UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Desarrollo de un simulador de vuelo SIL en el entorno Simulink para el autopiloto Veronte

Trabajo Fin de Máster

Máster Universitario en Ingeniería Aeronáutica

AUTOR/A: Robles Gómez, Juan Roberto

Tutor/a: Rodas Jordá, Ángel

Cotutor/a externo: LOPEZ BAEZA, EDUARDO

CURSO ACADÉMICO: 2022/2023

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

EMBENTION

Trabajo de Fin de Máster

Desarrollo de un simulador de vuelo SIL en el entorno Simulink para el Autopiloto Veronte

Autor
Robles Gómez, Juan Roberto

Tutor
Rodas Jordá, Ángel

Tutor externo
López Baeza, Eduardo

Universitat Politècnica de València

Escuela Técnica Superior de Ingeniería del Diseño
Máster Universitario en Ingeniería Aeronáutica

Curso 2022 - 2023

31 de julio de 2023

Resumen

El simulador de vuelo desarrollado en este trabajo, llamado Embention Flight Simulator (EFS), reproduce el comportamiento de diversos tipos de aeronaves empleando una serie de modelos matemáticos configurables. EFS es un bloque de Simulink que, además, simula las señales que generarían los sensores embarcados en la aeronave real. Esto le permite trabajar en conjunto con el bloque Veronte de Simulink creado por Embention, que emula el funcionamiento del autopiloto Veronte, permitiendo simulaciones SIL (*Software In the Loop*). De esta forma, EFS facilita la validación e integración del autopiloto en las aeronaves de los clientes.

Palabras clave: *Software in the loop* (SIL); Simulink; Simulador de vuelo; Mecánica de vuelo; 6DOF; Autopiloto; Embention; Veronte.

Abstract

The flight simulator developed in this project, called Embention Flight Simulator (EFS), reproduces the performance of several aircraft types using a set of configurable mathematical models. EFS is a Simulink block that can also simulate the signals sent by sensors on board the actual aircraft. This allows the simulator to work together with Veronte Simulink block, created by Embention. This block emulates the operation of Veronte Autopilot, allowing SIL (Software in the loop) simulations, so that EFS eases validation and integration process in the customer's aircraft.

Key words: Software in the loop (SIL); Simulink; Simulador de vuelo; Mecánica de vuelo; 6DOF; Autopiloto; Embention; Veronte.

Resum

El simulador de vol desenvolupat en aquest treball, anomenat Embention Flight Simulator (EFS), reprodueix el comportament de diversos tipus d'aeronaus emprant una sèrie de models matemàtics configurables. EFS és un bloc de Simulink que, a més, simula les senyals que generarien els sensors embarcats a l'aeronau real. Això li permet treballar conjuntament amb el bloc Veronte de Simulink creat per Embention, que emula el funcionament de l'autopilot Veronte, permetent simulacions SIL (*Software In the Loop*). D'aquesta manera, EFS facilita la validació i integració de l'autopilot en les aeronaus dels clients.

Paraules clau: *Software in the loop* (SIL); Simulink; Simulador de vol; Mecànica de vol; 6DOF; Autopilot; Embention; Veronte

Agradecimentos

Me gustaría agradecer a Embention la oportunidad de realizar este trabajo y, en concreto, al departamento de PDI. Os deseo lo mejor en lo personal, en lo profesional sé que lograréis lo que os propongáis.

A mi tutor Ángel Rodas, por todos los consejos que me ha dado durante su realización.

A mis abuelos.

A mis padres, por haberme hecho la persona que soy.

Y a mi hermana, por servirme de ejemplo.

Índice general

Índice de figuras

Índice de tablas

Parte I

MEMORIA

Capítulo 1

Motivación y objetivos

Embention Sistemas Inteligentes S.A. es una empresa española, con sede en Alicante, dedicada al desarrollo de autopilotos y componentes para vehículos autónomos [1], principalmente aeronaves. Su producto central, llamado Veronte, es un autopiloto personalizable capaz de adaptarse a diferentes tipos de aeronaves y con certificaciones DO178C/ED-12, DO254 y DO160 [1].

Embention, además de Veronte, ha desarrollado todo un ecosistema de periféricos y productos alrededor de su autopiloto. Uno de estos productos es su *Autopilot SIL Simulator*, un bloque de Simulink que se comporta como un autopiloto Veronte real [2], permitiendo realizar simulaciones y pruebas sin necesidad de tener ningún dispositivo físico conectado.

El objetivo de este bloque es realizar pruebas *Software in the loop* (SIL), un tipo de test para validar el funcionamiento del sistema dentro de un entorno que replica el mundo real mediante modelos matemáticos [3], en este caso, Simulink. Según Embention [2], su modelo de Veronte en Simulink permite realizar simulaciones, en tiempo real o más rápido que el tiempo real, test de errores y análisis de vuelo, con aeronaves de diversos tipos dada su facilidad de configuración.

El bloque Veronte requiere las señales de los sensores como *inputs* para funcionar, por lo que éstas también tienen que ser simuladas de alguna forma. De aquí nace la motivación de desarrollar un simulador de vuelo en Simulink que genere las señales de los sensores según el estado de la aeronave, también simulada, en el mismo entorno en el que se ejecuta Veronte. Por otro lado, Veronte devuelve como *output* las señales de control, que pueden ser introducidas de nuevo en el simulador, cerrando el lazo de control. En la Figura 1.1, se muestra esto de forma esquemática, añadiendo además un bloque de representación de resultados para comprobar su validez.

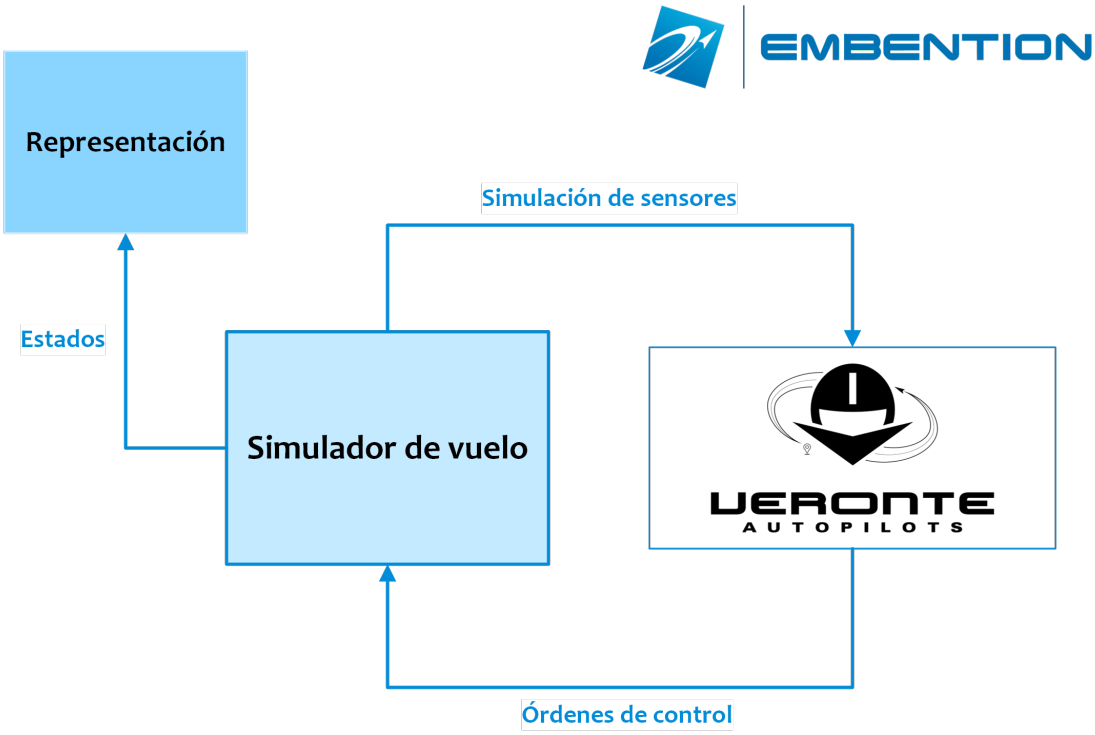

Figura 1.1: Diagrama general del proyecto

El autopiloto Veronte (Figura 1.2), como ya se ha comentado, puede adaptarse a diferentes aeronaves y vehículos. Por ello, es necesario que el simulador pueda ser configurable para replicar el comportamiento de diferentes tipos de aeronaves (multicópteros, ala fija, VTOL...).

Figura 1.2: Autopiloto Veronte real [1]

En definitiva, el objetivo del presente trabajo de fin de máster es desarrollar un simulador de vuelo en Simulink, que permita simular aeronaves diversas, configurable y adaptable y que genere las señales de los sensores embarcados en la aeronave, en el formato requerido por el bloque Veronte. De esta forma, se tiene una herramienta que posibilita realizar test SIL sin necesidad de modelar desde cero la aeronave requerida y sin disponer de los equipos reales.

Esto supone una ventaja considerable para una empresa como Embention, puesto que trabajan con clientes de todo el mundo y con aeronaves muy dispares. Contar con un simulador de estas características permite no necesitar la aeronave real del cliente, evitando envíos, en muchos casos internacionales. También puede ser vendido como herramienta complementaria al bloque Veronte SIL, que también está disponible para su compra, permitiendo a los clientes comprobar cómo Veronte controlaría su aeronave antes de ser construida, o en etapas de diseño.

Aunque todas las condiciones a tener en cuenta durante el desarrollo del simulador se reflejan en el Pliego de condiciones adjunto, las características más relevantes requeridas se muestran a continuación:

- El simulador debe ser un bloque de Simulink.
- La simulación debe ser en 6 grados de libertad (6DOF).
- El bloque debe contener los modelos matemáticos de la aeronave, modificables en cada servicio según el tipo de aeronave. Se debe poder modelar tanto aeronaves de ala fija como multicópteros.
- Se debe contar con un modelo de actuadores.
- Se deben simular las señales de los sensores que estarían equipados en la aeronave. El formato de estas señales debe ser compatible con el requerido por Veronte.
- Como entradas se debe tener, al menos, la posición de los actuadores.
- Como salidas se debe tener las señales de los sensores simulados y los estados de la aeronave necesarios para su representación.

Cabe destacar que el trabajo se trata de un proyecto interno de Embention, llamado *Embention Flight Simulator* (o EFS por sus siglas) y desarrollado en un contexto de prácticas en empresa.

Capítulo 2

Estado del arte en simulación de vuelo

Los simuladores de vuelo son sistemas que tratan de replicar el comportamiento de una aeronave y del entorno en el que vuelan. Esto se puede realizar con diversos objetivos, como el entrenamiento de pilotos, realización de pruebas o creación de videojuegos [4].

Los primeros avances en este ámbito se dieron debido al riesgo que suponía formar a los pilotos de aeronaves tripuladas. De hecho, el entrenamiento de pilotos era el objetivo del *Tonneau Antoinette*, considerado como el precursor de los simuladores de vuelo [4] y fabricado en 1910. Este dispositivo se movía de forma manual por instructores de vuelo para presentar el movimiento de alabeo y cabeceo de la aeronave al piloto entrenado, como se observa en la Figura 2.1. El alumno debía utilizar sus controles para alinear una barra de referencia con el horizonte, imitando el control de un avión real [5].

En la actualidad, el entrenamiento de pilotos sigue siendo una de las principales aplicaciones de los simuladores. Muchos cuentan con reproducciones físicas de la cabina y de los instrumentos, de forma que el alumno puede familiarizarse con los instrumentos y los procedimientos básicos. Los más avanzados reproducen todos los aspectos de la experiencia de vuelo (como sonidos y vibraciones), pueden ser certificados por agencias de seguridad aérea como la FAA (Federal Aviation Administration) o EASA (European Aviation Safety Agency) y pueden ser empleados en la expedición de licencias de piloto [4].

Otra aplicación de los simuladores de vuelo es la realización de pruebas de una aeronave, facilitando la detección de errores y eliminando riesgos [4]. Además, presenta la ventaja de no tener que disponer de la aeronave real, con todo lo que eso conlleva a nivel práctico y económico.

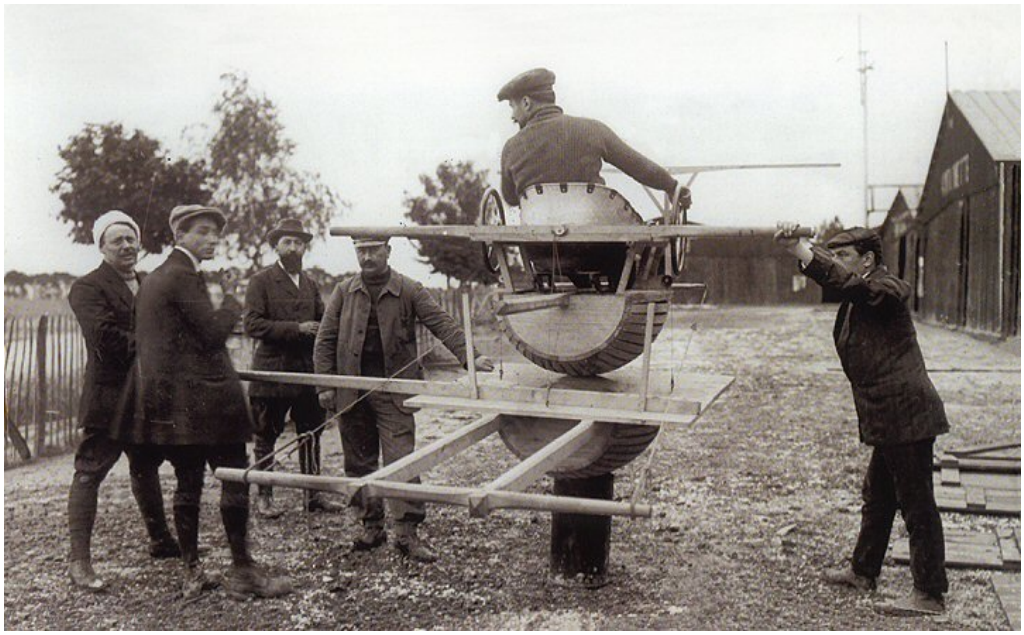

Figura 2.1: *Tonneau Antoinette* en funcionamiento (1910) [6]

Por último, otra aplicación de interés de los simuladores de vuelo es el entretenimiento. Diversos simuladores, como *X-Plane* o *Microsoft Flight Simulator*, tratan de recrear el vuelo y añaden a la simulación gráficos de la aeronave y del entorno con fines principalmente recreativos. En estos simuladores amateur existen comunidades con el objetivo de replicar el funcionamiento del sector aviación dentro del videojuego, creando incluso aerolíneas virtuales que imitan a las reales [7].

El simulador de vuelo desarrollado en este trabajo se encuadra en la segunda aplicación expuesta, ya que se trata de una herramienta para realizar pruebas de vuelo sin necesidad de disponer de la aeronave real.

2.1. Funcionamiento de algunos simuladores actuales

El principio de funcionamiento de un simulador de vuelo son los modelos matemáticos que describen el comportamiento de la aeronave simulada y de su entorno [5]. Principalmente, se requiere tener modelos para la aerodinámica, la propulsión, el contacto con el suelo y el entorno en el que se desplaza la aeronave para simular una misión completa.

Partiendo de los modelos se obtienen las fuerzas y momentos aplicados en la aeronave, a través de las cuales, usando las ecuaciones de movimiento, se calcula cómo se mueve la aeronave. Sobre estas ecuaciones de movimiento se profundiza en el Capítulo 3.

Los modelos y los métodos para obtenerlos son diversos según el nivel de precisión que se requiera, el tipo de aeronave y la potencia de cómputo. En el caso de los simuladores más avanzados para formación de pilotos, se tiene modelizada con mucha precisión la aeronave en concreto que se pretende aprender a pilotar, alcanzando una gran fidelidad. Los pilotos pueden aprender a volar en estos simuladores, en un entorno sin riesgo, y llegar mucho más preparados al vuelo real. Los más avanzados, capaces de recrear un vuelo completo y calificados por la FAA llegan a costar 10 millones de dólares [8].

Por otro lado, los simuladores amateur o los desarrollados para videojuegos, emplean simplificaciones que faciliten su implementación y ejecución en ordenadores domésticos y modelos flexibles que permitan recrear el vuelo de diversas aeronaves, aunque, como ya se ha comentado, también tratan de imitar lo máximo posible a la realidad.

Dadas las características de *Embention Flight Simulator*, es necesario emplear modelos flexibles y rápidos de obtener, ya que su finalidad principal es apoyar en la labor de integración y validación de las configuraciones del autopiloto Veronte, si bien es conveniente un comportamiento preciso para que el ajuste final del autopiloto en la aeronave real no requiera mucho tiempo adicional.

Ahora se van a describir los métodos usados por dos simuladores de vuelo, empleados tanto de forma recreativa como profesional, que cumplen algunos de los requerimientos que se piden de EFS: ambos son simuladores de vuelo 6DOF, permiten modelar nuevas aeronaves y han sido usados por Embention en ciertas ocasiones en conjunto con Veronte. Cabe destacar que el simulador desarrollado no necesita ningún tipo de gráficos ni de representación del vuelo de la aeronave, más allá de mostrar los resultados de forma clara. Por lo tanto, se van a exponer únicamente los métodos por los cuales estos simuladores obtienen las fuerzas y los momentos aplicados en el vehículo, según sus condiciones de vuelo.

2.1.1. Funcionamiento de Microsoft Flight Simulator

Figura 2.2: Logo de Microsoft Flight Simulator 2020, Microsoft [10]

Microsoft Flight Simulator (Figura 2.2) comenzó su desarrollo en 1977, a manos de Bruce Atwick y de su compañía Sublogic [9]. Su primera *release* se produjo en 1982, y la última hasta la fecha se ha dado en 2020. A partir de la documentación de su *Software Development Kit* (SDK), el manual en el cual explican cómo crear nuevos modelos de aeronave, se va a exponer su funcionamiento interno.

Según el manual del SDK del simulador, el problema de simular un vuelo se resume en obtener la configuración dinámica de una aeronave conociendo su configuración anterior (posición, orientación y velocidades lineales y angulares) y las acciones de control aplicadas a la aeronave (alerones, elevadores, aerofrenos, etc.) [11]. Microsoft Flight Simulator plantea este problema de forma matemática empleando un sistema de ecuaciones diferenciales ordinarias de primer orden, como describe la Ecuación 2.1.

$$\frac{\mathrm{d}\boldsymbol{X}_{\text{state}}}{\mathrm{d}t}(t) = H\left(\boldsymbol{X}_{\text{state}}(t), \boldsymbol{\delta}_{\text{controls}}(t)\right) \tag{2.1}$$

La incógnita de esta ecuación diferencial son los estados de la aeronave, que se exponen en la Ecuación 2.2. Por otro lado, $\boldsymbol{\delta}_{\text{controls}}$ recoge los controles de la aeronave y H representa la función vectorial que describe cómo el vehículo se comporta según las acciones de control aplicadas. Este comportamiento viene determinado por las ecuaciones fundamentales de la mecánica del sólido en 6 grados de libertad, de forma similar a las utilizadas en EFS y expuestas en el Capítulo 3.

$$\boldsymbol{X}_{\text{state}} = \left(\begin{array}{l} x_G \\ y_G \\ z_G \\ \theta_x \\ \theta_y \\ \theta_z \\ v_{x,G} \\ v_{y,G} \\ v_{z,G} \\ p \\ q \\ r \end{array}\right) \quad \begin{array}{l} \left.\begin{array}{l} \\ \\ \end{array}\right\} \text{Posición} \\ \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \text{Orientación o actitud} \\ \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \text{Velocidad translacional} \\ \left.\begin{array}{l} \\ \\ \\ \end{array}\right\} \text{Velocidad rotacional} \end{array} \tag{2.2}$$

Para obtener estos estados, Microsoft Flight Simulator realiza una integración numérica, calculando los estados de la aeronave a partir de los estados en el instante anterior,

empleando la regla del trapecio [11]. Estos cálculos requieren conocer las fuerzas y los momentos aplicados, que se estiman empleando coeficientes aerodinámicos y derivadas de estabilidad. En las versiones anteriores a la última (2020), se debían especificar estos datos y tablas de consulta. Sin embargo, la versión publicada en 2020 crea modelos discretizando la geometría de la aeronave en un total de 640 superficies [11], de forma que se minimizan los datos necesarios a los que se muestran en la Tabla 2.1. A partir de estos parámetros de partida y de la geometría se estiman el resto de los datos requeridos mediante su túnel de viento virtual [12] donde, finalmente, se calculan los coeficientes aerodinámicos de cada elemento de la geometría [11]. Estos coeficientes son los que se usan durante la simulación del vuelo.

Por otro lado, como el simulador pretende recrear todas las etapas del vuelo, también calcula las fuerzas de contacto con el suelo, así como el rozamiento debido a la rodadura del tren de aterrizaje, teniendo en cuenta el tipo de superficie en la que se tiene contacto y las condiciones ambientales. Además, en la documentación del SDK destacan la posibilidad de modelizar la entrada en pérdida, el efecto de la estela de los motores y de otras superficies o la condición de *deep stall* [11].

En la Figura 2.3 se muestra un diagrama de la documentación del SDK donde se resume el funcionamiento del simulador. En resumen, es un bucle de cálculo que, mediante integración numérica, obtiene los estados del instante actual a partir del instante anterior, considerando diferentes entradas (clima, controles, contacto con el suelo, etc.). Como se puede comprobar en la Sección 5.1, es un esquema similar al seguido durante el desarrollo de *Embention Flight Simulator*.

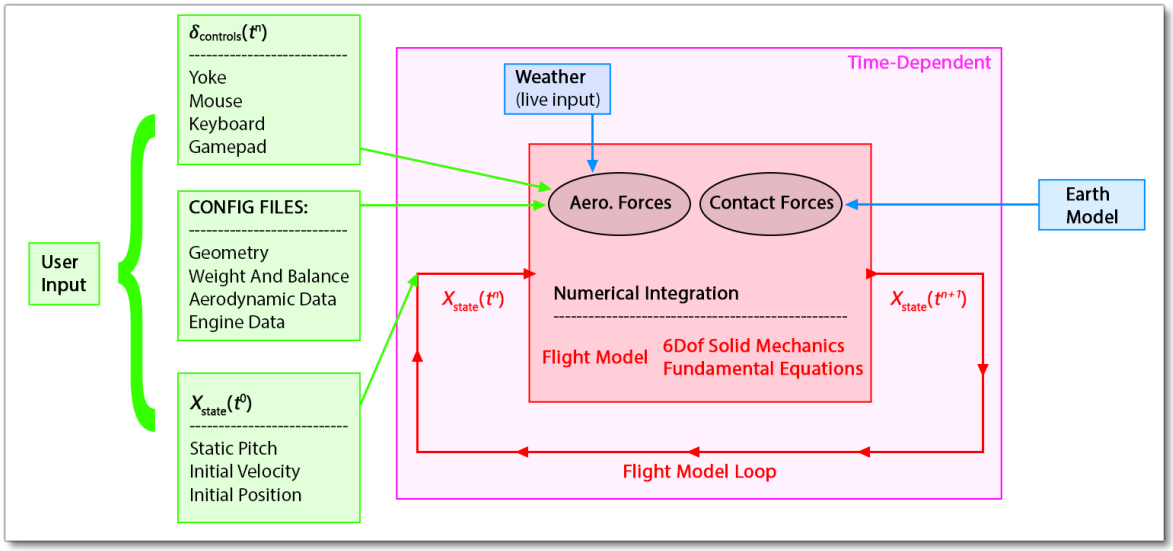

Figura 2.3: Esquema de funcionamiento de Microsoft Flight Simulator [11]

Tabla 2.1: Datos mínimos para crear un modelo en Microsoft Flight Simulator [13]

Ámbito	**Parámetro**	**Unidades**
Dimensiones	Superficie alar	[ft^2]
	Envergadura	[ft]
Peso	Máximo peso al despegue (MTOW)	[lbs]
	Peso en vacío	[lbs]
Entrada en pérdida	Velocidad de *stall*, configuración limpia	[kcas]
	Velocidad de *stall*, flaps al máximo	[kcas]
Motores	Potencia, empuje o par máximos	[hp][lbs][ftlbs]
Prestaciones (máx. potencia)	Velocidad máxima	[kcas]
	Máxima tasa de ascenso	[ft/min]
	Velocidad de crucero	[kcas]
	Máxima velocidad de ascenso	[kcas]
Prestaciones aerodinámicas	Máximo ratio de planeo	[-]
	Máxima velocidad de planeo	[kcas]

2.1.2. Funcionamiento de X-Plane

La primera *release* de X-Plane (Figura 2.4) fue publicada en 1995. Es un simulador creado por Austin Meyer y desarrollado por Laminar Research, con una gran flexibilidad a la hora de crear aeronaves, escenarios o funcionalidades [14]. Además, cuenta con versiones profesionales, certificadas por la FAA de forma que los pilotos pueden realizar horas de vuelo en el simulador, siempre y cuando se cumplan ciertas condiciones y se tenga el hardware requerido [14]. Todas estas características lo convierten en un simulador ampliamente utilizado tanto de forma recreativa como profesional.

Figura 2.4: Logo de *X-Plane 12*, Laminar Research [14]

Una de las razones por las que también es usado profesionalmente, es su capacidad de predecir el comportamiento de una aeronave cuando no se tienen datos que permitan modelar su vuelo [14]. Esto es útil en tareas de diseño, y lo aleja del modo de funcio-

namiento típico de otros simuladores de sus características. Su capacidad de predecir el funcionamiento de la aeronave se debe a cómo X-Plane obtiene las fuerzas y momentos aerodinámicos, necesitando para ello únicamente la geometría.

Según la página web oficial de X-Plane [15], el simulador emplea la teoría del elemento de pala para calcular la aerodinámica de la aeronave. Realmente, esta teoría está concebida para ser aplicada sobre hélices, pero en la descripción del funcionamiento del simulador emplean este término independientemente de la parte de la aeronave donde se aplique.

En primer lugar, antes de comenzar la simulación, X-Plane discretiza la geometría de la aeronave, es decir, la separa en elementos. El número de elementos empleados es variable y se indica por el usuario en la herramienta de creación de modelos de aeronave (*Plane-Maker*), siendo 10 elementos el máximo por lado en un ala o estabilizador.

Posteriormente, se requiere obtener la velocidad del aire percibida por el elemento. X-Plane realiza la estimación de la velocidad del elemento dos veces por ciclo. La velocidad del aire en el elemento depende de la velocidad angular y la distancia al punto de giro, de la velocidad lineal de la aeronave y de la velocidad inducida por otros elementos de la aeronave (*propwash*, *downwash* y ángulo de ataque inducido por superficies hipersustentadoras). La velocidad inducida por el propulsor, o *propwash*, se calcula complementando la teoría del elemento de pala aplicada en la hélice junto con la teoría de cantidad de movimiento. El uso conjunto de estas dos teorías es ampliamente utilizado a la hora de estimar el desempeño de una hélice. Por otro lado, el ángulo de ataque inducido por las superficies sustentadoras (*downwash*), se obtiene a través de tablas de consulta que lo relacionan con el coeficiente de sustentación.

Una vez conocidas las velocidades que afectan a cada elemento, se determinan los coeficientes aerodinámicos necesarios para el cálculo de las fuerzas. Las fuerzas que se aplican a cada elemento dependen de su superficie, obtenida durante la discretización y de la presión dinámica calculada a partir de la estimación previa de la velocidad. Los coeficientes se determinan a través de los datos de los perfiles aerodinámicos, especificados en la herramienta Part-Maker. Como estos perfiles son bidimensionales, X-Plane aplica diversas correcciones para ala finita como reducción de la sustentación y de la sustentación máxima, aplicación de la resistencia inducida y otros efectos debidos al alargamiento del ala, la flecha o el estrechamiento. También se consideran algunos efectos de compresibilidad del aire mediante el método de Prandtl-Glauert y modelos para vuelo supersónico.

Finalmente, considerando todas las fuerzas obtenidas en cada elemento, se tiene la fuerza total aplicada en toda la aeronave. De esta forma, con las ecuaciones del movimiento, se simula el movimiento de la aeronave en el espacio, repitiendo el proceso al menos 15 veces por segundo.

Aparte de describir el método de funcionamiento de X-Plane, en su página web se defiende dicho método frente a otros enfoques de simulación clásicos como el uso de coeficientes y derivadas de estabilidad. X-Plane calcula la aerodinámica según la geometría de la aeronave en cada instante de la simulación, por lo que, según exponen, es capaz de predecir cómo se comportaría una aeronave dada su forma, su peso y su propulsión, sin necesidad de estudios previos sobre el vuelo de dicho vehículo [15].

2.2. Simuladores similares a *Embention Flight Simulator*

Si bien X-Plane y Microsoft Flight Simulator pueden ser empleados junto con Veronte, son simuladores con capacidades superiores que las que se requieren de EFS. A continuación, se presentan algunos trabajos comparables a *Embention Flight Simulator*:

- *Quadcopter Dynamics and Simulation* [16]: Es un simulador de vuelo de cuadricópteros, desarrollado en Python, con una gran cantidad de simplificaciones. Utiliza modelos fáciles de implementar, algunos de los cuales se han incluido en EFS.

- *Aircraft 6-DOF Modular Modeling Based on MATLAB Simulink* [17]: En este artículo publicado en Atlantic Press, los investigadores desarrollan un simulador en 6 grados de libertad en Simulink. Asume ciertas simplificaciones que también se aplican en EFS, como el empleo de un sistema de referencia de Tierra plana o considerar la aeronave como un sólido rígido.

- *Modeling, Simulation, and Control of a Quadcopter* [18]: Este trabajo también realiza modelización y simulación de cuadricópteros en Simulink. Los autores crean un modelo de cuadricóptero con fallo en un motor, lo cual es una funcionalidad interesante a la hora de validar el comportamiento de Veronte en situaciones de emergencia.

Capítulo 3

Fundamentos teóricos

Como se ha podido apreciar en el Capítulo 2, existen diversos métodos para recrear el comportamiento de una aeronave. Sin embargo, en general, la simulación de vuelo se basa en obtener las fuerzas y momentos que se aplican, a partir de unas entradas de control y de la interacción con el entorno. Posteriormente, se emplean las ecuaciones del movimiento para calcular la posición y orientación del vehículo en el instante siguiente, partiendo de la posición y orientación anterior, mediante algún tipo de integración numérica.

Embention Flight Simulator sigue estos principios, por lo que en este capítulo se va a explicar cómo EFS maneja las ecuaciones del movimiento de la aeronave simulada. Cabe destacar que la obtención de fuerzas, dada la flexibilidad del simulador, depende de los modelos empleados. Por eso, la formulación matemática y teórica de las fuerzas y momentos aplicados se encuentra en la descripción del modelo en concreto (Sección 6.2).

3.1. Ecuaciones del movimiento

Se conoce como ecuaciones del movimiento a las fórmulas matemáticas que describen el movimiento en función del tiempo de un sistema físico [19], en este caso, una aeronave. Para desarrollar el simulador, es necesario plantear ecuaciones en el espacio, es decir, en 6 grados de libertad (6DOF) y teniendo en cuenta las fuerzas y momentos aplicados.

En Simulink y, en concreto, en el *Aerospace Blockset*, sobre los cuales se profundizará en el Capítulo 4, se pueden encontrar bloques que implementan estas ecuaciones en el espacio. EFS emplea el bloque *Custom Variable Mass 6DOF (Quaternion)* (Figura 3.1), que contiene las ecuaciones en 6 grados de libertad, con masa variable y representación por cuaterniones [20].

El planteamiento de estas ecuaciones parte de la Segunda Ley de Netwon (Ecuación 3.1),

donde se relaciona la fuerza y el momento aplicado con la cantidad de movimiento lineal ($\vec{p}$) y angular ($\vec{L}$).

$$\begin{aligned} \vec{F} &= \frac{d\vec{p}}{dt} \\ \vec{M} &= \frac{d\vec{L}}{dt} \end{aligned} \tag{3.1}$$

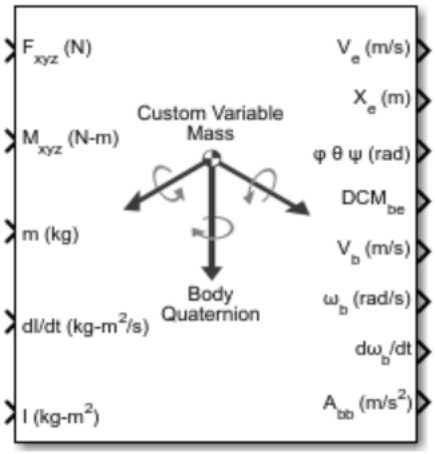

Figura 3.1: Bloque *Custom Variable Mass 6DOF (Quaternion)* [20]

3.1.1. Sistemas de referencia

Antes de continuar con el desarrollo de las ecuaciones del movimiento, es necesario conocer los sistemas de referencia con los que se va a trabajar. Esta versión de EFS emplea tres sistemas de referencia a lo largo de sus cálculos que se detallan a continuación.

Por un lado, el bloque *Custom Variable Mass 6DOF*, emplea dos sistemas de referencia. Considera el movimiento como la traslación y la rotación de un sistema fijado en el cuerpo de la aeronave respecto de un sistema fijo en el suelo, considerado inercial, tal y como se muestra en la Figura 3.2.

El sistema fijo en el cuerpo tiene su origen en el centro de gravedad de la aeronave. Su eje x apunta hacia el morro de la aeronave, el eje y hacia estribor y el z hacia abajo, de forma que es perpendicular al plano xy y sigue la regla de la mano derecha (Figura 3.3). Este sistema fijo en el cuerpo, además, sigue el criterio de Embention, expuesto en el Pliego de condiciones adjunto.

El segundo sistema, el sistema inercial, llamado sistema de Tierra plana puesto que no considera la curvatura de la Tierra, se trata de un sistema fijo en el suelo, a partir del cual

se conoce la posición de la aeronave [21]. Las transformaciones entre estos dos sistemas se realizan en el bloque de las ecuaciones de movimiento que, además, proporciona la matriz de rotación entre ambos sistemas, DCM_{be} (*Direction Cosine Matrix*).

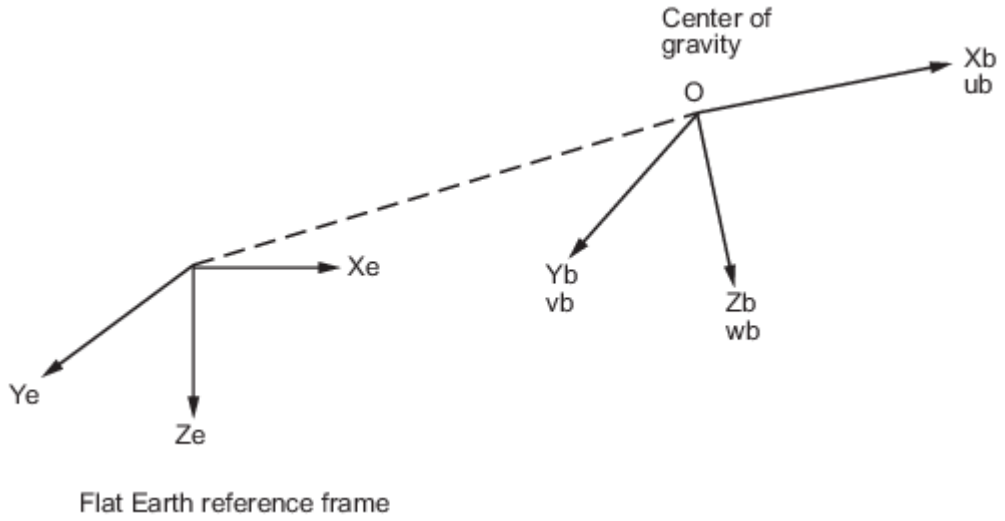

Figura 3.2: Sistemas empleados por las ecuaciones del movimiento 6DOF [21]

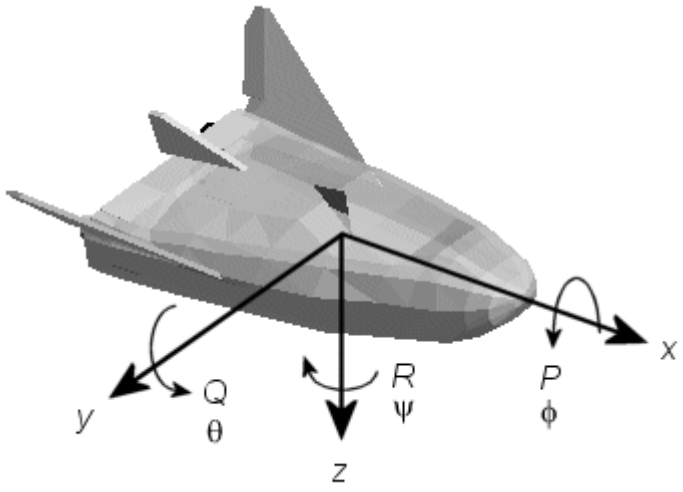

Figura 3.3: Sistema de referencia fijo en el cuerpo [22]

Por otro lado, es necesario para el funcionamiento del simulador conocer la posición sobre la Tierra, de forma que se necesita transformar el sistema de Tierra plana a uno LLA (Latitud, Longitud y Altura). Para ello, se especifica la posición LLA del origen del sistema Tierra plana y se transforma la posición de la aeronave, conocida en el sistema inercial.

El cambio a LLA se realiza en el bloque de *Aerospace blockset Flat Earth to LLA* [23]. Cabe destacar que este cambio presenta una serie de limitaciones que reduce su precisión cuando la aeronave se aleja considerablemente del punto de referencia o cuando la simulación se realiza cerca de los polos [23], limitaciones generalmente despreciables en los casos de uso del simulador propuestos por la empresa.

El bloque parte de la latitud y longitud de referencia (μ_0, l_0) y obtiene los radios de curvatura en la vertical (R_N) y en el meridiano (R_M) (Ecuación 3.2), a través del radio de la Tierra en el ecuador, R, y del achatamiento, f.

$$\begin{aligned} R_N &= \frac{R}{\sqrt{1-(2f-f^2)\sin^2\mu_0}} \\ R_M &= R_N \frac{1-(2f-f^2)}{1-(2f-f^2)\sin^2\mu_0} \end{aligned} \tag{3.2}$$

Con estos radios, se calcula el cambio en latitud y longitud, según el cambio de la posición (x, y) en el sistema Tierra plana. Si no se especifica lo contrario en el bloque, el eje x coincide con el norte y el y con el este, resultando en la Ecuación 3.3.

$$\begin{aligned} \Delta\mu &= \arctan\frac{1}{R_M}\Delta x \\ \Delta l &= \arctan\frac{1}{R_M\cos\mu}\Delta y \end{aligned} \tag{3.3}$$

Finalmente, como se indica en la Ecuación 3.4, la latitud y la longitud se obtiene a través de la latitud y longitud de referencia y del cambio previamente calculado, mientras que la altura es simplemente la altitud menos la elevación del terreno (h_{ref}), teniendo en cuenta que el eje z positivo apunta hacia el centro de la Tierra.

$$\begin{aligned} \mu &= \mu_0 + \Delta\mu \\ l &= l_0 + \Delta l \\ h &= -z - h_{ref} \end{aligned} \tag{3.4}$$

3.1.2. Representación por cuaterniones

Otro detalle importante del bloque *6DOF* es la representación por cuaterniones. Los cuaterniones son números hipercomplejos que siguen la forma $\mathbf{q} = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$, siendo $\mathbf{i}$,

j y **k** unidades imaginarias, cuyo cuadrado es igual a -1, y a, b, c y d números reales [24]. Los productos entre las unidades imaginarias de los cuaterniones se recogen en la tabla de multiplicación mostrada en la Figura 3.4. Estos cuaterniones son empleados en dinámica de vuelo y representan una mejor opción que las matrices de rotación a la hora de realizar rotaciones espaciales, ya que son más eficientes, compactos (necesitan 4 términos en comparación con los 9 requeridos por las matrices de rotación) y numéricamente estables [25]. Además, también presentan ventajas ante los ángulos de Euler para representar la actitud de una aeronave, puesto que evitan el problema del bloqueo del cardán [26].

	1	**i**	**j**	**k**
1	1	**i**	**j**	**k**
i	**i**	-1	**k**	$-$**j**
j	**j**	$-$**k**	-1	**i**
k	**k**	**j**	$-$**i**	-1

Figura 3.4: Tabla de multiplicación (columna izquierda × fila superior) [27]

Se llama cuaternión puro al cuaternión cuya parte real es cero ($\mathbf{p} = x\mathbf{i} + y\mathbf{j} + z\mathbf{k}$). Estos cuaterniones tienen la característica de poder representar un punto en $\mathbb{R}^3$, en este caso, $\mathbf{p}$ representa el punto (x, y, z) [24]. Por otro lado, según el teorema de rotación de Euler, cualquier rotación de un sistema de coordenadas entorno a un punto fijo se puede describir como una rotación de un ángulo θ sobre un eje fijo [25], siendo este el caso de la rotación del sistema de coordenadas fijo en el cuerpo. Los cuaterniones permiten representar estas rotaciones extendiendo la fórmula de Euler. Por ejemplo, en la Ecuación 3.5 se representa la rotación de un ángulo θ alrededor de un eje (a, b, c) mediante el cuaternión $\mathbf{q}$.

$$\mathbf{q} = e^{\frac{\theta}{2}(a\mathbf{i}+b\mathbf{j}+c\mathbf{k})} = \cos\frac{\theta}{2} + (a\mathbf{i} + b\mathbf{j} + c\mathbf{k})\sin\frac{\theta}{2} \tag{3.5}$$

Para aplicar esta rotación a un punto cualquiera, se construye un cuaternión puro ($\mathbf{p}$ en este caso) que represente dicho punto (x, y, z) y se aplica sobre él la Ecuación 3.6, siendo $\mathbf{p\prime}$ el punto rotado y $\bar{\mathbf{q}}$ el cuaternión conjugado de $\mathbf{q}$ [26]. El conjugado de un cuaternión $\mathbf{q} = a + b\mathbf{i} + c\mathbf{j} + d\mathbf{k}$ es $\bar{\mathbf{q}} = a - b\mathbf{i} - c\mathbf{j} - d\mathbf{k}$, además, cabe destacar que esta relación se cumple si el cuaternión que representa la rotación es unitario, es decir, $\|\mathbf{q}\| = 1$ [24].

$$\boldsymbol{p\prime} = \mathbf{q}\mathbf{p}\bar{\mathbf{q}} \tag{3.6}$$

Como ya se ha comentado, la representación por cuaterniones presenta ciertas ventajas.

Por ello, se ha optado por el uso del bloque 6DOF del *Aerospace blockset* que emplea este tipo de representación espacial. Los cuaterniones están internos en el bloque, que devuelve la actitud de la aeronave como ángulos de Euler, por lo que no es necesario que el usuario trate con los cuaterniones para emplear el simulador.

3.1.3. Implementación de las ecuaciones de movimiento

Una vez expuesto el sistema de representación por cuaterniones y los sistemas de referencia empleados, se van a explicar las ecuaciones del movimiento que implementa el bloque 6DOF de Simulink.

En concreto, el bloque empleado es *Custom Variable Mass 6DOF (Quaternion)*, ya que es necesario contemplar la variación de masa en algunos casos, como en aeronaves que consumen combustible o que liberan una parte de su estructura o una carga de pago. Este bloque emplea unas entradas para aplicar en ellas las ecuaciones del movimiento y devolver unas salidas. Dichas entradas y salidas se muestran en las tablas 3.1 y 3.2.

Tabla 3.1: Entradas al bloque *Custom Variable Mass 6DOF (Quaternion)* [20]

Entrada	**Descripción**
F_{xyz}	Vector fuerza aplicada, en el sistema fijo en el cuerpo
M_{xyz}	Vector momento aplicado, en el sistema fijo en el cuerpo
m	Masa de la aeronave
dI/dt o $\dot{I}$	Ratio de cambio de la matriz de inercia
I	Matriz de inercia

Tabla 3.2: Salidas del bloque *Custom Variable Mass 6DOF (Quaternion)* [20]

Salida	**Descripción**
V_e	Vector velocidad en el sistema de referencia Tierra plana
X_e	Vector posición en el sistema de referencia Tierra plana
$\phi\ \theta\ \psi$ (rad)	Ángulos de Euler
DCM_{be}	Matriz de rotación entre el sistema Tierra plana y el fijo en el cuerpo
V_b	Velocidad en el sistema fijo en el cuerpo
ω_b (rad/s)	Vector velocidad angular en el sistema fijo en el cuerpo
A_{bb}	Vector aceleración en el sistema fijo en el cuerpo

Con las entradas descritas, el bloque implementa la Ecuación 3.7 para la dinámica traslacional y la Ecuación 3.8 para la rotacional [21]. $\bar{V_b}$ representa la velocidad lineal, y $\bar{\omega}$ la velocidad angular en los ejes fijos al cuerpo.

$$\bar{F_b} = \begin{bmatrix} F_x \\ F_y \\ F_z \end{bmatrix} = m(\dot{\bar{V_b}} + \bar{\omega} \times \bar{V_b}) \tag{3.7}$$

$$\bar{M_b} = \begin{bmatrix} L \\ M \\ N \end{bmatrix} = I\dot{\bar{\omega}} + \bar{\omega} \times (I\bar{\omega}) + \dot{I}\bar{\omega} \tag{3.8}$$

Estas ecuaciones relacionan las fuerzas y momentos con las derivadas temporales de la posición y de la orientación, es decir, velocidades y aceleraciones. Como el objetivo es conocer la posición de la aeronave, se deben integrar estas ecuaciones.

3.2. Integración numérica

La integración numérica es un método para resolver ecuaciones diferenciales como, por ejemplo, las ecuaciones de movimiento. En este caso, es Simulink el que se encarga de resolver las ecuaciones diferenciales empleando un *solver*, es decir, un algoritmo de integración numérica. Simulink cuenta con diversos *solvers* para ajustarse al tipo de ecuaciones que debe resolver.

En la Figura 3.5 se muestra un diagrama de flujo para elegir el integrador según el tipo de problema. En el caso de *Embention Flight Simulator*, se ha escogido un *solver* de paso fijo, necesario para el correcto funcionamiento del bloque Veronte. Además, como se tienen estados continuos, se debe emplear un *Fixed-Step Continuous Solver*.

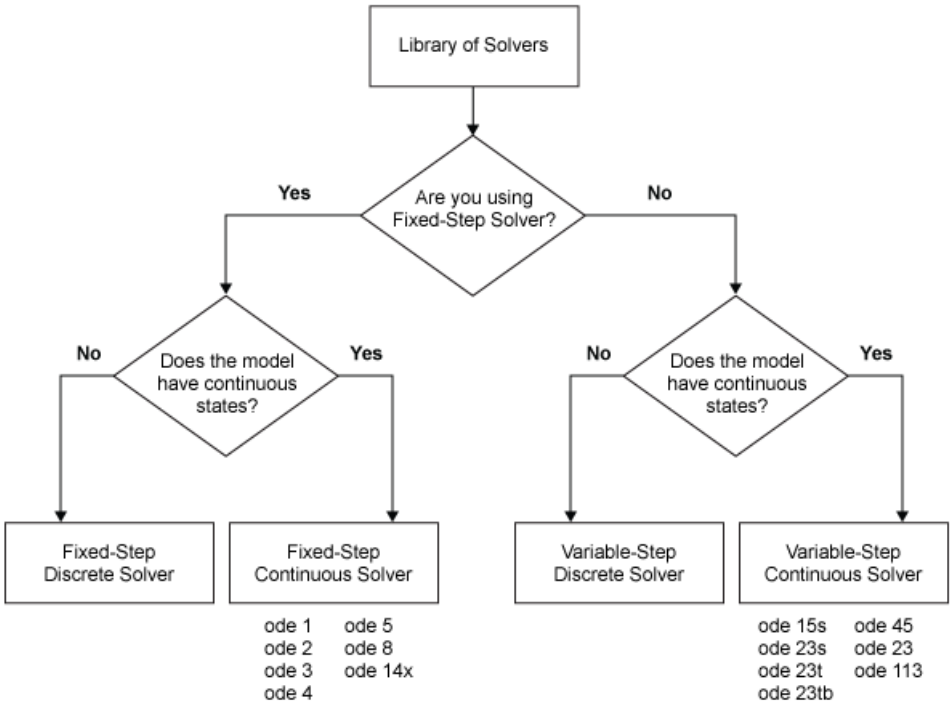

Figura 3.5: Diagrama de flujo para escoger *solver* [28]

Si se deja a Simulink escoger el *solver* óptimo, escoge ode3, es decir, un método numérico para ecuaciones diferenciales ordinarias (*Ordinary Differential Equations*) de orden 3 y que se trata de una implementación del método de Bogacki–Shampine (también empleado en ode23) [29].

El método de Bogacki–Shampine es un par de fórmulas de Runge–Kutta de orden 3 y 2, propuesto en 1989 como una alternativa a otros pares del mismo orden y con mejor fiabilidad, estabilidad y eficiencia [32]. El hecho de emplear dos fórmulas le permite estimar el error local de truncado, permitiendo ajustar el paso de integración y convirtiéndolo en un método adaptativo. Este tipo de *solvers* emplean dos métodos de orden p y $p-1$ (3 y 2 en este caso) empleando los mismos pasos intermedios para reducir el coste computacional [31]. En el caso de ode3, como ya se ha comentado, es un *solver* de paso fijo, por lo que a pesar de que Simulink lo denomine en su documentación como Bogacki–Shampine, solo requiere emplear el método de orden 3, llamado método de Ralston de tercer orden.

En general, un método Runge-Kutta explícito emplea la Ecuación 3.9 [30], donde s representa el número de etapas. Los coeficientes k_i de la ecuación dependen, a su vez, de otros coeficientes a_{ij} (con $1 \leq j < i \leq s$) y c_i (con $i = 2,3,...,s$), tal y como se expone en la Ecuación 3.10. La matriz a_{ij} se llama matriz de Runge–Kutta, los coeficientes b_i son conocidos como pesos y los c_i como nodos. De esta forma, para definir un método Runge–Kutta es necesario indicar la matriz, sus pesos y sus nodos [31].

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i \tag{3.9}$$

$$\begin{aligned}
k_1 &= f(t_n,\ y_n)\\
k_2 &= f(t_n + c_2\,h,\ y_n + (a_{21}\,k_1)h)\\
k_3 &= f(t_n + c_3\,h,\ y_n + (a_{31}\,k_1 + a_{32}\,k_2)h)\\
&\vdots\\
k_s &= f(t_n + c_s\,h,\ y_n + (a_{s1}\,k_1 + a_{s2}\,k_2 + \cdots + a_{s,s-1}\,k_{s-1})h)
\end{aligned} \tag{3.10}$$

La Tabla 3.3 es la llamada tabla de Butcher. Es este caso, se muestra la del método de Ralston de tercer orden. Esta tabla define el método recogiendo su matriz Runge–Kutta, sus pesos y nodos.

0			
$c_2 = 1/2$	$a_{21} = 1/2$		
$c_3 = 3/4$	$a_{31} = 0$	$a_{32} = 3/4$	
	$b_1 = 2/9$	$b_2 = 1/3$	$b_3 = 4/9$

Tabla 3.3: Tabla de Butcher del método Ralston de tercer orden [32]

Con la tabla de Butcher del método, lo único que se necesita para aplicar ode3 en la resolución de las ecuaciones diferenciales es el valor inicial y h, que representa el paso temporal. Ambos son parámetros requeridos por Simulink y configurables. En definitiva, especificando la posición y velocidad inicial de la aeronave (lineales y angulares), además del paso temporal de la simulación (tiempo de muestreo), el simulador puede resolver las ecuaciones del movimiento expuestas.

Capítulo 4

Software empleado

En este capítulo se va a exponer el diferente *software* empleado para la realización del simulador su funcionamiento y su validación.

4.1. Simulink

Embention Flight Simulator se ha desarrollado sobre Simulink. Se trata de un entorno de programación gráfico en el cual, empleando bloques, se pueden modelizar, simular y analizar sistemas dinámicos, desarrollado por MathWorks [33]. Esta basado en MATLAB, también de MathWorks y su última versión estable hasta le fecha de realización del presente documento es la versión 10.7 (2023a).

Simulink facilita el desarrollo del simulador dada su interfaz gráfica basada en bloques y sus herramientas enfocadas a la modelización y la simulación. Además, como ya se ha comentado en la Sección 3.2, Simulink se encarga de integrar numéricamente las ecuaciones de la dinámica del sistema. Esto permite centrarse en la construcción de los modelos de la aeronave.

Otro detalle a tener en cuenta es que el bloque del autopiloto Veronte para simulaciones SIL funciona en Simulink. De esta forma, el uso de este *software* como base para desarrollar el simulador se trata de un requerimiento de la empresa. En concreto, el desarrollo del simulador comenzó en la versión de MATLAB 2018a y se finalizó en la versión 2020a. El uso de estas versiones es igualmente un requerimiento de Embention.

4.1.1. Modelos y proyectos de Simulink

Teniendo en cuenta lo anterior, *Embention Flight Simulator* tiene formato de modelo de Simulink, es decir, es un archivo de extensión *.slx*.

En estos archivos, una vez abiertos por Simulink, se colocan y conectan los bloques que conforman el modelo. Los bloques se obtienen principalmente de la librería que ofrece Simulink, aunque para el desarrollo de EFS se han empleado bloques de otras librerías como el *Aerospace Blockset*, que se expondrá posteriormente.

Los modelos (archivos *.slx*), pueden funcionar como si fueran un bloque si se le indican las entradas y salidas que debe tener el modelo. Este es el caso de EFS, tal y como se muestra en la Figura 1.1, que tiene como entradas las órdenes de control y de salidas los estados de la aeronave simulada y la señal de los sensores simulados.

El modelo de EFS, además, está organizado mediante un *Simulink Project*. Estos proyectos permiten preparar el entorno de Matlab para la ejecución del modelo. Es decir, al ser abiertos, realizan las tareas de configuración previas, como cargar datos o añadir las carpetas del proyecto al *path* de Matlab, facilitando su uso y evitando errores.

Dada la complejidad del simulador, es interesante que el usuario no tenga que modificar el interior del bloque del simulador para hacerlo funcionar. Para conseguir esto, se han explotado algunas características de Simulink que se detallan a continuación.

4.1.2. Sistemas de variantes de Simulink

Los sistemas de variantes de Simulink permiten recoger diversos modelos en un solo bloque [34]. Antes de ejecutar una simulación se selecciona uno de los modelos, por lo que se evita tener que modificar el interior del bloque, haciendo el simulador flexible y configurable. Los modelos incluidos en este bloque de variantes se identifican mediante objetos *Simulink.Variant*, en los cuales se especifica la condición para la selección de cada modelo. El uso de variantes de Simulink consigue que el usuario no tenga que cambiar de modelo entre simulaciones, dotando a EFS de flexibilidad para simular desde aeronaves de ala fija hasta multicópteros.

4.1.3. Modelos *enabled*

Otra de las herramientas que ofrece Simulink y que son empleadas por *Embention Flight Simulator* son los modelos *enabled*. Cuando se realiza una simulación SIL del autopiloto en conjunto con el simulador, ambos bloques se ejecutan a la vez en un mismo *.slx*. En muchas ocasiones, esto produce efectos indeseados, puesto que la aeronave comienza a moverse desde el primer instante de simulación, impidiendo realizar comprobaciones del autopiloto en el *software* de Embention que se expondrá posteriormente.

Un modelo *enabled* es un modelo que se ejecuta de forma condicional, cuando una señal

determinada alcanza un valor mayor que cero [34]. Cuando esta señal no es mayor que cero, el modelo no se ejecuta y sus salidas devuelven unos valores iniciales determinados previamente. EFS es un bloque *enabled*, de forma que, aunque se ejecute la simulación, EFS no comenzará a calcular el movimiento de la aeronave hasta que el usuario lo indique, permitiendo realizar comprobaciones previas de la configuración del autopiloto.

4.1.4. Tablas de consulta o *Lookup tables*

Los bloques *Lookup Table* también han sido ampliamente empleados durante el desarrollo de *Embention Flight Simulator*. En ellos, el bloque emplea una tabla de consulta para generar el *output* dada una entrada determinada [34]. La tabla de consulta debe tener el valor de salida relacionado con un valor de entrada. En caso de que la entrada al bloque no se encuentre en la tabla almacenada, la salida se obtiene realizando según un algoritmo determinado de interpolación o extrapolación [34]. También es importante destacar que las tablas de consulta pueden ser de n dimensiones, es decir, la salida puede depender de más de una entrada.

La importancia de estos bloques *Lookup Table* en *Embention Flight Simulator* radica en la capacidad que ofrecen de replicar el comportamiento experimental de una aeronave. Si, por ejemplo, se ensaya la fuerza de un motor variando sus revoluciones por minuto y la densidad del aire, se pueden aplicar directamente los resultados obtenidos para cada valor de revoluciones y densidad al modelo. La fiabilidad de este tipo de modelización depende de la calidad del ensayo, pero puede alcanzar grandes precisiones dada la capacidad de modelizar fácilmente comportamientos no lineales.

4.1.5. *Aerospace Blockset*

Como se ha comentado, para el desarrollo de EFS no se ha empleado únicamente la librería de bloques de Simulink. *Aerospace Blockset* es una expansión de Simulink que incorpora bloques para la simulación y la modelización de vehículos aeroespaciales [35].

Tal y como se expone en el Capítulo 3, el bloque de ecuaciones del movimiento se ha obtenido de este *blockset*. También se pueden encontrar en esta librería multitud de bloques con modelos estándar, como es el caso de la Atmósfera Estándar Internacional (*International Standard Atmosphere*, ISA), el Modelo Magnético Mundial (*World Magnetic Model*, WMM), modelos gravitacionales o de ráfagas y turbulencias, entre otros.

Aerospace Blockset proporciona una gran cantidad de herramientas específicas para la simulación de aeronaves, por lo que el diseño de EFS trata de maximizar su uso a través de un estudio del *blockset* y los ejemplos que ofrece previo al comienzo del desarrollo.

4.2. *Software* propio de Embention

Además de Simulink para el desarrollo del simulador, se han empleado programas desarrollados por Embention para el autopiloto Veronte. El uso de este *software* es necesario para realizar validaciones del funcionamiento conjunto de EFS y el bloque Veronte en simulaciones SIL, que, en definitiva, es el objetivo del presente Trabajo de Fin de Máster.

Por su importancia, se va a exponer brevemente el funcionamiento de dos de los programas de Embention usados para la validación del simulador.

4.2.1. Veronte Link

Veronte Link permite comprobar si existe algún autopiloto conectado al ordenador, ya sea un Veronte real o uno simulado. Los Veronte reales (Figura 1.2) se pueden conectar al ordenador con diversos motivos (realizar configuraciones, simulaciones *hardware in the loop*, comunicaciones tierra-aire, entre otros), aunque en simulaciones SIL se emplea un autopiloto simulado. En la Figura 4.1 se puede apreciar un autopiloto recogido por Veronte Link (1x - 1599), conectado, correctamente configurado y en funcionamiento. En este caso se trata de un autopiloto simulado, es decir, en el ordenador se está ejecutando un bloque Veronte de Simulink. En cualquier caso, Veronte Link no realiza distinción entre autopilotos reales y simulados.

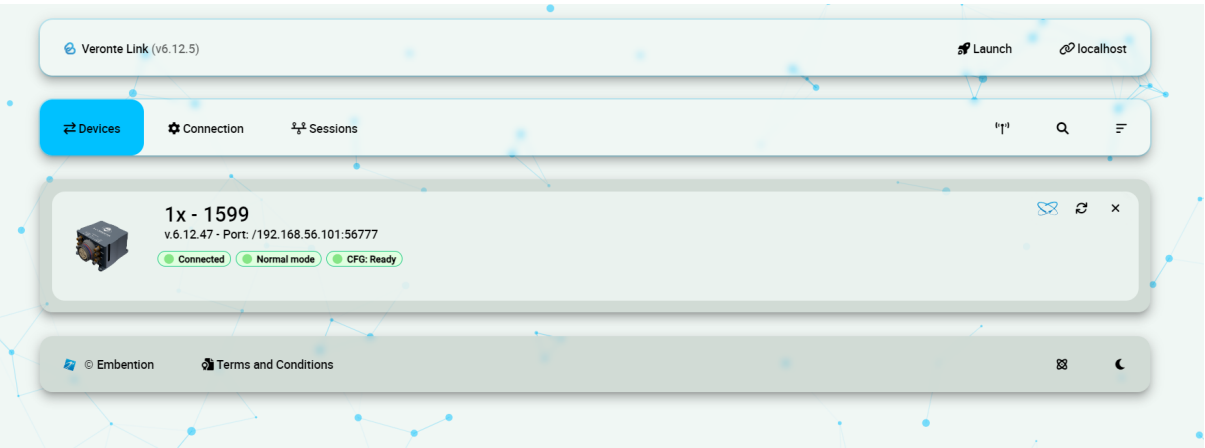

Figura 4.1: Veronte Link 6.12 con un autopiloto conectado (Embention)

Es necesario para comprobar la configuración del autopiloto y su funcionamiento durante la ejecución de la simulación en Simulink.

4.2.2. Veronte Ops

Veronte Ops es el *software* de Embention destinado a la realización de operaciones o misiones de la aeronave. Permite visualizar la geolocalización de la aeronave, el valor de ciertos parámetros y variables internas del autopiloto o la señal leída por sus sensores. También se puede controlar la aeronave o especificar una etapa dentro de la misión. En la Figura 4.2 se tiene una captura de Veronte Ops durante la misión simulada de un hexacóptero, realizada durante el desarrollo de *Embention Flight Simulator*. Se puede apreciar la posición del acelerador de cada motor, el horizonte artificial en funcionamiento a partir de la señal de los acelerómetros simulados, así como un panel de control de las etapas de la misión.

Como se detallará posteriormente, la validación del funcionamiento de EFS se ha realizado comprobando si una aeronave simulada podía realizar un vuelo utilizando la configuración del autopiloto embarcado en la aeronave real, habiendo comprobado previamente que la aeronave real completaba la misión correctamente. Veronte Ops es la herramienta empleada para realizar operaciones con la aeronave simulada y comprobar su desempeño.

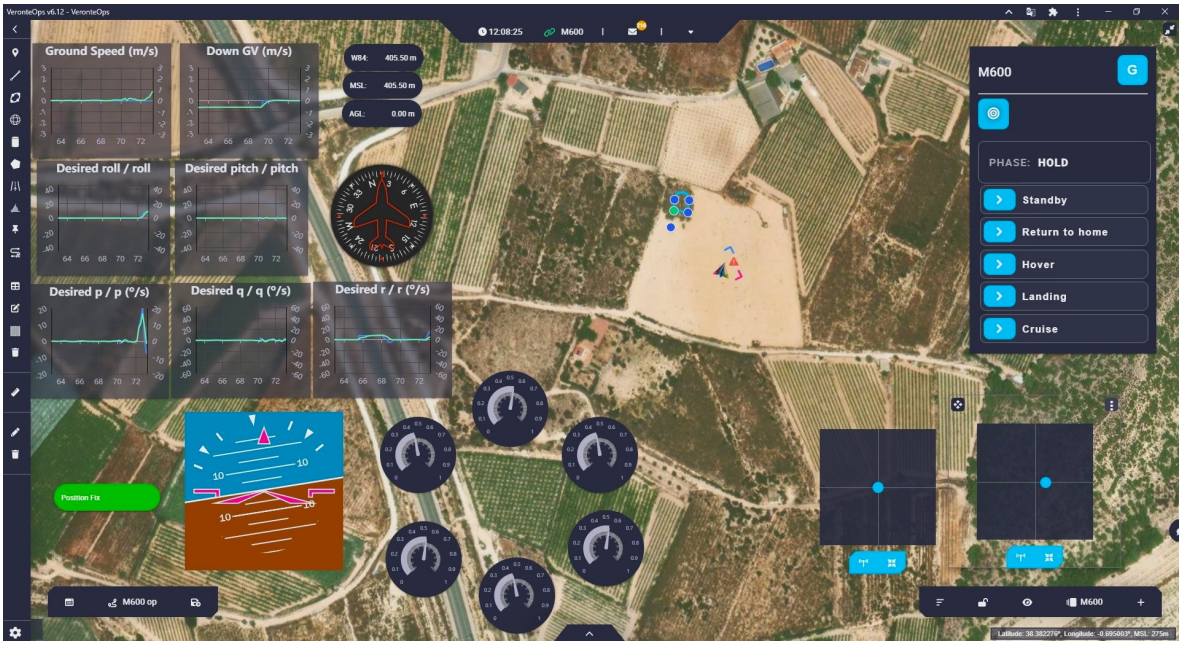

Figura 4.2: Veronte Ops 6.12 durante una simulación (Embention)

Capítulo 5

Embention Flight Simulator

Una vez expuestos los objetivos, realizado un estudio previo del estado del arte en simulación de vuelo y recopiladas las herramientas necesarias, se procede a explicar detalladamente el simulador de vuelo desarrollado: *Embention Flight Simulator*.

5.1. Estructura del simulador

La Figura 5.2 muestra la estructura interna de *Embention Flight Simulator*. Aparecen representadas muchas de las características que se han expuesto en capítulos anteriores. Por un lado, los bloques que están superpuestos representan variantes de Simulink, es decir, en estos bloques se debe escoger y configurar un modelo antes de ejecutar la simulación. Por otro lado, el bloque Ecuaciones del movimiento (6DOF) corresponde con *Custom Variable Mass 6DOF (Quaternion)* descrito en 3.1, que obtiene la posición y orientación a partir de las fuerzas y momentos proporcionadas por el modelo de aeronave.

Otro detalle que se aprecia en el esquema es que el entorno de vuelo y la aeronave recogen varios modelos. Respecto a la caracterización de una aeronave, el modelo aerodinámico obtiene las fuerzas y momentos derivados de su interacción con el aire; el modelo de empuje, las generadas por los motores, mientras que el modelo de masa e inercia calcula el peso y contempla las variaciones en la masa y la matriz de inercia que puede sufrir durante su operación. En el caso del modelo del entorno, EFS obtiene parámetros del aire según su modelo atmosférico (densidad, temperatura, presión, entre otros). El modelo magnetosférico devuelve el campo magnético terrestre, necesario para la simulación de las brújulas. Por último, el modelo de viento permite considerar el efecto aerodinámico del aire cuando la atmósfera no está en calma.

Aunque el funcionamiento concreto de cada modelo se detalla en Sección 6.2, la Figura 5.2 facilita seguir el proceso que realiza el simulador para calcular las salidas:

1. El simulador recibe las entradas, que son las señales de los actuadores que controlan la

aeronave. En una simulación SIL, estas señales serán la salida de control de Veronte.

2. Las señales pasan por el modelo de actuadores seleccionado, que tratan de replicar el funcionamiento del actuador real (motores y servomotores principalmente).

3. En el modelo de la aeronave se recibe la posición de los actuadores, el estado cinemático de la aeronave y las condiciones ambientales. Con estos datos, el modelo envía las fuerzas y los momentos aplicados, así como otros datos requeridos por el bloque de ecuaciones del movimiento, como la masa y el tensor de inercia.

4. A partir de estas fuerzas y los momentos, el bloque 6DOF calcula la posición, velocidad y aceleración de la aeronave en el siguiente instante, como se describe en la Sección 3.1.

5. Estos estados son una de las salidas del simulador, empleados principalmente para representaciones y comparaciones. También se envían a los modelos del simulador que los requieren para calcular en el instante siguiente (por ejemplo, se necesita la velocidad para obtener la resistencia aerodinámica, o la altitud para calcular la densidad del aire).

6. El modelo del entorno devuelve ciertos parámetros ambientales a partir de la posición de la aeronave. Estos se envían a los modelos de aeronave y también se emplean para simular los sensores.

7. Finalmente, a partir de los estados y de las condiciones ambientales, se generan las señales de los sensores simulados, como se detalla en Sección 6.3.

Por otro lado, en la Figura 5.1, se muestra la estructura de archivos y carpetas que forman el proyecto de Simulink de *Embention Flight Simulator*. Cada carpeta recoge los siguientes archivos:

- **aerodynamicModels**: almacena los archivos de Simulink (*.slx*) que modelan el comportamiento aerodinámico de la aeronave simulada.
- **configuration**: contiene todos los *scripts* de Matlab en los que se leen los parámetros necesarios para el funcionamiento del simulador (parámetros de configuración del simulador y de los modelos).
- **develop**: es una carpeta auxiliar donde se almacenan los modelos y herramientas en desarrollo.
- **examples**: contiene modelos ya configurados de Embention, a modo de ejemplo.
- **mainModels**: en esta carpeta se recogen los modelos de Simulink principales, es decir, el modelo del simulador de vuelo y otros modelos preparados para el funcionamiento del simulador de forma aislada o en conjunto con el bloque Veronte.
- **massInertiaModels**: contiene los modelos de Simulink que caracterizan la masa y la inercia de la aeronave.

- **outputs**: almacena los archivos de salida del simulador, como la generación de los archivos *.kml* o los resúmenes de simulación.
- **projectScripts**: contiene ciertos ficheros relacionados con el proyecto de Simulink, como atajos o el archivo de bienvenida.
- **resources**: es la carpeta donde se almacenan imágenes y logos, además de ciertos recursos creados automáticamente por el proyecto de Simulink.
- **thrustModels**: contiene los modelos de Simulink que caracterizan el empuje de los motores de la aeronave simulada.
- **tools**: recoge todas las herramientas desarrolladas para el simulador.
- **VeronteSIL**: es la carpeta donde se encuentran los archivos relacionados con el bloque Veronte para simulaciones SIL.
- **work**: en esta carpeta Simulink almacena los archivos generados durante la compilación y el funcionamiento del simulador.
- **EFSlauncher.m**: es el archivo del lanzador del simulador en formato *script* de Matlab.
- **EFSlauncher_live.mlx**: lanzador del simulador en formato *live script* de Matlab.

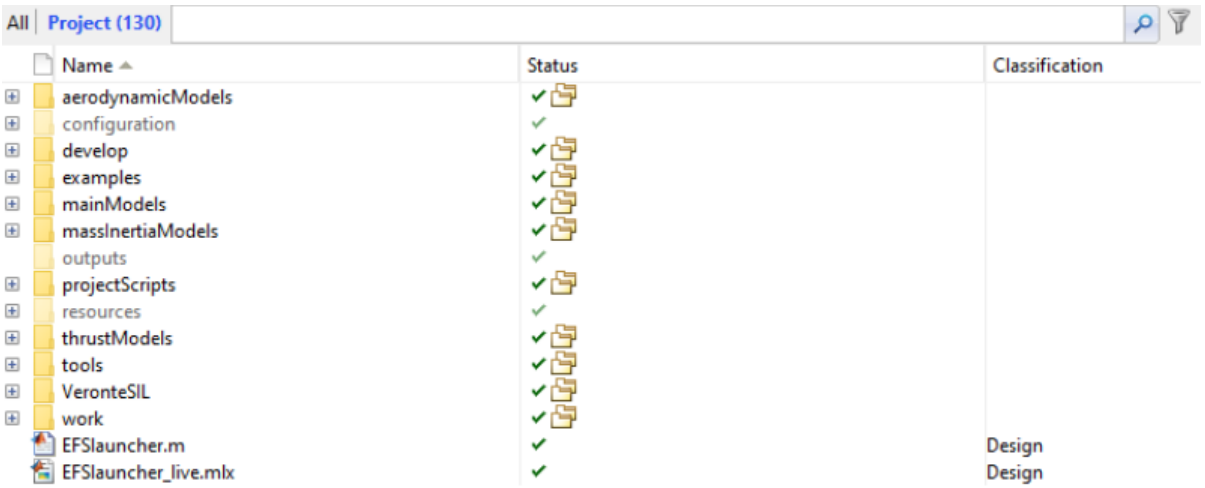

Figura 5.1: Estructura de las carpetas en el *Simulink Project*

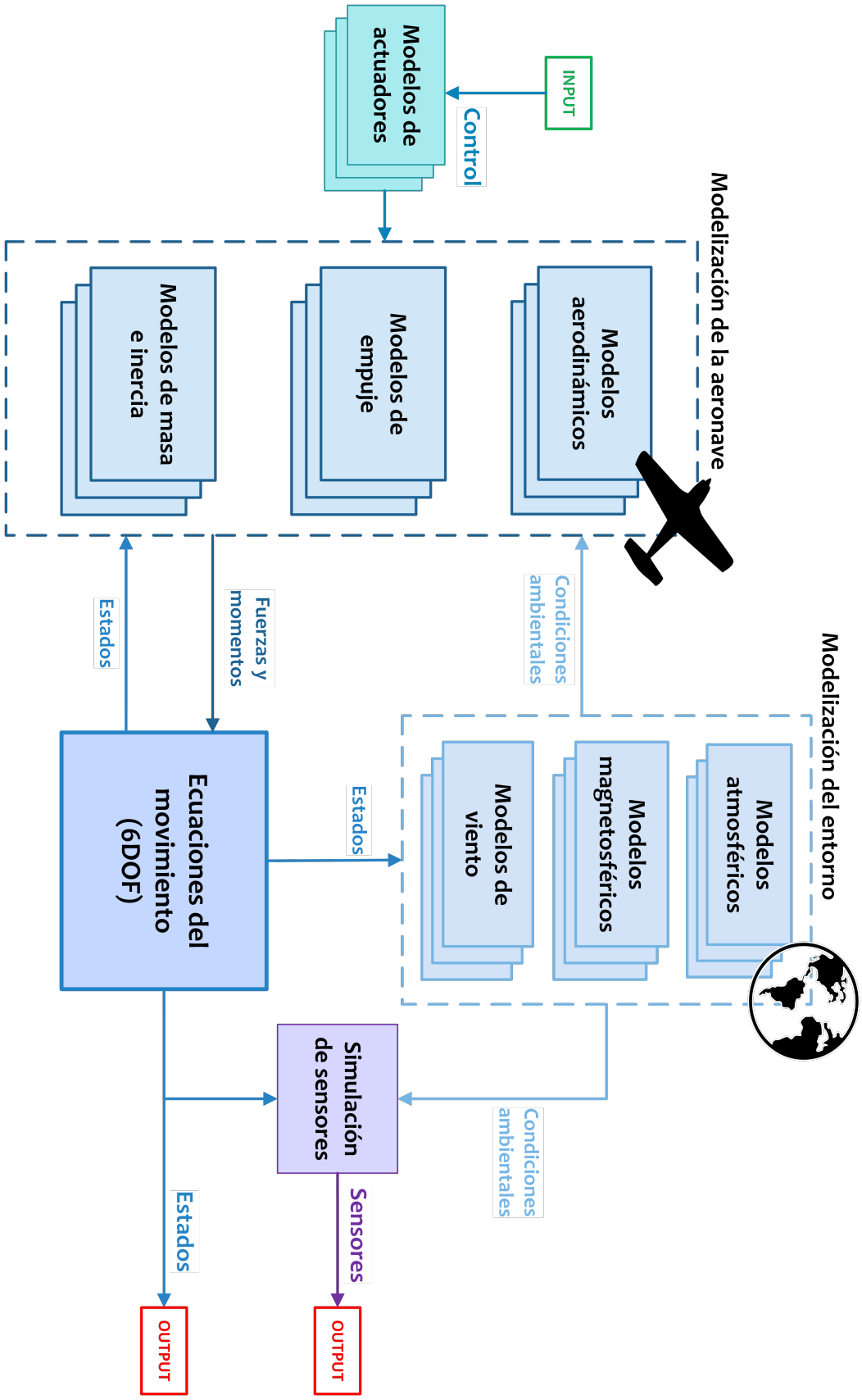

Figura 5.2: Esquema del funcionamiento interno de EFS

5.2. Uso y utilidades del simulador

En general, para realizar una simulación, se ejecuta el lanzador (*.m* o *.mlx*, según se prefiera). En este lanzador se especifican los modelos empleados para representar una aeronave, y se leen los archivos de la carpeta *configuration* requeridos, según los modelos escogidos.

El empleo de *Simulink Projects* facilita la ejecución de simulaciones y, además, permite crear atajos (*shortcuts*) a los ejemplos y a otras herramientas que se han desarrollado con la intención de ayudar a la modelización de las aeronaves. Estas herramientas se detallan en el Capítulo 7. Además, en Anexo I - Manual de uso y desarrollo, se expone con más detalle cómo trabajar con el simulador, desde el punto de vista del usuario.

5.3. El bloque *Embention Flight Simulator*

Toda la estructura descrita se encuentra dentro del bloque *Embention Flight Simulator* de Simulink, mostrado en la Figura 5.3. A la izquierda del bloque se tienen las entradas; a la derecha, las salidas, y en la parte superior el puerto *enabled* que permite controlar la ejecución del simulador de vuelo, como se ha descrito en 4.1.3.

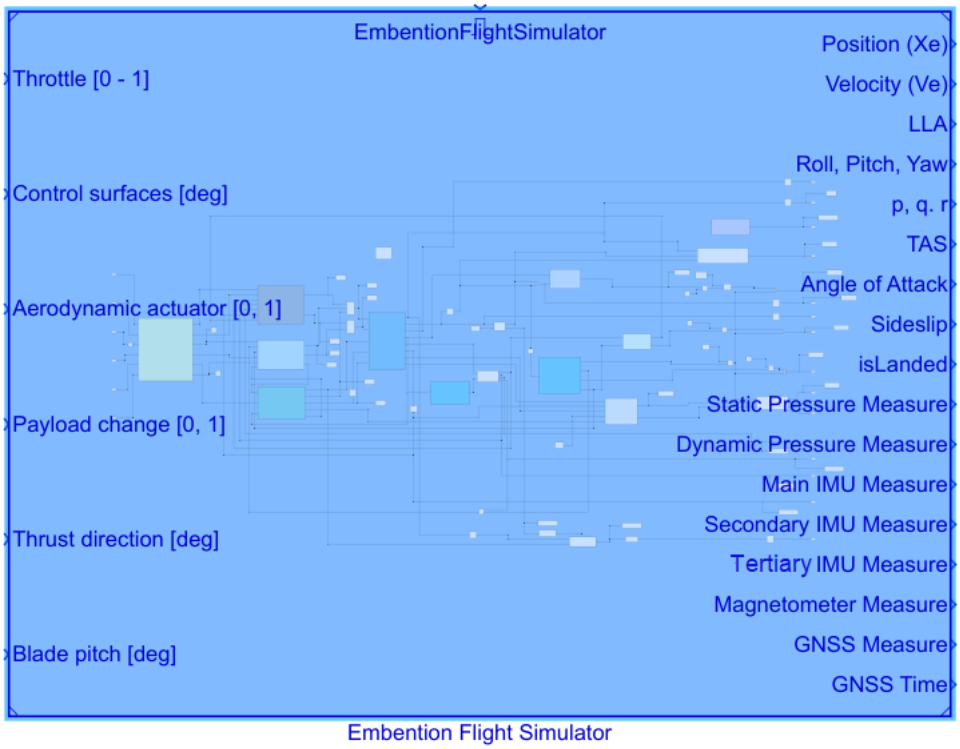

Figura 5.3: Bloque de *Embention Flight Simulator* en Simulink

Este bloque se añade en un modelo de Simulink junto con otros bloques para su funcionamiento. En el caso de simulaciones SIL, se realiza la conexión con el bloque Veronte, como en el ejemplo mostrado en la Figura 5.4. Cabe destacar que no es necesario conectar todos los puertos, ya que esto depende del caso de uso y de los modelos empleados. Además, se ha incluido un bloque de representación de resultados que, aunque no es requerido para el funcionamiento, permite comprobar el funcionamiento y cumple con el esquema mostrado en la Figura 1.1.

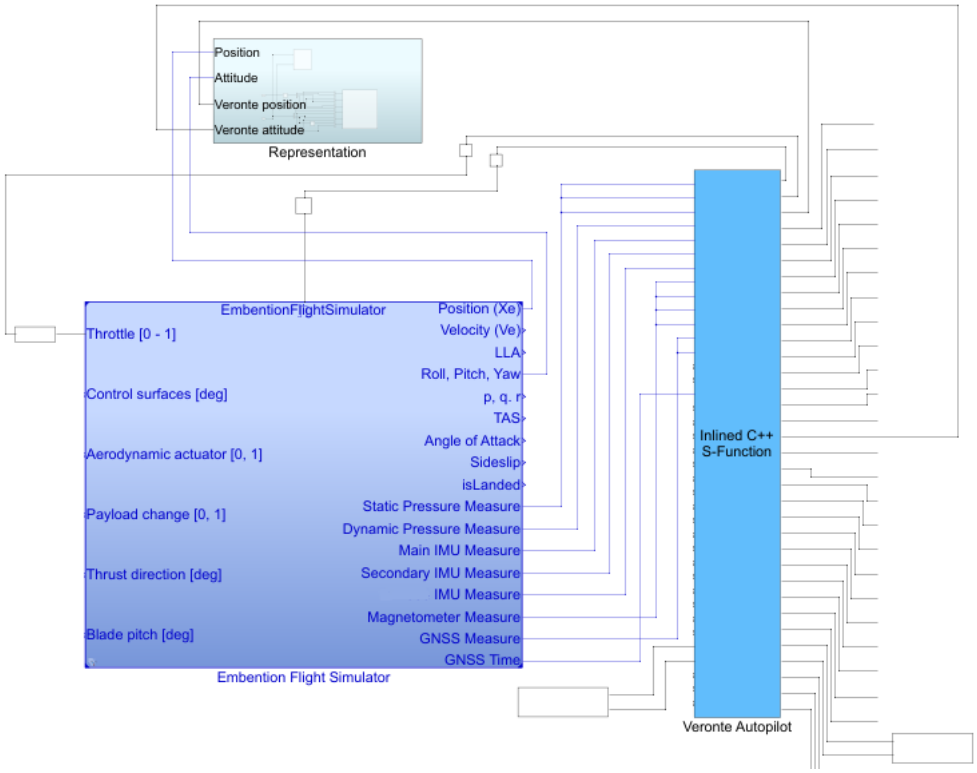

Figura 5.4: Ejemplo de modelo para simulaciones SIL (EFS + Veronte)

5.3.1. Entradas y salidas del simulador

En las Tablas 5.1 y 5.2 se recogen las salidas y entradas del bloque EFS, respectivamente, junto con su formato requerido y una breve descripción de los parámetros que pasan por cada puerto.

Las entradas necesarias para el funcionamiento del simulador dependen de los modelos escogidos, por lo que algunos puertos pueden quedar desconectados. El simulador ignora las entradas proporcionadas que no requiere. Además, para mayor flexibilidad, los vectores o matrices de entrada son de tamaño variable, de forma que no se requiere modificar el simulador cuando el número de actuadores cambia. En cualquier caso, es necesario comprobar que las señales de entrada tienen el formato y las unidades requeridas por el simulador, siendo necesario adaptarlas en algunos casos.

Respecto a las salidas, se tienen estados y señales de sensores. Las salidas de la parte superior del bloque son estados, mientras que las inferiores son señales de los sensores simulados, con el formato que indica las entradas del bloque Veronte. De esta forma, se puede conectar la salida del bloque EFS directamente con la entrada correspondiente de Veronte, según indican en su manual [36].

Tabla 5.1: Salidas del bloque *Embention Flight Simulator*

Salida	**Formato**	**Descripción**
Position (Xe)	[m, m, m]	Posición (sistema Tierra plana)
Velocidad (Ve)	[m/s, m/s, m/s]	Velocidad (sistema Tierra plana)
LLA	[deg, deg, m]	Latitud, Longitud, Altitud
Roll, Pitch, Yaw	[rad, rad, rad]	Alabeo, cabeceo y guiñada
TAS	[m/s]	*True Airspeed*
Angle of attack	[rad]	Ángulo de ataque (α)
Sideslip	[rad]	Ángulo de derrape (β)
isLanded	0:OFF, 1:ON	1 en caso de contacto con el suelo
Static Pressure Measure	Como Veronte SIL	Medida de presión estática
Dynamic Pressure Measure	Como Veronte SIL	Medida de presión dinámica
Main IMU Measure	Como Veronte SIL	Medida de IMU 1 de Veronte
Secondary IMU Measure	Como Veronte SIL	Medida de IMU 2 de Veronte
Tertiary IMU Measure	Como Veronte SIL	Medida de IMU 3 de Veronte
Magnetometer Measure	Como Veronte SIL	Medida del magnetómetro
GNSS Measure	Como Veronte SIL	Medida de posición GNSS
GNSS Time	Como Veronte SIL	Medida de tiempo GNSS

Tabla 5.2: Entradas al bloque *Embention Flight Simulator*

Entrada	**Formato**	**Descripción**
Throttle	De 0 a 1	Vector posición del acelerador de cada motor
Control surfaces	[deg]	Vector deflexión de cada superficie de control
Aerodynamic actuator	0:OFF, 1:ON	Actuadores discretos aerodinámicos
Payload change	0:OFF, 1:ON	Cambio de carga de pago
Thrust direction	[deg]	Matriz con las direcciones de empuje (*tilt*)
Blade pitch	[deg]	Ángulo de paso colectivo de la hélice

Capítulo 6

Modelos de *Embention Flight Simulator*

Como se ha expuesto en este trabajo de fin de máster, la simulación del vuelo basa su funcionamiento en los modelos matemáticos que caracterizan el comportamiento de la aeronave simulada. En este capítulo se pretende exponer la forma en la que *Embention Flight Simulator* recoge los modelos matemáticos de las aeronaves para su posterior simulación.

6.1. Geometría de la aeronave

La geometría de la aeronave suele ser determinante en su comportamiento. EFS requiere conocer, en la mayoría de casos, la posición y orientación de la planta propulsora y las superficies y longitudes características.

Como se comenta en la sección Sistemas de referencia, el sistema fijo en el cuerpo es el empleado para indicar la posición de los motores. En la Figura 6.1 se muestra un ejemplo de la geometría de los motores de un avión bimotor. Este tipo de representaciones son generadas automáticamente por EFS una vez se introducen los datos, facilitando el proceso de modelización.

La dirección del empuje se indica mediante coordenadas polares. También se indica el sentido de rotación de motor, un dato relevante en hélices para conocer el sentido del momento de reacción que provocan al girar. El símbolo (+) indica un sentido de rotación en contra de la agujas del reloj, mientras que (-) indica sentido horario.

Los parámetros requeridos para determinar la dirección del empuje son el acimut (ϕ) y la colatitud (o polar, θ) [37]. El acimut es el ángulo respecto al eje x, en el plano xy, y puede tener valores entre 0 y 360 grados. La colatitud, por otro lado, es el ángulo respecto al eje

z. De esta forma, utilizando estos dos ángulos, queda determinada la dirección del empuje. En la Figura 6.1 se han situado dos motores (*Thrusters*) con un acimut de 0 grados y una colatitud de 90 grados. Cabe destacar que la magnitud del empuje se determina en cada paso de simulación según el modelo de empuje escogido, como se detalla posteriormente.

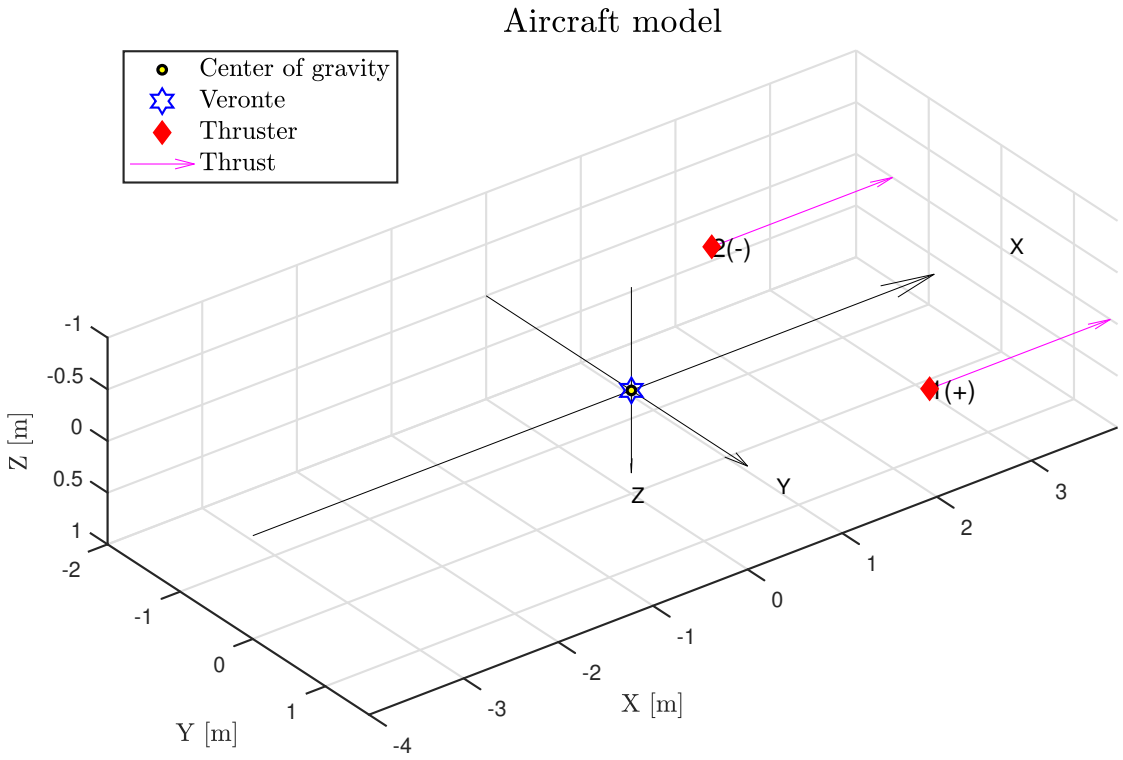

Figura 6.1: Geometría de la planta propulsora de un avión bimotor

Por otro lado, respecto a la geometría de las superficies aerodinámicas, se requiere generalmente proporcionar la cuerda media aerodinámica (*mean aerodynamic chord*, c), la envergadura (b), y la superficie alar (S_w). Estos parámetros se utilizan para calcular fuerzas aerodinámicas utilizando coeficientes y derivadas de estabilidad en algunos modelos. Algunos tipos de aeronave, como los multicópteros, no disponen esta información ni la necesitan para ser modelados en EFS.

6.2. Modelos desarrollados

A continuación, se van a detallar los modelos desarrollados para EFS, con los cuales se caracteriza el vuelo de las aeronaves simuladas. Estos modelos se recogen en la Tabla 6.1. Cabe recordar que los modelos de un mismo grupo se encuentran dentro del mismo bloque, con formato de variante de Simulink (*Simulink.Variant*).

Tabla 6.1: Modelos incluidos en *Embention Flight Simulator*

Grupo (*Simulink.Variant*)	**Modelos incluidos**
Modelos aerodinámicos	Sin fuerzas aerodinámicas
	Drag lineal
	Aerodinámica por coeficientes
Modelos de empuje	Sin empuje
	Empuje simple
	Empuje simple vectorial
	Empuje tabulado
	Empuje tabulado con paso colectivo variable
Modelos de masa e inercia	Masa e inercia constante
	Reparto de paquetes
	Consumo de combustible simple
	Consumo de combustible
Modelos de actuadores	Actuadores ideales
	Actuadores de segundo orden
Modelos de viento	Sin viento
	Viento constante
Modelos atmosféricos	Atmósfera Estándar Internacional (ISA)
Modelos magnetosféricos	Modelo Magnético Mundial (WMM)

6.2.1. Modelos aerodinámicos

Los modelos aerodinámicos son los modelos de la aeronave que obtienen las fuerzas y momentos derivados de su interacción con el aire. Los modelos desarrollados son tres: sin fuerzas aerodinámicas, *drag* lineal y aerodinámica por coeficientes.

6.2.1.1. Sin fuerzas aerodinámicas

El primer modelo ignora la interacción de la aeronave con el aire, devolviendo fuerzas y momentos nulos. Por lo general, esto está muy alejado de la realidad, pero permite aislar y comprobar otras fuerzas, como el peso y el empuje. También puede ser empleado como primera aproximación de aeronaves que se desplacen a bajas velocidades.

6.2.1.2. *Drag* lineal

El modelo de *drag* lineal relaciona, mediante una constante, la velocidad relativa con el aire y la resistencia aerodinámica, como describe la Ecuación 6.1. Este modelo, empleado en algunos simuladores básicos de multicópteros descritos anteriormente [16], se basa en la resistencia de Stokes.

$$\begin{bmatrix} FD_x \\ FD_y \\ FD_z \end{bmatrix} = \begin{bmatrix} kd_x \cdot V_x \\ kd_y \cdot V_y \\ kd_z \cdot V_z \end{bmatrix} \tag{6.1}$$

La ley de Stokes proporciona el *drag* de una esfera que se desplaza por un fluido a bajo número de Reynolds, siguiendo la Ecuación 6.2. En esta fórmula se aprecia una dependencia de la resistencia con la velocidad de desplazamiento por el fluido, u. También depende de la viscosidad dinámica, ν y del radio de la esfera, R, es decir, de las características del fluido y de la geometría del objeto.

$$FD = 6\pi\nu Ru \tag{6.2}$$

El modelo de resistencia lineal desprecia la variación de la viscosidad dinámica del aire en función de las condiciones de vuelo. Además, asume que la aeronave es una esfera y que el número de Reynolds del vuelo es bajo, es decir, que las velocidades son bajas. De esta forma, $6\pi\nu R$ se puede asumir constante, agrupándolo en un solo término kd, según cada eje de la aeronave.

Se trata de un modelo simple, con diversas suposiciones que lo alejan de la realidad, pero que añade resistencia aerodinámica a multicópteros de baja velocidad, empleando

únicamente una constante. Por otro lado, los momentos aerodinámicos son nulos en este modelo.

6.2.1.3. Aerodinámica por coeficientes

Los coeficientes aerodinámicos de aeronaves, principalmente de ala fija, son usados de forma amplia en simulación de vuelo [39]. EFS incluye un modelo siguiendo conceptos expuestos en algunos trabajos de simulación de vuelo en 6 grados de libertad [40] [41].

El modelo de aerodinámica por coeficientes calcula las fuerzas y momentos mediante coeficientes y derivadas aerodinámicas. Es un modelo con gran flexibilidad ya que permite incluir coeficientes relacionados con el estado de la aeronave: α, $\dot{\alpha}$, β, $\dot{\beta}$, p, q, r; relacionados con las superficies de control (alerones, *flaps*, *slats*, *rudder*, elevadores...) y con actuadores discretos (tren de aterrizaje o gimbales, entre otros). Cabe destacar que los coeficientes relacionados con estados han sido escogidos siguiendo el criterio de algunos tratados de mecánica de vuelo [42], [41].

Los coeficientes se proporcionan en forma de matriz, en tres matrices distintas. En primer lugar, la matriz *aeroCoeff* (Ecuación 6.3) recoge coeficientes relacionados con estados de la aeronave. La matriz *controlCoeff* (Ecuación 6.4), relaciona la deflexión de las superficies de control con las fuerzas y momentos generados. Por último, *discreteAeroCoeff* contiene los coeficientes de actuadores discretos, es decir, se aplican cuando el sistema que representan está desplegado como, por ejemplo, un tren de aterrizaje. En estas matrices puede haber valores nulos en caso de que no se quiera aplicar o se desconozca el coeficiente determinado. Además, *controlCoeff* y *discreteAeroCoeff* pueden ser de dimensión variable para adaptarse a diferente número de actuadores.

$$\text{aeroCoeff} = \begin{pmatrix} CD_0 & CD_\alpha & CD_{\dot{\alpha}} & CD_\beta & CD_{\dot{\beta}} & CD_p & CD_q & CD_r \\ CY_0 & CY_\alpha & CY_{\dot{\alpha}} & CY_\beta & CY_{\dot{\beta}} & CY_p & CY_q & CY_r \\ CL_0 & CL_\alpha & CL_{\dot{\alpha}} & CL_\beta & CL_{\dot{\beta}} & CL_p & CL_q & CL_r \\ Cl_0 & Cl_\alpha & Cl_{\dot{\alpha}} & Cl_\beta & Cl_{\dot{\beta}} & Cl_p & Cl_q & Cl_r \\ Cm_0 & Cm_\alpha & Cm_{\dot{\alpha}} & Cm_\beta & Cm_{\dot{\beta}} & Cm_p & Cm_q & Cm_r \\ Cn_0 & Cn_\alpha & Cn_{\dot{\alpha}} & Cn_\beta & Cn_{\dot{\beta}} & Cn_p & Cn_q & Cn_r \end{pmatrix} \tag{6.3}$$

$$\text{controlCoeff} = \begin{pmatrix} CD_{\text{control surface 1}} & \cdots & CD_{\text{control surface } n} \\ CY_{\text{control surface 1}} & \cdots & CY_{\text{control surface } n} \\ CL_{\text{control surface 1}} & \cdots & CL_{\text{control surface } n} \\ Cl_{\text{control surface 1}} & \cdots & Cl_{\text{control surface } n} \\ Cm_{\text{control surface 1}} & \cdots & Cm_{\text{control surface } n} \\ Cn_{\text{control surface 1}} & \cdots & Cn_{\text{control surface } n} \end{pmatrix} \tag{6.4}$$

$$\text{discreteAeroCoeff} = \begin{pmatrix} CD_{\text{discrete actuator 1}} & \cdots & CD_{\text{discrete actuator } n} \\ CY_{\text{discrete actuator 1}} & \cdots & CY_{\text{discrete actuator } n} \\ CL_{\text{discrete actuator 1}} & \cdots & CL_{\text{discrete actuator } n} \\ Cl_{\text{discrete actuator 1}} & \cdots & Cl_{\text{discrete actuator } n} \\ Cm_{\text{discrete actuator 1}} & \cdots & Cm_{\text{discrete actuator } n} \\ Cn_{\text{discrete actuator 1}} & \cdots & Cn_{\text{discrete actuator } n} \end{pmatrix} \tag{6.5}$$

Conociendo los mencionados coeficientes, se obtienen los totales sumando las contribuciones, como se muestra en la Ecuación 6.6 para un coeficiente genérico. Los coeficientes de actuadores discretos se multiplican por 0 o por 1 según el estado del sistema discreto. Por otro lado, los coeficientes de superficies de control se multiplican por la deflexión de dicha superficie en radianes (δ).

$$\begin{aligned} C = C_0 \; + \; C_\alpha \; \alpha \; + \; C_{\dot{\alpha}} \; \dot{\alpha} \; + \; C_\beta \; \beta \; + \; C_{\dot{\beta}} \; \dot{\beta} \; + \; C_p \; p \; + \; C_q \; q \; + \; C_r \; r \; + \\ C_{\text{control surfaces}} \; \delta_{\text{control surfaces}} \; + \; C_{\text{discrete actuators}} \end{aligned} \tag{6.6}$$

Finalmente, las fuerzas y momentos se obtienen aplicando las definiciones de los coeficientes (Ecuación 6.7), siendo q la presión dinámica. El *drag* (D), la fuerza lateral (Y) y el *lift* (L) son fuerzas aerodinámicas en ejes viento, es decir, la resistencia aerodinámica tiene la misma dirección que la velocidad relativa de la aeronave respecto al aire. Debido a esto, es necesario realizar una rotación para obtener las fuerzas en ejes fijos al cuerpo, como requiere el bloque 6DOF. Esta rotación se realiza empleando el ángulo de ataque (α) y de deslizamiento (β). Por otro lado, los momentos en los ejes x, y y z de la aeronave están determinados por l, m y n, respectivamente.

$$\begin{aligned} D &= q\ S_w\ C_D \\ Y &= q\ S_w\ C_Y \\ L &= q\ S_w\ C_L \\ l &= q\ S_w\ b\ C_l \\ m &= q\ S_w\ c\ C_m \\ n &= q\ S_w\ b\ C_n \end{aligned} \tag{6.7}$$

También es posible indicar la resistencia como función de la sustentación, según la curva polar (Ecuación 6.8). Para ello, se proporciona el coeficiente de resistencia inducida, K y se establecen los coeficientes de resistencia innecesarios a 0, dentro de la matriz *aeroCoeff*.

$$CD = CD_0 + K \cdot (CL - CL_0)^2 \tag{6.8}$$

Este modelo no tiene en cuenta la entrada en pérdida, aunque sí limita el coeficiente de sustentación a un valor máximo. Además, se debe tener en cuenta que los coeficientes se obtienen generalmente a partir de una linealización entorno a unas condiciones de vuelo, por lo que su validez se ve afectada en maniobras alejadas de dichas condiciones. En general, se trata de una limitación asumible, puesto que Embention planea usar el simulador principalmente en validación de etapas de crucero.

6.2.2. Modelos de empuje

Los modelos de empuje obtienen las fuerzas y momentos provocados por los motores o propulsores montados en la aeronave. Se han desarrollado principalmente modelos para hélices debido a que es empleado por la mayoría de clientes de Embention, aunque se pueden añadir modelos de otros tipos de motor como el turbofán disponible en *Aerospace Blockset*.

6.2.2.1. Sin empuje

El primer modelo se trata del modelo sin empuje, que representa una aeronave sin motores. En este caso, las fuerzas y momentos derivados de los propulsores son nulos.

6.2.2.2. Empuje simple

El modelo de empuje simple permite relacionar de forma sencilla la posición del acelerador (u, de 0 a 1) con el empuje y el par generados por el motor. En este modelo se ha optado por un empuje linealmente dependiente con la velocidad de rotación de la hélice, mientras

que el momento se relaciona de forma cuadrática. Las ecuaciones 6.9 y 6.10 muestran esta relación a través de los parámetros k y b para el empuje y el par, respectivamente.

$$\begin{bmatrix} T_x \\ T_y \\ T_z \end{bmatrix} = \begin{bmatrix} k\ \Omega\ \sin\theta\cos\phi \\ k\ \Omega\ \sin\theta\sin\phi \\ k\ \Omega\ \cos\theta \end{bmatrix} \tag{6.9}$$

La entrada del modelo es la posición del acelerador, que se relaciona linealmente con la velocidad de rotación, Ω, mediante la velocidad máxima de cada motor: $\Omega = u \cdot rpm_{\max}$. Por otro lado, para obtener la dirección del empuje, se emplean los ángulos de acimut, ϕ y colatitud, θ, descritos en la modelización de la geometría.

$$\begin{bmatrix} Q_x \\ Q_y \\ Q_z \end{bmatrix} = \begin{bmatrix} b\ \Omega^2\ \sin\theta\cos\phi + T_y\ z_{\text{motor}} + T_z\ y_{\text{motor}} \\ b\ \Omega^2\ \sin\theta\sin\phi - T_z\ x_{\text{motor}} + T_x\ z_{\text{motor}} \\ b\ \Omega^2\ \cos\theta - T_y\ x_{\text{motor}} - T_x\ y_{\text{motor}} \end{bmatrix} \tag{6.10}$$

En el cálculo del par del motor se tiene en cuenta el sentido de rotación de la hélice, de forma que el momento generado en la aeronave, puesto que es una reacción, tiene sentido contrario. Además, en este cálculo se incluye también el momento debido al empuje en relación con su distancia al resto de ejes, empleando la posición de los motores y aplicando el criterio de signos correspondiente.

El modelo de empuje también obtiene el consumo de combustible requerido por el motor. Este consumo viene determinado por el consumo específico por empuje ($TSFC$, *Thrust-specific fuel consumption*), que relaciona el empuje con la masa de combustible consumida por unidad de tiempo. De esta forma, el modelo proporciona el gasto de combustible por unidad de tiempo al resto de modelos del simulador, según el empuje total generado. Cabe destacar que este consumo puede ser nulo para aeronaves eléctricas, aunque, en cualquier caso, el consumo es tratado por los modelos de masa, descritos posteriormente.

6.2.2.3. Empuje simple vectorial

El modelo de empuje simple vectorial funciona de forma análoga al modelo de empuje simple. Sin embargo, este modelo incluye como entradas los ángulos de acimut y de colatitud de los motores, de forma que se puede variar la dirección del empuje durante el vuelo, simulando aeronaves con empuje vectorial.

6.2.2.4. Empuje tabulado

El modelo de empuje tabulado emplea las ya comentadas *Lookup Table* para relacionar un valor de entrada, la posición del acelerador, con un valor de empuje y de par del motor. El hecho de emplear una única entrada facilita la creación del modelo, pero limita su precisión cuando las condiciones de vuelo se alejan de las del experimento realizado para la obtención de la tabla.

Las fuerzas y momentos en cada eje se obtienen de forma análoga a las ecuaciones 6.9 y 6.10, mediante la orientación y posición de cada motor.

Por otro lado, el consumo de combustible se obtiene con otra tabla de consulta que relaciona el empuje con el gasto de fuel por unidad de tiempo, de forma similar al modelo de empuje simple.

6.2.2.5. Empuje tabulado con paso colectivo variable

El modelo de empuje tabulado con paso colectivo variable es una mejora del caso anterior, realizada para la modelización de la aeronave de un cliente, descrita en la Sección 8.3. En este caso, las tablas de consulta tienen tres entradas.

La primera entrada es el ángulo de paso colectivo de la hélice, θ. El ángulo de paso, como se aprecia en la Figura 6.2, es el ángulo de la cuerda de la pala respecto al plano de rotación de la hélice. En este caso, es colectivo puesto que todas las palas adoptan el mismo ángulo durante toda la rotación. Esto no siempre es así, ya que muchos rotores permiten variar el paso de forma cíclica, es decir, dando un ángulo de paso determinado a la pala para cada posición angular. Por otro lado, los ángulos de batimiento y arrastre no son tenidos en cuenta por el modelo.

La segunda entrada al modelo es la densidad del aire, ρ. La densidad, como es sabido, afecta directamente a las fuerzas aerodinámicas y, por tanto, a las prestaciones de la hélice. De este modo se tiene un modelo útil para diferentes alturas de vuelo.

Por último, la tercera entrada del modelo es la velocidad de rotación, Ω. Esta velocidad se relaciona con la posición del acelerador, que es la entrada al simulador, mediante el parámetro de máxima velocidad de rotación del motor, análogamente al modelo de empuje simple.

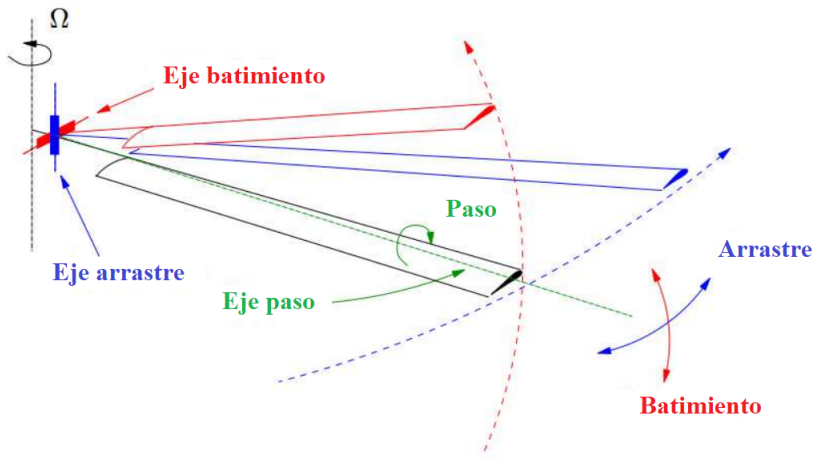

Figura 6.2: Ángulos de la pala de una hélice [43]

A partir de estas tres entradas, se obtiene el par y el empuje del motor, que es tratado de la misma forma que el resto de modelos. Según el acimut y la colatitud de cada motor, así como su posición, se obtienen los vectores de fuerza y momento aplicados en la aeronave. También el consumo de combustible se obtiene con *Lookup Tables* a partir del empuje total generado.

6.2.3. Modelos de masa e inercia

La masa y el tensor de inercia de la aeronave son parámetros de especial importancia en las ecuaciones del movimiento. Los modelos de masa e inercia de EFS son los encargados de representar su variación durante el vuelo de la aeronave simulada. Además, obtienen el vector gravedad y peso en los ejes fijos de la aeronave, según su orientación.

6.2.3.1. Masa e inercia constante

El primer modelo, masa e inercia constante, es el más simple. Para que ocurra una variación en la masa total de la aeronave es necesario que ésta se desprenda de partes o que consuma combustible. Por otro lado, el tensor de inercia (Ecuación 6.11) puede variar el valor de sus elementos si cambia la masa o la geometría.

$$I = \begin{bmatrix} I_{xx} & I_{xy} & I_{xz} \\ Iyx & I_{yy} & I_{yz} \\ I_{zx} & I_{zy} & I_{zz} \end{bmatrix} \tag{6.11}$$

Algunas aeronaves eléctricas, si no tienen equipos desplegables, mantienen constante su masa y su matriz de inercia. De esta forma, el modelo simplemente envía la masa y la

inercia inicial, proporcionada por el usuario.

Por otro lado, el modelo recibe la matriz de rotación de ejes Tierra plana a ejes fijos en el cuerpo, DCM_{be}, que es obtenido por el bloque 6DOF. Realizando el producto matricial como se muestra en la Ecuación 6.12, se transforma un vector con coordenadas en el sistema Tierra plana (superíndice e) a uno con coordenadas fijas en el cuerpo de la aeronave (superíndice b). Puesto que el vector gravedad es conocido en el sistema Tierra plana, con la matriz de rotación según la actitud de la aeronave en cada instante, es posible obtener el vector gravedad en el sistema fijo en la aeronave y, por tanto, el peso. El vector gravedad es necesario para simular los acelerómetros, mientras que el peso es una de las fuerzas que actúan en la aeronave.

$$\bar{v}^b = DCM_{be}\ \bar{v}^e \tag{6.12}$$

6.2.3.2. Reparto de paquetes

El modelo de reparto de paquetes almacena dos masas y dos matrices de inercia. Su objetivo es representar a una aeronave que se desprende de una carga de pago de forma instantánea, es decir, contiene la masa y la inercia de la aeronave con la carga de pago y sin la carga de pago. El modelo cambia de caso de carga mediante la entrada booleana *Payload change*.

La principal utilidad de este modelo es la simulación de aeronaves eléctricas de reparto de paquetes. Este tipo de aeronaves autónomas presentan unas características que las sitúan como potenciales clientes de Veronte.

6.2.3.3. Consumo de combustible simple (inercia constante)

Las aeronaves con motores de combustión pierden masa durante su uso. El modelo de consumo de combustible simple mantiene el tensor de inercia constante, pero calcula la masa de fuel perdida durante el vuelo. Para ello, integra el consumo de combustible por unidad de tiempo que obtiene del modelo de empuje, expuesto previamente. De esta forma, calcula la masa total consumida desde el inicio de la simulación.

Cabe destacar que el usuario también puede proporcionar la masa de la aeronave con los tanques vacíos, de forma que, cuando la masa de la aeronave alcanza este valor, el modelo envía una señal al modelo de empuje para detenerlo. De esta forma se simula una aeronave que se queda sin combustible durante el vuelo, o con un fallo de motor, permitiendo validar la respuesta de Veronte mediante aterrizajes de emergencia.

Por otro lado, también se calculan los vectores de gravedad y de fuerza, igual que en los

modelos anteriores.

6.2.3.4. Consumo de combustible

El modelo de consumo de combustible mejora al modelo simple incluyendo una tabla de consulta que relaciona valores de masa con tensores de inercia. De esta forma, se pueden simular aeronaves con tanques de combustible más grandes que no puedan despreciar la variación de inercia. En el resto de aspectos es análogo al modelo simple.

6.2.4. Modelos de actuadores

La aceleración de un motor o el cambio de posición del servomotor que mueve una superficie de control no se produce instantáneamente. Por tanto, es necesario modelar el comportamiento de los actuadores reales para conseguir simulaciones precisas. Éste es el objetivo de los modelos de actuadores.

6.2.4.1. Actuadores ideales

El primer modelo contempla actuadores ideales, es decir, los cambios en la posición de los actuadores se realizan de forma inmediata y perfecta. Aunque el comportamiento no es real, es útil cuando se desconoce la respuesta del actuador, o a la hora de simular aeronaves en etapas de diseño.

6.2.4.2. Actuadores de segundo orden

En este caso se modela la respuesta del actuador como una de segundo orden. Es un modelo inspirado en el bloque de *Aerospace Blockset Linear Second-Order Actuator*, que utiliza dos parámetros: la frecuencia natural del sistema (ω_n, *natural frequency*) y la tasa de amortiguamiento (ζ, *damping ratio*) [**?**].

En el caso del modelo desarrollado para *Embention Flight Simulator*, se utilizan estos dos parámetros para construir una función de transferencia de segundo orden en el dominio de Laplace, como la mostrada en la Ecuación 6.13 [44]. La *Control System Toolbox* facilita la creación de dichas funciones de transferencia, mediante su comandos *tf* y *c2d*. La función *tf* crea la función de transferencia tomando como parámetros la frecuencia natural y la tasa de amortiguamiento, mientras que *c2d* la transforma para adaptarse al funcionamiento discreto del simulador.

$$H(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n + \omega_n^2} \tag{6.13}$$

En la Figura 6.3 se muestra la comparativa entre la respuesta de un actuador ideal respecto a uno real, modelado mediante una función de transferencia de segundo orden. En el segundo 1 se produce el cambio del valor del actuador de 0 a 1.

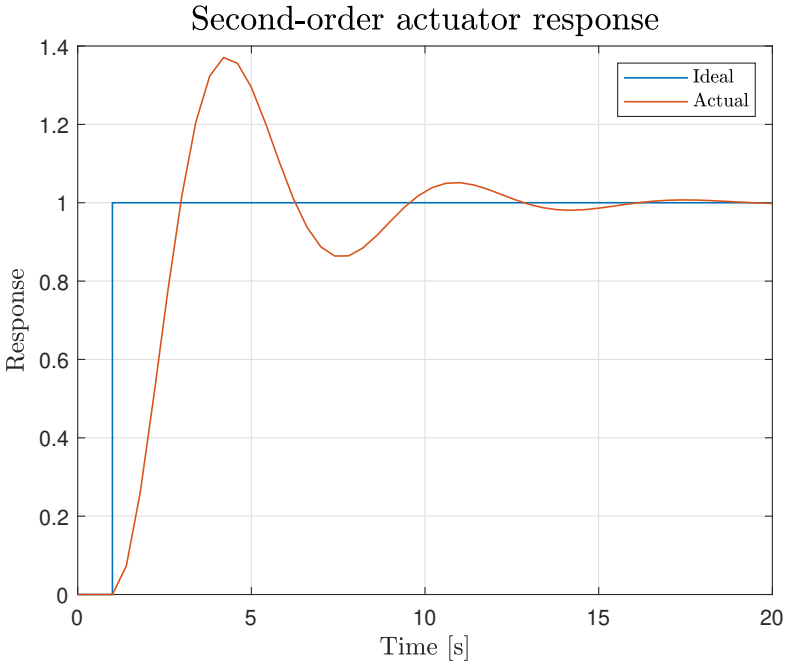

Figura 6.3: Respuesta de un actuador ideal y uno de segundo orden ($\omega_n = 1Hz$, $\zeta = 0{,}3$)

6.2.5. Modelos de viento

Los modelos de viento son modelos del entorno de vuelo que tratan de incluir los efectos de ciertos fenómenos atmosféricos en la simulación. En EFS se han desarrollado dos modelos, ya que *Aerospace Blockset* presenta una gran cantidad de bloques de turbulencias y ráfagas de viento [35] que se pueden incluir de forma sencilla en el simulador.

6.2.5.1. Sin viento

El primer modelo no incluye ningún efecto atmosférico, replicando una atmósfera en calma. Este modelo es útil en la mayoría de casos que no requieran comprobar una funcionalidad determinada o la respuesta de algún sistema ante ráfagas de viento o la aparición de turbulencias.

6.2.5.2. Viento constante

El segundo modelo de viento incluye viento constante, determinado por su dirección y su velocidad, ambos constantes. Esta velocidad se suma a la velocidad con la que la aeronave

obtiene sus fuerzas aerodinámicas, por lo que su efecto se ve afectado por la calidad de la modelización aerodinámica. Se trata de un modelo simple que permite añadir deriva a la aeronave y comprobar su corrección por parte de Veronte.

6.2.6. Modelos atmosféricos

Los modelos atmosféricos caracterizan el aire de la atmósfera, proporcionando ciertos parámetros de interés como su presión, su densidad o su temperatura.

6.2.6.1. *Inernational Standard Atmosphere* (ISA)

El modelo incluido en EFS para caracterizar la atmósfera es el bloque ISA de *Aerospace Blockset*. La atmósfera estándar internacional (ISA, *International Standard Atmosphere*) utiliza la altitud para calcular la temperatura, la presión, la viscosidad y la densidad del aire. Se trata de un modelo estandarizado, según la norma ISO 2533:1975 [45]. Además, es el modelo empleado por la OACI (Organización de Aviación Civil Internacional), si bien esta organización amplia el rango de altitudes modeladas [45].

Los parámetros del aire a la altitud de vuelo dada son enviados a los modelos que lo requieren, como los aerodinámicos o de empuje. Además, son empleados para la simulación de ciertos sensores, como el de presión estática.

6.2.7. Modelos magnetosféricos

Los modelos magnetosféricos caracterizan la magnetosfera terrestre, permitiendo conocer el campo magnético en la posición de la aeronave y posibilitando la simulación de brújulas.

6.2.7.1. *World Magnetic Model* (1 medida)

El modelo magnetosférico que se ha incluido en EFS es el Modelo Magnético Mundial (WMM, *World Magnetic Model*). Aunque este modelo también tiene un bloque en *Aerospace Blockset*, se ha optado por realizar la medida previamente a la simulación, obteniendo el campo magnético en la posición de despegue.

El hecho de tomar una sola medida limita la validez de la simulación a operaciones en las que la aeronave no se aleje del punto de partida. Sin embargo, el coste computacional se reduce enormemente. En cualquier caso, puesto que se dispone del bloque WMM, es posible realizar más medidas si fuera necesario, asumiendo el coste de cómputo.

El Modelo Magnético Mundial, desarrollado por la *National Geospatial-Intelligence Agency* de Estados Unidos y el *British Geological Survey* de Reino Unido, es un modelo estándar para navegación usando el campo magnético de la Tierra [46]. El modelo implementado en *Aerospace Blockset* devuelve el campo magnético terrestre, la declinación, la inclinación, la intensidad horizontal y la intensidad total. Como entradas requiere la fecha y la posición [35].

El vector campo magnético se emplea para recrear la lectura de un magnetómetro, uno de los sensores que emplea Veronte. Como el vector dado por WMM se encuentra en coordenadas NED, es necesario rotar el vector según la actitud de la aeronave, de forma similar al vector gravedad expuesto con anterioridad.

6.3. Simulación de señales para el autopiloto

Embention Flight Simulator, como se pide en el Pliego de condiciones de Embention, simula señales de sensores utilizadas por el bloque Veronte, permitiendo simulaciones SIL. Estas señales son salidas de EFS (véase Tabla 5.1), y se presentan con el formato, dimensiones y unidades requeridos para su funcionamiento directo con Veronte.

Para simular la señal, en primer lugar, se obtiene el dato real que mediría el sensor montado en la aeronave. Posteriormente, se adaptan sus unidades si es necesario y se añade ruido. Las medidas de algunos sensores son vectores que requieren ser rotados según la actitud de la aeronave y según la orientación con la que están montados. En estos casos, se tratan estas rotaciones durante la generación de las señales simuladas. Finalmente, se envían las medidas de los sensores por los puertos de salida.

En la mayoría de casos, Veronte requiere conocer la temperatura del sensor, ya que es capaz de aplicar correcciones a la medida utilizando este dato. EFS incluye un parámetro que indica el incremento de temperatura de cada sensor respecto a la temperatura ambiente.

EFS también permite incluir ruido blanco a las señales. Las medidas de sensores reales son ruidosas y, por tanto, Veronte puede llegar a despreciar lecturas constantes ya que suele ser provocado por un fallo en el sensor o en su conexión. De esta forma, es necesario añadir ruido a las medidas simuladas. El ruido blanco se incluye mediante el bloque *Band-Limited White Noise*, que emplea como parámetro la densidad espectral de potencia (*Power Spectral Density*, PSD).

6.4. Limitaciones

Los modelos expuestos anteriormente hacen de *Embention Flight Simulator* un simulador flexible y modulable. No obstante, es necesario conocer las simplificaciones que realiza para su funcionamiento y las limitaciones asociadas, de forma que se tenga claro en qué condiciones EFS proporciona resultados fiables. Por otro lado, en la Sección 9.1 se plantean algunas mejoras futuras para ampliar el rango de validez del simulador.

Las limitaciones más relevantes de EFS son las siguientes:

- La aeronave se comporta como un sólido rígido y no se consideran efectos aeroelásticos de ningún tipo.
- **Modelos aerodinámicos:** El modelo de aerodinámica por coeficientes, puesto que se trata de una linealización del problema, no es válido en maniobras alejadas del punto de equilibrio. Este modelo tampoco tiene en cuenta otros efectos aerodinámicos como la interacción con la estela de los motores, la entrada en pérdida o la influencia del suelo en la sustentación.
- **Modelos de empuje:** Los modelos de empuje no consideran la velocidad de desplazamiento de la aeronave a la hora de calcular las prestaciones de la hélice. Estos modelos pierden validez para aeronaves que se desplazan a gran velocidad.
- **Modelos atmosféricos:** Aunque la atmósfera ISA es ampliamente utilizada, presenta ciertas simplificaciones que pueden ser relevantes, como ignorar los efectos de la humedad [45].
- **Modelos magnetosféricos:** Como ya se ha comentado, el modelo de magnetosfera que se incluye en EFS reduce el coste de cálculo evitando el uso constante del Modelo Magnético Mundial. En caso de que se emplee este modelo, se debe tener en cuenta que la medida del magnetómetro proporcionada por el simulador no es válida para vuelos de largas distancias.
- **Simulación de los sensores:** Las señales generadas solo incluyen ruido blanco. Algunos sensores, como el magnetómetro, pueden requerir otros modelos de ruido según las condiciones que se quieran simular.
- **Modelización del suelo:** Una limitación importante de EFS es que no modela el suelo ni sus efectos. El simulador utiliza las ventajas de los bloques *enabled* para evitar que la aeronave caiga indefinidamente antes de comenzar la simulación. Aunque no se impide descender más allá del suelo, se ofrece la posibilidad de detener la simulación si se hace contacto (mediante la salida *isLanded*). Además, calcula la energía de impacto, con la intención de evaluar la calidad del aterrizaje.

Capítulo 7

Herramientas del simulador

Además del propio simulador, se han desarrollado una serie de herramientas complementarias que permiten ampliar las funcionalidades de EFS o facilitar su uso.

7.1. Generación de archivos *.kml*

La primera herramienta permite generar archivos *.kml* con la trayectoria de una aeronave después de la simulación. Para ello, emplea la evolución de la posición LLA, permitiendo representar el vuelo de la aeronave en un *software* externo. En la Figura 7.1 se muestra la trayectoria del vuelo simulado de ascenso de un multicóptero, realizada en las inmediaciones del puerto de Alicante y representada en Google Earth.

Figura 7.1: Trayectoria de un vuelo simulado en Google Earth

7.2. Teorías de elemento de pala y de cantidad de movimiento

Muchas de las aeronaves que usan Veronte son propulsadas empleando hélices. Las teorías de elemento de pala y de cantidad de movimiento se emplean para estimar las prestaciones que puede ofrecer una hélice. Cuando se utilizan de forma conjunta, se le conoce como *Blade Element Momentum Theory* o BEMT, por sus siglas [47]. EFS incluye una herramienta que utiliza estas teorías para calcular el empuje y el par que la hélice puede generar, considerando vuelo a punto fijo y, además, crea tablas de consulta para ser empleadas directamente en los modelos de empuje.

7.2.1. Teoría de cantidad de movimiento

La teoría de cantidad de movimiento fue introducida en el año 1865 por William J. M. Rankine [48]. Se basa en considerar toda la hélice como un disco actuador donde se produce un salto de presiones, empleando las siguientes hipótesis y simplificaciones [49]:

1. Se considera fluido no viscoso e incompresible.
2. El movimiento, en vuelo axial, es en una sola dimensión.
3. El movimiento es casi estacionario.
4. El movimiento en la estela del rotor es vertical, no tiene rotación.
5. No se aplica ninguna fuerza externa sobre el fluido.
6. Todo el plano del rotor presenta velocidad inducida uniforme.

A partir de las ecuaciones de conservación de la masa, de la cantidad de movimiento y de la energía, planteadas en el volumen de control de la Figura 7.2 y teniendo en cuenta las simplificaciones anteriores, se obtiene el empuje total generado por el disco actuador (T, Ecuación 7.1), el gasto másico total que trasiega el disco ($\dot{m}$, Ecuación 7.2) y la potencia ideal (P_{id}, Ecuación 7.3), útil para conocer los límites teóricos del rotor [49]. Las ecuaciones incluyen la velocidad vertical de la aeronave, v_z, aunque EFS no tiene modelos desarrollado que contemplen el desplazamiento de la aeronave en el cálculo del empuje.

$$T = 2\rho S(v_z + v_i)v_i \tag{7.1}$$

$$\dot{m} = \rho S(v_z + v_i) \tag{7.2}$$

$$P_{id} = 2\rho S(v_z + v_i)^2 v_i \tag{7.3}$$

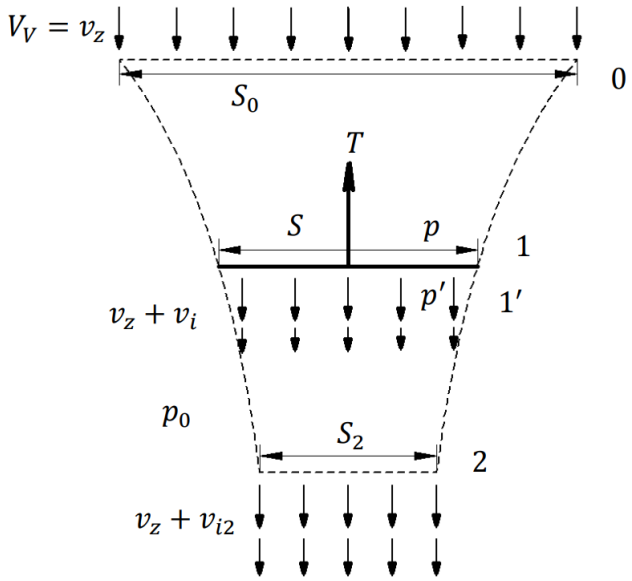

Figura 7.2: Volumen de control para la teoría de cantidad de movimiento [49]

7.2.2. Teoría del elemento de pala

Por otro lado, la teoría de elemento de pala fue introducida por William Froude en 1878 [48]. En ella, se divide la pala de la hélice en secciones, llamadas elementos de pala, y se realiza un estudio bidimensional en cada elemento, tratándolos como perfiles aerodinámicos. De esta forma, se tiene en cuenta la geometría de la pala, a diferencia de la teoría de cantidad de movimiento.

En general, las hipótesis y simplificaciones asumidas en el empleo de la teoría de elemento de pala son las siguientes [49]:

1. Los elementos de pala son tratados de forma bidimensional.
2. La velocidad inducida, se considera una variación del ángulo de ataque del perfil.
3. El término de velocidad radial, es decir, el que tiene la misma dirección que el eje de la pala, se desprecia.
4. La componente normal de la velocidad, con dirección normal al plano de rotación, es mucho menor que la velocidad tangencial. La velocidad tangencial es la que percibe cada perfil debido a la rotación de la pala.

5. Cada elemento de pala viene caracterizado por su coeficiente de sustentación y de resistencia, c_l y c_d, respectivamente. El coeficiente de sustentación tiene dependencia lineal con el ángulo de ataque, mientras que el de resistencia tiene dependencia cuadrática.

6. La sustentación generada por cada elemento es mucho mayor que la resistencia ($dL \gg dD$).

Si se considera un elemento de pala diferencial, como el presentado en la Figura 7.3, se aprecia una velocidad percibida por el perfil, U, que generará un diferencial de sustentación (dL) y de resistencia (dD). Si se rotan las fuerzas a los ejes del rotor, se tienen diferenciales de empuje (dT) y diferenciales de fuerza tangencial (dF_T) que, multiplicados por la distancia del elemento al eje de giro del rotor, permiten conocer el par o torque generado por el diferencial.

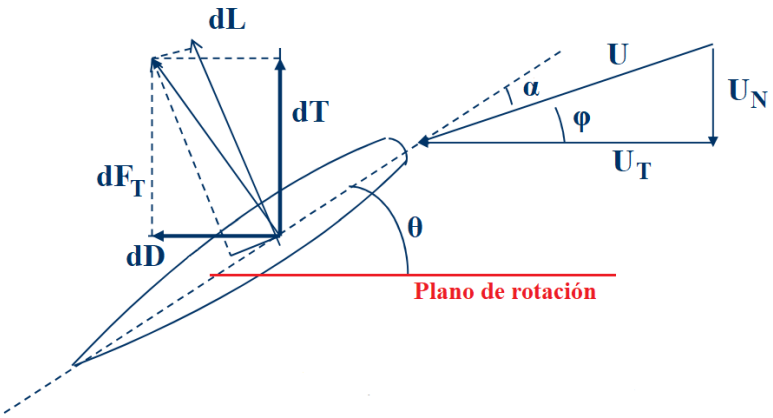

Figura 7.3: Fuerzas y velocidades en el elemento de pala [50]

A partir de la hipótesis 4, se sacan las conclusiones expuestas en la Ecuación 7.4, siendo ϕ el ángulo formado por las componentes tangencial y normal de la velocidad, llamado ángulo de ataque inducido [49].

$$\begin{aligned} |U_T| \ll |U_N| \Rightarrow U \approx U_T \\ \phi = \arctan U_N/U_T \approx U_N/U_T \\ \phi \ll 1 \Rightarrow \cos\phi \approx 1,\ \sin\phi \approx \phi \end{aligned} \tag{7.4}$$

Por otro lado, sabiendo que la velocidad tangencial se debe a la rotación de la pala y que, en vuelo en punto fijo, la velocidad normal será la velocidad inducida por el rotor, se puede obtener la velocidad percibida por el elemento de pala, como muestra la Ecuación 7.5, llamando r a la distancia al eje de rotación del elemento.

$$U = \sqrt{U_T^2 + U_N^2} \approx \sqrt{(\Omega\ r)^2 + v_i^2} \tag{7.5}$$

Otro ángulo relevante que se aprecia en la Figura 7.3 es el ángulo de paso, θ. Cabe destacar que la herramienta BEMT implementada en EFS permite proporcionar el paso de cada elemento, teniendo en cuenta así la torsión geométrica con la que está construida la pala.

Con todos estos datos, se calculan las fuerzas aerodinámicas del elemento, como expresa la Ecuación 7.6. La superficie del elemento está determinada por su cuerda, c, y su espesor, dr. A la hora de implementar las ecuaciones en el simulador, dr depende del número de elementos en los que se quiera dividir la pala. Por otro lado, siguiendo la hipótesis 5, los coeficientes aerodinámicos del perfil son los mostrados en la Ecuación 7.7.

$$\begin{aligned} dL &= \frac{1}{2}\rho U^2 c c_l \; dr \\ dD &= \frac{1}{2}\rho U^2 c c_d \; dr \end{aligned} \tag{7.6}$$

$$\begin{aligned} c_l &= c_{l\alpha} \cdot \alpha \\ c_d &= c_{d0} + c_{d\alpha} \; \alpha + c_{d\alpha^2} \; \alpha^2 \end{aligned} \tag{7.7}$$

Finalmente, se obtiene la fuerza de empuje (dT) y la fuerza tangencial (dF_T) del elemento, que son las componentes de la fuerza aerodinámica representadas en los ejes del rotor. La transformación se realiza se acuerdo con la Ecuación 7.8 y, aplicando las hipótesis 4 y 6, se obtiene la versión simplificada de la Ecuación 7.9 [49].

$$\begin{aligned} dT &= dL \; \cos\phi - dD \; \sin\phi \\ dF_T &= dL \; \sin\phi + dD \; \cos\phi \end{aligned} \tag{7.8}$$

$$\begin{aligned} dT &\approx dL - dD \; \phi \approx dL \\ dF_T &\approx dL \; \phi + dD \end{aligned} \tag{7.9}$$

7.2.3. Teoría del momento del elemento de pala

Para usar estas dos teorías de forma conjunta (BEMT), se obtiene la velocidad inducida por el rotor mediante la teoría de cantidad de movimiento y se aplica esta velocidad en el elemento de pala. La velocidad inducida se considera como un cambio en el ángulo de ataque, ya que hace variar el ángulo de ataque inducido: $\alpha = \theta + \phi$.

Para obtener la velocidad inducida y, por tanto, el ángulo de ataque inducido, utilizando la BEMT, se iguala el empuje que realiza un diferencial de disco actuador [49], es decir, un anillo diferencial, según ambas teorías, siguiendo la Ecuación 7.10. En la teoría de cantidad de movimiento (Ecuación 7.1) se debe cambiar la superficie del disco actuador, S, por la superficie de un anillo diferencial, $2r\,dr\ +\ dr^2$), mientras que el empuje del elemento de pala (Ecuación 7.8) se debe multiplicar por el número de palas.

$$dT_{CM} = dT_{EP} \tag{7.10}$$

El resultado de la Ecuación 7.10 es la velocidad inducida de cada elemento de pala. Sin embargo, esta teoría no tiene en cuenta que la pala es finita, por lo que se aplica una corrección para considerar que la punta de la pala no produce sustentación ni par, llamada *Prandtl Tip Loss Correction*. La corrección se aplica sobre la velocidad inducida, como indica la Ecuación 7.11, con c_0 siendo la cuerda al comienzo de la pala; c_t, la cuerda en la punta y B, el factor de corrección de Prandtl [49].

$$\begin{aligned} v_{i\ \text{Tip correction}} &= v_i/B \\ B &= 1 - \frac{c_0\ (1+0{,}7c_t/c_0)}{1{,}5\ R} \end{aligned} \tag{7.11}$$

Finalmente, conociendo la velocidad inducida corregida, se calcula nuevamente ϕ y se aplica en las Ecuaciones 7.8. Teniendo el empuje y la fuerza tangencial de cada elemento, se obtiene el empuje y el par total del rotor sumando la contribución de todos los diferenciales y considerando el número de palas (n), como indican las Ecuaciones 7.12. Con estos resultados, la herramienta implementada en EFS puede estimar las prestaciones de la hélice para diferentes velocidades de giro, ángulos de paso y densidades, generando las tablas de consulta requeridas por los modelos.

$$T = n\ \sum dT_i \tag{7.12}$$

$$Q = n\ \sum dF_{T\,i}\ r_i \tag{7.13}$$

7.3. Visualización de la geometría de la aeronave

Se han desarrollado también herramientas de representación de la geometría de la aeronave. Aunque los modelos incluidos en EFS no requieren información detallada sobre las superficies aerodinámicas, más allá de las empleadas en la definición de los coeficientes, sí es posible especificar la posición de alas y estabilizadores, así como de superficies de control.

De esta forma, la herramienta de visualización de la geometría muestra un esquema completo de la aeronave, incluyendo los motores y las superficies aerodinámicas, teniendo

en cuenta sus ángulos de diedro y de flecha, como se aprecia en la Figura 7.4.

Como se detalla en la Sección 9.1, donde se exponen futuros desarrollos planteados para *Embention Flight Simulator*, esta funcionalidad puede ser un punto de partida para futuras herramientas de estimación de coeficientes aerodinámicos según la geometría y configuración de la aeronave.

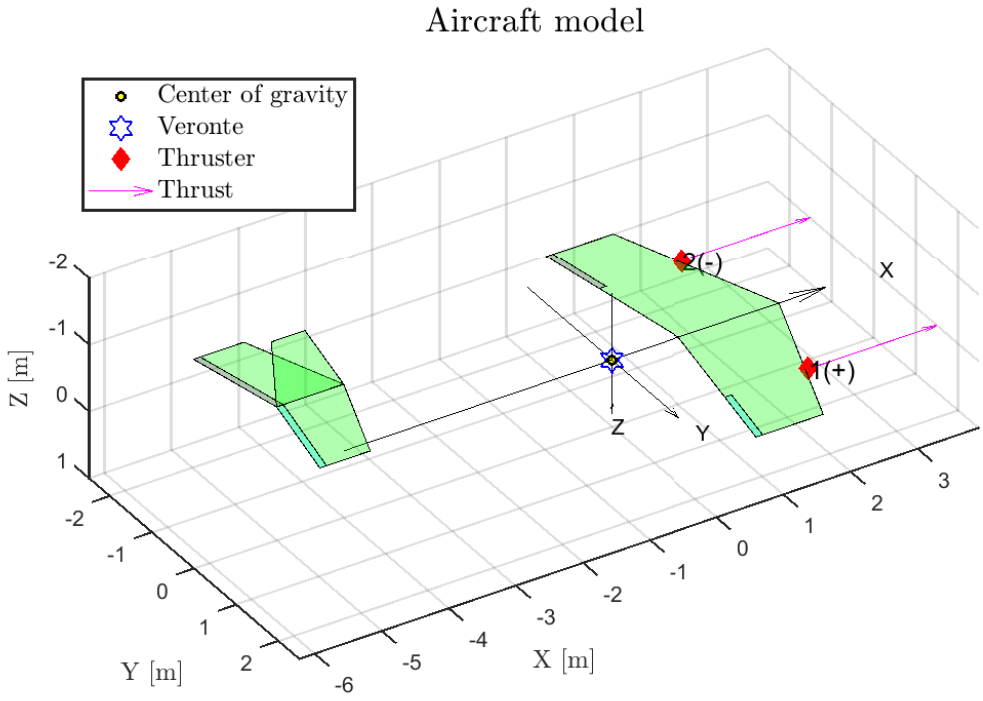

Figura 7.4: Visualizador de superficies aerodinámicas

7.4. Generación de resúmenes

La herramienta de generación de resúmenes crea archivos *.txt* con información sobre el simulador, almacenada en el *workspace* de MATLAB. Permite gestionar las simulaciones realizadas y, añadiendo los parámetros necesarios, puede servir como herramienta de guardado y de carga. La posibilidad de guardar aeronaves ya modeladas o resultados de simulaciones permite aumentar la flexibilidad del simulador, a la vez que automatizar la ejecución de diferentes simulaciones para validación o testeo.

Capítulo 8

Resultados obtenidos en casos prácticos

Una vez conocidos los principios de funcionamiento de *Embention Flight Simulator*, se van a exponer una serie de casos prácticos y experimentos que han servido para la validación del simulador.

8.1. Caída libre

El primer experimento se trata de una caída libre, con el que se puede validar el funcionamiento del simulador en su conjunto. La caída se produce desde una altitud de 1000 m, en un lugar con una elevación de 300 m sobre el nivel del mar. En la Figura 8.1 muestra el resumen de la configuración que ofrece el lanzador de EFS cuando se ejecuta.

Command Window

```
EMBENTION FLIGHT SIMULATOR LAUNCHER
Embention, 2023

AIRCRAFT MODELS
Aerodynamic model: Linear drag model (no lift)
Thrust model: No thrust
Mass-Inertia model: Constant Mass-Inertia model

ENVIRONMENT MODELS
Atmosphere model: International Standard Atmosphere (ISA) model
Magnetosphere model: World Magnetic Model (1 measure)
Wind model: No wind

ACTUATOR MODELS
Actuators model: Feedthrough

Launching simulator...
Embention Flight Simulator is ready
```

Figura 8.1: Resumen de los modelos empleados en la caída libre

El primer resultado está representado en la Figura 8.2, donde se muestra la evolución de la altura durante la caída libre. Con el modelo de *drag* lineal, lo esperado es que el objeto alcance una velocidad terminal de caída, como indica la ley de Stokes [38] y como se confirma en la representación de la velocidad de descenso (Figura 8.3).

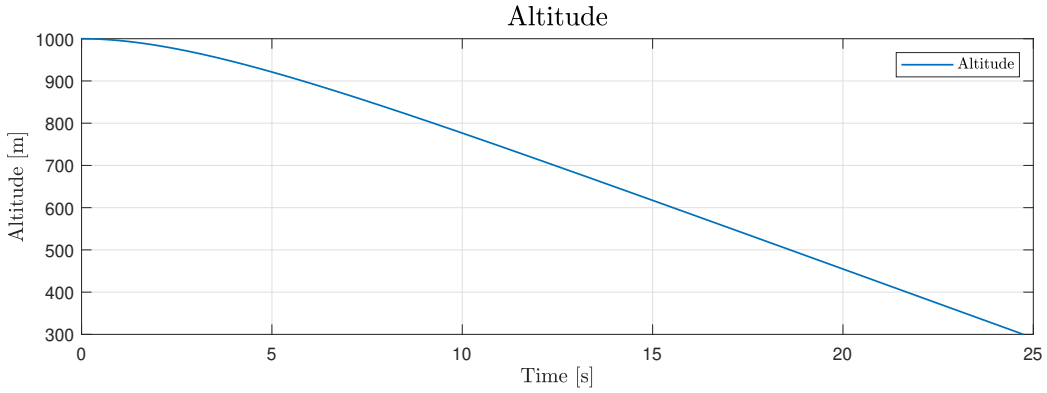

Figura 8.2: Altitud del objeto en caída libre

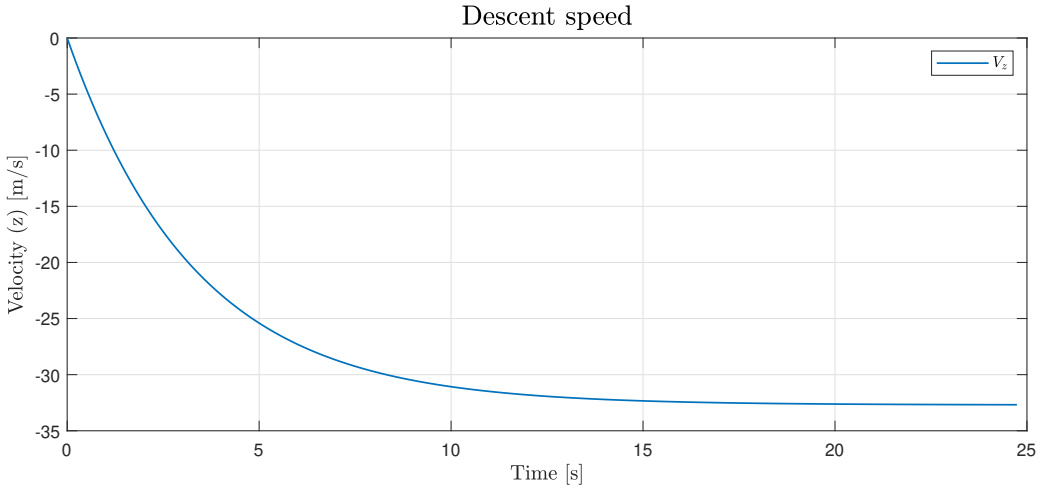

Figura 8.3: Velocidad de descenso del objeto en caída libre

Este experimento también permite validar el correcto funcionamiento de los sensores simulados. En la Figura 8.4 se puede apreciar la variación de la presión estática durante la caída. La presión aumenta al mismo ritmo que la altitud baja. También se puede observar que la medida del sensor es ruidosa.

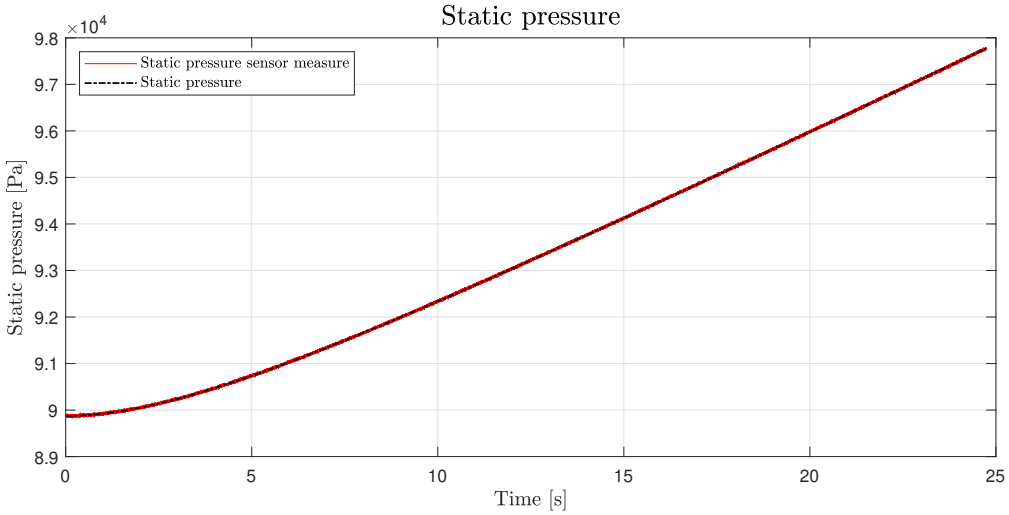

Figura 8.4: Presión estática medida durante la caída libre

Cuando el objeto alcanza el suelo, con una altitud de 300 m, la simulación se detiene mostrando un aviso en pantalla. Además, la energía de impacto es calculada siguiendo la Ecuación 8.1 (energía cinética) y se almacena en los resultados de la simulación. En este caso, puesto que la masa del objeto es de 10 kg, la energía de impacto calculada es de 5340 J.

$$E_c = \frac{1}{2} m v_z^2 \tag{8.1}$$

8.2. Hexacóptero de motores eléctricos

Embention dispone de algunas aeronaves de prueba donde valida sus desarrollos. Una de estas aeronaves es un hexacóptero eléctrico, que es capaz de volar de forma autónoma controlado por Veronte. El objetivo de este experimento es modelar el hexacóptero de forma que pueda volar en el simulador empleando la configuración de Veronte real.

El hexacóptero se muestra en la Figura 8.5. Para su modelización, se tomaron medidas y pesos de sus diferentes componentes para aplicarlos a los modelos seleccionados, recogidos en la Tabla 8.1.

Tabla 8.1: Modelos empleados para el hexacóptero

	Modelo empleado
Modelo aerodinámico	*Drag* simple
Modelo de empuje	Empuje tabulado
Modelo de masa e inercia	Masa e inercia constante

El primer paso en la toma de medidas es estimar la posición del centro de gravedad general del hexacóptero, ya que es el origen de coordenadas del sistema fijo en el cuerpo. Posteriormente, se procede a medir la posición y el peso de los motores y de los componentes que forman la aeronave (batería, autopiloto Veronte, chasis, entre otros). Los datos medidos se muestran en la Tabla 8.2.

Componente	**Masa [kg]**	**Posición [m, m, m]**
Chasis	1,976	(0, 0, 0)
Batería	3 kg	(−0,16, 0, 0,06)
Contrapeso	0,6	(0,32, 0, −0,04)
Veronte	0,197	(0,05, 0, −0,07)
Motor 1	0,154	(0,3897, 0,225, −0,1)
Motor 2	0,154	(0, 0,45, −0,1)
Motor 3	0,154	(−0,3897, 0,225, −0,1)
Motor 4	0,154	(−0,3897, −0,225, −0,1)
Motor 5	0,154	(0, −0,45, −0,1)
Motor 6	0,154	(0,3897, −0,225, −0,1)
Total	6,6970	

Tabla 8.2: Medidas tomadas del hexacóptero

Figura 8.5: Hexacóptero con motores eléctricos modelado

Para obtener la matriz de inercia del hexacóptero, se considera cada componente como una masa puntual y se obtiene su inercia respecto al centro de gravedad de la aeronave, aplicando las Ecuaciones 8.2.

$$\begin{aligned} I_{xx} &= m\ (y^2 + z^2) \\ I_{yy} &= m\ (x^2 + z^2) \\ I_{zz} &= m\ (x^2 + y^2) \\ I_{xy} = I_{yx} &= -m\ x\ y \\ I_{xz} = I_{zx} &= -m\ x\ z \\ I_{yz} = I_{zy} &= -m\ y\ z \end{aligned} \tag{8.2}$$

Con esta información, se puede realizar el modelo del hexacóptero en EFS, teniendo como resultado la aeronave mostrada en la Figura 8.6. En las primeras simulaciones, la aeronave era capaz de despegar pero no de mantener el vuelo en punto fijo siguiendo las instrucciones de Veronte. Con unos pequeños ajustes al modelo de empuje y a la matriz de inercia, realizados de forma experimental, se obtuvo un modelo satisfactorio.

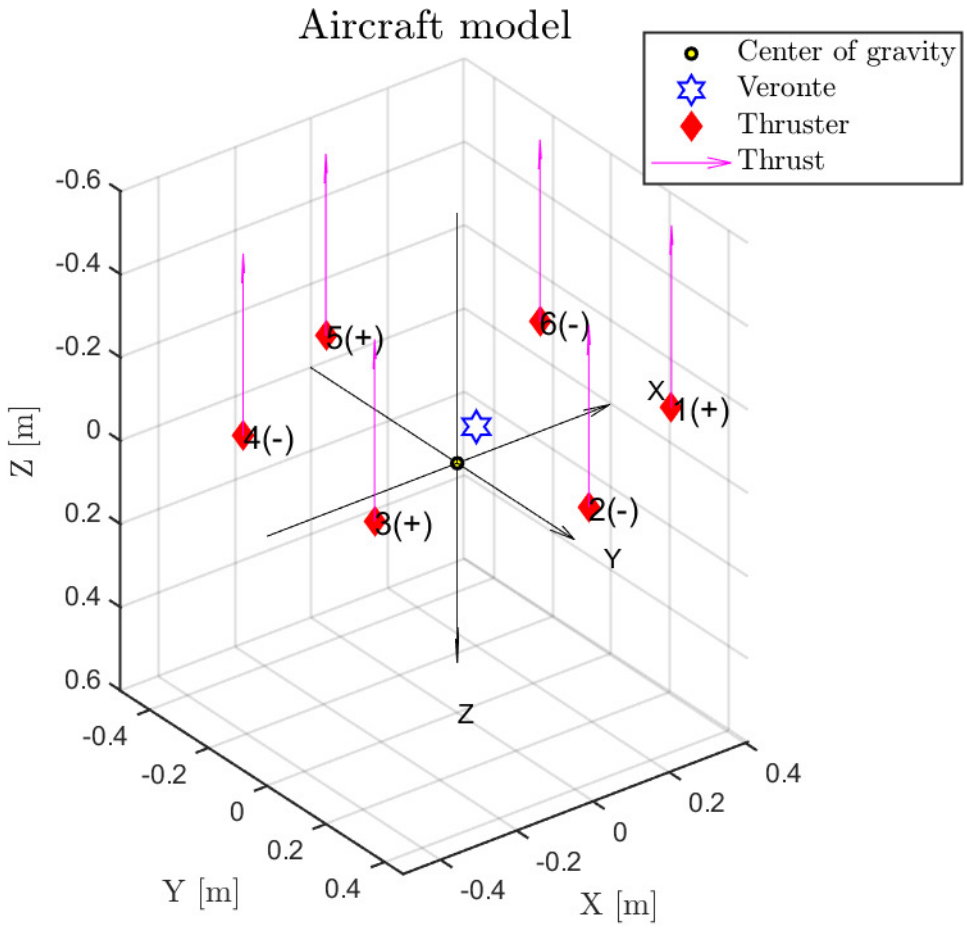

Figura 8.6: Modelo de la geometría del hexacóptero en EFS

En definitiva, el modelo de hexacóptero desarrollado en EFS permite replicar el comportamiento de la aeronave real, validando el trabajo realizado.

8.3. Cuadricóptero con motor de combustión y colectivo variable

Después de validar el simulador con una aeronave de Embention se realizó el modelo de la aeronave de un cliente. En este caso, se trata de un cuadricóptero con cuatro hélices movidas por un motor de combustión interna y controlado mediante el cambio del paso colectivo de cada rotor.

El punto de partida, dadas las particularidades de este caso y al tratarse de un cliente, son los resultados de una serie de experimentos realizados. El objetivo, por tanto, es crear

un modelo que proporcione los mismos resultados que los experimentos con el cuadricóptero real. De esta forma, se puede validar el funcionamiento de Veronte como sistema de control de la aeronave del cliente, sin necesidad de disponer físicamente de la misma.

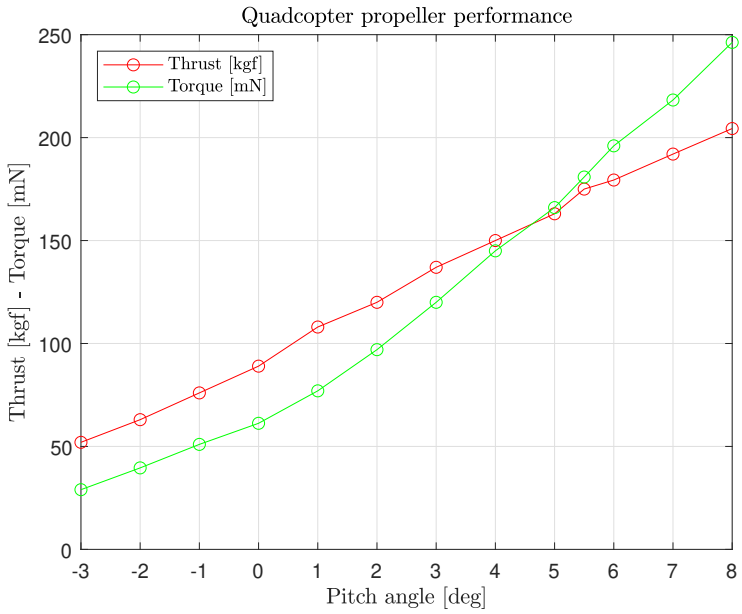

Figura 8.7: Datos experimentales del empuje y par del cuadricóptero, $\Omega = 1900$ rpm, $\rho = 1{,}225$ kg/m^3

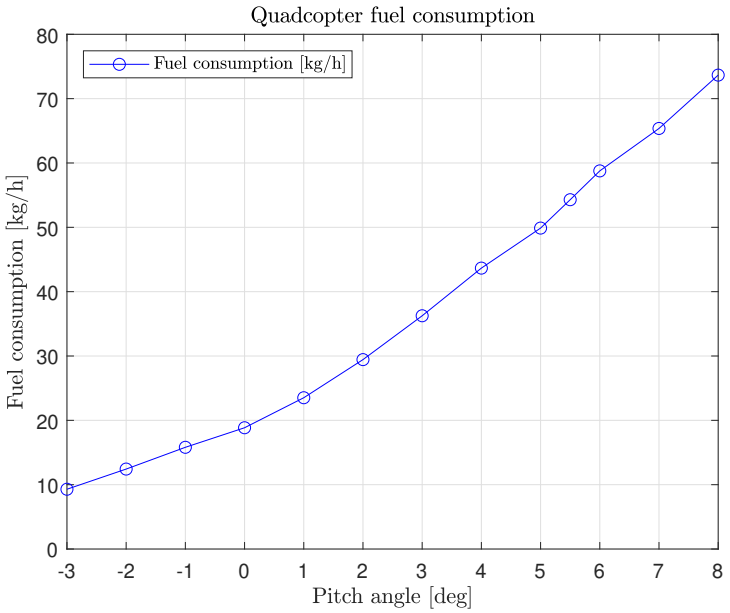

Figura 8.8: Datos experimentales de consumo del cuadricóptero, $\Omega = 1900$ rpm, $\rho = 1{,}225$ kg/m^3

Por un lado, en la Figura 8.7, se tiene el par y el empuje proporcionado por la hélice

según el ángulo de paso del colectivo, manteniendo las revoluciones por minuto del rotor en 1900 rpm y la densidad a nivel del mar. Por otro lado, en la Figura 8.8, se recoge la relación entre ángulo de paso y consumo de combustible, con las mismas condiciones que el caso anterior. Estos son los resultados a los que se debe ajustar el modelo del simulador.

Para llevar a cabo la modelización del cuadricóptero, se escogen los modelos de la Tabla 8.3. Empleando la implementación de la teoría de elemento de pala y cantidad de movimiento, descrita en la Sección 7.2, se ajustan los parámetros de la hélice hasta obtener los resultados mostrados en las Figuras 8.9 y 8.10. Como medida cuantitativa del error cometido al ajustar el modelo a los datos se proporciona el error estándar de regresión, $\hat{\sigma}$.

Tabla 8.3: Modelos empleados para el cuadricóptero

	Modelo empleado
Modelo aerodinámico	*Drag* simple
Modelo de empuje	Empuje tabulado con paso colectivo variable
Modelo de masa e inercia	Consumo de combustible

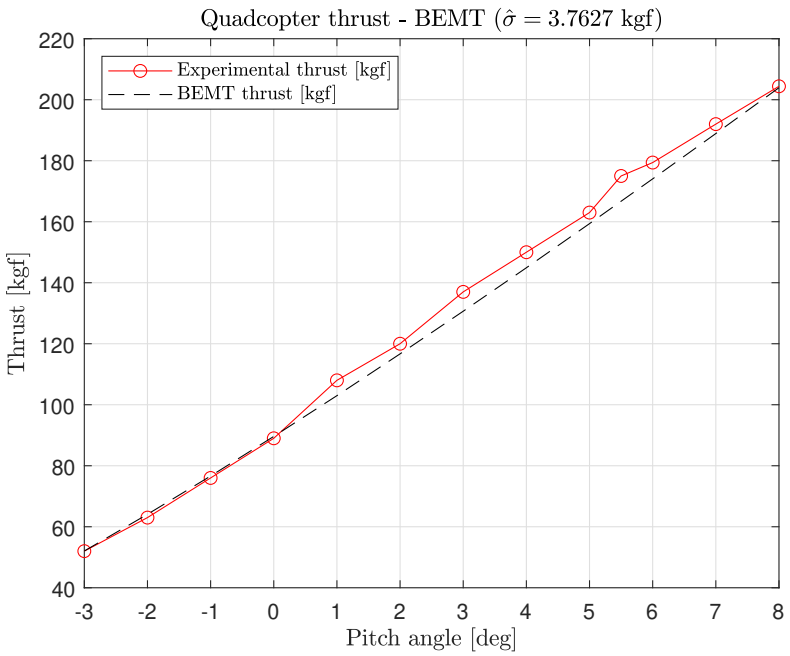

Figura 8.9: Ajuste mediante BEMT del empuje del cuadricóptero

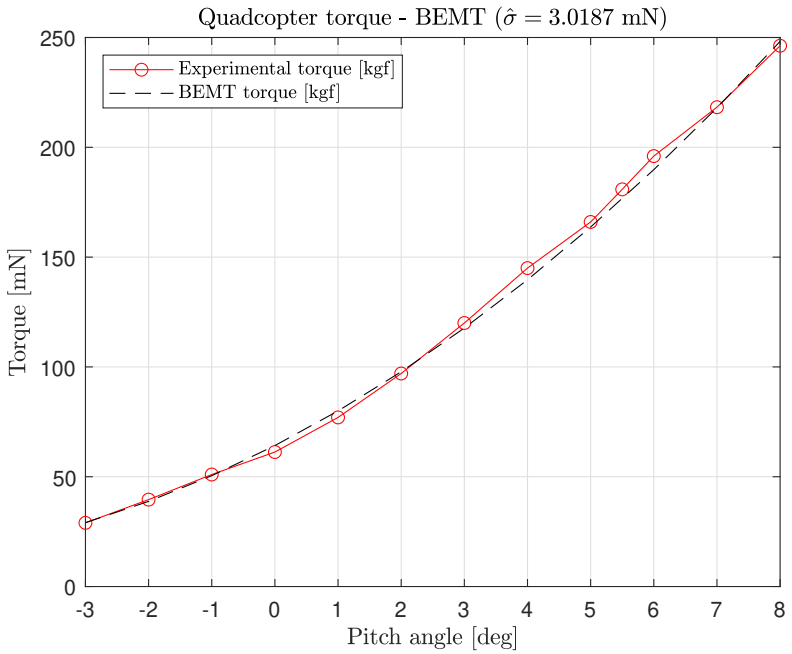

Figura 8.10: Ajuste mediante BEMT del par del cuadricóptero

Con la herramienta BEMT ajustada para la densidad y revoluciones del rotor de los resultados experimentales, es sencillo crear tablas de consulta para el modelo de empuje con un mayor rango de altitudes y velocidades de rotación útiles. Estas tablas están representadas en las Figuras 8.11 y 8.12, y recrean el comportamiento de los cuatro rotores de la aeronave durante toda la operación.

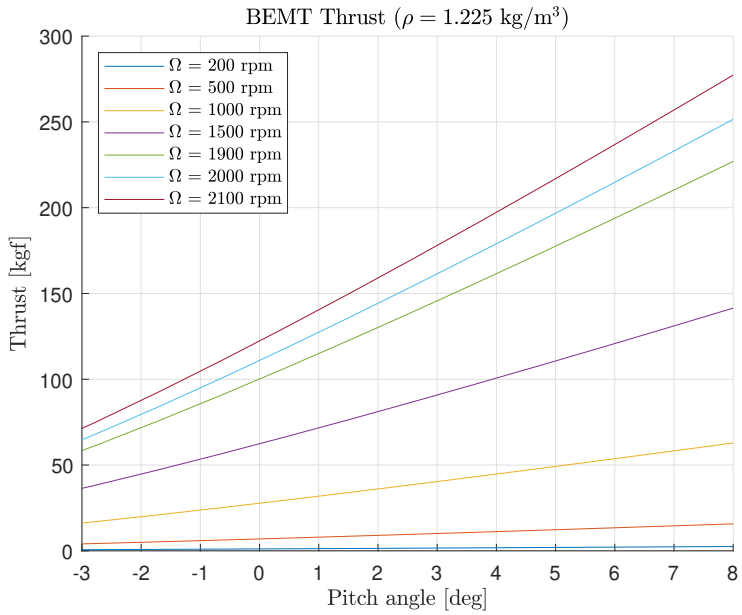

Figura 8.11: Extensión del modelo BEMT del cuadricóptero para varias Ω

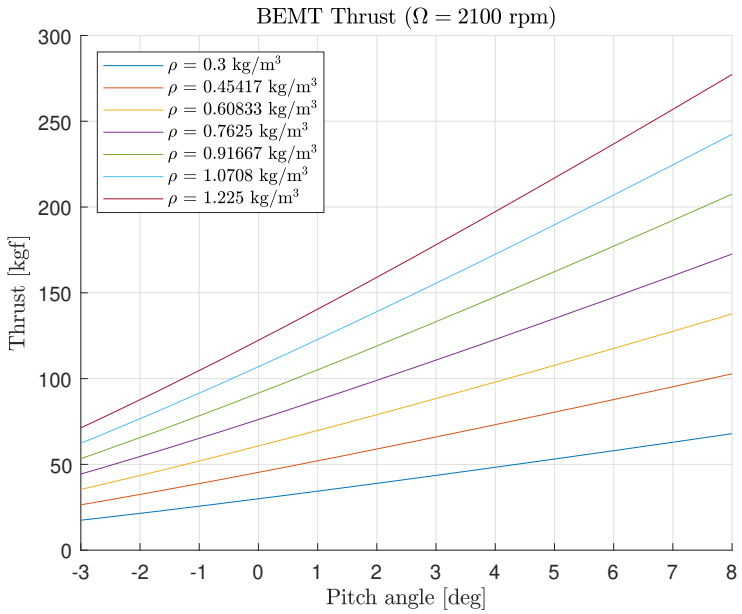

Figura 8.12: Extensión del modelo BEMT del cuadricóptero para varios ρ

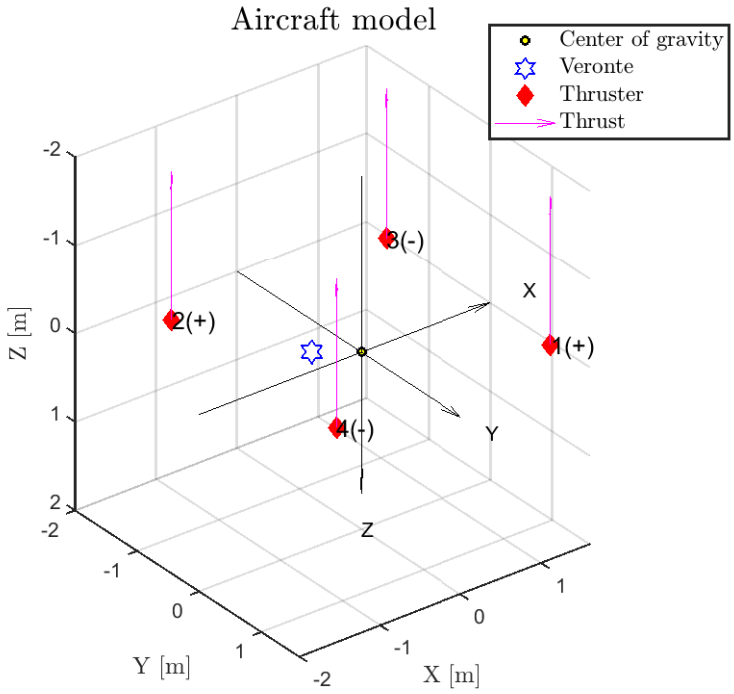

Figura 8.13: Modelo de la geometría del cuadricóptero

Respecto al gasto de fuel, con los datos de la Figura 8.8 y conociendo el empuje generado, se obtiene el consumo de combustible por empuje (TSFC), que se aplica en el modelo de consumo de combustible. Cabe destacar que también se facilita la masa total, la masa

en vacío y las matrices de inercia en ambos casos. De esta forma, empleando tablas de consulta que relacionan un peso de la aeronave con una matriz de inercia, se tiene completo el modelo de masa e inercia.

Por último, el modelo de resistencia aerodinámica se ajusta experimentalmente, realizando simulaciones hasta conseguir el comportamiento deseado. Con todo esto, se tiene el modelo del cuadricóptero, representado en EFS por la Figura 8.13.

Capítulo 9

Conclusiones y trabajos futuros

El objetivo de este capítulo es plantear el futuro de *Embention Flight Simulator* basado en la experiencia adquirida durante su desarrollo y las limitaciones que presenta la versión hasta ahora descrita. Habiendo expuesto el estado actual del simulador y las expectativas de futuro, se obtendrán conclusiones.

9.1. Desarrollo futuro de *Embention Flight Simulator*

Puesto que EFS está concebido como un Trabajo de Fin de Máster, su extensión es limitada. Sin embargo, como proyecto empresarial puede continuar su desarrollo, implementando una larga lista de ideas planteadas que se presenta a continuación:

- EFS está desarrollado principalmente en MATLAB y Simulink 2018a. El simulador podría aprovechar nuevas funcionalidades añadidas si se realizara una migración a la última versión de MATLAB. En particular, los parámetros que utiliza EFS pueden ser organizados de forma más efectiva y tolerante a fallos.
- Las herramientas de guardado y carga de modelos pueden ser perfeccionadas. Aunque una vez realizado el modelo se puede guardar el *script* de MATLAB con toda la información, resulta interesante guardarlo en otro formato externo.
- Aunque EFS se plantea en este trabajo como un proyecto de Simulink, en el futuro puede ser interesante construir una aplicación de Simulink a su alrededor, mediante MATLAB *App Designer*. Este tipo de aplicaciones presenta grandes ventajas, destacando la capacidad de añadir interfaz gráfica al simulador, así como distribuir el simulador a usuarios que no dispongan de MATLAB.
- Para poder validar la etapa de aterrizaje de forma correcta, es necesario modelar el suelo. Para ello, se debe aplicar una fuerza normal a la aeronave cuando está en

contacto con el suelo, así como la fuerza de impacto. También es importante incluir algún modelo de efecto suelo, tanto para aeronaves de ala fija como multicópteros.

- EFS tiene potencial para simular el vuelo de más tipos de vehículos. Por ejemplo, realizando modificaciones en los modelos actuales se podría caracterizar de forma simplificada el vuelo de un helicóptero, añadiendo el control cíclico. Por otro lado, la inclusión de motores cohete y de modelos de turbofán ampliaría el número de aeronaves simuladas en EFS.

- La modelización de los sensores puede ser mejorada si se incluye alguna entrada al simulador que controle su funcionamiento durante el vuelo. Se podrían simular fallos en los sensores durante el vuelo o pérdidas de cobertura. También se pueden incluir otros modelos de ruido, más allá del blanco, o sesgos en la medida.

- Se podría desarrollar un modelo de aerodinámica basado en tablas de consulta. Una vez determinadas las entradas relevantes del modelo se podrían construir *Lookup Tables* para considerar efectos no lineales o entrada en pérdida.

- *Aerospace Blockset* dispone de multitud de modelos de ráfaga y turbulencia que son interesantes para validar la respuesta de Veronte ante dichas situaciones.

- Empleando la capacidad de Veronte de comunicarse con un ordenador, se podrían realizar simulaciones *Hardware In the Loop*, de forma que un Veronte real controle una aeronave simulada en EFS.

- Se podrían incluir modelos de consumo eléctrico de los motores y de baterías, de forma que, igual que el modelo de empuje se detiene cuando se termina el combustible, lo haga cuando se agoten las baterías en las aeronaves eléctricas.

- Algunas herramientas de estimación de coeficientes o de caracterización de aeronaves de ala fija pueden ser implementadas. Por un lado, se plantea la utilización del método de la malla de torbellinos (*Vortex-Lattice Method*) para obtener el comportamiento de una aeronave conociendo únicamente su geometría, en conjunto con la herramienta de visualización de geometrías (véase [51]). El *software* TORNADO está desarrollado en MATLAB y tiene licencia GNU [52]. Por otro lado, también es interesante implementar el bloque *Digital DATCOM* de *Aerospace Blockset*, que se trata de una herramienta muy potente para obtener derivadas de estabilidad [53]. Estas dos herramientas ampliarían en gran medida las capacidades de EFS a la hora de simular aeronaves de ala fija.

9.2. Conclusiones

Embention Flight Simulator demuestra que puede replicar el comportamiento de aeronaves reales, que puede comunicarse de forma efectiva con el bloque Veronte de Simulink y que permite realizar modelos de forma sencilla gracias a las herramientas desarrolladas

y los modelos implementados.

Comparando EFS con los trabajos descritos en la Sección 2.2, se tiene un simulador mucho más flexible en el tipo de aeronaves que se pueden simular. Además, agrega herramientas de modelización que este tipo de simuladores no incluye, como las implementaciones de la *Blade Element Momentum Theory*, entre otras.

Otro punto a favor de EFS es la simulación de los sensores. Los simuladores mencionados no están diseñados para trabajar con un autopiloto externo, por lo que no incluyen esta funcionalidad.

Respecto a simuladores más avanzados, como X-Plane, EFS presenta una mejor compatibilidad a la hora de realizar simulaciones SIL con Veronte. En cambio, X-Plane tiene modelizado el suelo y el impacto con el suelo.

Embention Flight Simulator ha sido desarrollado de forma modular, permitiendo una expansión sencilla de sus funcionalidades, de forma que Embention puede emplearlo para simular cualquier situación que se necesite, facilitando la tarea de validación y la realización de los test requeridos para las diferentes certificaciones que posee.

Finalmente, teniendo en cuenta que se trata de un Trabajo de Fin de Máster desarrollado en conjunto con Embention, se puede concluir que EFS cumple con los objetivos propuestos por la empresa. El simulador cuenta con la aprobación de Embention, llegando incluso a ser usada con clientes reales.

Aplicando los desarrollos propuestos en este capítulo, *Embention Flight Simulator* puede potenciar enormemente las herramientas de simulación *Software In the Loop* que ofrece Embention.

Bibliografía

[1] Embention. *Embention Web Page*. Recuperado de https://www.embention.com/. Accedido el 8 de mayo de 2023.

[2] Embention. *Autopilot SIL Simulator*. Recuperado de https://www.embention.com/product/drone-flight-simulator-autopilot-sil/. Accedido el 8 de mayo de 2023.

[3] Valls, Joan. ITI, *Investigate To Innovate. Verificación de sistemas críticos mediante Software-in-the-loop*. Recueprado de https://www.iti.es/blog/verificacion-de-sistemas-criticos-mediante-software-in-the-loop/. Accedido el 8 de mayo de 2023.

[4] Wikipedia contributors. Wikipedia, *The Free Encyclopedia. Flight simulator*. Recuperado de https://en.wikipedia.org/w/index.php?title=Flight_simulator&oldid=1152968822. Accedido el 8 de mayo de 2023.

[5] Baarspul, M. Delft University of Technology. *Lecture Notes on Flight Simulation Techniques* (1989). Recuperado de http://resolver.tudelft.nl/uuid:269fa44e-0b09-4fc5-9260-83b6affb7cdd

[6] Wikimedia Commons. *File:Antoinette tonno.jpg*. Recuperado de https://commons.wikimedia.org/wiki/File:Antoinette_tonno.jpg. Dominio público.

[7] Wikipedia *contributors*. Wikipedia, *The Free Encyclopedia. Virtual airline (hobby)*. Recuperado de https://en.wikipedia.org/w/index.php?title=Virtual_airline_(hobby)&oldid=1147380331. Accedido el 9 de mayo de 2023.

[8] Parsons, Dan. Aviation Today. *Full Flight Simulators Incorporate VR for Next Generation of Pilots* (2019). Recuperado de https://www.aviationtoday.com/2019/08/01/training-brain-mind/#:~:text=FAA%2Dqualified%20full%20flight%20simulators,cost%20up%20to%20%241%20million. Accedido el 9 de mayo de 2023.

[9] Wikipedia contributors. Wikipedia, *The Free Encyclopedia. Microsoft Flight Simulator*. Recueprado de https://en.wikipedia.org/w/index.php?title=Microsoft_Flight_Simulator&oldid=1148624113. Accedido el 11 de mayo de 2023.

[10] Microsoft. Wikipedia, *The Free Encyclopedia. Microsoft Flight Simulator logo (2020).png*, Recuperado de https://en.wikipedia.org/w/index.php?curid=61017148. *Fair use*.

[11] Microsoft Flight Simulator, SDK Documentation. *Flight Model Physics*. Recuperado de https://docs.flightsimulator.com/html/Samples_And_Tutorials/Primers/Flight_Model_Physics.htm. Accedido el 11 de mayo de 2023.

[12] Microsoft Flight Simulator, SDK Documentation. *Basic Aerodynamics*. Recuperado de https://docs.flightsimulator.com/html/mergedProjects/How_To_Make_An_Aircraft/Contents/Files/Flight_Model/Basic_Aerodynamics.htm. Accedido el 11 de mayo de 2023.

[13] Microsoft Flight Simulator, SDK Documentation. *Defining a Flight Model*. Recuperado de https://docs.flightsimulator.com/html/Samples_And_Tutorials/Tutorials/Defining_A_Flight_Model.htm. Accedido el 11 de mayo de 2023.

[14] Wikipedia contributors. Wikipedia, *The Free Encyclopedia. X-Plane (simulator)*. Recuperado de https://en.wikipedia.org/w/index.php?title=X-Plane_(simulator)&oldid=1153422811. Accedido el 15 de mayo de 2023.

[15] X-Plane. *How X-Plane works*. Recuperado de https://www.x-plane.com/desktop/how-x-plane-works/. Accedido el 15 de mayo de 2023.

[16] Gibiansky, Andrew. *Quadcopter Dynamics and Simulation* (2012). Recuperado de https://andrew.gibiansky.com/blog/physics/quadcopter-dynamics/. Accedido el 18 de julio de 2023.

[17] Huixian Wang; Dongli Ma. Proceedings of the 2nd International Conference on Computer Engineering, Information Science and Application Technology (ICCIA 2017). *Aircraft 6-DOF Modular Modeling Based on MATLAB Simulink* (2016). Recuperado de (DOI) 10.2991/iccia-17.2017.176

[18] Abid Sulficar; Hartkrishnan Suresh; Aravind Varma; Arjun Radhakrishnan. National Institute of Technology Karnataka. *Modeling, Simulation, and Control of a Quadcopter*. Recuperado de https://harikrishnansuresh.github.io/assets/quadcopter_control_project_report.pdf

[19] Wikipedia contributors. Wikipedia, *The Free Encyclopedia. Equations of motion*. Recuperado de https://en.wikipedia.org/w/index.php?title=Equations_of_motion&oldid=1154359156. Accedido el 15 de mayo de 2023.

[20] MathWorks Help Center. *Custom Variable Mass 6DOF (Quaternion)*. Recuperado de https://es.mathworks.com/help/releases/R2020a/aeroblks/customvariablemass6dofquaternion.html. Accedido el 16 de mayo de 2023.

[21] MathWorks Help Center. *Custom Variable Mass 6DOF (Euler Angles)*. Recuperado de https://es.mathworks.com/help/releases/R2020a/aeroblks/customvariablemass6dofeulerangles.html. Accedido el 16 de mayo de 2023.

[22] MathWorks Help Center. *About Aerospace Coordinate System*. https://es.mathworks.com/help/releases/R2020a/aeroblks/about-aerospace-coordinate-systems.html. Accedido el 16 de mayo de 2023.

[23] MathWorks Help Center. *Flath Earth to LLA*. Recuperado de https://es.mathworks.com/help/releases/R2020a/aeroblks/flatearthtolla.html. Accedido el 16 de mayo de 2023.

[24] Torres del Castillo, G.F. Universidad Autónoma de Puebla (México). *La representación de rotaciones mediante cuaterniones*. Recuperado de https://web.archive.org/web/20130512083041if_/http://www.miscelaneamatematica.org:80/Misc29/torres_c.pdf.

[25] Wikipedia contributors. Wikipedia, *The Free Encyclopedia. Quaternions and spatial rotation*. Recuperado de https://en.wikipedia.org/w/index.php?title=Quaternions_and_spatial_rotation&oldid=1147848141. Accedido el 18 de mayo de 2023.

[26] Colaboradores de Wikipedia. Wikipedia, la enciclopedia libre. *Cuaterniones y rotación en el espacio*. Recuperado de https://es.wikipedia.org/w/index.php?title=Cuaterniones_y_rotaci%C3%B3n_en_el_espacio&oldid=150416257. Accedido el 19 de mayo de 2023.

[27] Wikipedia contributors. Wikipedia, *The Free Encyclopedia. Quaternion*. Recuperado de https://en.wikipedia.org/w/index.php?title=Quaternion&oldid=1156440893. Accedido el 18 de mayo de 2023.

[28] MathWorks Help Center. *Choose a solver*. Recuperado de https://es.mathworks.com/help/simulink/ug/choose-a-solver.html. Accedido el 6 de junio de 2023.

[29] MathWorks Help Center. *Solver*. Recuperado de https://es.mathworks.com/help/simulink/gui/solver.html. Accedido el 6 de junio de 2023.

[30] Roselló Ferragud, María Dolores; et al. Editorial Universitat Politècnica de València. *Introducción a los métodos numéricos para ecuaciones diferenciales*. Segunda edición (2014).

[31] Wikipedia colaborators. Wikipedia, *The Free Encyclopedia. Runge–Kutta methods*. Recuperado de https://en.wikipedia.org/w/index.php?title=Runge%E2%80%93Kutta_methods&oldid=1151548276. Accedido el 6 de junio de 2023.

[32] Bogacki, Przemysław; Shampine, Lawrence F. Applied Mathematics Letters, Vol. 2, No. 4, pp. 321-325. *A 3(2) Pair of Runge–Kutta Formulas* (1989). Recuperado de (DOI) https://doi.org/10.1016%2F0893-9659%2889%2990079-7

[33] Wikipedia colaborators. Wikipedia, *The Free Encyclopedia. Simulink*. Recuperado de https://en.wikipedia.org/w/index.php?title=Simulink&oldid=1150194024. Accedido el 30 de junio de 2023.

[34] The MathWorks, Inc. *Simulink® User's Guide* (2018). Recuperado de https://es.mathworks.com/help/releases/R2018a/simulink/index.html

[35] The MathWorks, Inc. *Aerospace Blockset™ User's Guide* (2018). Recuperado de https://es.mathworks.com/help/releases/R2018a/aeroblks/index.html

[36] Embention. *SIL Simulink - Embention Manuals*. Recuperado de https://manuals.embention.com/sil-simulator/en/6.8/sil%20simulink/index.html. Accedido el 17 de julio de 2023.

[37] Colaboradores de Wikipedia. Wikipedia, la enciclopedia libre. *Coordenadas esféricas*. Recuperado de https://es.wikipedia.org/w/index.php?title=Coordenadas_esf%C3%A9ricas&oldid=151925167. Accedido el 18 de julio de 2023.

[38] Wikipedia colaborators. Wikipedia, *The Free Encyclopedia. Stokes' law*. Recuperado de https://en.wikipedia.org/w/index.php?title=Stokes%27_law&oldid=1150638379. Accedido el 18 de julio de 2023.

[39] Mi, B.; Zhan, H. Arch Computat Methods Eng 27. *Review of Numerical Simulations on Aircraft Dynamic Stability Derivatives* (2020). Recuperado de (DOI) https://doi.org/10.1007/s11831-019-09370-8

[40] Josselson, Robert. ITT Aerospace Systems Group. *Introduction to 6-DOF Simulation of Air Vehicles*. Recuperado de http://avionics.nau.edu.ua/files/doc/VisSim.doc/6dof.pdf

[41] Zapico, Eduardo; Giraudo, Pedro; Abbate, Horacio; Luiso, Javier. Asociación Argentina de Mecánica Computacional. *Modelos de dinámica del vuelo y aerodinámico de aeronaves para un simulador de vuelo 6-DOF en tiempo real* (2009). Recuperado de https://cimec.org.ar/ojs/index.php/mc/article/view/2932

[42] Roger Ull, Alejandro. Universitat Politècnica de Catalunya. *Apuntes: Mecánica de vuelo II* (2012). Recuperado de https://upload.wikimedia.org/wikipedia/commons/8/82/Mec%C3%A1nica_del_Vuelo_II_-_Ingenier%C3%ADa_aeron%C3%A1utica_-_ETSEIAT_-_UPC.pdf

[43] Rueda Monje, Francisco. Universidad Carlos III de Madrid. *Análisis simplificado del comportamiento vibratorio de una pala de helicóptero*. Recuperado de https://e-archivo.uc3m.es/bitstream/handle/10016/28755/TFG_Francico_Rueda_Monje_(corregido)_2017.pdf

[44] University of Surrey. *2nd Order System*. Recuperado de http://www.ee.surrey.ac.uk/Projects/CAL/control/2ndOrderSys.htm. Accedido el 25 de julio de 2023.

[45] Wikipedia colaborators. Wikipedia, *The Free Encyclopedia. International Standard Atmosphere*. Recuperado de https://en.wikipedia.org/w/index.php?title=International_Standard_Atmosphere&oldid=1165395213. Accedido el 25 de julio de 2023.

[46] National Centers for Environmental Information. *World Magnetic Model (WMM)*. Recuperado de https://www.ncei.noaa.gov/products/world-magnetic-model. Accedido el 26 de julio de 2023.

[47] Wikipedia colaborators. Wikipedia, *The Free Encyclopedia. Blade element momentum theory*. Recuperado de https://en.wikipedia.org/w/index.php?title=Blade_element_momentum_theory&oldid=1153809857. Accedido el 27 de julio de 2023.

[48] Ledoux, Jeremy; Riffo, Sebastián; Salomon, Julien. SIAM Journal on Applied Mathematics, 2021, 81 (6), pp.2596-2621. *Analysis of the Blade Element Momentum Theory.* Recuperado de `https://hal.science/hal-02550763v2`

[49] Roger Ull, Alejandro. Universitat Politècnica de Catalunya. *Apuntes: Diseño de helicópteros y aeronaves diversas* (2011). Recuperado de `https://upload.wikimedia.org/wikipedia/commons/1/12/Dise%C3%B1o_de_helic%C3%B3pteros_y_otras_aeronaves_diversas_-_Ingenier%C3%ADa_aeron%C3%A1utica_-_ETSEIAT_-_UPC.pdf`

[50] Barcala Montejano, Miguel A.; Rodríguez Sevillano, Ángel A. Universidad Politécnica de Madrid. *Helicópteros. Aerodinámica del rotor: Teoría del Elemento de Pala, Vuelo Vertical Ascendente.* Recuperado de `http://ocw.upm.es/pluginfile.php/438/mod_label/intro/tep-vva.pdf`

[51] NASA. Langley Research Center. *NASA SP-405. Vortex-Lattice Utilization* (1976). Recuperado de `https://ntrs.nasa.gov/citations/19760021075`

[52] Melin, Thomas. *About - Tornado. A Vortex-Lattice Method implemented in MATLAB.* Recuperado de `http://tornado.redhammer.se/index.php/about`

[53] Wikipedia colaborators. Wikipedia, *The Free Encyclopedia. United States Air Force Stability and Control Digital DATCOM.* Recuperado de `https://en.wikipedia.org/w/index.php?title=United_States_Air_Force_Stability_and_Control_Digital_DATCOM&oldid=1138229863`. Accedido el 28 de julio de 2023.

Parte II

PLIEGO DE CONDICIONES

Capítulo 1

Pliego de condiciones

1.1. Introducción

En este documento se presentan las condiciones sobre las que se desarrolla el presente Trabajo de Fin de Máster. Se van a exponer los requerimientos exigidos a *Embention Flight Simulator*, las condiciones de trabajo y los métodos de seguimiento y evaluación del desarrollo.

1.2. Pliego de condiciones

1.2.1. Condiciones de trabajo

El trabajo se desarrolla durante un convenio de prácticas en empresa, acordado con la Universidad Politécnica de Valencia y Embention Sistemas Inteligentes S.A. Se trata de un contrato a jornada completa, remunerado y presencial, desarrollado en las oficinas de Embention en Alicante.

Siguiendo la normativa de la Escuela Técnica Superior de Ingeniería del Diseño, se realiza un convenio curricular y otro extracurricular, con las condiciones que se muestran en la Tabla 1.1.

Tabla 1.1: Convenios de prácticas en empresa para el desarrollo de EFS

Modalidad	**Fecha incio**	**Fecha fin**	**Horas**
Curricular	20/02/2023	19/04/2023	337,5
Extracurricular	20/04/2023	16/06/2023	332

Tabla 1.2: Características del equipo informático facilitado

Equipo	Ordenador portátil MSI
Procesador	AMD Ryzen 7 5800H with Radeon Graphics 3,20 GHz
RAM instalada	16,0 GB (15,4 GB usable)
Sistema operativo	Windows 10. 64 bits, procesador basado en x64

En las oficinas de Embention se proporciona un puesto de trabajo, con el equipo informático descrito en la Tabla 1.2 acompañado de periféricos: ratón, teclado y monitor.

En general, por requerimiento de Embention, las tareas se desarrollaban en una máquina virtual, con las características de la Tabla 1.3.

Tabla 1.3: Características de la máquina virtual

Máquina virtual	Oracle VirtualBox
Procesador	AMD Ryzen 7 5800H with Radeon Graphics 3,19 GHz
RAM instalada	5,86 GB
Sistema operativo	Windows 10. 64 bits, procesador basado en x64

De las 40 horas semanales, la mayor parte son dedicadas al desarrollo del simulador. No obstante, al tratarse de un proyecto del departamento PDI (*Parameter Data Items*), también se dedica una pequeña cantidad de tiempo al apoyo de tareas del departamento, principalmente integraciones del autopiloto Veronte en aeronaves de los clientes.

1.2.2. Requerimientos exigidos a *Embention Flight Simulator*

El desarrollo del simulador se plantea como una tarea en GitHub, con los siguientes detalles:

Issue: Development of Simulink Simulator

References: -

Description: Desarrollo de un simulador en Simulink. La salida de este issue debe ser un bloque completo de Simulink con las siguientes características:

- Entradas:

- Posición de actuadores: A ser posible todos los valores en el rango [0,1], representando el movimiento del actuador de 0 a 100 %. El tamaño del vector dependerá del modelo. En principio, no se identifican más entradas indispensables, pero se podría añadir otras para modificar dinámicamente condiciones de entorno o de la simulación como el viento, condiciones atmosféricas, configuración de la aeronave, carga de pago, carga de combustible/batería, etc.

- Contenido:
  - Modelo matemático de la aeronave: Este bloque se reemplazará/modificará para cada servicio. De salida, deberíamos tener al menos dos bloques genéricos, incluyendo una serie de parámetros, que permitan modelar:
    - Multicóptero: número y posición de los motores, características de motores y rotores, masas e inercia, etc.
    - Ala fija: planta de potencia, masas e inercia, posición y tamaño de los actuadores, aeroderivadas, coeficientes aerodinámicos...
  - Simulador 6DOF.
  - Modelado actuadores.
  - Cálculo de sensores: una vez determinado el estado, se ha de generar las señales de los sensores oportunas para que Veronte pueda calcular su estado.

- Salidas:
  - Sensores: debe haber una salida por sensor disponible en Veronte. Se ha de tener en cuenta que las entradas de sensores en Veronte no solo incluyen la medida, pueden incluir más cosas como *flags* o temperaturas. Estas salidas deben cuadrar exactamente con las entradas de sensores requeridas por el bloque SIL de Veronte.
  - Estados: Todos los que se quieran, pero, al menos, posición y actitud para poder inyectarlos en un visualizador.

Además de las condiciones particulares del simulador, también se deben seguir los procedimientos y formatos exigidos en todos los trabajos de Embention. La más relevante es el convenio de signos e implementación, que da uniformidad y cohesión a todos los trabajos de la empresa, que trabaja con aeronaves muy diversas.

Convenio de signos e implementación:

- Estos convenios se eligen segun criterios de facilidad de uso, intuitividad y versatilidad, y deben utilizarse a menos que haya una razon de peso para no hacerlo (i.e. requisito de cliente).

- Unidades: Por regla general, las unidades utilizadas deben ser las unidades del Sistema Internacional. En algunos casos, se podrán utilizar unidades mas convenientes (i.e. grados para ángulos en lugar de radianes).

- Actitud: El convenio de signos para la actitud en Veronte es fijo y no se puede modificar. El *pitch* es positivo cuando sube el morro, el *roll* es positivo a derechas y el *yaw* también es positivo a derechas, como indica la Figura 1.1.

- Actuadores:

  - Motores: Tanto por 1. Posición mínima 0 (0%), posición máxima 1 (100%). -1 (-100%) para reversa.

  - Actuadores lineales: Servos y similares. De 0 a 1, representando 0 la posición más "encogida" y 1 la posición más "extendida". También se puede usar una unidad de posición si es muy representativa (por ejemplo ángulo o distancia). De -1 a 1 si la posición de reposo es intermedia (i.e. rueda).

  - Superficies de control: Grados. El movimiento positivo es siempre el que desplaza la superficie hacia arriba, para superficies horizontales, o hacia la derecha, para superficies verticales. Esto es contrario al criterio aeronáutico, pero permite una mayor coherencia para un gran número de configuraciones. En caso de una inclinación de 45 deg, tiene preferencia el movimiento vertical (positivo hacia "arriba").

  - Actuadores discretos: Como sistemas de paracaídas, frenos o luces. Apagado o desactivado: 0. Encendido o activado: 1.

- Motores:

  - Numeración de motores: Por lo general, para la numeración se utilizará aquella definida por el cliente. En caso de no estar definida o de ser un proyecto interno, los motores deben numerarse en sentido horario, siendo el numero 1 el motor situado directamente a la derecha de la parte frontal del vehículo (delante-derecha).

  - En caso de motores coaxiales, se numerará primero el motor superior, de forma que queden arriba los motores impares y abajo los motores pares.

  - En cuanto a la dirección de rotación, por lo general, el motor 1 deberá rotar en sentido contrario a las agujas del reloj, y a partir de ese punto se alternará como corresponda al vehículo en cuestión.

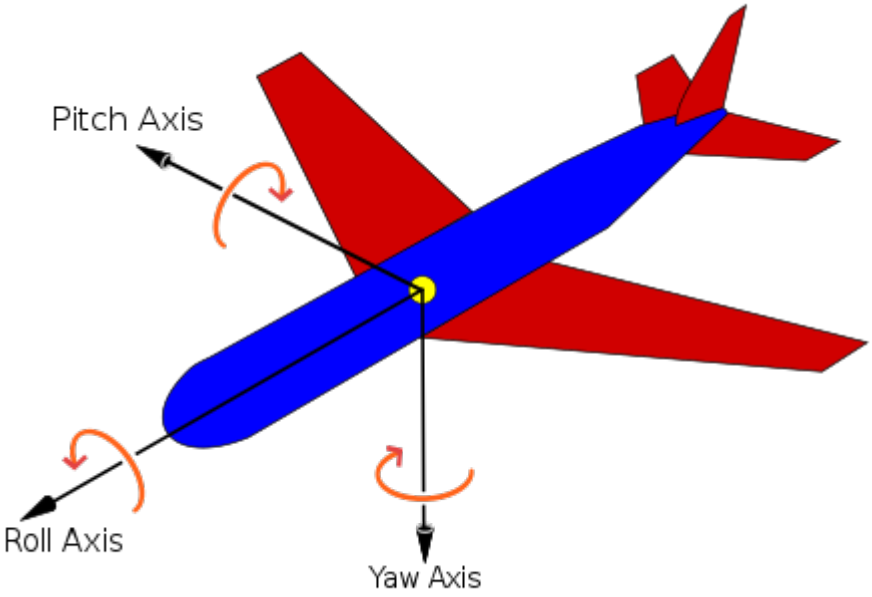

Figura 1.1: Ejes y criterio de actitud, Embention 2023

1.2.3. Evaluación de los desarrollos

El trabajo se realiza bajo el amparo del departamento PDI. El departamento se reúne semanalmente para evaluar los avances realizados, plantear problemas encontrados y plantear los siguientes pasos del desarrollo.

La versión final del simulador se evalúa mediante una presentación ante el CEO y los jefes de departamento, donde se exponen los logros alcanzados.

Bibliografía

[1] Escuela Técnica Superior de Ingeniería del Diseño. Universitat Politècnica de València. Recuperado de https://www.etsid.upv.es/alumnos/practicas-en-empresa/requisitos-y-condiciones/

Parte III

PRESUPUESTO

Capítulo 1

Presupuesto

1.1. Introducción

El objetivo de este documento es exponer la estimación del coste de la realización del presente Trabajo de Fin de Máster.

La divisa en la que se va a realizar este presupuesto es el Euro [€].

1.2. Presupuesto

Puesto que el trabajo se trata de elaboración de *software*, los principales gastos son debidos a las horas de ingeniería y las licencias informáticas.

Por un lado, se han dedicado 580 horas al desarrollo de *Embention Flight Simulator*, según la plataforma de control de tiempo de Embention, repartidas durante 17 semanas, con una retribución de 4,3 €/h brutos.

Por otro lado, se requieren licencias de MATLAB, Simulink, Aerospace Toolbox y Aerospace Blockset. MathWorks ofrece licencias anuales o perpetuas. En este presupuesto se incluye el precio anual de cada licencia, según los precios de su web [1].

Al resultado mostrado en la Tabla 1.1 debe añadirse una partida de gastos derivados del funcionamiento de la empresa, cubiertos por Embention. Los más relevantes son:

- Costes asociados a las oficinas y naves de trabajo, situadas en el Polígono de las Atalayas, en Alicante. Costes de mantenimiento, de electricidad y agua e impuestos.

- Costes asociados al uso de los equipos informáticos cedidos y sus periféricos. Compra y reparación de equipos.
- Costes del mantenimiento de los servidores de Embention y sus páginas web.
- Pago de licencias de *software* de gestión y administración de la empresa.

Tabla 1.1: Presupuesto de *Embention Flight Simulator*

Concepto	Unidades	Coste unitario [€]	Coste total [€]
Ingeniero en prácticas	580 h	4,3 €/h	2494 €
Licencia MATLAB	1	860 €	860 €
Licencia Simulink	1	1300 €	1300 €
Licencia Aerospace Toolbox	1	540 €	540 €
Licencia Aerospace Blockset	1	740 €	740 €
		Total	5934 €

En conclusión, el coste derivado del desarrollo de *Embention Flight Simulator*, sin considerar gastos indirectos de la empresa, asciende a **5934 euros**.

Bibliografía

[1] MathWorks. *Pricing and Licensing*. Recuperado de `https://es.mathworks.com/pricing-licensing.html?prodcode=AE&intendeduse=comm`

Parte IV

ANEXOS

Anexo I - Manual de uso y desarrollo

Manual de uso y desarrollo

Embention Flight Simulator

Embention

Alicante - Mayo de 2023

Índice

Índice de tablas

Índice de figuras

Tabla de símbolos

Símbolo	Definición
6DOF	6 *Degrees of freedom*, 6 grados de libertad
α	Ángulo de ataque
β	Ángulo de derrape
D	*Drag*, resistencia aerodinámica
EFS	Embention Flight Simulator
FTF	Función de transferencia en frecuencia
GNSS	*Global Navigation Satellite System*, Sistema global de navegación por satélite
L	*Lift*, sustentación
l	*Roll*, alabeo
m	*Pitch*, cabeceo
MAC	*Mean Aerodynamic Chord*, Cuerda media aerodinámica
MLS	*Mean Sea Level*, altitud respecto al nivel del mar
n	*Yaw*, guiñada
ω	Velocidad angular del rotor del motor
ω_n	Frecuencia natural
p	Velocidad de rotación respecto del eje x
p_{inf}	Presión estática
ϕ	Acimut: ángulo de empuje del motor respecto al eje x en el plano xy, en ejes cuerpo
Q	*Torque*, par generado por el motor
q	Velocidad de rotación respecto del eje y
q_{inf}	Presión dinámica
r	Velocidad de rotación respecto del eje z
SIL	*Software in the loop*
T	*Thrust*, empuje del motor
θ	Colatitud o polar: ángulo de empuje del motor respecto al eje z, en ejes cuerpo
$TSFC$	*Thrust-specific fuel consumption*, Consumo específico de combustible por empuje
Y	*Side force*, fuerza lateral
ζ	Ratio de amortiguamiento

Adendas

0.1. Embention Flight Simulator v0.1 - 8 de junio de 2023

La primera versión publicada en el repositorio cuenta con ciertas mejoras y modificaciones respecto a la versión descrita en este documento (mayo 2023).

0.1.1. Mejoras realizadas

- **Nuevos modelos**
  Se han añadido nuevos modelos: el modelo de consumo de combustible con variación de la inercia y el modelo de empuje mediante paso variable. Estos emplean *Lookup tables* principalmente.

- ***Propeller estimation***
  Se ha añadido la herramienta *Propeller estimation* que obtiene las tablas de empuje y par de una hélice conociendo ciertos parámetros geométricos. Esta herramienta emplea una implementación de las teorías de elemento de pala y de cantidad de movimiento, muy usadas para realizar primeras estimaciones conociendo el perfil aerodinámico de la hélice y su geometría.

- **Carpeta *examples*** En la carpeta *examples* se ha incluido un archivo totalmente preparado para la realización de una simulación con la aeronave M600. Se ha creado y ajustado un modelo del M600 que permite simular una misión empleando la misma configuración de Veronte que emplea la aeronave real.

- **Reestructuración**
  Se ha reestructurado ligeramente la carpeta del proyecto, aunque las bases mostradas en este documento se mantienen. Por ejemplo, los archivos de *.SimulinkProject* se guardan ahora en la carpeta *resources*. Esto se debe a que se ha migrado completamente el desarrollo a Matlab 2020a, que trabaja llamando *resources* a la carpeta que en 2018a llamaba *.SimulinkProject*. Otra consecuencia de la migración es la aparición de ciertos modelos retrocompatibles con 2018a. Se han mantenido estos modelos en las carpetas del simulador, aunque se desaconseja su uso puesto que no se han realizado las mismas validaciones que en los modelos 2020a.

1. Introducción al simulador

Embention Flight Simulator (EFS) se trata de un simulador de vuelo flexible y configurable, de 6 grados de libertad, desarrollado en Simulink. Está diseñado para trabajar en conjunto con el bloque de Veronte en Simulink para simulaciones *Software in the loop* (SIL), siguiendo el esquema que se muestra en la Figura 1.

El simulador de vuelo contiene modelos matemáticos que describen el comportamiento de una aeronave según las órdenes de control especificadas por el usuario o, en el caso de una simulación SIL, por el autopiloto Veronte. A través de estos modelos matemáticos, el simulador calcula el estado de la aeronave (posición, actitud, velocidad, condiciones ambientales...) y los envía como salida para su representación. Por otro lado, a partir de estos estados, el simulador genera las señales que leerían los sensores si estuvieran equipados en una aeronave en las condiciones calculadas por el simulador. Estas señales son las que se envían a Veronte para que el autopiloto decida las órdenes de control, cerrando así el lazo.

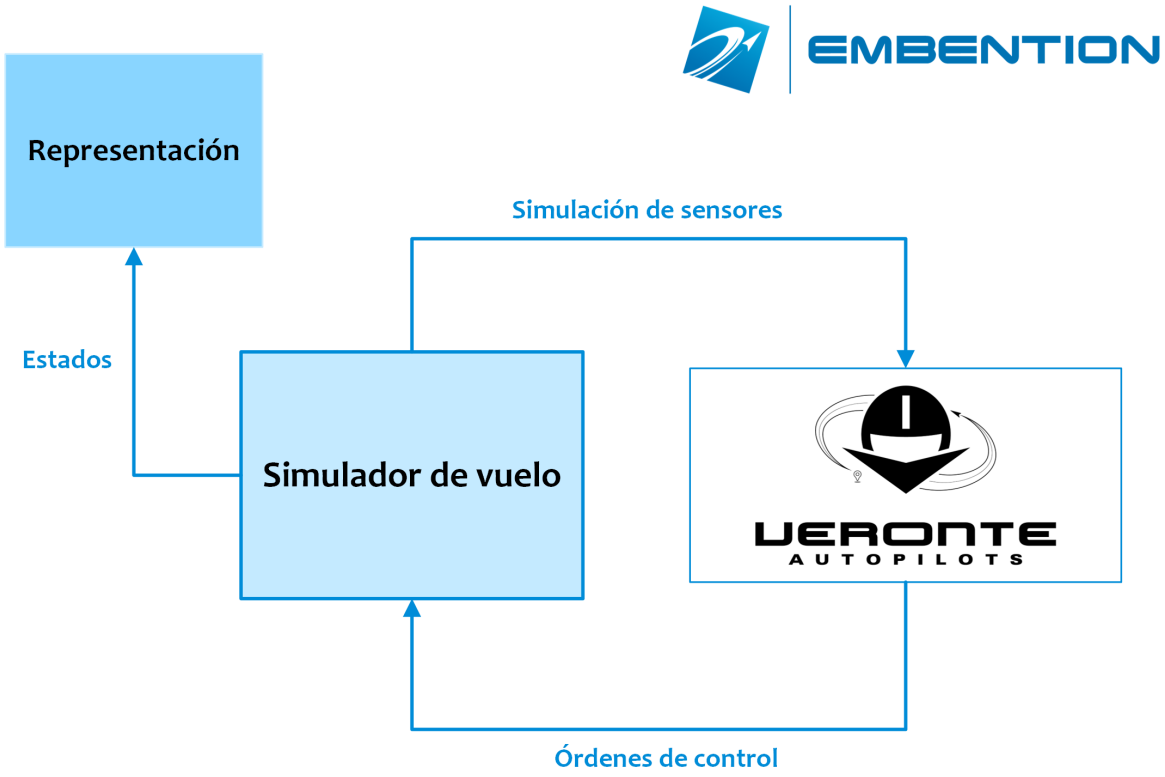

Figura 1: Esquema general de funcionamiento EFS + Veronte SIL

Por otro lado, en la Figura 2, se muestra un diagrama detallado del funcionamiento interno del simulador. Como entradas se tienen las órdenes de control, que se aplican al modelo de la aeronave a través de los modelos de actuadores. A partir de estos actuadores, de las condiciones ambientales y del estado de la aeronave, se obtienen las fuerzas y los momentos aplicados y que el bloque de ecuaciones del movimiento emplea para obtener el estado de la aeronave en el instante siguiente.

Estos estados se envían como salida del simulador y se emplean de forma interna para obtener las condiciones ambientales (presión atmosférica, temperatura ambiente, campo magnético terrestre, viento...), las fuerzas y momentos aplicados o la simulación de lecturas de los sensores.

El segundo *output* se trata de las señales de los sensores simulados, las cuales se adaptan para funcionar directamente en el bloque Veronte SIL.

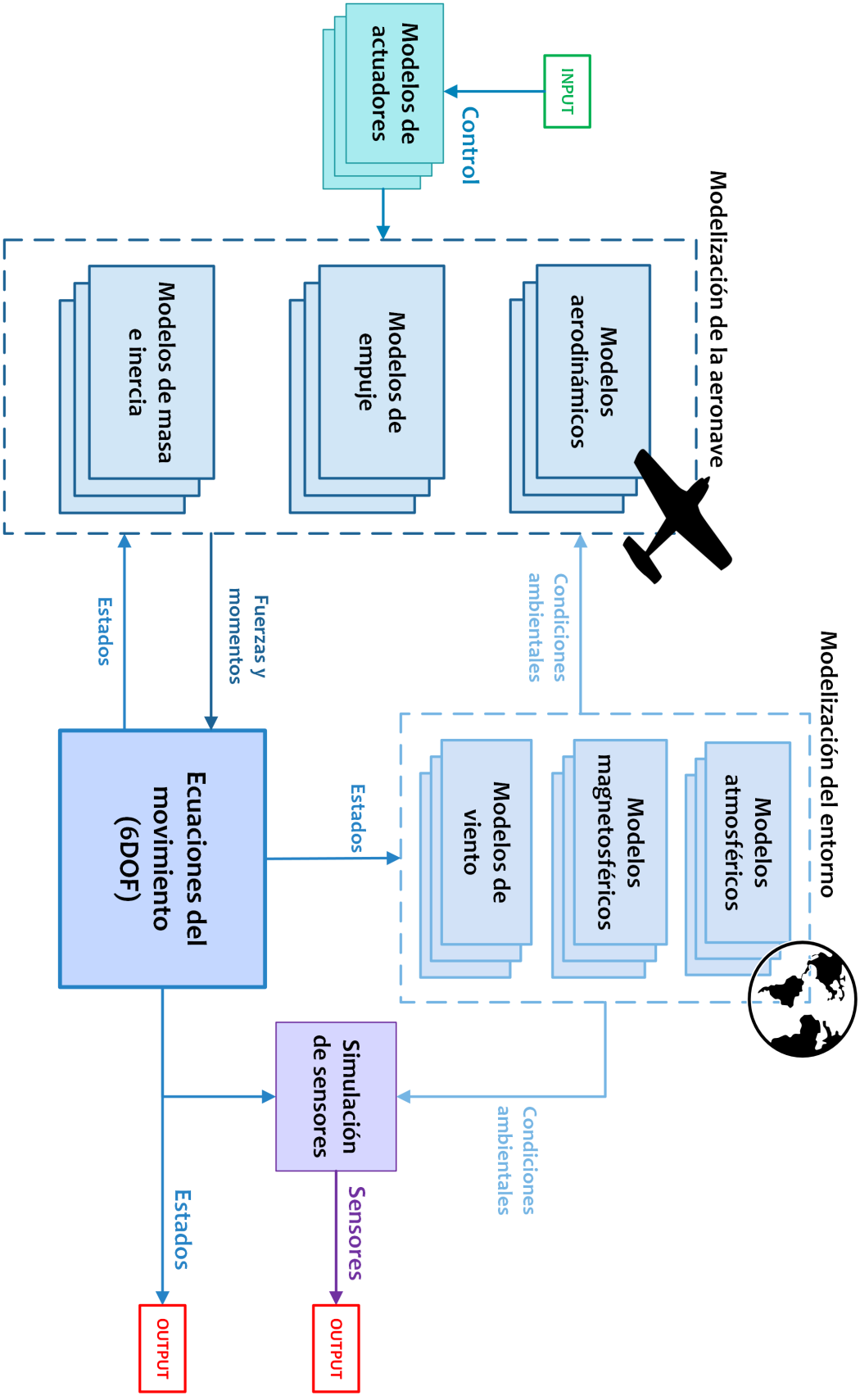

Figura 2: Esquema detallado del funcionamiento de EFS

Respecto a su implementación en Simulink, cabe destacar el empleo de modelos variantes (*Variant Models*). Estos bloques de Simulink contienen dentro varios modelos, de forma que el usuario selecciona previamente el modelo con el que va a ser ejecutada la simulación. Esto permite, en un único archivo de Simulink, tener un simulador flexible y adaptable, lo cual es de especial interés en este caso, puesto que se requiere que EFS funcione para diversos tipos de plataformas (desde ala fija hasta multicópteros). Para ayudar en la tarea de configuración previa a la ejecución del simulador, se ha desarrollado un lanzador (*EFSLauncher.m* o *EFSLauncher_live.mlx*) que se asegura de que se han especificado todos los parámetros necesarios para el correcto funcionamiento de EFS.

En resumen, se emplean modelos matemáticos para obtener los parámetros requeridos por el bloque de ecuaciones de movimiento (véase *Custom Variable Mass 6DOF (Quaternion)*, incluido en el Aerospace Blockset de Simulink). A partir de los estados de la aeronave calculados por el bloque, se simulan los sensores y se obtienen de nuevo los parámetros necesarios para la siguiente iteración. El estado inicial del simulador es conocido (se debe indicar en la configuración).

2. Manual de uso de EFS

2.1. Estructura del *Simulink Project*

Embention Flight Simulator está desarrollado como un proyecto de Simulink (*Simulink Project*). El archivo con extensión *.prj* es el que se encarga de abrir el proyecto, añadiendo las carpetas necesarias al *path* de Matlab para garantizar el correcto funcionamiento del simulador y abriendo el *live script* (*.mlx*) de bienvenida. En la Figura 3 se puede ver la estructura de las carpetas que conforman el simulador, y que se detallan a continuación:

- **aerodynamicModels**: almacena los archivos de Simulink (*.slx*) que modelan el comportamiento aerodinámico de la aeronave simulada.
- **configuration**: contiene todos los *scripts* de Matlab en los que se leen los parámetros necesarios para el funcionamiento del simulador (parámetros de configuración del simulador y de los modelos).
- **mainModels**: en esta carpeta se recogen los modelos de Simulink principales, es decir, el modelo del simulador de vuelo y otros modelos preparados para el funcionamiento del simulador de forma aislada o en conjunto con el bloque Veronte.
- **massInertiaModels**: contiene los modelos de Simulink que caracterizan la masa y la inercia de la aeronave.
- **outputs**: almacena los archivos de salida del simulador, como la generación de los archivos *.kml* o los resúmenes de simulación.
- **projectScripts**: contiene ciertos ficheros relacionados con el proyecto de Simulink, como atajos o el archivo de bienvenida.
- **resources**: es la carpeta donde se almacenan imágenes y logos.
- **thrustModels**: contiene los modelos de Simulink que caracterizan el empuje de los motores de la aeronave simulada.
- **tools**: recoge todas las herramientas desarrolladas para el simulador.
- **VeronteSIL**: es la carpeta donde se encuentran los archivos relacionados con el bloque Veronte para simulaciones SIL.
- **work**: en esta carpeta Simulink almacena los archivos generados durante la compilación y el funcionamiento del simulador.
- **EFSlauncher.m**: es el archivo del lanzador del simulador en formato *script* de Matlab.
- **EFSlauncher_live.mlx**: lanzador del simulador en formato *live script* de Matlab.

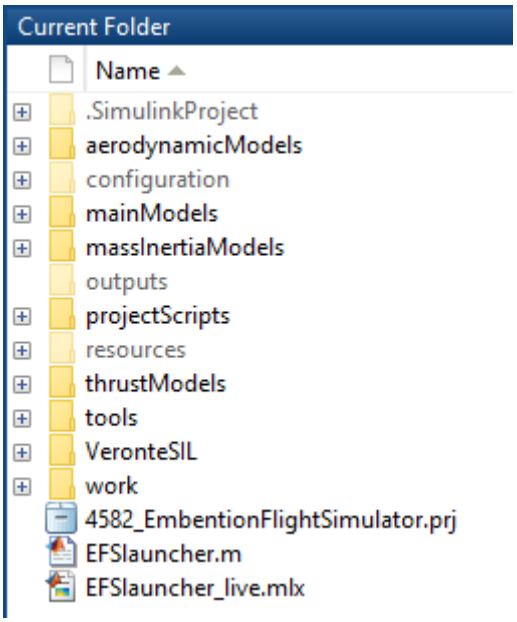

Figura 3: Estructura de carpetas de Embention Flight Simulator

2.2. Ejecución y uso del lanzador

Como se ha comentado en el apartado anterior, Embention Flight Simulator es un proyecto de Simulink, de forma que para ejecutarlo se debe abrir el archivo *4582_EmbentionFlightSimulator.prj* (el nombre del archivo de proyecto puede cambiar en versiones posteriores). Una vez abierto, se muestran dos pestañas: el proyecto, con las carpetas y archivos que lo conforman, y el *live scrpit* de bienvenida, tal y como se muestra en la Figura 4. En la pestaña de proyecto se pueden usar diversas herramientas ofrecidas por Matlab para este tipo de archivos, como el análisis de dependencia. Por otro lado, el archivo de bienvenida muestra cierta información relevante del simulador y permite incluir enlaces a la documentación, a los manuales de Embention o a otros lugares de interés.

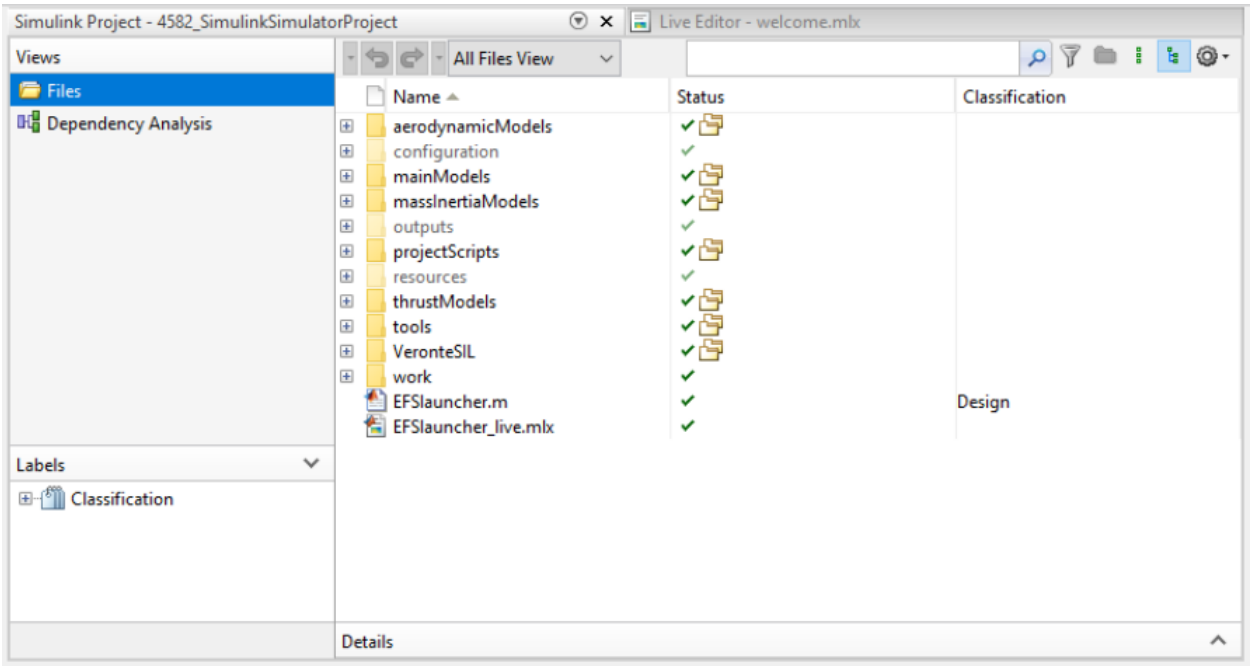

Figura 4: Vista de Matlab al abrir el proyecto

En la cinta de herramientas superior del IDE de Matlab (Figura 5) aparece otra de las funcionalidades de los proyectos de Simulink: los atajos o *shortcuts*. En esta versión de EFS se tienen tres apartados: *launchers*, *reset* y *tools*.

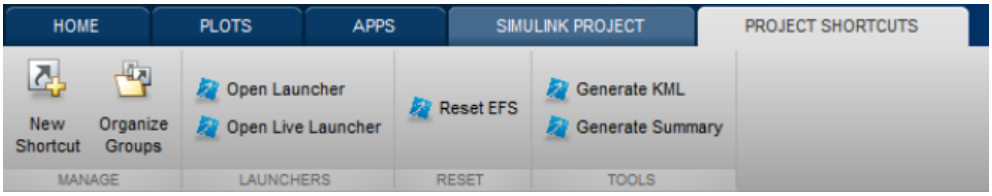

Figura 5: Cinta de herramientas del editor de Matlab 2018a

Por un lado, los lanzadores abren en el editor los archivos que permiten configurar el simulador y ejecutarlo. Sobre estos lanzadores se profundiza en el apartado siguiente. Por otro lado, *reset* borra todos los parámetros del *workspace* de Matlab, reiniciando el simulador. Finalmente, en esta versión se cuenta con dos herramientas: *Generate KML* y *Generate Summary*, que crean un *.kml* con la trayectoria seguida por la aeronave en la simulación anterior y un resumen de los parámetros y modelos utilizados, respectivamente.

2.2.1. Lanzadores de Embention Flight Simulator

Los lanzadores o *launchers* son los archivos donde se eligen y configuran los modelos. Se tienen dos *launchers*, con un funcionamiento idéntico pero dos formatos.

EFSlauncher.m es un *script* de Matlab clásico. Para seleccionar los modelos, simplemente hay que seguir las líneas de código y editar las variables requeridas y debidamente indicadas en los comentarios del código. Según el modelo seleccionado, se ejecutan también unos archivos de configuración (de la carpeta *configuration*), y que hay que editar para modelar la aeronave requerida. Una vez terminado el proceso, al ejecutar el lanzador, se leen todos los parámetros necesarios y se abre el modelo principal, preparado para realizar la simulación.

Si se opta por el lanzador *live script*, se pueden aprovechar las características de este tipo de ficheros de Matlab. En la versión en la que se ha desarrollado el simulador (Matlab 2018a), se puede alternar entre código, texto e imágenes, permitiendo explicar con mayor facilidad la funcionalidad de cada sección del código del lanzador. Además, se pueden añadir controles que dotan de cierta interfaz gráfica al lanzador, si bien estas características son mucho más avanzadas en versiones posteriores de Matlab, por lo que se podría mejorar en Matlab 2020a, empleado en las nuevas versiones del bloque SIL de Veronte (las últimas pruebas ya se han realizado en 2020a, por lo que puede ser que el simulador se presente directamente en esta versión).

En definitiva, para realizar una simulación, se deben elegir los modelos en el lanzador, editar los archivos de configuración (el lanzador ofrece la posibilidad de abrir en el editor directamente todos los archivos requeridos) y ejecutar el lanzador. Esto abrirá Simulink con el modelo configurado y listo para ejecutar.

2.3. Uso de los modelos: recomendaciones y limitaciones

Como se observa en la Figura 2, el simulador tiene la posibilidad de elegir entre diversos modelos para adaptarse al tipo de aeronave que se quiere simular. En general, hay modelos de aeronave, de entorno, de actuadores y de sensores.

2.3.1. Modelos de aeronave

Para modelar una aeronave en Embention Flight Simulator, es necesario especificar su comportamiento aerodinámico, el funcionamiento de su planta de empuje y su masa e inercia.

Modelos aerodinámicos

- Modelo sin fuerzas aerodinámicas
  Este modelo no contempla fuerzas debidas al movimiento relativo de la aeronave respecto al aire. Esto, por lo general, está muy alejado de la realidad, pero permite aislar y comprobar otras fuerzas, como el peso y el empuje. También se puede utilizar como primera aproximación de aeronaves sin alas fijas.

- Modelo de *drag* lineal
  Este modelo considera la resistencia aerodinámica como una fuerza lineal con la velocidad. Se basa en el *drag* de Stokes, válido para números de Reynolds muy bajos. Permite modelar la resistencia aerodinámica para aeronaves pequeñas y que vuelan a bajas velocidades. Puede emplearse en primeras estimaciones de pequeños multicópteros que vuelan a baja velocidad, aunque ofrece poca precisión y su validez es muy limitada.

- Modelo de coeficientes aerodinámicos
  Es un modelo flexible, pensado para aeronaves con alas fijas, que emplea diferentes coeficientes aerodinámicos para calcular las fuerzas y momentos debido a la interacción de la aeronave con el aire. Los coeficientes debido a los estados de la aeronave (α, $\dot{\alpha}$, β, $\dot{\beta}$, p, q, r), los debidos a superficies de control (alerones, *flaps*, *slats*, *rudder*, elevadores, etc.) y los debidos a actuadores discretos (tren de aterrizaje, gimbales...) se indican por separado, permitiendo una gran adaptación. Estos coeficientes se obtienen a partir de una linealización del comportamiento de la aeronave entorno al crucero, por lo que su validez se ve afectada en otras etapas del vuelo o en maniobras bruscas. Se recomienda su uso en aeronaves con ala fija, que no efectúen maniobras lejos del rango de validez de los coeficientes aerodinámicos.

Modelos de empuje

- Modelo sin empuje
  El modelo sin empuje se puede emplear para realizar pruebas o bien para el vuelo de aeronaves sin propulsión, como planeadores.

- Modelo de empuje simple
  Este modelo obtiene el empuje de un motor a través de una constante que relaciona las revoluciones por minuto del motor con su empuje y su par producido. Es un modelo muy simple, que permite caracterizar de forma sencilla algunos motores con rangos de funcionamiento pequeños. También permite obtener el consumo de combustible especificando el $TSFC$.

- Modelo de empuje vectorial simple
  Se trata de un modelo idéntico al de empuje simple, con la particularidad de que la dirección del empuje se indica como *input*, permitiendo modelar rotores con *tilt* variable o empuje vectorial.

- Modelo *lookup table* de empuje
  En este caso se modeliza el empuje a través de una tabla que relaciona el *throttle* con el empuje, el par producido por el motor y el consumo. Este modelo, aunque no tiene en cuenta todos los factores que afecta al empuje, permite una buena aproximación si las condiciones de vuelo son similares a las condiciones en las que se obtuvieron los datos de la tabla.

Modelos de masa e inercia

- Modelo de masa e inercia constante
  El modelo de masa e inercia constante es válido para aeronaves que no cambian de masa ni inercia durante el vuelo, principalmente aeronaves con motores eléctricos que no experimentan cambios significativos en su geometría durante el vuelo (como por ejemplo, despliegue de cámaras de peso considerable).

- Modelo de reparto de paquetes
  Este modelo considera una masa y un tensor de inercia de la aeronave con una carga equipada, y otra masa e inercia sin la carga equipada. A través del input *Payload change*, se cambia de forma instantánea entre estas masas e inercias, modelando de forma simple un cambio repentino de la carga de pago como, por ejemplo, soltar un paquete.

- Consumo de combustible simple
  El modelo de consumo de combustible simple considera la matriz de inercia como constante. Únicamente modela la pérdida de masa debida al consumo que los modelos de empuje calculan. Este modelo cuenta con una masa mínima, a partir de la cual los motores se apagan y dejan de producir empuje, simulando una aeronave que se queda sin combustible. Es útil para aeronaves con motores de combustión que no experimentan un cambio considerable de la inercia durante el vaciado de los depósitos.

2.3.2. Modelos de entorno

Modelos atmosféricos

- Modelo *International Standard Atmosphere* (ISA)
  El modelo ISA obtiene las características del aire (presión, temperatura, densidad...) a partir de la altitud a la que se encuentra la aeronave.

Modelos magnetosféricos

- Modelo *World Magnetic Model* (WMM) con una medida
  El modelo WMM recoge el campo magnético de la Tierra en función de la latitud, la longitud y la altura. Este modelo solo realiza una medida del campo magnético antes de realizar la simulación, y rota este campo según la orientación de la aeronave. El hecho de tomar una única medida se debe al alto coste computacional que el bloque WMM de Simulink tiene. Esto limita el modelo a misiones en las que la aeronave no se aleje considerablemente de la posición inicial, pero permite simulaciones más rápidas. Por otro lado, cabe destacar que el modelo que emplea esta versión de Matlab es válido entre 2015 y 2020, por lo que para maximizar la precisión se debería actualizar el WMM.

Modelos de viento

- Modelo sin viento
  El modelo sin viento no aplica viento de ningún tipo, siendo útil para simulaciones de días con la atmósfera en calma.

- Modelo de viento constante
  En este caso se modeliza el viento como una velocidad constante que se añade a la velocidad relativa entre la aeronave y el aire, afectando así a las fuerzas aerodinámicas. Debe tenerse en cuenta que el efecto de este modelo en la aeronave dependerá de cómo se haya caracterizado su comportamiento aerodinámico.

2.3.3. Modelos de actuadores

Modelos de actuadores

- Modelo de actuadores ideales
  En el caso de emplear el modelo de actuadores ideales, las señales enviadas por Veronte se aplican de forma directa e ideal al simulador, provocando una respuesta instantánea en la simulación. Aunque esto no se da en la realidad, es útil para realizar primeras estimaciones cuando no se conoce el comportamiento de los actuadores reales.

- Modelo de actuadores de segundo orden
  El modelo de actuadores de segundo orden caracteriza la respuesta del actuador mediante una función de transferencia de segundo orden, determinada por su frecuencia natural y su ratio de amortiguamiento, imitando de mejor forma el comportamiento no ideal de los actuadores reales.

2.3.4. Modelos de sensores

La señal de los sensores se construye a partir de los estados del simulador. Se puede indicar la posición de algunos sensores, como las IMUs o la dirección del tubo de Pitot. Además, se puede añadir ruido blanco generado por los bloques *Band-limited White Noise*, indicando la potencia en decibelios del ruido requerido. Cabe destacar que Veronte ignora las señales de algunos sensores en caso de que la medida sea totalmente constante, por lo que añadir cierto ruido, además de acercarse más a la realidad, puede ayudar a evitar errores.

2.4. Configuración y parámetros necesarios

En el lanzador del simulador se pueden ajustar algunos parámetros requeridos en cualquier tipo de simulación, como el tiempo de muestreo o la posición inicial de la aeronave. Estos parámetros se exponen en la Tabla 1.

Por otro lado, los parámetros que dependen de los modelos seleccionados se ajustan en los archivos de configuración propios del modelo. Todos ellos se encuentran en su respectiva carpeta dentro de *configuration*. En las Tablas 2, 3, 4, 5 y 6 se indican los archivos que ejecuta el lanzador para configurar los modelos.

Tabla 1: Parámetros generales del simulador

Parámetros generales del simulador			
Parámetro	**Variable**	**Unidades**	**Descripción**
Tiempo de muestreo	sampleTime	[s]	Tiempo de muestreo del simulador
Latitud inicial	initialLat	[deg]	Latitud de la posición inicial
Longitud inicial	initialLon	[deg]	Longitud de la posición inicial
Altitud inicial	initialAlt	[m]	Altitud (MSL) de la posición inicial
Velocidad inicial	initialVelocity	[m/s, m/s, m/s]	Vector velocidad inicial (ejes cuerpo)
Actitud inicial	initialEuler	[rad, rad, rad]	*Roll*, *pitch* y *yaw* iniciales
Velocidad de rotación inicial	initialRot	[rad/s, rad/s, rad/s]	p, q, r iniciales

Tabla 2: Archivos de configuración de la geometría de la aeronave

Configuración de la geometría		
Geometría	**Archivo de configuración**	**Modelos que lo emplean**
Geometría de los motores	*thrust_geometry*	Thrust = 2, 3, 4
Geometría de la aeronave	*aircraft_geometry*	Aero = 2, 3

Tabla 3: Archivos de configuración de los modelos de aeronave

Configuración de los modelos de aeronave		
Modelos aerodinámicos		
Modelo	**Selección de modelo**	**Archivo de configuración**
Sin fuerzas aerodinámicas	Aero = 1	-
Drag lineal	Aero = 2	*linear_drag_config*
Coeficientes aerodinámicos	Aero = 3	*coef_aerodynamic_config*
Modelos de empuje		
Modelo	**Selección de modelo**	**Archivo de configuración**
Sin empuje	Thrust = 1	-
Empuje simple	Thrust = 2	*simple_thrust_config*
Empuje vectorial simple	Thrust = 3	*simple_thrust_config*
Empuje tabulado	Thrust = 4	*lookup_table_thrust_config*
Modelos de masa e inercia		
Modelo	**Selección de modelo**	**Archivo de configuración**
Masa e inercia constante	MassInertia = 1	*mass_inertia_config*
Reparto de paquetes	MassInertia = 2	*package_delivery_config*
Consumo de combustible simple	MassInertia = 3	*fuel_consumption_config*

Tabla 4: Archivos de configuración de los modelos de entorno

Configuración de los modelos de entorno		
Modelos atmosféricos		
Modelo	**Selección de modelo**	**Archivo de configuración**
Atmósfera Estándar Internacional	Atmosphere = 1	-
Modelos magnetosféricos		
Modelo	**Selección de modelo**	**Archivo de configuración**
Modelo magnético mundial (1 medida)	Magnetosphere = 1	*wmm1mag_config*
Modelos de viento		
Modelo	**Selección de modelo**	**Archivo de configuración**
Sin viento	Wind = 1	-
Viento constante	Wind = 2	*constant_wind_config*

Tabla 5: Archivos de configuración de los modelos de actuadores

Configuración de los modelos de actuadores		
Modelo	**Selección de modelo**	**Archivo de configuración**
Actuadores ideales	Actuators = 1	-
Actuadores de segundo orden	Actuators = 2	*actuators_config*

Tabla 6: Archivos de configuración de los modelos de sensores

Configuración de los modelos de sensores
Archivo de configuración
sensors_config

En las Tablas 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18 y 19 se exponen los parámetros de cada archivo de configuración y que el usuario debe proporcionar para ajustar el modelo al tipo de aeronave y las condiciones deseadas. Además, para mayor claridad, en las Figuras 6 y 7 se exponen todos los parámetros requeridos por el simulador de forma esquemática.

Tabla 7: Parámetros de *thrust_geometry*

Parámetros de *thrust_geometry*			
Ruta:	configuration/geometry/thrust_geometry		
Parámetro	**Variable**	**Unidades**	**Descripción**
Número de motores	nThrusters	[-]	Número total de motores
Coordenadas X	thrusterX	[m]	Vector de coordenadas X de los motores
Coordenadas Y	thrusterY	[m]	Vector de coordenadas Y de los motores
Coordenadas Z	thrusterZ	[m]	Vector de coordenadas Z de los motores
Dirección de empuje	thrustDirection	[deg, deg]	Matriz con (θ, ϕ) de cada motor (en filas)

Tabla 8: Parámetros de *aircraft_geometry*

Parámetros de *aircraft_geometry*			
Ruta:	configuration/geometry/aircraft_geometry		
Parámetro	**Variable**	**Unidades**	**Descripción**
Superficie alar	Sw	[m^2]	Superficie alar de referencia para los coeficientes
Envergadura	b	[m]	Envergadura de la aeronave
MAC	c	[m]	Cuerda de referencia para los coeficientes

Tabla 9: Parámetros de *linear_drag_config*

Parámetros de *linear_drag_config*			
Ruta:	configuration/aerodynamic/linear_drag_config		
Parámetro	**Variable**	**Unidades**	**Descripción**
Coeficiente de resistencia en x	kdx	[N/(m/s)]	Coeficiente de resistencia en el eje x del cuerpo
Coeficiente de resistencia en y	kdy	[N/(m/s)]	Coeficiente de resistencia en el eje y del cuerpo
Coeficiente de resistencia en z	kdz	[N/(m/s)]	Coeficiente de resistencia en el eje z del cuerpo

Nota: Los coeficientes mostrados relacionan la velocidad de la aeronave con la resistencia: $D_x = kdx \cdot v_x$.

Tabla 10: Parámetros de *coef_aerodynamic_config*

Parámetros de *coef_aerodynamic_config*			
Ruta:	configuration/aerodynamic/coef_aerodynamic_config		
Parámetro	**Variable**	**Unidades**	**Descripción**
Coeficientes aerodinámicos	aeroCoeff	[-]	Matriz de coeficientes aerodinámicos
Coef. resistencia inducida	K	[-]	Coeficiente resistencia inducida (opcional)
Coef. aerodinámicos control	controlCoeff	[-]	Matriz de coeficientes de superficies de control
Coef. aerodinámicos discretos	discreteAeroCoeff	[-]	Matriz de coeficientes de actuadores discretos

Nota: Las matrices de coeficientes aerodinámicos se especifican en las ecuaciones (1), (2) y (3). Las matrices de coeficientes de control y de actuadores discretos tienen un número de columnas variable, según el número de superficies de control o actuadores. Por otro lado, se puede especificar el *drag* mediante sus coeficientes o empleando la curva polar $(CD = CD_0 + K \cdot (CL - CL_0)^2$. Para ello, se establecen los coeficientes de resistencia de la matriz aeroCoeff a cero (excepto CD_0) y se le da un valor a K.

$$\text{aeroCoeff} = \begin{pmatrix} CD_0 & CD_\alpha & CD_{\dot\alpha} & CD_\beta & CD_{\dot\beta} & CD_p & CD_q & CD_r \\ CY_0 & CY_\alpha & CY_{\dot\alpha} & CY_\beta & CY_{\dot\beta} & CY_p & CY_q & CY_r \\ CL_0 & CL_\alpha & CL_{\dot\alpha} & CL_\beta & CL_{\dot\beta} & CL_p & CL_q & CL_r \\ Cl_0 & Cl_\alpha & Cl_{\dot\alpha} & Cl_\beta & Cl_{\dot\beta} & Cl_p & Cl_q & Cl_r \\ Cm_0 & Cm_\alpha & Cm_{\dot\alpha} & Cm_\beta & Cm_{\dot\beta} & Cm_p & Cm_q & Cm_r \\ Cn_0 & Cn_\alpha & Cn_{\dot\alpha} & Cn_\beta & Cn_{\dot\beta} & Cn_p & Cn_q & Cn_r \end{pmatrix} \tag{1}$$

$$\text{controlCoeff} = \begin{pmatrix} CD_{\text{control surface 1}} & \cdots & CD_{\text{control surface } n} \\ CY_{\text{control surface 1}} & \cdots & CY_{\text{control surface } n} \\ CL_{\text{control surface 1}} & \cdots & CL_{\text{control surface } n} \\ Cl_{\text{control surface 1}} & \cdots & Cl_{\text{control surface } n} \\ Cm_{\text{control surface 1}} & \cdots & Cm_{\text{control surface } n} \\ Cn_{\text{control surface 1}} & \cdots & Cn_{\text{control surface } n} \end{pmatrix} \tag{2}$$

$$\text{discreteAeroCoeff} = \begin{pmatrix} CD_{\text{discrete actuator 1}} & \cdots & CD_{\text{discrete actuator } n} \\ CY_{\text{discrete actuator 1}} & \cdots & CY_{\text{discrete actuator } n} \\ CL_{\text{discrete actuator 1}} & \cdots & CL_{\text{discrete actuator } n} \\ Cl_{\text{discrete actuator 1}} & \cdots & Cl_{\text{discrete actuator } n} \\ Cm_{\text{discrete actuator 1}} & \cdots & Cm_{\text{discrete actuator } n} \\ Cn_{\text{discrete actuator 1}} & \cdots & Cn_{\text{discrete actuator } n} \end{pmatrix} \tag{3}$$

Tabla 11: Parámetros de *simple_thrust_config*

Parámetros de *simple_thrust_config*			
Ruta:	configuration/thrust/simple_thrust_config		
Parámetro	**Variable**	**Unidades**	**Descripción**
Relación T - ω	thrusterK	[N/rpm]	Relación empuje - velocidad angular del rotor
Relación Q - ω^2	thrusterB	[mN/rpm^2]	Relación momento - cuadrado de ω
Sentido de rotación	thrusterRotation	[-]	Sentido de rotación del rotor
Revoluciones máximas	thrusterMaxRPM	[rpm]	Revoluciones por minuto máximas del rotor
$TSFC$	TSFC	[g/(Ns)]	Consumo específico de combustible por empuje

Tabla 12: Parámetros de *lookup_table_thrust_config*

Parámetros de *lookup_table_thrust_config*			
Ruta:	configuration/thrust/simple_thrust_config		
Parámetro	**Variable**	**Unidades**	**Descripción**
Sentido de rotación	thrusterRotation	[-]	Sentido de rotación del rotor
Tabla de empuje	Thrust_table	[N]	Valores de empuje según posición del acelerador
Breakpoints de empuje	Thrust_BP	[-]	*Breakpoints* de Thrust_table
Tabla de momento	Torque_table	[N]	Valores de momento según posición del acelerador
Breakpoints de momento	Torque_BP	[-]	*Breakpoints* de Torque_table
Tabla de consumo	FuelC_table	[N]	Valores de consumo según posición del acelerador
Breakpoints de consumo	FuelC_BP	[-]	*Breakpoints* de FuelC_table

Tabla 13: Parámetros de *mass_inertia_config*

Parámetros de *mass_inertia_config*			
Ruta:	configuration/massInertia/mass_inertia_config		
Parámetro	**Variable**	**Unidades**	**Descripción**
Masa	mass	[kg]	Masa inicial total de la aeronave
Inercia	inertia	[kg m^2]	Tensor de inercia inicial de la aeronave

Tabla 14: Parámetros de *package_delivery_config*

Parámetros de *package_delivery_config*			
Ruta:	configuration/massInertia/mass_inertia_config		
Parámetro	**Variable**	**Unidades**	**Descripción**
Masa con paquete	mass	[kg]	Masa inicial total de la aeronave
Inercia con paquete	inertia	[kg m^2]	Tensor de inercia inicial de la aeronave
Masa sin paquete	mass_delivery	[kg]	Masa de la aeronave después de la entrega
Inercia sin paquete	inertia_delivery	[kg m^2]	Tensor de inercia de la aeronave después de la entrega

Tabla 15: Parámetros de *fuel_consumption_config*

Parámetros de *fuel_consumption_config*			
Ruta:	configuration/massInertia/fuel_consumption_config		
Parámetro	**Variable**	**Unidades**	**Descripción**
Masa inicial	mass	[kg]	Masa inicial total de la aeronave
Inercia inicial	inertia	[kg m^2]	Tensor de inercia inicial de la aeronave (constante)
Masa sin combustible	mass_empty	[kg]	Masa de la aeronave sin combustible

Tabla 16: Parámetros de *wmm1mag_config*

Parámetros de *wmm1mag_config*			
Ruta:	configuration/environment/wmm1mag_config		
Parámetro	**Variable**	**Unidades**	**Descripción**
Medida inicial	initialMag	[T]	Campo magnético de la posición inicial

Tabla 17: Parámetros de *constant_wind_config*

Parámetros de *constant_wind_config*			
Ruta:	configuration/environment/constant_wind_config		
Parámetro	**Variable**	**Unidades**	**Descripción**
Velocidad del viento	windVelocity	[m/s]	Vector velocidad del viento, en coordenadas Tierra

Tabla 18: Parámetros de *actuators_config*

Parámetros de *actuators_config*			
Ruta:	configuration/actuators/actuators_config		
Parámetro	**Variable**	**Unidades**	**Descripción**
ω_n acelerador	wnThrottle	[rad/s]	ω_n FTF del acelerador
ζ acelerador	zThrottle	[-]	ζ FTF del acelerador
ω_n superficies	wnSurfaces	[rad/s]	ω_n FTF de superficies de control
ζ superficies	zSurfaces	[-]	ζ FTF de superficies de control
ω_n *tilt* del motor	wnThrustDir	[rad/s]	ω_n FTF de *tilt* del motor
ζ *tilt* del motor	zThrustDir	[-]	ζ FTF de *tilt* del motor

Tabla 19: Parámetros de *sensors_config*

Parámetros de *actuators_config*			
Ruta:	configuration/sensors/sensors_config		
Parámetro	**Variable**	**Unidades**	**Descripción**
Posición x Veronte	Veronte_X	[m]	Posición x de Veronte, ejes cuerpo
Posición y Veronte	Veronte_Y	[m]	Posición y de Veronte, ejes cuerpo
Posición z Veronte	Veronte_Z	[m]	Posición z de Veronte, ejes cuerpo
Número de semana	week_number	[-]	Número de semana GNSS
Segundos iniciales	initialSeconds	[s]	Segundos de semana GNSS iniciales
Precisión horizontal GNSS	GNSS_hr_accu	[mm]	Precisión horizontal del GNSS
Precisión vertical GNSS	GNSS_vt_accu	[mm]	Precisión vertical del GNSS
Precisión velocidad GNSS	GNSS_v_accu	[mm/s]	Precisión de velocidad del GNSS
Dirección Pitot	pitot_dir	[-, -, -]	Vector dirección tubo Pitot
Ruido p_{inf}	staticPressure_noise	[dB]	Potencia de ruido de p_{inf}
Ruido q_{inf}	dynamicPressure_noise	[dB]	Potencia de ruido de q_{inf}
Ruido posición GNSS	GNSSPosition_noise	[dB]	Potencia de ruido de la posición GNSS
Ruido altitud GNSS	GNSSAltitude_noise	[dB]	Potencia de ruido de la altitud GNSS
Ruido velocidad GNSS	GNSSVelocity_noise	[dB]	Potencia de ruido de la velocidad GNSS
Ruido magnetómetro	magnetometer_noise	[dB]	Potencia de ruido del magnetómetro
Ruido IMU	IMU_noise	[dB]	Potencia de ruido de la IMU
Offset temperatura	offsetT	[K]	*Offset* de temperatura de los sensores

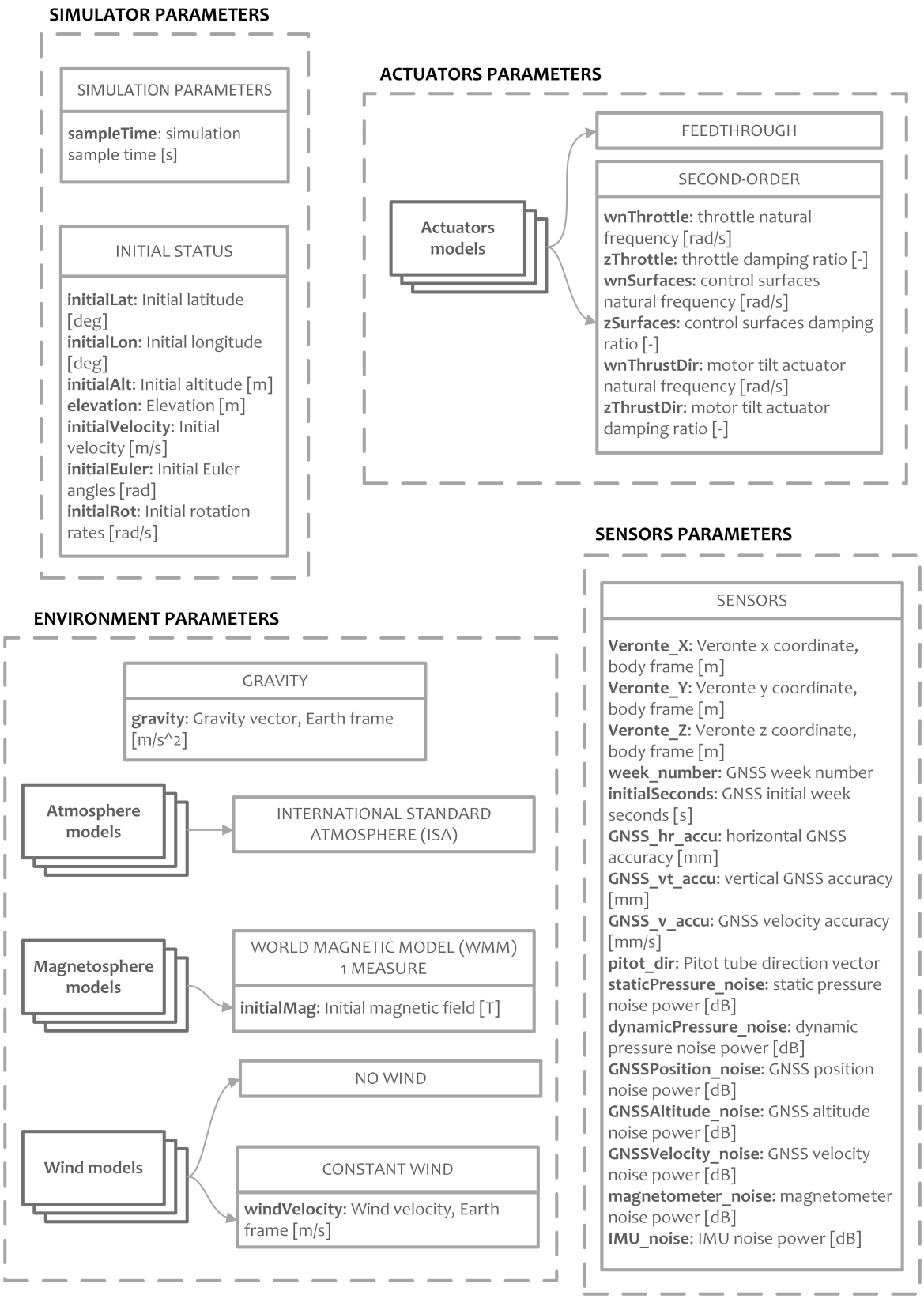

Figura 6: Esquema de parámetros requeridos (simulador, actuadores, entorno y sensores)

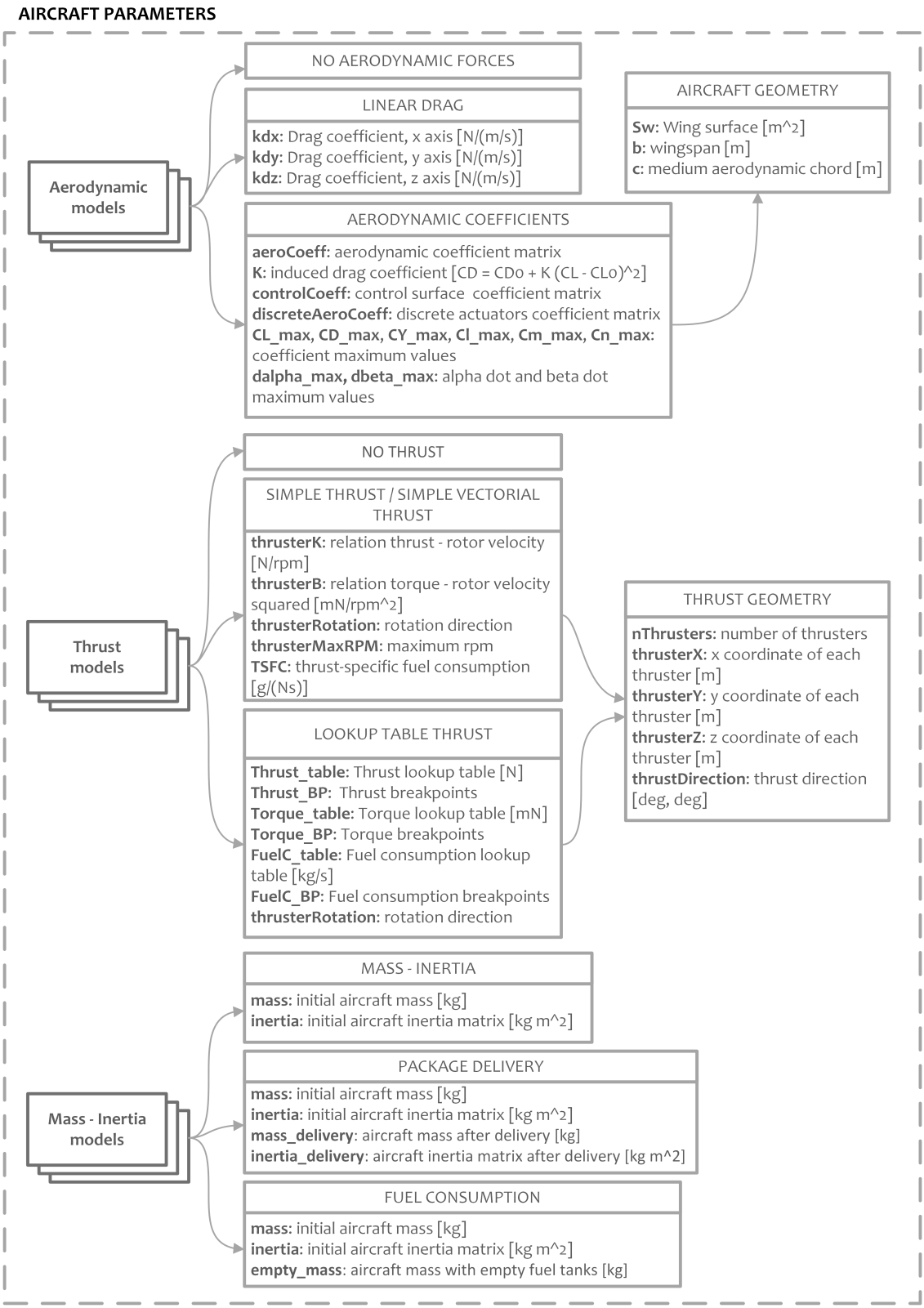

Figura 7: Esquema de parámetros requeridos (aeronave)

2.5. Ejecución de una simulación

Una vez terminada la configuración, según el modelo principal abierto (*main* permite el funcionamiento del simulador de forma aislada y *mainSIL* tiene conectado al simulador el bloque Veronte para simulaciones SIL), la simulación está lista para ser ejecutada. Presionando el botón *Run* de Simulink, comenzará la simulación. En caso de que se trate de una simulación junto con Veronte, debería aparecer un nuevo autopiloto en Link.

Un dato importante sobre el funcionamiento de EFS es que se trata de un *Enabled model*. Este tipo de modelos de Simulink presentan una entrada especial, un puerto ubicado en la parte superior del bloque, como se aprecia en la Figura 8. A través de este puerto se controla el funcionamiento del bloque, de forma que mientras la señal de esta entrada sea menor o igual que cero, el simulador se encontrará en estado de *standby*, enviando por las salidas únicamente su posición inicial. En cuanto la señal enviada por el puerto sea mayor que cero, el simulador comenzará a calcular el movimiento de la aeronave. Esto permite ejecutar Simulink para realizar conexiones con el autopiloto (bloque Veronte) y hacer comprobaciones sin que la aeronave comience a moverse.

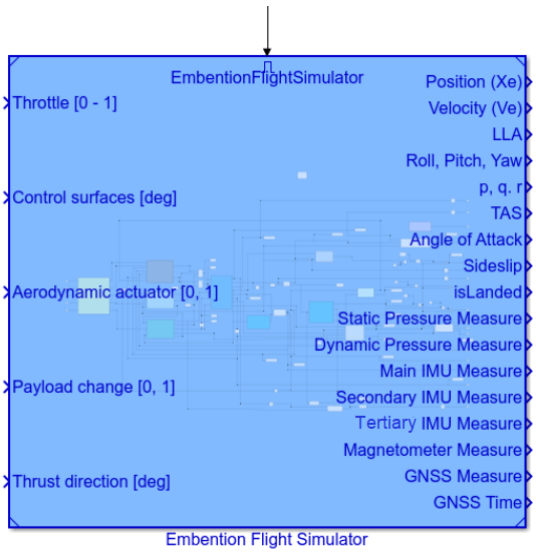

Figura 8: Bloque EFS con puerto *enabled*

2.6. Entradas y salidas del bloque

En las Tablas 20 y 21 se recogen las entradas y salidas del bloque EFS. Cabe destacar que las salidas de los sensores están totalmente adaptadas para funcionar directamente con el bloque Veronte, es decir, siguen el formato indicado en el manual de SIL disponible en Embention Manuals. En caso de que se quiera conocer alguna variable o estado del simulador, se recomienda incluir un *To Workspace* o *To File* dentro del bloque EFS y realizar el análisis de la variable en la etapa de post-proceso. Por otro lado, algunas salidas son estados de interés de la aeronave para su representación u otros usos.

Tabla 20: Entradas al bloque *Embention Flight Simulator*

Entrada	Formato	Descripción
Throttle	De 0 a 1	Vector con la posición del acelerador de cada motor
Control surfaces	[deg]	Vector con la deflexión de cada superficie de control
Aerodynamic actuator	0:OFF, 1:ON	Actuadores discretos aerodinámicos
Payload change	0:OFF, 1:ON	Cambio de carga de pago
Thrust direction	[deg]	Matriz con las direcciones de empuje (*tilt*)

Las entradas representan los diferentes controles de la aeronave. Dependen del modelo empleado y del tipo de aeronave, por ejemplo, en esta versión del simulador, *Payload change* es ignorado por el simulador siempre que no se emplee el modelo *Package delivery*. En caso de crear nuevos modelos, se pueden añadir otras entradas. En la Tabla 22 se relacionan los modelos y las entradas empleadas. Corresponde al usuario, antes de realizar las simulaciones, adaptar las entradas al formato requerido por el simulador.

Tabla 21: Salidas del bloque *Embention Flight Simulator*

Salida	**Formato**	**Descripción**
Position (Xe)	[m, m, m]	Posición en sistema de referencia Tierra
Velocidad (Ve)	[m/s, m/s, m/s]	Velocidad en sistema de referencia Tierra
LLA	[deg, deg, m]	Latitud, Longitud, Altitud
Roll, Pitch, Yaw	[rad, rad, rad]	Alabeo, cabeceo y guiñada
TAS	[m/s]	*True Airspeed*, velocidad relativa con el aire
Angle of attack	[rad]	Ángulo de ataque (α)
Sideslip	[rad]	Ángulo de derrape (β)
isLanded	0:OFF, 1:ON	1 en caso de que la aeronave toque el suelo
Static Pressure Measure	Equivalente a Veronte SIL	Medida de presión estática
Dynamic Pressure Measure	Equivalente a Veronte SIL	Medida de presión dinámica
Main IMU Measure	Equivalente a Veronte SIL	Medida de IMU principal de Veronte
Secondary IMU Measure	Equivalente a Veronte SIL	Medida de IMU secundaria de Veronte
Tertiary IMU Measure	Equivalente a Veronte SIL	Medida de IMU terciaria de Veronte
Magnetometer Measure	Equivalente a Veronte SIL	Medida del magnetómetro
GNSS Measure	Equivalente a Veronte SIL	Medida de posición GNSS
GNSS Time	Equivalente a Veronte SIL	Medida de tiempo GNSS

Tabla 22: Entradas empleadas por cada modelo

Entrada	**Modelos que la emplean**
Throttle	Empuje simple, Empuje vectorial simple, Empuje *Lookup table*
Control surfaces	Coeficientes aerodinámicos
Aerodynamic actuators	Coeficientes aerodinámicos
Payload change	Reparto de paquetes
Thrust direction	Empuje vectorial simple

3. Manual de desarrollo de EFS

3.1. Modificación del proyecto de Simulink

El proyecto de Simulink permite añadir las carpetas necesarias para el funcionamiento correcto del simulador al *path* de Matlab. Además, ejecuta los archivos que se requieran al abrir y cerrar el proyecto. Todas estas opciones se pueden modificar en la cinta de herramientas del IDE de Matlab, en la pestaña *Simulink Project*.

Se debe tener en cuenta que al crear un nuevo archivo o carpeta, éste se encuentre dentro del proyecto, en la vista *Files* del proyecto de Simulink.

3.2. Creación de nuevos modelos

Para crear un nuevo modelo para el simulador se debe generar el modelo de Simulink, un archivo de extensión *.mdl* o *.slx*, en el cual se encuentra el diagrama de bloques. En los bloques que lo requieran se pueden emplear variables a las que posteriormente se les asignará un valor en un archivo de configuración del modelo. Es necesario tener en cuenta que el simulador funciona de forma discreta, lo cual puede afectar al desarrollo del modelo (por ejemplo, a la hora de incluir bloques de integral o derivada). Además, para evitar posibles errores en el futuro, se recomienda indicar en los bloques que lo requieran que su tiempo de muestreo es *sampleTime* (*sampleTime* es una variable, indicada en el lanzador, donde se indica el tiempo de muestreo del simulador). En principio, si no se realiza este paso, no se deberían causar problemas, pero en caso de que Simulink indique errores derivados del tiempo de muestreo, se recomienda activar la opción *Display* → *Sample Time* → *Colors*, que asigna un color a cada *sample time*. De esta forma se puede apreciar fácilmente donde se encuentran las discrepancias en los tiempos de muestreo.

Una vez se tiene el modelo de Simulink listo, se puede integrar en el simulador. El modelo principal de EFS es el archivo *EmbentionFlightSimulator.slx*, donde se encuentran los bloques del tipo *Variant Subsystem*. Estos subsistemas de variantes son los bloques donde se almacenan todos los modelos relacionados, por ejemplo, en *Aerodynamics model* (Figura 9), se encuentran los tres modelos aerodinámicos presentes en esta versión, como se aprecia en la Figura 10.

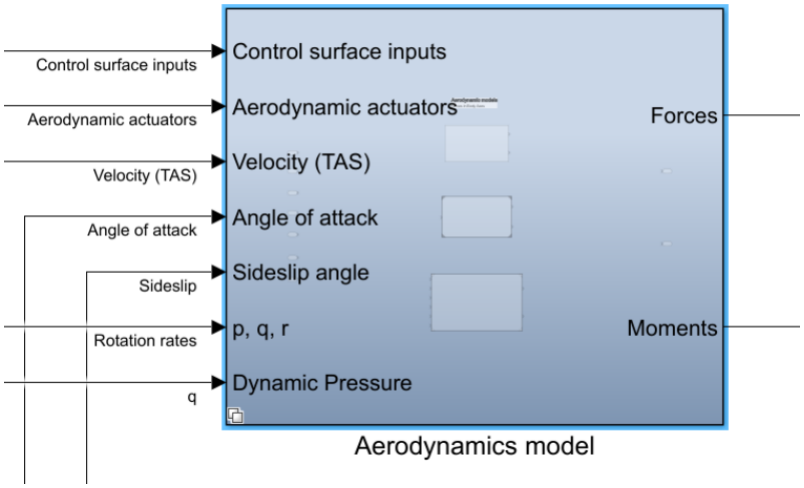

Figura 9: Bloque *Variant Subsystem* con modelos aerodinámicos

El modelo desarrollado se debe incluir en el *Variant Subsystem* correspondiente, es decir, si se trata de un nuevo modelo atmosférico, debe incluirse en el bloque *Atmosphere model*. Para incluir un modelo, primero se añade un bloque *Model* de la librería de bloques de Simulink. En este bloque se puede indicar el nombre del archivo que contiene el diagrama de bloques creado. Posteriormente, se le da un nombre descriptivo al bloque (es decir, el nombre que aparece debajo del bloque).

Una vez añadido el modelo al bloque de variantes, se deben añadir las entradas y salidas necesarias, en el caso de que no estén ya añadidas. Como se aprecia en la Figura 10, las entradas y salidas no están conectadas. Esto se debe a que la conexión la realiza Simulink de forma interna una vez se selecciona el modelo requerido. Siguiendo el ejemplo de los modelos aerodinámicos, cuando en el lanzador guarda la variable *Aero* con el valor 1, Simulink selecciona el modelo *No aerodynamic forces*. Además, ignora todas las entradas, puesto que no requiere ninguna, y envía sus salidas a través de los puertos *Forces* y *Moments*. La unión entre entradas y salidas se realiza a partir del nombre, es decir, la salida *Forces* de *No aerodynamic forces* se conecta con la salida *Forces*

del modelo de variantes porque su nombre coincide. De esta forma, *Aerodynamics model* funciona según se haya seleccionado un modelo entre todos los que tiene internamente.

Una vez fuera del subsistema de variantes (como en Figura 9), se debe configurar el bloque para que elija el modelo interno según una variable de Matlab, en este caso *Aero*. Se hace click derecho y se selecciona la opción *Block Parameters (Subsystem)*, apareciendo la ventana mostrada en la Figura 11. En la fila que correspondería al nuevo nuevo modelo, se añade un *Variant control*, por ejemplo, si se quisiera añadir un nuevo modelo de *drag* cuadrático con la velocidad, se podría llamar QUADRATIC_DRAG. Por último, se debe asignar una variable de selección (columna *Condition*). Este paso se realiza en el espacio de trabajo de Matlab, mediante un objeto Simulink.Variant. Como ya existen tres modelos aerodinámicos, se le asignaría la condición *Aero* == *4*, de forma que el comando de Matlab sería:

```
QUADRATIC_DRAG = Simulink.Variant('Aero == 4');
```

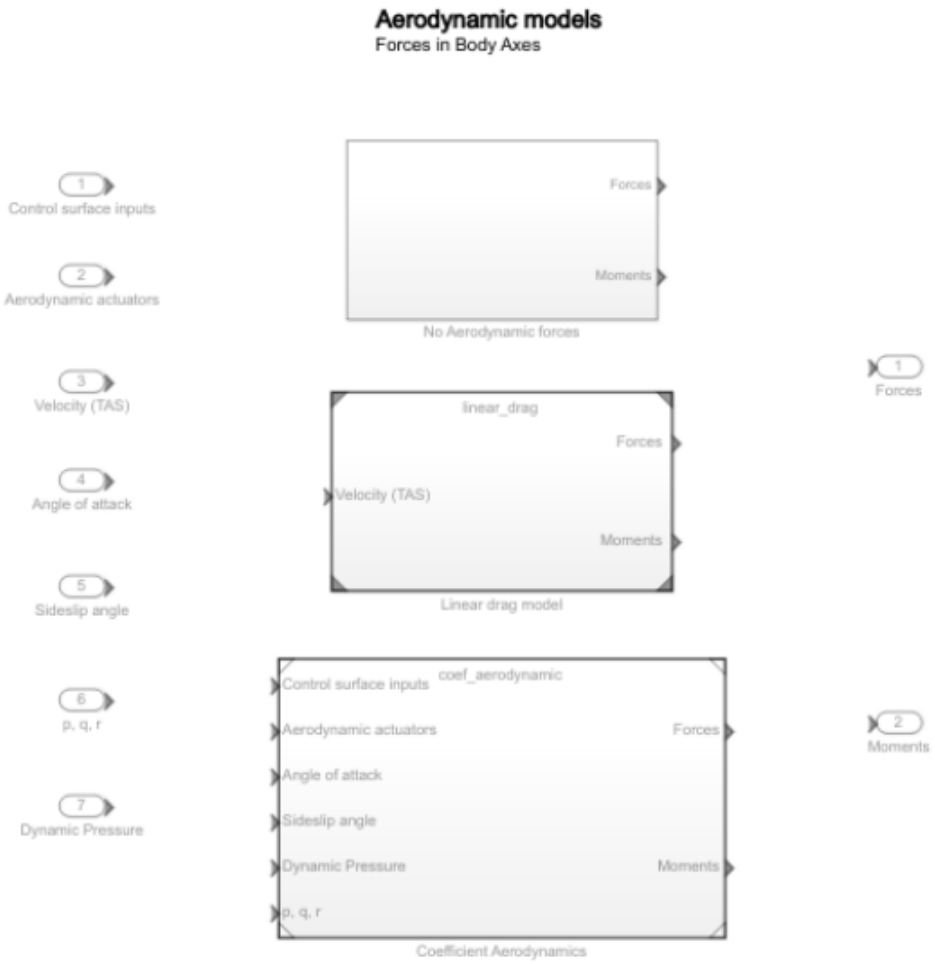

Figura 10: Modelos aerodinámicos dentro del bloque *Aerodynamics model*

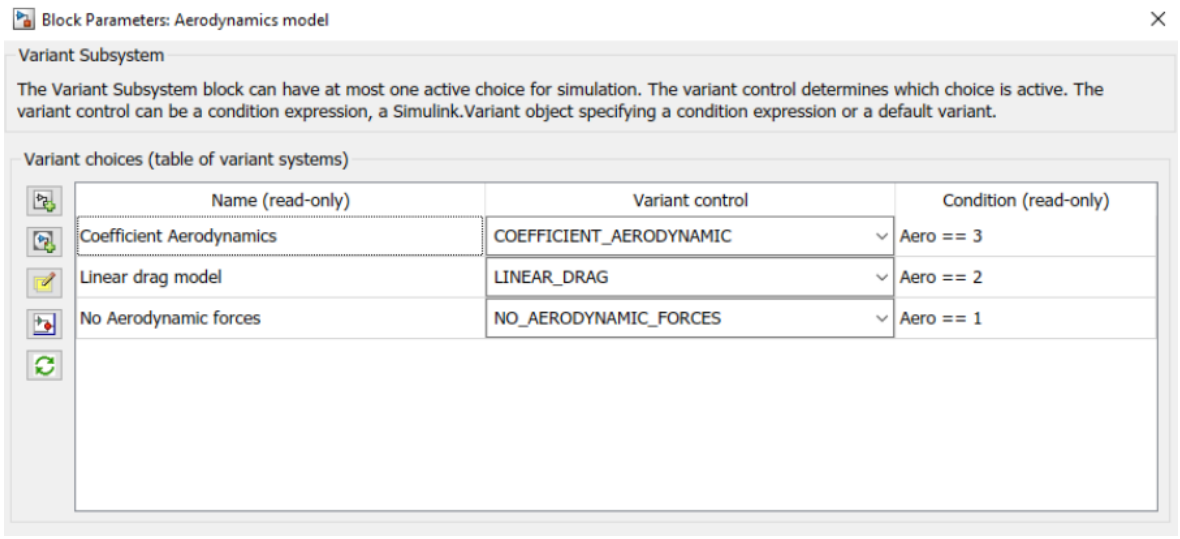

Figura 11: Parámetros del bloque *Variant Subsystem*

Por último, hay que crear un archivo de configuración que considere las variables que se han empleado durante la creación del modelo, si las hubiere. Por ejemplo, si nuestro modelo tuviera unos bloques que emplearan la variable *CD*, representando el coeficiente de resistencia de una aeronave, sería necesario crear un archivo .m en la carpeta de configuración de los modelos aerodinámicos. Si este archivo se llamara *quadratic_drag_config.m*, se podría añadir al lanzador del simulador junto con una condición que haga que se ejecute cuando el usuario emplee el nuevo modelo, de forma que el código quedaría así:

```
% Aerodynamic model
% 1: No aerodynamic forces
% 2: Linear drag model (no lift)
% 3: Coefficient-based aerodynamic forces and moments
% 4: Quadratic drag

Aero = 4;

% Model configuration
NO_AERODYNAMIC_FORCES = Simulink.Variant('Aero == 1');
LINEAR_DRAG = Simulink.Variant('Aero == 2');
COEFFICIENT_AERODYNAMIC = Simulink.Variant('Aero == 3');
QUADRATIC_DRAG = Simulink.Variant('Aero == 4');

switch Aero
    case 1
        fprintf('Aerodynamic model: No aerodynamic forces\n');
    case 2
        fprintf('Aerodynamic model: ');
        fprintf('Linear drag model (no lift)\n');
        run(['configuration/aerodynamic/'...
            'linear_drag_config']);
        if openScripts
            edit 'configuration/aerodynamic/linear_drag_config';
        end
    case 3
        fprintf('Aerodynamic model: ');
        fprintf('Aerodynamic forces and moments using coefficients\n');
        run(['configuration/aerodynamic/'...
```

```
        'coef_aerodynamic_config']);
    if openScripts
        edit 'configuration/aerodynamic/coef_aerodynamic_config';
    end
case 4
    fprintf('Aerodynamic model: ');
    fprintf('Quadratic drag\n');
    run(['configuration/aerodynamic/'...
        'quadratic_drag_config']);
    if openScripts
        edit 'configuration/aerodynamic/quadratic_drag_config';
    end
otherwise
    msgbox('Invalid aerodynamic model', 'Aerodynamic model', "error");
    check = false;
```

De esta forma, cuando la variable *Aero* fije su valor en 4 y las variables requeridas por el modelo estén en el espacio de trabajo de Matlab, el simulador empleará el nuevo modelo desarrollado, usando también las entradas que sean convenientes (la velocidad respecto al aire y la densidad, por ejemplo) y devolviendo como salida las fuerzas y los momentos aerodinámicos.

3.3. Nuevos modelos sugeridos

Una vez explicado el método para crear nuevos modelos, se van a exponer los diferentes modelos planteados y propuestos para futuras versiones:

- Masa e inercia variable
  En el caso de algunas aeronaves con grandes depósitos de combustible, el gasto de éste durante el vuelo provoca cambios considerables en la inercia. Se podría partir del modelo de consumo de combustible simple, relacionando de alguna forma masa e inercia. Otra opción sería tabular la inercia según la masa de la aeronave, a través de *lookup tables*.
- Drag cuadrático con la velocidad
  Si bien este modelo está en parte recogido por la aerodinámica basada en coeficientes, es interesante tener un modelo que contemple la resistencia aerodinámica como cuadrática con la velocidad, para multicópteros que vuelen a mayores Reynolds.
- Aerodinámica tabulada
  Las *lookup tables* permiten modelar el comportamiento aerodinámico no lineal de una aeronave (es decir, se podrían modelar las entradas en pérdida y otros efectos). La principal complejidad a la hora de desarrollar este modelo es asignar de antemano las dependencias de cada coeficiente y encontrar un balance entre precisión y complejidad a la hora de obtener y elaborar las tablas. Una buena primera aproximación es obtener cada coeficiente según un estado y una superficie de control, por ejemplo, el coeficiente de sustentación (C_L) podría depender del ángulo de ataque y de la deflexión del elevador. Esto permitiría modelar de forma más precisa una aeronave que empleando el método de coeficientes ya implementado, a costa de perder flexibilidad en el número y tipo de superficies de control.
- Modelos del Aerospace Blockset
  En el Aerospace Blockset de Simulink hay modelos ya hechos que pueden ser implementados de forma sencilla según se requiera. Por ejemplo, en modelos de empuje se puede incluir el turbofán. También existen una gran cantidad de bloques de modelos de entorno: de viento, ráfagas y turbulencias, modelos atmosféricos y magnetosféricos.

3.4. Creación de herramientas

Las herramientas son *scripts* de Matlab, almacenados en la carpeta *tools*, con funciones adicionales del simulador. Se pueden crear funciones auxiliares para facilitar el desarrollo del código, pero debe haber un archivo

ejecutable para asignarlo a un *shortcut* de Simulink. Esta operación se realiza en la cinta de herramientas de Matlab (Figura 5).

A continuación se muestran algunas herramientas sugeridas y planteadas en futuras versiones que pueden enriquecer el simulador:

- Generación de resúmenes de simulación.
  Esta herramienta generaría resúmenes de la simulación, almacenando la configuración del simulador y sus resultados.

- Guardado y carga de configuraciones.
  Se debería poder guardar las configuraciones empleadas y poder ser cargadas de forma rápida, evitando tener que modificar todos los archivos de configuración.

- Estimación de coeficientes aerodinámicos.
  La estimación de coeficientes aerodinámicos permitiría poder simular una aeronave de ala fija a partir de la geometría de sus superficies aerodinámicas. *USAF DATCOM* recoge fórmulas teóricas y experimentales para estimar estos coeficientes en etapas de diseño, por lo que podrían ser implementadas. Otra opción es integrar el software Tornado, un programa de Matlab que calcula estos paráemtros empleando el método *Vortex-Lattice* (malla de torbellinos). Tornado permite trabajar con cualquier geometría, siendo más flexible que los métodos de *DATCOM*, pensados para aeronaves con geometrías más convencionales. Además, se trata de software *open source*. Otra opción es implementar un código propio de Embention basado en *Vortex-Lattice Method*.

- Estimación de las características de una hélice.
  La herramienta permitiría obtener las características propulsivas de una hélice sobre la que no se tengan datos empleando la teoría conjunta de Elemento de Pala y de Cantidad de Movimiento. Para ello, sería necesario conocer la geometría de las hélices.

- Obtención de la inercia de la aeronave.
  A partir de la masa y de la geometría se podría estimar el tensor de inercia de la aeronave de forma simple. Además, permitiría obtener algún modelo que relacione la masa con la inercia, de forma que exista un modelo para aeronaves que varían su inercia durante el vuelo.

4. Desarrollos futuros a partir de la *release* v0.1

El objetivo de este apartado es trazar un plan de futuros desarrollos para el simulador a partir de la primera *release*. Algunos avances que se podrían hacer ya se han introducido a lo largo de este documento; otros, en cambio, ya se han incluido en la *release* (por ejemplo, la herramienta de estimación de características de la hélice o los modelos de masa e inercia variable). Los futuros desarrollos que se proponen son:

- Optimización del código.
  Es importante realizar un estudio sobre optimización de Simulink antes de emprender ninguna modificación. En general, se deben evitar bucles algebraicos, pasos temporales muy bajos, precisiones innecesariamente altas y llamadas al intérprete de Matlab (*Interpreted Matlab function*). Además, se debe asegurar que, después de realizar las pruebas necesarias, se eliminan los bloques *scope* usados durante el *debugging*. También se puede desconectar el bloque de representación si se necesita mayor velocidad de cálculo.

- Revisión de los parámetros empleados.
  Como se ha podido comprobar, el simulador emplea una gran cantidad de parámetros para funcionar. En Matlab 2018a, Simulink dificultaba el uso de estructuras (*structs*) de Matlab como argumento o parámetro del bloque *Matlab function* (ya que debían ser tratados como objetos *Bus* de Simulink), por lo que se optó por usar variables del *workspace*. Para facilitar su comprensión, se han realizado esquemas de todos los parámetros del simulador. Sin embargo, con las nuevas posibilidades que ofrece Matlab 2020a, se debería revisar el formato en el que se presentan dichos parámetros.
  Una opción es agrupar las variables en estructuras, de forma que se tuvieran tres estructuras: *simulator*, *aircraft* y *environment* (los nombres son orientativos). La primera recogería la configuración del simulador,

la segunda, los modelos de la aeronave y la tercera, los modelos de entorno. Esto facilitaría, por ejemplo, guardar el modelo de aeronave de forma sencilla (usando la función *save* de Matlab para almacenar la estructura *aircraft*), mantendría el *workspace* más limpio y dificultaría que algunas variables se sobreescribieran por error.
Otras opciones que se deberían estudiar antes de realizar cambios en el formato de los parámetros son el uso de *Data Dictionaries* de Simulink, el uso de clases para crear objetos (por ejemplo, un objeto con el modelo de la aeronave, con sus atributos públicos y privados, que evitarían errores indeseados durante la manipulación de los modelos), si Simulink tiene algún tipo de compatibilidad con este tipo de datos y la posibilidad de establecer los parámetros del un bloque de Simulink de forma programática (véase *set_param*).

- Modelado del suelo.
  Sería interesante para desarrollos futuros modelar las fuerzas del suelo sobre la aeronave, de forma que se puedan simular etapas de despegue en aeronaves de ala fija. En la versión actual se calcula la energía de impacto, pero no se evita que la aeronave siga bajando una vez toca el suelo. A partir de la energía se puede estimar la fuerza que realizaría el suelo sobre la aeronave cuando entran en contacto.

- Implementación de *Digital DATCOM*.
  El bloque DATCOM del Aerospace Blockset de Simulink es de gran utilidad si se tiene el *software* de DATCOM y se conoce como emplearlo. Embention cuenta con este *software*, por lo que se recomienda implementar el bloque para modelar de forma eficaz y simple el comportamiento de aeronaves de ala fija y geometrías convencionales.

- Uso de *Vortex-Lattice Method*.
  Como ya se ha comentado, el método de la malla de torbellinos obtiene aproximaciones para geometrías arbitrarias. En caso de que se requiera modelar una aeronave poco convencional, se podría desarrollar una herramienta (o adaptar el *software* Tornado) para estimar coeficientes empleando este método.

- Modelos de ruido y de interferencia.
  Para poder utilizar el simulador como herramienta de *testing*, es interesante desarrollar ciertas herramientas que permitan provocar errores en el sistema. Por ejemplo, serían útiles modelos de ruido (más allá del ruido blanco ya aplicado) para los sensores, modelos de sesgos (*bias*) e interferencias del modelo magnético.

- Modelos de ráfaga y turbuencia.
  En Aerospace Blockset se encuentran muchos modelos de ráfaga de viento y turbulencia, empleados en tareas de certificación, que podrían ser incluidos en el simulador para realizar ensayos.

- Integración con HIL.
  El simulador ha sido probado en conjunto con el bloque SIL de Veronte. El siguiente paso para integrarlo en la familia Veronte sería realizar pruebas con un autopiloto real, realizando simulaciones *Hardware In the Loop*.

- Modelado de helicópteros.
  En la versión 0.1 se incluye un modelo *Lookup table* de control del ángulo de paso colectivo. Si se desarrollara un modelo de paso cíclico, teniendo en cuenta la dinámica de vuelo de los helicópteros (complejidad del vuelo de avance, efectos de entrada en pérdida en punta de pala, modelización del rotor antipar, etc.) se podría ampliar el simulador a helicópteros.

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

La Tabla 1 recoge el grado de relación del Trabajo de Fin de Máster con los Objetivos de Desarrollo Sostenible.

Tabla 1: Grado de relación del trabajo con los ODS

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

Descripción de la alineación del TFG/TFM con los ODS con un grado de relación más alto:

- ODS 9. *Embention Flight Simulator* tiene como objetivo facilitar la integración de autopilotos para aeronaves autónomas. Se está produciendo una gran innovación en el sector, ampliando el número de tareas que son realizadas por drones, y el simulador desarrollado es un eslabón más de este proceso de innovación.
- ODS 8. El uso de aeronaves autónomas como sustituto de otras máquinas con un coste mayor, como avionetas, favorece el desarrollo económico.
- ODS 11. Muchas de las aeronaves autónomas son eléctricas y tienen un impacto menor en el medio ambiente que sus alternativas. Por ejemplo, el reparto de paquetes con drones eléctricos es un método más sostenible que el uso de furgones de reparto.