Troposfera (z<11 000m)	1.2
Estratosfera (z>11 000m)	1
Turbofan	**a**
Troposfera (z<11 000m)	1
Estratosfera (z>11 000m)	1

Tabla 2.2.2: Valor de a para el modelo de D.Hull

2.2.3. Modelo aerodinámico

Los únicos términos que faltan por describir de las *Ecuaciones de Bryan*, recogidas en la Sección 2.2.1, son los de las fuerzas aerodinámicas. Para su obtención se ha seguido un proceso de linealización, se aplicaron primero pequeñas perturbaciones y se consiguieron después las fuerzas y momentos a partir del desarrollo en serie de Taylor. De esa manera, es posible obtener las derivadas aerodinámicas del modelo, las cuales se consideran constantes a pesar de que algunas de ellas dependen de parámetros como las condiciones de vuelo. Este procedimiento no es estrictamente necesario, pero en este caso se considera que se trabaja con valores de velocidad y ángulos aerodinámicos dentro de la zona en la que la aerodinámica se comporta de forma lineal.

Para representar dichas fuerzas y momentos es conveniente hacer una distinción entre aquellos correspondientes a la dinámica longitudinal y aquellos de la dinámica lateral-direccional pues, aunque estén acopladas en las ecuaciones, en ocasiones es posible hacer simplificaciones y estudiar su respuesta por separado.

En la Ecuación 2.12 se muestran los coeficientes aerodinámicos de las fuerzas principales de la dinámica longitudinal e, igualmente, el momento asociado a dicha dinámica. Destacar que para la resistencia se toma la simplificación de polar parabólica de coeficientes constantes.

$$\begin{aligned} C_D &= C_{D0} + K\ C_L^2 \\ C_L &= C_{L0} + C_{L\alpha}\ \alpha + C_{L\delta E}\ \delta_E + C_{Lq}\frac{c_w}{2V}\ q + C_{L\dot{\alpha}}\frac{c_w}{2V}\ \dot{\alpha} \\ C_M &= C_{M0} + C_{M\alpha}\ \alpha + C_{M\delta E}\ \delta_E + C_{Mq}\frac{c_w}{2V}\ q + C_{M\dot{\alpha}}\frac{c_w}{2V}\ \dot{\alpha} \end{aligned} \tag{2.12}$$

Dando lugar a las siguientes fuerzas y momento:

$$L = \frac{1}{2}\rho(z)V^2 S_w C_L \qquad D = \frac{1}{2}\rho(z)V^2 S_w C_D \qquad M = \frac{1}{2}\rho(z)V^2 S_w c_w C_M \tag{2.13}$$

De un modo similar es posible realizar dicha descomposición de los coeficientes de la dinámica lateral-direccional (Ecuación 2.14). A diferencia del caso anterior, ahora se tiene una fuerza aerodinámica y dos momentos.

$$
\begin{aligned}
C_Y &= C_{Y\beta}\ \beta + C_{Y\dot{\beta}}\frac{b_w}{2V}\ \dot{\beta} + C_{Yp}\frac{b_w}{2V}\ p + C_{Yr}\frac{b_w}{2V}\ r + C_{Y\delta A}\ \delta_A + C_{Y\delta R}\ \delta_R \\
C_l &= C_{l\beta}\ \beta + C_{l\dot{\beta}}\frac{b_w}{2V}\ \dot{\beta} + C_{lp}\frac{b_w}{2V}\ p + C_{lr}\frac{b_w}{2V}\ r + C_{l\delta A}\ \delta_A + C_{l\delta R}\ \delta_R \\
C_N &= C_{N\beta}\ \beta + C_{N\dot{\beta}}\frac{b_w}{2V}\ \dot{\beta} + C_{Np}\frac{b_w}{2V}\ p + C_{Nr}\frac{b_w}{2V}\ r + C_{N\delta A}\ \delta_A + C_{N\delta R}\ \delta_R
\end{aligned}
\tag{2.14}
$$

La fuerza y momentos de la dinámica lateral-direccional quedan:

$$
Y = \frac{1}{2}\rho(z)V^2 S_w C_Y \qquad L = \frac{1}{2}\rho(z)V^2 S_w b_w C_l \qquad N = \frac{1}{2}\rho(z)V^2 S_w b_w C_N \tag{2.15}
$$

Una vez calculadas las fuerzas y los momentos en Ejes Viento a partir de las derivadas aerodinámicas, será necesario realizar las transformaciones al sistema de referencia pertinentes [7] para poder incluir dichos términos en la Ecuación 2.1 y 2.7.

2.2.4. Factores de carga

Los factores de carga son fundamentales para entender las tensiones estructurales y la maniobrabilidad de una aeronave. En los procesos de diseño de aeronaves es crucial tener en cuenta los factores esperados para conocer las cargas estructurales que va a soportar la aeronave durante todas las fases de operación. Estas cargas suelen venir expresadas en términos de la aceleración debida a la gravedad (g).

En la versión del simulador desarrollada en este trabajo se ha añadido la variación temporal de estos factores durante la maniobra simulada debido a su importancia a la hora de limitar las actuaciones de la aeronave. En la Ecuación 2.16 se muestra la expresión de los factores de carga en Ejes Cuerpo, cuyo cálculo se ha efectuado al dividir la fuerza aerodinámica correspondiente en cada eje entre el peso de la aeronave. Cabe destacar que, debido a la orientación en la que están definidos los ejes del sistema de referencia, se ha creído conveniente expresar los factores de carga en valor absoluto, pero esto puede dar lugar a confusiones cuando los factores tomen realmente valores negativos. Por ello, se deberá prestar gran atención al comportamiento de la aeronave y analizar detenidamente los resultados obtenidos, con tal de evitar valoraciones erróneas.

$$
\begin{aligned}
n_x &= \left|\frac{X_A}{W}\right| = \left|\frac{1}{2W}\rho(z)S_w\left(u^2+v^2+w^2\right)\left(-C_D\cos\alpha\cos\beta + C_L\sin\alpha + C_{Y,w}\cos\alpha\sin\beta\right)\right| \\
n_y &= \left|\frac{Y_A}{W}\right| = \left|\frac{1}{2W}\rho(z)S_w\left(u^2+v^2+w^2\right)\left(-C_D\cos\beta + C_{Y,w}\sin\beta\right)\right| \\
n_z &= \left|\frac{Z_A}{W}\right| = \left|\frac{1}{2W}\rho(z)S_w\left(u^2+v^2+w^2\right)\left(-C_D\sin\alpha\cos\beta - C_L\cos\alpha + C_{Y,w}\sin\alpha\sin\beta\right)\right|
\end{aligned}
\tag{2.16}
$$

donde $CY_w = -CD\sin\beta - CY\cos\beta$.

Los factores de carga normalmente suelen proporcionarse en Ejes Cuerpo, ya que se utilizan para el cálculo estructural, pero en la asignatura de *Mecánica del Vuelo* muchas veces se han definido a partir de las fuerzas aerodinámicas de sustentación y resistencia, por lo que también se incluirán los factores de carga con esas definiciones. Por tanto, la expresión de los factores de carga para Ejes Viento es la mostrada en la Ecuación 2.17

$$
\begin{aligned}
n_{x,w} &= \left|\frac{1}{2W}\rho(z)S_wV^2C_{X,w}\right| = \left|\frac{1}{2W}\rho(z)S_wV^2(-C_D\cos\beta + C_Y\sin\beta)\right| \\
n_{y,w} &= \left|\frac{1}{2W}\rho(z)S_wV^2C_{Y,w}\right| = \left|\frac{1}{2W}\rho(z)S_wV^2(-C_D\sin\beta - C_Y\cos\beta)\right| \\
n_{z,w} &= \left|\frac{1}{2W}\rho(z)S_wV^2C_{Z,w}\right| = \left|\frac{1}{2W}\rho(z)S_wV^2(-C_L)\right|
\end{aligned}
\tag{2.17}
$$

2.3. Documentación del programa

Como se había comentado en el Capítulo 1, el desarrollo de la interfaz dinámica toma como base el trabajo de L. Villanueva y su programación del simulador. En esta sección se pretende recoger y explicar las funciones y archivos desarrollados por la autora que se han implementado en la aplicación del simulador.

Se realizará una distinción entre los archivos y funciones de MATLAB para ayudar al lector a entender las partes que componen el programa.

Funciones integradas en el simulador

El modelo de empuje con el que se realizará el estudio dinámico de la aeronave vendrá dado por la función *thrustmodel.m*, en la que se recogen los diferentes modelos vistos en la Sección 2.2.2 y que el usuario será capaz de elegir para las simulaciones. La expresión genérica se puede ver en la Ecuación 2.18:

$$
[T, T_x, T_y, T_z, T_{x,w}, T_{y,w}, T_{z,w}] = thrustmodel(\delta_P, z, V, M, \alpha, \beta) \tag{2.18}
$$

Esta función permite obtener el módulo del empuje T y sus componentes en coordenadas Cartesianas, tanto en Ejes Cuerpo como en Ejes Viento.

De la misma manera, también se tiene una función de MATLAB que incorpora el modelo aerodinámico lineal y permite calcular los coeficientes aerodinámicos a partir de la velocidad, los ángulos aerodinámicos, las velocidades angulares y los controles aerodinámicos. En este caso el cálculo se lleva a cabo con la función *coefficients.m*, cuya expresión es la mostrada en la Ecuación 2.19.

$$[C_L, C_D, C_Y, C_l, C_M, C_N] = coefficients(V, \alpha, \dot{\alpha}, p, q, r, \beta, \dot{\beta}, \delta_E, \delta_A, \delta_R) \tag{2.19}$$

Una vez calculadas las componentes del empuje y los coeficientes aerodinámicos se deben incorporar dichos valores a las *Ecuaciones de Bryan* y resolver el sistema mediante el método adecuado. Puesto que se está trabajando con tres maneras distintas de orientar la aeronave en el espacio, existe una función que relaciona las ecuaciones con cada uno de estos sistemas de referencia.

$$\begin{gathered} \dot{F} = BRYANBODY(t, F) \\ \dot{F} = BRYANWIND(t, F) \\ \dot{F} = BRYANBODYQUATERNIONS(t, F) \end{gathered} \tag{2.20}$$

Como es evidente, si se desea estudiar la evolución temporal de la dinámica de la aeronave la respuesta obtenida debe ser función del tiempo. Adicionalmente, estas funciones dependen del vector de estado F, el cual recoge las variables de estado de cada uno de los marcos de referencia.

$$\begin{gathered} F_{body} = [u, v, w, p, q, r, \phi, \theta, \psi, x, y, z, \dot{m}_f] \\ F_{wind} = [V, \beta, \alpha, p, q, r, \mu, \gamma, \chi, x, y, x, \dot{m}_f] \\ F_{quaternions} = [u, v, w, p, q, r, q_0, q_1, q_2, q_3, x, y, z, \dot{m}_f] \end{gathered}$$

También se ha añadido una ecuación extra (Ecuación 2.21) que permite calcular el consumo de combustible en la maniobra, de ahí que aparezca una variable que no se acostumbra a ver en las *Ecuaciones de Bryan*.

$$\dot{m}_f = -C_e\, T \tag{2.21}$$

A continuación, se exponen los archivos propiamente dichos en los que se implementan estas funciones y otros que también son necesarios para la correcta operación del simulador.

Archivos de programa del simulador

En primer lugar, existe un archivo de MATLAB que recoge todos los parámetros geométricos, másicos, inerciales y aerodinámicos de la aeronave de estudio. El archivo viene nombrado como *AircraftParameters.m* y está pensado para que el estudiantado pueda modificar los parámetros incluidos en él con los del avión cuya dinámica desee estudiar.

En segundo lugar, existen una serie de archivos que se crearon con la finalidad de incluir en ellos las condiciones iniciales y finales de la maniobra a simular, así como la variación de los controles en un momento determinado. De nuevo, al tener tres posibles sistemas de referencia con los que trabajar, se creó un archivo para dichos sistemas bajo los nombres de *InitialConditionsBody.m*, *InitialConditionsWind.m* e *InitialConditions-Quaternions.m*. A partir de las condiciones de vuelo, los valores en el equilibrio de los ángulos aerodinámicos y la maniobra seleccionada, es posible calcular los valores iniciales de todas las variables del vector de estado y comenzar así la simulación.

Todas estas funciones y archivos se integran en un último script llamado *FLIGHTSIMULATOR.m*. En él se cargan todos los datos previamente obtenidos y se procede a la simulación propiamente dicha mediante la integración de las ecuaciones, recogidas en las funciones ya mencionadas, a través de un solver numérico. En la Figura 2.3.1 se representa un diagrama de flujo sobre los procesos que sigue el simulador para ayudar a la comprensión de su funcionamiento.

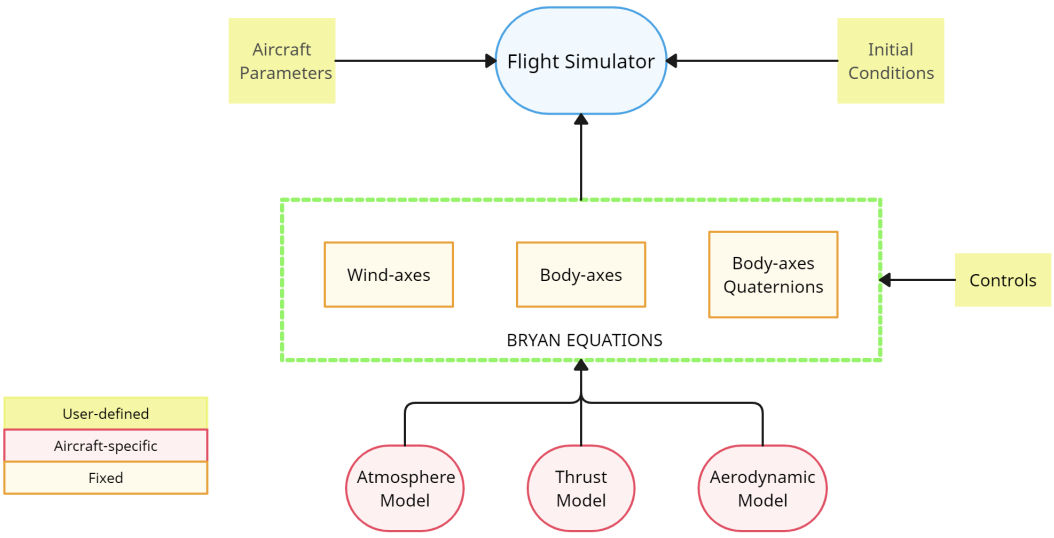


Figura 2.3.1: Diagrama de funcionamiento del simulador. [2]

2.4. Documentación de la aplicación

Para la elaboración de la interfaz dinámica se ha optado por crear una aplicación en MATLAB con la herramienta *App Designer*. Esta funcionalidad recoge las dos tareas principales en la creación de una aplicación: la distribución de los componentes visuales de una interfaz gráfica de usuario (GUI) y la programación del comportamiento de la app.

La conexión entre los elementos y su función se lleva a cabo mediante *callbacks*, que son las acciones por las que se llama al código implementado en esos elementos y hacen que el funcionamiento de la aplicación sea orgánico. Estos *callbacks* pueden ser: modificar un valor numérico, pulsar un botón, abrir un desplegable o seleccionar una opción de una lista. Así se consigue una gran libertad a la hora de crear las interacciones aplicación-usuario.

A continuación se procederá a exponer el diseño final de la aplicación junto con el funcionamiento de sus elementos.

La aplicación cuenta con 4 pestañas diferentes en las que se deberá interactuar para preparar la simulación de la respuesta temporal de la aeronave y realizar posteriormente la propia representación de las variables. Estas pestañas se describirán más detenidamente en las siguientes subsecciones de forma individual. El código con el que se han programado todos los *callbacks* y las interacciones del programa se encuentra recogido en el Anexo A.

Parámetros aeronave

La primera ventana que aparece al iniciar la aplicación es la que va a servir para mostrar los parámetros principales de la aeronave (Figura 2.4.1). La idea del funcionamiento de esta pestaña es que el usuario sea capaz de elegir el avión que desea estudiar mediante un desplegable que, al iniciarse el programa, examine el directorio en busca de todos los archivos que recogen los parámetros de las aeronaves descritas en la Sección 5. La ventaja de este sistema es que el usuario puede crear sus propios archivos de parámetros e incluirlos en el directorio para que aparezcan en el desplegable al abrir la aplicación.

Además, se ha incluido un desplegable en el modelo de empuje con el que se puede seleccionar el modelo de interés para el caso que se esté resolviendo, o comparar los resultados para diversos modelos. En caso de que haya algún parámetro del modelo que no se haya seleccionado correctametente la aplicación notificará de que existe un error en los datos.

Justo después de haber seleccionado el avión, basta con pulsar el botón de *"Cargar datos del avión"* para que MATLAB lea las variables recogidas en el archivo correspondiente y muestre su valor en los campos de edición numéricos. De esta manera permite la previsualización de los datos de la aeronave que se va a estudiar.

Figura 2.4.1: Ventana *Parámetros aeronave* de la app.

Derivadas aerodinámicas

Si se presta atención a la ventana representada en el apartado anterior, se comprobará que no se incluyen los datos correspondientes al modelo aerodinámico de la aeronave. Estos valores aparecen en la segunda ventana disponible en la aplicación (Figura 2.4.2) y se encuentran divididos en dos grupos: aquellos propios de la dinámica longitudinal y los que pertenecen a la dinámica lateral-direccional.

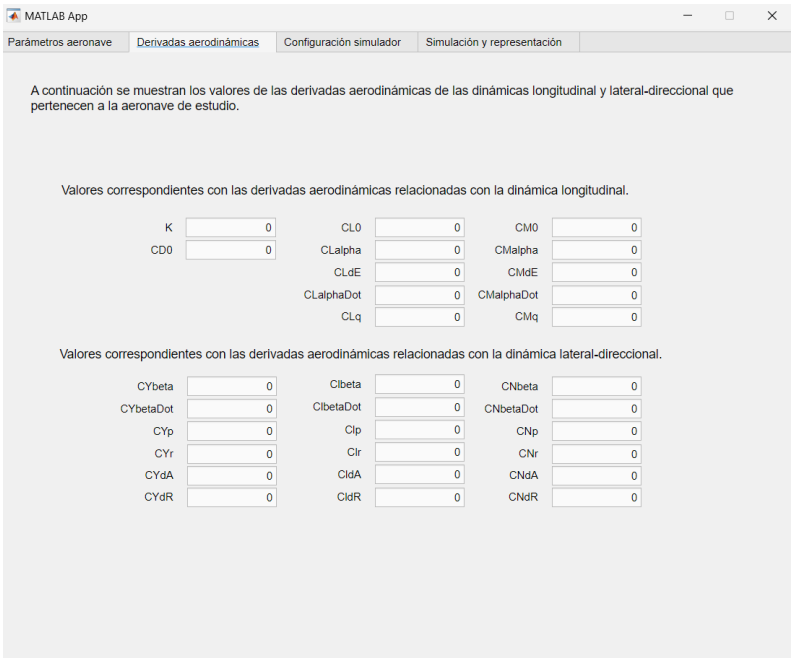


Figura 2.4.2: Ventana *Derivadas aerodinámicas* de la app.

En esta ventana sí que se ha habilitado la posibilidad de modificar los valores una vez se hayan cargado los datos de la aeronave, ya que puede ser interesante comparar la dinámica del avión en función del signo o valor de sus derivadas. El poder visualizar los datos con anterioridad puede resultar útil para comprobar que no se ha cometido ningún error en la incorporación de los datos en el archivo de la aeronave, evitando así realizar cálculos incorrectos.

Configuración simulador

Una vez se tienen cargados todos los datos relacionados con la aeronave es el turno de añadir los parámetros que definirán la maniobra que se va a estudiar. Para ello el usuario deberá introducir los valores de las condiciones de vuelo (altitud y velocidad) y de los ángulos aerodinámicos en el instante inicial. También es posible añadir un valor inicial de masa de combustible consumida, permitiendo, por ejemplo, calcular el consumo total de combustible de varias simulaciones.

Con las condiciones de vuelo iniciales ya especificadas, se precisa elegir en qué sistema de referencia se van a resolver las *Ecuaciones de Bryan* e imponer los valores iniciales de todas las variables que componen el vector de estado definido en la Sección 2.3. También es conveniente indicar la maniobra con la que se pretende arrancar la simulación, ya que el cálculo de determinadas variables en el inicio depende del movimiento a analizar. Igualmente, siempre es posible comenzar la simulación con la aeronave equilibrada en el plano vertical y cambiar los valores de los controles a posteriori.

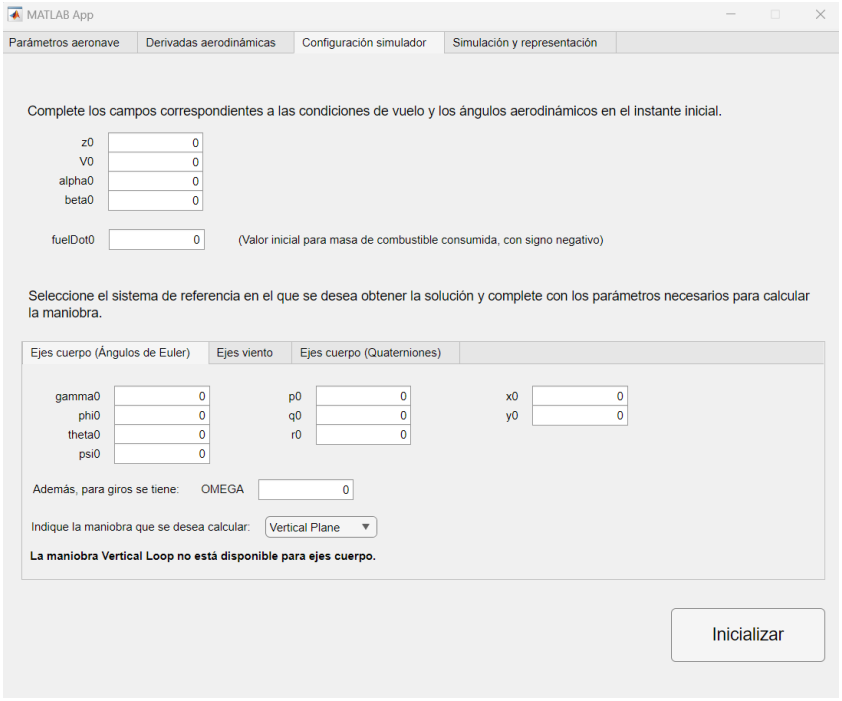


Figura 2.4.3: Ventana *Configuración simulador* de la app.

Todos estos valores deben incorporarse en la pestaña representada en la Figura 2.4.3, para posteriormente calcular el valor del resto de variables (como las componentes lineales de la velocidad) a partir de los datos proporcionados. Este proceso se lleva a cabo al pulsar el botón *"Inicializar"*, el cual, además, devuelve una ventana flotante indicando que se ha terminado el cálculo.

Simulación y representación

En la parte final de la aplicación es donde se encuentran los elementos necesarios para realizar la simulación propiamente dicha, en la Figura 2.4.4 se representan estos elementos. El recuadro superior es en el que se deben establecer los valores de los controles al comienzo de la simulación y donde se pueden modificar una vez se ha iniciado la representación de las variables. Esta representación se elige mediante el cuadro en el que aparece un listado de todas las posibilidades de visualización, siendo únicamente necesario seleccionar con el cursor la variable de interés. Tras elegirse los parámetros a graficar, se debe pulsar el botón *"Start"* para iniciar la simulación y el programa empezará a dibujar la solución de las ecuaciones en el gráfico adjunto. De manera análoga, es posible parar la simulación con el botón *"Stop"*, con lo que además se modifica la interacción de estos botones. Al pararse la simulación, el texto contenido en los botones pasa a mostrarse como *"Continue"* y *"Restart"*, para poder reanudar la simulación desde ese punto o para borrar la solución e iniciar así una nueva.

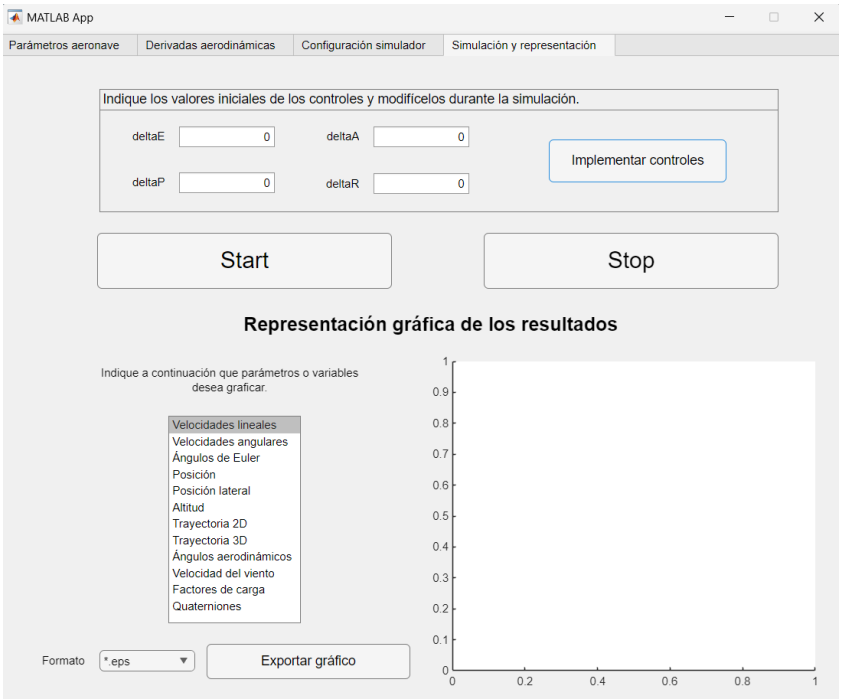


Figura 2.4.4: Ventana *Simulación y representación* de la app.

Adicionalmente, al pulsar el botón de *"Stop"* se exportan los datos de la solución y el vector de tiempos al espacio de trabajo de MATLAB. Con esto es posible realizar tareas de posprocesado una vez se ha simulado la operación de la aeronave. Al borrarse la solución representada en la gráfica pulsando el botón *"Restart"*, también se borra el contenido de las variables exportadas en el espacio de trabajo. Para poder exportar directamente la gráfica que aparece en la aplicación, se ha creado otro botón que permite ubicar la imagen que se está visualizando en el directorio que se quiera, pudiendo además elegir el formato en el que hacer la exportación con el desplegable de la izquierda.

Tras haber explicado el funcionamiento de la aplicación y los elementos que la componen, es posible validar la implementación del código en la GUI mediante la comparación de los resultados del presente trabajo con lo obtenido por L. Villanueva. Sin embargo, antes de realizar la validación se va a estudiar la posibilidad de utilizar un método numérico mejor para resolver el sistema de ecuaciones.

Capítulo 3

Método numérico

3.1. Comparación métodos numéricos

Tal y como se indicaba en los objetivos del trabajo, se ha valorado la posibilidad de usar un solver numérico distinto al *ode45* utilizado en el trabajo de L. Villanueva [2] para resolver las *Ecuaciones de Bryan*. La finalidad de este estudio es, principalmente, encontrar un solver que mantenga, o incluso mejore la precisión de los resultados disminuyendo el tiempo de cálculo. Para ello se ha resuelto el conjunto completo de las *Ecuaciones de Bryan* con una serie de maniobras en el simulador para cada uno de los sistemas de referencia, consiguiendo de esta manera comparar los resultados de todas las posibilidades.

Para realizar dicha comparación se estudiará, por una parte, el tiempo de cálculo de cada solver en cada uno de los casos de resolución y, además, se calculará el error o la diferencia de los nuevos solvers con respecto al *ode45*, el cual se ha utilizado como referencia.

Existen muchas formas [8] de calcular el error entre una variable o función de referencia y la variable/función con la que se quiere comparar. En este capítulo se pretende calcular el error relativo que existe entre dos curvas, y para ello se ha optado por usar el *Root Mean Square Error (RMSE)* relativo. La formulación para obtener este error es la siguiente:

$$RMSE_{rel} = \sqrt{mean\left(\frac{(y_0 - y_1)^2}{y_0^2}\right)}$$

donde y_0 representa la referencia (en este caso la solución del solver *ode45*) e y_1 la curva con la que se pretende comparar para obtener el error.

Los métodos utilizados proceden tanto de la documentación de MATLAB como de programación externa, a partir de lo que se ha estudiado en otras asignaturas de cursos previos [9]. Los métodos que son propios de MATLAB comparten una característica en común, y es que su uso se reduce a problemas no rígidos, es decir, aquellos en los que la solución del *ode* no tiene dos componentes consecutivas que varían drásticamente en escalas de tiempo.

Para algunos de los solvers que se han programado es necesario especificar el paso de integración, en cuyo caso se debe llegar a un compromiso entre la rapidez de cálculo y la precisión de los resultados. Esto es porque puede suceder que sea muy rápido pero la solución obtenida difiera mucho de la referencia, o viceversa. Es por eso que se han utilizado diferentes pasos de integración dependiendo del método, siendo estos valores los recomendados según la bibliografía [9] para obtener un buen equilibrio entre precisión y tiempo de cálculo.

A continuación se muestran los métodos numéricos que se han probado en este estudio junto con una breve descripción, características de dichos métodos, una primera estimación de la precisión que se puede esperar de ellos [10][11] y lo sencillo o difícil que puede llegar a ser su programación. Este último aspecto se ha evaluado principalmente por el tiempo que se ha tardado en programar y la compresión del funcionamiento del código, siendo de esta manera una valoración más cualitativa.

Método	Descripción	Características	Precisión	Programación
ode45	Es una implementación del método explícito de Runge-Kutta basado en el par (4,5) de Dormand y Prince.	Usa una interpolación para resolver valores de la solución en 4 puntos igualmente espaciados en cada paso.	Media-Alta	MATLAB
Euler	Es el más simple de todos los métodos, se obtiene considerando el desarrollo de Taylor de primer orden.	Método con posibilidad de especificar el paso de integración. Es un método limitado pero útil para una primera aproximación.	Media-Baja	Sencilla
Euler modificado	Runge-Kutta explícito de orden2. Es necesario evaluar la función dos veces en cada paso de integración.	Requiere más operaciones que en el método de Euler.	Media	Sencilla
ode78	Implementación del par 8(7) de Verner para un Runge-Kuta "más eficiente". Método explícito de paso único.	Puede ser más eficiente que el *ode45* en problemas no rígidos que sean diferenciables.	Alta	MATLAB
ode89	Implementación del par 9(8) de Verner para un Runge-Kuta "más robusto".	Puede ser más eficiente que el *ode78* para integraciones de intervalos grandes o con grandes tolerancias.	Alta	MATLAB
ode113	Es un método de Adams-Bashfoth-Moulton multivariable y multipaso (VSVO) de orden 1 hasta 13.	Puede ser útil con tolerancias estrictas o con funciones caras de evaluar.	Alta	MATLAB
Heun	Runge-Kuta explícito de orden 2 en el que también se puede especifiar el paso de integración.	Misma estructura que el método de *Euler modificado* pero los parámetros numéricos del algoritmo son diferentes.	Media	Sencilla
RK4	Método estándar Runge-Kutta de orden 4. Requiere de 4 evaluaciones por paso.	Mantiene un buen balance entre el coste del método y la precisión que se consigue, por lo que es muy conocido.	Media-Alta	Sencilla
P-C ABM 4	Método predictor-corrector ABM de 4 pasos. Se denomina predictor-corrector ya que combina una técnica explícita con una implícita.	Se trata de un método multipaso ya que la aproximación obtenida tiene en cuenta aproximaciones calculadas anteriormente.	Media-Alta	Sencilla-Intermedia

3.1.1. Resultados del tiempo de cálculo.

Primero se comparará el tiempo que le ha llevado a todos los solvers resolver las ecuaciones en diferentes maniobras de la aeronave.

Para una mejor representación de los resultados se nombrarán las maniobras calculadas mediante siglas, siendo la correspondencia:

- Equilibro en el plano vertical (VP)
- Loop vertical (VL)
- Vuelo de planeo (GF)
- Recuperación en picado (DR)
- Giro en ascenso (AT)
- Giro horizontal (HT)

Para los métodos de Euler, Euler modificado y Heun se ha usado un paso de $h = 0.1$, y para los métodos Runge-Kutta de orden 4 y el predictor-corrector se ha utilizado un paso de $h = 0.2$.

Además, se representarán en 3 bloques distintos según las maniobras que se pueden calcular en cada uno de los sistemas de referencia disponibles: Ejes Cuerpo (ángulos de Euler), Ejes Viento y Ejes Cuerpo (quaterniones).

Método	VP	GF	DR	AT	HT
ode45	5.61	14.01	87.85	16.81	6.26
Euler	9.47	9.48	9.54	9.62	9.64
Euler mod.	18.65	18.82	18.81	19.00	19.27
ode78	8.10	16.30	41.07	17.57	8.86
ode89	12.84	24.20	51.16	24.94	13.33
ode113	4.26	6.80	14.12	7.17	6.60
Heun	18.67	18.77	19.14	19.07	19.55
RK4	18.60	18.76	18.76	18.97	18.65
P-C ABM 4	9.54	9.34	9.50	9.70	9.37

(a) Ángulos de Euler

VP	VL
5.68	93.13
9.59	9.52
18.93	18.89
8.21	43.52
12.93	55.82
4.36	16.64
18.96	18.99
18.63	19.00
9.42	9.73

(b) Quaterniones

Tabla 3.1.1: Resultados del tiempo de cálculo en segundos para Ejes Cuerpo

Método	VP	VL	GF	DR	AT	HT
ode45	4.88	56.36	7.32	60.67	13.26	5.51
Euler	7.55	7.52	7.45	7.43	7.45	7.57
Euler mod.	14.75	14.96	14.87	14.73	14.72	15.05
ode78	6.01	27.68	9.69	29.22	12.90	6.91
ode89	9.59	34.98	16.58	37.51	19.98	10.69
ode113	3.65	10.33	5.23	10.62	5.97	5.55
Heun	14.75	14.71	14.68	15.05	14.90	14.98
RK4	14.67	14.62	14.67	14.85	14.84	14.72
P-C ABM 4	7.47	7.38	7.47	7.47	7.46	7.65

Tabla 3.1.2: Resultados del tiempo de cálculo en segundos en Ejes Viento.

Las celdas resaltadas en gris de las Tablas 3.1.1 y 3.1.2, donde se muestran los resultados, hacen referencia al método que menos tiempo de cálculo ha requerido para resolver el sistema de ecuaciones diferenciales en esa maniobra concreta. Comparando todos los valores se puede ver como, en la mayoría de los casos, el *ode113* es el que obtiene mejores resultados. También destacan el *ode45*, el método de Euler y el método predictor-corrector de paso 4.

El siguiente paso es comprobar si los resultados logrados en este estudio se corresponden a su vez con un error muy bajo al compararlo con el *ode45*.

3.1.2. Resultados del error relativo.

En el estudio se han obtenido los errores de las 12 variables propias del sistema que se está resolviendo en todas las maniobras y sistemas de referencia, sin embargo, debido a la gran cantidad de información recogida, solo se mostrarán los resultados para un par de variables en cada caso.

Para elegir estas variables se ha observado cuáles de ellas se modifican durante todas las maniobras, ya que en muchos casos hay variables que no cambian de valor y por tanto el error es nulo. También se ha intentado seleccionar una variable común para todos los casos y otra que sea representativa del sistema de referencia en el que se esté realizando la comparación.

De la misma forma que con los tiempos de cálculo, se usarán diferentes tablas para mostrar el error relativo, en porcentaje, según el sistema de referencia utilizado y/o maniobra particular.

En la Tabla 3.1.3 se muestra el error para el ángulo de asiento longitudinal (θ) y para la altitud (z) en Ejes Cuerpo (Ángulos de Euler). Nuevamente se ha marcado en gris la celda correspondiente al método cuyo error relativo sea menor con respecto a la referencia.

Método	VP		GF		DR		AT		HT	
	θ	z	θ	z	θ	z	θ	z	θ	z
ode45	-	-	-	-	-	-	-	-	-	-
Euler	$1.2 \cdot 10^{-5}$	$5.6 \cdot 10^{-7}$	1.67	0.069	16.51	2.62	0.33	$1.5 \cdot 10^{-2}$	$1.1 \cdot 10^{-4}$	$1.9 \cdot 10^{-6}$
Euler mod.	$1.8 \cdot 10^{-6}$	$7.0 \cdot 10^{-9}$	$2.9 \cdot 10^{-2}$	$5.3 \cdot 10^{-4}$	2.33	0.11	$4.0 \cdot 10^{-3}$	$2.9 \cdot 10^{-4}$	$1.6 \cdot 10^{-5}$	$4.5 \cdot 10^{-8}$
ode78	$1.6 \cdot 10^{-8}$	$1.1 \cdot 10^{-11}$	$3.2 \cdot 10^{-8}$	$9.3 \cdot 10^{-10}$	$1.7 \cdot 10^{-9}$	$1.7 \cdot 10^{-10}$	$3.2 \cdot 10^{-9}$	$1.5 \cdot 10^{-11}$	$1.6 \cdot 10^{-8}$	$1.3 \cdot 10^{-11}$
ode89	$1.1 \cdot 10^{-8}$	$7.5 \cdot 10^{-12}$	$2.2 \cdot 10^{-8}$	$9.2 \cdot 10^{-10}$	$1.4 \cdot 10^{-9}$	$1.7 \cdot 10^{-10}$	$1.2 \cdot 10^{-9}$	$1.5 \cdot 10^{-11}$	$1.5 \cdot 10^{-8}$	$1.2 \cdot 10^{-11}$
ode113	$2.9 \cdot 10^{-8}$	$1.1 \cdot 10^{-9}$	$1.3 \cdot 10^{-6}$	$4.7 \cdot 10^{-8}$	$1.1 \cdot 10^{-9}$	$4.3 \cdot 10^{-9}$	$7.1 \cdot 10^{-8}$	$2.9 \cdot 10^{-9}$	$1.8 \cdot 10^{-8}$	$1.4 \cdot 10^{-11}$
Heun	$1.8 \cdot 10^{-6}$	$7.0 \cdot 10^{-9}$	$2.9 \cdot 10^{-2}$	$5.3 \cdot 10^{-4}$	2.29	0.11	$4.0 \cdot 10^{-3}$	$3.0 \cdot 10^{-4}$	$1.6 \cdot 10^{-5}$	$4.5 \cdot 10^{-8}$
RK4	$1.8 \cdot 10^{-6}$	$6.7 \cdot 10^{-9}$	$2.8 \cdot 10^{-2}$	$4.9 \cdot 10^{-4}$	1.79	0.10	$3.8 \cdot 10^{-3}$	$2.9 \cdot 10^{-4}$	$1.5 \cdot 10^{-5}$	$4.4 \cdot 10^{-8}$
P-C ABM 4	$1.8 \cdot 10^{-6}$	$6.7 \cdot 10^{-9}$	$2.8 \cdot 10^{-2}$	$4.9 \cdot 10^{-4}$	1.97	0.10	$3.8 \cdot 10^{-3}$	$2.9 \cdot 10^{-4}$	$1.6 \cdot 10^{-5}$	$4.4 \cdot 10^{-8}$

Tabla 3.1.3: Resultados del error relativo [%] en Ejes Cuerpo (ángulos de Euler).

El siguiente sistema de referencia a estudiar es el de Ejes Cuerpo (quaterniones). Para este análisis las variables a comparar serán la velocidad lineal en el eje X (u) y, de nuevo, la altitud (z) en las dos maniobras posibles. Los resultados aparecen representados en la Tabla 3.1.4.

Finalmente, en la Tabla 3.1.5 se realiza la comparación correspondiente a los resultados obtenidos al resolver las ecuaciones de Bryan en Ejes Viento. En este caso se muestra el error relativo obtenido en las variables del ángulo de ataque (α) y la altitud (z).

Método	VP		VL	
	u	z	u	z
ode45	-	-	-	-
Euler	$5.4 \cdot 10^{-7}$	$5.6 \cdot 10^{-7}$	23.95	21.85
Euler mod.	$9.8 \cdot 10^{-9}$	$7.0 \cdot 10^{-9}$	0.27	0.15
ode78	$6.5 \cdot 10^{-11}$	$1.1 \cdot 10^{-11}$	$1.7 \cdot 10^{-7}$	$1.2 \cdot 10^{-7}$
ode89	$4.3 \cdot 10^{-11}$	$6.9 \cdot 10^{-12}$	$2.5 \cdot 10^{-8}$	$2.3 \cdot 10^{-8}$
ode113	$1.5 \cdot 10^{-9}$	$1.6 \cdot 10^{-9}$	$7.0 \cdot 10^{-8}$	$6.0 \cdot 10^{-8}$
Heun	$9.8 \cdot 10^{-9}$	$7.0 \cdot 10^{-9}$	0.21	$7.5 \cdot 10^{-2}$
RK4	$9.5 \cdot 10^{-9}$	$6.7 \cdot 10^{-9}$	0.24	0.12
P-C ABM 4	$9.6 \cdot 10^{-9}$	$6.7 \cdot 10^{-9}$	0.42	0.30

Tabla 3.1.4: Resultados del error relativo [%] en Ejes Cuerpo (quaterniones).

Método	VP		VL		GF	
	α	z	α	z	α	z
ode45	-	-	-	-	-	-
Euler	$5.4 \cdot 10^{-7}$	$5.6 \cdot 10^{-7}$	2.92	0.79	$2.6 \cdot 10^{-3}$	$6.9 \cdot 10^{-2}$
Euler mod.	$7.5 \cdot 10^{-9}$	$7.0 \cdot 10^{-9}$	1.14	$7.5 \cdot 10^{-2}$	$8.2 \cdot 10^{-4}$	$5.3 \cdot 10^{-4}$
ode78	$9.9 \cdot 10^{-11}$	$4.6 \cdot 10^{-11}$	$7.8 \cdot 10^{-9}$	$3.3 \cdot 10^{-10}$	$3.9 \cdot 10^{-8}$	$6.2 \cdot 10^{-9}$
ode89	$7.8 \cdot 10^{-11}$	$4.2 \cdot 10^{-11}$	$2.9 \cdot 10^{-9}$	$3.4 \cdot 10^{-10}$	$1.2 \cdot 10^{-8}$	$6.2 \cdot 10^{-9}$
ode113	$8.2 \cdot 10^{-10}$	$8.4 \cdot 10^{-10}$	$2.9 \cdot 10^{-8}$	$1.6 \cdot 10^{-9}$	$5.9 \cdot 10^{-8}$	$5.5 \cdot 10^{-8}$
Heun	$7.5 \cdot 10^{-9}$	$7.0 \cdot 10^{-9}$	1.14	$7.5 \cdot 10^{-2}$	$8.2 \cdot 10^{-4}$	$5.3 \cdot 10^{-4}$
RK4	$7.2 \cdot 10^{-9}$	$6.7 \cdot 10^{-9}$	1.11	$7.3 \cdot 10^{-2}$	$8.1 \cdot 10^{-4}$	$4.9 \cdot 10^{-4}$
P-C ABM 4	$7.3 \cdot 10^{-9}$	$6.7 \cdot 10^{-9}$	1.12	$7.3 \cdot 10^{-2}$	$8.3 \cdot 10^{-4}$	$4.9 \cdot 10^{-4}$

DR		AT		HT	
α	z	α	z	α	z
-	-	-	-	-	-
4.06	1.46	$6.5 \cdot 10^{-3}$	$1.4 \cdot 10^{-2}$	$2.2 \cdot 10^{-3}$	$9.5 \cdot 10^{-5}$
2.02	0.11	$1.2 \cdot 10^{-3}$	$3.2 \cdot 10^{-4}$	$6.2 \cdot 10^{-4}$	$3.9 \cdot 10^{-6}$
$7.3 \cdot 10^{-9}$	$2.0 \cdot 10^{-10}$	$3.8 \cdot 10^{-8}$	$2.4 \cdot 10^{-11}$	$9.4 \cdot 10^{-8}$	$8.9 \cdot 10^{-11}$
$2.5 \cdot 10^{-9}$	$2.0 \cdot 10^{-10}$	$4.1 \cdot 10^{-9}$	$2.0 \cdot 10^{-11}$	$7.5 \cdot 10^{-8}$	$7.9 \cdot 10^{-11}$
$3.3 \cdot 10^{-8}$	$1.4 \cdot 10^{-8}$	$4.1 \cdot 10^{-8}$	$2.2 \cdot 10^{-9}$	$9.4 \cdot 10^{-8}$	$1.0 \cdot 10^{-10}$
2.02	0.11	$1.2 \cdot 10^{-3}$	$3.2 \cdot 10^{-4}$	$6.2 \cdot 10^{-4}$	$3.9 \cdot 10^{-6}$
1.86	0.10	$1.1 \cdot 10^{-3}$	$3.1 \cdot 10^{-4}$	$6.0 \cdot 10^{-4}$	$3.8 \cdot 10^{-6}$
1.87	0.10	$1.1 \cdot 10^{-3}$	$3.1 \cdot 10^{-4}$	$6.1 \cdot 10^{-4}$	$3.8 \cdot 10^{-6}$

Tabla 3.1.5: Resultados del error relativo [%] en Ejes Viento.

Observando los datos recogidos en estas tablas, es sencillo comprobar como el método con el que se obtienen unos resultados más parecidos a los calculados con el *ode45* es el *ode89*. Esto puede ser debido a que este solver, según la descripción dada por MATLAB, se ha programado para ser más robusto y más eficiente en integraciones de intervalos grandes (para este caso el intervalo simulado era de 300 segundos).

También es interesante mencionar que, para la mayoría de los casos, los resultados conseguidos mediante los métodos de programación más sencilla brindan resultados que difieren más de la referencia y, por tanto, tienen un error mayor. Para cuantificar esta diferencia se pueden comparar los órdenes de magnitud obtenidos con estos métodos y los métodos propios de MATLAB, siendo la diferencia en algunos errores de hasta 10 órdenes por encima en el caso de los solvers sencillos. Esto es lógico, ya que en los solvers *ode* se han implementado diferentes técnicas y métodos para mejorar la eficiencia de los mismos.

Para tener una primera idea de aquellos métodos que se pueden descartar, es posible definir un *threshold* a partir del cual no se acepte el solver. Para este estudio se ha optado por fijar este límite en un error [%] de 10^{-3} en las variables de todas las maniobras calculadas. Cualquier método que proporcione un error relativo superior al *threshold* se rechazará como candidato para sustituir al *ode45*.

3.1.3. Resultados con menor paso

Al inicio del capítulo se ha comentado que es posible modificar el paso de integración de los métodos de programación "casera" para ver si esto puede dar lugar a un menor error en los resultados. Para ello se va a probar un caso en el que se va a reducir el paso de integración a la mitad en los métodos de *Euler, Euler modificado* y *Heun* para la maniobra de giro en ascenso en dos sistemas de referencia distintos, por lo que el nuevo paso de integración será de $h = 0.05$.

En la Tabla 3.1.6 se recoge el tiempo de cálculo con el paso de integración original, el tiempo de cálculo con el nuevo valor de paso y el incremento en porcentaje de ambos casos.

Método	Ejes Cuerpo (Euler)			Ejes Viento		
	$h = 0.1$	$h = 0.05$	Δt [%]	$h = 0.1$	$h = 0.05$	Δt [%]
Euler	9.62	21.35	121.98	7.45	16.28	118.53
Euler mod.	18.99	41.53	118.65	14.71	33.60	128.33
Heun	19.07	41.53	117.78	14.90	33.95	127.82

Tabla 3.1.6: Comparación del tiempo de cálculo [s] para dos pasos de integración.

Hasta ahora, y con lo recogido en la tabla anterior, se puede determinar que al reducir a la mitad el paso de integración el tiempo de cálculo de estos métodos se ve duplicado. En realidad el valor obtenido es algo más del doble, pues se ve incrementado en un 120 % aproximadamente. Ahora se deberá comprobar si este incremento del tiempo de computación también se traduce en una mayor precisión de los resultados, es decir, que el error con respecto a la referencia sea menor.

A continuación, se muestran los errores correspondientes a la maniobra estudiada en este apartado para las dos variables características que se establecieron en la Sección 3.1.2

Método	Ejes Cuerpo (Euler)				Ejes Viento			
	$h = 0.1$		$h = 0.05$		$h = 0.1$		$h = 0.05$	
	θ	z	θ	z	α	z	α	z
Euler	0.3276	0.0148	0.1626	0.00743	0.00652	0.0143	0.00288	0.00723
Euler mod.	0.00403	0.00030	0.00395	0.00029	0.00121	0.00032	0.00119	0.00032
Heun	0.00402	0.00030	0.00395	0.00029	0.00121	0.00032	0.00119	0.00032

Tabla 3.1.7: Comparación del error [%] para dos pasos de integración.

En la Tabla 3.1.7 se aprecia como, para el método de *Euler*, el error sí que se consigue reducir a la mitad, pero en los otros métodos la diferencia es muy pequeña.

3.2. Conclusión del estudio.

Una vez obtenidos los tiempos de cálculo de todos los métodos estudiados y los errores relativos para diferentes casos, se puede valorar si existe un método que sea mejor que el que se usaba previamente.

Por un lado, se tiene que el *ode113* era el método que menor tiempo de cálculo daba para la mayoría de los casos, aunque había otros donde era superado. En cuanto al error cometido, la situación está bastante más clara, el *ode89* supera a los demás en casi todos los casos. El problema está en que el tiempo de cálculo de este último método era bastante elevado, incluso en las maniobras más sencillas.

Por otro lado, el error relativo obtenido con el *ode113*, en ocasiones, es varios órdenes superior al del *ode89*, pero recuperando el *threshold* definido anteriormente, el solver *ode113* sobrepasa este límite con bastante margen. De esta manera se puede concluir que el error obtenido por este último método puede ser considerado lo suficientemente pequeño como para despreciarse.

Por tanto, juntando la rapidez del método y la alta precisión de los resultados al compararlos con la referencia, se puede deducir que el **solver ode113** puede ser un buen sustituto al *ode45* para el simulador.

Incluso si se considerase utilizar uno de los métodos extraídos de la bibliografía, pero reduciendo el paso de integración, se ha visto que solo en algunos casos se consigue mejorar la precisión de los resultados (el error sigue siento alto) y siempre a costa de un incremento en el tiempo de cálculo. Es por eso por lo que estos métodos se descartan como candidatos.

Capítulo 4

Exposición de resultados

En este punto del trabajo ya ha sido presentado el programa, los elementos que lo componen, las funciones que desempeñan todos estos elementos y se ha realizado un estudio con el que se ha obtenido un método numérico más rápido para la resolución de las ecuaciones. El siguiente paso es comprobar que la implementación del código del simulador se ha realizado correctamente y no se produce ningún fallo al interactuar con la GUI.

Para ello, se realizará el análisis de una maniobra concreta, con unas condiciones de vuelo determinadas, y se compararán los valores obtenidos con los alcanzados por L. Villanueva en su trabajo [2]. También se cotejará el valor de la masa de combustible consumida durante dicha maniobra.

Una vez se haya verificado el programa, se pasará a exponer el resultado de la principal aportación del presente trabajo: la interfaz dinámica que vaya mostrando la solución en cada instante. Se comentarán las diferentes estrategias de representación seguidas y los problemas que han ido surgiendo.

Finalmente, se llevará a cabo una comparación adicional para ilustrar las diferencias que existen entre utilizar el sistema de Ejes Cuerpo con los Ángulos de Euler o con los quaterniones de Euler-Rodrigues.

4.1. Validación del programa

Para la verificación del funcionamiento de la aplicación se ha determinado que la maniobra a analizar sea un ascenso helicoidal, pues representa un movimiento tanto en el plano vertical como en el horizontal. La aeronave que va a realizar dicha maniobra será el F/A-18, cuyos parámetros aparecen recogidos en la Sección 5.1. El estudio se efectuará en Ejes Cuerpo, iniciándose con las condiciones que aparecen en la Tabla 4.1.1. La velocidad angular de giro será $\Omega = 0.0262$ rad/s y los controles se mantendrán fijos durante el ascenso con los valores de la Tabla 4.1.2, siendo estos valores los calculados para llevar a cabo la maniobra de manera estacionaria.

u_0	173.90	$[m/s]$	p_0	-0.0045	$[rad/s]$	ϕ_0	0.2618	$[rad]$	x_0	0	$[m]$
v_0	-14.47	$[m/s]$	q_0	0.0067	$[rad/s]$	θ_0	0.1745	$[rad]$	y_0	0	$[m]$
w_0	13.24	$[m/s]$	r_0	0.0249	$[rad/s]$	ψ_0	0	$[rad]$	z_0	3000	$[m]$

Tabla 4.1.1: Condiciones iniciales para el ascenso helicoidal.

α	0.0760	$[rad]$	δ_E	-0.0675	$[rad]$
β	-0.0828	$[rad]$	δ_A	-0.0961	$[rad]$
δ_P	0.4015	$[rad]$	δ_R	-0.0019	$[rad]$

Tabla 4.1.2: Valores de los controles durante la maniobra.

Con la configuración preliminar ya finalizada, es el momento de pasar a simular la respuesta de la aeronave y visualizar la evolución temporal de sus parámetros. Para realizar la comparación se presentarán los mismos parámetros que estudió la autora del trabajo anterior. Adicionalmente, también se mostrará la trayectoria conseguida en ambos casos. Para intentar ser lo más fiel posible a los resultados de L. Villanueva, se simulará también la maniobra durante un intervalo de 300 segundos.

Pasando a la comparación gráfica, destacar que la solución que se encuentra representada mediante rayas es aquella obtenida por la GUI, mientras que la mostrada en línea continua es la del programa original.

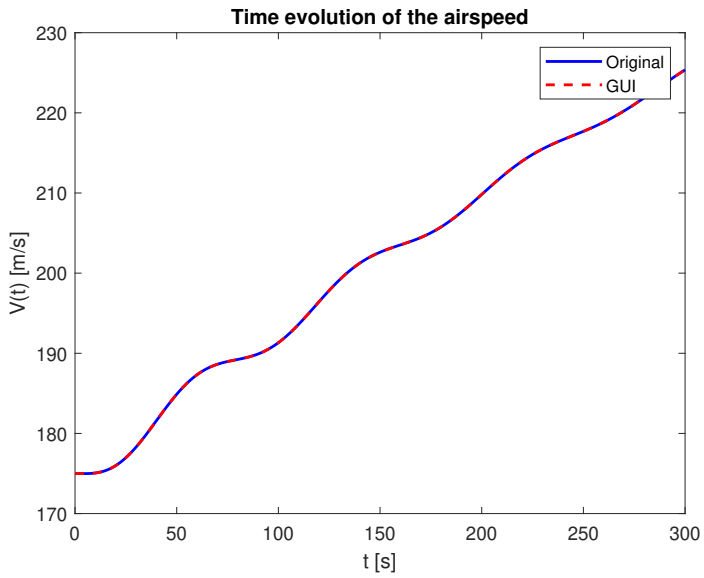


Figura 4.1.1: Representación de la velocidad de vuelo en un ascenso helicoidal.

Se comprueba, para este caso, que el resultado alcanzado con el simulador original y el logrado a partir de la nueva aplicación coinciden en todo el intervalo de representación. A continuación, se pasa a representar el resto de parámetros incluidos en este estudio para verificar que el comportamiento es correcto independientemente de las variables representadas.

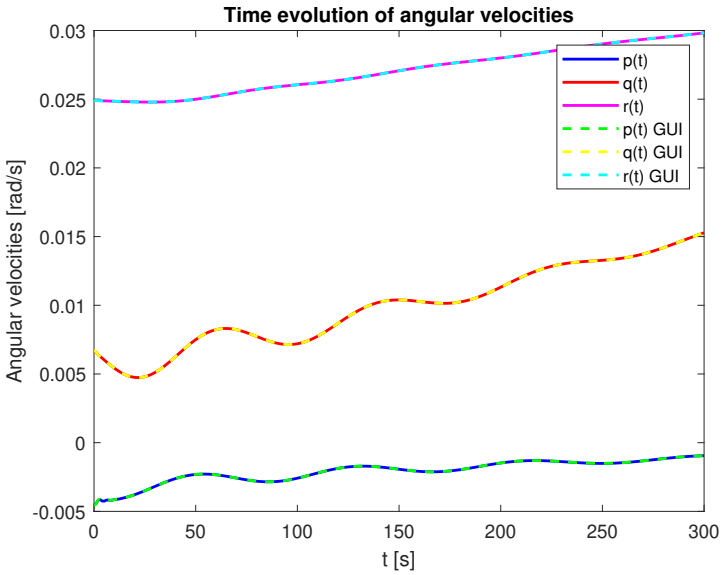


Figura 4.1.2: Representación de las velocidades angulares en un ascenso helicoidal.

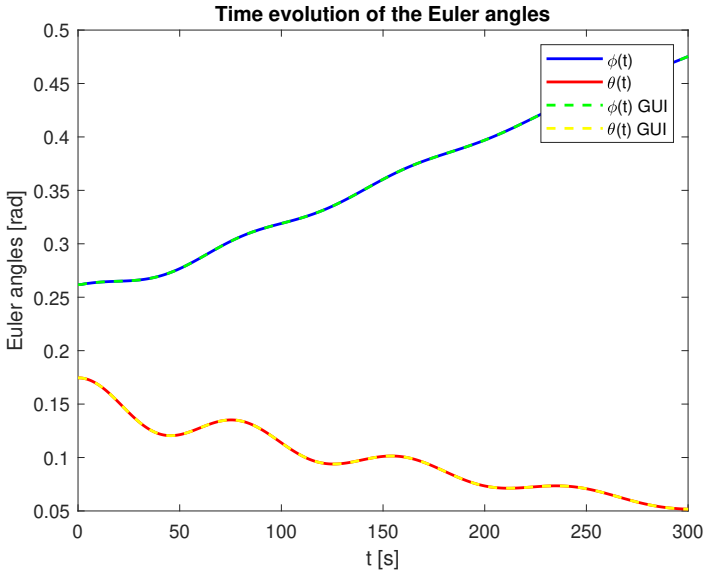


Figura 4.1.3: Representación del ángulo de alabeo y del de cabeceo.

Para obtener la velocidad de giro durante la maniobra será suficiente con derivar el resultado de ψ proporcionado por la aplicación.

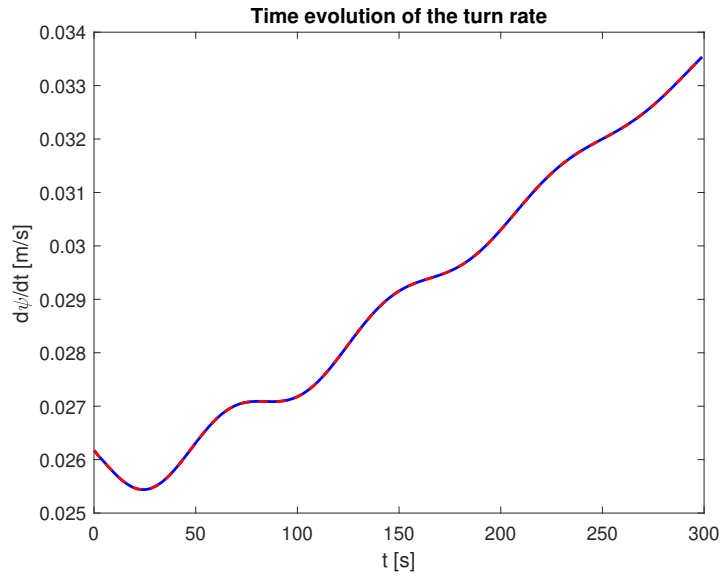


Figura 4.1.4: Representación del ratio de giro en un ascenso helicoidal.

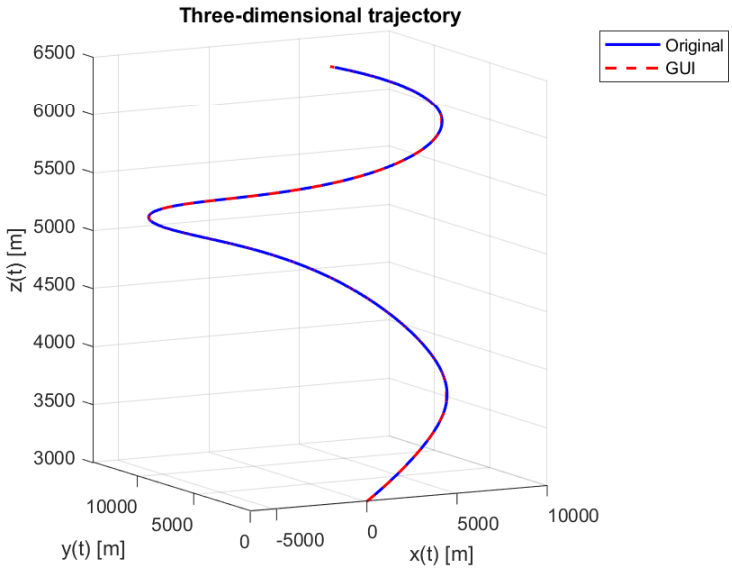


Figura 4.1.5: Representación de la trayectoria.

Durante este periodo de tiempo la aplicación devuelve que el valor de la masa de combustible consumida es de 163.81 kg, la misma cantidad que la calculada en el trabajo de referencia.

Como se puede comprobar en las comparaciones, el resultado logrado con los dos programas es el mismo. De esta manera, es posible dar por concluida la verificación de la implementación del código en la GUI con una evaluación positiva.

4.2. Interfaz dinámica

En este capítulo se pretende explicar la nueva herramienta con la que cuenta la versión actual del simulador, la cual ha sido el objetivo perseguido en el presente trabajo. Como se ha ido comentando a lo largo del informe, la principal aportación se trata de la posibilidad de visualizar la evolución temporal de las variables en tiempo real y poder actuar sobre los controles de tal manera que pueda verse su efecto de forma inmediata.

En las siguientes secciones se desarrollarán estas ideas, así como las diferentes estrategias que se han llevado a cabo para conseguir el comportamiento deseado y los problemas encontrados.

4.2.1. Estrategia de representación

En un primer momento, se probó a programar el código para que resolviese las ecuaciones en un intervalo y fuese mostrando la solución de cada punto contenido en ese intervalo. Para ilustrar mejor el procedimiento que sigue esta programación, se ha creado un diagrama de flujo (Figura 4.2.1) en el que se representan los pasos que sigue el programa.

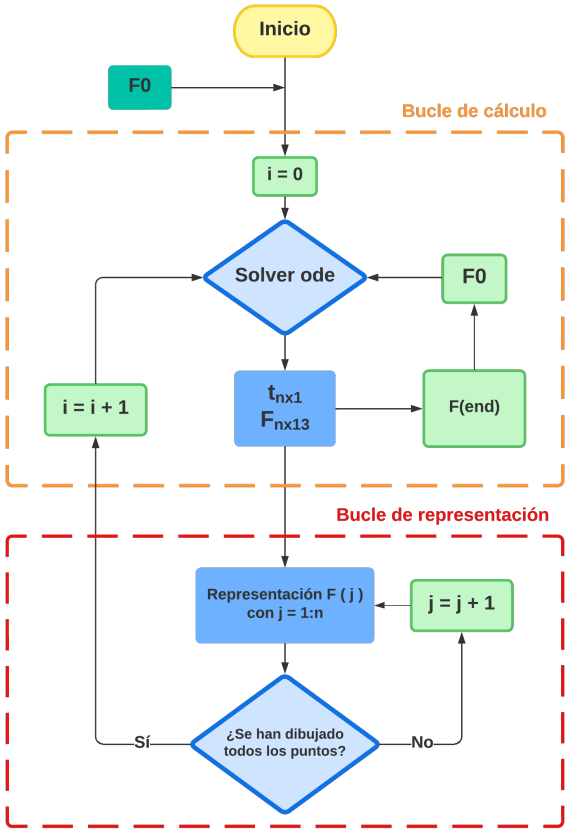


Figura 4.2.1: Diagrama de flujo de la representación punto a punto.

En la imagen anterior se puede ver como el proceso empieza con las condiciones iniciales definidas por el usuario y, a continuación, pasa a resolver las ecuaciones en un intervalo definido por el índice *i*. En cada resolución se calcularán los valores de las variables en el rango *(i i+1)*, para posteriormente devolver los correspondientes vectores de tiempo y estado, cada uno de ellos con n puntos dentro del intervalo. Una vez se tienen dichos vectores se inicia otro bucle en el que se mostrarán gráficamente cada uno de los n valores contenidos en la variable de F representada. Cuando se han terminado de representar todos los valores, se vuelve a iniciar el bucle de cálculo en un nuevo intervalo.

Esta implementación se terminó descartando, pues requería una alta demanda de recursos (llegando en ocasiones a emplear el cincuenta por ciento de la capacidad del procesador) y los resultados logrados distaban bastante de ser adecuados: la representación era extremadamente lenta, pudiendo ser veinte veces más lento que la realidad, y el intervalo temporal de resolución era variable. Por ello, se optó por cambiar el método de resolución, que pasaría a ser calcular los valores de la respuesta en un intervalo de tiempo fijo. La expresión para lograr este comportamiento es la siguiente:

$$[t, F] = ode113\,(odefun, [i \quad i + intervalo], F_0, options)$$

El funcionamiento del comando es sencillo, mientras la simulación se vaya ejecutando el solver calcula los resultados para el rango de tiempo indicado entre corchetes y, posteriormente, los muestra en la figura correspondiente, repitiendo este proceso en un bucle hasta que se detenga la simulación. El parámetro que permite avanzar en la resolución es el índice i, iniciándose con un valor nulo y que en cada vuelta del bucle se le suma el valor de *intervalo*. Este último parámetro es el que determina el rango temporal para el que se van a resolver las ecuaciones y que marcará el funcionamiento de la aplicación. La principal diferencia con respecto al método anterior radica en que antes se obtenían cientos de puntos para cada segundo calculado (y se graficaban todos ellos), y ahora este proceso se realiza cada un tiempo fijo, representándose todo el intervalo calculado a la vez. Tras probarse diversos valores, se ha llegado a la conclusión que para $intervalo = 0.2$, es decir, que calcule la solución cada 0.2 segundos, se llega a un buen compromiso entre fluidez y funcionalidad. A diferencia del caso anterior, ahora el tiempo real transcurrido puede ser 3 o 4 veces mayor que el simulado. Finalmente, en cada vuelta del bucle se toman los últimos valores del vector solución y se designan como el vector de estado inicial F_0 de la siguiente vuelta.

Conociendo como se obtiene la solución para un intervalo de tiempo determinado, ahora es necesario saber cómo se ha operado para intentar representar las variables en tiempo real. Seguidamente se mostrarán las diferentes opciones que se han probado para llevar a cabo dicha tarea.

1. Comando *pause(intervalo)*

   Con *pause(n)* se consigue pausar la ejecución del programa durante n segundos antes de continuar. La idea de utilizar este comando era que parase la ejecución de la representación durante el valor de *intervalo* y que al terminar dicha pausa retomase el cálculo y la representación. De esta manera, se conseguirían representar 0.2 segundos de la simulación cada 0.2 segundos reales. El inconveniente viene cuando se ha estado simulando durante un tiempo y se desea realizar alguna modificación sobre la aplicación, ya sea cambiar los controles o detener la representación. Por lo que se ha visto, al utilizar *pause*, si la demanda de recursos del programa es elevada, las interacciones posteriores pueden tardar hasta minutos en ejecutarse tras darse la orden. Este comportamiento será mucho más acusado cuanta más información se represente en la gráfica. Además, se produce un gran retraso entre el tiempo simulado y el que ha pasado en realidad.

2. Comando *drawnow limitrate*

   Este comando, según la propia documentación de MATLAB, actualiza las figuras y ejecuta cualquier *callbak* pendiente. Su uso está recomendado si se modifican objetos gráficos y se desean ver las actualizaciones de manera inmediata. Además, al acompañar el comando con *limitrate*, el programa limita el número de actualizaciones por segundo, buscando así crear animaciones más rápidas. Su uso es aconsejado cuando se trabaje con representaciones gráficas en un bucle.
   Con este comando se soluciona el problema del retraso en la ejecución de los *callbacks*, pero sigue dejando pendiente la diferencia entre el tiempo simulado y el real (en representaciones de una única variable, incluso la simulación va por delante del tiempo real).

3. Representación de los últimos segundos de la simulación

   Dado que se ha encontrado un problema en el retraso de la simulación, y está influenciado por la cantidad de información que se grafica en cada momento, se ha considerado actualizar la representación de las variables de tal forma que solo se muestren los últimos segundos de la solución. Para ilustrar el funcionamiento se tomará un ejemplo en el que se representa el vuelo equilibrado del F/A-18 a $z = 3000m$ y $V = 175m/s$ (Figura 4.2.2).

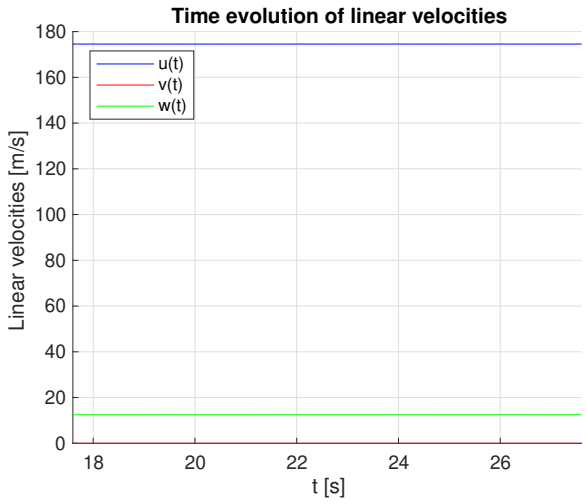


Figura 4.2.2: Representación en la que se muestran los últimos segundos de la solución.

Con esta técnica se busca reducir la información representada y conseguir así que el retraso en la visualización de las variables disminuya.

Como se ha mencionado, el principal problema hallado en el desarrollo de la GUI es el retraso en la representación de las variables simuladas con respecto al tiempo real, y que este retraso también tiene que ver con la cantidad de información que se grafique. Una alternativa para aliviar las diferencias temporales podría ser representar los últimos segundos de la simulación, aunque este método puede dificultar la interpretación de los resultados al no mostrar toda la respuesta en el mismo gráfico. La solución a dicho problema se solventa con lo explicado en la Sección 2.3, pues al pulsar el botón de *"Stop"* se exporta el vector de los resultados al espacio de trabajo de MATLAB para su posterior análisis.

4.2.2. Secuencia temporal en condiciones de equilibrio

Para ilustrar el funcionamiento de la interfaz gráfica dinámica se van a elaborar diversas secuencias temporales con las que se mostrará la visualización de la gráfica en varios instantes de tiempo. Para la creación de las secuencias se simulará la maniobra de vuelo de planeo para el F/A-18, y se han seleccionado la altitud, la pendiente de vuelo y el ángulo de asiento lateral como las variables a representar. Las condiciones iniciales de vuelo y los controles que se deben implementar en el simulador para realizar la maniobra de forma estacionaria son los recogidos en las Tablas 4.2.1 y 4.2.2.

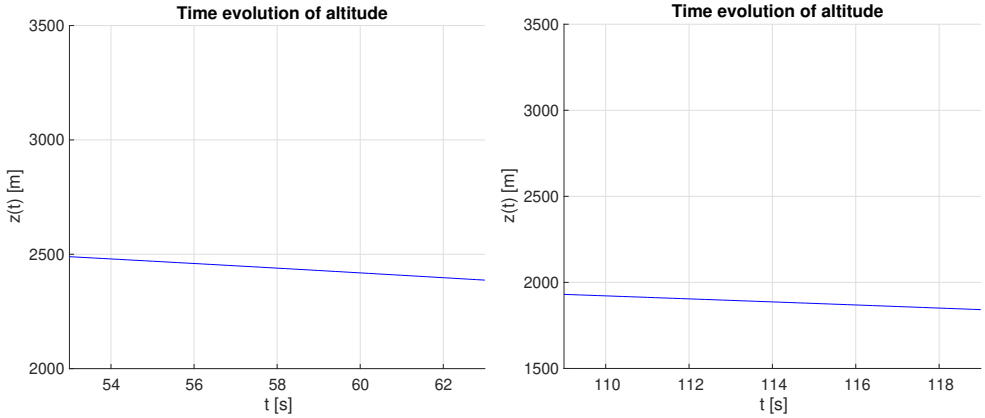
V_0	175	$[m/s]$	p_0	0	$[rad/s]$	μ_0	0	$[rad]$	x_0	0	$[m]$
β_0	0	$[m/s]$	q_0	0	$[rad/s]$	γ_0	-0.0665	$[rad]$	y_0	0	$[m]$
α_0	0.0715	$[m/s]$	r_0	0	$[rad/s]$	χ_0	0	$[rad]$	z_0	3000	$[m]$

Tabla 4.2.1: Condiciones iniciales para el vuelo de planeo.

δ_E	-0.0635	$[rad]$	δ_A	0	$[rad]$
δ_P	0	$[rad]$	δ_R	0	$[rad]$

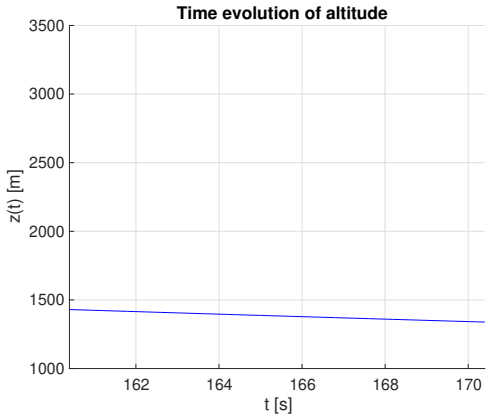
Tabla 4.2.2: Valores de los controles durante la maniobra.

La primera secuencia temporal que se va a evaluar es la de la altitud, representada en la Figura 4.2.3. Se ha capturado la imagen mostrada por la GUI en seis momentos diferentes, siendo el último de ellos el instante en el que la aeronave toca suelo. Los tiempos representados en las seis imágenes se corresponden con el tiempo que ha pasado realmente desde que se inició la simulación. Más adelante se comentará la diferencia que existe entre el tiempo real y el que aparece simulado en las gráficas.

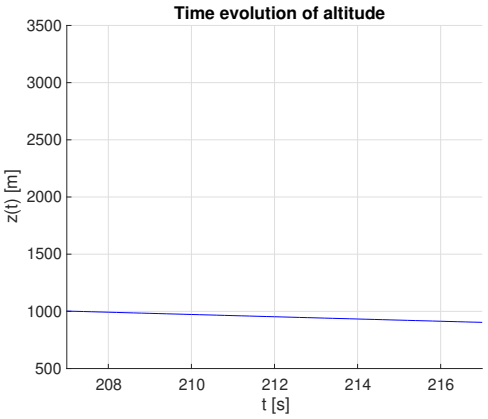


(a) $t = 15seg$

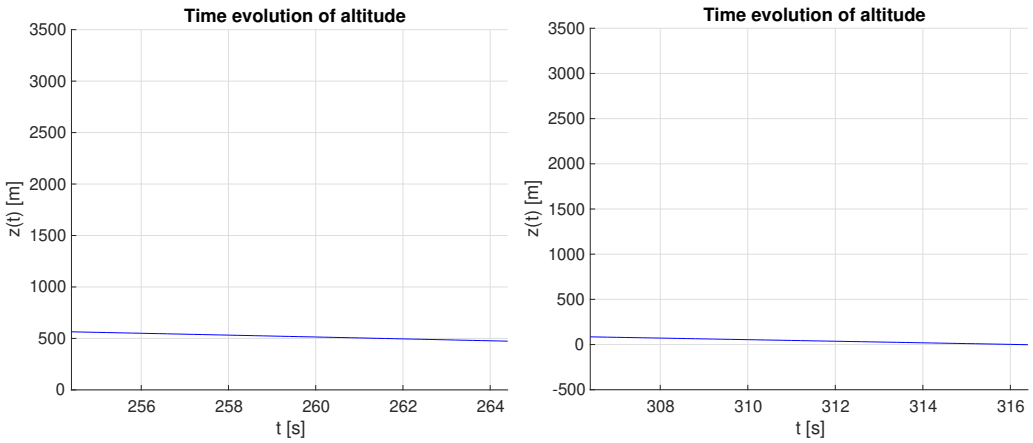
(b) $t = 30seg$



(c) $t = 45seg$



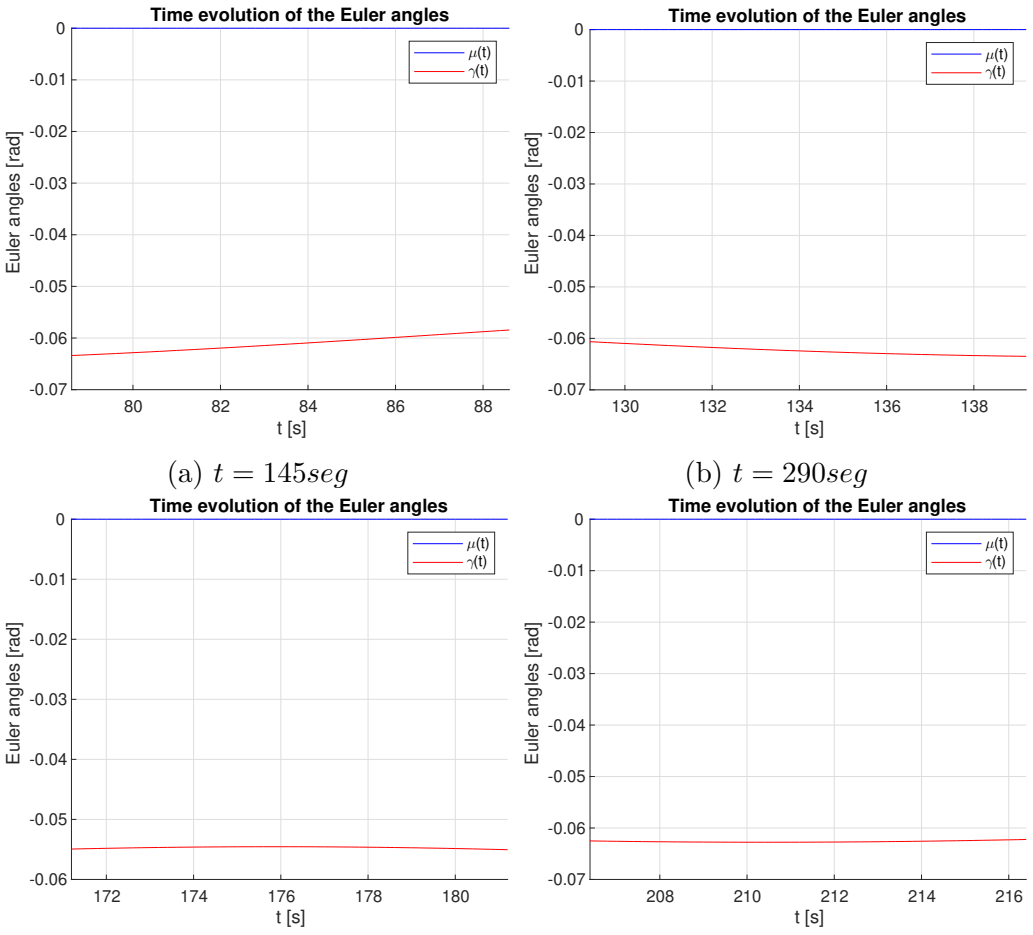
(d) $t = 60seg$



(e) $t = 75seg$ (f) $t = 105seg$

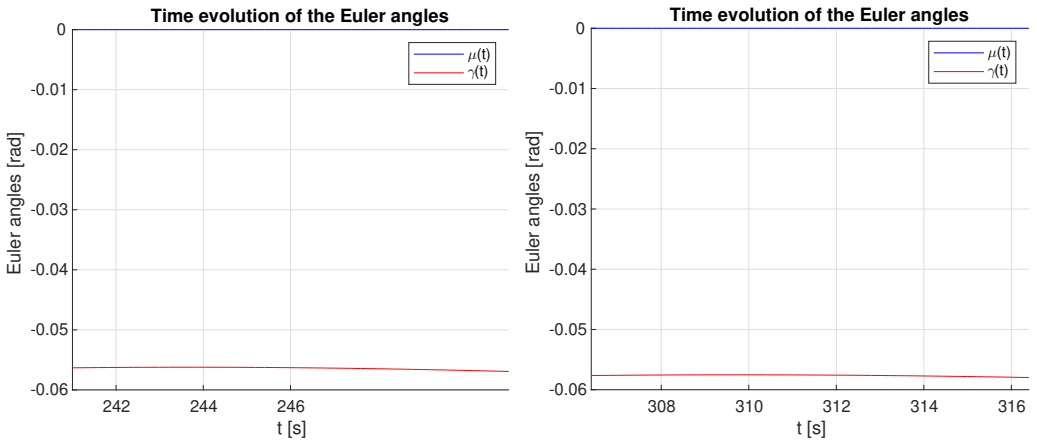
Figura 4.2.3: Secuencia temporal de la altitud en vuelo de planeo.

Se sigue un procedimiento similar para la secuencia temporal del asiento lateral y de la pendiente de vuelo (Figura 4.2.4).



(a) $t = 145seg$ (b) $t = 290seg$

(c) $t = 435seg$ (d) $t = 580seg$



(e) $t = 725seg$ (f) $t = 870seg$

Figura 4.2.4: Secuencia temporal del asiento lateral y la pendiente en vuelo de planeo.

En las figuras anteriores ha sido posible visualizar la evolución de las variables en distintos momentos de la simulación. En la Figura 4.2.3 se ve como la aeronave va descendiendo hasta llegar a tocar tierra, pues al no tener empuje y mantener los controles fijos, la resistencia frena la aeronave haciéndola perder altitud. Mientras tanto, la pendiente de vuelo oscila en torno a valores negativos, indicando que se trata de un descenso, y el asiento lateral permanece constante y nulo, ya que no hay ninguna acción o perturbación que saque al avión del plano vertical. Es de extrañar que el valor de la pendiente oscile cuando se está realizando una maniobra estacionaria, pero esto sucede ya que los decimales incorporados en el simulador no son exactos.

Algo que también se percibe en las secuencias temporales, y que es muy llamativo, es la diferencia en el tiempo transcurrido entre una simulación y otra, siendo de aproximadamente 8 veces más en el segundo caso. Como se había comentado en la Sección 4.2.1, esta disparidad se debe a la cantidad de información que se añade a la gráfica en cada actualización, ya que se representan en el segundo caso dos variables en vez de una.

Al analizar una variable la simulación se adelantaba y si se representan más la simulación se retrasa. Este retraso o adelanto en la simulación no implica una pérdida de precisión en los resultados, pues se obtiene la misma solución que con el programa de L. Villanueva [2].

4.2.3. Secuencia temporal con acción sobre los controles

Otro aspecto positivo que se ha logrado con la creación de la interfaz es la posibilidad de realizar tantos cambios en los controles como se desee mientras se está ejecutando la simulación. De esta manera se consigue una gran libertad a la hora de estudiar maniobras pues no es necesaria la implementación de código adicional si se quieren introducir diversas acciones sobre los controles.

Para ilustrar esta funcionalidad se extraerá otra secuencia temporal en la que se modificarán los controles en tres instantes de tiempo distintos y se observará el comportamiento de algunas variables ante dichos cambios. La maniobra se iniciará con la aeronave equilibrada en el plano vertical según los valores de la Tabla 4.2.3, posteriormente, se impondrán un incremento y una disminución del ángulo de deflexión del timón de profundidad y, finalmente, se establecerá un ángulo en los alerones de 15 grados. Esta última actuación será de décimas de segundo. El cambio en la deflexión de los alerones implicará la aparición de un momento de alabeo y aceleración de alabeo no nulas en ese instante, dando lugar a que p aumente.

De esta forma se conseguirá ver la respuesta de la aeronave mientras se encuentra contenida en el plano vertical y, también, al imponer una deflexión grande de alerones que la sacará de dicho plano. Los instantes y valores de las deflexiones de los controles aparecen recogidos en la Tabla 4.2.4.

u_0	174.55	$[m/s]$	p_0	0	$[rad/s]$	ϕ_0	0.2618	$[rad]$	x_0	0	$[m]$
v_0	0	$[m/s]$	q_0	0	$[rad/s]$	θ_0	0.0713	$[rad]$	y_0	0	$[m]$
w_0	12.48	$[m/s]$	r_0	0	$[rad/s]$	ψ_0	0	$[rad]$	z_0	3000	$[m]$

Tabla 4.2.3: Condiciones iniciales para equilibrio en el plano vertical.

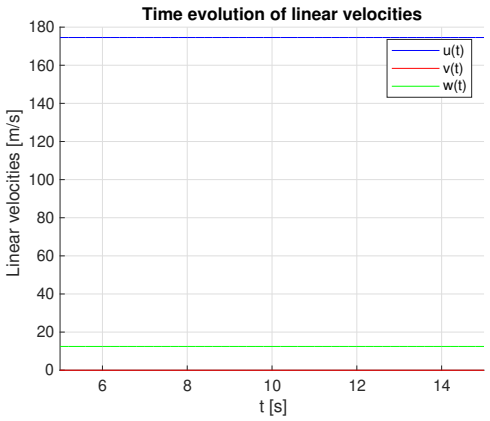
	δ_E	δ_A	δ_R	δ_P
$t = 0\ seg$	-0.0633	0	0	0.1192
$t = 20\ seg$	-0.09	0	0	0.1192
$t = 60\ seg$	-0.04	0	0	0.1192
$t = 100\ seg$	-0.04	-0.2618	0	0.1192

Tabla 4.2.4: Valores de los controles [rad] durante la simulación.

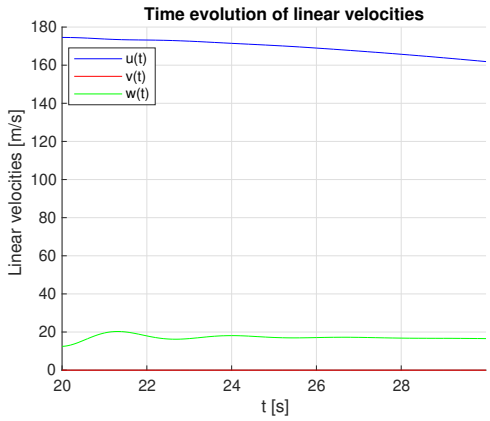
Los cambios en los controles se modelizan en el sistema como entradas escalón, de manera que la variación de los mismos en la simulación es inmediata.

Puesto que la deflexión del elevador es control directo sobre la velocidad de vuelo de la aeronave, la primera secuencia que se va a representar corresponde a la evolución temporal de las componentes lineales de la velocidad (Figura 4.2.5).

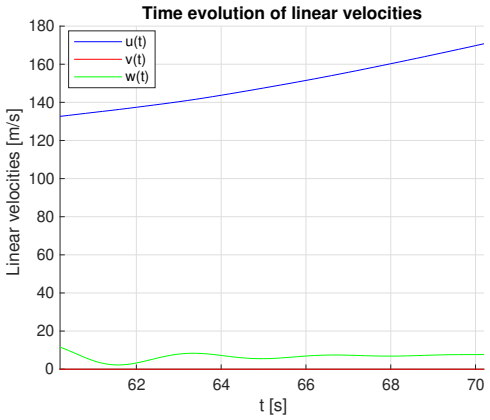
Más adelante se presentará la misma secuencia temporal para los ángulos aerodinámicos, cuya relación con las velocidades lineales se estableció en la Sección 2.2.1.



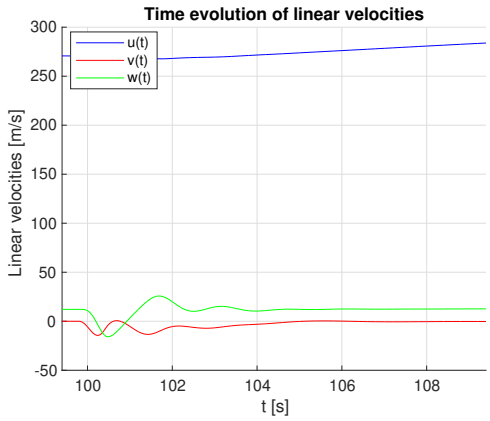
(a) Valores en el equilibrio



(b) Después de la primera modificación



(c) Después de la segunda modificación



(d) Instante de la colisión.

Figura 4.2.5: Secuencia temporal de u, v y w con actuación sobre los controles.

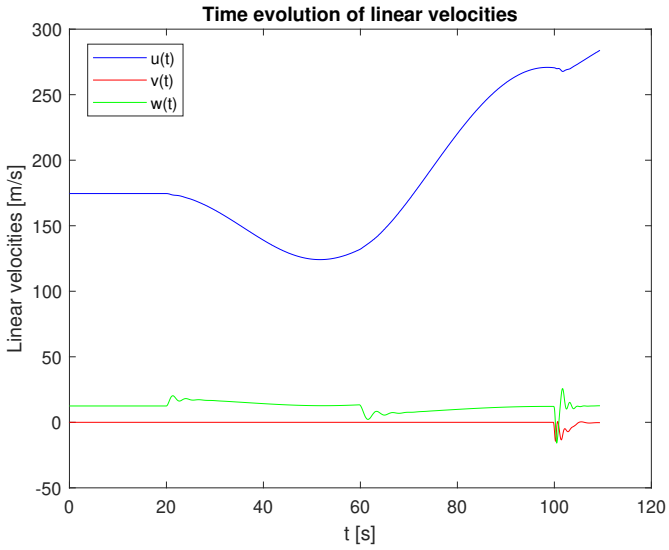


Figura 4.2.6: Reconstrucción de la evolución temporal de las velocidades lineales.

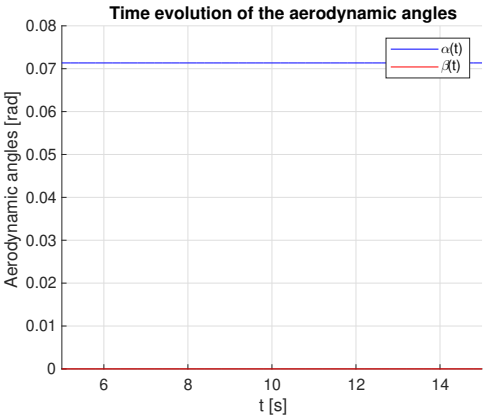
Al examinar la Figura 4.2.6 anterior se puede comprobar como, ante pequeñas variaciones de los controles (especialmente de δ_E), se consiguen grandes cambios en la velocidad. Para tener una idea sobre el comportamiento del módulo de la velocidad, en este caso es sencillo, basta con fijarse en la dinámica de la componente en dirección X, ya que predomina sobre las otras dos.

Pasando al análisis propiamente dicho, antes de actuar sobre ningún elemento se verifica que la aeronave se encuentra en equilibrio, ya que los valores de las velocidades se mantienen constantes en el tiempo. Tras realizar la primera modificación sobre δ_E se percibe una variación en las componentes longitudinal y vertical de la velocidad, notándose una reducción de su magnitud en el primer caso y un aumento en el segundo. Este es el comportamiento esperado según la teoría: al aumentarse la deflexión del timón de profundidad también se consigue un incremento en el ángulo de ataque, que se traduce en un aumento del coeficiente de sustentación. Al conseguir una mayor sustentación, deja de cumplirse el equilibrio de fuerzas verticales en las que esta fuerza era igual al peso, por lo que la aeronave tenderá a buscar otro punto de equilibrio mediante la reducción de la fuerza de sustentación a través de una reducción en la velocidad.

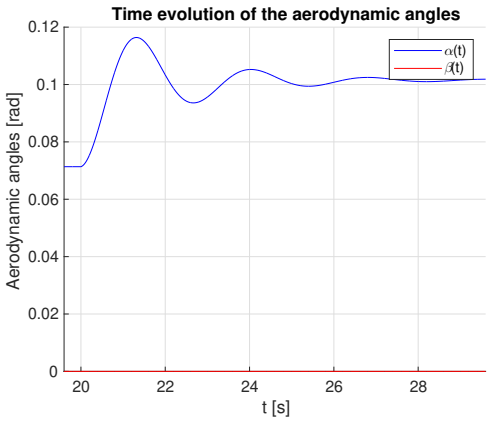
Además, en la figura se aprecian con claridad los dos modos característicos de la dinámica longitudinal: en la velocidad longitudinal se consigue un comportamiento oscilatorio de frecuencia y amortiguamiento reducidos, y en la componente vertical las oscilaciones son rápidas y se extinguen en pocos segundos. Estas dinámicas se corresponden con los modos Corto Período y Fugoide, respectivamente. Nuevamente, la teoría ya predecía dicho comportamiento tras estudiar de la matriz de sensibilidad del sistema longitudinal.

Si ahora se analiza la Figura 4.2.7c, es posible ver como la velocidad longitudinal vuelve a aumentar su valor tras imponer un decremento en la deflexión del empenaje horizontal, resultado esperado, pues es el contrario al que se ha comentado antes. Finalmente, cuando se impone un valor de deflexión de alerones, la velocidad lateral deja de ser nula y empieza a oscilar rápidamente, al igual que la componente w. Estas oscilaciones son producidas por una variación de una superficie de control de la dinámica lateral-direccional durante un instante muy pequeño, pero que tiene gran influencia. Transcurridos pocos segundos desde el movimiento de los alerones, y después de las oscilaciones, la aeronave llega a tocar tierra.

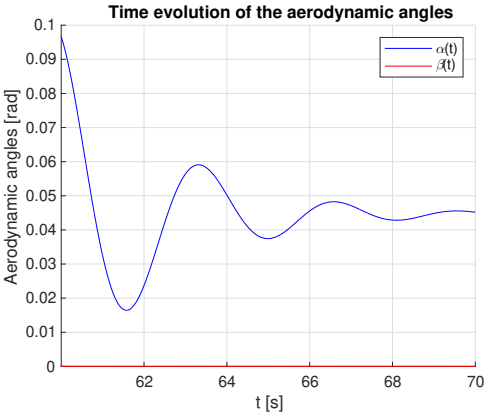
Puesto que previamente se han definido las relaciones que existen entre estas variables y los ángulos aerodinámicos, el siguiente paso es obtener la secuencia de la dinámica de los ángulos α y β ante la misma actuación de los controles. De esta manera será posible constatar las conclusiones sobre las fuerzas aerodinámicas a las que se ha llegado en el análisis anterior. Para ello, se han extraído 4 instantáneas en los mismos puntos que antes (Figure 4.2.5), con lo que se busca tener una mejor comprensión de las relaciones existentes entre los dos grupos de variables estudiados.



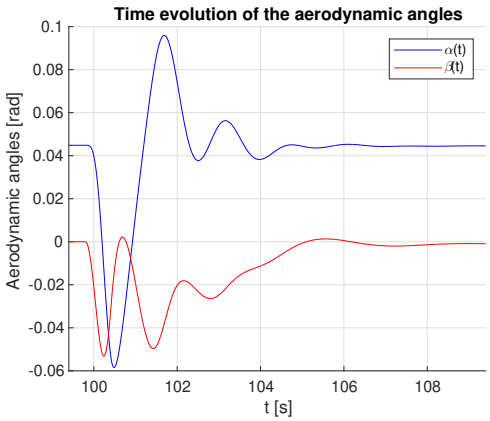
(a) Valores en el equilibrio



(b) Después de la primera modificación



(c) Después de la segunda modificación



(d) Instante de la colisión.

Figura 4.2.7: Secuencia temporal de α y β con actuación sobre los controles.

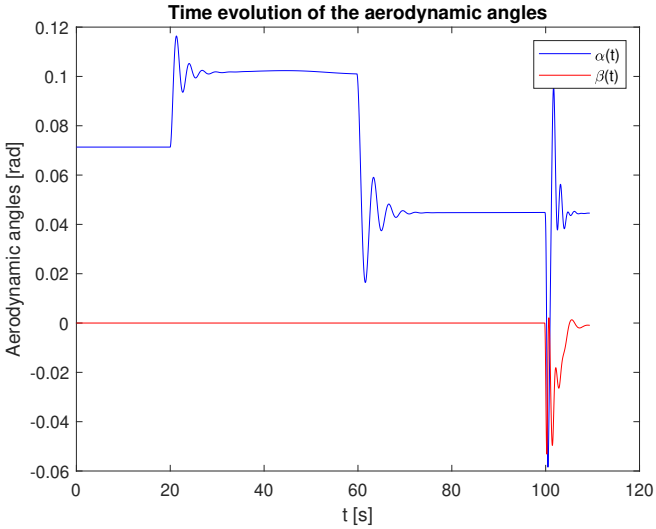


Figura 4.2.8: Reconstrucción de la evolución temporal de los ángulos aerodinámicos.

Como era de esperar, al iniciar la simulación las variables adquieren un valor constante y es al incluir la primera actuación del control del timón de profundidad cuando aparecen las oscilaciones en el ángulo de ataque. Según se había predicho en el análisis de las velocidades lineales, un aumento en el valor de δ_E se traduce en un incremento del ángulo de ataque. En esta secuencia es más fácil visualizar las oscilaciones características del modo Corto Período, que demuestra su relevancia al tratar con la variable estudiada. Al pasar algo menos de 10 segundos estas oscilaciones desaparecen, dando paso a otro movimiento oscilatorio de frecuencia mucho menor, aunque de mucha menos importancia. Esta es la misma respuesta que tiene la componente vertical de la velocidad, pues como se había visto antes, estas dos variables están relacionadas.

En la segunda modificación de δ_E se logra, de nuevo, un comportamiento similar al efectuado por w en la primera secuencia temporal. Como hasta ahora no se ha excitado ninguna variable de la dinámica latera-direccional de la aeronave, el valor del ángulo de derrape permanece en un valor nulo durante toda la simulación.

Cuando se modifica el valor del control de alabeo a los 100 segundos de simulación, vuelven a aparecer rápidas oscilaciones en los dos ángulos aerodinámicos. Dado que el ángulo de derrape deja de ser nulo, se tendrá una velocidad lateral (Figura 4.2.5d) que dará lugar a la aparición de fuerzas laterales y momentos de alabeo y guiñada. Esta acción sacará a la aeronave del plano vertical y propiciará la pérdida de control de la misma, lo que provocará que se estrelle contra el suelo si no se vuelve a actuar sobre los controles.

Después de haber extraído, analizado y comentado las dos secuencias temporales con diferentes actuaciones sobre los controles, queda demostrada la versatilidad y la ventaja que supone contar con una interfaz dinámica con la que se pueda interactuar activamente para visualizar en tiempo real el comportamiento dinámico de las aeronaves.

4.3. Diferencias entre Ángulos de Euler y quaterniones

Tal como se comentó al inicio del presente capítulo, se va a llevar a cabo una comparación con la que se pretende ilustrar las diferencias que existen al usar los Ángulos de Euler y los quaterniones cuando se trabaja en Ejes Cuerpo. El motivo de usar unos parámetros u otros viene de la aparición de una singularidad conocida como *gimbal lock* cuando se alcanzan valores del ángulo de cabeceo de $\theta = \pm\pi/2$. Matemáticamente, este fenómeno se puede visibilizar al obtener las relaciones inversas de la Ecuación 2.3, dando lugar a las expresiones de la Ecuación 2.4.

Este fenómeno ya había sido expuesto por L. Villanueva en su trabajo del simulador, pero no se había llevado a cabo ningún ejemplo en el que se mostrase el problema que supone la singularidad ni como el uso de los quaterniones de Euler-Rodrigues solventa este inconveniente.

Para el estudio se realizará el análisis de la maniobra de giro en el plano vertical, conocida comúnmente como loop vertical, ya que durante su ejecución se produce el *gimbal lock*. Se partirá desde las condiciones de equilibrio representadas por la Tabla 4.2.3 y los controles según la Tabla 4.3.1.

δ_E	-0.2968	$[rad]$	δ_A	0	$[rad]$
δ_P	1	$[rad]$	δ_R	0	$[rad]$

Tabla 4.3.1: Valores de los controles para el loop vertical.

Antes de mostrar los valores de los Ángulos de Euler obtenidos en ambas simulaciones, se mostrará la trayectoria que sigue la aeronave para verificar que, efectivamente, se está llevando a cabo un giro vertical. En la Figura 4.3.1 se aprecia a la perfección el loop realizado por la aeronave.

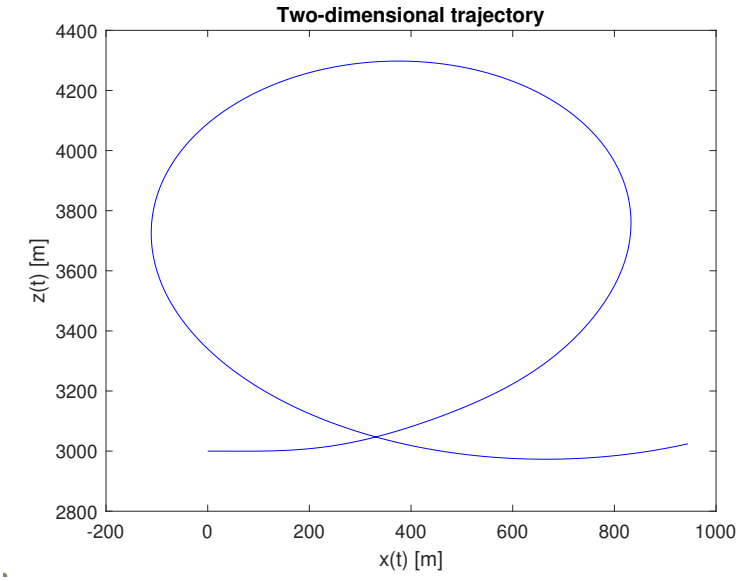


Figura 4.3.1: Trayectoria bidimensional de la aeronave durante el loop.

A continuación se muestran los dos resultados obtenidos al analizar la misma maniobra pero en los dos sistemas de referencia mencionados.

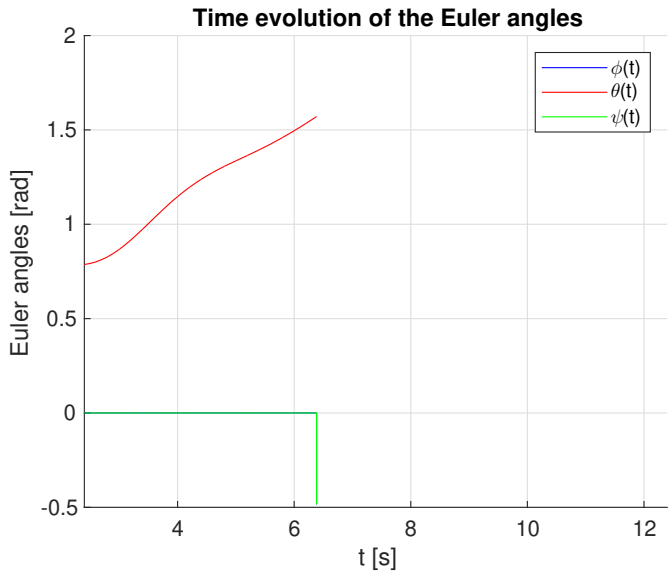


Figura 4.3.2: Evolución temporal de los Ángulos de Euler durante la maniobra de loop al estudiarse directamente con los ángulos.

Como se puede ver en este primer caso, cuando el ángulo de cabeceo llega al valor que da la singularidad, el simulador notifica que hay un error y deja de representar la solución.

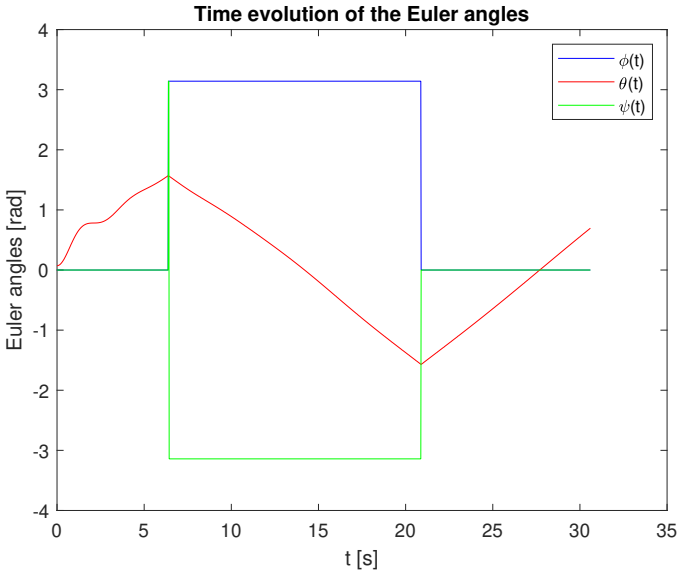


Figura 4.3.3: Evolución temporal de los ángulos de Euler durante la maniobra de loop al estudiarse mediante los quaterniones.

Tras la simulación de la operación con ambas opciones de cálculo, se puede ver como el resultado obtenido para los Ángulos de Euler es completamente distinto en función de los parámetros que se usen. En la Figura 4.3.2 se extrae que los ángulos de asiento lateral y de guiñada se mantienen constantes durante toda la maniobra y con un valor nulo, mientras que el ángulo de cabeceo sufre un incremento continuo hasta alcanzar el valor de $\theta = 2\pi$.

Sin embargo, al examinar la respuesta dinámica de las variables de la Figura 4.3.3 se muestra como el ángulo de asiento lateral permanece limitado entre los valores que dan lugar a la singularidad, evitando así que se produzca el fenómeno del *gimbal lock*. Además, los valores de alabeo y guiñada cambian en estos puntos críticos a $\phi = \pi$ y $\psi = -\pi$, poniendo de manifiesto que la aeronave queda invertida mientras está realizando la maniobra.

Una vez se ha analizado el comportamiento de la aeronave y de los parámetros que pueden conllevar a confusión, se pueden calcular los valores de las derivadas de los ángulos que, según la teoría, se esperan durante esta operación (Ecuación 4.1).

$$\dot{\phi} = 0 \qquad \dot{\theta} = q \qquad \dot{\psi} = 0 \tag{4.1}$$

Al no tener movimientos fuera del plano vertical, la única derivada que tiene un valor no nulo es la correspondiente a la dinámica longitudinal de la aeronave.

Capítulo 5

Biblioteca de aeronaves del simulador

En el presente capítulo se pretende realizar la recopilación y presentación de las características principales de algunas aeronaves de las que será posible realizar el estudio de su comportamiento dinámico.

De esta manera, el usuario dispondrá de varios tipos de aeronaves con los que podrá hacer comparaciones y ver la respuesta temporal de cada uno de ellos ante la variación de los controles aerodinámicos. Por supuesto, será posible añadir aeronaves concretas por parte del propio usuario si se tiene un interés especial en su estudio.

5.1. F/A-18 Hornet

Se trata de un avión de combate bimotor en configuración convencional que, como particularidad, incorpora dos estabilizadores verticales.

Figura 5.1.1: McDonnell Douglas F/A-18 [12]

Parámetros geométricos

Los parámetros geométricos y másicos más importantes aparecen resumidos en la Tabla 5.1.1.

b_w	11.43	$[m]$	m	13273	$[kg]$
c_w	3.02	$[m]$	I_{xx}	30673.6	$[kg \cdot m^2]$
S_w	37.16	$[m^2]$	I_{yy}	115752	$[kg \cdot m^2]$
L	17.07	$[m]$	I_{zz}	431240	$[kg \cdot m^2]$
x_{CoG}/c_w	0.702	$[-]$	I_{xz}	0	$[kg \cdot m^2]$

Tabla 5.1.1: Parámetros geométricos del F/A-18

Modelo de empuje

El F/A-18 cuenta con dos motores turbofan de baja tasa de derivación de la firma General Electric, obteniéndose los parámetros de la Tabla 5.1.2 para un solo motor.

T_0	48900	$[N]$	ε	0	$[rad]$	BPR	0.27
C_e	$2.3 \cdot 10^{-5}$	$[kg/N \cdot s]$	a	1	$[-]$	N_e	2

Tabla 5.1.2: Parámetros del modelo de empuje del F/A-18

Modelo aerodinámico

Una vez definidos los parámetros geométricos y el modelo de empuje de la aeronave, solo falta por determinar las derivadas aerodinámicas que componen el modelo lineal de las fuerzas y momentos aerodinámicos.

En la Tabla 5.1.3 se muestran los valores de los coeficientes de la dinámica longitudinal para la aeronave de esta sección.

C_{L0}	0	$C_{L\alpha}$	4.24237	$C_{M\alpha}$	-0.420158
C_{M0}	0	$C_{L\delta E}$	0.82536	$C_{M\delta E}$	-0.473495
C_{D0}	0.0100593	C_{Lq}	0.94699	C_{Mq}	-0.543272
K	0.10567	$C_{L\dot{\alpha}}$	0.48787	$C_{M\dot{\alpha}}$	-0.279883

Tabla 5.1.3: Derivadas aerodinámicas de la dinámica longitudinal del F/A-18

De la misma manera, las derivadas aerodinámicas de la dinámica lateral-direccional son las expuestas en la Tablas 5.1.4 y 5.1.5.

$C_{Y\beta}$	-0.575519	$C_{Y\dot{\beta}}$	0	C_{Yp}	-0.150519	C_{Yr}	0.27852
$C_{l\beta}$	-0.21195	$C_{l\dot{\beta}}$	0	C_{lp}	-0.258664	C_{lr}	0.122386
$C_{N\beta}$	0.0037412	$C_{N\dot{\beta}}$	0	C_{Np}	-0.0364167	C_{Nr}	-0.119257

Tabla 5.1.4: Derivadas aerodinámicas de la dinámica lat-dir del F/A-18

$C_{Y\delta A}$	0	$C_{l\delta A}$	0.183164	$C_{N\delta A}$	-0.001285
$C_{Y\delta R}$	0.352187	$C_{l\delta R}$	0.0443965	$C_{N\delta R}$	-0.147535

Tabla 5.1.5: Derivadas de control de la dinámica lat-dir del F/A-18

Se debe tener en cuenta que muchas de estas derivadas dependen de las condiciones de vuelo y de los valores de los ángulos aerodinámicos en el equilibrio, pero que por simplicidad se han calculado para una determinada condición y se mantienen constantes durante todo el estudio.

5.2. Mirage F-1

Con el fin de incluir otro avión de combate a la biblioteca, pero con una geometría algo diferente, entre todos los posibles se ha seleccionado el Mirage F-1 [13]. Este es un avión con el que se ha trabajado mucho durante el curso y que se caracteriza por ser una aeronave monomotor con un único estabilizador vertical, a diferencia del F/A-18.

Figura 5.2.1: Dassault Mirage F-1 [14]

Parámetros geométricos

En la Tabla 5.2.1 se exponen algunos parámetros geométricos e inerciales.

b_w	8.4	$[m]$	m	11100	$[kg]$
c_w	3.3	$[m]$	I_{xx}	13335	$[kg \cdot m^2]$
S_w	25	$[m^2]$	I_{yy}	92000	$[kg \cdot m^2]$
L	15.03	$[m]$	I_{zz}	130055	$[kg \cdot m^2]$
x_{CoG}/c_w	0.853	$[-]$	I_{xz}	8868.36	$[kg \cdot m^2]$

Tabla 5.2.1: Parámetros geométricos del F-1

Modelo de empuje

La planta motriz del Mirage F-1 es un turborreactor SNECMA Atar 9K-50 con capacidad de utilizar poscombustión. También se puede utilizar el modelo de D.Hull para su estudio, obteniéndose los parámetros de la Tabla 5.2.2.

T_0	49000	$[N]$	ε	0	$[rad]$	BPR	$-$
C_e	$2.78 \cdot 10^{-5}$	$[kg/N \cdot s]$	a	1.2	$[-]$	N_e	1

Tabla 5.2.2: Parámetros del modelo de empuje del F-1

Modelo aerodinámico

En la Tabla 5.2.3 aparecen recogidos los valores de las derivadas de la dinámica longitudinal para el F-1.

C_{L0}	0	$C_{L\alpha}$	3.7336	$C_{M\alpha}$	-0.84067
C_{M0}	0	$C_{L\delta E}$	0.8219	$C_{M\delta E}$	-1.07273
C_{D0}	0.014	C_{Lq}	4.5358	C_{Mq}	-4.0839
K	0.14	$C_{L\dot{\alpha}}$	3.6373	$C_{M\dot{\alpha}}$	-2.262

Tabla 5.2.3: Derivadas aerodinámicas de la dinámica longitudinal del F-1

Las derivadas de la dinámica lateral-direccional se resumen en las Tablas 5.2.4 y 5.2.5.

$C_{Y\beta}$	-1.0486	$C_{Y\dot{\beta}}$	0	C_{Yp}	0.16121	C_{Yr}	0.92714
$C_{l\beta}$	-0.1574	$C_{l\dot{\beta}}$	0	C_{lp}	-0.26367	C_{lr}	0.17414
$C_{N\beta}$	0.23801	$C_{N\dot{\beta}}$	0	C_{Np}	-0.05598	C_{Nr}	-0.53316

Tabla 5.2.4: Derivadas aerodinámicas de la dinámica lat-dir del F-1

$C_{Y\delta A}$	0	$C_{l\delta A}$	0.099	$C_{N\delta A}$	0
$C_{Y\delta R}$	0.11	$C_{l\delta R}$	0	$C_{N\delta R}$	-0.04

Tabla 5.2.5: Derivadas de control de la dinámica lat-dir del F-1

5.3. B737-500

Tras haber presentado un par de aeronaves que se encuentran en la categoría de aviones de combate, se va a pasar a exponer las características de varios aviones comerciales de pasajeros. El primero de ellos va a ser el Boeing 737-500 [15], con una capacidad de 120-130 pasajeros y dos motores turbofan situados bajos el ala.

Figura 5.3.1: Boeing 737-500 [16]

Además, la configuración de la cola es aquella que se conoce como convencional, con el estabilizador horizontal y vertical unidos al fuselaje en la parte trasera.

Parámetros geométricos

b_w	28.8	$[m]$	m	40868	$[kg]$
c_w	3.41	$[m]$	I_{xx}	618040	$[kg \cdot m^2]$
S_w	105.4	$[m^2]$	I_{yy}	1514020	$[kg \cdot m^2]$
L	33.4	$[m]$	I_{zz}	9905390	$[kg \cdot m^2]$
x_{CoG}/c_w	1.296	$[-]$	I_{xz}	0	$[kg \cdot m^2]$

Tabla 5.3.1: Parámetros geométricos del 737-500

Modelo de empuje

Como se ha comentado al principio de la sección, el B737-500 cuenta con dos motores turbofan de alta tasa de derivación y modelo CFM56-7B20. Para este tipo de aeronaves, y más concretamente por el tipo de sistema propulsivo, suele ser habitual emplear el modelo propulsivo de Howe, presentado en la Sección 2.2.2. Los datos para un solo motor se encuentran en la Tabla5.3.2.

T_0	104500	$[N]$	ε	0	$[rad]$	BPR	6
C_e	$1.89 \cdot 10^{-5}$	$[kg/N \cdot s]$	a	1	$[-]$	N_e	2

Tabla 5.3.2: Parámetros del modelo de empuje del 737-500

Modelo aerodinámico

En la Tabla 5.3.3 se muestran los valores de estos coeficientes de la dinámica longitudinal para el B737-500.

C_{L0}	0	$C_{L\alpha}$	5.47563	$C_{M\alpha}$	-3.05143
C_{M0}	0	$C_{L\delta E}$	0.65149	$C_{M\delta E}$	-2.26567
C_{D0}	0.0168	C_{Lq}	8.34872	C_{Mq}	-29.34872
K	0.04531	$C_{L\dot{\alpha}}$	2.37954	$C_{M\dot{\alpha}}$	-9.31849

Tabla 5.3.3: Derivadas aerodinámicas de la dinámica longitudinal del 737-500

En la Tablas 5.3.4 y 5.3.5 aparecen los valores correspondientes a las derivadas aerodinámicas de la dinámica lateral-direccional.

$C_{Y\beta}$	-1.003	$C_{Y\dot{\beta}}$	0	C_{Yp}	-0.229	C_{Yr}	0.673
$C_{l\beta}$	-0.112	$C_{l\dot{\beta}}$	0	C_{lp}	-0.493	C_{lr}	0.221
$C_{N\beta}$	0.215	$C_{N\dot{\beta}}$	0	C_{Np}	-0.0613	C_{Nr}	-0.323

Tabla 5.3.4: Derivadas aerodinámicas de la dinámica lat-dir del 737-500

$C_{Y\delta A}$	0	$C_{l\delta A}$	0.0485	$C_{N\delta A}$	0.000223
$C_{Y\delta R}$	0.276	$C_{l\delta R}$	0.0154	$C_{N\delta R}$	-0.163

Tabla 5.3.5: Derivadas de control de la dinámica lat-dir del 737-500

5.4. A320-200

Para seguir en la línea de aeronaves comerciales de pasajeros, se añadirán también los datos del Airbus A320-200 [17], teniendo así un ejemplo de los dos fabricantes de aviones más grandes a nivel internacional. Se ha seleccionado este modelo en concreto por las similitudes que existen con el 737-500.

Figura 5.4.1: Airbus A320-200 [18]

Parámetros geométricos

b_w	34.1	$[m]$	m	52650	$[kg]$
c_w	3.945	$[m]$	I_{xx}	2120270	$[kg \cdot m^2]$
S_w	122.4	$[m^2]$	I_{yy}	2382660	$[kg \cdot m^2]$
L	37.57	$[m]$	I_{zz}	1730970	$[kg \cdot m^2]$
x_{CoG}/c_w	1.214	$[-]$	I_{xz}	0	$[kg \cdot m^2]$

Tabla 5.4.1: Parámetros geométricos del A320-200

Modelo de empuje

Si bien estos aviones pueden montar varios modelos de motor, en este caso se estudiarán las características del CFM International CFM56-5B.

T_0	118000	$[N]$	ε	0	$[rad]$	BPR	6
C_e	$1.69 \cdot 10^{-5}$	$[kg/N \cdot s]$	a	1	$[-]$	N_e	2

Tabla 5.4.2: Parámetros del modelo de empuje del A320-200

Modelo aerodinámico

A continuación, se muestran los valores de las derivadas aerodinámicas pertenecientes a la dinámica longitudinal y a la lateral-direccional.

C_{L0}	0	$C_{L\alpha}$	5.582	$C_{M\alpha}$	-2.855
C_{M0}	0	$C_{L\delta E}$	1.019	$C_{M\delta E}$	-4.453
C_{D0}	0.018	C_{Lq}	8.892	C_{Mq}	-38.863
K	0.0454	$C_{L\dot{\alpha}}$	2.625	$C_{M\dot{\alpha}}$	-11.47

Tabla 5.4.3: Derivadas aerodinámicas de la dinámica longitudinal del A320-200

$C_{Y\beta}$	-0.63203	$C_{Y\dot{\beta}}$	0	C_{Yp}	-0.01065	C_{Yr}	0.05251
$C_{l\beta}$	-0.08591	$C_{l\dot{\beta}}$	0	C_{lp}	-0.04853	C_{lr}	0.02081
$C_{N\beta}$	0.08944	$C_{N\dot{\beta}}$	0	C_{Np}	-0.00994	C_{Nr}	-0.02875

Tabla 5.4.4: Derivadas aerodinámicas de la dinámica lat-dir del A320-200

$C_{Y\delta A}$	0	$C_{l\delta A}$	0.03323	$C_{N\delta A}$	0.00119
$C_{Y\delta R}$	0.21057	$C_{l\delta R}$	0.01588	$C_{N\delta R}$	-0.11605

Tabla 5.4.5: Derivadas de control de la dinámica lat-dir del A320-200

5.5. Cessna Citation II

Aunque la siguiente aeronave también esté destinada al transporte de pasajeros, su particularidad reside en su tamaño. Se trata de una Cessna Citation II (modelo 550) [19] [20], un avión civil privado. Este aspecto puede ser interesante de estudiar ya que supone cambios importantes en la geometría y, por consiguiente, en su aerodinámica.

Figura 5.5.1: Cessna Citation II [21]

Parámetros geométricos

Los parámetros geométricos y másicos más importantes aparecen resumidos en la Tabla 5.5.1.

b_w	15.99	$[m]$	m	4967	$[kg]$
c_w	2.09	$[m]$	I_{xx}	20944.1	$[kg \cdot m^2]$
S_w	31.15	$[m^2]$	I_{yy}	19330	$[kg \cdot m^2]$
L	14.54	$[m]$	I_{zz}	155716	$[kg \cdot m^2]$
x_{CoG}/c_w	0.71	$[-]$	I_{xz}	0	$[kg \cdot m^2]$

Tabla 5.5.1: Parámetros geométricos del Citation II

Modelo de empuje

Este modelo de aeronave cuenta con dos turbofan del fabricante Pratt & Whitney modelo Canada JT15D-4B. Los parámetros para un solo motor son los de la Tabla 5.5.2. Cabe mencionar que la posición de los motores no es la que se venía viendo hasta ahora, si no que, en este caso, se encuentran encastrados en la parta trasera del fuselaje.

T_0	11120	$[N]$	ε	0	$[rad]$	BPR	2.6
C_e	$1.56 \cdot 10^{-5}$	$[kg/N \cdot s]$	a	1	$[-]$	N_e	2

Tabla 5.5.2: Parámetros del modelo de empuje del Citation II

Modelo aerodinámico

C_{L0}	0	$C_{L\alpha}$	5.41974	$C_{M\alpha}$	-1.73675
C_{M0}	0	$C_{L\delta E}$	0.6477	$C_{M\delta E}$	-1.73149
C_{D0}	0.019	C_{Lq}	4.16819	C_{Mq}	-10.8821
K	0.0507	$C_{L\dot{\alpha}}$	3.58007	$C_{M\dot{\alpha}}$	-9.3467

Tabla 5.5.3: Derivadas aerodinámicas de la dinámica longitudinal del Citation II

Los valores de las derivadas aerodinámicas de la dinámica lateral-direccional aparecen recogidos en la Tablas 5.5.4 y 5.5.5.

$C_{Y\beta}$	-0.3188	$C_{Y\dot{\beta}}$	0	C_{Yp}	-0.0459	C_{Yr}	0.1811
$C_{l\beta}$	-0.0037	$C_{l\dot{\beta}}$	0	C_{lp}	-0.3988	C_{lr}	0.0835
$C_{N\beta}$	0.0452	$C_{N\dot{\beta}}$	0	C_{Np}	-0.0323	C_{Nr}	-0.0738

Tabla 5.5.4: Derivadas aerodinámicas de la dinámica lat-dir del Citation II

$C_{Y\delta A}$	0	$C_{l\delta A}$	0.04	$C_{N\delta A}$	0.0001
$C_{Y\delta R}$	0.1357	$C_{l\delta R}$	0.0015	$C_{N\delta R}$	-0.062

Tabla 5.5.5: Derivadas de control de la dinámica lat-dir del Citation II

5.6. A400M - Atlas

Para finalizar, el avión presentado a continuación aglutina diversas modificaciones con respecto a las aeronaves estudiadas que lo hacen muy interesante de analizar. Se trata del Airbus A400M Atlas, un carguero militar propulsado por cuatro motores turbohélice y que, además, cuenta con configuración de ala alta.

Figura 5.6.1: Airbus A400M - Atlas [22]

Parámetros geométricos

En la Tabla 5.6.1 aparecen sus parámetros másicos y geométricos más destacables.

b_w	42.4	$[m]$	m	103850	$[kg]$
c_w	5.711	$[m]$	I_{xx}	4839383	$[kg \cdot m^2]$
S_w	225.1	$[m^2]$	I_{yy}	5543571	$[kg \cdot m^2]$
L	45.1	$[m]$	I_{zz}	41332559	$[kg \cdot m^2]$
x_{CoG}/c_w	0.733	$[-]$	I_{xz}	0	$[kg \cdot m^2]$

Tabla 5.6.1: Parámetros geométricos del A400M

Modelo de empuje

Como se ha mencionado, el A400M dispone de cuatro motores turbohélice Europrop TP400-D6. Este cambio supone una gran novedad en los análisis hechos hasta ahora, ya que solo se habían estudiado sistemas propulsivos tipo turbojet o turbofan. De los modelos de empuje disponibles, el de J-C Wanner es el que puede proporcionar los valores de la tracción para el estudio, obteniéndose los parámetros de la Tabla 5.6.2 para un solo motor.

k_f	$8.2 \cdot 10^6$	$[W]$	ε	0	$[rad]$	BPR	$-$
C_e	$6.76 \cdot 10^{-8}$	$[kg/W \cdot s]$	λ_f	-1	$[-]$	N_e	4

Tabla 5.6.2: Parámetros del modelo de empuje del A400M

Modelo aerodinámico

De esta aeronave no se disponían los valores de las derivadas aerodinámicas, por lo que ha sido necesaria su obtención. El cálculo de las derivadas de la dinámica longitudinal se obtiene de los procedimientos recogidos en la asignatura *Mecánica del Vuelo* [23]. En la Tabla 5.6.3 se muestran los valores de estos coeficientes para el A400M.

C_{L0}	0	$C_{L\alpha}$	5.6836	$C_{M\alpha}$	-3.6937
C_{M0}	0	$C_{L\delta E}$	0.747	$C_{M\delta E}$	-0705
C_{D0}	0.0162	C_{Lq}	10.2422	C_{Mq}	-42.0977
K	0.0503	$C_{L\dot{\alpha}}$	3.1938	$C_{M\dot{\alpha}}$	-13.0152

Tabla 5.6.3: Derivadas aerodinámicas de la dinámica longitudinal del A400M

De la misma manera, se puede seguir un procedimiento análogo para calcular las derivadas aerodinámicas de la dinámica lateral-direccional, cuyos valores aparecen recogidos en la Tablas 5.6.4 y 5.6.5. En este caso el procedimiento seguido es el proporcionado por Marcello R. Napolitano [24].

$C_{Y\beta}$	-0.60984	$C_{Y\dot{\beta}}$	0	C_{Yp}	-0.10457	C_{Yr}	0.42449
$C_{l\beta}$	-0.09876	$C_{l\dot{\beta}}$	0	C_{lp}	-0.42191	C_{lr}	0.21728
$C_{N\beta}$	0.11542	$C_{N\dot{\beta}}$	0	C_{Np}	-0.10119	C_{Nr}	-0.20968

Tabla 5.6.4: Derivadas aerodinámicas de la dinámica lat-dir del A400M

$C_{Y\delta A}$	0	$C_{l\delta A}$	0.02820	$C_{N\delta A}$	-0.00077
$C_{Y\delta R}$	0.18971	$C_{l\delta R}$	0.01675	$C_{N\delta R}$	-0.10190

Tabla 5.6.5: Derivadas de control de la dinámica lat-dir del A400M

Capítulo 6

Conclusiones

Una vez que se ha completado el trabajo, se puede concluir que se han alcanzado los objetivos del mismo en un alto grado de satisfacción.

Por un lado, se ha diseñado una aplicación en MATLAB que implemente el código del simulador elaborado por L. Villanueva y se ha comprobado que la solución de las *Ecuaciones de Bryan* obtenida en ambos programas es la misma, por lo que se puede decir que la aplicación cumple su función adecuadamente. También se ha estudiado la posibilidad de utilizar otro método numérico para resolver las ecuaciones y conseguir así reducir el tiempo de cálculo en el análisis de las maniobras, llegando a la conclusión de que hay una alternativa que logra un mejor compromiso entre tiempo-error para la mayoría de operaciones simuladas.

Por otro lado, se han probado diferentes estrategias de representación en tiempo real que permiten visualizar la evolución temporal de los parámetros en cada instante, con lo que se consigue una herramienta que ayuda al entendimiento del comportamiento de dichas variables. Si bien es cierto que no se ha obtenido el mejor rendimiento en cuanto a fluidez en el graficado de las variables, el programa cumple perfectamente su función y resuelve las ecuaciones cada un intervalo muy pequeño de tiempo. Para demostrar el funcionamiento y la versatilidad de una GUI como esta se han extraído una serie de secuencias temporales en las que se muestra la evolución de la representación de las variables para distintos momentos.

El hecho de que se resuelvan las ecuaciones prácticamente en cada instante permite actuar activamente sobre los controles durante la simulación, logrando un alto grado de libertad por parte del usuario a la hora de estudiar diferentes comportamientos y dinámicas de la aeronave. De esta manera también es posible visibilizar en un mismo análisis los efectos que tienen los diferentes elementos de control sobre la misma variable.

Adicionalmente, se ha llevado a cabo una comparación de las diferencias existentes entre el uso de los Ángulos de Euler y los quaterniones de Euler-Rodrigues al emplear el conjunto de ecuaciones en Ejes Cuerpo. Se ha mostrado la evolución de las mismas variables durante la simulación de la maniobra de giro en el plano vertical y analizado las

disparidades de los resultados al aparecer el fenómeno del *gimbal lock.*

Finalmente, se ha creado una aplicación fácil de usar por el estudiantado con la que es posible analizar diferentes aeronaves, ya sean las que se han incluido en el presente trabajo a modo de biblioteca o las que resulten de interés para el propio estudiante. Para conseguir que el funcionamiento sea orgánico y ligero se han incluido botones en las diferentes partes de la aplicación con los que se busca reducir la carga de trabajo por parte del alumno a la hora de preparar las simulaciones.

6.1. Trabajo futuro

Después de haber expuesto la parte final del trabajo, se proponen diferentes alternativas con las que se podría mejorar el funcionamiento de la aplicación y la GUI para conseguir un programa más completo todavía. A continuación, se presentan algunas propuestas derivadas de la elaboración de la interfaz, y alguna ya se incluyó en el trabajo de L. Villanueva y siguen siendo interesantes.

- Al existir un retraso en el tiempo de simulación cuando se están visualizando varios parámetros sería muy conveniente buscar otra estrategia de representación con la que se consiga reducir o eliminar este retraso. Consiguiendo así que la interacción con la aplicación sea mucho más dinámica.

- Una incorporación que puede resultar muy interesante es la posibilidad de utilizar técnicas de control automático para reducir las oscilaciones que se producen al modificar los controles. Esta opción resultaría útil para comparar la respuesta natural de la aeronave y la que tendría si estuviese realizando un vuelo con una aeronave real, pues estas cuentan con sistemas de control para asegurar la comodidad de las personas a bordo.

- Incorporar nuevas funcionalidades a la aplicación, ya sea el cálculo de nuevos parámetros como la velocidad o ratio de giro, o implementar un modelo de empuje para motores turboprop que dependa de la velocidad de vuelo..

- Un análisis que suele llamar mucho la atención es el del caso de fallo de motor durante el vuelo. La integración de esta opción en el simulador supondría modificar las *Ecuaciones de Bryan* de tal manera que se tuviesen en cuenta las fuerzas y momentos generados por la pérdida de empuje en uno o varios motores de la aeronave. También sería necesario cambiar la forma en la que están expresados los modelos de empuje para incluir esta funcionalidad.

Capítulo 7

Relación del trabajo con los Objetivos de Desarrollo Sostenible de la Agenda 2030

Los ODS (Objetivos de Desarrollo Sostenible) son un conjunto de 17 objetivos que fueron establecidos por la ONU en 2015 y se encuentran recogidos en la Agenda 2030. Estos objetivos buscan que las instituciones tomen acción con el fin de proteger el planeta, erradicar la pobreza y mejorar la vida de las personas. En el presente capítulo se mostrará la relación existente entre los diferentes ODS y el Trabajo Fin de Grado.

Objetivos de Desarrollo Sostenible	Alto	Medio	Bajo	No procede
1. Fin de la pobreza				X
2. Hambre cero				X
3. Salud y bienestar				X
4. Educación de calidad		X		
5. Igualdad de género				X
6. Agua limpia y saneamiento				X
7. Energía asequible y no contaminante				X
8. Trabajo decente y crecimiento económico				X
9. Industria, innovación e infraestructura		X		
10. Reducción de las desigualdades				X
11. Ciudades y comunidades sostenibles				X
12. Producción y consumo responsable				X
13. Acción por el clima				X
14. Vida submarina				X
15. Vida de ecosistemas terrestres				X
16. Paz, justicia e instituciones sólidas				X
17. Alianzas para lograr objetivos				X

Tabla 7.0.1: Relación del TFG con los Objetivos de Desarrollo Sostenible.

De la Tabla 7.0.1 se puede extraer que hay dos objetivos con mayor relevancia que el resto sobre la relación que hay entre los ODS y la alineación del TFG. Estos dos objetivos hacen referencia a *4. Educación de calidad* y *9. Industria, innovación e infraestructura.*

Objetivo 4

Probablemente este sea el objetivo cuya relación sea más obvia con la alineación del TFG ya que la principal finalidad del mismo es el desarrollo de una herramienta académica. Con esta herramienta se pretende que el alumnado obtenga una mayor comprensión de los contenidos de la asignatura de una manera práctica y complementaria al resto de formación que se le proporciona.

Objetivo 9

De los tres elementos que aglutina este objetivo, el que establece mayor relación con el TFG es el de la innovación. Tanto este trabajo como el de L. Villanueva buscan solventar la escasez de simuladores académicos mediante la creación de un programa inexistente hasta la fecha. Para ello se ha intentado utilizar las mejores herramientas de simulación al alcance de los desarrolladores en el tiempo de elaboración del trabajo.

Capítulo 8

Pliego de condiciones

El pliego de condiciones es un documento contractual que recoge todas las normas, instrucciones, recomendaciones y directrices técnicas que afectan al presente Trabajo Fin de Grado.

8.1. Regulaciones de carácter normativo

En el presente apartado se pretenden detallar algunas de las leyes y normativas que rigen y velan por la salud de los trabajadores en sus puestos de trabajo. Se ha recopilado información de aquellas que tienen mayor relevancia e influencia sobre el trabajo expuesto en este documento debido a su naturaleza, siendo el caso de la *"Ordenanza General de Seguridad e Higiene en el Trabajo"*, el *Real Decreto 486/1997*, que trata sobre las disposiciones mínimas de seguridad y salud en los puestos de trabajo, y el *Real Decreto 488/1997* en el que se particulariza la normativa anterior para trabajos con pantallas de visualización.

8.1.1. Orden de 9 de marzo de 1971

Con esta Orden se aprueba la Ordenanza General de Seguridad e Higiene en el Trabajo. Dicha Ordenanza se encuentra recogida en el Boletín Oficial del Estado (referencia BOE-A-1971-380) y cuyas Disposiciones Generales establecen las obligaciones y derechos que tienen los trabajadores.

Artículo 11. *Obligaciones y derechos de los trabajadores.*

Incumbe a los trabajadores la obligación de cooperar en la prevención de riesgos profesionales en la Empresa y el mantenimiento de la máxima higiene en la misma, a cuyos fines deberán cumplir fielmente los preceptos de esta Ordenanza y sus instrucciones complementarias, así como las órdenes e instrucciones que a tales efectos les sean dados por sus superiores.

Los trabajadores, expresamente, están obligados a:

1. *Recibir las enseñanzas sobre Seguridad e Higiene y sobre salvamento y socorrismo en los centros de trabajo que les sean facilitadas por la Empresa o en las Instituciones del Plan Nacional.*

2. *Usar correctamente los medios de protección personal y cuidar de su perfecto estado y conservación.*

3. *Dar cuenta inmediata a sus superiores de las averías y deficiencias que puedan ocasionar peligros en cualquier centro o puesto de trabajo.*

4. *Cuidar y mantener su higiene personal, en evitación de enfermedades contagiosas o de molestias a sus compañeros de trabajo.*

5. *Someterse a los reconocimientos médicos preceptivos y a las vacunaciones o inmunizaciones ordenadas por las Autoridades Sanitarias competentes o por el Servicio Médico de Empresa.*

6. *No introducir bebidas u otras sustancias no autorizadas en los centros de trabajo, ni presentarse o permanecer en los mismos en estado de embriaguez o de cualquier otro género de intoxicación.*

7. *Cooperar en la extinción de siniestros y en el salvamento de las víctimas de accidentes de trabajo en las condiciones que, en cada caso, fueren racionalmente exigibles*

8.1.2. Real Decreto 486/1997, de 14 de abril

Tras cumplirse 25 años de la entrada en vigor de la Orden de 9 de marzo de 1971 se entiende que, por compromisos internacionales o por la evolución del progreso técnico, es necesario derogar algunos capítulos del Título II recogidos en dicha Ordenanza. Gran parte de los temas tratados en este Real Decreto habían estado regulados por dicha Ordenanza hasta la fecha.

En el RD se recogen una serie de Anexos con las características que debe tener el espacio de trabajo para ser adecuado en ámbitos como el orden y la limpieza, las condiciones ambientales, la iluminación o el material de primeros auxilios. Pero debido a la naturaleza del Trabajo de Fin de Grado y el equipo usado para su elaboración, se cree más conveniente exponer dichas condiciones para el caso particular de los puestos de trabajo con pantallas de visualización.

8.1.3. Real Decreto 488/1997, de 14 de abril

En el Real Decreto 488/1997 se establecen las normas y reglamentos por las que se protegen a todas aquellas personas que trabajan con pantallas de visualización, habitualmente en trabajos de oficinas. A continuación se muestran los anexos de dicho RD en los que se recogen las disposiciones mínimas sobre el equipo utilizado y el entorno de trabajo.

Anexo *Disposiciones mínimas.*

1. *Equipo*

   a) *Observación general. La utilización en sí misma del equipo no debe ser una fuente de riesgo para los trabajadores.*

   b) *Pantalla.*
   Los caracteres de la pantalla deberán estar bien definidos y configurados de forma clara, y tener una dimensión suficiente, disponiendo de un espacio adecuado entre los caracteres y los renglones.
   La imagen de la pantalla deberá ser estable, sin fenómenos de destellos, centelleos u otras formas de inestabilidad.
   El usuario de terminales con pantalla deberá poder ajustar fácilmente la luminosidad y el contraste entre los caracteres y el fondo de la pantalla, y adaptarlos fácilmente a las condiciones del entorno.
   La pantalla deberá ser orientable e inclinable a voluntad, con facilidad para adaptarse a las necesidades del usuario.
   Podrá utilizarse un pedestal independiente o una mesa regulable para la pantalla.
   La pantalla no deberá tener reflejos ni reverberaciones que puedan molestar al usuario.

   c) *Teclado. El teclado deberá ser inclinable e independiente de la pantalla para permitir que el trabajador adopte una postura cómoda que no provoque cansancio en los brazos o las manos. Tendrá que haber espacio suficiente delante del teclado para que el usuario pueda apoyar los brazos y las manos.*
   La superficie del teclado deberá ser mate para evitar los reflejos.
   La disposición del teclado y las características de las teclas deberán tender a facilitar su utilización.
   Los símbolos de las teclas deberán resaltar suficientemente y ser legibles desde la posición normal de trabajo.

d) *Mesa o superficie de trabajo.*

La mesa o superficie de trabajo deberán ser poco reflectantes, tener dimensiones suficientes y permitir una colocación flexible de la pantalla, del teclado, de los documentos y del material accesorio.

El soporte de los documentos deberá ser estable y regulable y estará colocado de tal modo que se reduzcan al mínimo los movimientos incómodos de la cabeza y los ojos.

El espacio deberá ser suficiente para permitir a los trabajadores una posición cómoda.

e) *Asiento de trabajo.*

El asiento de trabajo deberá ser estable, proporcionando al usuario libertad de movimiento y procurándole una postura confortable.

La altura del mismo deberá ser regulable.

El respaldo deberá ser reclinable y su altura ajustable.

Se pondrá un reposapiés a disposición de quienes lo deseen.

2. Entorno

a) *Espacio.*

El puesto de trabajo deberá tener una dimensión suficiente y estar acondicionado de tal manera que haya espacio suficiente para permitir los cambios de postura y movimientos de trabajo.

b) *Iluminación.*

La iluminación general y la iluminación especial (lámparas de trabajo), cuando sea necesaria, deberán garantizar unos niveles adecuados de iluminación y unas relaciones adecuadas de luminancias entre la pantalla y su entorno, habida cuenta del carácter del trabajo, de las necesidades visuales del usuario y del tipo de pantalla utilizado.

El acondicionamiento del lugar de trabajo y del puesto de trabajo, así como la situación y las características técnicas de las fuentes de luz artificial, deberán coordinarse de tal manera que se eviten los deslumbramientos y los reflejos molestos en la pantalla u otras partes del equipo.

c) *Reflejos y deslumbramientos.*

Los puestos de trabajo deberán instalarse de tal forma que las fuentes de luz, tales como ventanas y otras aberturas, los tabiques transparentes o translúcidos y los equipos o tabiques de color claro no provoquen deslumbramiento directo ni produzcan reflejos molestos en la pantalla.

Las ventanas deberán ir equipadas con un dispositivo de cobertura adecuado y regulable para atenuar la luz del día que ilumine el puesto de trabajo.

d) *Ruido.*
El ruido producido por los equipos instalados en el puesto de trabajo deberá tenerse en cuenta al diseñar el mismo, en especial para que no se perturbe la atención ni la palabra.

e) *Calor.*
Los equipos instalados en el puesto de trabajo no deberán producir un calor adicional que pueda ocasionar molestias a los trabajadores.

f) *Emisiones.*
Toda radiación, excepción hecha de la parte visible del espectro electromagnético, deberá reducirse a niveles insignificantes desde el punto de vista de la protección de la seguridad y de la salud de los trabajadores.

g) *Humedad.*
Deberá crearse y mantenerse una humedad aceptable.

3. *Interconexión ordenador/persona*
Para la elaboración, la elección, la compra y la modificación de programas, así como para la definición de las tareas que requieran pantallas de visualización, el empresario tendrá en cuenta los siguientes factores:

a) *El programa habrá de estar adaptado a la tarea que deba realizarse.*

b) *El programa habrá de ser fácil de utilizar y deberá, en su caso, poder adaptarse al nivel de conocimientos y de experiencia del usuario; no deberá utilizarse ningún dispositivo cuantitativo o cualitativo de control sin que los trabajadores hayan sido informados y previa consulta con sus representantes.*

c) *Los sistemas deberán proporcionar a los trabajadores indicaciones sobre su desarrollo.*

d) *Los sistemas deberán mostrar la información en un formato y a un ritmo adaptados a los operadores.*

e) *Los principios de ergonomía deberán aplicarse en particular al tratamiento de la información por parte de la persona.*

8.2. Regulaciones específicas

Este aparatado se centrará en la exposición de las especificaciones técnicas particulares que han sido necesarias para llevar a cabo el desarrollo del trabajo de forma correcta.

8.2.1. Condiciones de los materiales

La elaboración del proyecto se ha llevado a cabo en un entorno adecuado y cuyas herramientas principales han sido programas software y un equipo donde poder trabajar con él.

Hardware El equipo utilizado para la realización de las tareas exigidas por el proyecto debe ser lo suficientemente potente como para soportar una alta demanda de recursos durante un periodo de tiempo prolongado. Para la elaboración del trabajo ha sido suficiente con un portátil con un buen procesador, aunque es aconsejable el uso de un equipo de sobremesa. Las características del equipo utilizado son:

- Sistema operativo: Windows 11 Home
- Procesador: Intel(R) Core (TM) i7-10710U CPU
- Memoria RAM: 16 GB
- Disco duro: 1 TB
- Tarjeta gráfica: NVIDIA GeForce GTX 1650

Software Para la elaboración del TFG se han utilizado diversos programas, entre los que hay algunos de acceso libre y gratuito y otros cuyas licencias vienen proporcionadas por la Universidad Politécnica de Valencia.

- Overleaf
- MATLAB 2024a
- Mathematica 14.0
- Microsoft 365
- Zotero

Entorno El entorno de trabajo cuenta con un escritorio amplio con el que se permite trabajar cómodamente, una fuente de luz natural que favorece la visión sobre el área de

trabajo y un correcto aislamiento térmico y acústico que proporciona un ambiente desahogado para el desarrollo de las tareas.

Documentación Para la elaboración del Trabajo Fin de Grado ha sido necesario recopilar y consultar documentación en función de las tareas que se estuviesen llevando a cabo en cada momento. En algunos casos esta documentación pertenecía a la disponible en la asignatura *Ampliación de Mecánica del Vuelo* y en otros casos ha sido necesario consultar diversas fuentes de internet, entre las que destaca la documentación de MATLAB y su foro de preguntas.

8.2.2. Control de calidad

La empresa u organización que lleve a cabo el proyecto debe contar con los certificados pertinentes que aseguren la calidad final del producto y que su desarrollo se ha realizado bajo la normativa aplicable. Entre las regulaciones existentes sobre calidad destacan dos en particular que se enfocan en los estándares de la calidad de proyectos de software:

- *ISO-9001: Norma para la implementación de un método o Sistema de Gestión de la Calidad (SGC), supone la acreditación de la capacidad para satisfacer los requisitos de calidad.*
- *ISO-IEC-25000: Supone una familia de normas que define un marco de referencia para la calidad del producto de software.*

Capítulo 9

Presupuesto

En el último capítulo del Trabajo Fin de Grado se va a proceder al cálculo del coste estimado que habría tenido el desarrollo del proyecto si hubiese venido motivado por una oferta de una organización externa.

9.1. Costes directos

Cuando se habla de costes directos de un proyecto se hace referencia a los costes que guardan una estrecha relación con su desarrollo e inciden directamente en la producción y la fijación de su precio. Para el proyecto particular que se ha elaborado se tendrán en cuenta los costes asociados al personal implicado, los relacionados con el equipo utilizado y las licencias del software empleado.

En primer lugar, se estimarán los costes relacionados con el personal que ha participado en el desarrollo del trabajo. Por un lado, se calculará el sueldo del tutor en base a las retribuciones de cargos públicos publicados por la universidad. Por otro lado, se asumirá que el trabajo del alumno se pagaría como si tuviese un puesto de ingeniero aeroespacial junior y su sueldo se puede aproximar de la información hallada en internet.

Para calcular el sueldo bruto por horas en ambos casos de dividirá el sueldo bruto anual por el número de horas laborables de un año, tomándose como referencia las del año 2023. Si el sueldo bruto de un profesor titular ronda los 33300 €, y el de un ingeniero junior puede estar entorno a 25000€, al dividirlo por 1776 horas laborables quedan los siguientes ratios:

$$\text{Tarifa profesor} = \frac{33300}{1776} = 18.75\ €/\text{h}$$

$$\text{Tarifa ingeniero} = \frac{25000}{1776} = 14.08\ €/\text{h}$$

En la primera parte de la Tabla 9.1.1 se calcula el precio final de los servicios del personal en función de las horas trabajadas.

La siguiente estimación tiene que ver con el equipo con el que se ha trabajado. Para calcular este coste será necesario obtener la amortización del equipo descrito en el Capítulo 8 y posteriormente calcular la tasa horaria.

- Amortización:

$$a = \frac{V_c - V_R}{n}$$

donde V_c es el coste de adquisición, V_R es el valor residual y n el periodo de amortización.

Para calcular la amortización del ordenador portátil, modelo *MSI Prestige 15-A10SC*, se contará con un valor residual del 30 % y un periodo de amortización igual a 5 años.

$$a = \frac{1349 - 404.7}{5} = 188.86 \text{ €/año}$$

$$t_h = \frac{a}{h} = \frac{188.86}{1776} = 0.106 \text{ €/h}$$

Para el ratón se tendrá en cuenta únicamente el coste de adquisición.

Finalmente, se debe calcular el coste que suponen las diversas licencias del software utilizado en la elaboración del proyecto. Se tendrán en cuenta solo las licencias de MATLAB, Mathematica y Office 365 pues el resto de programas son de uso libre y gratuito.

- Licencia de MATLAB 2024a
  El precio de una licencia académica anual es de 262 €.

$$t_h = \frac{262}{1776} = 0.148 \text{ €/h}$$

- Licencia de Mathematica 14.0
  La licencia académica anual tiene un precio de 1185 €.

$$t_h = \frac{1185}{1776} = 0.667 \text{ €/h}$$

- Licencia de Office 365
  La licencia básica mensual para empresas tiene un precio de 5.60 €/h.

$$t_h = \frac{5.6 \cdot 12}{1776} = 0.038 \text{ €/h}$$

En la Tabla 9.1.1 se muestran desglosados todos los costes que repercuten en el cálculo total de los costes directos del trabajo.

Personal			
	Tiempo	**Coste unitario**	**Valor final**
Ingeniero aeroespacial junior	370	14.08	5209.6
Profesor titular de universidad	35	18.75	656.25
Subtotal			**5865.85 €**
Equipo			
	Tiempo	**Coste unitario**	**Valor final**
Ordenador portátil	370	0.106	39.92
Ratón			14.95
Subtotal			**54.17 €**
Software			
	Tiempo	**Coste unitario**	**Valor final**
Licencia MATLAB 2024a	280	0.148	41.44
Licencia Mathematica 14.0	20	0.667	13.34
Licencia Office 365	70	0.038	2.66
Subtotal			**57.44 €**
COSTES DIRECTOS TOTALES			**5977.46 €**

Tabla 9.1.1: Costes Directos

9.2. Costes indirectos

Los costes indirectos son aquellos que no pueden imputarse al coste del servicio o producto desarrollado. Se relacionan principalmente con los costes que supone la electricidad, el agua, el mobiliario, etc.

Para estimar el valor de los costes indirectos se acude al Real Decreto 1098/2001, de 12 de octubre, por el que se aprueba el Reglamento general de la Ley de Contratos de las Administraciones Públicas. En el RD se establece que los costes indirectos se pueden considerar entre un 13 % y 18 % de los costes directos.

Tomando un valor intermedio del rango de porcentaje, si se estima que suponen un 15 % de los costes directos, los costes indirectos ascienden a: **896.62 €/h**.

9.3. Beneficio industrial

Finalmente, si el proyecto se llevase a cabo, la organización encargada de la producción debe llevarse beneficios industriales aparte de recuperar la inversión realizada. Del Real Decreto 1098/2001 se extrae que este beneficio debe ser del 6 % del valor de los costes directos.

Por tanto, se tiene una cuantía de beneficio industrial de: **358.65 €/h**.

9.4. Coste Total

	Valor	21 % IVA	Coste
Costes directos	5977.46	1255.27	7232.73
Costes indirectos	896.62	188.29	1084.91
Beneficio industrial	358.65	75.32	433.97
Total	7232.73	1518.88	**8751.61 €**

Tabla 9.4.1: Coste Total

El coste total del proyecto asciende a:

OCHO MIL SETECIENTOS CINCUENTA Y UN EUROS CON SESENTA Y UN CÉNTIMOS

Bibliografía

[1] M. Portuario, *Centro de Airbus en Miami incorpora nuevo simulador de vuelo*, 2016. dirección: `https://mundoportuario.com/2016/04/centro-airbus-miami-incorpora-nuevo-simulador-vuelo/`.

[2] L. Villanueva Chulvi, "Development of an academic flight simulator for the study of the flight dynamics of conventional configuration aircraft", Tesis doct., Universitat Politècnica de València, 2023.

[3] S. Barnes, "User Friendly: A Short History of the Graphical User Interface", *Sacred Heart University Review*, vol. 16, 2010.

[4] J. Reimer, *A History of the GUI*, 2005. dirección: `https://arstechnica.com/features/2005/05/gui/`.

[5] C. Depcik y D. N. Assanis, "Graphical user interfaces in an engineering educational environment", *Computer Applications in Engineering Education*, vol. 13, págs. 48-59, 2005.

[6] P. Martí Gómez-Aldaraví y M. Carreres Talens, "Dinámica de un avión rígido. Parte A", *Ampliación de Mecánica del Vuelo*, 2022.

[7] P. Martí Gómez-Aldaraví y M. Carreres Talens, "Dinámica de un avión rígido. Parte B", *Ampliación de Mecánica del Vuelo*, 2022.

[8] M. Answers, *How to calculate error between 2 curves?*, 2020. dirección: `https://es.mathworks.com/matlabcentral/answers/531668-how-to-calculate-error-between-2-curves`.

[9] M. D. Roselló Ferragud, D. Ginestar Peiro y S. Blanes Zamora, *Introducción a los métodos numéricos para ecuaciones diferenciales*. Editorial Universitat Politècnica de València, 2020.

[10] MathWorks-España, *Elegir un solver de ODE*, (s.f.) dirección: `https://es.mathworks.com/help/matlab/math/choose-an-ode-solver.html`.

[11] L. F. Shampine y M. W. Reichelt, "The MATLAB ODE Suite", *SIAM Journal on Scientific Computing*, vol. 18, págs. 1-22, 1997.

[12] AirCharteradvisors, *Cessna Citation II*, 2012. dirección: https://www.aircharteradvisors.com/cessna-citation-ii/.

[13] P. Martí Gómez-Aldaraví y M. Carreres Talens, "Documentación de la asignatura.", *Mecánica del Vuelo. Ampliación de Mecánica del Vuelo.*, 2023.

[14] S. E. Harris, *KC-10 Refueling Operations*, 2012. dirección: https://www.dvidshub.net/image/344929/kc-10-refueling-operations.

[15] A. Zuazaga Calvo, "Trabajo Académico", *Ampliación de Mecánica del vuelo*, 2019.

[16] AEROBROKER, *B737-500*, 2021. dirección: https://aerobroker.es/aviones/b737-500/.

[17] B. Gómez Montañana, "Trabajo Académico", *Ampliación de Mecánica del vuelo*, 2023.

[18] J. Herzog, *Lufthansa Airbus A320-211*, 2013. dirección: https://commons.wikimedia.org/wiki/File:Lufthansa_Airbus_A320-211_D-AIQT_01.jpg.

[19] M. Ponce Argilés, "Trabajo Académico", *Ampliación de Mecánica del vuelo*, 2022.

[20] P. Gasco Casado, "Trabajo Académico", *Ampliación de Mecánica del vuelo*, 2019.

[21] AirCharteradvisors, *Cessna Citation II*, 2012. dirección: https://www.aircharteradvisors.com/cessna-citation-ii/.

[22] MilborneOne, *A400M*, 2013. dirección: https://commons.wikimedia.org/wiki/File:A400M-1969.jpg.

[23] P. Martí Gómez-Aldaraví y M. Carreres Talens, "Capítulo 0. Procedimientos.", *Mecánica del Vuelo*, 2023.

[24] M. R. Napolitano, "Aircraft dynamics: from modeling to simulation", *John Wiley Sons, Hoboken*, págs. 135-189, 2012.

Apéndice A

Código de MATLAB

A.1. Método de Euler

```
% Metodo de Euler
% f: campo vectorial
% x0: condiciones iniciales
% t0, tf: tiempo inicial y final
% N: divisiones del intervalo temporal (paso de integraci n h=(
   tf-t0)/N)

function [t,x] = Euler(f,t0,tf,x0,N)
h = (tf - t0)/N;
t = t0:h:tf;
x=x0';
y = x0;
for i = 1:N
    K1 = f(t(i),y);
    y = y + (K1)*h;
    x = [x;y'];
end
```

A.2. Método de Euler modificado

```
% Euler modificado
% f: campo vectorial
% x0: condiciones iniciales
% t0, tf: tiempo inicial y final
% N: divisiones del intervalo temporal (paso de integraci n h=(
   tf-t0)/N)

```

```
function [t,x] = Eulermod(f,t0,tf,x0,N)
h = (tf-t0)/N;
t=t0:h:tf;
x=x0';
y=x0;
for i=1:N
  t1 = t(i) + h;
  K1 = f(t(i),y);
  K2 = f(t1,y+h*K1);
  y = y + (K1 + K2)*h/2;
  x = [x ; y' ];
end
```

A.3. Método de Heun

```
% Heun
% f: campo vectorial
% x0: condiciones iniciales
% t0, tf: tiempo inicial y final
% N: divisiones del intervalo temporal (paso de integraci n h=(
   tf-t0)/N)

function [t,x] = Heun(f,t0,tf,x0,N)
h = (tf-t0)/N;
t=t0:h:tf;
x=x0';
y=x0;
for i=1:N
  t1 = t(i) + 2*h/3;
  K1 = f(t(i),y);
  K2 = f(t1,y+2*h*K1/3);
  y = y + (K1 + 3*K2)*h/4;
  x = [x ; y' ];
end
```

A.4. Método de Runge-Kutta estándar de orden 4

```
% 4-stage 4th-order estandard RK method
% f: vectorfield
% x0: initial conditions
% t0, tf: time span
```

```
% N: divisions of the time interval (timestep h=(tf-t0)/N)

function [t x] = rk4(f,t0,tf,x0,N)
h = (tf-t0)/N;
t=t0:h:tf; x=x0';y=x0;
for i=1:N;
  t1 = t(i) + h/2; t2 = t(i) + h/2; t3 = t(i) + h;
  K1 = f(t(i),y);
  K2 = f(t1,y+h*K1/2);
  K3 = f(t2,y+h*K2/2);
  K4 = f(t3,y+h*K3);
  y = y + (K1 + 2*K2 + 2*K3 + K4)*h/6;
  x = [x ; y' ];
end
```

A.5. Método predictor-corrector de orden 4

```
% Predictor: 4-stage explicit Adams-Bashforth method
% Corrector: 4-stage implicit Adams-Moulton
% The algorithm starts with two steps of the RK4 method
% f: vectorfield
% x0: initial conditions
% t0, tf: time span
% N: divisions of the time interval (timestep h=(tf-t0)/N)

function [t,x] = pcABM4(f,t0,tf,x0,N)

h = (tf - t0)/N; %Integration step
t = t0:h:tf;
x = x0';
y = x0;

npi = 3;  %3 pasos para Runge-Kutta orden 4
[trk4,x] = rk4(f,t0,t0 + npi*h,x0,npi);

%Condiciones iniciales y primeros 3 pasos del m todo
y0 = x(1,:)';   f0 = f(t0,y0);
y1 = x(2,:)';   f1 = f(t0 + h,y1);
y2 = x(3,:)';   f2 = f(t0 + 2*h,y2);
y3 = x(4,:)';   f3 = f(t0 + 3*h,y3);

```

```
25
26 for i = 4:N
27     %PREDICTOR:
28     y3p = y3 + h/24*(55*f3 - 59*f2 + 37*f1 - 9*f0);
29     f3p = f(t0 + i*h,y3p);
30     %CORRECTOR:
31     y3 = y3 + h/720*(251*f3p + 646*f3 - 264*f2 + 106*f1 - 19*f0);
32     f0 = f1;
33     f1 = f2;
34     f2 = f3;
35     f3 = f(t0 + i*h,y3); %Modificamos f0, f1 y f2 para los
       siguientes pasos
36     x = [x;y3'];
37 end
38 end
```

A.6. Función de inicio de la aplicación

```
727 % Code that executes after component creation
728 function startupFcn(app)
729     evalin('base','clear all');
730     clc
731
732     files = dir('*/AircraftParameters_*.m');
733     file_names = {files(1).name};
734
735     for i = 2:length(files)
736         file_names = cat(1,file_names,{files(i).name});
737     end
738     file_names = string(file_names);
739
740     airplanes = erase(file_names,'AircraftParameters_');
741     airplanes = erase(airplanes,'.m');
742     airplanesNew = replace(airplanes,"_","-");
743     app.AvindeestudioDropDown.Items = airplanesNew;
744 end
```

A.7. Carga de datos de la aeronave

```
746 % Button pushed function: CargardatosdelavinButton
747 function CargardatosdelavinButtonPushed(app, event)
748     airplane = app.AvindeestudioDropDown.Value;
749     airplaneOld = replace(airplane,"-","_");
750 
751     scriptname = ['AircraftParameters_' airplaneOld];
752     scriptname = string(scriptname);
753     pathStr = ["Aircrafts_Parameters\",scriptname];
754     path = join(pathStr,"");
755 
756     run(path)
757 
758     app.ThrustModel = modelchar2num(app,app.model);
759     if app.ThrustModel == 1
760         if isnan(kf) || isnan(lambdaf)
761             uiwait(msgbox('El modelo propulsivo no se ha
    completado correctamente.','Error.','error'));
762         else
763             app.kfEditField.Value = kf;
764             app.lambdafEditField.Value = lambdaf;
765             app.aEditField.Value = 0;
766             app.BPREditField.Value = 0;
767             app.BPREditField_2.Value =0;
768             app.nparameterEditField.Value = 0;
769             app.k1EditField.Value = 0;
770             app.k2EditField.Value = 0;
771             app.k3EditField.Value = 0;
772             app.k4EditField.Value = 0;
773 
774 
775             app.kf = kf;
776             app.lambdaf = lambdaf;
777         end
778 
779     elseif app.ThrustModel == 2 || app.ThrustModel == 3
780         app.aEditField.Value = 0;
781         app.kfEditField.Value = 0;
782         app.lambdafEditField.Value = 0;
```

```
783         app.BPREditField.Value = 0;
784         app.BPREditField_2.Value =0;
785         app.nparameterEditField.Value = 0;
786         app.k1EditField.Value = 0;
787         app.k2EditField.Value = 0;
788         app.k3EditField.Value = 0;
789         app.k4EditField.Value = 0;
790 
791     elseif app.ThrustModel == 4
792         if isnan(a)
793             uiwait(msgbox('El modelo propulsivo no se ha
    completado correctamente.','Error.','error'));
794         else
795             app.aEditField.Value = a;
796             app.kfEditField.Value = 0;
797             app.lambdafEditField.Value = 0;
798             app.BPREditField.Value = 0;
799             app.BPREditField_2.Value =0;
800             app.nparameterEditField.Value = 0;
801             app.k1EditField.Value = 0;
802             app.k2EditField.Value = 0;
803             app.k3EditField.Value = 0;
804             app.k4EditField.Value = 0;
805 
806             app.a = a;
807         end
808 
809     elseif app.ThrustModel == 5
810         if isnan(BPR)
811             uiwait(msgbox('El modelo propulsivo no se ha
    completado correctamente.','Error.','error'));
812         else
813             app.BPREditField.Value = BPR;
814             app.aEditField.Value = 0;
815             app.kfEditField.Value = 0;
816             app.lambdafEditField.Value = 0;
817             app.BPREditField_2.Value =0;
818             app.nparameterEditField.Value = 0;
819             app.k1EditField.Value = 0;
820             app.k2EditField.Value = 0;
```

```
821             app.k3EditField.Value = 0;
822             app.k4EditField.Value = 0;
823 
824             app.BPR = BPR;
825         end
826 
827     elseif app.ThrustModel == 6
828         if isnan(BPR) || isnan(nparameter) || isnan(k1) || isnan(
    k2) || isnan(k3) || isnan(k4)
829             uiwait(msgbox('El modelo propulsivo no se ha
    completado correctamente.','Error.','error'));
830         else
831             app.BPREditField_2.Value = BPR;
832             app.nparameterEditField.Value = nparameter;
833             app.k1EditField.Value = k1;
834             app.k2EditField.Value = k2;
835             app.k3EditField.Value = k3;
836             app.k4EditField.Value = k4;
837             app.BPREditField.Value = 0;
838             app.aEditField.Value = 0;
839             app.kfEditField.Value = 0;
840             app.lambdafEditField.Value = 0;
841 
842             app.BPR = BPR;
843             app.nparameter = nparameter;
844             app.k1 = k1;
845             app.k2 = k2;
846             app.k3 = k3;
847             app.k4 = k4;
848         end
849     end
850 
851     app.bwEditField.Value = bw;
852     app.SwEditField.Value = Sw;
853     app.ARwEditField.Value = ARw;
854     app.xCoGEditField.Value = xCoG;
855     app.cwEditField.Value = cw;
856     app.eEditField.Value = e;
857     app.mEditField.Value = m;
858     app.IxxEditField.Value = Ixx;
```

```
859     app.IyyEditField.Value = Iyy;
860     app.IzzEditField.Value = Izz;
861     app.IxzEditField.Value = Ixz;
862     app.ThrustSLEditField.Value = ThrustSL;
863     app.CeEditField.Value = Ce;
864 
865     app.KEditField.Value = K;
866     app.CD0EditField.Value = CD0;
867     app.CL0EditField.Value = CL0;
868     app.CLalphaEditField.Value = CLalpha;
869     app.CLdEEditField_2.Value = CLdE;
870     app.CLalphaDotEditField.Value = CLalphaDot;
871     app.CLqEditField.Value = CLq;
872     app.CM0EditField.Value = CM0;
873     app.CMalphaEditField.Value = CMalpha;
874     app.CMdEEditField.Value = CMdE;
875     app.CMalphaDotEditField.Value = CMalphaDot;
876     app.CMqEditField.Value = CMq;
877 
878     app.CYbetaEditField.Value = CYbeta;
879     app.CYbetaDotEditField.Value = CYbetaDot;
880     app.CYpEditField.Value = CYp;
881     app.CYrEditField.Value = CYr;
882     app.CYdAEditField.Value = CYdA;
883     app.CYdREditField.Value = CYdR;
884     app.ClbetaEditField.Value = Clbeta;
885     app.ClbetaDotEditField.Value = ClbetaDot;
886     app.ClpEditField.Value = Clp;
887     app.ClrEditField.Value = Clr;
888     app.CldAEditField.Value = CldA;
889     app.CldREditField.Value = CldR;
890     app.CNbetaEditField.Value = CNbeta;
891     app.CNbetaDotEditField.Value = CNbetaDot;
892     app.CNpEditField.Value = CNp;
893     app.CNrEditField.Value = CNr;
894     app.CNdAEditField.Value = CNdA;
895     app.CNdREditField.Value = CNdR;
896 
897     app.bw = bw;
898     app.Sw = Sw;
```

```
899     app.ARw = ARw;
900     app.xCoG = xCoG;
901     app.cw = cw;
902     app.e = e;
903     app.m = m;
904     app.Ixx = Ixx;
905     app.Iyy = Iyy;
906     app.Izz = Izz;
907     app.Ixz = Ixz;
908     app.ThrustSL = ThrustSL;
909     app.Ce = Ce;
910 
911     app.K = K;
912     app.CD0 = CD0;
913     app.CL0 = CL0;
914     app.CLalpha = CLalpha;
915     app.CLdE = CLdE;
916     app.CLalphaDot = CLalphaDot;
917     app.CLq= CLq;
918     app.CM0 = CM0;
919     app.CMalpha = CMalpha;
920     app.CMdE = CMdE;
921     app.CMalphaDot = CMalphaDot;
922     app.CMq = CMq;
923 
924     app.CYbeta= CYbeta;
925     app.CYbetaDot = CYbetaDot;
926     app.CYp = CYp;
927     app.CYr = CYr;
928     app.CYdA = CYdA;
929     app.CYdR = CYdR;
930     app.Clbeta = Clbeta;
931     app.ClbetaDot = ClbetaDot;
932     app.Clp = Clp;
933     app.Clr = Clr;
934     app.CldA = CldA;
935     app.CldR = CldR;
936     app.CNbeta = CNbeta;
937     app.CNbetaDot = CNbetaDot;
938     app.CNp = CNp;
```

```
939     app.CNr = CNr;
940     app.CNdA = CNdA;
941     app.CNdR = CNdR;
942 
943 end
```

A.8. Configuración del simulador

```
1363 % Button pushed function: InicializarButton
1364         function InicializarButtonPushed(app, event)
1365             switch app.axisSel
1366                 case 'Ejes cuerpo ( ngulos  de Euler)'
1367                     if app.manoeuvreSelB == 1
1368                         app.u0 = app.V0*cos(app.alpha0)*cos(app.
    beta0);
1369                         app.v0 = app.V0*sin(app.beta0);
1370                         app.w0 = app.V0*sin(app.alpha0)*cos(app.
    beta0);
1371 
1372                         app.theta0B = app.alpha0;
1373 
1374                     elseif app.manoeuvreSelB == 2
1375                         app.u0 = app.V0*cos(app.alpha0)*cos(app.
    beta0);
1376                         app.v0 = app.V0*sin(app.beta0);
1377                         app.w0 = app.V0*sin(app.alpha0)*cos(app.
    beta0);
1378 
1379                         app.theta0B = app.alpha0 + app.gamma0B;
1380 
1381                     elseif app.manoeuvreSelB == 3
1382                         app.u0 = app.V0*cos(app.alpha0)*cos(app.
    beta0);
1383                         app.v0 = app.V0*sin(app.beta0);
1384                         app.w0 = app.V0*sin(app.alpha0)*cos(app.
    beta0);
1385 
1386                         app.theta0B = alpha0B - pi/2;
1387 
1388                     elseif app.manoeuvreSelB == 4 || app.
    manoeuvreSelB == 5
```

```
1389                        Radius = app.V0/app.OMEGAB;
1390
1391                        app.u0 = app.V0*cos(app.alpha0)*cos(app.
    beta0);
1392                        app.v0 = app.V0*sin(app.beta0);
1393                        app.w0 = app.V0*sin(app.alpha0)*cos(app.
    beta0);
1394
1395                        psiDot0 = app.OMEGAB;
1396                        phiDot0 = 0;
1397                        thetaDot0 = 0;
1398
1399                        app.p0B = phiDot0-psiDot0*sin(app.theta0B
    );
1400                        app.q0B = thetaDot0*cos(app.phi0B)+
    psiDot0*cos(app.theta0B)*sin(app.phi0B);
1401                        app.r0B = psiDot0*cos(app.theta0B)*cos(
    app.phi0B)-thetaDot0*sin(app.phi0B);
1402                    end
1403
1404                case 'Ejes viento'
1405                    if app.manoeuvreSelW == 1 || app.
    manoeuvreSelW == 2 || app.manoeuvreSelW == 3 || app.
    manoeuvreSelW == 4
1406
1407                        app.p0W = app.p0EditField.Value;
1408                        app.q0W = app.q0EditField.Value;
1409                        app.r0W = app.r0EditField.Value;
1410
1411                        app.mu0W = app.mu0EditField.Value;
1412                        app.gamma0W = app.gamma0EditField.Value;
1413                        app.chi0W = app.chi0EditField.Value;
1414
1415                    elseif app.manoeuvreSelW == 5 || app.
    manoeuvreSelW == 6
1416                        Radius = app.V0/app.OMEGAW;
1417
1418                        app.p0W = -app.OMEGAW*(cos(app.mu0W)*sin(
    app.alpha0)+cos(app.alpha0)*sin(app.beta0)*sin(app.mu0W));
1419                        app.q0W = app.OMEGAW*cos(app.beta0)*sin(
```

```
app.mu0W);
                    app.r0W = app.OMEGAW*(cos(app.mu0W)*cos(
app.alpha0)-sin(app.alpha0)*sin(app.beta0)*sin(app.mu0W));

                end

            case 'Ejes cuerpo (Quaterniones)'

                if app.manoeuvreSelQ == 1
                    app.u0 = app.V0*cos(app.alpha0)*cos(app.
beta0);
                    app.v0 = app.V0*sin(app.beta0);
                    app.w0 = app.V0*sin(app.alpha0)*cos(app.
beta0);

                    app.theta0Q = app.alpha0;

                    % Quaternions initial values
                    app.q00 = cos(app.psi0Q/2)*cos(app.
theta0Q/2)*cos(app.phi0Q/2) + sin(app.psi0Q/2)*sin(app.theta0Q
/2)*sin(app.phi0Q/2);
                    app.q10 = cos(app.psi0Q/2)*cos(app.
theta0Q/2)*sin(app.phi0Q/2) - sin(app.psi0Q/2)*sin(app.theta0Q
/2)*cos(app.phi0Q/2);
                    app.q20 = cos(app.psi0Q/2)*sin(app.
theta0Q/2)*cos(app.phi0Q/2) + sin(app.psi0Q/2)*cos(app.theta0Q
/2)*sin(app.phi0Q/2);
                    app.q30 = sin(app.psi0Q/2)*cos(app.
theta0Q/2)*cos(app.phi0Q/2) - cos(app.psi0Q/2)*sin(app.theta0Q
/2)*sin(app.phi0Q/2);

                elseif app.manoeuvreSelQ == 2
                    app.u0 = app.V0*cos(app.alpha0)*cos(app.
beta0);
                    app.v0 = app.V0*sin(app.beta0);
                    app.w0 = app.V0*sin(app.alpha0)*cos(app.
beta0);

                    app.theta0Q = app.alpha0;

```

```
1446                        % Quaternions initial values
1447                        app.q00 = cos(app.psi0Q/2)*cos(app.
    theta0Q/2)*cos(app.phi0Q/2) + sin(app.psi0Q/2)*sin(app.theta0Q
    /2)*sin(app.phi0Q/2);
1448                        app.q10 = cos(app.psi0Q/2)*cos(app.
    theta0Q/2)*sin(app.phi0Q/2) - sin(app.psi0Q/2)*sin(app.theta0Q
    /2)*cos(app.phi0Q/2);
1449                        app.q20 = cos(app.psi0Q/2)*sin(app.
    theta0Q/2)*cos(app.phi0Q/2) + sin(app.psi0Q/2)*cos(app.theta0Q
    /2)*sin(app.phi0Q/2);
1450                        app.q30 = sin(app.psi0Q/2)*cos(app.
    theta0Q/2)*cos(app.phi0Q/2) - cos(app.psi0Q/2)*sin(app.theta0Q
    /2)*sin(app.phi0Q/2);
1451                    end
1452            end
1453
1454            uiwait(msgbox('Inicializaci n terminada.','Info.','
    help'));
1455        end
```

A.9. Simulación y representación

```
1468 % Button pushed function: StartButton
1469 function StartButtonPushed(app, event)
1470     switch app.axisSel
1471         case 'Ejes cuerpo ( ngulos  de Euler)'
1472
1473         % Vector of initial conditions
1474         F0 = [app.u0; app.v0; app.w0; app.p0B; app.q0B; app.r0B;
    app.phi0B; app.theta0B; app.psi0B; app.x0B; app.y0B; app.z0;
    app.fuelDot0];
1475
1476         i = 0;
1477         intervalo = 0.2;
1478         nsup = 1;
1479         ninf = 1;
1480         nsup2 = 1;
1481         ninf2 = 1;
1482
1483         if strcmpi(app.StartButton.Text,'Continue')
1484
```

```
1485            F0 = transpose(app.F(end,:));
1486            i = app.i_ini;
1487            nsup = app.nsup_ini;
1488            ninf = app.ninf_ini;
1489            nsup2 = app.nsup2_ini;
1490            ninf2 = app.ninf2_ini;
1491            app.StartButton.Text = 'Start';
1492            app.StopButton.Text = 'Stop';
1493
1494        end
1495
1496        app.x = 0;
1497        options = odeset('RelTol', 1e-10, 'AbsTol', 1e-10);
1498
1499        while app.x == 0
1500
1501            if app.x == 1
1502                break
1503            end
1504
1505            if F0(12) <= 0
1506                uiwait(msgbox('La aeronave ha alcanzado el suelo.
    ','Info.','warn'));
1507                break
1508            end
1509
1510            % Solver
1511            [app.t,app.F] = ode113(@(tspan,F) BRYANBODYAPP(app,
    tspan,F), [i i+intervalo], F0, options);
1512            psiDot = (app.F(:,5).*sin(app.F(:,7))+app.F(:,6).*cos
    (app.F(:,7))./cos(app.F(:,8));
1513
1514            F0 = transpose(app.F(end,:));
1515
1516            i = i+intervalo;
1517            app.i_ini = i;
1518
1519            app.F_total = [app.F_total; app.F];
1520            app.t_total = [app.t_total; app.t];
1521
```

```
1522            % Fuel consumption
1523            app.fuelconsumption = app.F(end,13);
1524
1525            %Load factors
1526            [~,~,~,rhoat] = atmosisa(app.F(:,12));
1527
1528            alpha = atan(app.F(:,3)./app.F(:,1));
1529            beta = asin(app.F(:,2)./(sqrt(app.F(:,1).^2+app.F
    (:,2).^2+app.F(:,3).^2)));
1530            alphaDot = 0;
1531            betaDot = 0;
1532
1533            [CL,CD,CY,~,~,~] = coefficientsAPP(app,sqrt(app.F
    (:,1).^2+app.F(:,2).^2+app.F(:,3).^2),alpha,alphaDot,app.F(:,4)
    ,app.F(:,5),app.F(:,6),beta,betaDot,app.dE0,app.dA0,app.dR0);
1534
1535            CYw = -CD.*sin(beta)-CY.*cos(beta);
1536
1537            app.nx = abs(0.5/(app.m*app.g).*rhoat.*app.Sw.*(app.F
    (:,1).^2+app.F(:,2).^2+app.F(:,3).^2).*(-CD.*cos(alpha).*cos(
    beta)+CL.*sin(alpha)+CYw.*cos(alpha).*sin(beta)));
1538            app.ny = abs(0.5/(app.m*app.g).*rhoat.*app.Sw.*(app.F
    (:,1).^2+app.F(:,2).^2+app.F(:,3).^2).*(-CD.*sin(beta)+CYw.*cos
    (beta)));
1539            app.nz = abs(0.5/(app.m*app.g).*rhoat.*app.Sw.*(app.F
    (:,1).^2+app.F(:,2).^2+app.F(:,3).^2).*(-CL.*cos(alpha)-CD.*cos
    (beta).*sin(alpha)+CYw.*sin(alpha).*sin(beta)));
1540
1541            switch app.Graphics
1542                case 'Velocidades lineales'
1543                    plot(app.UIAxes,app.t,app.F(:,1),'b',app.t,
    app.F(:,2),'r',app.t,app.F(:,3),'g')
1544                    app.UIAxes.Title.String = 'Time evolution of
    linear velocities';
1545                    app.UIAxes.XLabel.String = 't [s]';
1546                    app.UIAxes.YLabel.String = 'Linear velocities
     [m/s]';
1547                    legend(app.UIAxes, 'u(t)','v(t)','w(t)')
1548                    if app.t(end)<10
1549                        app.UIAxes.XLim = [0 10];
```

```
1550                    else
1551                        app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
1552                    end
1553                    grid(app.UIAxes,'on')
1554                    drawnow limitrate
1555
1556                case 'Velocidades angulares'
1557                    plot(app.UIAxes,app.t,app.F(:,4),'b',app.t,
    app.F(:,5),'r',app.t,app.F(:,6),'g')
1558                    app.UIAxes.Title.String = 'Time evolution of
    angular velocities';
1559                    app.UIAxes.XLabel.String = 't [s]';
1560                    app.UIAxes.YLabel.String = 'Angular
    velocities [rad/s]';
1561                    legend(app.UIAxes,'p(t)','q(t)','r(t)')
1562                    if app.t(end)<10
1563                        app.UIAxes.XLim = [0 10];
1564                    else
1565                        app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
1566                    end
1567                    grid(app.UIAxes,'on')
1568                    drawnow limitrate
1569
1570
1571                case ' ngulos  de Euler'
1572                    plot(app.UIAxes,app.t,app.F(:,7),'b',app.t,
    app.F(:,8),'r',app.t,app.F(:,9),'g')
1573                    app.UIAxes.Title.String = 'Time evolution of
    the Euler angles';
1574                    app.UIAxes.XLabel.String = 't [s]';
1575                    app.UIAxes.YLabel.String = 'Euler angles [rad
    ]';
1576                    legend(app.UIAxes,'\phi(t)','\theta(t)','\psi
    (t)')
1577                    if app.t(end)<10
1578                        app.UIAxes.XLim = [0 10];
1579                    else
1580                        app.UIAxes.XLim = [app.t(end)-10 app.t(
```

```
    end)];
                    end
                    grid(app.UIAxes,'on')
                    drawnow limitrate

                case 'Posici n'
                    plot(app.UIAxes,app.t,app.F(:,10),'b')
                    app.UIAxes.Title.String = 'Time evolution of
    position';
                    app.UIAxes.XLabel.String = 't [s]';
                    app.UIAxes.YLabel.String = 'x(t) [m]';
                    legend(app.UIAxes,'off')
                    if app.t(end)<10
                        app.UIAxes.XLim = [0 10];
                    else
                        app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
                    end
                    grid(app.UIAxes,'on')
                    drawnow limitrate

                case 'Posici n lateral'
                    plot(app.UIAxes,app.t,app.F(:,11),'b')
                    app.UIAxes.Title.String = 'Time evolution of
    lateral position';
                    app.UIAxes.XLabel.String = 't [s]';
                    app.UIAxes.YLabel.String = 'y(t) [m]';
                    legend(app.UIAxes,'off')
                    if app.t(end)<10
                        app.UIAxes.XLim = [0 10];
                    else
                        app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
                    end
                    if app.F(end,11)-app.y0B >= nsup*100
                        app.UIAxes.YLim = [app.y0B-100*ninf app.
    y0B+100*(nsup+1)];
                        nsup = nsup+1;
                        app.nsup_ini = nsup;
                    elseif app.F(end,11)-app.y0B <= -ninf*100
```

```
1615                        app.UIAxes.YLim = [app.y0B-100*(ninf+1)
    app.y0B+100*nsup];
1616                        ninf = ninf+1;
1617                        app.ninf_ini = ninf;
1618                    else
1619                        app.UIAxes.YLim = [app.y0B-100*ninf app.
    y0B+100*nsup];
1620                    end
1621                    grid(app.UIAxes,'on')
1622                    drawnow limitrate
1623
1624                case 'Altitud'
1625                    plot(app.UIAxes,app.t,app.F(:,12),'b')
1626                    app.UIAxes.Title.String = 'Time evolution of
    altitude';
1627                    app.UIAxes.XLabel.String = 't [s]';
1628                    app.UIAxes.YLabel.String = 'z(t) [m]';
1629                    legend(app.UIAxes,'off')
1630                    if app.t(end)<10
1631                        app.UIAxes.XLim = [0 10];
1632                    else
1633                        app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
1634                    end
1635                    if app.F(end,12)-app.z0 >= nsup*500
1636                        app.UIAxes.YLim = [app.z0-500*ninf app.z0
    +500*(nsup+1)];
1637                        nsup = nsup+1;
1638                        app.nsup_ini = nsup;
1639                    elseif app.F(end,12)-app.z0 <= -ninf*500
1640                        app.UIAxes.YLim = [app.z0-500*(ninf+1)
    app.z0+500*nsup];
1641                        ninf = ninf+1;
1642                        app.ninf_ini = ninf;
1643                    else
1644                        app.UIAxes.YLim = [app.z0-500*ninf app.z0
    +500*nsup];
1645                    end
1646                    grid(app.UIAxes,'on')
1647                    drawnow limitrate
```

```
1648
1649                case 'Trayectoria 2D'
1650                    plot(app.UIAxes,app.F(:,10),app.F(:,12),'b')
1651                    app.UIAxes.Title.String = 'Two-dimensional
    trajectory';
1652                    app.UIAxes.XLabel.String = 'x(t) [m]';
1653                    app.UIAxes.YLabel.String = 'z(t) [m]';
1654                    legend(app.UIAxes,'off')
1655                    if app.F(end,12)-app.z0 >= nsup*500
1656                        app.UIAxes.YLim = [app.z0-500*ninf app.z0
    +500*(nsup+1)];
1657                        nsup = nsup+1;
1658                        app.nsup_ini = nsup;
1659                    elseif app.F(end,12)-app.z0 <= -ninf*500
1660                        app.UIAxes.YLim = [app.z0-500*(ninf+1)
    app.z0+500*nsup];
1661                        ninf = ninf+1;
1662                        app.ninf_ini = ninf;
1663                    else
1664                        app.UIAxes.YLim = [app.z0-500*ninf app.z0
    +500*nsup];
1665                    end
1666                    grid(app.UIAxes,'on')
1667                    drawnow limitrate
1668
1669                case 'Trayectoria 3D'
1670                    view(app.UIAxes,[-3 -5 3])
1671                    plot3(app.UIAxes,app.F(:,10),app.F(:,11),app.
    F(:,12),'b')
1672                    app.UIAxes.Title.String = 'Three-dimensional
    trajectory';
1673                    app.UIAxes.XLabel.String = 'x(t) [m]';
1674                    app.UIAxes.YLabel.String = 'y(t) [m]';
1675                    app.UIAxes.ZLabel.String = 'z(t) [m]';
1676                    legend(app.UIAxes,'off')
1677                    if app.F(end,11)-app.y0B >= nsup*100
1678                        app.UIAxes.YLim = [app.y0B-100*ninf app.
    y0B+100*(nsup+1)];
1679                        nsup = nsup+1;
1680                        app.nsup_ini = nsup;
```

```
1681                    elseif app.F(end,11)-app.y0B <= -ninf*100
1682                        app.UIAxes.YLim = [app.y0B-100*(ninf+1)
    app.y0B+100*nsup];
1683                        ninf = ninf+1;
1684                        app.ninf_ini = ninf;
1685                    else
1686                        app.UIAxes.YLim = [app.y0B-100*ninf app.
    y0B+100*nsup];
1687                    end
1688                    if app.F(end,12)-app.z0 >= nsup2*500
1689                        app.UIAxes.ZLim = [app.z0-500*ninf2 app.
    z0+500*(nsup2+1)];
1690                        nsup2= nsup2+1;
1691                        app.nsup2_ini = nsup2;
1692                    elseif app.F(end,12)-app.z0 <= -ninf2*500
1693                        app.UIAxes.ZLim = [app.z0-500*(ninf2+1)
    app.z0+500*nsup2];
1694                        ninf2 = ninf2+1;
1695                        app.ninf2_ini = ninf2;
1696                    else
1697                        app.UIAxes.ZLim = [app.z0-500*ninf2 app.
    z0+500*nsup2];
1698                    end
1699                    grid(app.UIAxes,'on')
1700                    drawnow limitrate
1701
1702                case ' ngulos  aerodin micos'
1703                    plot(app.UIAxes,app.t,atan(app.F(:,3)./app.F
    (:,1)),'b',app.t,asin(app.F(:,2)./(sqrt(app.F(:,1).^2+app.F
    (:,2).^2+app.F(:,3).^2))),'r')
1704                    app.UIAxes.Title.String = 'Time evolution of
    the aerodynamic angles';
1705                    app.UIAxes.XLabel.String = 't [s]';
1706                    app.UIAxes.YLabel.String = 'Aerodynamic
    angles [rad]';
1707                    legend(app.UIAxes,'\alpha(t)','\beta(t)')
1708                    if app.t(end)<10
1709                        app.UIAxes.XLim = [0 10];
1710                    else
1711                        app.UIAxes.XLim = [app.t(end)-10 app.t(
```

```
    end)];
1712                    end
1713                    grid(app.UIAxes,'on')
1714                    drawnow limitrate
1715
1716                case 'Velocidad del viento'
1717                    plot(app.UIAxes,app.t,sqrt(app.F(:,1).^2+app.
    F(:,2).^2+app.F(:,3).^2),'b')
1718                    app.UIAxes.Title.String = 'Time evolution of
    the airspeed';
1719                    app.UIAxes.XLabel.String = 't [s]';
1720                    app.UIAxes.YLabel.String = 'V(t) [m/s]';
1721                    legend(app.UIAxes,'off')
1722                    if app.t(end)<10
1723                        app.UIAxes.XLim = [0 10];
1724                    else
1725                        app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
1726                    end
1727                    if sqrt(app.F(:,1).^2+app.F(:,2).^2+app.F
    (:,3).^2)-app.V0 >= nsup*20
1728                        app.UIAxes.YLim = [app.V0-20*ninf app.V0
    +20*(nsup+1)];
1729                        nsup = nsup+1;
1730                        app.nsup_ini = nsup;
1731                    elseif sqrt(app.F(:,1).^2+app.F(:,2).^2+app.F
    (:,3).^2)-app.V0 <= -ninf*20
1732                        app.UIAxes.YLim = [app.V0-20*(ninf+1) app
    .V0+20*nsup];
1733                        ninf = ninf+1;
1734                        app.ninf_ini = ninf;
1735                    else
1736                        app.UIAxes.YLim = [app.V0-20*ninf app.V0
    +20*nsup];
1737                    end
1738                    grid(app.UIAxes,'on')
1739                    drawnow limitrate
1740
1741                case 'Factores de carga'
1742                    plot(app.UIAxes,app.t,app.nx,'b',app.t,app.ny
```

```
,'r',app.t,app.nz,'g')
                    app.UIAxes.Title.String = 'Time evolution of
load factors';
                    app.UIAxes.XLabel.String = 't [s]';
                    app.UIAxes.YLabel.String = 'Load factors [-]'
;
                    legend(app.UIAxes, 'nx(t)','ny(t)','nz(t)')
                    if app.t(end)<10
                        app.UIAxes.XLim = [0 10];
                    else
                        app.UIAxes.XLim = [app.t(end)-10 app.t(
end)];
                    end
                    grid(app.UIAxes,'on')
                    drawnow limitrate
            end

        end

        case 'Ejes viento'

        % Vector of initial conditions
        F0 = [app.V0; app.beta0; app.alpha0; app.p0W; app.q0W;
app.r0W; app.mu0W; app.gamma0W; app.chi0W; app.x0W; app.y0W;
app.z0; app.fuelDot0];

        i = 0;
        intervalo = 0.2;
        nsup = 1;
        ninf = 1;
        nsup2 = 1;
        ninf2 = 1;

        if strcmpi(app.StartButton.Text,'Continue')

            F0 = transpose(app.F(end,:));
            i = app.i_ini;
            nsup = app.nsup_ini;
            ninf = app.ninf_ini;
            nsup2 = app.nsup2_ini;
```

```
1777            ninf2 = app.ninf2_ini;
1778            app.StartButton.Text = 'Start';
1779            app.StopButton.Text = 'Stop';
1780
1781        end
1782
1783        app.x = 0;
1784        options = odeset('RelTol', 1e-10, 'AbsTol', 1e-10);
1785
1786        while app.x == 0
1787
1788            if app.x == 1
1789                break
1790            end
1791
1792            if F0(12) <= 0
1793                uiwait(msgbox('La aeronave ha alcanzado el suelo.
    ','Info.','warn'));
1794                break
1795            end
1796
1797            % Solver
1798            [app.t,app.F] = ode113(@(tspan,F) BRYANWINDAPP(app,
    tspan,F), [i i+intervalo], F0, options);
1799
1800            F0 = transpose(app.F(end,:));
1801
1802            i = i+intervalo;
1803            app.i_ini = i;
1804
1805            app.F_total = [app.F_total; app.F];
1806            app.t_total = [app.t_total; app.t];
1807
1808            % Fuel consumption
1809            app.fuelconsumption = app.F(end,13);
1810
1811            %Load factors
1812            [~,~,~,rhoat] = atmosisa(app.F(:,12));
1813
1814            alphaDot = 0;
```

```
1815            betaDot = 0;
1816
1817            [CL,CD,CY,~,~,~] = coefficientsAPP(app,app.F(:,1),app
    .F(:,3),alphaDot,app.F(:,4),app.F(:,5),app.F(:,6),app.F(:,2),
    betaDot,app.dE0,app.dA0,app.dR0);
1818
1819            app.nx = abs(0.5/(app.m*app.g).*rhoat.*app.Sw.*app.F
    (:,1).^2.*(-CD.*cos(app.F(:,2))+CY.*sin(app.F(:,2))));
1820            app.ny = abs(0.5/(app.m*app.g).*rhoat.*app.Sw.*app.F
    (:,1).^2.*(-CD.*sin(app.F(:,2))-CY.*cos(app.F(:,2))));
1821            app.nz = abs(0.5/(app.m*app.g).*rhoat.*app.Sw.*app.F
    (:,1).^2.*(-CL));
1822
1823            switch app.Graphics
1824
1825                case 'Velocidades lineales'
1826                    plot(app.UIAxes,app.t,app.F(:,1),'b',app.t,0,
    'r',app.t,0,'g')
1827                    app.UIAxes.Title.String = 'Time evolution of
    linear velocities';
1828                    app.UIAxes.XLabel.String = 't [s]';
1829                    app.UIAxes.YLabel.String = 'Linear velocities
     [m/s]';
1830                    legend(app.UIAxes, 'u(t)','v(t)','w(t)')
1831                    if app.t(end)<10
1832                        app.UIAxes.XLim = [0 10];
1833                    else
1834                        app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
1835                    end
1836                    grid(app.UIAxes,'on')
1837                    drawnow limitrate
1838
1839                case 'Velocidades angulares'
1840                    plot(app.UIAxes,app.t,app.F(:,4),'b',app.t,
    app.F(:,5),'r',app.t,app.F(:,6),'g')
1841                    app.UIAxes.Title.String = 'Time evolution of
    angular velocities';
1842                    app.UIAxes.XLabel.String = 't [s]';
1843                    app.UIAxes.YLabel.String = 'Angular
```

```
velocities [rad/s]';
                    legend(app.UIAxes,'p(t)','q(t)','r(t)')
                    if app.t(end)<10
                        app.UIAxes.XLim = [0 10];
                    else
                        app.UIAxes.XLim = [app.t(end)-10 app.t(
end)];
                    end
                    grid(app.UIAxes,'on')
                    drawnow limitrate

                case ' ngulos  de Euler'
                    plot(app.UIAxes,app.t,app.F(:,7),'b',app.t,
app.F(:,8),'r',app.t,app.F(:,9),'g')
                    app.UIAxes.Title.String = 'Time evolution of
the Euler angles';
                    app.UIAxes.XLabel.String = 't [s]';
                    app.UIAxes.YLabel.String = 'Euler angles [rad
]';
                    legend(app.UIAxes,'\mu(t)','\gamma(t)','\chi'
)
                    if app.t(end)<10
                        app.UIAxes.XLim = [0 10];
                    else
                        app.UIAxes.XLim = [app.t(end)-10 app.t(
end)];
                    end
                    grid(app.UIAxes,'on')
                    drawnow limitrate

                case 'Posici n'
                    plot(app.UIAxes,app.t,app.F(:,10),'b')
                    app.UIAxes.Title.String = 'Time evolution of
position';
                    app.UIAxes.XLabel.String = 't [s]';
                    app.UIAxes.YLabel.String = 'x(t) [m]';
                    legend(app.UIAxes,'off')
                    if app.t(end)<10
                        app.UIAxes.XLim = [0 10];
                    else
```

```
1876                        app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
1877                    end
1878                    grid(app.UIAxes,'on')
1879                    drawnow limitrate
1880
1881                case 'Posici n lateral'
1882                    plot(app.UIAxes,app.t,app.F(:,11),'b')
1883                    app.UIAxes.Title.String = 'Time evolution of
    lateral position';
1884                    app.UIAxes.XLabel.String = 't [s]';
1885                    app.UIAxes.YLabel.String = 'y(t) [m]';
1886                    legend(app.UIAxes,'off')
1887                    if app.t(end)<10
1888                        app.UIAxes.XLim = [0 10];
1889                    else
1890                        app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
1891                    end
1892                    if app.F(end,11)-app.y0B >= nsup*100
1893                        app.UIAxes.YLim = [app.y0B-100*ninf app.
    y0B+100*(nsup+1)];
1894                        nsup = nsup+1;
1895                    elseif app.F(end,11)-app.y0B <= -ninf*100
1896                        app.UIAxes.YLim = [app.y0B-100*(ninf+1)
    app.y0B+100*nsup];
1897                        ninf = ninf+1;
1898                    else
1899                        app.UIAxes.YLim = [app.y0B-100*ninf app.
    y0B+100*nsup];
1900                    end
1901                    grid(app.UIAxes,'on')
1902                    drawnow limitrate
1903
1904                case 'Altitud'
1905                    plot(app.UIAxes,app.t,app.F(:,12),'b')
1906                    app.UIAxes.Title.String = 'Time evolution of
    altitude';
1907                    app.UIAxes.XLabel.String = 't [s]';
1908                    app.UIAxes.YLabel.String = 'z(t) [m]';
```

```
1909                    legend(app.UIAxes,'off')
1910                    if app.t(end)<10
1911                        app.UIAxes.XLim = [0 10];
1912                    else
1913                        app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
1914                    end
1915                    if app.F(end,12)-app.z0 >= nsup*500
1916                        app.UIAxes.YLim = [app.z0-500*ninf app.z0
    +500*(nsup+1)];
1917                        nsup = nsup+1;
1918                        app.nsup_ini = nsup;
1919                    elseif app.F(end,12)-app.z0 <= -ninf*500
1920                        app.UIAxes.YLim = [app.z0-500*(ninf+1)
    app.z0+500*nsup];
1921                        ninf = ninf+1;
1922                        app.ninf_ini = ninf;
1923                    else
1924                        app.UIAxes.YLim = [app.z0-500*ninf app.z0
    +500*nsup];
1925                    end
1926                    grid(app.UIAxes,'on')
1927                    drawnow limitrate
1928
1929                case 'Trayectoria 2D'
1930                    plot(app.UIAxes,app.F(:,10),app.F(:,12),'b')
1931                    app.UIAxes.Title.String = 'Two-dimensional
    trajectory';
1932                    app.UIAxes.XLabel.String = 'x(t) [m]';
1933                    app.UIAxes.YLabel.String = 'z(t) [m]';
1934                    legend(app.UIAxes,'off')
1935                    if app.F(end,12)-app.z0 >= nsup*500
1936                        app.UIAxes.YLim = [app.z0-500*ninf app.z0
    +500*(nsup+1)];
1937                        nsup = nsup+1;
1938                    elseif app.F(end,12)-app.z0 <= -ninf*500
1939                        app.UIAxes.YLim = [app.z0-500*(ninf+1)
    app.z0+500*nsup];
1940                        ninf = ninf+1;
1941                    else
```

```
1942                        app.UIAxes.YLim = [app.z0-500*ninf app.z0
    +500*nsup];
1943                    end
1944                    grid(app.UIAxes,'on')
1945                    drawnow limitrate
1946
1947                case 'Trayectoria 3D'
1948                    view(app.UIAxes,[-3 -5 3])
1949                    plot3(app.UIAxes,app.F(:,10),app.F(:,11),app.
    F(:,12),'b')
1950                    app.UIAxes.Title.String = 'Three-dimensional
    trajectory';
1951                    app.UIAxes.XLabel.String = 'x(t) [m]';
1952                    app.UIAxes.YLabel.String = 'y(t) [m]';
1953                    app.UIAxes.ZLabel.String = 'z(t) [m]';
1954                    legend(app.UIAxes,'off')
1955                    if app.F(end,11)-app.y0B >= nsup*100
1956                        app.UIAxes.YLim = [app.y0B-100*ninf app.
    y0B+100*(nsup+1)];
1957                        nsup = nsup+1;
1958                    elseif app.F(end,11)-app.y0B <= -ninf*100
1959                        app.UIAxes.YLim = [app.y0B-100*(ninf+1)
    app.y0B+100*nsup];
1960                        ninf = ninf+1;
1961                    else
1962                        app.UIAxes.YLim = [app.y0B-100*ninf app.
    y0B+100*nsup];
1963                    end
1964                    if app.F(end,12)-app.z0 >= nsup2*500
1965                        app.UIAxes.ZLim = [app.z0-500*ninf2 app.
    z0+500*(nsup2+1)];
1966                        nsup2= nsup2+1;
1967                    elseif app.F(end,12)-app.z0 <= -ninf2*500
1968                        app.UIAxes.ZLim = [app.z0-500*(ninf2+1)
    app.z0+500*nsup2];
1969                        ninf2 = ninf2+1;
1970                    else
1971                        app.UIAxes.ZLim = [app.z0-500*ninf2 app.
    z0+500*nsup2];
1972                    end
```

```
1973                    grid(app.UIAxes,'on')
1974                    drawnow limitrate
1975
1976                case ' ngulos  aerodin micos'
1977                    plot(app.UIAxes,app.t,app.F(:,2),'b',app.t,
    app.F(:,3),'r');
1978                    app.UIAxes.Title.String = 'Time evolution of
    the aerodynamic angles';
1979                    app.UIAxes.XLabel.String = 't [s]';
1980                    app.UIAxes.YLabel.String = 'Aerodynamic
    angles [rad]';
1981                    legend(app.UIAxes,'\beta(t)','\alpha(t)')
1982                    if app.t(end)<10
1983                        app.UIAxes.XLim = [0 10];
1984                    else
1985                        app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
1986                    end
1987                    grid(app.UIAxes,'on')
1988                    drawnow limitrate
1989
1990                case 'Velocidad del viento'
1991                    plot(app.UIAxes,app.t,sqrt(app.F(:,1).^2+app.
    F(:,2).^2+app.F(:,3).^2),'b')
1992                    app.UIAxes.Title.String = 'Time evolution of
    the airspeed';
1993                    app.UIAxes.XLabel.String = 't [s]';
1994                    app.UIAxes.YLabel.String = 'V(t) [m/s]';
1995                    legend(app.UIAxes,'off')
1996                    if app.t(end)<10
1997                        app.UIAxes.XLim = [0 10];
1998                    else
1999                        app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
2000                    end
2001                    if sqrt(app.F(:,1).^2+app.F(:,2).^2+app.F
    (:,3).^2)-app.V0 >= nsup*20
2002                        app.UIAxes.YLim = [app.V0-20*ninf app.V0
    +20*(nsup+1)];
2003                        nsup = nsup+1;
```

```
                        app.nsup_ini = nsup;
                    elseif sqrt(app.F(:,1).^2+app.F(:,2).^2+app.F
(:,3).^2)-app.V0 <= -ninf*20
                        app.UIAxes.YLim = [app.V0-20*(ninf+1) app
.V0+20*nsup];
                        ninf = ninf+1;
                        app.ninf_ini = ninf;
                    else
                        app.UIAxes.YLim = [app.V0-20*ninf app.V0
+20*nsup];
                    end
                    grid(app.UIAxes,'on')
                    drawnow limitrate

                case 'Factores de carga'
                    plot(app.UIAxes,app.t,app.nx,'b',app.t,app.ny
,'r',app.t,app.nz,'g')
                    app.UIAxes.Title.String = 'Time evolution of
load factors';
                    app.UIAxes.XLabel.String = 't [s]';
                    app.UIAxes.YLabel.String = 'Load factors [-]'
;
                    legend(app.UIAxes, 'nx(t)','ny(t)','nz(t)')
                    if app.t(end)<10
                        app.UIAxes.XLim = [0 10];
                    else
                        app.UIAxes.XLim = [app.t(end)-10 app.t(
end)];
                    end
                    grid(app.UIAxes,'on')
                    drawnow limitrate
            end

        end

        case 'Ejes cuerpo (Quaterniones)'

        % Vector of initial conditions
        F0 = [app.u0; app.v0; app.w0; app.p0Q; app.q0Q; app.r0Q;
app.q00; app.q10; app.q20; app.q30; app.x0Q; app.y0Q; app.z0;
```

```
    app.fuelDot0];
2036
2037        i = 0;
2038        intervalo = 0.2;
2039        nsup = 1;
2040        ninf = 1;
2041        nsup2 = 1;
2042        ninf2 = 1;
2043
2044        if strcmpi(app.StartButton.Text,'Continue')
2045
2046            F0 = transpose(app.F(end,:));
2047            i = app.i_ini;
2048            nsup = app.nsup_ini;
2049            ninf = app.ninf_ini;
2050            nsup2 = app.nsup2_ini;
2051            ninf2 = app.ninf2_ini;
2052            app.StartButton.Text = 'Start';
2053            app.StopButton.Text = 'Stop';
2054
2055        end
2056
2057        app.x = 0;
2058        options = odeset('RelTol', 1e-10, 'AbsTol', 1e-10);
2059
2060        while app.x == 0
2061
2062            if app.x == 1
2063                break
2064            end
2065
2066            if F0(13) <= 0
2067                uiwait(msgbox('La aeronave ha alcanzado el suelo.
    ','Info.','warn'));
2068                break
2069            end
2070
2071            % Solver
2072            [app.t,app.F] = ode113(@(tspan,F)
    BRYANBODYQUATERNIONSAPP(app,tspan,F), [i i+intervalo], F0,
```

```
    options);
2073
2074            F0 = transpose(app.F(end,:));
2075
2076            i = i+intervalo;
2077            app.i_ini = i;
2078
2079            app.F_total = [app.F_total; app.F];
2080            app.t_total = [app.t_total; app.t];
2081
2082            % Fuel consumption
2083            app.fuelconsumption = app.F(end,14);
2084
2085            %Load factors
2086            [~,~,~,rhoat] = atmosisa(app.F(:,13));
2087
2088            alpha = atan(app.F(:,3)./app.F(:,1));
2089            beta = asin(app.F(:,2)./(sqrt(app.F(:,1).^2+app.F
    (:,2).^2+app.F(:,3).^2)));
2090            alphaDot = 0;
2091            betaDot = 0;
2092
2093            [CL,CD,CY,~,~,~] = coefficientsAPP(app,sqrt(app.F
    (:,1).^2+app.F(:,2).^2+app.F(:,3).^2),alpha,alphaDot,app.F(:,4)
    ,app.F(:,5),app.F(:,6),beta,betaDot,app.dE0,app.dA0,app.dR0);
2094
2095            CYw = -CD.*sin(beta)-CY.*cos(beta);
2096
2097            app.nx = abs(0.5/(app.m*app.g).*rhoat.*app.Sw.*(app.F
    (:,1).^2+app.F(:,2).^2+app.F(:,3).^2).*(-CD.*cos(alpha).*cos(
    beta)+CL.*sin(alpha)+CYw.*cos(alpha).*sin(beta)));
2098            app.ny = abs(0.5/(app.m*app.g).*rhoat.*app.Sw.*(app.F
    (:,1).^2+app.F(:,2).^2+app.F(:,3).^2).*(-CD.*sin(beta)+CYw.*cos
    (beta)));
2099            app.nz = abs(0.5/(app.m*app.g).*rhoat.*app.Sw.*(app.F
    (:,1).^2+app.F(:,2).^2+app.F(:,3).^2).*(-CL.*cos(alpha)-CD.*cos
    (beta).*sin(alpha)+CYw.*sin(alpha).*sin(beta)));
2100
2101            switch app.Graphics
2102
```

```
2103                case 'Velocidades lineales'
2104                    plot(app.UIAxes,app.t,app.F(:,1),'b',app.t,
    app.F(:,2),'r',app.t,app.F(:,3),'g')
2105                    app.UIAxes.Title.String = 'Time evolution of
    linear velocities';
2106                    app.UIAxes.XLabel.String = 't [s]';
2107                    app.UIAxes.YLabel.String = 'Linear velocities
     [m/s]';
2108                    legend(app.UIAxes, 'u(t)','v(t)','w(t)')
2109                    if app.t(end)<10
2110                        app.UIAxes.XLim = [0 10];
2111                    else
2112                        app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
2113                    end
2114                    grid(app.UIAxes,'on')
2115                    drawnow limitrate
2116
2117                case 'Velocidades angulares'
2118                    plot(app.UIAxes,app.t,app.F(:,4),'b',app.t,
    app.F(:,5),'r',app.t,app.F(:,6),'g')
2119                    app.UIAxes.Title.String = 'Time evolution of
    angular velocities';
2120                    app.UIAxes.XLabel.String = 't [s]';
2121                    app.UIAxes.YLabel.String = 'Angular
    velocities [rad/s]';
2122                    legend(app.UIAxes,'p(t)','q(t)','r(t)')
2123                    if app.t(end)<10
2124                        app.UIAxes.XLim = [0 10];
2125                    else
2126                        app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
2127                    end
2128                    grid(app.UIAxes,'on')
2129                    drawnow limitrate
2130
2131                case 'Quaterniones'
2132                    plot(app.UIAxes,app.t,app.F(:,7),'b',app.t,
    app.F(:,8),'r',app.t,app.F(:,9),'g',app.t,app.F(:,10),'c',app.t
    ,sqrt(app.F(:,7).^2+app.F(:,8).^2+app.F(:,9).^2+app.F(:,10).^2)
```

```
,'m')
                    app.UIAxes.Title.String = 'Time evolution of
the quaternions';
                    app.UIAxes.XLabel.String = 't [s]';
                    app.UIAxes.YLabel.String = 'Euler angles [rad
]';
                    legend(app.UIAxes,'q0(t)','q1(t)','q2(t)','q3
(t)','vq(t)')
                    if app.t(end)<10
                        app.UIAxes.XLim = [0 10];
                    else
                        app.UIAxes.XLim = [app.t(end)-10 app.t(
end)];
                    end
                    app.UIAxes.YLim = [-1.1 1.1];
                    grid(app.UIAxes,'on')
                    drawnow limitrate

                case 'Posici n'
                    plot(app.UIAxes,app.t,app.F(:,11),'b')
                    app.UIAxes.Title.String = 'Time evolution of
position';
                    app.UIAxes.XLabel.String = 't [s]';
                    app.UIAxes.YLabel.String = 'x(t) [m]';
                    legend(app.UIAxes,'off')
                    if app.t(end)<10
                        app.UIAxes.XLim = [0 10];
                    else
                        app.UIAxes.XLim = [app.t(end)-10 app.t(
end)];
                    end
                    grid(app.UIAxes,'on')
                    drawnow limitrate

                case 'Posici n lateral'
                    plot(app.UIAxes,app.t,app.F(:,12),'b')
                    app.UIAxes.Title.String = 'Time evolution of
lateral position';
                    app.UIAxes.XLabel.String = 't [s]';
                    app.UIAxes.YLabel.String = 'y(t) [m]';
```

```
2165                     legend(app.UIAxes,'off')
2166                     if app.t(end)<10
2167                         app.UIAxes.XLim = [0 10];
2168                     else
2169                         app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
2170                     end
2171                     if app.F(end,12)-app.y0B >= nsup*100
2172                         app.UIAxes.YLim = [app.y0B-100*ninf app.
    y0B+100*(nsup+1)];
2173                         nsup = nsup+1;
2174                     elseif app.F(end,12)-app.y0B <= -ninf*100
2175                         app.UIAxes.YLim = [app.y0B-100*(ninf+1)
    app.y0B+100*nsup];
2176                         ninf = ninf+1;
2177                     else
2178                         app.UIAxes.YLim = [app.y0B-100*ninf app.
    y0B+100*nsup];
2179                     end
2180                     grid(app.UIAxes,'on')
2181                     drawnow limitrate
2182 
2183                 case 'Altitud'
2184                     plot(app.UIAxes,app.t,app.F(:,13),'b')
2185                     app.UIAxes.Title.String = 'Time evolution of
    altitude';
2186                     app.UIAxes.XLabel.String = 't [s]';
2187                     app.UIAxes.YLabel.String = 'z(t) [m]';
2188                     legend(app.UIAxes,'off')
2189                     if app.t(end)<10
2190                         app.UIAxes.XLim = [0 10];
2191                     else
2192                         app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
2193                     end
2194                     if app.F(end,13)-app.z0 >= nsup*500
2195                         app.UIAxes.YLim = [app.z0-500*ninf app.z0
    +500*(nsup+1)];
2196                         nsup = nsup+1;
2197                     elseif app.F(end,13)-app.z0 <= -ninf*500
```

```
2198                        app.UIAxes.YLim = [app.z0-500*(ninf+1)
    app.z0+500*nsup];
2199                        ninf = ninf+1;
2200                    else
2201                        app.UIAxes.YLim = [app.z0-500*ninf app.z0
    +500*nsup];
2202                    end
2203                    grid(app.UIAxes,'on')
2204                    drawnow limitrate
2205
2206                case 'Trayectoria 2D'
2207                    plot(app.UIAxes,app.F(:,11),app.F(:,13),'b')
2208                    app.UIAxes.Title.String = 'Two-dimensional
    trajectory';
2209                    app.UIAxes.XLabel.String = 'x(t) [m]';
2210                    app.UIAxes.YLabel.String = 'z(t) [m]';
2211                    legend(app.UIAxes,'off')
2212                    if app.F(end,13)-app.z0 >= nsup*500
2213                        app.UIAxes.YLim = [app.z0-500*ninf app.z0
    +500*(nsup+1)];
2214                        nsup = nsup+1;
2215                    elseif app.F(end,13)-app.z0 <= -ninf*500
2216                        app.UIAxes.YLim = [app.z0-500*(ninf+1)
    app.z0+500*nsup];
2217                        ninf = ninf+1;
2218                    else
2219                        app.UIAxes.YLim = [app.z0-500*ninf app.z0
    +500*nsup];
2220                    end
2221                    grid(app.UIAxes,'on')
2222                    drawnow limitrate
2223
2224                case 'Trayectoria 3D'
2225                    view(app.UIAxes,[-3 -5 3])
2226                    plot3(app.UIAxes,app.F(:,11),app.F(:,12),app.
    F(:,13),'b')
2227                    app.UIAxes.Title.String = 'Three-dimensional
    trajectory';
2228                    app.UIAxes.XLabel.String = 'x(t) [m]';
2229                    app.UIAxes.YLabel.String = 'y(t) [m]';
```

```
2230                    app.UIAxes.ZLabel.String = 'z(t) [m]';
2231                    legend(app.UIAxes,'off')
2232                    if app.F(end,12)-app.y0B >= nsup*100
2233                        app.UIAxes.YLim = [app.y0B-100*ninf app.
    y0B+100*(nsup+1)];
2234                        nsup = nsup+1;
2235                    elseif app.F(end,12)-app.y0B <= -ninf*100
2236                        app.UIAxes.YLim = [app.y0B-100*(ninf+1)
    app.y0B+100*nsup];
2237                        ninf = ninf+1;
2238                    else
2239                        app.UIAxes.YLim = [app.y0B-100*ninf app.
    y0B+100*nsup];
2240                    end
2241                    if app.F(end,13)-app.z0 >= nsup2*500
2242                        app.UIAxes.ZLim = [app.z0-500*ninf2 app.
    z0+500*(nsup2+1)];
2243                        nsup2= nsup2+1;
2244                    elseif app.F(end,13)-app.z0 <= -ninf2*500
2245                        app.UIAxes.ZLim = [app.z0-500*(ninf2+1)
    app.z0+500*nsup2];
2246                        ninf2 = ninf2+1;
2247                    else
2248                        app.UIAxes.ZLim = [app.z0-500*ninf2 app.
    z0+500*nsup2];
2249                    end
2250                    grid(app.UIAxes,'on')
2251                    drawnow limitrate
2252
2253                case ' ngulos  aerodin micos'
2254                    plot(app.UIAxes,app.t,atan(app.F(:,3)./app.F
    (:,1)),'b',app.t,asin(app.F(:,2)./(sqrt(app.F(:,1).^2+app.F
    (:,2).^2+app.F(:,3).^2))),'r');
2255                    app.UIAxes.Title.String = 'Time evolution of
    the aerodynamic angles';
2256                    app.UIAxes.XLabel.String = 't [s]';
2257                    app.UIAxes.YLabel.String = 'Aerodynamic
    angles [rad]';
2258                    legend(app.UIAxes,'\alpha(t)','\beta(t)')
2259                    if app.t(end)<10
```

```
2260                         app.UIAxes.XLim = [0 10];
2261                     else
2262                         app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
2263                     end
2264                     grid(app.UIAxes,'on')
2265                     drawnow limitrate
2266 
2267                 case 'Velocidad del viento'
2268                     plot(app.UIAxes,app.t,sqrt(app.F(:,1).^2+app.
    F(:,2).^2+app.F(:,3).^2),'b')
2269                     app.UIAxes.Title.String = 'Time evolution of
    the airspeed';
2270                     app.UIAxes.XLabel.String = 't [s]';
2271                     app.UIAxes.YLabel.String = 'V(t) [m/s]';
2272                     legend(app.UIAxes,'off')
2273                     if app.t(end)<10
2274                         app.UIAxes.XLim = [0 10];
2275                     else
2276                         app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
2277                     end
2278                     if sqrt(app.F(:,1).^2+app.F(:,2).^2+app.F
    (:,3).^2)-app.V0 >= nsup*20
2279                         app.UIAxes.YLim = [app.V0-20*ninf app.V0
    +20*(nsup+1)];
2280                         nsup = nsup+1;
2281                     elseif sqrt(app.F(:,1).^2+app.F(:,2).^2+app.F
    (:,3).^2)-app.V0 <= -ninf*20
2282                         app.UIAxes.YLim = [app.V0-20*(ninf+1) app
    .V0+20*nsup];
2283                         ninf = ninf+1;
2284                     else
2285                         app.UIAxes.YLim = [app.V0-20*ninf app.V0
    +20*nsup];
2286                     end
2287                     grid(app.UIAxes,'on')
2288                     drawnow limitrate
2289 
2290                 case ' ngulos  de Euler'
```

```
2291                    plot(app.UIAxes,app.t,atan2((2*(app.F(:,7).*
    app.F(:,8)+app.F(:,9).*app.F(:,10))),(1-2*app.F(:,8).^2-2*app.F
    (:,9).^2)),'b',app.t,asin(2*(app.F(:,7).*app.F(:,9)-app.F(:,8)
    .*app.F(:,10))),'r',app.t,atan2((2.*(app.F(:,7).*app.F(:,10)+
    app.F(:,8).*app.F(:,9))),(1-2.*app.F(:,9).^2-2.*app.F(:,10).^2)
    ),'g');
2292                    app.UIAxes.Title.String = 'Time evolution of
    the Euler angles';
2293                    app.UIAxes.XLabel.String = 't [s]';
2294                    app.UIAxes.YLabel.String = 'Euler angles [rad
    ]';
2295                    legend(app.UIAxes,'\phi(t)','\theta(t)','\psi
    (t)')
2296                    if app.t(end)<10
2297                        app.UIAxes.XLim = [0 10];
2298                    else
2299                        app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
2300                    end
2301                    grid(app.UIAxes,'on')
2302                    drawnow limitrate
2303
2304                case 'Factores de carga'
2305                    plot(app.UIAxes,app.t,app.nx,'b',app.t,app.ny
    ,'r',app.t,app.nz,'g')
2306                    app.UIAxes.Title.String = 'Time evolution of
    load factors';
2307                    app.UIAxes.XLabel.String = 't [s]';
2308                    app.UIAxes.YLabel.String = 'Load factors [-]'
    ;
2309                    legend(app.UIAxes, 'nx(t)','ny(t)','nz(t)')
2310                    if app.t(end)<10
2311                        app.UIAxes.XLim = [0 10];
2312                    else
2313                        app.UIAxes.XLim = [app.t(end)-10 app.t(
    end)];
2314                    end
2315                    grid(app.UIAxes,'on')
2316                    drawnow limitrate
2317            end
```

```
        end
    end
end
```

```
% Button pushed function: StopButton
function StopButtonPushed(app, event)
    if strcmpi(app.StopButton.Text,'Restart')
        cla(app.UIAxes,'reset')
        app.F_total = [];
        app.t_total = [];
        assignin('base','F_sol',app.F_total)
        assignin('base','t_sol',app.t_total)
        app.UIAxes.NextPlot = 'add';
        app.StartButton.Text = 'Start';
        app.StopButton.Text = 'Stop';
        return
    end
    if app.x == 0
    app.x = 1;
    disp(['El consumo de combustible es de ', num2str(app.
   fuelconsumption)])
    assignin('base','F_sol',app.F_total)
    assignin('base','t_sol',app.t_total)
    app.StartButton.Text = 'Continue';
    app.StopButton.Text = 'Restart';
    end
end
```