LIBIS Flight Simulator.

Documentación

Departmento de Aeronaves y Vehículos Espaciales
Universidad Politécnica de Madrid

Pérez Sancha, Carlos
Suárez Tapia, Uxio

Mayo, 2017

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Objetivo

El presente documento trata de ser una breve documentación de la versión preliminar del simulador de vuelo desarrollado en el transcurso de las prácticas curriculares para el RPAS "*LIBIS*" diseñado por el departamento de Aeronaves y Vehículos Espaciales de la Universidad Politécnica de Madrid (UPM).

Sin embargo, no hay que olvidar que el simulador todavía se encuentra en fase de diseño, por lo que todavía no se encuentra implementado en su totalidad y continúan existiendo numerosos aspectos que deben ser depurados. Esto implica que la presente documentación no es definitiva, y deberá ser actualizada según se añadan o modifiquen aspectos de éste.

1.2. Filosofía de diseño

El simulador se encuentra parcialmente desarrollado en MATLAB y parcialmente en Simulink bajo el siguiente criterio.

La parte de importación y validación de datos, cálculo matemático e interfaz con programas externos (de ser necesaria) se realiza en MATLAB, elaborándose de esta forma una estructura llamada "LD" (LIBIS Data) el Workspace con todos los datos necesarios a la que posteriormente accederá Simulink para leer los valores de cada parámetro. Por otra parte, la definición de los diferentes bloques que componen el simulador, las relaciones entre ellos, la integración en el dominio del tiempo y el postprocesado y visualización, se encuentra programada en Simulink.

Este criterio permite modificar fácilmente los datos de entrada, al encontrarse éstos en texto plano, pudiendo de esta forma probar fácilmente nuevas configuraciones, motores, hélices, ... con apenas mínimos cambios de código, a la vez que se hace uso de las ventajas de Simulink.

1.3. Descripción del modelo

El modelo completo, así como las conexiones entre sus componentes se puede observar en la figura 1.1. Más adelante se explicarán cada uno de los bloques por separado, detallando el contenido de éstos, los scripts de MATLAB relacionados donde se importan los datos del modelos correspondientes y cualquier otro aspecto del mismo que sea considerado de interés para un futuro usuario/desarrollador.

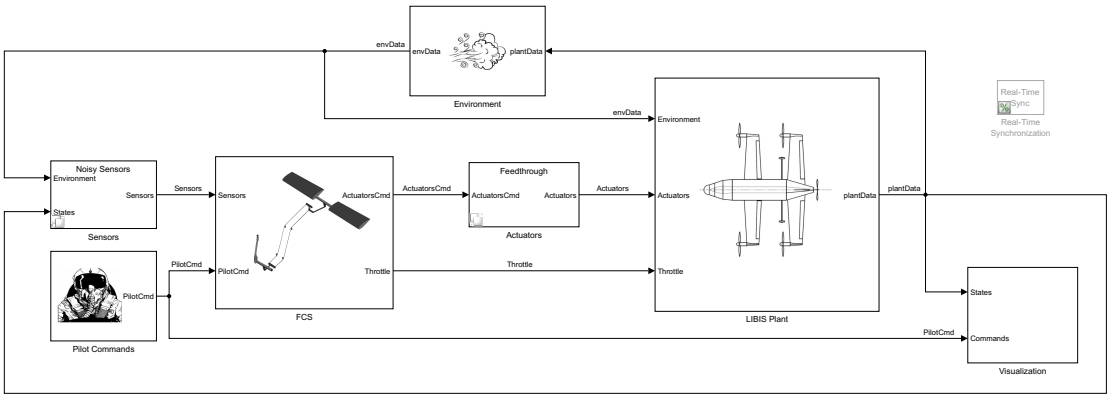

Figura 1.1: Esquema global del modelo

A la hora de inicializar el modelo, se ejecutan de forma automática los scripts "*/utilities/setUpProject.m*" que se encarga de añadir al Path las carpetas pertenecientes al modelo y el script "*/dataImport/initializeVars.m*" encargado de definir el tiempo de sampleo, arrancar el script de importación de datos "*/dataImport/loadData.m*" y cargar los valores iniciales de algunas variables y estados a usar en las condiciones iniciales de Simulink.

Capítulo 2

Entorno Atmosférico

Para obtener los valores de las variables dependientes del entorno en el que se encuentra en cada instante la aeronave, se ha implementado un bloque que simule el entono atmosférico y que proporcione dichos valores en función de la posición, geometría y actuaciones de la aeronave.

El modelo generado se puede ver en la figura 2.1, y se descompone a su vez en los subsistemas encargados de proporcionar las reacciones del terreno con la aeronave, los modelos de viento atmosférico, el modelo de atmósfera estandar, de gravedad y de campo magnético.

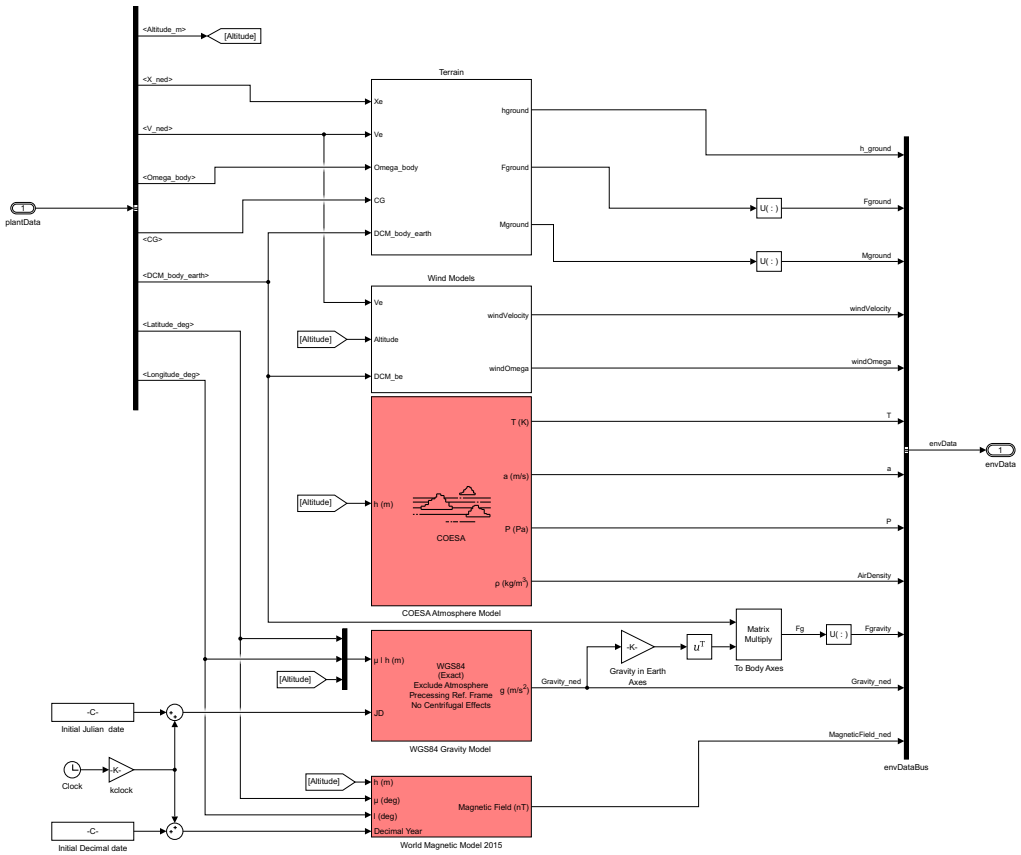

Figura 2.1: Modelo de entorno atmosférico

2.1. Modelo de reacción con el terreno

Para detectar la situación de contacto de la aeronave con el terreno y calcular las reacciones que éste genera sobre la aeronave se definen en el archivo "*/dataImport/LandingGear/loadLandingGear.m*" las posiciones de las tres patas de la aeronave a partir de la posición longitudinal de éstas, la posición vertical del punto de contacto respecto del centro de gravedad y de la ballesta del tren principal (track), tal y como se muestra en la figura 2.2.

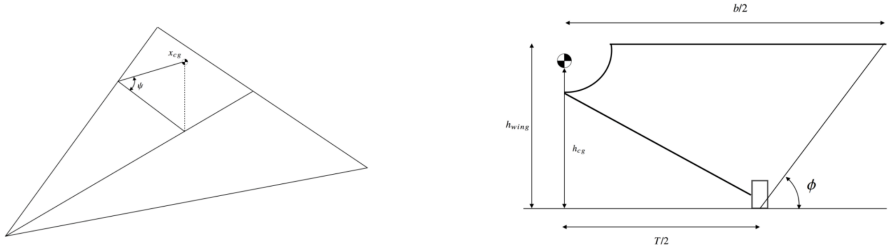

Figura 2.2: Esquema del tren de aterrizaje

A partir de estas variables, se modela el suelo como un elemento elástico que reacciona de forma proporcional a la distancia que se introducen bajo tierra los puntos de control de las patas y a la velocidad con la que se introducen, ejerciendo una reacción nula en caso de que no se encuentren en contacto las patas y el suelo, tal y como se muestra en la figura 2.3.

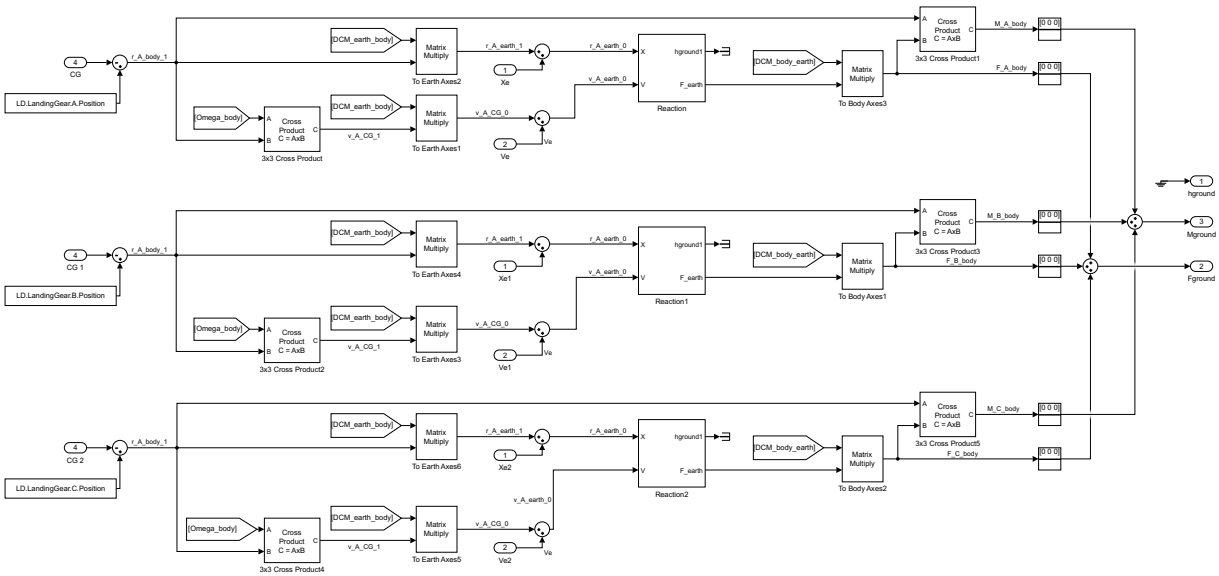

Figura 2.3: Modelo de reacción del terreno

Es importante destacar que éste es un modelo limitado con gran márgen de mejora, donde el suelo únicamente proporciona una reacción normal, sin existir componente tangencial. A su vez, sólo se detecta contacto con las patas, pudiéndose producir el caso de que cualquier otra parte de la aeronave entre en contacto con el suelo, sin que en ese caso se produzca reacción de ningun tipo.

2.2. Modelo de viento atmosférico

Para modelizar el viento, se ha realizado una composición de los modelos de viento cortante, turbulento y ráfaga, a partir de los propios modelos que incorporla la Aerospace Toolbox de Simulink, tal y como se muestra en la figura 2.4.

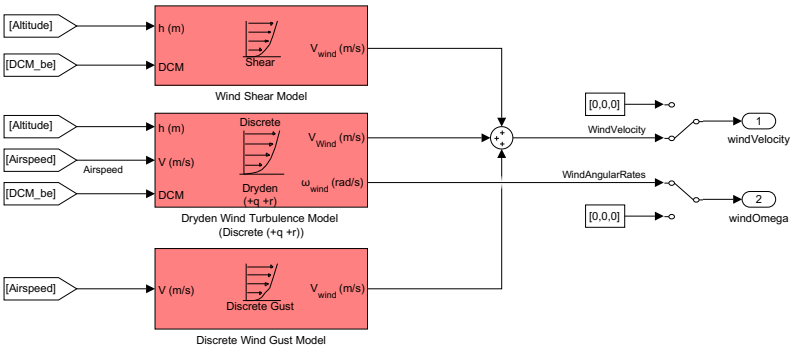

Figura 2.4: Modelos de viento atmosférico

2.3. Modelos de atmósfera, campo gravitatorio y campo magnético

A su vez, para los modelos de atmósfera estandar, campo gravitatorio y campo magnético se incorporan los implementados en la Aerospace Toolbox, para a la posición y altitud a la que se encuentre la aeronave, así como al instante temporal en que se realice la simulación, el cual se define en "*/dataImport/initializeVars*".

2.4. Bus de entorno atmosférico

Por último, todos los datos atmosféricos se almacenan en un bus de datos no virtual denominado "envDataBus", el cual se encuentra definido en "*/dataImport/Buses/loadEnvironmentBus.m*" y se ejecuta a través de "*/dataImport/loadData.m*" de forma automática al inicializar el proyecto.

Capítulo 3

Modelo de la Aeronave

Este bloque es con diferencia el más complejo y el que más explicación necesita, por lo que se tratará de poner especial detalle en los aspectos más complejos, así como las diferentes funcionalidades adiccionales que se han dejado implementadas y los aspectos inacabados que requieren de revisión.

En este bloque, mostrado en la figura 3.1, se calculan las fuerzas que actúan sobre la aeronave, se integran en el tiempo y se obtienen las variaciones de posición y actitud que experimenta la aeronave.

Este procedimiento se puede separar en varias partes claramente diferenciadas, véase, una correspondiente al cálculo de las fuerzas y momentos aerodinámicos a partir del estado de la aeronave y las derivadas de estabilidad, otra encargada del cálculo del sistema propulsivo y las fuerzas y momentos que éste genera, otro bloque encargado del cálculo de la variación del centro de gravedad y los momentos de inercia, un subsistema encargado de la integración de estas fuerzas en el dominio del tiempo y finalmente una última parte donde se estructuran los datos obtenidos y se almacenan en un bus de estados denominado "*PlantDataBus*" para facilitar su comunicación con otros bloques.

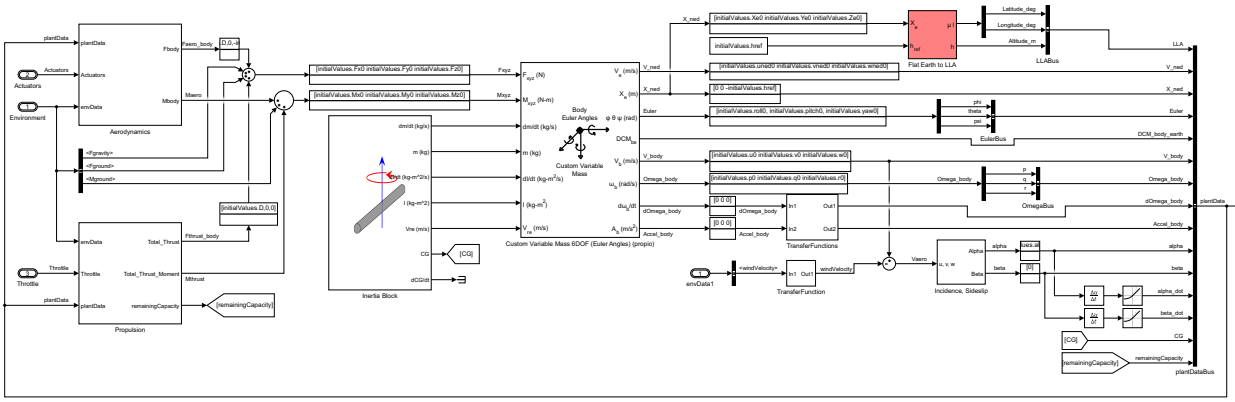

Figura 3.1: Vista general del modelo de la planta de la aeronave

3.1. Aerodinámica

Para el cálculo de las fuerzas y momentos aerodinámicos se ha implementado el esquema mostrado en la figura 3.2, que consiste en la obtención de los valores de los coeficientes aerodinámicos en ejes estabilidad a partir de los valores de las derivadas de estabilidad correspondientes a los valores instantáneos de α, β, altitud, posición del centro de gravedad y deflexión existente de las superficies aerodinámicas.

Por último, se dimensionalizan dichos coeficientes aerodinámicos obtenidos previamente con la presión dinámica y las magnitudes de referencia declaradas en "*/dataImport/Stability/loadStabilityData.m*" y se obtienen las fuerzas y momentos en cada eje.

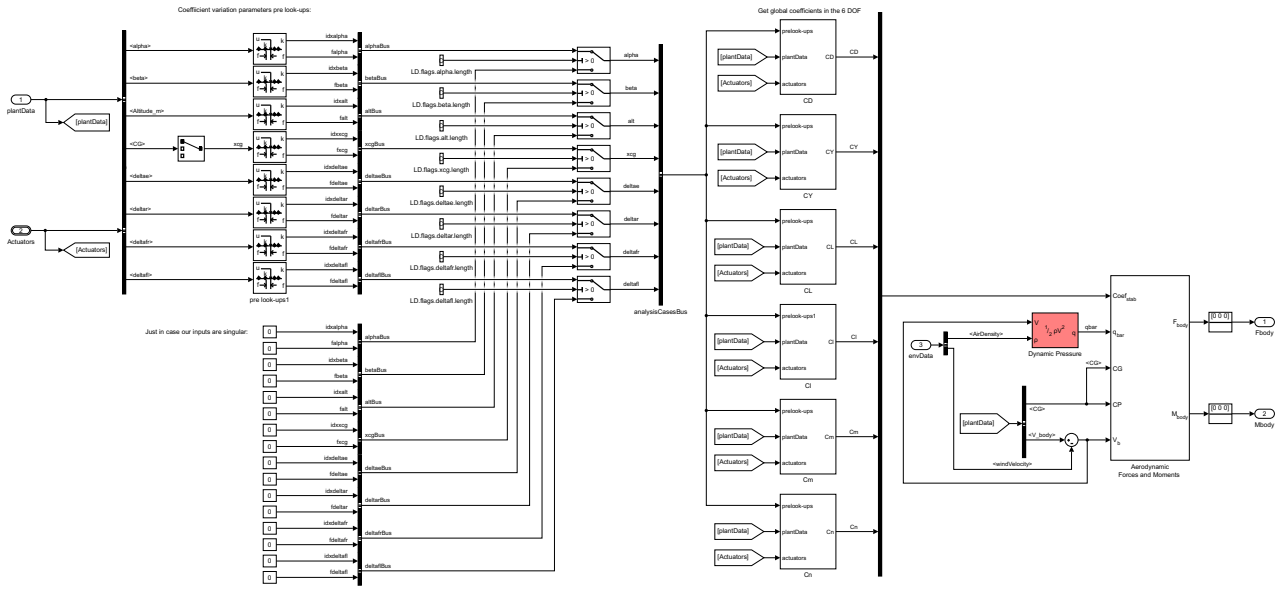

Figura 3.2: Esquema del modelo aerodinámico

3.1.1. Coeficientes de Estabilidad

El proceso de definición e importación de los coeficientes de estabilidad, o derivadas de estabilidad, se ha implementado de una forma, a priori poco intuitiva, pero que se ha considerado que aporta una gran versatilidad y logra solventar los problemas asociados a acoplar en el mismo proceso de cálculo la aerodinámica del vuelo como ala fija y como quadrotor, por lo que este paso se explicará en detalle.

En los simuladores de aeronaves de ala fija, es habitual seguir el procedimien-

to indicado anteriormente de definir las derivadas de estabilidad, calcular con ellas los coeficientes aerodinámicos y a partir de éstos las fuerzas y momentos aerodinámicos. Sin embargo, este procedimiento sólo es válido cuando las hipótesis hechas al calcular estas derivadas se siguen cumpliendo, es decir, ángulos de ataque y de resbalamiento pequeños, etc. Si bien, en el vuelo habitual de una aeronave éstas hipótesis se cumplen, ésto ya no es así cuando se realiza un vuelo como quadrotor, situación en la que el ángulo de entrada de la velocidad proporciona valores de α y β que distan mucho de ser pequeños.

A su vez, para modelizar la posibilidad de entrada en pérdida, o para poder simular las variaciones de actuaciones en función del centrado de la aeronave, es necesario definir que las derivadas de estabilidad dejen de ser constantes y que experimenten variaciones con una serie de parámetros. Pese a que en la vida real los parámetros de los que dependen son numerosos, de momento se han seleccionado los ocho parámetros siguientes: ángulo de ataque (α), ángulo de resbalamiento (β), altitud de vuelo, posición longitudinal del centro de gravedad ($\mathrm{x_{cg}}$), deflexión del estabilizador horizontal (δ_e), deflexión del timón de dirección (δ_r), deflexión del flaperon derecho (δ_{fr}) y deflexión del flaperon izquierdo (δ_{fl}). Sin embargo, como se verá, resulta extraordinariamente sencillo variar éstos, o cambiar el número de parámetros de los que depende dicha variación.

Toda la definición relacionada con la aerodinámica o la estabilidad se realiza desde el sript "*/dataImport/Stability/loadStabilityData.m*". El él se deben declarar las variables de referencia, véase, superficie, cuerda, envergadura y posición longitudinal del borde de ataque. A su vez, se definirán en el campo "*LD.Stability.analysisCases*" el nombre de los parámetros de los que se ha decidido que dependan los coeficientes de estabilidad y en el campo "*LD.Stability.Coeffs*" los nombres de los diferentes coeficientes.

A continuación se declararán, en los campos "*LD.XXX*", los valores de cada uno de los parámetros para los que se dispone de información ordenados de menor a mayor, donde "*XXX*" hace referencia al nombre definido para dicho parámetro, de forma que posteriormente Simulink interpolará linealmente los valores de cada coeficiente en función del valor instantáneo de cada parámetro entre los dos valores más cercanos de los aquí definidos. Por ejemplo, si uno de los parámetros es el ángulo de ataque y hemos definido que su nombre sea "*alpha*", se deberá definir un vector con los valores de ángulo de ataque para los que se han calculado las derivadas de estabilidad y almacenarlo en el campo "*LD.alpha*". Es importante tener en cuenta que ningún parámetro puede quedarse sin definir, de forma que aunque en alguna situación no consideremos variación en ese parámetro, se debe indicar el valor para

el que han sido calculadas las derivadas de estabilidad.

Por último, para introducir los valores de cada derivada de estabilidad, se debe crear una matriz del mismo número de dimensiones que el número de parámetros definidos, respetando el orden definido en "*LD.Stability.analysisCases*", donde el tamaño de cada dimensión sea acorde a la longitud del vector de valores de dicho parámetro y almacenarla en "*LD.Stability.XXX*", donde "*XXX*" hace referencia al nombre definido para dicho coeficiente.

Es decir, siguiendo con el ejemplo anterior, si los valores definidos en "*LD.alpha*" son $[-\pi, -\pi/2, 0, \pi/2, \pi]$, y "*alpha*" es el primer y único parámetro de los ocho declarados que experimenta variaciones, la matriz definida en "*LD.Stability.CD0*" deberá ser de [5x1x1x1x1x1x1x1] donde el primer valor se corresponda con el primer valor definido de "*alpha*", el segundo con el segundo y así sucesivamente. Si en vez de experimentar variaciones únicamente el ángulo de ataque, se definen también 10 valores de variaciones con la altitud (definida la tercera en el campo "*LD.Stability.analysisCases*") la matriz de cada coeficiente deberá ser de [5x1x10x1x1x1x1x1] y cada uno de los 50 valores de cada coeficiente resultantes se deben colocar en su posición adecuada de la matriz.

Como realizar este proceso de colocación de cada coeficiente de forma manual es tremendamente costoso y muy propenso a cometer errores en cuanto se definan variaciones con varios parámetros no consecutivos, se ha elaborado el script "*/dataImport/Stability/fillStabilityCoeffs.m*" que se encarga de realizarlo de forma automática.

Para ello, se le deben pasar como argumentos: la estructura "*LD*" de la que leerá los parámetros definidos, un vector de campos de texto con los nombres de los parámetros que experimentan variaciones y la matriz con los valores de los coeficientes, donde la primera dimensión de ésta representa la variación con el primer parámetro definido en el segundo argumento, la segunda dimensión la variación con el segundo parámetro y así sucesivamente.

En el caso de que haya un único parámetro que varíe, por comodidad, no es necesario introducirlo como vector columna (lo cual sería introducirlo en la primera dimensión de una matriz), sino que acepta indistintamente un vector fila o columna.

Si en algún momento se decide cambiar o aumentar el número de parámetros de los que dependen las derivadas de estabilidad, sólo es necesario modificar los campos de "*LD.Stability.analysisCases*" y modificar el nombre y número de prelookups del modelo aerodinámico de Simulink para que refleje dicha variación.

3.1.2. Validación de los coeficientes

Dado que el proceso de introducción de los coeficientes es complejo y pueden haber existido errores al introducirlos que impidan al modelo de Simulink su correcta lectura, como que sea diferente el número de parámetros establecidos en "*LD.Stability.analysisCases*" que el número de dimensiones de las matrices de cada coeficiente, que se hayan indicado más coeficientes en el campo "*LD.Stability.Coeffs*" que los finalmente declarados o que alguno de los parámetros se encuentre vacío, despues de cargar los coeficientes de estabilidad se ejecuta desde "*/dataImport/loadData.m*" el script de validación "*/utilities/dataValidations.m*". Este script es el encargado de comprobar estos casos y, en caso de que se hayan producido, mostrar por pantalla una advertencia para solucionarlo, junto con la pertinente información para localizar fácilmente la causa de dicho error, así como de dejar constancia de ello activando las flags pertinentes dentro de la estructura de "*checks*".

Otra situación detectada como error y que este script debe solucionar, es el caso de que en un parámetro no se consideren variaciones, ya que el procedimiento de interpolación de los valores de los coeficientes a partir de prelookups necesita de al menos un mínimo de dos valores. Para solucionar esto, se añade al final de cada vector de parámetros un nuevo elemento que no se usará y en la dimensión pertinente de cada coeficiente se añade un elemento más. A su vez, se activa un flag de que dicho parámetro no experimenta variación, de forma que Simulink no use en ese caso la información proveniente de los prelookups, utilizando directamente la información proporcionada por el usuario.

3.2. Planta Propulsiva

El LIBIS obtiene la tracción necesaria para el vuelo de cinco motores eléctricos alimentados mediante dos baterías conectadas en paralelo. Cuatro de ellos se encuentran ubicados en las puntas de las alas, proporcionando tracción en el eje vertical para el vuelo como quadrotor y otro ubicado en la cola, empujando a la aeronave y proporcionando tracción para el vuelo como ala fija.

Para poder estudiar la variación de las actuaciones y actitud en función de cada uno de los motores, éstos se han definido de forma independiente, tal y como se observa en la figura 3.3, siguiendo el siguiente orden: el primer motor es el de la cola, el encargado del vuelo horizontral, el segundo motor es el correspondiente al semiala delantera derecha , y se sigue en sentido horario, siendo el tercero el de la

semiala trasera derecha, el cuarto el de la trasera izquierda y el quinto motor el de la delantera izquierda.

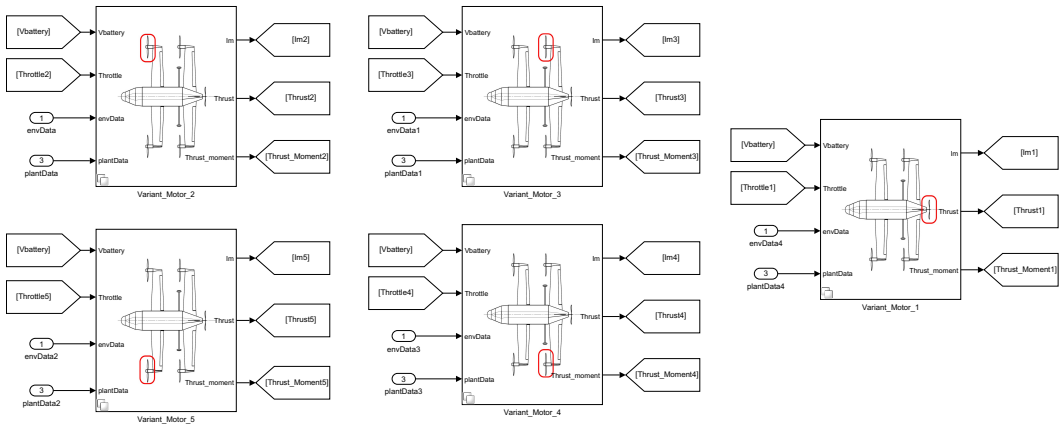

Figura 3.3: Modelos independientes para cada uno de los motores

Los datos de cada uno de los motores, hélices y variadores se definen en el script "*/dataImport/Propulsion/loadPropulsionData.m*", como se explicará en la sección 3.2.2, junto con las flags pertinentes para elegir el método de cálculo del sistema propulsivo deseado, véase, que en cada instante de ejecución Simulink resuelva el acoplamiento entre la hélice y el motor para calcular la tracción, lo cual es más lento pero permite probar diferentes motores y hélices de forma instantánea, o que al cargar el proyecto se ejecute un script que calcule el acoplamiento para un determinado rango de velocidades de vuelo, alturas y voltajes de salida del variador, lo cual permite agilizar la simulación pero requiere un significativo tiempo de cálculo cada vez que se modifican los motores o las hélices.

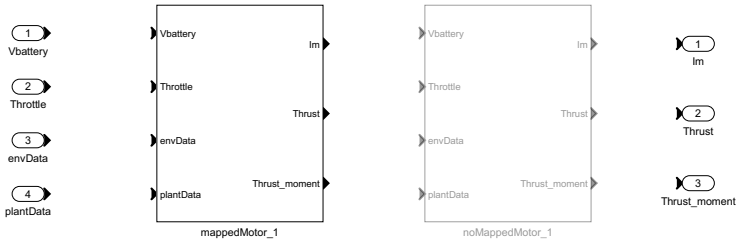

Figura 3.4: Definición de la planta propulsiva como "*Simulink Variants*"

Para posibilitar esta dualidad en el cálculo del sistema propulsivo, se ha definido cada motor como un "*Simulink Variant*", tal y como se muestra en la figura 3.4, y en el script anterior se declara la variable "*mappedMotors*" de forma que en función del valor que ésta tome, se encuentre activo uno u otro modelo. Los valores que puede

tomar esta variable en función del modelo de cálculo que se desee se muestran en la figura 3.5.

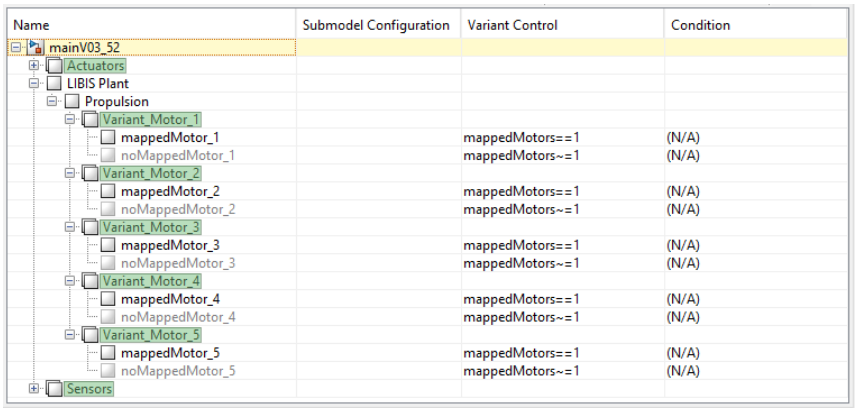

Figura 3.5: Valores de "*mappedMotors*" en función del modo de cálculo del sistema propulsivo deseado

3.2.1. Fundamento Teórico

El conjunto motor eléctrico-hélice se modela con un sistema de 3 ecuaciones donde intervienen las siguientes variables:

- V_m: voltaje de entrada al motor, regulado a través del ESC.
- I_m: Intensidad de entrada al motor, demandada en función de las condiciones de carga.
- n: RPM del conjunto motor-hélice. Motor y hélice giran a la misma velocidad debido al acoplamiento mecánico. No existe reductora.
- $P_{eje_{motor}}$: Potencia en el eje suministrada por el motor.
- $P_{eje_{hélice}}$: Potencia demandada por la hélice en el eje a un régimen de giro (n) y en unas condiciones de vuelo concretas (Velocidad de vuelo, densidad del aire).

Las ecuaciones correspondientes al motor aislado son las siguientes:

$$I_m = \frac{P_{eje_{motor}}}{n/K_v} + I_0 \tag{3.1}$$

$$V_m = \frac{n}{K_v} + I_m R_m \tag{3.2}$$

Donde $I_0\,[A]$, $R_m\,[\Omega]$ y $K_v\,[^V/\Omega]$ son constantes características del motor (intensidad en vacío, resistencia interna y constante de vueltas respectivamente). Se trata de un sistema de 4 variables, donde el valor de V_m es el parámetro de control y vendrá fijado por el sistema de control de vuelo a través de $\delta_{throttle}$. Queda por tanto un sistema de 2 ecuaciones con 3 incógnitas (I_m, $P_{eje_{motor}}$ y n).

Se necesita la ecuación de acoplamiento de potencias en el eje para cerrar el sistema. Puesto que la potencia que proporciona el motor debe ser la misma que absorbe la hélice para una condición de vuelo:

$$P_{eje_{motor}} = P_{eje_{hélice}} = f\left(n, V_{flight}, \rho\right)\Big|_{condición\,de\,vuelo} = f(n) \tag{3.3}$$

Es decir, para una condición de vuelo (V_{flight},ρ dados), la hélice demanda una potencia en el eje para mantenerse girando a ciertas n. La potencia demandada en el eje por la hélice es una función de n de la que no se dispone expresión analítica. Se utilizan por tanto datos tabulados del fabricante de la hélice.

Quedan por tanto determinadas todas las variables del sistema para un control y una condición de vuelo dados. Para obtener la tracción del conjunto, basta con dimensionalizar el coeficiente de tracción C_T de la hélice (datos del fabricante) para la condición y control especificados.

Se procede de modo similar para hallar el momento de reacción del grupo motor-hélice sobre la aeronave: interpolando el torque en las tablas del fabricante. Así mismo, los motores también producen otro momento sobre la aeronave, debido a que los vectores de tracción no pasan por el centro de gravedad.

3.2.2. Obtención de datos del motor y la hélice

Para importar los datos de los motores y de las hélices, y facilitar la posibilidad de cambiar fácilmente entre distintos modelos, se ha decidido implementar una librería de motores y otra de hélices, donde el usuario pueda ir añadiendo tantos elementos como estime oportuno, y desde la cual seleccionen cuales serán los que se usarán en cada caso.

El script que alberga la libería de motores y desde el cual se importa ésta es "*/dataImport/Propulsion/importMotorData.m*" y en él se define el vector de estructuras "*motorData*" donde cada elemento del vector representa un motor diferente y al que posteriormente se hará referencia a partir del nombre del modelo que se defina en los campos de ésta, siendo por tanto importante que dos motores nunca

tengan el mismo nombre de modelo. Aparte del modelo de motor, en la actualidad se usan los campos de intensidad en vacío (I_0), resistencia interna del motor (R_m) y el parámetro de vueltas (K_v), los cuales resulta imprescindible rellenar, y se han añadido otra serie de campos que se consideran de utilidad para futuras mejoras del programa.

La librería de hélices se obtiene a partir de los archivos proporcionados por el fabricante de hélices "APC Propellers" de datos de rendimientos de sus hélices en la siguiente página web https://www.apcprop.com/v/PERFILES_WEB/listDatafiles.asp.

Para ello, APC Propellers, a partir de la geometría de éstas y haciendo uso del software TAIR (Transonic Airfoil Analysis Computer Code) de la NASA, basado en Vortex Theory, obtienen una estimación de la sustentación y resistencia que éstas generan, proporcionando los parámetros mostrados en la figura , aunque ya avisan que la resistencia puede estar subestimada a bajas velocidades.

$$J = \frac{V}{nD} \qquad C_T = \frac{T}{\rho n^2 D^4} \qquad C_P = 2\pi C_Q \qquad \eta = \frac{C_T J}{C_P}$$

$$C_P = \frac{P}{\rho n^3 D^5} \qquad C_Q = \frac{C_P}{2\pi} \qquad \eta = \frac{TV}{P}$$

$$C_Q = \frac{Q}{\rho n^2 D^5} \qquad n = \text{revolutions per sec}$$

Figura 3.6: Parámetros existentes en los archivos de APC Propellers

El usuario sólo debe descargar los ficheros de datos de cuantas hélices desee, guardarlos en la carpeta "*/dataImport/Propulsion/Propellers/*" con el formato de nombre "*PER3_XXX.txt*" donde "*XXX*" será el nombre del modelo con el que se guardará la hélice en la librería y a partir del cual, posteriormente se hará referencia a ésta, y realizar una serie de revisiones para garantizar que el script "*/dataImport/Propulsion/importPropellerData.m*" los pueda procesar.

Dentro de las revisiones que deber realizar el usuario se encuentran el garantizar que existen 30 datos de velocidades para cada valor de RPM, valor que se ha encontrado que es estandar en este tipo de ficheros, y garantizar que no existe ninguna línea en la que por error, en vez de valores de rendimientos, los ficheros descargados muestran *NaN* (Not a Number), de producirse este fallo, que ha sido encontrado en varios ficheros, se recomienda realizar una interpolación manual para cada valor entre los correspondientes a una velocidad inmediatamente superior e inmediatamente inferior a la afectada.

Una vez se encuentan cargados todos los modelos de interés en las librerías, se puede proceder a cargar los modelos deseados en la estructura de datos "*LD*".

Para ello, es necesario modificar, y posteriormente ejecutar, el script "*/dataImport/Propulsion/loadPropulsionData.m*".

En éste, primero se debe indicar los valores deseados de los flags "mappedMotors" y "improveMapping", tal como se muestran en la figura 3.5 para el primero de ellos y como se explica en la sección 3.2.4 para el segundo. A continuación se carga la librería entera de motores, se definen los nombres que tomarán en la estructura "*LD*" cada uno de los 5 motores, las posiciones y los modelos de la librería que se asignan a cada uno de ellos, teniendo en cuenta que si se modifican los nombres, es necesario cambiar todas las referencias en el modelo de Simulink. Finalmente se repite este mismo procedimiento con la librería de hélices, identificando las deseadas para cada motor y asignándolas en la estructura "*LD*".

3.2.3. Motores sin mapear

Como ya se ha indicado anteriormente, existirán dos formas alternativas de calcular el sistema propulsivo, en este apartado explicaremos una de ellas.

La primera de forma, denominada coloquialmente "motores no mapeados", se trata de un modelo implementado en Simulink, mostrado en la figura 3.7, que resuelve en cada iteración el sistema de ecuaciones expuesto en el apartado de planteamiento teórico. El bloque 'motor_1' recibe el volatje del ESC (V_ESC, que sería equivalente a V_m) y un valor de la intensidad I_m (cuyo cálculo se explicará más adelante). Con estos valores se resuelven las ecuaciones 3.1 y 3.2, obteniéndose a la salida del bloque los valores de RPM (n) y Peje ($P_{eje_{motor}}$).

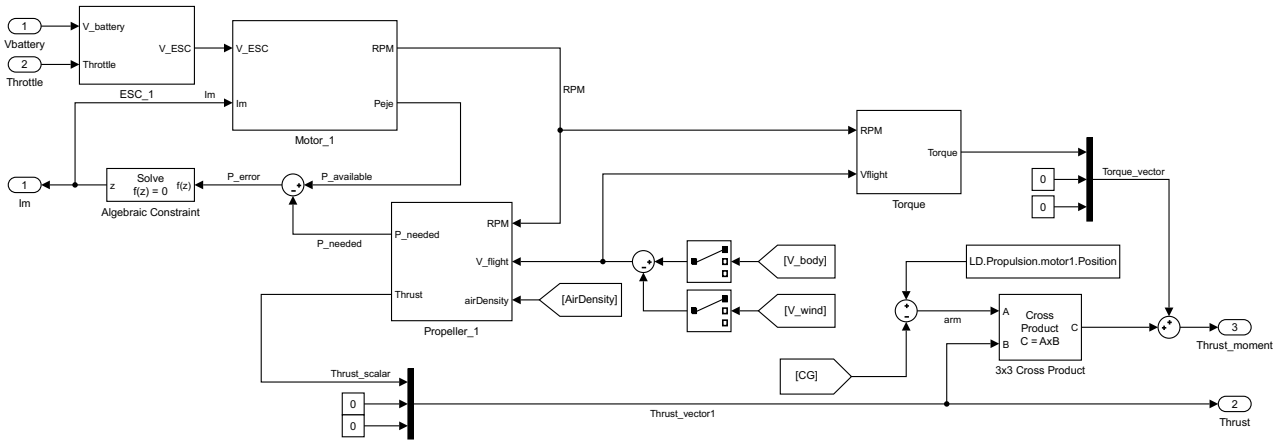

Figura 3.7: Modelo del sistema propulsivo sin necesidad de mapeado

El valor de RPM, junto con la condición de vuelo (V_flight y airDensity) entran en el bloque 'Propeller_1'. De esta forma se obtienen la potencia demandada en el

eje por la hélice y el empuje, interpolando en las tablas del fabricante.

Queda por obligar a que se cumpla el acoplamiento (ecuación 3.3). Para ello se utiliza el bloque de 'algebraic constraint', cuya entrada será $P_{eje_{motor}} - P_{eje_{hélice}}$. Este bloque irá variando un parámetro a su salida (I_m en este caso) hasta que se anule el valor a su entrada: $P_{eje_{motor}} - P_{eje_{hélice}} = 0$. De esta forma, el valor de I_m que entra al bloque por primera vez no es más que una estimación inicial. Simulink realizará este proceso iterativo hasta encontrar el valor de intensidad absorbida por el motor I_m que haga que se cumpla la ecuación 3.3.

Finalmente se construyen los vectores de tracción y momentos en ejes cuerpo para cada motor, en función de su orientación y posición respecto al centro de gravedad.

3.2.3.1. Modelo del motor eléctrico

En la figura 3.8 se muestra la implementación en Simulink de la resolución analítica del sistema de ecuaciones del motor.

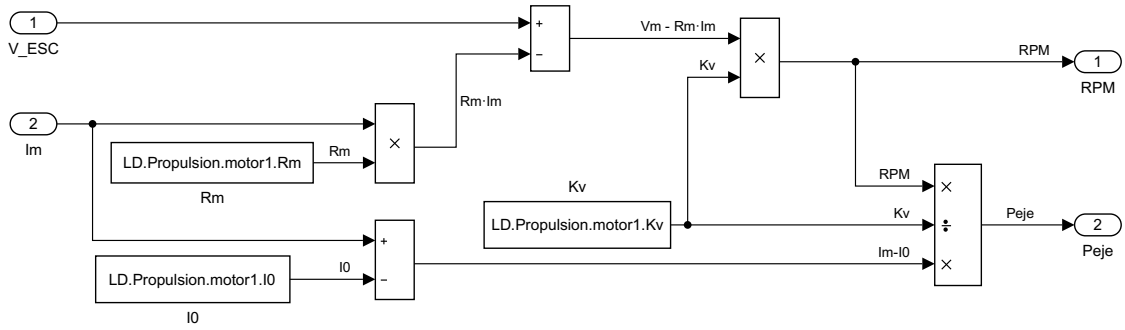

Figura 3.8: Modelo del motor eléctrico

En los casos en los que el voltaje de entrada al motor sea menor que $I_0 R_m$, el motor no sería capaz de ponerse en movimiento. Puesto que este caso en particular no se recoge implementando las ecuaciones del motor, ha sido necesario recurrir al modelo de la figura 3.9 para que el modelo no proporcione resultados erróneos.

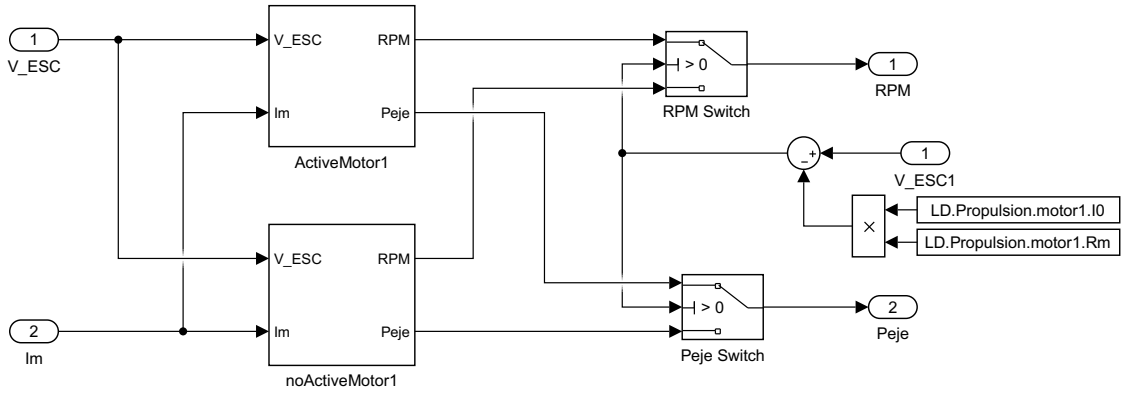

Figura 3.9: Esquema del modelo implementado para poder apagar el motor

3.2.3.2. Modelo de la hélice

Consiste en la interpolación en las tablas del fabricante y en una dimensionalización de los coeficientes de potencia y empuje a través de las siguientes ecuaciones:

$$T = \frac{1}{2}\rho n^2 D^4 C_T$$

$$P = \frac{1}{2}\rho n^3 D^5 C_P$$

Un esquema de la implementación realizada se muestra en la figura 3.10.

3.2.3.3. Modelo de torque

Funciona interpolando el torque en las tablas de la hélice. Debido a que el valor inferior de RPM proporcionado en las tablas es de 1000 y el tipo de intepolación del lookup es clip, para valores inferiores a 1000 rpm, el bloque devolvería un torque correspondiente a 1000 RPM. Esto se soluciona con el Switch de la figura 3.11 que produce que por debajo de 1000 RPM los motores no provoquen torque al girar muy despacio.

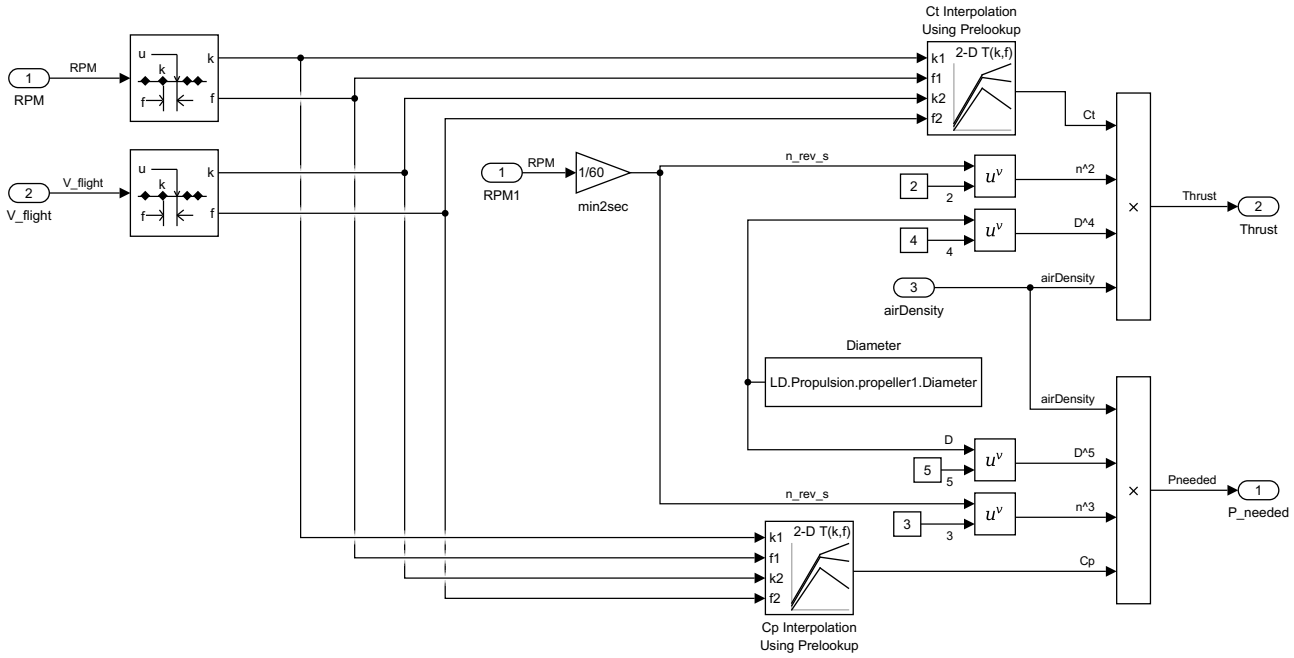

Figura 3.10: Modelo de la hélice a partir de prelookups

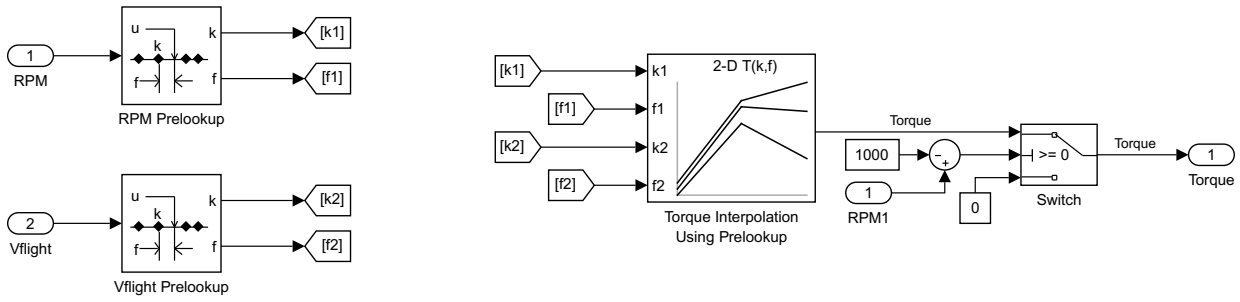

Figura 3.11: Implementación del modelo de torque

3.2.4. Motores mapeados

La segunda opción implementada para el cálculo de la planta propulsiva es la mostrada en la figura 3.12 y coloquialmente denominada "*motores mapeados*". En este caso el procedimiento de cálculo de la potencia transmitida por el motor al eje y el acoplamiento de ésta con la potencia demandada por la hélice, se realiza a través de un solver numérico en MATLAB para un barrido en los tres parámetros de entrada definidos anteriormente, véase, voltaje proporcionado por el variador, altura y velocidad de vuelo.

Por último, a partir de estas soluciones almacenadas, se obtienen los valores de tracción, potencia, intensidad y RPM usados en cada instante de simulación a par-

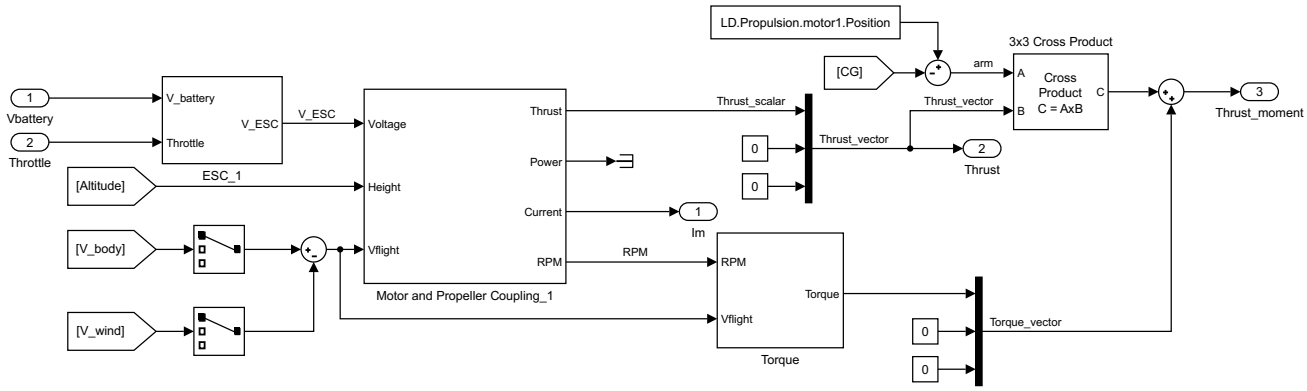

Figura 3.12: Modelo de la planta propulsiva mapeada

tir de una interpolación lineal entre los valores calculados más cercanos mediante prelookups, tal y como se muestra en la figura 3.13.

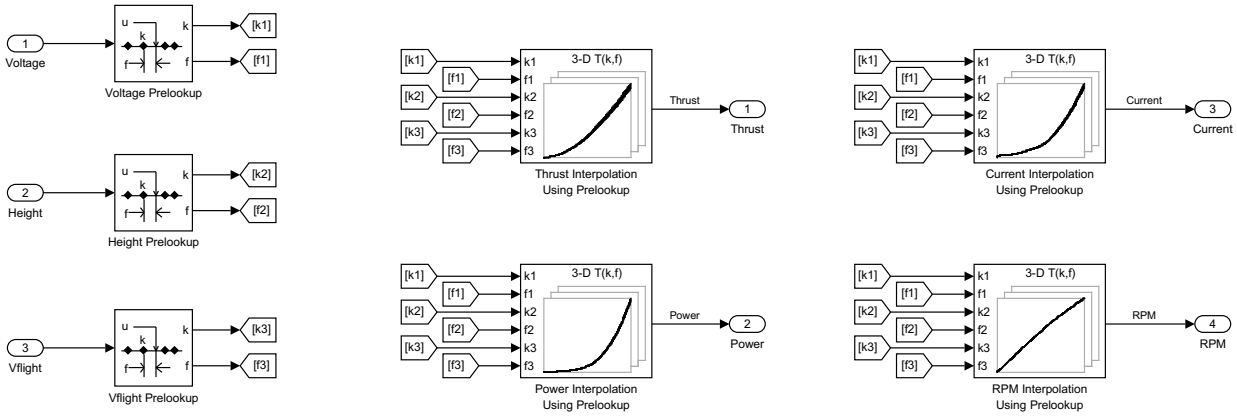

Figura 3.13: Acoplamientro del motor y la hélice

La función encargada de realizar el mapeado de las soluciones a partir de los parámetros del barrido definidos previamente es "*/dataImport/Propulsion/motorPropellerCoupling/propulsionDataMapping.m*". Ésta calcula los diferentes valores del mapeado a partir de sus límites y tamaño, llama al solver numérico con cada caso haciendo uso del cálculo multinucleo, reescala las soluciones y las almacena de forma estructurada, junto con los flags del grado de convergencia conseguido, mostrando los correspondientes mensajes de error de no alcanzarse una convergencia adecuada.

La función a resolver por este solver es la ubicada en "*/dataImport/Propulsion/-motorPropellerCoupling/motorPropellerCoupling.m*" que es la implementación matemática del fundamento teórico expuesto en el apartado 3.2.1, teniendo en cuenta la necesidad de apagar el motor cuando el voltaje proporcionado por el variador no sea suficiente para mover el motor como se indica en la sección 3.2.3.1.

La complejidad de este apartado radica en la necesidad de reducir al mínimo el número de veces que se realiza el mapeado, ya que, pese a que se ha implementado de forma que se usen todos los núcleos del ordenador para el cálculo, lo cual reduce significativamente el tiempo de computación frente a la programación mononúcleo habitual de MATLAB, cada vez que se ejecuta éste, a poco fino que sea el barrido en cada parámetro, requiere de varias horas para finalizar.

Para ello, es necesario almacenar los valores calculados, junto con los datos de los motores, las hélices y el barrido de parámetros usados para generarlos, y comprobar en cada ejecución del proyecto si se están pidiendo los mismos del caso anterior, o si alguno de ellos ha experimentado alguna variación. El script encargado de ello es "*/dataImport/Propulsion/motorPropellerCoupling/loadMappedData.m*", el cual recibe los valores del nombre de fichero correspondiente a ese motor y hélice, así como los valores del mapeado deseado, comprueba los si los existentes en el interior del fichero de datos son los mismos y de ser así los carga, o si no, devuelve campos vacios para que se calcule la solución con el nuevo mapeado.

Los valores de los límites del barrido y de el número de valores de éste se definen en "*/dataImport/Propulsion/loadPropulsionData.m*". A su vez, en el inicio de este script existe otro flag denominado "*improveMapping*", cuyo valor debería encontrarse siempre activo, y que determina, en el caso de encontrarse previamente activado el flag "*mappedMotors*", el comportamiento del programa en el caso de que al calcular el acoplamiento, o al cargar los datos de un fichero de soluciones, se encuentre algun flag de que la solución no ha convergido para algunos valores de los parámetros al realizar mapeado, o que lo haya hecho sin la suficientre precisión.

En el caso de que este flag se encuentre activo, se volverán a calcular estos casos únicamente, a partir de otro punto inicial generado de forma aleatoria, tratando de que se obtenga en este caso una convergencia correcta, y se generará un fichero de soluciones con los datos calculados previamente sustituyendo los valores corregidos de la solución. Si por el contrario, se encuentra desactivado, se ignoran los flags de convergencia y se utilizan directamente los valores calculados/cargados.

3.2.5. Batería

La batería se encuentra declarada dentro del propio bloque de Simulink por medio de constantes tal y como se muestra en la figura 3.14.

En el momento de declarar la batería no se encontró una ventaja significativa en crear expresamente un script de MATLAB para ello. Sin embargo, echando la vista atrás, pudiese haber sido interesante de cara a manterner uniformidad con el resto

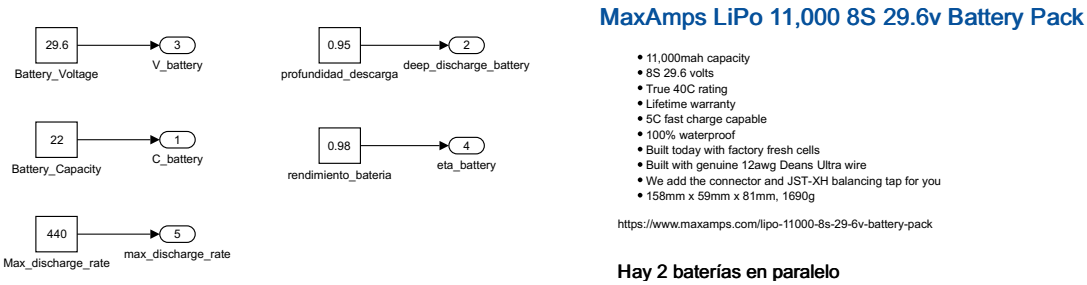

Figura 3.14: Datos de la batería

de las declaraciones de parámetros, que prácticamente la totalidad de ellas se realiza en scripts externos, quedando pendiente para una futura ampliación el realizarlo, así como crear una biblioteca de baterías entre las que se pueda elegir para estudiar las actuaciones de la aeronave en función de ésta.

También queda pendiente para una futura fase de mejora el limitar la máxima descarga de la batería a través del parámetro de máxima descarga, garantizando de esta forma con el simulador que nunca se alcanza dicha tasa de descarga y que no peligra la batería.

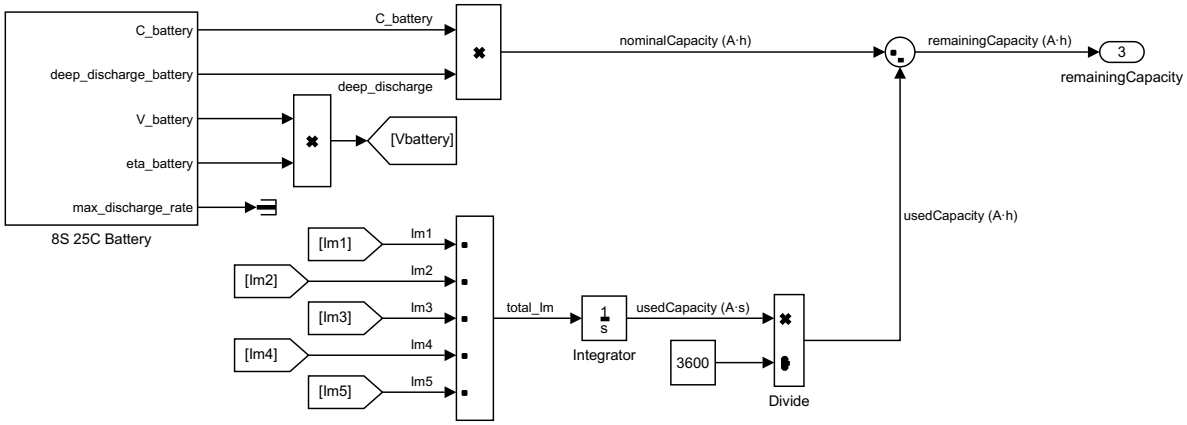

Figura 3.15: Esquema del cálculo de la autonomía restante

Por último a partir de la capacidad de la batería y la profundidad de descarga, se puede calcular la capacidad nominal de ésta, de la cual se puede obtener en cada instante la capacidad restante símplemente restandole la capacidad consumida hasta entonces, tal y como se muestra en la figura 3.15.

3.3. Modelo de Inercia

El bloque de inercia (figura 3.16) es el encargado de importar en el simulador todos los parámetro másicos e inerciales necesarios. Estos parámetros se cargan a la estructura LD cuando se inicia el proyecto a través del script "*/dataImport/Inertia/loadInertiaData.m*".

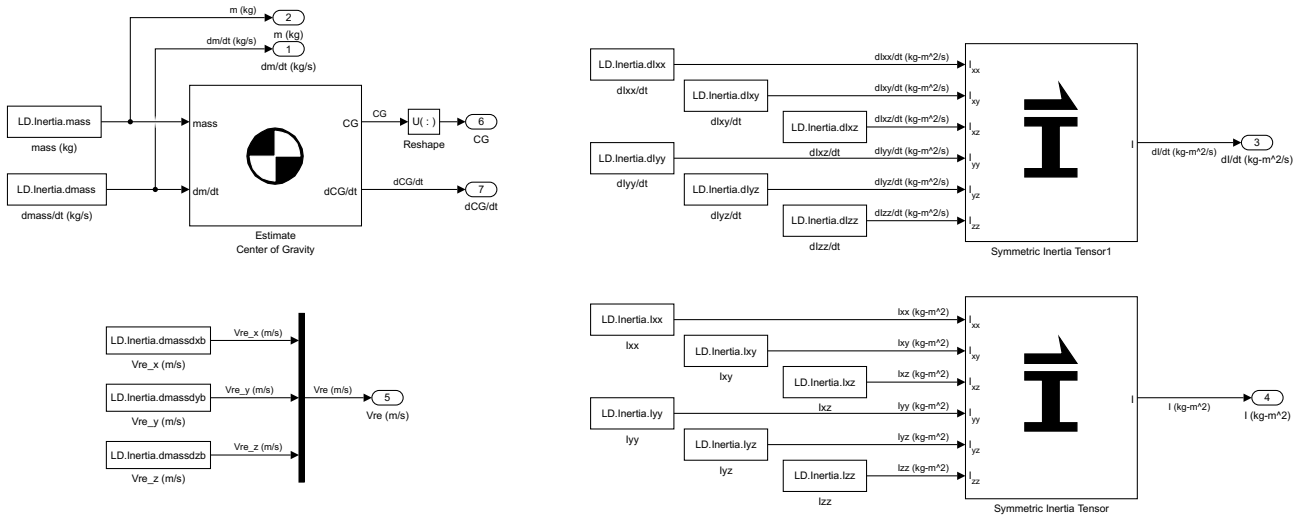

Figura 3.16: Modelo de inercia

A continuación se definen los parámetros básicos del modelo de inercia, así como una serie de parámetros avanzados a utilizar si se contempla variación de masa.

3.3.1. Parámetros básicos

- LD.Inertia.mass: Masa al despegue del LIBIS.
- LD.Inertia.CG: Vector de posición del centro de gravedad, en ejes cuerpo con origen en el morro de la aeronave.
- Tensor de inercia en ejes cuerpo, construido a partir de los valores de cada una de sus componentes.

3.3.2. Variacion de la masa

Si se contempla la variación de la masa durante el vuelo, hay una serie de parámetros adiccionales que deben ser especificados (su valor es nulo por defecto):

- LD.Inertia.dmass: variación de la masa respecto al tiempo.

- LD.Inertia.dmassdxb, LD.Inertia.dmassdyb, LD.Inertia.dmassdzb: Componentes del vector de velocidad a la cual la masa se expulsa (o se gana) en ejes cuerpo.
- Variaciones del tensor de inercia en ejes cuerpo respecto al tiempo.

3.3.3. Variación del centro de gravedad

En caso de que la masa varíe, puede hacerlo también la posición del centro de gravedad. Se define una ley lineal que permite obtener la posición del centro de gravedad en función de la masa en cada instante. Para ello es necesario definir los siguientes valores:

- Al despegue:
  - LD.Inertia.fullMass: Masa al despegue.
  - LD.Inertia.fullCG: Posición en ejes X cuerpo situados en el morro del centro de gravedad.
- En vacío (cuando se haya soltado toda la carga eyectable):
  - LD.Inertia.emptyMass: Masa al eyectar toda la carga.
  - LD.Inertia.emptyCG: Posición en ejes X cuerpo situados en el morro del centro de gravedad al eyectar la carga.

Capítulo 4

Comandos del Piloto

En este bloque se establecen todos los comandos necesarios para controlar la aeronave. A continuación se definen las salidas de este bloque, los diferentes modos de control existentes y su comportamiento al activar o desactivar el joystick. Las salidas de este bloque demandan una actitud del RPAS y entran al FCS, que en función de las variables demandadas establecerá las deflexiones de las superficies de mando y tracciones de los motores oportunas.

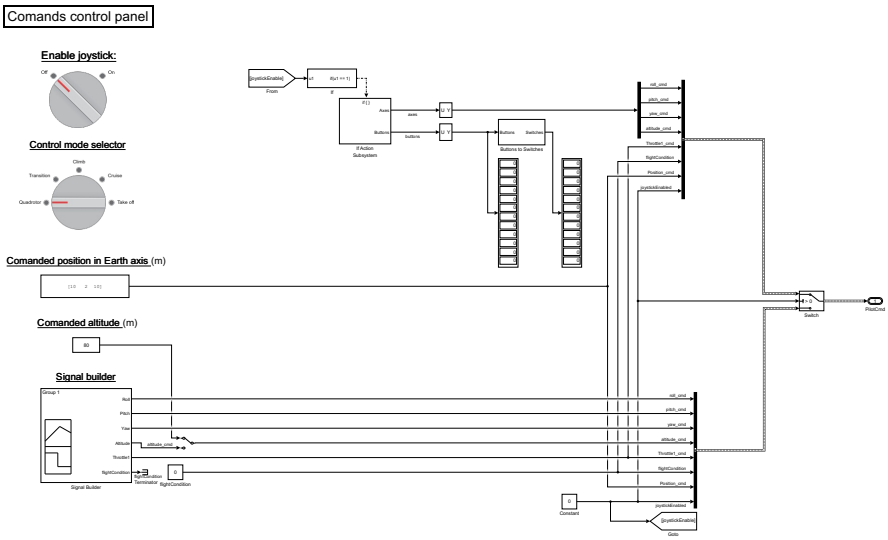

Figura 4.1: Diagrama de bloques de Comandos del piloto

4.1. Variables demandadas

A través de este bloque, el piloto puede establecer una serie de comandos para definir la posición y actitud de la aeronave. Son los siguientes:

- roll_cmd: controla el ángulo de balance de la aeronave.

- pitch_cmd: controla el ángulo de cabeceo de la aeronave.
- yaw_cmd: controla el ángulo de guiñada de la aeronave. En modo 'Quadrotor', controla la velocidad de guiñada de la aeronave.
- altitude_cmd: en el modo 'Fixed-wing - Cruise' permite mantener la altitud de la aeronave al valor comandado. En el modo 'Quadrotor' con el joystick activado, permite variar la altura respecto al valor de referencia establecido en la componente Z de Position_cmd.
- Throttle1_cmd: controla el $\delta_{throttle1}$ del motor trasero. No se utiliza en los modos de control implementados hasta el momento.
- flightCondition: indica el modo de control activo en cada momento.
- Position_cmd: posición en ejes XYZ tierra comandada en modo 'Quadrotor'.
- joystickEnabled: indica si el joystick está activo.

4.2. Control por joystick

Existen dos modos de control principales: 'joystickEnabled: off' y 'joystickEnabled: on', en función de si se desea controlar al RPAS a través de señales numéricas o utilizar un joystick. Esto interacciona de forma diferente con cada uno de los modos de control. Para activar o desactivar el joystick se utiliza el switch 'Enable joystick' (ver figura 4.1).

Con el joystick activo, los valores de roll_cmd, pitch_cmd, yaw_cmd se sobreescriben con la lectura de los 3 ejes correspodientes del joystick. La variable altitude_cmd se sobreescribe con la lectura del valor de throttle del joystick.

Con el joystick desactivado, los valores de estas señales se pueden establecer en función del tiempo a través del Signal Builder.

4.3. Modos de control

Existen 5 modos de control implementados actualmente. Se puede escoger cada uno de ellos a través del switch 'Control mode selector' que aparece en la figura 4.1.

4.3.1. Quadrotor

Permite comandar una posición en ejes tierra introduciendo su valor en el bloque 'Comanded position in Earth axis'. También permite controlar el ángulo de guiñada

a través de yaw_cmd, orientando la aeronave en la dirección deseada.

En caso de que el joystick se encuentre activo, los valores de las posiciones XY de Position_cmd serán ignorados y el piloto tendrá control directo sobre los ángulos de cabeceo, balanceo y velocidad de guiñada a través de los 3 ejes del joystick. El valor de la posición Z de Position_cmd seguirá activo y actuará como una altitud de referencia que la aeronave intentará mantener. Variaciones de ±10m sobre esta altitud de referencia se pueden obtener a través del throttle del joystick.

4.3.2. Transition

Este modo permite realizar una transición segura entre el vuelo como quadrotor y el vuelo como aeronave de ala fija. El piloto seleccionará este modo partiendo de una posición estabilizada en modo 'Quadrotor'. En ese momento se activará el motor trasero y la aeronave comenzará a acelerar hasta alcanzar una velocidad de 19,5$^{m}/_{s}$, valor de seguridad sobre la velocidad de entrada en pérdida que permite a la aeronave continuar ascendiendo. Para garantizar la estabilidad durante la transición, se desactiva cualquier comando del piloto durante esta fase y se mantienen los ángulos de actitud los más cercanos posibles a 0. Una vez alcanzada la velocidad de seguridad, se considera que la transición ha sido completada y el FCS activará el modo 'Climb' de manera automática.

4.3.3. Climb

En este modo, la aeronave intenta mantener una velocidad constante que le permita ir ganando altitud sin entrar en pérdida. Sería equivalente a un autopiloto tipo 'airspeed hold'. Para ello se utilizará el elevador horizontal, por lo que la variable pitch_cmd no tendrá ningún efecto. Se conserva el control lateral-direccional a través de roll_cmd y yaw_cmd.

4.3.4. Cruise

Con el joystick desactivado, la aeronave intentará mantener la altitud especificada en la variable altitude_cmd.. El piloto sigue manteniendo el control lateral-direccional a través de roll_cmd y yaw_cmd.

Si el joystick está activo, además de mantener el control lateral-direccional, el piloto puede comandar un ángulo de asiento a través de pitch_cmd pero se ignorará el valor de altitud_cmd.

4.3.5. Take-Off

Modo utilizado para el despegue de la aeronave como ala fija. Se mantiene el control lateral-direccional. Una vez alcanzada una altura mínima de seguridad, se recomienda activar el modo Climb para iniciar el ascenso.

4.3.6. Otros modos

Sería interesante implentar más modos de control que permitan la operación de la aeronave a lo largo de toda su misión (Aterrizaje, Transición ala fija- quadrotor, airspeed hold usando throttle, etc.), así como implementar un controlador que elija entre los diversos modos para cada condición de vuelo.

4.4. Ejemplos de maniobras

4.4.1. Despegue quadrotor - transición - crucero

Para realizar esta maniobra debe iniciarse la simulación desde el modo de control 'Quadrotor' y con el control por joystick desactivado. La aeronave volará hasta el punto especificado por defecto en 'Comanded position in Earth axis' $[10, 2, 10]$. Una vez estabilizada en este punto, debe seleccionarse el modo 'Transition'. Al alcanzar la velocidad de seguridad para el ascenso, la aeronave entrará automáticamente en modo 'Climb'.

Cuando la aeronave se encuentre a una altura superior a la del crucero elegido, se activará el modo 'Cruise' y se introducirá la altura de crucero en el campo 'Comanded altitude'.

La maniobra puede realizarse de manera similar con el joystick activado, teniendo un control más directo sobre el estado de la aeronave en todas las fases.

4.4.2. Despegue ala fija - ascenso - crucero

Para realizar esta maniobra debe iniciarse la simulación desde el modo de control 'Take-off'. Cuando la aeronave despegue del suelo y tome cierta altura de seguridad, debe cambiarse al modo 'Climb, lo que permitirá ascender hasta régimen de crucero. Una vez alcanzada la altura objetivo, se activará el modo 'Cruise' y se considera finalizada la maniobra.

4.5. Bus de comandos

Las variables demandadas se almacenan en un bus de datos no virtual denominado "CommandBus", el cual se encuentra definido en el script "*/dataImport/Buses/loadCommandBus.m*" y se ejecuta a través de "*/dataImport/loadData.m*" de forma automática al inicializar el proyecto.

Capítulo 5

Sistema de Control de Vuelo (FCS)

El sistema de control de vuelo establece las deflexiones de las superficies de mando y porcentaje de voltaje de los motores necesarios para cumplir con las demandas del piloto. El bloque consta de 5 subsistemas principales correspondientes a cada uno de los modos de control definidos en el capítulo 4. Dentro de las peculiaridades de cada uno de los modos de control, se puede hacer una primera clasificación global de los autopilotos implementados:

1. Quadrotor
    - *a*) Control longitudinal
    - *b*) Control lateral
    - *c*) Control direccional
    - *d*) Control en altitud
2. Ala fija
    - *a*) Control en altitud (Altitude hold)
    - *b*) Control en velocidad con elevador (Airspeed hold with elevator)
    - *c*) Control en ángulo de cabeceo
    - *d*) Control en ángulo de balance
    - *e*) Control en ángulo de guiñada

Los diferentes modos de control se construyen combinando los autopilotos anteriores y/o modificando ciertos parámetros en los mismos.

La solución adoptada para el diseño del FCS no pretende ser definitiva, sino permitir realizar una serie de maniobras básicas que contribuyan a validar el conjunto del simulador. Sin el FCS no sería posible realizar un vuelo a punto fijo como quadrotor o mantener un vuelo horizontal rectilíneo y uniforme.

El tuneado de las ganacias se ha realizado de manera iterativa, comprobando que las respuestas de los inner-loops fueran lo suficientemente rápidas y precisas antes de tunear los bucles externos.

A continuación se realiza una descripción breve de los controladores implementados (en cascada y con ganacias proporcionales en su gran mayoría) que pueden servir como base para futuros desarrollos.

5.1. Quadrotor

El vuelo como quadrotor del LIBIS se diferencia de un quadrotor habitual en que el momento generado por los motores traseros y delanteros no es el mismo aunque proporcionen la misma tracción, puesto que se encuentran a longitudes diferentes respecto al centro de gravedad. Esto, unido a que el exceso de tracción disponible respecto al vuelo a punto fijo es pequeño, provoca un fuerte acoplamiento entre los diferentes canales (longitudinal, lateral, direccional, altura).

La principal consecuencia del acoplamiento es que al demandar un posición que involucre control en varios canales, la respuesta difiera considerablemente respecto a la que se obtendría con el canal aislado, pudiendo no llegar a satisfacer la demanda del piloto.

Se ha conseguido una respuesta satisfactoria limitando las demandas de cada canal por separado y ajustando las ganancias en conjunto, a costa de una respuesta más lenta.

5.1.1. Control longitudinal

Controlador proporcional en cascada donde se realimentan las siguientes variables (de inner-loop a outer-loop): velocidad angular de cabeceo q, ángulo de asiento θ, velocidad longitudinal, posición X en ejes tierra. El control de realiza a través de un aumento de $\delta_{throttle}$ para los motores delanteros (T5, T2) y una disminución para los traseros (T3,T4).

En el caso de control por joystick sólo se realimentan: velocidad angular de cabeceo q y ángulo de asiento θ, donde el piloto demanda un ángulo de asiento.

5.1.2. Control lateral

Controlador proporcional en cascada donde se realimentan las siguientes variables (de inner-loop a outer-loop): velocidad angular de balance p, ángulo de balance

ϕ, velocidad lateral, posición Y en ejes tierra. El control de realiza a través de un aumento de $\delta_{throttle}$ para los motores del lado izquierdo (T5, T4) y una disminución para los del lado derecho (T3,T2).

En el caso de control por joystick sólo se realimentan: velocidad angular de balance p y ángulo de balance ϕ, donde el piloto demanda un ángulo de balance.

5.1.3. Control direccional

Controlador proporcional en cascada donde se realimentan las siguientes variables (de inner-loop a outer-loop): velocidad angular de guiñada r, ángulo de guiñada ψ. El control de realiza a través de un aumento de $\delta_{throttle}$ para los motores del lado izquierdo (T2, T4) y una disminución para los del lado derecho (T3,T5).

5.1.4. Control en altitud

Controlador PID donde se realimenta la altitud medida y se controla aumentando la tracción de los 4 motores verticales por igual.

5.2. Ala fija

5.2.1. Control en altitud (Altitude hold)

Consiste en un controlador proporcional en cascada donde se realimentan las siguientes variables (de inner-loop a outer-loop): velocidad angular de cabeceo q, ángulo de ataque α, y altitud. El control se realiza a través de la deflexión del elevador δ_e.

5.2.2. Control en velocidad con elevador (Airspeed hold with elevator)

Consiste en un controlador proporcional donde se realimenta únicamente el módulo de la velocidad en ejes cuerpo. El control se realiza a través del elevador δ_e. Se utiliza durante los modos de control 'Transition' y 'Climb'.

5.2.3. Control en cabeceo

Consiste en un controlador proporcional en cascada realimentando (de inner-loop a outer-loop): velocidad angular de cabeceo q y ángulo de asiento θ. El control se realiza a través de la deflexión del elevador δ_e. Se utiliza en el modo de control 'Cruise' cuando el joystick está activo.

5.2.4. Control en guiñada

Controlador proporcional en cascada donde se realimentan las siguientes variables (de inner-loop a outer-loop): velocidad angular de guiñada r, ángulo de guiñada ψ. El control de realiza a través de una deflexión del rudder δ_r.

5.2.5. Control en balance

Controlador proporcional en cascada donde se realimentan las siguientes variables (de inner-loop a outer-loop): velocidad angular de balance p y ángulo de balance ϕ. El control se realiza a través de la deflexión de alerones $\delta_{fr} = -\delta_{fl}$.

Capítulo 6

Sensores y Actuadores

6.1. Sensores

Para poder trabajar con variables obtenidas en la planta, es necesario que éstas pasen por los sensores, al igual que sucedería en la vida real. Para aumentar progresivamente el realismo del modelo se han implementado unos sensores ideales (Feedthrough) donde únicamente se renombran las variables y se devuelven como si fuesen las medidas por los sensores, y otro modelo donde se ha empezado a introducir ruido en algunos sensores, sin embargo este último modelo no está acabado y sería necesario hacer un estudio del ruido que introduce cada sensor así como realizar la implementación oportuna de éste.

Para definir estas dos opciones y que al inicio de cada simulación se pueda elegir la deseada, se ha hecho uso de los "*Simulink Variants*", cuyo esquema se puede ver en la figura 6.1, en este caso la variable que determina la selección deseada es "*noisySensors*", inicializada en "*/dataImport/Sensors and Actuators/loadSensorsActuators.m*" y como se muestra en la figura 6.2, un valor de 1 en esta variable selecciona los sensores reales y cualquier otro valor los sensores ideales.

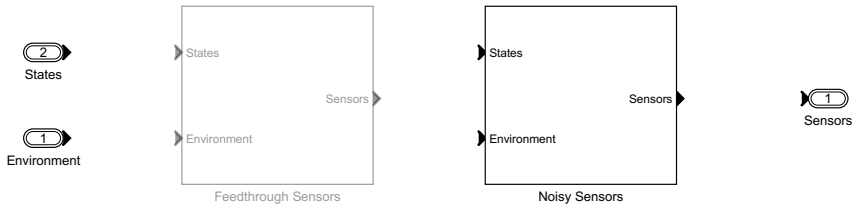

Figura 6.1: Modelos de Sensores implementados

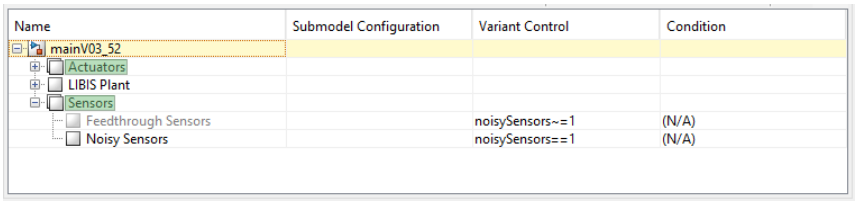

Figura 6.2: Valor de "*noisySensors*" para cada modelo de sensor

6.1.1. Sensores Ideales (Feedtrough)

Los sensores ideales no introducen ruido en las variables medidas, por lo que no son realistas. Sin embargo, evitan que se cometan errores conceptuales como podría ser trabajar en el FCS con variables que no se conocerían en la realidad.

En el momento de la creación de ésta documentación, no son conocidos los sensores que llevará el RPAS LIBIS, por lo que se han implementado los sensores que se han estimado oportunos (GPS, IMU, tubo de Pitot, veleta y una CPU), pudiendo ser que falte alguno o que alguno de ellos finalmente no sea necesario.

Como se muestra en la figura 6.3, los sensores necesitan información del bus de datos de la planta y del bus de datos del ambienente, y a su vez escriben toda la información en un bus propio que se denominará de sensores y se definirá más adelante.

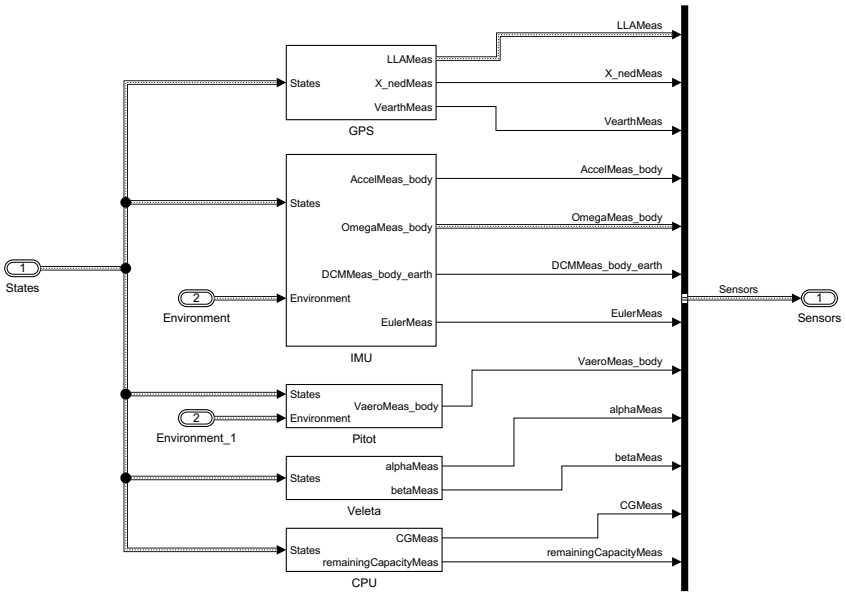

Figura 6.3: Sensores ideales implementados

6.1.2. Sensores Reales (Sensores con ruido)

Los sensores reales, como ya se ha comentados, no se encuentran definidos ni modelizados en su totalidad y es un trabajo que queda propuesto para un futuro trabajo. En esta etapa de diseño únicamente se ha implementado una IMU con ruido, sin ni siquiera haberse podido hacer un estudio de los valores adecuados de la frecuencia natural o amortiguamiento del acelerometro o el giróscopo, así como de los ruidos introducidos, sino que únicamente se han dejado los valores existentes por defecto en el modelo de la librería de Simulink.

6.1.3. Bus de Sensores

Para transportar de forma cómoda y organizada la información de los sensores entre bloques, ésta se introduce en un bus no virtual denominado "SensorsBus" que tiene que ser declarado e inicializado de forma previa a la ejecución de la simulación y que se encuentra declarado en "*/dataImport/Buses/loadSensorsBus.m*" y se ejecuta de forma automática al cargar el proyecto desde "*/dataImport/loadData.m*".

Si bien, no es necesario, si se recomienda para facilitar la comprensión del modelo y evitar errores involuntarios al conectar los bloques, definir en los imputs y outputs de los modelos que lo usen la el tipo de variable esperado como "*SensorsBus*".

6.2. Actuadores

A la hora de implementar los actuadores se han creado los modelos mostrados en la figura 6.4, véase, actuador ideal (Feedtrough), actuador lineal de primer orden, actuador lineal de segundo orden y actuador no lineal de segundo orden. Se considera que estos modelos son lo suficientemente conocidos como para no necesitar profundizar en ellos. Sin embargo, sí que merece la pena indicar que los parámetros de cada uno de ellos se encuentran referenciados al campo "*LD.Actuators*", por lo que para personalizar cada actuador sólo es necesario editar los campos correspondientes dentro del archivo "*/dataImport/Sensors and Actuators/loadSensorsActuators.m*", siendo especialmente necesario modificar los campos correspondientes al actuador no lineal de segundo orden, puesto que no se han buscado valores típicos de este tipo de actuador y símplemente se han inicializado los campos como 0.

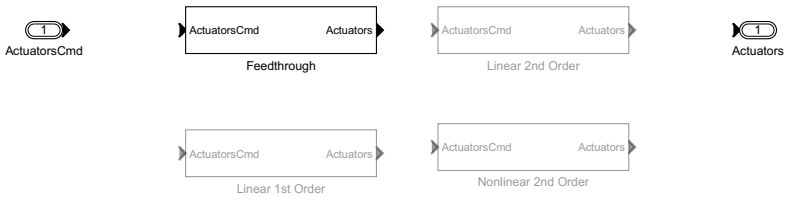

Figura 6.4: Modelos de actuadores implementados

Estos modelos, al igual que en el caso de los sensores, se han definido como "Simulink Variants" y en este caso la variable que realiza la selección es "*actuatorsDynamics*", la cual se inicializa en "*/dataImport/Sensors and Actuators/loadSensorsActuators.m*" y en la figura 6.5 se muestran los posibles valores que puede tomar en función del modelo que se quiera seleccionar.

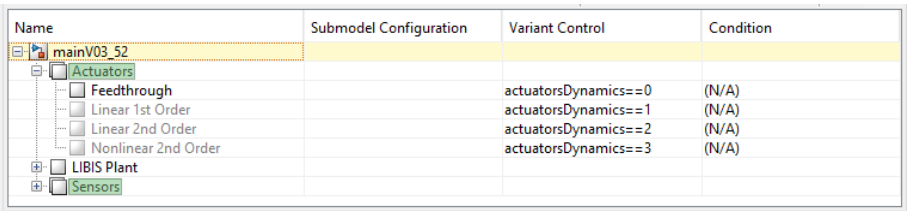

Figura 6.5: Valor de "*actuatorsDynamics*" para cada modelo de actuador

6.2.1. Bus de Actuadores

En el caso de los actuadores, tanto la entrada como la salida es un bus de actuadores, definido en "*/dataImport/Buses/loadActuatorsBus.m*" y que se ejecuta de forma automática al cargar el proyecto desde "*/dataImport/loadData.m*".

Capítulo 7

Visualización

Este bloque permite el seguimiento de todas las variables que definen el estado de la aeronave. Se compone de 3 elementos principales: Scopes, Visualización 3D y panel de instrumentos.

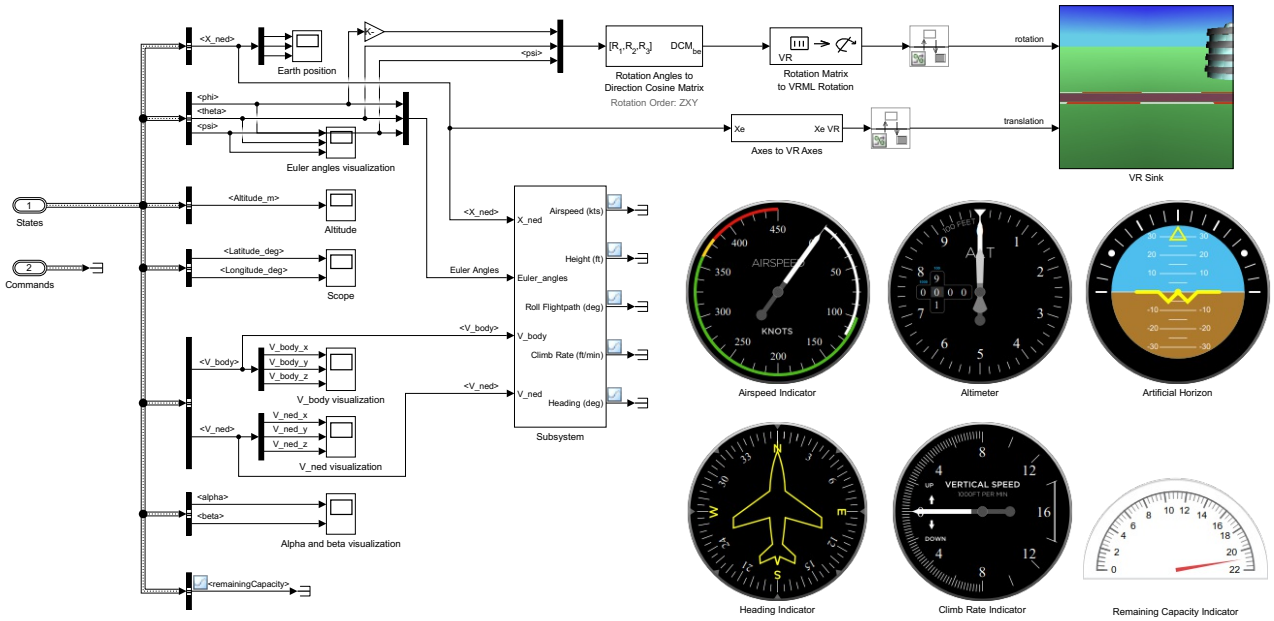

Figura 7.1: Visualización: Bloque Simulink

7.1. Scopes

Pueden abrirse de manera individual y muestran la evolución temporal del vector de posición en ejes tierra, ángulos de Euler, altitud, latitud, longitud, velocidades en ejes cuerpo y tierra, ángulos alpha y beta y capacidad restante en las baterías.

7.2. Visualización 3D

Se muestra por defecto al abrir el modelo. Para volver a abrirlo en caso de cierre, hay que hacer doble click sobre el bloque 'VR sink'.

Su utilidad principal es comprobar que el comportamiento de la aeronave coincide con lo esperado sin tener que consultar varios scopes de manera simultánea.

El modelo consta de una visualización de ejemplo de Matlab en el que se ha añadido el modelo 3D del LIBIS. Se puede cambiar entre las diferentes vistas utlizando las herramientas de la barra inferior ('Next Viewpoint').

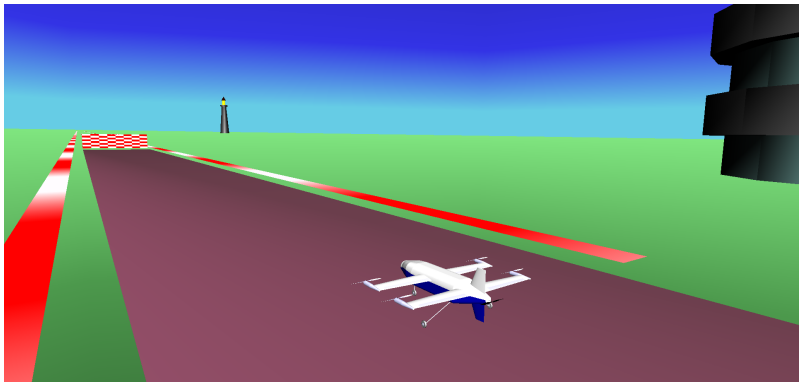

Figura 7.2: Entorno 3D

7.3. Panel de instrumentos

Consta de una serie de displays que emulan el panel de instrumentos de una cabina convencional.

Capítulo 8

Utilities

En la carpeta del proyecto *'/utilities'* se encuentran una serie de scripts que pueden resultar de utilidad en las futuras ampliaciones del programa.

8.1. getTransferFunctions.m

Calcula las funciones de transferencia de la planta del LIBIS en bucle abierto y las guarda en un archivo *LibisTransferFunctions.mat* . De aquí se puede extraer información sobre los modos propios (guardados en la estructura 'Modes') y la respuesta de la aeronave frente a entradas en los mandos.

8.2. SAS_Long.m

Partiendo de las funciones de transferencia calculadas a través del script anterior, calcula las funciones de transferencia en bucle cerrado para un Sistema de Aumento de Estabilidad del canal longitudinal en el que se realimentan el ángulo de ataque y la velocidad angular de cabeceo. Hace un barrido en ganancias de realimentación y muestra la posición de las raíces para las diferentes combinaciones ganancias.

8.3. getCruiseConditions.m

Este script sirve para calcular las variables de trimado necesarias para mantener un vuelo horizontal, rectilíneo y uniforme a la altura y velocidad seleccionadas en la sección input. Las variables calculadas se sobreescribirán sobre los valores iniciales de la simulación, de tal forma que ésta comenzará en la condición de vuelo seleccionada.