CAPÍTULO 3

SIMULACIÓN EN TIEMPO REAL

3.1 INTRODUCCIÓN

La simulación en tiempo real de un modelo es necesaria cuando se quieren verificar las interacciones del modelo respecto a un operador humano, o bien cuando se quiere que el modelo sea capaz de interaccionar con la instrumentación o con los actuadores de la aeronave, que normalmente son diseñados para un funcionamiento exclusivamente en tiempo real.

Justo por la necesidad de integrar la simulación con el hardware existente en la aeronave, se desea cada vez más el funcionamiento en tiempo real de la simulación cuando ésta es cada vez más usada en las fases más avanzadas de la integración de un proyecto aviónico.

Existen diversos métodos para realizar una simulación en tiempo real, los más conocidos y difundidos de los cuales están disponibles en sistemas operativos comerciales, como los sistemas operativos basados en Unix o aquellos basados en el *core* de Windows NT (Windows 2000, Windows XP) [7] [8].

La enorme cantidad de software comercial disponible en entorno Windows orienta a menudo la elección del entorno de simulación a aplicaciones capaces de correr en los sistemas operativos Microsoft, como Matlab/Simulink de MathWorks o MATRIXx de SystemBuild.

En fin, la difusión y la simplicidad de utilización restringen la elección del entorno a Matlab/Simulink, ya que esta aplicación está muy difundida en el entorno de la investigación y en la industria aeroespacial, gracias también al número elevado de software y hardware que han sido diseñados para ser compatibles con este entorno.

3.2 LA SIMULACIÓN EN ENTORNO MICROSOFT WINDOWS

Desde el momento en que Windows evolucionó a la versión Windows NT – que se difundió, en las primeras versiones (heredando el kernel de OS/2), hacia el 1993 – se introdujo en el sistema operativo un estrato de abstracción (HAL – Hardware Abstraction Layer) con el objetivo de impedir a las aplicaciones acceder directamente a la memoria de sistema, a la CPU, o a los componentes hardware como por ejemplo la tarjeta de video o la tarjeta de audio. Esto representa un paso adelante en la prevención de muchos conflictos y errores críticos debidos a problemas

hardware/software, pero de hecho impide a las aplicaciones acceder directamente a los recursos hardware como las interrupciones generadas por el procesador, fundamentales para poder sincronizar las aplicaciones y hacerlas correr en tiempo real [6].

Además, Windows es notoriamente un sistema operativo Multi-Tasking de tipo *preemptive*[2] y el tiempo del procesador debe ser compartido en el tiempo entre las diversas aplicaciones en modo impredecible, en el cual el único control que se puede efectuar sobre el tiempo asignado a una aplicación sería a través de un mecanismo de prioridades, que permite como mínimo asignar una cantidad de tiempo del procesador superior para las aplicaciones que tienen una prioridad mayor, sin definir, eso sí, una cantidad específica de tiempo. Esta organización en la gestión del tiempo del procesador no permite a un sistema operativo de tipo Windows (a menos que no se recurra a algunas soluciones muy complejas) asignar a una aplicación un tiempo mínimo de procesador, condición necesaria para ejecutar una simulación en tiempo real, la cual requiere un tiempo mínimo definido para ejecutar un paso de simulación [9].

Simples actividades como el desplazamiento de una ventana sobre la pantalla realizado a través del ratón, el acceso a un file en un dispositivo lento, son operaciones que, siendo realizadas por dispositivos que tienen un acceso prioritario (en Windows la interfaz de usuario tiene prioridad respecto a la ejecución de las mismas aplicaciones) pueden requerir una tal cantidad de tiempo de procesador que cada actividad de las aplicaciones del usuario puede sufrir un retraso considerable.

Algunas estrategias posibles para la ejecución en tiempo real de aplicaciones en entorno Windows y adoptadas en el Matlab/Simulink son enumeradas en el párrafo siguiente.

3.3 LA SIMULACIÓN EN TIEMPO REAL CON SIMULINK

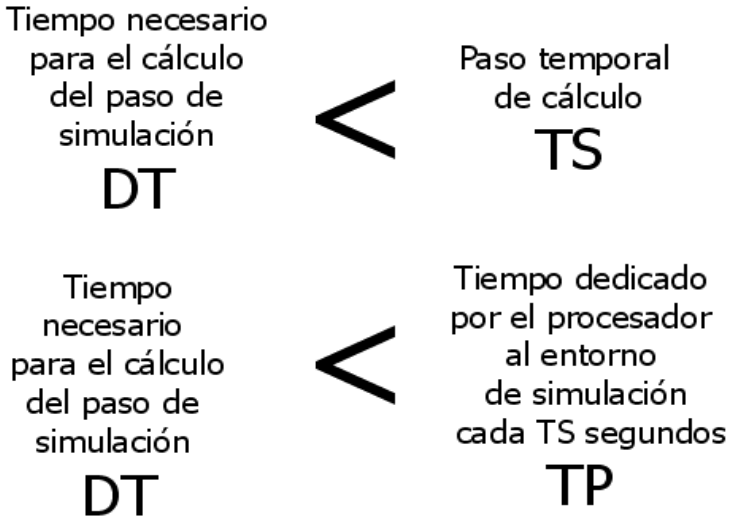
Simulink es el entorno de simulación de Matlab en el cual se pueden ejecutar los modelos matemáticos del helicóptero, de los sistemas de control o de los sensores. Para la ejecución de un modelo de simulación se ha de elegir un paso temporal de cálculo TS que puede ser más o menos grande en función de cómo de precisa deba ser la integración de los modelos de simulación y en función de la dinámica de los sistemas de control o a controlar[3].

---

2 En un S.O. Multitasking de tipo Preemptive, el S.O. asocia un tiempo finito del procesador a cada tarea, después del cual pasa a la ejecución de la tarea sucesiva. In un S.O. Multitasking de tipo Non-Preemptive, la tarea mantiene el control del procesador hasta que lo desee, después de lo cual el S.O. pasa el control del procesador a la tarea sucesiva.

3 Según la regla base dictada por el Teorema de Muestreo.

Para la ejecución de la simulación en tiempo real es necesario que el tiempo necesario para el cálculo del paso de simulación DT sea inferior al paso temporal de cálculo y que el tiempo dedicado por el procesador TP sea siempre mayor de DT.



La primera de las dos condiciones es un vínculo sobre el cual el diseñador tiene poca pero significativa libertad de operación, como utilizar un PC con prestaciones más elevadas o usar varios instrumentos Simulink para acelerar el tiempo de cálculo como el Simulink Accelerator o la compilación y ejecución externa realizada con el Real Time Workshop.

Para satisfacer la segunda condición el diseñador depende totalmente de Windows y las exigencias del sistema operativo de subdividir el tiempo de procesador entre todos los procesos que actualmente están en ejecución, más las acciones que vienen solicitadas de modo asíncrono por el operador a través de la interfaz gráfica. Hemos de tener presente que incluso la actualización de un gráfico de Simulink o la memorización de algunos resultados requieren mucho tiempo de cálculo, por lo cual la segunda condición está muy condicionada por una serie de eventos asíncronos y absolutamente imprevisibles.

En fin, un problema fundamental que tendremos que resolver para la ejecución de modelos Simulink en tiempo real es el hecho que Simulink, de base, no prevee ningún método para que el tiempo de simulación se quede “enganchado” al tiempo real, con el resultado que el modelo se ejecuta simplemente lo más rápidamente posible en función del tiempo que el procesador asigna a la aplicación Simulink, independientemente del paso temporal de cálculo TS. Por lo tanto, definiendo k = TS/DT, la simulación vendría ejecutada k veces más velozmente respecto al tiempo real.

3.3.1 Real-Time Workshop

RTW – Real Time Workshop es un componente de Simulink que permite la generación, a partir de un modelo matemático de simulación diseñado en Simulink con instrumentos gráficos, de un código C/C++ que puede ser compilado y ejecutado en un contexto externo a Simulink, llamado *target*, que puede ser el mismo sistema operativo Windows o un sistema operativo externo, como xPC, VxWorks Tornado u otros. En los siguientes párrafos describiremos los target xPC y el target RTWT, dos de los más utilizados (y económicos) en el ámbito de la investigación [10].

3.3.2 Real-Time Windows Target

RTWT – Real Time Windows Target es una de las posibles soluciones disponibles en Simulink para la ejecución de modelos de simulación en tiempo real. Este blockset ha sido creado para ejecutar simulaciones en tiempo real en el mismo sistema operativo Windows superando el vínculo del *multitasking preemptive*, operación que puede ser realizada solamente accediendo directamente a aquello que viene definido, en Windows, el ring 0[4], o bien el nivel más bajo del sistema operativo, conocido también como *Kernel mode*.

RTWT consiste de hecho en un Real-Time Kernel que se conecta directamente a los estratos inferiores del sistema operativo, accediendo a las interrupciones de sistema y creando los privilegios suficientes para obtener una prioridad superior a la de cualquier otro proceso que se esté ejecutando en Windows. RTWT garantiza además la ejecución del paso de cálculo con una tasa máxima igual a 20 Khz, puesto que el procesador sea capaz de calcular un paso de cálculo cada 0,05 msec (operación no imposible para modelos matemáticos no muy complejos). Además, RTWT garantiza que el modelo se ejecute con una temporización que consiga mantener alineados el tiempo real y el tiempo de simulación.

El procedimiento de creación de una simulación real-time con RTWT prevee la siguiente serie de operaciones:

- escritura del modelo de simulación
- compilación del modelo con Real-Time Workshop en modalidad externa
- ejecución del modelo compilado con el target RTWT, como un proceso separado e independiente de Simulink

4 Ring 0 es el nivel con la mayoría de privilegios y que interacciona más directamente con el hardware físico, como por ejemplo la CPU y la memoria.

- conexión entre Simulink y el target ya en ejecución para visualizar gráficos y resultados de la simulación real-time en curso

Las ventajas de utilizar RTWT son por lo tanto la no necesidad de utilizar un PC con un sistema operativo real-time dedicado ya que el sistema operativo sobre el que corre la simulación es Windows (NT, 2000, XP). Además, la simulación puede ser llevada a cabo con una tasa máxima de 20 Khz en código compilado que es notablemente más rápido que una simulación ejecutada en modalidad normal.

Una desventaja enorme de RTWT es debida al hecho de que el acceso con un Kernel separado del subestrato de Windows no permite el acceso a las librerías localizadas en los niveles superiores del Kernel de Windows, o bien a casi todas las librerías o drivers que permiten a los dispositivos conectarse a Windows, incluidos el acceso a files, el acceso a las interfaces seriales y a los canales de comunicación TCP/IP. Este límite reduce fuertemente el número de las aplicaciones que pueden ser realizadas con RTWT, restringiendo el campo de las posibilidades a las simulaciones no conectadas al exterior o a aquel hardware para el cual se han hecho drivers compatibles con RTWT.

Actualmente algunas decenas de tarjetas de adquisición de las marcas más conocidas (National Instruments, Keithley-Metrabyte, Analog Devices) son compatibles con RTWT.

Las escritura/lectura de files, la comunicación TCP/IP o serial, no son compatibles con RTWT propio por su naturaleza asíncrona [11].

3.3.3 xPC Target

En el ámbito de la investigación, xPC Target es uno de los componentes económicos más difundidos para la realización, a través de Real-Time Workshop, de aplicaciones ejecutables sobre el Kernel xPC, un sistema operativo monotasking que tiene como punto fuerte sus reducidas dimensiones y su escaso acceso a la CPU, lo cual permite una frecuencia máxima elevada con la cual se puede ejecutar una simulación.

El kernel xPC no se puede ejecutar en paralelo con otro sistema operativo, por lo cual es necesario cargarlo en un PC (basado en arquitectura x86) separado de aquel donde se desarrolla el modelo de simulación. El kernel es tan compacto que no necesita recursos particulares para ser ejecutado y puede ser fácilmente alojado en un miniPC del tipo PC104, típicamente utilizado por aplicaciones *embedded*.

Además, xPC posee la característica de utilizar exclusivamente un disco virtual, por lo que no necesita un disco fijo y, una vez creado el prototipo, puede ser alojado

en una memoria ROM sin la necesidad (como sucede para sistemas operativos basados en Windows) de una memoria flash sobre la cual descargar periódicamente la configuración del sistema operativo.

El conjunto del sistema operativo xPC y la aplicación realizada a través de Real Time Workshop son capaces de ejecutarse en tiempo real con una frecuencia de muestreo muy elevada (hasta 100 Khz), función de la capacidad del procesador utilizado en el PC que alberga el sistema operativo. La aplicación creada contiene además un mecanismo para mantener alineados el tiempo real con el tiempo de simulación.

El procedimiento de creación de una simulación real-time con xPC prevee la siguiente serie de operaciones:

- escritura del modelo de simulación

- compilación del modelo con Real-Time Workshop en modalidad externa, utilizando como sistema operativo target xPC

- envío, vía comunicación serie o TCP/IP, del modelo a simular al sistema operativo xPC

- conexión entre Simulink y el target ya en ejecución para visualizar gráficos y resultados de la simulación real-time en curso

Entre las ventajas del xPC Target está también la posibilidad de visualizar, sobre el monitor conectado al PC en el que corre el mini sistema operativo, algunos gráficos y valores de interés (se trata de una visualización muy simplificada), así como algunas capturas de depurado sobre la actividad del modelo de simulación.

La simple adaptabilidad de xPC a plataformas x86 de dimensiones mínimas ha hecho muy popular el blockset xPC Target en los proyectos aviónicos, sobretodo en lo que se refiere a los UAV y en general a todos los sistemas para los cuales el prototipado rápido que permite Simulink representa una ventaja notable. No es una casualidad que se hayan desarrollado expresamente para xPC una serie de librerías que permiten tener una interfaz con buses aviónicos (MIL-STD-1553 y ARINC 429), tarjetas de entrada/salida analógico y digital de muchos tipos, encoders, thermocouples[5], buses CAN y otros.

Es interesante la posibilidad de conectar xPC también con la interfaz RS232, además de poder hacerlo también con la interfaz UDP, que juntas proporcionan un

5 Sensores de temperatura basados en el acoplamiento entre dos metales

amplio abanico de aplicaciones realizables.

La flexibilidad del kernel de xPC lo hace también apto para ser utilizado en cualquier máquina de tipo x86, incluido cualquier ordenador o portátil que, a través de una selección en la fase de inicio, se puede arrancar con el sistema operativo estándar o con xPC.

Todas estas características de xPC lo hacen particularmente apto para el uso en fases avanzadas de realización, por ejemplo con el objetivo de simular en contextos Hardware-In-Loop objetos como sensores, sistemas de control o sistemas controlados en perfecto real-time, o para funcionar, si está alojado en una plataforma basada en x86, como componente de aviónica (en el caso en el que el componente no tenga peticiones particulares de adhesión a altos niveles de determinados estándares de diseño, come el estándar DO-178).

Disponemos además de una serie muy numerosa de funciones (API – Application Program Interface) para poder realizar funciones particulares con los dispositivos compatibles con xPC. Por ejemplo, se pueden realizar interfaces para interpretar protocolos seriales como NMEA o aquellos utilizados por sensores inerciales a la salida.

No existen particulares aspectos negativos de xPC, a parte del hecho de que, obviamente, para ser interconectable a una tarjeta de datos HW, tiene que estar ya disponible el driver relativo. Además, xPC no tiene la posibilidad de escribir datos en ficheros, lo cual, según el tipo de aplicaciones para las cuales ha sido pensado xPC, no debería constituir un problema: una grabadora de datos conectada a través de la interfaz RS232 controlable directamente desde xPC puede superar fácilmente este inconveniente [12].

3.3.4 RT Blockset

Este blockset actualmente está basado en un bloque de Simulink, realizado usando una S-function escrita en lenguaje C++ y utilizando las librerías multimediales de Windows. Respecto a otros productos, el blockset RT no usa un sistema operativo separado o necesita un kernel para generar una prestación real-time. El blockset está basado en el simple concepto que, para conseguir que Simulink ejecute el código con una temporización real-time, el tiempo de ciclo (el tiempo DT que necesita Simulink para calcular un paso de simulación), que es función del hardware y del sistema operativo en el que está ejecutándose Simulink, debe ser menor que el step TS de simulación deseado. Si esta hipótesis no es válida, no será posible la simulación en tiempo real, sea cual sea el método de scheduling aplicado.

Obviamente, esta afirmación no es completamente válida, por ejemplo, si

suponemos que trabajamos con un Sistema Operativo real-time, en el cual el scheduling puede ser completamente controlado. En cualquier caso, Windows no pertenece a esta categoría, ya que se trata de un ambiente multitasking con una aproximación tal que el scheduling de los procesos puede ser controlado sólo parcialmente.

La única característica controlable de Windows es la posibilidad de asignar una mayor prioridad al proceso en ejecución. En el bloque RT, de hecho, es posible definir la prioridad deseada para Simulink (hay que considerar que Simulink es un subproceso de Matlab, por lo que no basta controlar la prioridad del proceso (Matlab) sino que además habrá que controlar la del subproceso o *thread*).

El blockset RT simplemente mantiene la ejecución de la simulación de Simulink alineada al tiempo real, en el sentido de que, si el tiempo de ciclo DT es menor del tiempo de simulación TS, el bloque espera el tiempo necesario para "rellenar" o completar el step de simulación, dejando el tiempo de CPU sobrante a todos los procesos de Windows que lo necesiten. El concepto es muy simple pero eficaz.

Otra característica que hace preferible el uso de este bloque respecto a xPC o a RTWT, es la falta de limitaciones que el bloque impone al uso de las instrucciones de base o de Win32 en las S-function C/C++, algunas de las cuales son necesarias para conectarse a tarjetas particulares o incluso, simplemente, para la escritura en un fichero.

Por las características del blockset, el bloque RT se debe simplemente insertar en el modelo de simulación; es necesario fijar la prioridad del proceso o del thread y, finalmente, el paso de simulación. En simulación en modalidad normal (ni acelerada ni externa) el RT blockset puede controlar la ejecución de la simulación con una precisión de 1 msec sobre un paso de simulación igual a 5 msec (200 Hz). Idealmente, es posible obtener una tasa incluso mayor hasta un límite de 500 Hz, pero estando Simulink sujeto a todas las posibles interferencias sobre la interfaz gráfica (desplazamiento de una ventana, apertura de un menú a través del ratón), este límite no podrá alcanzarse seguramente en modalidad de ejecución normal de la simulación.

Para simulaciones más precisas desde el punto de vista de la temporización es necesario compilar el modelo con RTW y ejecutarlo en modalidad externa. En este modo, la prioridad del proceso externo creado con RTW puede ser tan alta como deseemos sin que por ello se comprometa la estabilidad del sistema operativo y de Matlab, y sin tener que sufrir retardos debidos a las operaciones sobre la interfaz gráfica de Windows.

En el caso de ejecución en modalidad normal, el procedimiento de creación de una simulación real-time con RT Blockset prevee la siguiente serie de operaciones:

- escritura del modelo de simulación

- inserción del bloque RT y configuración del bloque en modo que sincronicemos el tiempo de simulación con el tiempo real

- ejecución del modelo

En el caso de ejecución en modalidad externa, el procedimiento de creación de una simulación real-time con RT Blockset conllevará las siguiente serie de operaciones:

- escritura del modelo de simulación

- inserción del bloque RT y configuración en modo que sincronicemos el tiempo de simulación con el tiempo real

- compilación del modelo con Real-Time Workshop en modalidad externa, utilizando como sistema operativo DOS

- ejecución del modelo a través de la interfaz Matlab o a través de Explorer

- conexión de Simulink con el modelo que se encuentra ya en ejecución para visualizar gráficos y resultados de la simulación real-time en curso

En las siguientes figuras se muestran los tiempos de ejecución o de ocupación de la CPU para un caso de ejecución externa de la simulación en modalidad con prioridad normal y con prioridad realtime. En el primer caso son bastante visibles los efectos (deseados) de la "pesada" interacción del usuario con la interfaz gráfica de Windows, cuya prioridad es igual a aquella del proceso en el que corre la simulación. En el segundo caso, los efectos de la interacción del usuario con la interfaz gráfica son mínimos, ya que el proceso que calcula la simulación corre con prioridad mucho más elevada. Es inevitable que haya todavía presentes algunos retrasos imperceptibles incluso en este segundo caso, son debidos a la "pesada" interacción con la interfaz gráfica de Windows que ha sido inducida durante el test [13].

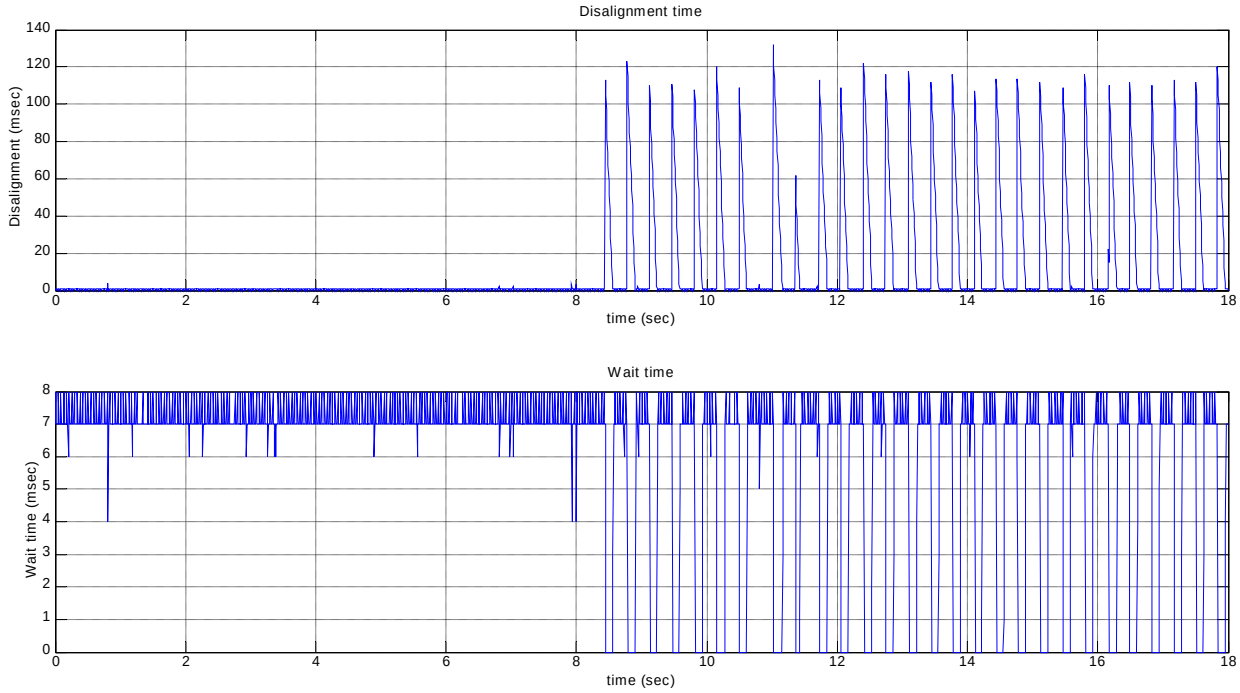


Figura 3.1: Prestaciones real-time con proceso de simulación de igual prioridad (Process Priority = NORMAL, Thread Priority = NORMAL) a la del proceso "ruido"

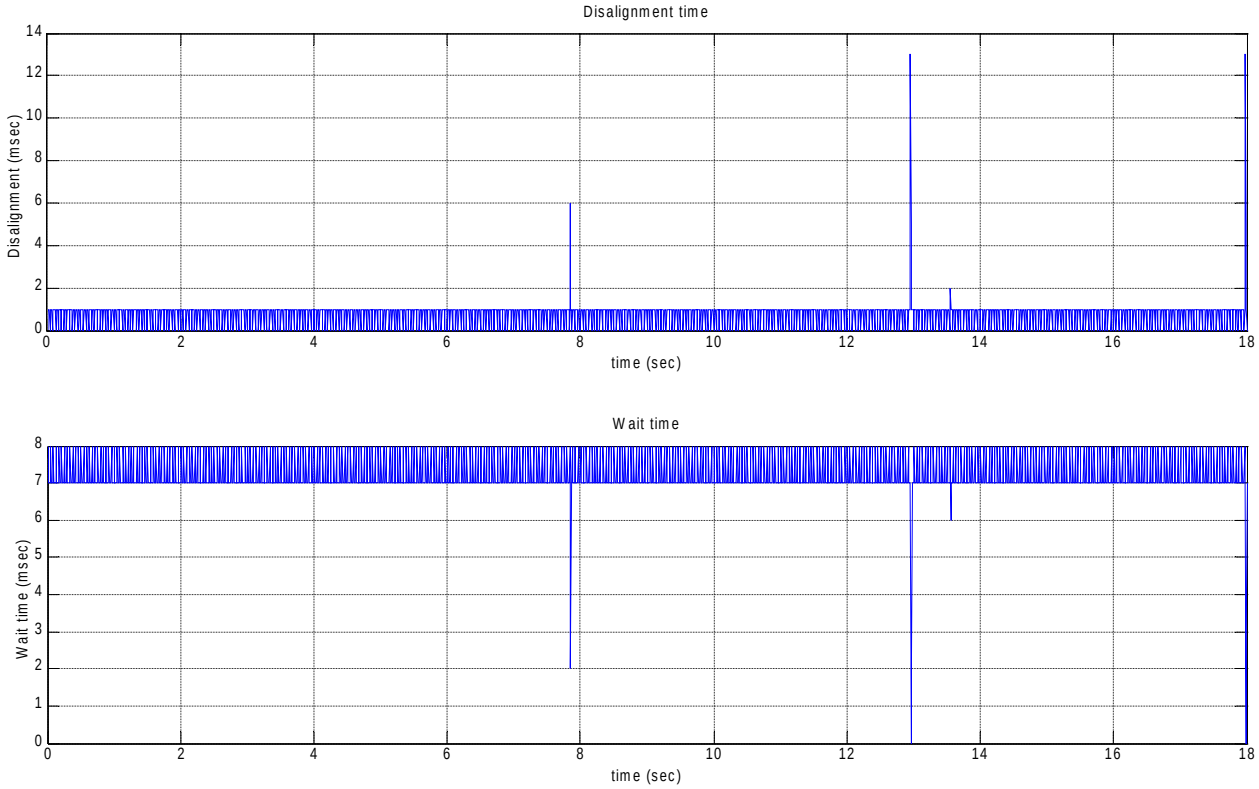


Figura 3.2: Prestaciones real-time con proceso de simulación de igual prioridad (Process Priority = HIGH, Thread Priority = TIME_CRITIAL) a la del proceso "ruido"

3.4 INSTRUMENTOS EMPLEADOS

El esquema completo utilizado para la simulación es el que se presenta en la figura 3.4:

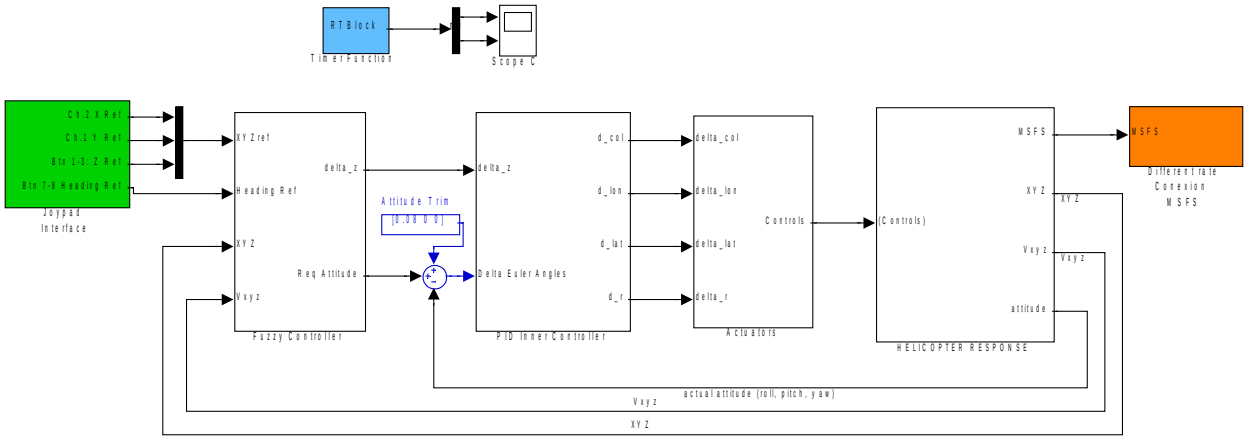


Figura 3.4: Esquema de sistema de control y de las interfaces externas de la simulación completa

El esquema utilizado comprende los siguientes componentes, que vendrán descritos en los párrafos siguientes:

- la interfaz hacia el joystick
- el controlador Fuzzy y el controlador PID
- los actuadores y el modelo del helicóptero
- la conexión con Flight Simulator
- el bloque RT

Habiendo sido construido el controlador Fuzzy para operar en coordenadas locales y no en coordenadas NED, mientras las coordenadas y la velocidad de entrada son NED, la interfaz de salida del joystick ha sido modificada de manera que pueda proporcionar a la entrada del controlador la posición y la dirección de referencia en la cual nos desplazaremos en lugar de la inclinación longitudinal y lateral o el comando del rotor como viene dado para los comandos clásicos de un helicóptero. Se ha realizado también una modificación análoga en el controlador Fuzzy, tal y como se ve en el párrafo relativo.

3.4.1 La interfaz Joystick

La interfaz con el joystick ha sido realizada utilizando como base la interfaz contenida en el blockset de Aerosim, producto de Unmanned Dynamics. La base está constituida por una S-function *sfunjoy*, que lee las 6 entradas analógicas y las 32 entradas digitales provenientes del joystick y las presenta a su salida. En la figura 3.5, se presenta el esquema de funcionamiento del bloque adaptado a un joypad, en el cual las entradas analógicas son utilizadas para el control de la posición XY (avance longitudinal y lateral), los botones 1 y 3 se utilizan para el control de la cuota y los botones 7 y 8 (rotación a izquierda y derecha) se utilizan para el control del heading.

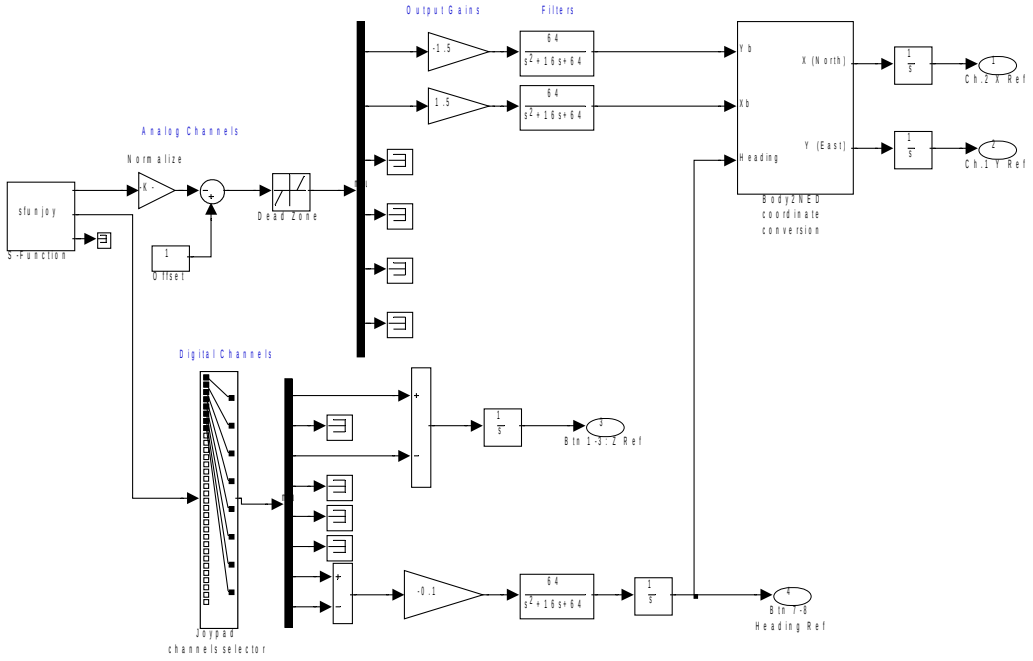


Figura 3.5: Esquema de conexión del joystick

El bloque trabaja a la misma frecuencia de la simulación, por lo que no es necesario efectuar conversiones de frecuencia.

Los canales analógicos son primeramente normalizados (la salida de un canal va de 0 a +65535) con un factor multiplicativo igual a 1/32768, así que se añade un offset para que la entrada vaya de -1 a +1. Una *deadzone* impide que posiciones del joystick ligeramente descentradas generen una entrada no deseada. Las entradas analógicas 1 y 2 son por lo tanto amplificadas (para reproducir una típica petición de desplazamiento), filtradas para eliminar los armónicos más elevados del input y por consiguiente convertidos en ejes NED para ser comparados con las otras entradas del bloque Fuzzy (posición y velocidad en NED). Siendo el desplazamiento hacia delante del joystick una petición de avance y, por lo tanto, de velocidad, esta señal debe ser integrada para poder traducirla en una petición de posición y constituir la referencia del controlador Fuzzy.

Análogamente se procede para los canales digitales, los cuales deben ser simplemente convertidos desde una entrada -1/0/+1 a una petición en términos de variación de cuota y variación de dirección, que integradas (y eventualmente filtradas) llevan después a la cuota y a la dirección deseadas [14].

3.4.2 El controlador Fuzzy y el controlador PID

El controlador Fuzzy y el controlador PID han sido ya ligeramente comentados en el capítulo precedente. Una única modificación que ha sido aportada, respecto al esquema ya presentado del controlador Fuzzy, afecta a la transformación de las coordinadas que tiene lugar, ya que el controlador opera en coordenadas locales (ejes XYZ con XY paralelos a la superficie terrestre, centro en el centro de masa de la aeronave, Z directo hacia abajo y X en dirección de la proyección de la proa de la aeronave sobre el plano XY, como se muestra en la figura 3.6).

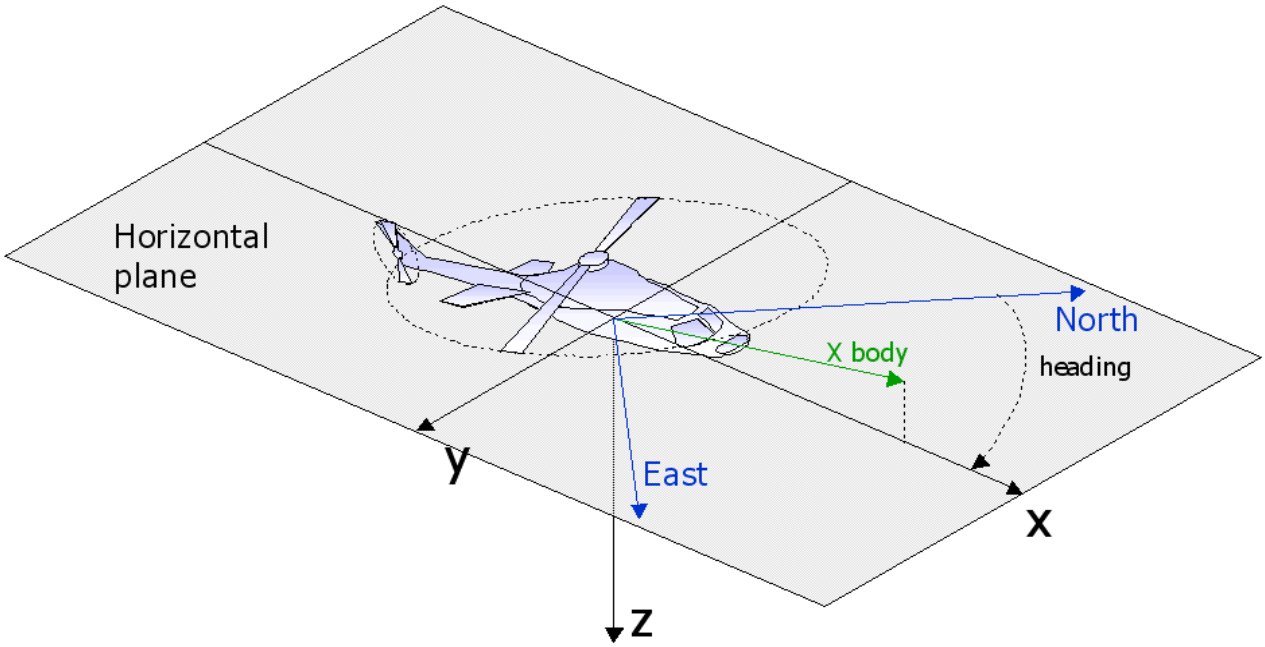


Figura 3.6: Sistema de referencia xyz en el cual opera el controlador Fuzzy

Siendo las coordinadas XYZ y la velocidad Vxyz de entrada al controlador expresadas en coordinadas NED, es necesario realizar la transformación utilizando la referencia de heading, como muestran las siguientes figuras.

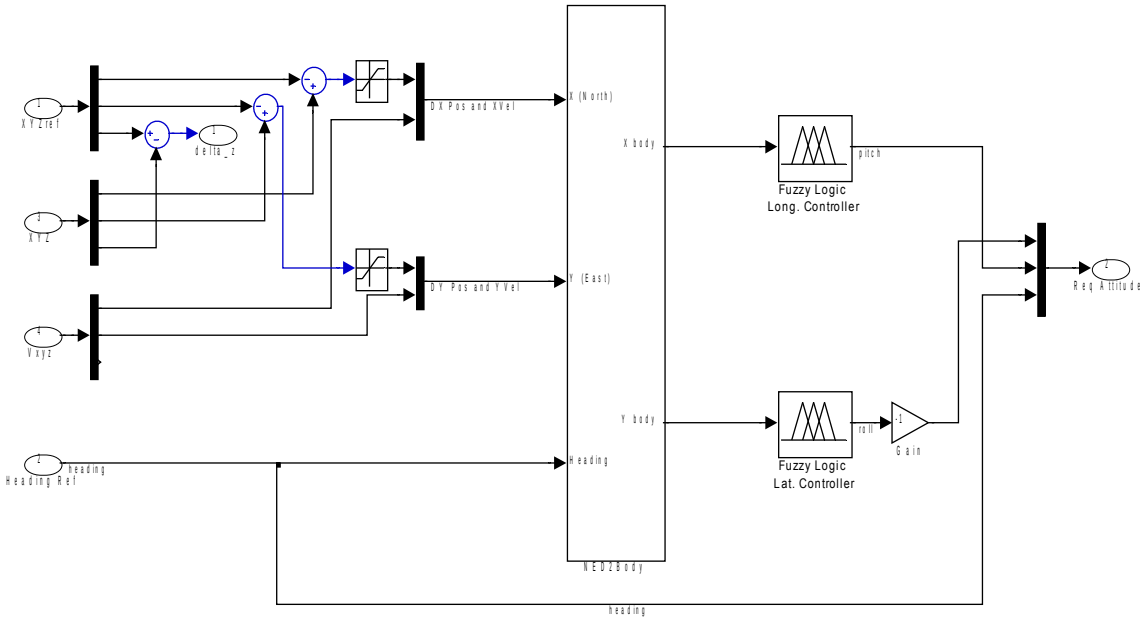


Figura 3.7: Esquema del controlador Fuzzy modificado para la conversión de las coordenadas NED en coordenadas XYZ locales

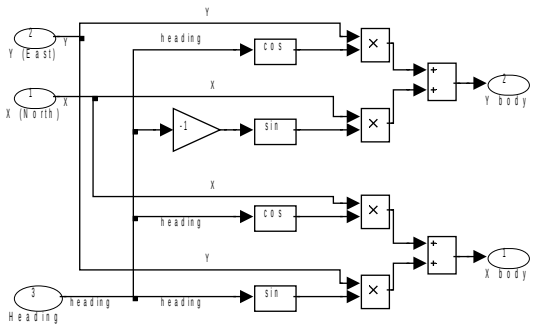


Figura 3.8: Esquema de la conversión de coordenadas [Nord, Est, Heading] a [X, Y] ***en un sistema de referencia solidad a la aeronave (sistema de ejes cuerpo)***

Los parámetros que definen el funcionamiento del controlador Fuzzy (las reglas) son definidos en una pareja de variables (*Xcontrol, Ycontrol*) que son cargadas en el momento de la inicialización de la simulación, accesible a través del comando *InitializeModel* [4].

3.4.3 El modelo de helicóptero: Proyecto Colibrí

Los bloques que reproducen la dinámica del sistema a controlar, o bien el modelo de helicóptero Colibrí, están organizados en dos subsistemas: el bloque de los actuadores y el bloque de la dinámica y de la cinemática de la aeronave.

Estos bloques son parte de una librería desarrollada por la Universidad EAFIT de Medellin (Colombia) en el contexto del proyecto Colibrí, para modelar el comportamiento de un helicóptero modelo de tipo X-Cell (ver figura 3.9).

Figura 3.9: Modelo de helicóptero X-Cell 0.60 usado en el proyecto Colibrí de la universidad EAFIT

El primer bloque, representado en la figura 3.10, contiene sólo la descripción de la dinámica y el valor mínimo y máximo de utilización (bloques de saturación) de los actuadores.

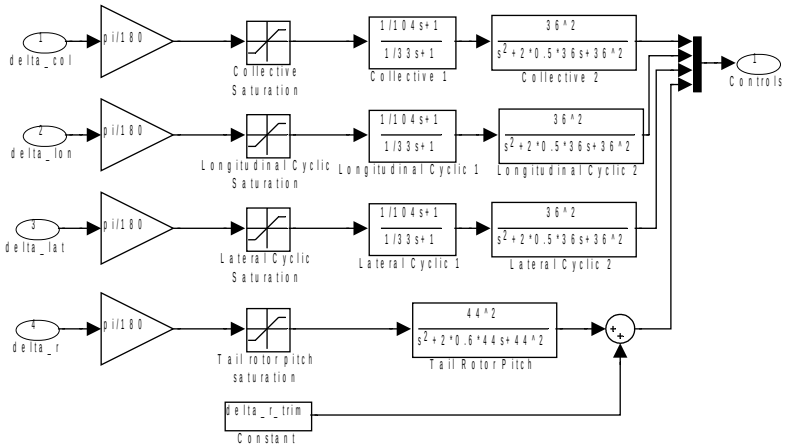


Figura 3.10: Bloque que representa la dinámica de los actuadores

El segundo bloque, representado en la figura 3.11, está compuesto por un primer estrato, en el cual se calcula el valor de referencia que habrá que dar para obtener automáticamente el valor de Omega deseado (o bien la velocidad de rotación del rotor principal), más un estrato sucesivo el en cual viene subdividida en bloques la aerodinámica (la dinámica y la cinemática del helicóptero).

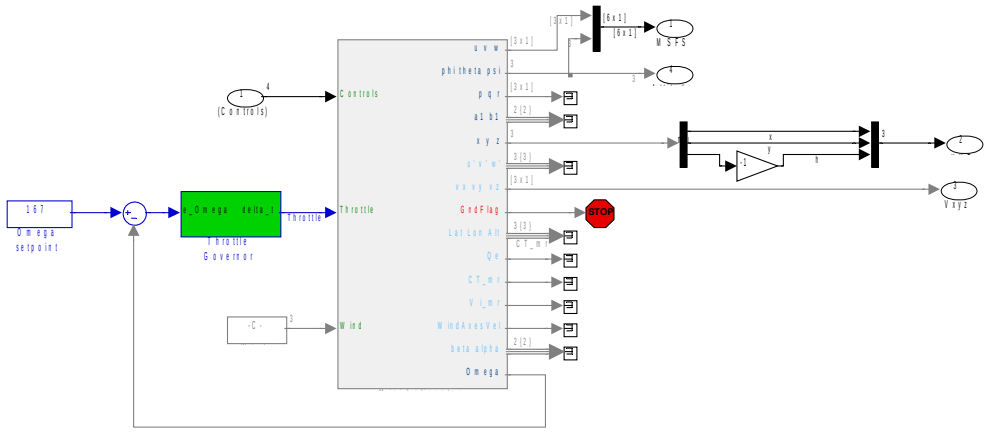


Figura 3.11: Estrato superior del modelo de la respuesta del helicóptero

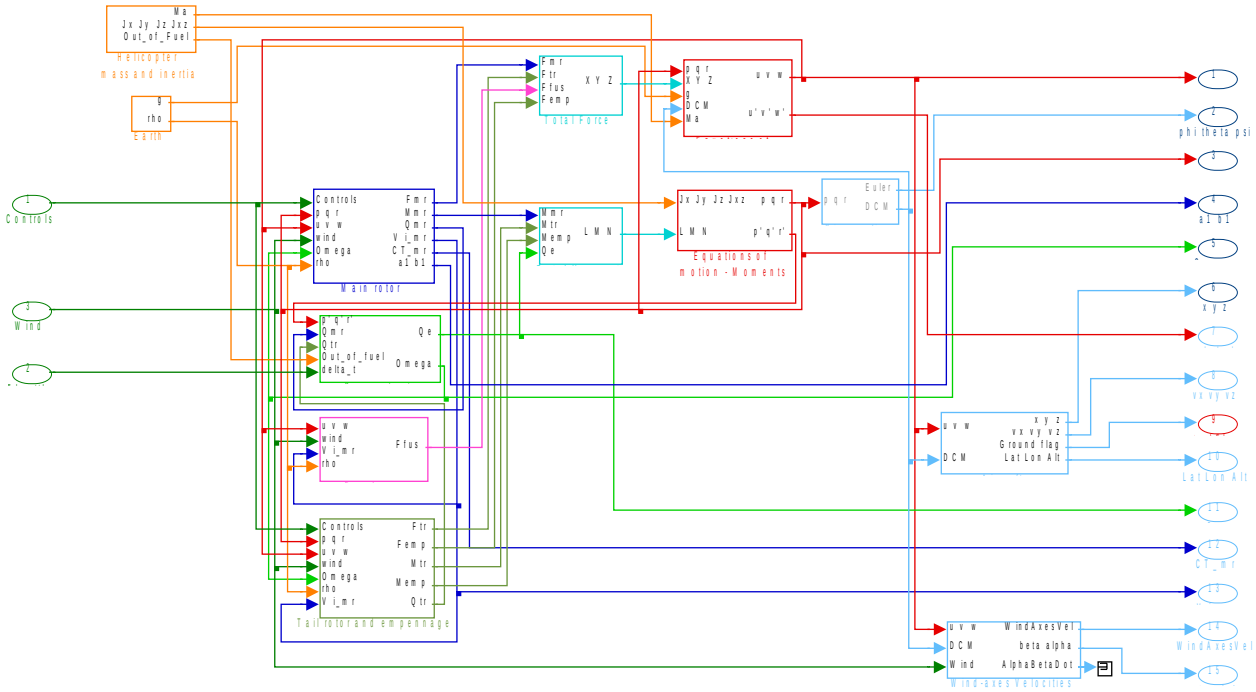


Figura 3.12: Estrato inferior del modelo de la respuesta del helicóptero

En la siguiente lista se enumeran las características del modelo X-Cell de helicóptero [4]:

- el modelo tiene 15 variables de estado

- asumimos que el centro de presión del fuselaje coincide con el centro de gravedad, y por lo tanto los momentos creados por las fuerzas aerodinámicas del fuselaje son despreciadas

- se usa una representación de *actitud* mediante quaternions para permitir *actitudes* extremas

- la cabeza de rotor es relativamente más rígida, permitiendo grandes momentos en el control del rotor

- las aspas del rotor principal no presentan torsión

- algunos efectos son "amplificados" (por ejemplo, la interacción entre el flujo de rotor y el fuselaje o la cola)

- hay una barra estabilizadora la cual es útil para controlar las dinámicas de *actitud*

- asumimos que el flujo es estable y uniforme (flujo de aire sin turbulencias)

- las fuerzas y momentos del rotor dominan ampliamente la respuesta dinámica, y esto simplifica significativamente la tarea de modelado

- las ecuaciones acopladas del rotor y de la barra estabilizadora se pueden agrupar en una ecuación de movimiento de primer orden del rotor (para el batimiento lateral y longitudinal en el plano de punta de pala (*tip-path*[6])

Las entradas y estados del modelo de simulación Matlab/Simulink son las siguientes:

$$u = [\delta_{col}, \delta_{lon}, \delta_{lat}, \delta_r, \delta_t]$$

donde:

δcol – entrada de control colectivo

δlon – entrada de control cíclico longitudinal

δlat – entrada de control cíclico lateral

δr – entrada de control del rotor de cola yaw

δt – entrada de control del throttle

mientras que el estado es un vector de 18 elementos:

6 El plano Tip-Path sería el plano descrito por la punta de la pala.

$$X = [u, v, w, p, q, r, \phi, \theta, \psi, x, y, z, a_1, b_1, \Omega]^T$$

donde:

- (u, v, w): velocidades lineares: vector 3x1 de las velocidades de la aeronave en un sistema de ejes cuerpo

- (p, q, r): vector 3x1 de las velocidades angulares

- (φ, θ, ψ): (balanceo, cabeceo, guiñada). Vector 3x1 de los ángulos de Euler

- (x, y, z): coordenadas cartesianas de la aeronave referidas a un sistema fijo con origen en el centro de la Tierra (eje Z directo como el eje terrestre y eje X que pasa por el cruce entre el ecuador y el meridiano cero)

- a_1, b_1: ángulos de batimiento longitudinal y lateral

- Ω: velocidad angular del rotor principal

El mini-helicóptero es un sistema con ocho grados de libertad: tres desplazamientos lineales (u, v, w), tres movimientos angulares (p, q, r) y dos ángulos de batimiento del rotor principal (a_1, b_1).

Los parámetros para la configuración del modelo de simulación son modificables a través de la interfaz gráfica, representada en la figura 3.13:

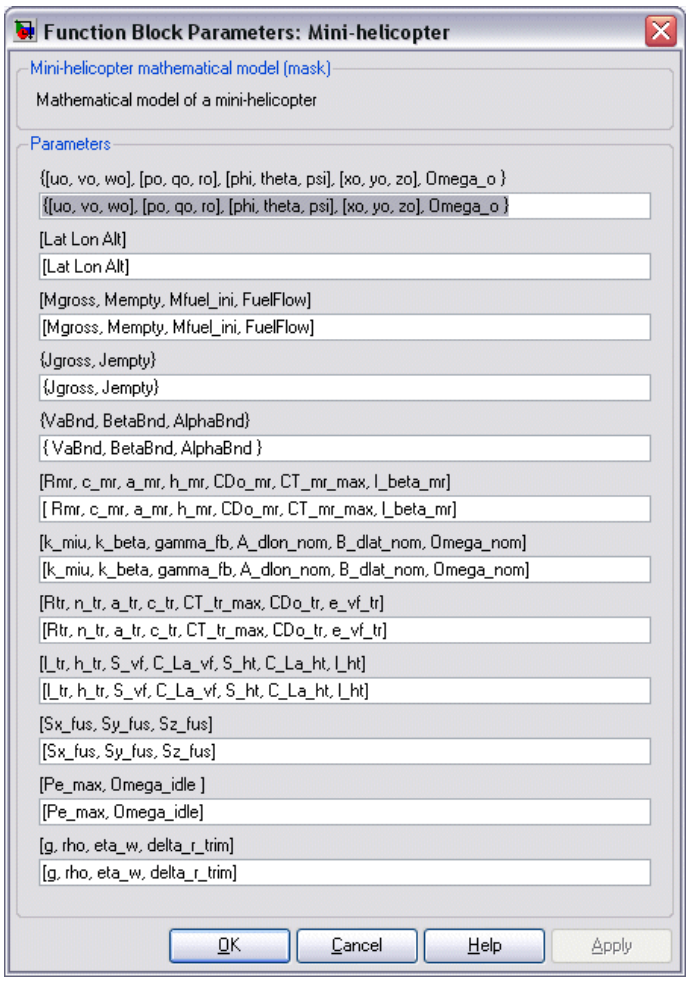


Figura 3.13: Interfaz para la modificación de los parámetros del modelo de helicóptero

Los ajustes del bloque están contenidos en la interfaz de inicialización de la simulación, accesible a través del comando *InitializeModel*.

3.4.4 Descripción de los bloques utilizados para la conexión con FS2002

La representación del estado actual de la aeronave tiene lugar en el simulador de vuelo Microsoft Flight Simulator. En uno de los párrafos sucesivos vendrá indicado cómo configurar el simulador para hacer que la conexión del simulador con la simulación en Simulink se realice correctamente.

La librería Aerosim contiene ya un bloque para la transmisión del estado de la aeronave al Flight Simulator, pero a causa de algunos mal funcionamientos y para permitir el funcionamiento del bloque en un contexto real-time, ha sido necesario utilizar una versión modificada.

El bloque de comunicación de Aerosim, de hecho, se encarga de *negociar* con el FS2002 el envío del nuevo estado a representar, esperando del Flight Simulator, el permiso para escribir sobre un área de memoria compartida entre el bloque Aerosim

(en realidad, la relativa S-function) y el FS2002.

La espera necesaria para la negociación puede ser incluso de diversas decenas de milisegundos (y de todas formas es una espera que bloquea la simulación), por lo cual el tiempo real en estas condiciones no es posible.

Con el fin de obtener una simulación en tiempo real, se han utilizado los bloques del FS2002 Blockset, que modifica los bloques Aerosim según la estructura dada por la figura 3.14:

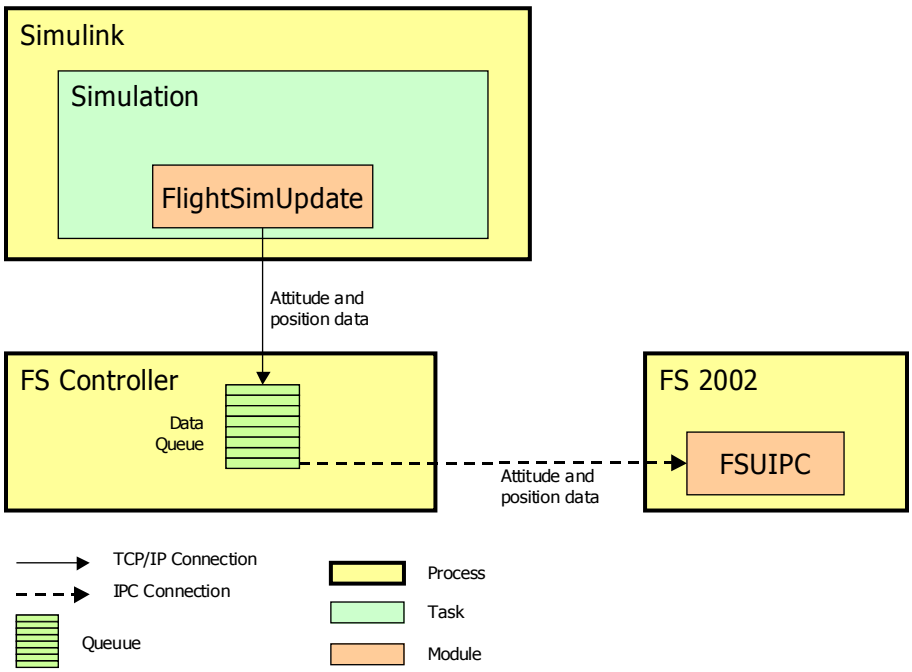


Figura 3.14: Esquema de conexión entre Simulink y FS2002

El bloque de Simulink llamado FlightSimUpdate recibe a su entrada la información a enviar a FS2002 (posición geográfica y *actitud*). Estos datos son enviados vía TCP/IP a un proceso externo de nombre FSController que deber ejecutarse en la misma máquina en que se esté ejecutando el Flight Simulator. Esta última comunicación tiene lugar a través de IPC (Inter-Process Communication: protocolo disponible gracias al plug-in del Flight Simulator llamado FSUIPC) y por lo tanto debe ser ejecutada necesariamente al interno del mismo PC.

La primera comunicación, siendo de tipo TCP/IP, no necesita ser ejecutada al interno del mismo PC, por lo que el FSController y el FS2002 pueden residir en una máquina dedicada. Además, la comunicación tiene lugar sin espera por parte de Simulink, ya que es el FSController el que se ocupa de enviar la información y de esperar que el canal IPC del Flight Simulator esté libre.

Los parámetros del bloque son configurables a través de la interfaz gráfica representada en la figura 3.15, en la cual deben ser definidos dirección y puerto TCP/IP de la aplicación FSController, la posición inercial de la aeronave (integrada autónomamente a partir de la velocidad de entrada, el paso de simulación y la tasa de envío de datos al Flight Simulator).

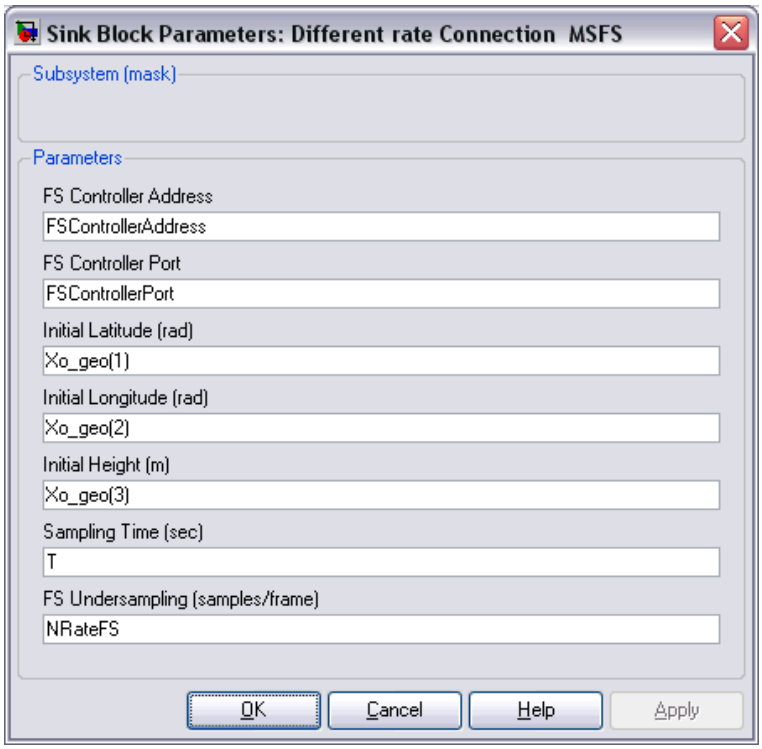


Figura 3.15: Interfaz para la configuración de los parámetros de la visualización en FS2002

El parámetro FS Undersampling indica cada cuántos pasos de simulación debe ser actualizada la posición de la aeronave en Flight Simulator. Siendo el ojo humano de hecho capaz de percibir un máximo de 20 *frames* por segundo, es inútil mandar 125, como sucedería para el tiempo estándar elegido para la simulación. La elección de configurar NrateFS = 5 puede ser considerada un óptimo compromiso para obtener una visualización fluida de la imagen en FS2002.

La configuración del bloque está contenida en la interfaz de inicialización de la simulación, accesible a través del comando *InitializeModel*.

Para arrancar la aplicación FSController, en cambio, es suficiente poner en ejecución la aplicación *FSController.exe* proporcionada en el blockset FS2002 [15].

3.4.5 Real-Time Blockset

El bloque RT ha sido ya descrito en los capítulos precedentes. Para esta simulación, basta definir los parámetros en modo apropiado, o bien el *step* de

simulación y la prioridad asignada al proceso y al *thread*, que en modalidad Normal son respectivamente *Matlab* y *Simulink*; en el caso de ejecución externa, se trata de la aplicación compilada [10].

La interfaz para la definición de los parámetros viene representada en la figura 3.16:

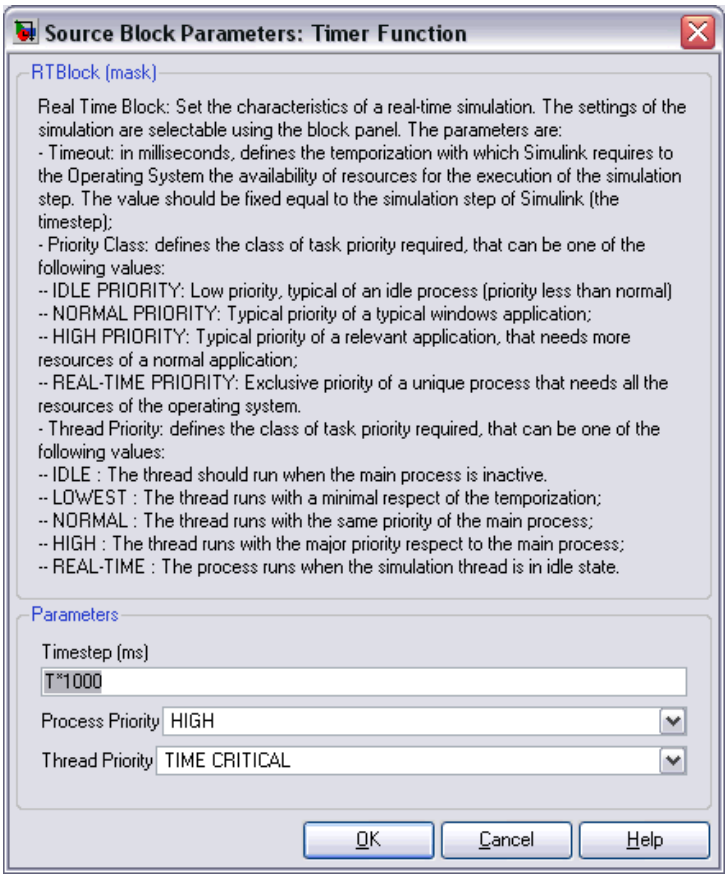


Figura 3.16: Interfaz para la definición de los parámetros del bloque RT

3.5 INICIALIZACIÓN DE LA SIMULACIÓN

Antes de cargar el modelo de simulación, es necesario definir todos los parámetros relativos al helicóptero del proyecto Colibrí, a la temporización, a la interfaz hacia Flight Simulator y al controlador Fuzzy. Estos pueden ser cargados a través de la interfaz gráfica que viene abierta a través del comando *InitializeModel*. La interfaz muestra el valor asociado a todas las variables necesarias para la simulación, cuyo valor viene definido inicialmente en el contenido del fichero *inicio.m*.

En la figura 3.17 se muestra el contenido de la interfaz para la modificación de los parámetros del modelo del helicóptero. Pulsando el botón “Accept”, se aceptan las modificaciones aportadas; pulsando el botón “Close”, no se salvan las

modificaciones; pulsando el botón “Default” las modificaciones vienen anuladas y se cargan los valores por defecto contenidos en el fichero *inicio.m*.

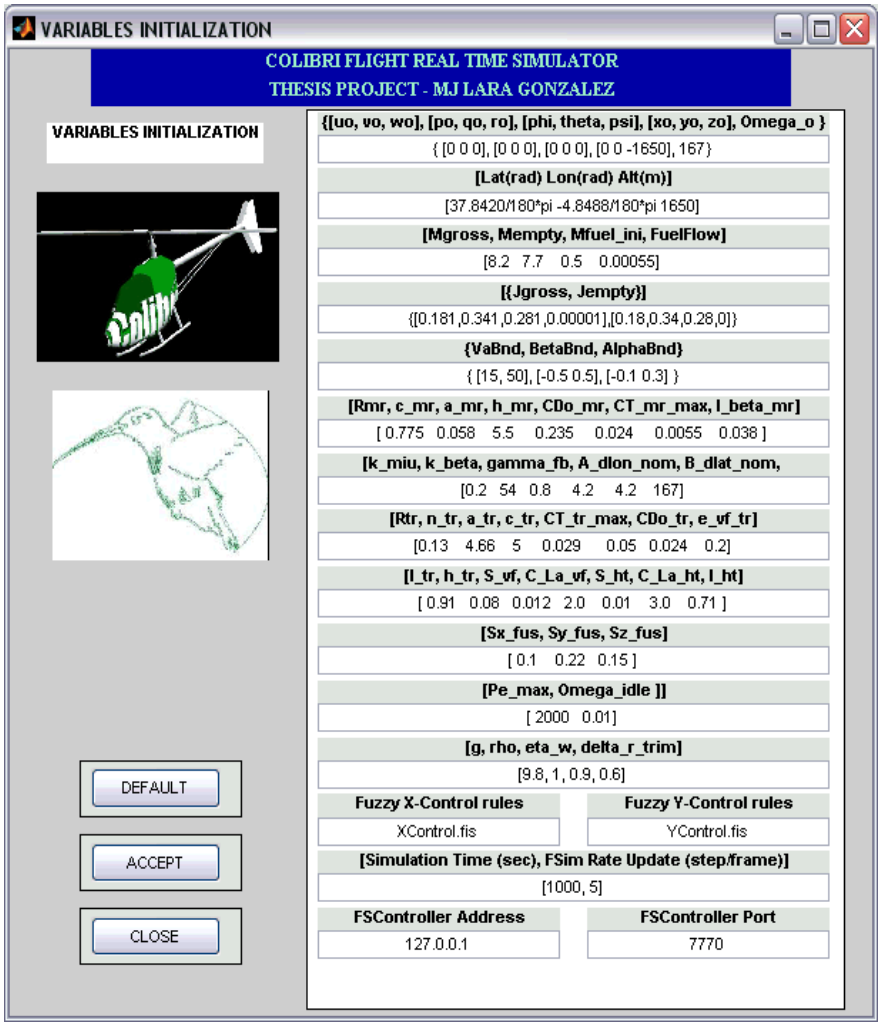


Figura 3.17: Inicialización de las variables a través de la interfaz *InitilizeModel*

3.6 CONFIGURACIÓN DE LA SIMULACIÓN EN MODALIDAD NORMAL

Para que la simulación en modalidad normal tenga éxito, el directorio que contenga la simulación deberá contener además los ficheros contenidos en la siguiente tabla:

Nombre del fichero	Descripción
FlightSimUpdate.dll	S-Function para la conexión con Flight Simulator

RTBlock.dll	S-Function relativa al blockset real-time

3.7 CONFIGURACIÓN DE LA SIMULACIÓN EN MODALIDAD EXTERNA CON REAL-TIME WORKSHOP

Para transformar la simulación en un ejecutable es necesario modificar algunos parámetros del panel de la simulación contenidos entre las opciones de compilación de RTW, visualizables usando el comando del menú *Tools → Real Time Workshop → Options*. Las opciones a modificar son las que se presentan en la siguiente lista:

- Panel *Real Time Workshop* → recuadro *Target Selection* → Item *System Target File:* grt.tlc

- Panel *Real Time Workshop* → recuadro *Target Selection* → Item *Language:* C++

- Panel *Real Time Workshop* → recuadro *Build Process* → Item *Make Command:* make_rtw S_FUNCTIONS_LIB="..\winmm.lib..\rt.lib..\FlightSimUpdateLibRTW.lib"

Para que la simulación en modalidad externa tenga éxito, el directorio que contiene la simulación debe contener además los ficheros contenidos en la tabla que se muestra a continuación. La compilación del modelo puede ser realizada a través del comando del menú *Tools → Real Time Workshop →Build*.

Nombre del fichero	Descripción
FlightSimUpdate.dll	S-Function para la conexión con Flight Simulator
RTBlock.dll	S-Function relativa al blockset real-time
FlightSimUpdate.cpp	S-Function (C++) relativa al blockset real-time, a incluir en el ejecutable generado

sfunjoy.cpp	S-Function (C++) relativa al blockset real-time, a incluir en el ejecutable generado
RTBlock.cpp	S-Function (C++) relativa al blockset real-time, a incluir en el ejecutable generado
FlightSimUpdateLibRTW.lib	Librería de funciones para la comunicación con el protocolo TCP/IP
winmm.lib	Librería multimedia de Microsoft
rt.lib	Librería de funciones para la gestión del real-time

3.8 RESUMEN PASO A PASO PARA REALIZAR LA SIMULACIÓN EN TIEMPO REAL

A continuación explicaremos los pasos a seguir para realizar la simulación en tiempo real, tanto en modalidad normal como externa.

En primer lugar, nos aseguraremos de haber instalado las siguientes herramientas:

- Matlab/Simulink (además instalaremos Visual C++ y lo elegiremos como compilador de Matlab mediante el comando *mex -setup*; será el que se usará a la hora de *construir* nuestro modelo de tiempo real en Real Time Workshop – modalidad externa).

- blockset de Aerosim: al instalarlo elegiremos la opción de añadir dicho blockset al path de Matlab y así poder usarlo como un blockset más de Simulink.

- Microsoft Flight Simulator: una vez instalado, copiaremos la librería fsuipc.dll (podemos descargar FSUIPC como una de las utilidades del MSFS en la página web de Aerosim) en la carpeta *modules* que encontraremos dentro de la carpeta del programa; esto, como ya se dijo anteriormente, nos servirá para habilitar la comunicación IPC que tendrá lugar entre el FSController y el Flight

Simulator.

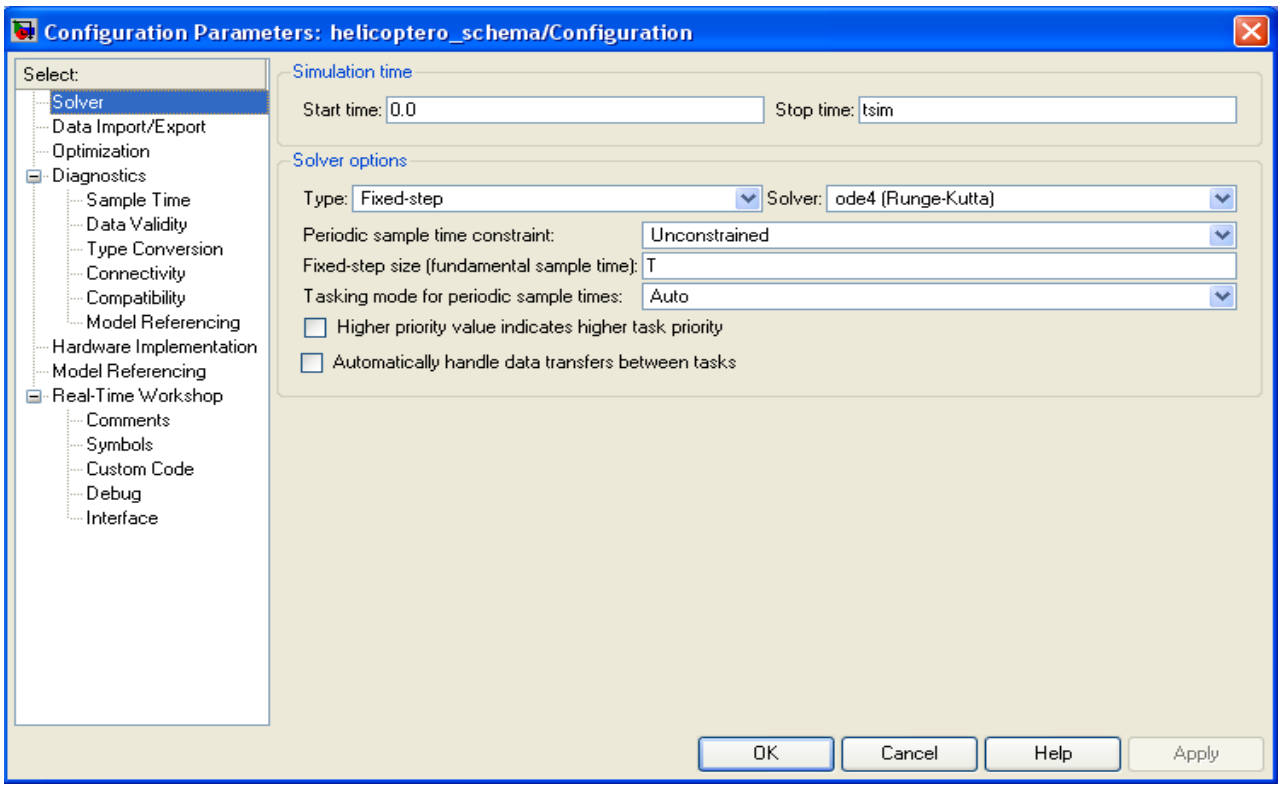
Una vez instaladas todas las herramientas mencionadas anteriormente, sólo tendremos que seguir los pasos que enumeramos a continuación para llevar a cabo nuestra simulación en tiempo real[7].

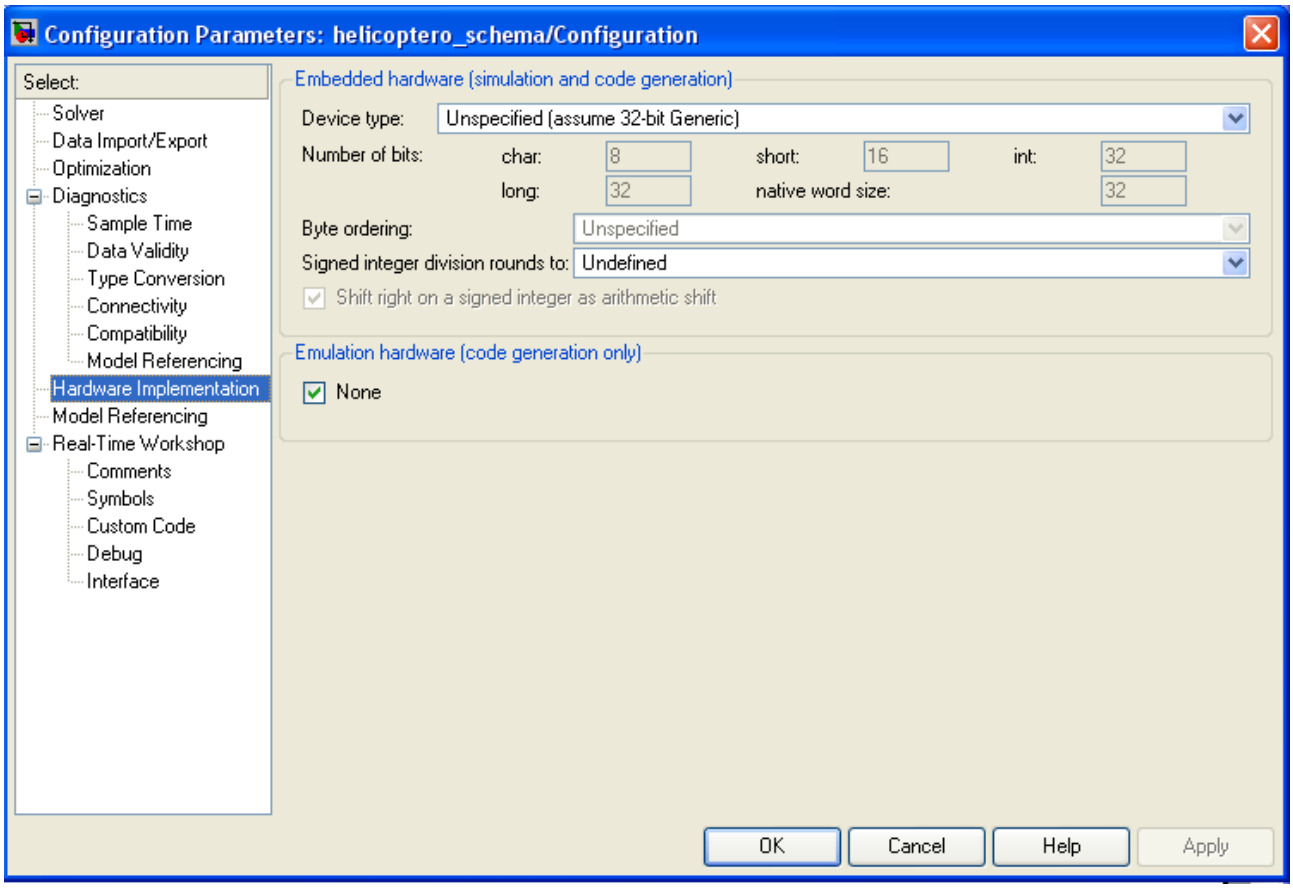
3.8.1 Simulación en tiempo real en modalidad normal

En primer lugar abrimos Matlab y, por ejemplo en la carpeta *work*, será dónde tengamos nuestro modelo de helicóptero *helicoptero_schema.mdl* (modelo completo), junto con los ficheros mencionados en el apartado **3.6**, y los ficheros *InitializeModel.m*, *inicio.m* e *ini_variables.m* que utilizaremos para la inicialización de los parámetros del modelo. *InitializeModel.m* llamará a *inicio.m* cuando abramos por primera vez la interfaz de inicialización de parámetros o cuando pulsemos DEFAULT. En cambio, *ini_variables.m,* fichero al que se llama cuando pulsamos la opción ACCEPT, lee los datos introducidos en la interfaz de inicialización, que son de tipo string y los convierte en números. Pulsando el botón CLOSE, los datos que hemos introducido en la interfaz non serán interpretados y simplemente volveremos a asumir los valores que había precedentemente en la última apertura de la interfaz.

Hay que tener siempre en cuenta que para que haya conexión entre Simulink y el MSFS, tenemos que ejecutar FSController.exe (en la misma máquina donde se esté ejecutando MSFS). Pero para que esta conexión funcione tenemos que cargar antes la librería fsuipc.dll (ya que ésta es la que permite la conexión IPC entre el FSController y el MSFS). Por lo tanto, abriremos el MSFS (de esta forma cargaremos fsuipc.dll), luego ejecutaremos FSController.exe y a continuación sólo tendremos que pulsar *Start simulation* en la barra de herramientas de nuestro modelo para comenzar con la simulación en tiempo real y modalidad normal. Eso sí, previamente habremos configurado algunos parámetros de simulación (sobre la barra de herramientas de nuestro modelo vamos a *Simulation → Configuration Parameters...*) del modo que sigue:

7 Para que la conexión entre dos máquinas funcione, es necesario configurar en el panel abierto con *InitializeModel.m* la dirección IP de la máquina en la cual se ejecuta el Flight Simulator: 127.0.0.1. De hecho, se corresponde en la convención internacional con la dirección interna de la máquina y se utilizará sólo en el caso en el que el Flight Simulator y el Simulink se ejecuten en la misma máquina.





Además de visualizar los resultados de la simulación en el MSFS podemos ver

toda la información que deseemos durante la simulación, a través de bloques de SCOPE, DISPLAY, etc...

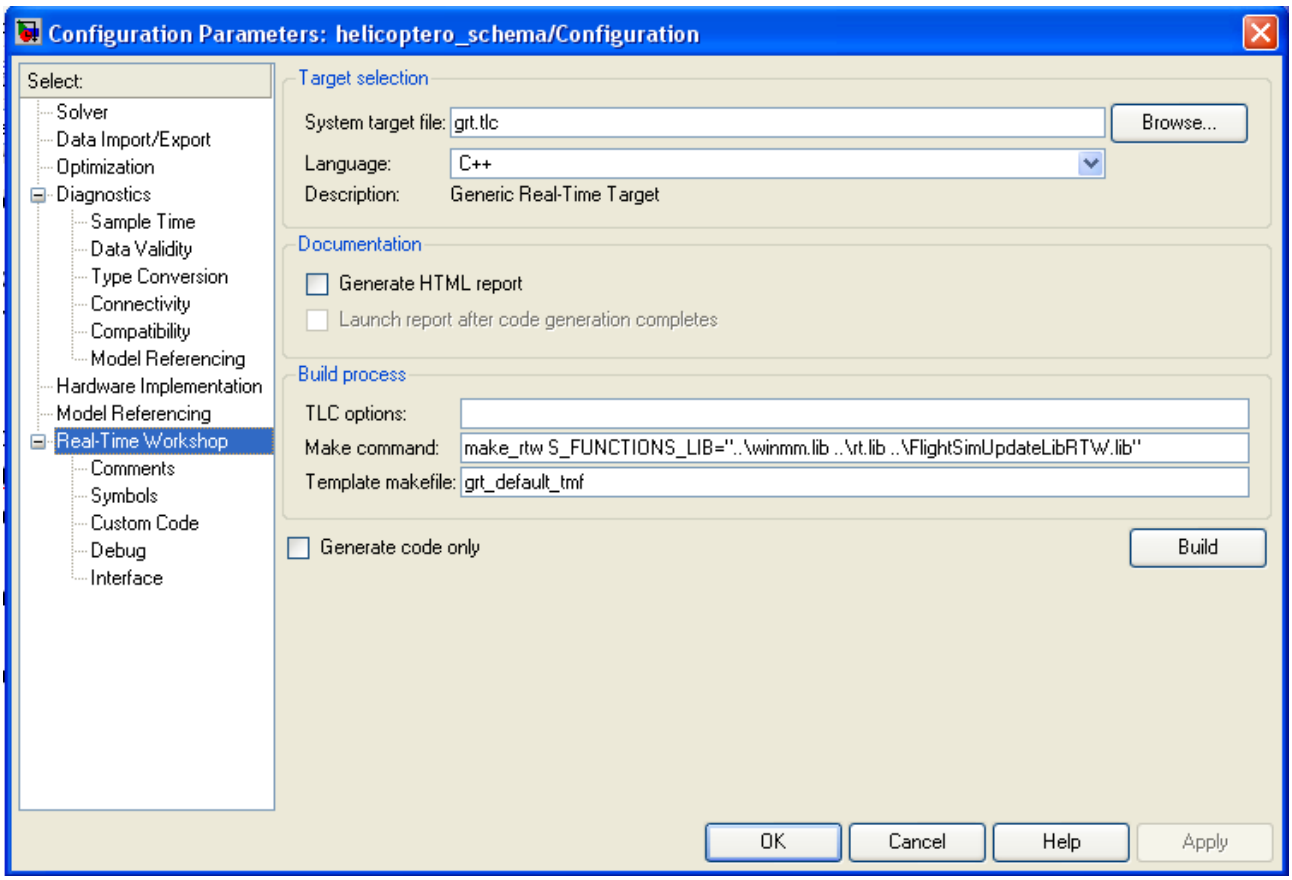
La simulación en modalidad normal suele utilizarse para ver que todas las conexiones funcionan correctamente, y una vez que veamos que todo funciona correctamente podremos pasar a la simulación en modalidad externa, la cual nos proporciona una simulación en tiempo real sin retrasos en condiciones críticas (en la modalidad externa el proceso posee una prioridad mucho mayor y por lo tanto no habrá prácticamente interrupciones – el hecho de que el usuario esté por ejemplo abriendo una ventana o haciendo cualquier otra cosa no influirá en nuestra simulación, es decir, non la ralentizará).

3.8.2 Simulación en tiempo real en modalidad externa

En este caso necesitaremos al igual que en el caso anterior, nuestro modelo *helicoptero_schema.mdl*, los ficheros para la inicialización de parámetros y el resto de ficheros mencionados en el apartado **3.7**. Vemos que en este caso necesitamos además algunos ficheros *.cpp y *.lib. Esto es porque a la hora de *construir* nuestro ejecutable de tiempo real, necesitaremos compilar *en línea* todo el código, de manera que obtendremos las *.dll a partir de los *.cpp y de los *.lib. En el caso de la modalidad normal, el Simulink se conecta a las *.dll, las cuales tienen todas una interfaz común (la cual es reconocida por el Simulink). Pero la conexión a una *.dll externa no es eficiente. En cambio, cuando compilamos con RTW, cogemos todo el código que está dentro de la *.dll y lo metemos dentro de un único proceso, y este proceso, además de ser más eficiente, es independiente de Simulink (le puedes asignar una prioridad,... es decir, le puedes asignar características que al Simulink no le puedes asignar).

Una vez que dispongamos de todos estos ficheros en nuestra carpeta de trabajo de Matlab, pasaremos a configurar los parámetros de simulación y a "construir" nuestro modelo.

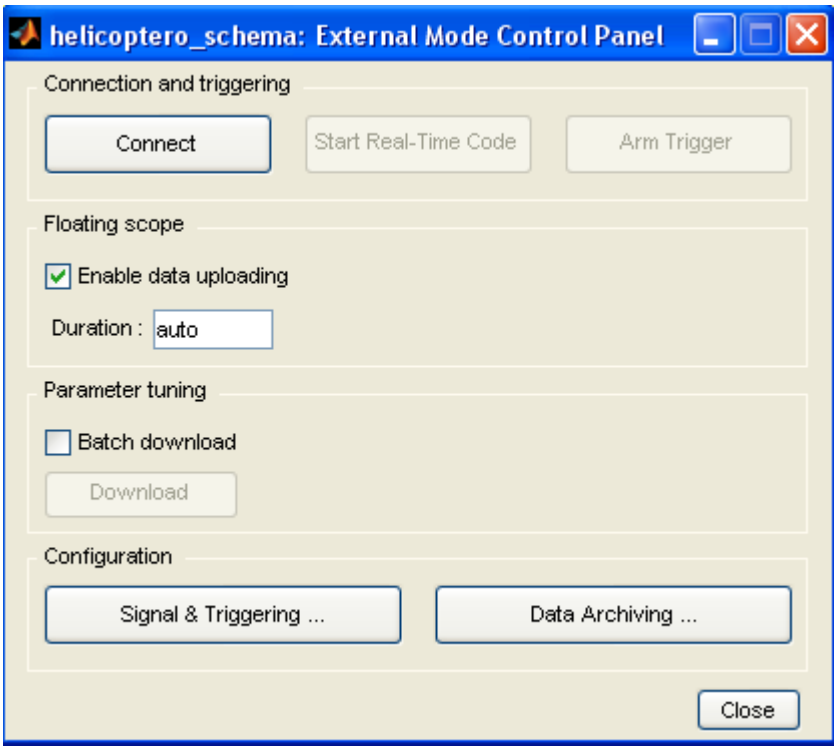
En primer lugar, abrimos *helicoptero_schema.mdl* y sobre la barra de herramientas abrimos *Simulation → Configuration Parameters...* Las opciones a modificar serán las siguientes:



Después de haber configurado los parámetros de simulación, pasaremos a construir nuestro modelo o ejecutable de tiempo real. Para ello iremos, sobre la barra de herramientas de nuestro modelo Simulink, a *Tools → Real-Time Workshop → Build Model*. Una vez finalizado dicho proceso, veremos que en nuestra carpeta de trabajo, se ha generado, entre otras cosas, un ejecutable con el nombre de nuestro modelo *helicoptero_schema.exe*. Éste será el ejecutable de nuestro modelo en tiempo real. Ahora ya, para realizar la simulación en modalidad externa, tendremos que, como en el caso anterior, abrir el MSFS de manera que se cargue la librería fsuipc.dll, ejecutar FSController.exe y finalmente ejecutar el *.exe que acabamos de generar, *helicoptero_schema.exe*.

En este caso veremos como obtenemos una simulación sin retrasos (con respecto a aquella en modalidad normal).

Si además quisiéramos ver algún tipo de dato representado en SCOPE, tendríamos que configurarlo, ya que por defecto, en la modalidad externa no podríamos visualizar resultados sobre SCOPE, tal y como podíamos hacer en modalidad normal sin ningún tipo de problema. La configuración de los SCOPEs que queremos ver se haría de la siguiente forma: sobre la barra de herramientas de nuestro modelo iríamos a *Tools → External Mode Control Panel → Signals & Triggering*.



Ahí elegiríamos los SCOPEs que queremos visualizar, y a continuación, sobre la misma ventana anterior, haríamos *Connect* y luego *Arm Trigger*. Si queremos además archivar algunos de estos datos, podemos elegir la opción de *Data Archiving*.

Finalmente, añadir que, para un mejor manejo de todas las aplicaciones, es conveniente ejecutar el MSFS en modo de pantalla minimizada (para así poder controlar mejor el resto de aplicaciones).