Trabajo Fin de Grado
ETSID
UNIVERSIDAD POLITECNICA DE VALENCIA

DISEÑO Y VALIDACIÓN DE SISTEMAS DE CONTROL PARA AERONAVES BASADO EN LAS HERRAMIENTAS SOFTWARE FLIGHTGEAR Y MATLAB©. APLICACIÓN AL DISEÑO DE PILOTOS AUTOMÁTICOS

Documentos:

Autor: **Pablo Brusola Fernández-Portolés**
Director de proyecto: **Xavier Blasco Ferragud**
Especialidad: **Ingeniería de Sistemas Automáticos**
Valencia, 5 de Septiembre de 2016

Trabajo Fin de Grado
ETSID
UNIVERSIDAD POLITECNICA DE VALENCIA

DISEÑO Y VALIDACIÓN DE SISTEMAS DE CONTROL PARA AERONAVES BASADO EN LAS HERRAMIENTAS SOFTWARE FLIGHTGEAR Y MATLAB©. APLICACIÓN AL DISEÑO DE PILOTOS AUTOMÁTICOS

1. MEMORIA

Autor: **Pablo Brusola Fernández-Portolés**
Director de proyecto: **Xavier Blasco Ferragud**
Especialidad: **Ingeniería de Sistemas Automáticos**
Valencia, 5 de Septiembre de 2016

Índice

Índice de Tablas:

Índice de figuras:

1. Objeto del proyecto.

1.1 Introducción:

Hoy en día a la hora de diseñar un controlador para cualquier tipo de aeronave, es necesario realizar múltiples pruebas y ensayos para asegurar la integridad de la aeronave y por supuesto de la tripulación y pasajeros.

Además cada ensayo real va a producir una serie de costes tanto en gastos de consumo (combustible, horas de vuelo del piloto etc), como gastos de revisiones y reparaciones (si el controlador no funciona bien puede llegar a estrellarse). Así pues el diseño de controladores mediante interfaces y computadoras puede ser una gran opción antes de poner en práctica y equipar a una aeronave de un controlador.

Por ello para este proyecto haremos uso del simulador de vuelo FlightGear en el entorno de programación de Matlab© para diseñar y validar controladores de una manera experimental.

1.2 Objetivos:

El objetivo final, o por lo menos el más visible, es que nuestro controlador orientado a pilotos automáticos funcione, eso indicará que hemos podido diseñar y validar un buen controlador. Para ello la aeronave deberá seguir una serie de waypoints (coordenadas) que introduciremos a través del entorno de programación y veremos ejecutados en el simulador. Pero en este proyecto se quiere abarcar muchos objetivos más:

1. **Interfaz Matlab©-FlightGear**

    - Implementar una interfaz que conecte de una manera dinámica el entorno de programación Matlab© con FlightGear.
    - Implementar una plataforma en la que se puedan hacer ensayos de una manera controlada desde el entorno de programación para recoger datos con la intención de usarlos para modelar el sistema.
    - Implementar una plataforma para poder validar el diseño de los modelos y los controladores en un ensayo lo más realista posible.

2. **Simulador de vuelo FlightGear**

    - Demostrar que el simulador de código abierto FlightGear posee modelos dinámicos de vuelo fiables para hacer ensayos.
    - Abaratar los costes en ensayos no destructivos y con el simulador de código abierto (licencia gratuita).
    - Demostrar que puede ser bueno para su uso en el ámbito docente para clases de práticas.

3. **Entorno de programación Matlab©**

   - Diseñar una plataforma que optimice el modelado del sistema del avión. Implementar una plataforma para tratar los datos extraídos de forma eficaz.
   - Implementar una plataforma para el diseño y elección de controladores y su validación de forma lineal.
   - Abaratar los costes en licencias de sistemas $Toolbox^{TM}$ de Matlab©.

4. **Aeronáutica**

   - Obtener soluciones viables de los problemas aerodinámicos reales que limiten la funcionabilidad de nuestros controladores.

2. Ámbito de aplicación

2.1 Especificaciones del cliente.

En el caso de este proyecto, ya se ha mencionado que el destinatario es el departamento de Ingeniería de Sistemas y Automática, el cual quiere promover unas prácticas de diseño de controladores en la asignatura de Control Automático para grado en Ingeniería Aeroespacial más realistas.

Como la asignatura no posee las horas necesarias, ni tampoco el objetivo de analizar las ecuaciones del movimiento de una aeronave, linealizarlas y establecer funciones de transferencia a partir de ahí, se busca en este proyecto una manera más dinámica y especifica de obtener un modelo. Así pues limitaremos este proyecto al análisis experimental de las respuestas de los actuadores para esta causa.

Nuestro cliente, al ser de objetivo docente, nos deja en la postura de intentar abaratar lo máximo posible los costes de licencias de software para que afecte lo mínimo posible el presupuesto del departamento. Al venir de una entidad pública, toda optimización de coste será buena, por ello la importancia de usar el simulador de vuelo FlightGear de código abierto.

3. Antecedentes.

3.1 Simuladores de vuelo actuales.

Hoy día existe una verdadera multitud de simuladores de vuelo para ordenadores, pero en general la mayoría suelen ser de tipo bélico o de bastante mala calidad en cuanto a fiabilidad de los modelos dinámicos.

Simuladores comerciales:
Entre ellos existen los del tipo comercial, en el cual tendremos que pagar una suma de dinero para licencias. En el mercado siempre han estado compitiendo en este sector el simulador X-Plane y su competidor el Microsoft Flight Simulator. Pero este primero venció de largo a su competidor cuando sacaron el X-Plane 10 en el cual se ofrecía tal realismo en la simulación que es el único simulador de vuelo para PC que ha recibido el certificado oficial de la Administración Federal de Aviación de los Estados Unidos para entrenar a pilotos de vuelo reales (usando una versión profesional no la de escritorio). Al final Microsoft Flight Simulator se ha quedado como un juego árcade más que como un simulador de vuelo.

FlightGear:
Este simulador de código abierto que nació en 1997, surgió a causa de que un numeroso grupo de personas no estaban satisfechas con los simuladores de vuelo que existían en ese momento (el X-Plane no era lo que es ahora). Lo que más les molestaba era la poca flexibilidad que tenía el usuario con el programa. Por eso mismo decidieron empezar un simulador de vuelo multiplataforma y libre que plantase cara a los simuladores comerciales. Es el único programa de vuelo que es de código libre, el cual lo vuelve mucho más flexible y barato, pero a la vez más complejo de controlar.

Hagamos una comparativa de estos dos simuladores:

Simulador	FlightGear	X-Plane 10
Modelos dinámicos de vuelo	Tres modelos dinámicos: -JSBSim -YASim -UIUC (este ultimo creado por la NASA)	-Tiene modelos dinámicos de régimen subsónico y supersónico. -Tiene modelos de simulaciones de fallos de sistemas del avión.
Base de datos de escenarios	-20 mil aeropuertos reales -Escenario del mundo con ríos, carreteras, ciudades etc -Escenario nocturno realista. -Para mapa completo del mundo se necesitan 3 DVD	-33 aeropuertos. -Escenario del mundo completo con una resolución de 74º norte a 60º Sur.
Modelo atmosférico	Modelo a tiempo real de la atmosfera.	Modelo a tiempo real de la atmosfera. Fácil maleabilidad de ésta.
Flexible	Gran flexibilidad, puedes crear tus propios modelos de aviones. Vienen 14 por defecto.	Flexibilidad dentro de sus límites: -Puedes descargar los modelos de 1400 aviones más.

	Como poder se puede hasta crear los propios modelos aerodinámicos.	-La interfaz te ofrece la posibilidad de personalizar tu aeronave.
Conectividad	Muy buena conectividad, muchos tipos de protocolos input/output	Muy buena conectividad, muchos tipos de protocolos input/output.
Requisitos mínimos del sistema	No se necesita una gran potencia de sistema para que el simulador funcione correctamente (tampoco una maquina demasiado poco potente).	- Procesador Dual Core 2.5 GHz - 4GB de RAM - Tarjeta gráfica de 1GB de VRAM
Coste	0€ (Licencia GNU)	Varias licencias: -60$ por computadora el básico. -Para usos profesionales (que requieran el certificado FAA, licencia de 750$)-Solo una para todas las computadoras.
Certificados	No tiene validez para pruebas de vuelo oficiales.	Certificado de la FAA de EEUU para entrenar pilotos reales.
Interfaz de usuario	Interfaz poco interactiva, muchos comandos hay que ejecutarlos desde la consola.	Gran interfaz interactiva, se puede y se debe hacer todo desde el programa.
Gráficos	Gráficos poco potentes, no es la finalidad del simulador.	Gráficos muy potentes y realistas.

Tabla 1: Comparación de simulador de vuelo comercial con FlightGear

Como podemos ver, es lógico que X-Plane sea un simulador mucho más completo, pero en este proyecto, FlightGear tiene todos los requisitos que necesitamos para el uso docente y de investigación a un coste muchísimo menor. Vemos que los modelos de vuelo no tienen nada que envidiarle al simulador comercial. Por ello hemos decidido usar este simulador de código abierto. El único inconveniente puede ser su complejidad de uso, ya que esta hecho para gente preparada. Pero el objetivo de este proyecto es implementar los sistemas para que el análisis, la conectividad y el diseño usando este simulador, se vuelvan algo más sencillo.

Una vez esto claro, queda elegir que aeronave utilizaremos a lo largo de este proyecto.

3.2. Análisis de la aeronave: Cessna 172 Skyhawk

Presentación

El Cessna 172 Skyhawk es una de las avionetas más populares y comunes de todo el mundo. Los primeros modelos fueron fabricados en 1956 como una actualización del Cessna 170 con la única diferencia del tren de aterrizaje "triciclo" sustituyendo al convencional. Sin embargo a lo largo de los años se ha ido haciendo más complejo. El modelo del simulador más específicamente es el 172-P, que se empezó a fabricar en 1981.

Este modelo lleva tanto tiempo fabricándose gracias a su estabilidad a bajas velocidades y vientos, lo que le convierte en una gran avioneta para misiones relativamente estáticas (vigilancias aéreas , fumigaciones etc.).

Características generales

Característica	Valor	Unidad/Propiedad
Tripulación	1	piloto
Capacidad	3	pasajeros
Superficie Alar	16.2	m^2
Perfil Alar	2412	NACA
Peso al vacío	743	kg
Peso máximo al despegue	1110	kg
Longitud	8.3	m
Altura	2.7	m
Envergadura	11	m
Cuerda media	1.4935	m

Tabla 2: Caracteristicas generales del Cessna 172

Características del motor:

- Número de motores → 1 (Monomotor)
- Empresa de fabricación → Avco Lycoming
- Modelo → 0-320-DD2J
- Tipo → Pistón
- Cilindros → 4
- Refrigeración → Por aire

Características de la hélice:

- Empresa de fabricación → McCauley Accessory Division
- Modelo → 1C160/DTM7557
- Número de palas → 2
- Diámetro de la hélice → 75 pulgadas
- Tipo → Pala fija (fixed pitch)

Dimensiones de la Cessna 172

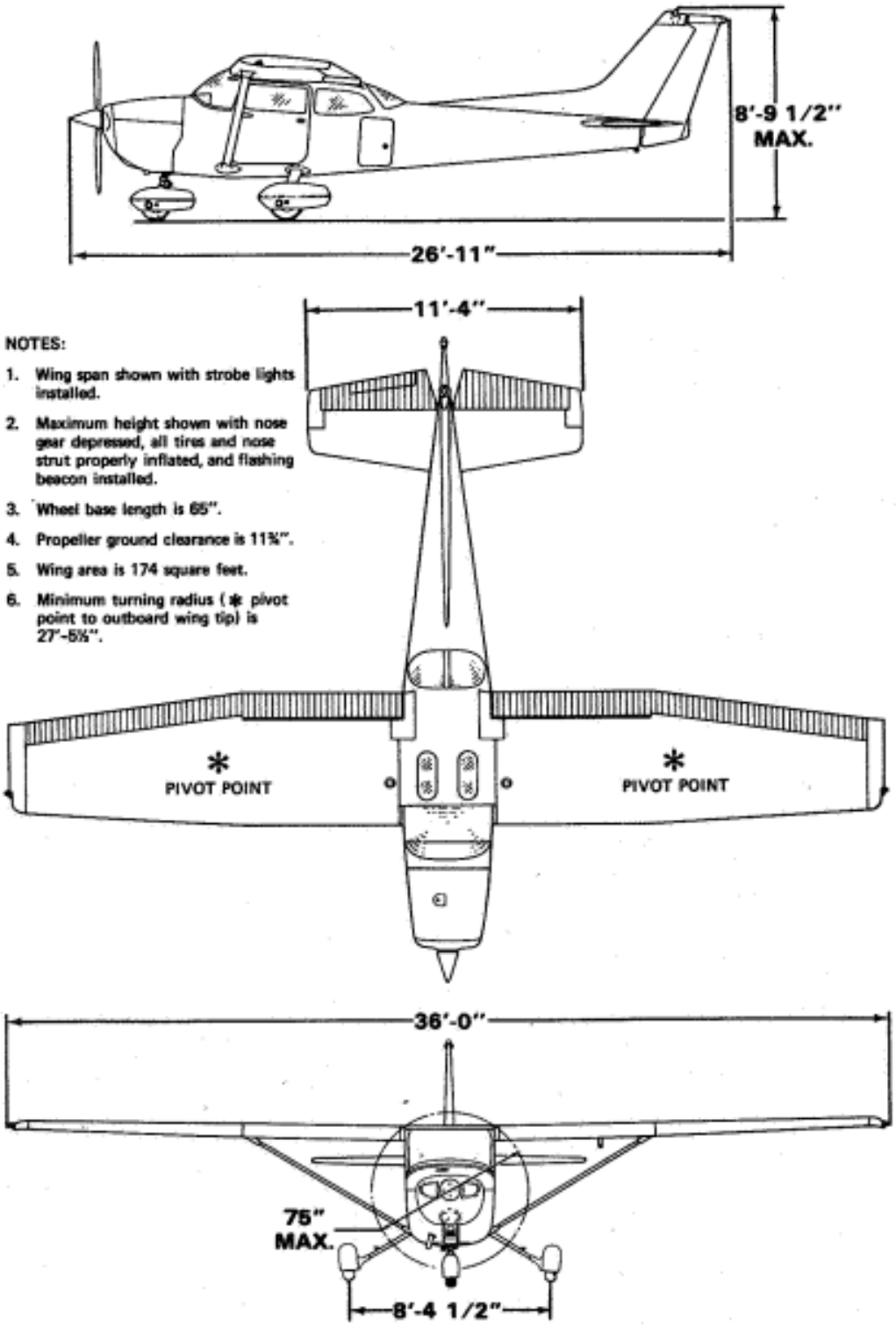


Figura 1: "Three View" de la Cessna 172 extraído de un manual de información de Cessna Aircraft Company. Unidades en pulgadas (Inches*)*

4. Dinámica y cinemática del movimiento de un avión.

4.1 Introducción:

Este estudio tiene como objetivo poder establecer desde un punto de vista teórico el comportamiento del avión para cualquier simulador de vuelo. Luego a partir de lo que conocemos de los modelos del simulador (JSBSim en este caso), lo usaremos para ampliar las ecuaciones que aquí vamos a desarrollar.

Para establecer el comportamiento del avión y a partir de este poder diseñar unos controladores, es necesario obtener las ecuaciones diferenciales que rigen estos modelos. Las ecuaciones que vamos a utilizar no son ni mucho menos nuevas, sino que provienen de la Mecánica Rotacional y esencialmente en la segunda Ley de Newton.

4.2 Objetivo:

Como hemos dicho al principio, no vamos a centrar nuestro estudio en el carácter teórico sino experimental del comportamiento del avión frente a los actuadores. Por eso mismo buscamos en este apartado en primer lugar conocer que ángulos o efectos nos tienen que preocupar analizar y en segundo lugar demostrar que no hay tiempo para analizar este modelo lineal en el tiempo de las practicas docentes, que es por lo que se ha hecho este proyecto.

4.3 Hipótesis establecidas antes del análisis matemático.

Antes de nada cabe destacar que un avión en vuelo en tres dimensiones tiene seis grados de libertad, de los cuales tres son de translación y otros tres de rotación.

Hipótesis 1: Las ecuaciones van a ser establecidas del movimiento de un cuerpo rígido, es decir la distancia entre dos puntos del avión va a ser siempre constante.

Hipótesis 2: Vamos a establecer el origen del centro de coordenadas en el CDG del avión (Centro de Gravedad), por lo que las ecuaciones de rotación serán independientes de las de translación.

Hipótesis 3: Vamos a suponer el movimiento de giro de la tierra como despreciable. Si hacemos los cálculos, siempre que la velocidad de vuelo con respecto a la tierra no supere los 900 m/s, la aceleración centrípeta y la aceleración coriolis asociadas a la rotación de la tierra nos queda que $\frac{V^2}{R_{earth}} < 1\% * g$ y que $2 * V * \Omega_{earth} < 1\% g$

Hipótesis 4: Supondremos que el viento es nulo.

Hipótesis 5: Supondremos una atmósfera sin turbulencias.

Hipótesis 6: Para una primera aproximación, no incluiremos los efectos de los pares del motor y de la hélice sobre la inercia del avión, lo detallaremos los efectos.

Hipótesis 7: Vamos a suponer que la masa del avión va a ser constante. Vamos a realizar el diseño de un controlador para pequeños giros intentando no estar mucho rato en vuelo ya que la masa del avión variará según el consumo de combustible.

Hipótesis 8: Vamos a suponer los flaps siempre a 0.

4.4 Sistemas de ejes de referencia:

Ahora que ya hemos establecido las hipótesis que nos van a permitir simplificar un poco los cálculos, es necesario comprender y establecer en que ejes de referencia vamos a trabajar y como pasar de uno a otro sistema de referencia. Esto es necesario ya que para calcular fuerzas y momentos, necesitaremos la proyección de sus componentes en el sistema de referencia que hayamos escogido.

En Mecánica de vuelo, se utilizan sobre todo estos tres sistemas de referencia.

- **Sistema Ejes Cuerpo ("Body Axes"):** Este sistema tiene el origen en el centro de masas del avión, y los ejes están fijos con respecto a la estructura. Si el avión gira, el eje girará con respecto a este. En la figura 2 se muestran los distintos ejes.

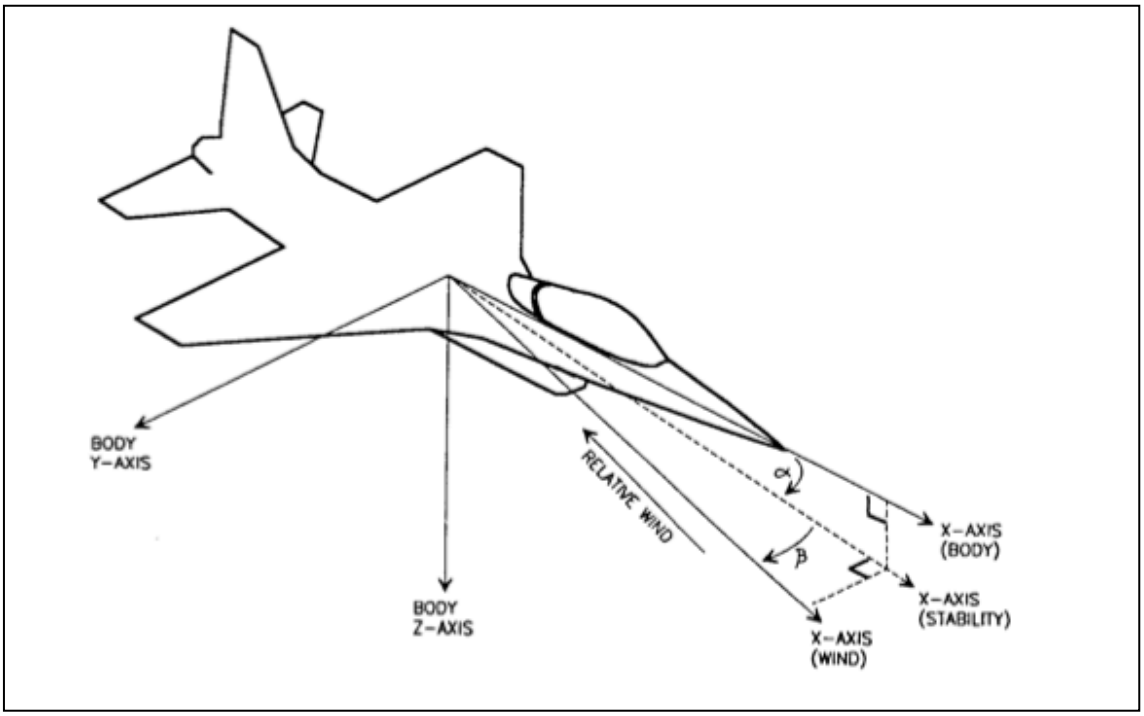


Figura 2: Sistema de referencia ejes cuerpo en una aeronave.

1. Eje Xb: Dirección longitudinal al avión cuyo sentido positivo lo estableceremos. hacia delante (sentido del vuelo).
2. Eje Yb: Perpendicular al plano de simetría del avión.
3. Eje Zb: Perpendicular a los anteriores cuyo sentido positivo lo estableceremos hacia abajo.

Cabe destacar, y esto es un punto muy importante para el estudio posterior, que la dirección del viento relativo no coincide con el eje Xb ni tampoco es perpendicular a Yb, sino que incide con cierto ángulo.

- El ángulo de ataque del avión (α): Así pues llamaremos al ángulo que existe entre el vector de viento relativo y el eje Xb.
- El ángulo de deslizamiento (β): Será por otra parte, el ángulo que formarán el viento con el eje Yb.
- **Sistema Ejes Tierra:** El sistema esta fijo a la superficie en la tierra con su centro de coordenadas al nivel del mar. Consideraremos este sistema como aproximadamente inercial.

1. Eje Xt: Paralelo a la superficie terrestre cuyo sentido positivo es el Norte
2. Eje Yt: Paralelo a la superficie terrestre, perpendicular al eje Xt y cuyo sentido positivo es hacia el Este.
3. Eje Zt: Perpendicular de los dos anteriores cuyo sentido positivo es hacia abajo.

- **Sistema de Ejes Horizonte Local:** Este sistema sus ejes son paralelos a sus correspondientes del sistema eje tierra. Sin embargo su centro de coordenadas se sitúa en el centro de gravedad del avión. Esto quiere decir que el sistema se mueve con el avión desde un punto de vista de translación, pero no se mueve cuando efectúa un movimiento de rotación.

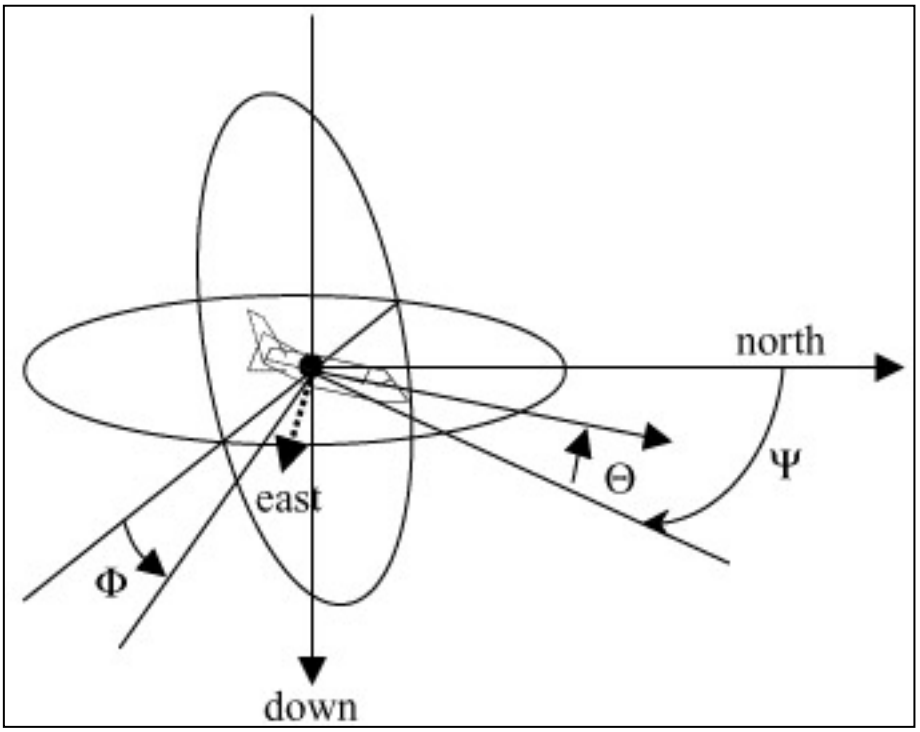


Figura 3: Dibujo sistema de ejes horizonte local

1. Eje XHL: Dirigido hacia el Norte.
2. Eje YHL: Dirigido hacia el Este.
3. Eje ZHL: Dirigido hacia abajo.

Como hemos mencionado antes, este eje obedecerá a los movimientos de translación, pero no de rotación. Así pues cuando el avión esté rotando, los ángulos que se formarán entre el Sistema de Horizonte Local y el Sistema Cuerpo, serán los ángulos de Euler, cuyos nombres serán:

$$\left\{\begin{array}{l|c} \theta & \acute{a}ngulo\ de\ asiento\ longitudinal \\ \phi & \acute{a}ngulo\ de\ asiento\ lateral \\ \psi & \acute{a}ngulo\ de\ rumbo \end{array}\right\}$$

- θ designa el ángulo de cabeceo del avión (también llamado Pitch) es el ángulo que se forma cuando se produce una rotación alrededor del eje YHL.

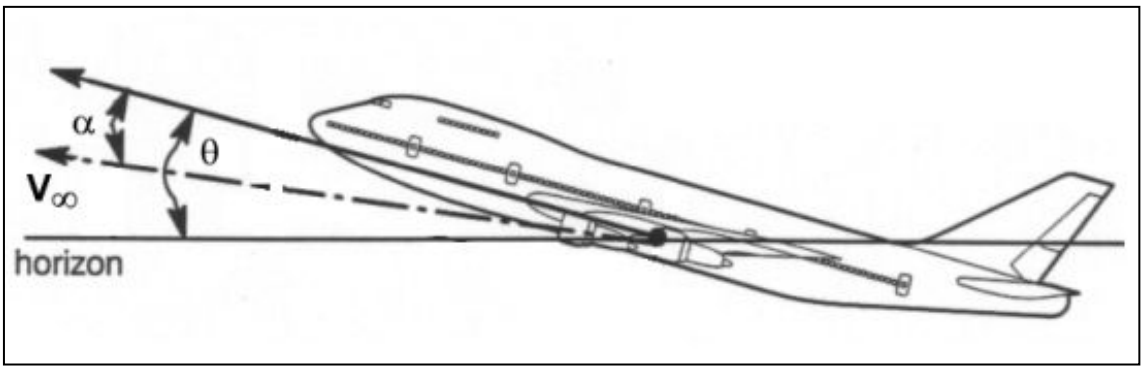


Figura 4: Dibujo del cabeceo de una aeronave

Si solo se girase con respecto a ese eje, la matriz de rotación sería la siguiente:

$$m\theta = \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix}$$

- ϕ designa el alabeo del avión ("Roll"), que es el ángulo que se forma entre los dos ejes Y de los sistemas. Es decir que aparece cuando existe una rotación alrededor del eje

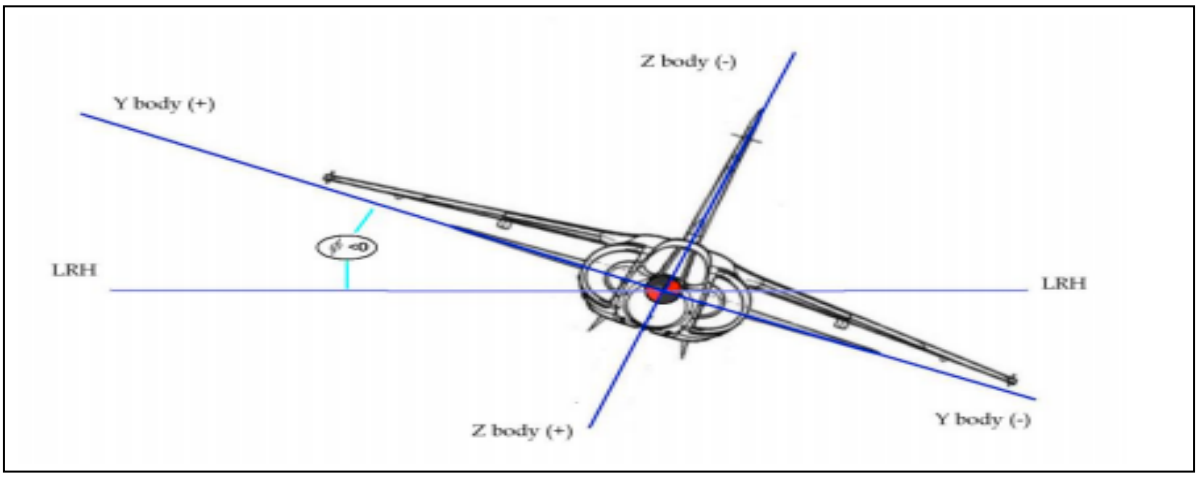


Figura 5: Dibujo del alabeo de una aeronave

Si únicamente girásemos sobre este eje, la matriz de rotación que resultaría sería la siguiente:

$$m\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{bmatrix}$$

- ψ designa la Guiñada ("Yaw"), que es el ángulo que se forma cuando se produce una rotación alrededor del eje Z. Es el famoso ángulo que indica el rumbo del avión.

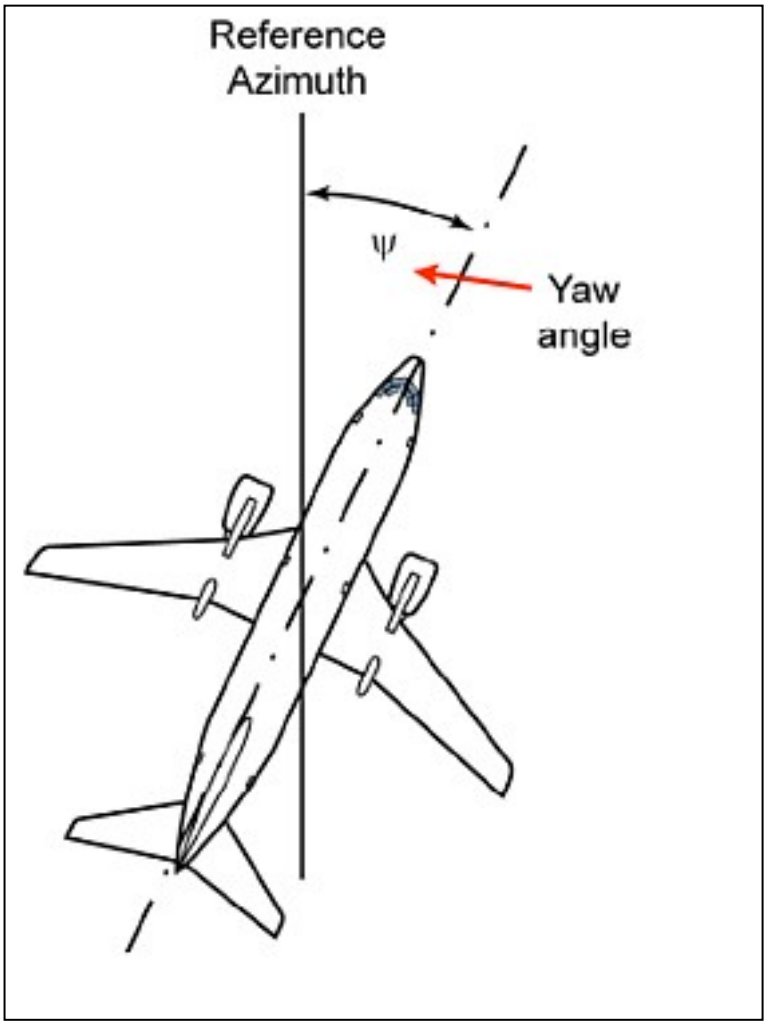


Figura 6: Dibujo de la Guiñada en una aeronave

Si únicamente girásemos alrededor del eje Z, la matriz de rotación que resultaría sería la siguiente:

$$m\psi = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

4.5 Matrices de Paso

Dados los distintos sistemas de referencia es necesario establecer unas matrices de rotación para poder obtener las proyecciones en los distintos ejes a nuestro antojo. Como veremos más adelante, el sistema de referencia que vamos a emplear para nuestra ecuaciones será el sistema de referencia Cuerpo, ya veremos por qué resulta más ventajoso de este modo.

De Horizonte Local a Cuerpo:
Así pues lo primero de todo es establecer la matriz de paso de Horizonte a Ejes Cuerpo , que será la resultante del producto de las tres matrices anteriores, es decir:

$$[HLtoB] = [m\theta] * [m\phi] * [m\psi]$$

Cuyo resultado es:

$$[HLtoB] = \begin{bmatrix} cos\phi cos\theta & sin\phi cos\theta & -sin\theta \\ -sin\psi cos\phi + cos\psi sin\theta sin\phi & cos\psi cos\phi + sin\psi sin\theta sin\phi & sin\phi cos\theta \\ sin\psi sin\phi + cos\psi sin\theta cos\phi & -cos\psi sin\phi + sin\psi sin\theta cos\phi & cos\phi cos\theta \end{bmatrix}$$

Así pues podremos obtener las proyecciones de cualquier vector en coordenadas de Horizonte Local en el sistema Cuerpo.

Y además simplemente haciendo la inversa de [HLtoB], podremos obtener lo contrario, es decir pasar de Cuerpo a Horizonte local. Al ser una matriz ortogonal, si inversa equivale a la transpuesta, asi que:

$$[BtoHL] = [HLtoB]^{-1} = [HLtoB]^{T}$$

De "ejes viento" a Cuerpo:

Como hemos mencionado antes, el sistema ejes Cuerpo, no coincide con el vector del viento relativo. Esto es importante apuntarlo ya que las componentes de las fuerzas aerodinámicas que veremos luego (Lift, Side y Drag), están situadas paralelas o perpendiculares al vector del viento. Así pues, parece coherente establecer una matriz de paso de ejes viento a ejes Cuerpo.

Si solo hubiese ángulo de ataque pero no de deslizamiento, su matriz de rotación sería la siguiente:

$$m\alpha = \begin{bmatrix} cos\alpha & 0 & -sin\alpha \\ 0 & 1 & 0 \\ sin\alpha & 0 & cos\alpha \end{bmatrix}$$

Por el otro lado, si solo hubiese ángulo de deslizamiento pero no de ataque, la matriz de rotación sería la siguiente:

$$m\beta = \begin{bmatrix} cos\beta & -sin\beta & 0 \\ sin\beta & cos\beta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Así pues para encontrar la matriz de paso de Viento relativo a Cuerpo, haciendo la multiplicación de estas dos, obtenemos:

$$[VtoB] = \begin{bmatrix} cos\alpha cos\beta & -sin\beta cos\alpha & -sin\alpha \\ sin\beta & cos\beta & 0 \\ cos\beta sin\alpha & -sin\beta sin\alpha & cos\alpha \end{bmatrix}$$

Para poder proseguir con los cálculos sin problema, usaremos la nomenclatura siguiente:

Nombre	Velocidades lineales y angulares	Momentos y fuerzas aplicados al avión	Distancias y ángulos
Frontal ("Forward")	u	Fx	x
Lateral ("Side")	v	Fy	y
Vertical	w	Fz	z
Alabeo ("Roll")	p	L	ϕ
Cabeceo ("Pitch")	q	M	θ
Guiñada ("Yaw")	r	N	ψ

Tabla 2: Tabla de la nomenclatura de las fuerzas y giros en el sistema ejes Cuerpo

En la figura 7, podemos observar los ejes que corresponden dichos elementos nombrados en la tabla de la tabla 2.

Con esto ya lo tenemos todo para empezar a deducir las ecuaciones del movimiento de la aeronave, ya sean de translación como de rotación.

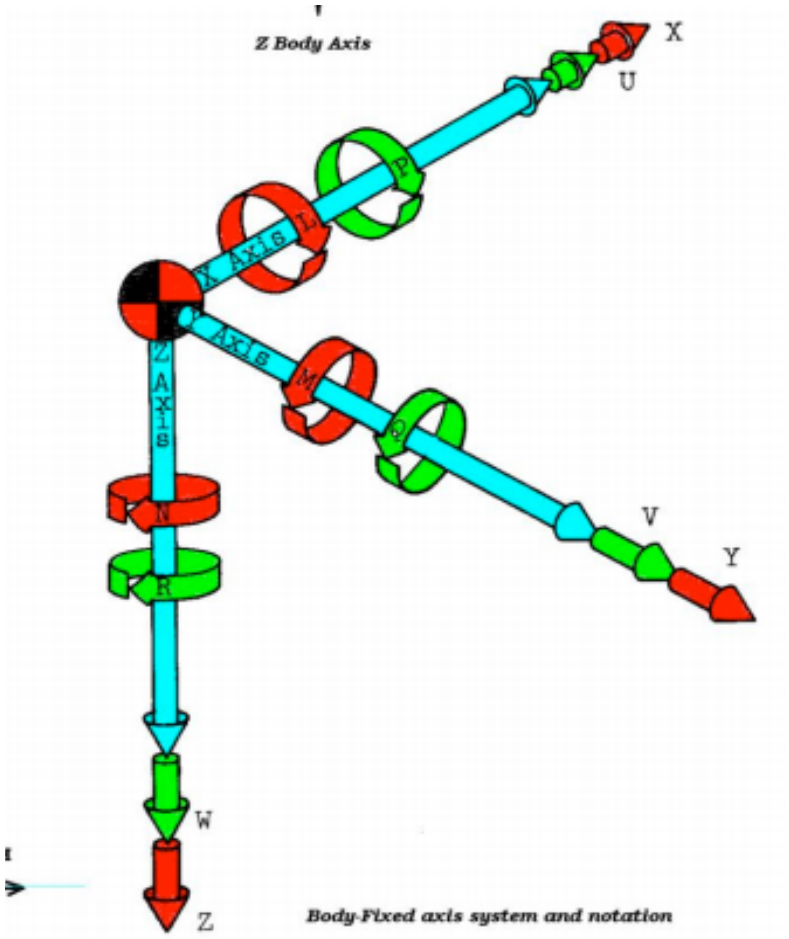


Figura 7: Visualización de las fuerzas y momentos en los ejes cuerpo.

4.6 Ecuaciones de traslación:

Como hemos dicho, vamos a usar el sistema de referencia en ejes Cuerpo. Así pues tendremos:

- Velocidad de traslación del avión → $\vec{V} = \{u, v, w\}$
- Velocidad angular del avión → $\vec{\omega} = \{p, q, r\}$
- Fuerzas externas y propulsivas → $\vec{F} = \{Fx, Fy, Fz\}$

Aplicando la primera Ley de la Mecánica sabemos que:

$$\vec{F} = \{Fx, Fy, Fz\} = m * a = m * \dot{V}_{abs}$$

Siendo

$$\dot{V}_{abs} = \dot{V}_{rel} + \vec{\omega} * \vec{V} = \{\dot{u}, \dot{v}, \dot{w}\} + \{p, q, r\} \text{ x } \{u, v, w\}$$
$$\dot{V}_{abs} = \{\dot{u} + qw - rv, \dot{v} + ru - pw, \dot{w} + pv - qu\}$$

Así pues llegamos a las tres ecuaciones de las fuerzas:

$$[1]\ m(\dot{u} + qw - rv) = Fx$$
$$[2]\ m(\dot{v} + ru - pw) = Fy$$
$$[3]\ m(\dot{w} + pv - qu) = Fz$$

Las fuerzas que afectan a un avión (para nuestras ecuaciones simplificadas) van a ser el peso (fuerza de la gravedad), las fuerzas propulsivas (el empuje en nuestro caso) y las fuerzas aerodinámicas (Lift, Drag y Side).

Fuerza de la gravedad:

La gravedad se encuentra en el sistema de referencias horizonte local, en el eje Z. Pues bien usando la matriz de paso [HLtoB] de la ecuación x.x, obtenemos dicha fuerza en ejes cuerpo:

$$\begin{pmatrix} g_{xb} \\ g_{yb} \\ g_{zb} \end{pmatrix} = [HLtoB] * \begin{pmatrix} 0 \\ 0 \\ g \end{pmatrix} = \begin{pmatrix} -g * sin\theta \\ g * cos\theta * sin\phi \\ g * cos\theta * cos\phi \end{pmatrix}$$

Fuerzas aerodinámicas:

Estas fuerzas son el Lift, el Side y el Drag y están situadas ortogonalmente con respecto al viento relativo:

- Lift → Perpendicular al viento relativo con sentido hacia arriba
- Drag → Paralelo al viento relativo con sentido hacia detrás
- Side → Perpendicular a los otros dos con sentido hacia la derecha (positivo)

Así pues tendremos que:

$$FA_{Vrel} = \begin{pmatrix} -Drag \\ Side \\ -Lift \end{pmatrix}$$

Con la matriz de paso de Viento relativo a Cuerpo de la ecuación x.x nos queda que:

$$\begin{pmatrix} FA_{xb} \\ FA_{yb} \\ FA_{zb} \end{pmatrix} = [VtoB] * \begin{pmatrix} -Drag \\ Side \\ -Lift \end{pmatrix}$$

$$= \begin{pmatrix} -D * cos\alpha * cos\beta - Side * cos\alpha * sin\beta + Lift * sin\alpha \\ -D * sin\beta + Side * cos\beta \\ -D * sin\alpha * cos\beta - Side * sin\alpha * sin\beta - Lift * cos\alpha \end{pmatrix}$$

Fuerzas propulsivas:

En este caso, vamos añadir una hipótesis nueva (no por elección propia, sino por que hemos observado que el simulador de vuelo trabaja con esta hipótesis y nos puede simplificar los cálculos).

Hipótesis 8: Supondremos que no existe desviación entre el eje Xb y el empuje proporcionado por el motor.

Así pues nos quedará que:

$$\begin{pmatrix} FT_{xb} \\ FT_{yb} \\ FT_{zb} \end{pmatrix} = \begin{pmatrix} T \\ 0 \\ 0 \end{pmatrix}$$

De este modo, las ecuaciones de translación quedarán:

$$[1]\ m(\dot{u} + qw - rv) = T - D * cos\alpha * cos\beta - Side * cos\alpha * sin\beta + Lift * sin\alpha - g * sin\theta$$

$$[2]\ m\dot{(}v + ru - pw) = -D * sin\beta + Side * cos\beta + g * cos\theta * sin\phi$$

$$[3]\ m(\dot{w} + pv - qu) = -D * sin\alpha * cos\beta - Side * sin\alpha * sin\beta - Lift * cos\alpha + g * cos\theta * cos\phi$$

4.7 Ecuaciones de rotación:

- Momento angular → $\vec{H} = \{Hx, Hy, Hz\}$
- Momentos externos → $\vec{M} = \{L, M, N\}$

La derivada temporal absoluta del momento angular, es igual al momento total externo aplicado sobre el avión, es decir:

$$\vec{M} = \{L, M, N\} = \dot{\vec{H}}_{abs}$$

Donde como antes:

$$\dot{\vec{H}}_{abs} = \left(\dot{\vec{H}}_{REL}\right) + \vec{w}\ x\ \vec{H}$$

$$\dot{\vec{H}}_{REL} = (Ixx * \dot{p} - Ixy * \dot{q} - Ixz * \dot{r})\hat{\imath} + (-Iyx * \dot{p} + Iyy * \dot{q} - Iyz * \dot{r})\hat{\jmath} + (-Izx * \dot{p} - Izy * \dot{q} + Izz * \dot{r})\hat{k}$$

Así pues si desarrollamos la expresión de los momentos externos (Ecuación X), obtendremos las tres ecuaciones que definen el movimiento de rotación de la aeronave:

[4] $L = Ixx * \dot{p} - Ixz * \dot{r} - Ixz * p * q + (Izz - Iyy) * q * r - Iyz(q^2 - r^2) - Ixy * (\dot{q} - r * p)$

[5] $M = Iyy * \dot{q} + Ixz * (p^2 - r^2) + (Ixx - Izz) * p * r - Ixy * (\dot{p} + q * r) - Iyz * (\dot{r} - p * q)$

[6] $N = Izz * \dot{r} - Ixz * \dot{p} + Ixz * r * q + (Iyy - Ixx) * p * q - Ixy * (p^2 - q^2) - Iyz * (\dot{q} + r * p)$

Como hemos visto en el análisis del avión en la matriz de inercias de nuestra aeronave (matriz x.x), podemos simplificar estas tres ecuaciones por que Iyx=Ixz=Iyz=0, quedarían tal que:

[4] $L = Ixx * \dot{p} + (Izz - Iyy) * q * r$

[5] $M = Iyy * \dot{q} + (Ixx - Izz) * p * r$

[6] $N = Izz * \dot{r} + (Iyy - Ixx) * p * q$

4.8 Relación entre velocidad angular y derivadas de los ángulos de Euler:

Necesitamos saber la relación entre la velocidad angular y las derivadas de Euler ya que estos ángulos pueden ser directamente medidos mediante giróscopos. Sin embargo las velocidades angulares no son ortogonales a las derivadas de dichos ángulos.

Las relaciones desarrolladas serán tal que:

$$[7]\ p = \dot{\phi} - \dot{\psi} * sin\theta$$

$$[8]\ q = \dot{\theta} * cos\phi + \dot{\psi} * cos\theta * sin\phi$$

$$[9]\ r = \dot{\psi} * cos\theta * cos\phi - \dot{\theta} * sin\phi$$

4.9 Ecuaciones cinemáticas:

Para este caso, los ejes cuerpo no nos serán de ayuda puesto que permanece inmóvil con respecto al avión. Por eso mismo debemos utilizar la matriz de paso que habíamos calculado de ejes cuerpo a horizonte local para despejar la derivada de la posición, es decir la velocidad, en el sistema horizonte local. La resultante de esto son las tres ecuaciones cinemáticas:

$$\begin{pmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \\ \frac{dz}{dt} \end{pmatrix} = [BtoHL] * \begin{pmatrix} u \\ v \\ w \end{pmatrix}$$

$$[10]\ \frac{dx}{dt} = u\,cos\psi\,cos\theta + v(cos\psi\,sin\theta\,sin\phi - cos\phi\,sin\psi) + w(sin\theta\,cos\phi\,cos\psi + sin\phi\,sin\psi)$$

$$[11]\ \frac{dy}{dt} = u(cos\theta\,sin\psi) + v(cos\phi\,cos\psi + sin\theta\,sin\phi\,sin\psi) + w(-cos\psi\,sin\phi + cos\phi\,sin\theta\,sin\psi)$$

$$[12]\ \frac{dz}{dt} = -u * sin\theta + v(cos\theta\,sin\phi) + w(cos\theta\,cos\phi)$$

4.10 Relación entre ángulos aerodinámicos y velocidades en el cuerpo

Los ángulos aerodinámicos $\alpha\ y\ \beta$, establecen la relación entre el vector viento relativo y el sistema de referencia Cuerpo. Por ello es necesario establecer una relación entre las velocidades. Así pues la definición más aceptada es la siguiente:

$$[13]\ V = \sqrt{u^2 + v^2 + w^2}$$

$$[14]\ \alpha = ArcTan\left[\frac{w}{u}\right]$$

$$[15]\ \beta = ArcSin[\frac{v}{\sqrt{u^2 + v^2 + w^2}}]$$

4.11 Resumen general de las ecuaciones que vamos a utilizar

Este va a ser el resultado de agrupar y despejar las ecuaciones dejándolas en función de derivadas:

Ecuaciones de traslación
$$[1]\ \frac{du}{dt} = -qw + rv + \frac{T - D*cos\alpha*cos\beta - Side*cos\alpha*sin\beta + Lift*sin\alpha - g*sin\theta}{m}$$
$$[2]\ \frac{dv}{dt} = -ru + pw + \frac{-D*sin\beta + Side*cos\beta + g*cos\theta*sin\phi}{m}$$
$$[3]\ \frac{dw}{dt} = -pv + qu + \frac{-D*sin\alpha*cos\beta - Side*sin\alpha*sin\beta - Lift*cos\alpha + g*cos\theta*cos\phi}{m}$$
Ecuaciones de rotación
$$[4]\ \frac{dp}{dt} = \frac{-L + (Izz - Iyy)*q*r}{Ixx}$$
$$[5]\ \frac{dq}{dt} = \frac{-M + (Ixx - Izz)*p*r}{Iyy}$$
$$[6]\ \frac{dr}{dt} = \frac{-N + (Iyy - Ixx)*p*q}{Izz}$$
Relación entre velocidad angular y ángulos de Euler
$$[7]\ \frac{d\psi}{dt} = \frac{q\ sin\phi + r\ cos\phi}{cos\theta}$$
$$[8]\ \frac{d\theta}{dt} = q\ cos\phi - r\ sin\phi$$
$$[9]\ \frac{d\phi}{dt} = p + (q\ sin\phi + r\ cos\phi)tan\theta$$
Ecuaciones Cinemáticas
$$[10]\ \frac{dx}{dt} = u\ cos\psi\ cos\theta + v(cos\psi\ sin\theta\ sin\phi - cos\phi\ sin\psi) + w(sin\theta\ cos\phi\ cos\psi + sin\phi\ sin\psi)$$
$$[11]\ \frac{dy}{dt} = u(cos\theta\ sin\psi) + v(cos\phi\ cos\psi + sin\theta\ sin\phi\ sin\psi) + w(-cos\psi\ sin\phi + cos\phi\ sin\theta\ sin\psi)$$
$$[12]\ \frac{dz}{dt} = -u * sin\theta + v(cos\theta\ sin\phi) + w(cos\theta\ cos\phi)$$
Relación entre ángulo aerodinámicos y velocidades del cuerpo
$$[13]\ V = \sqrt{u^2 + v^2 + w^2}$$
$$[14]\ \alpha = ArcTan\left[\frac{w}{u}\right]$$
$$[15]\ \beta = ArcSin[\frac{v}{\sqrt{u^2 + v^2 + w^2}}]$$

Tabla 3: Cuadro resumen de las ecuaciones diferenciales del movimiento de una aeronave

4.12 Efectos en los aviones de hélices:

Los aviones que poseen hélices al frente del avión tienen como consecuencia algunos efectos que pueden afectar a la estabilidad de nuestro avión. Aquí pues tratamos de explicar estos fenómenos que van a afectar a nuestras fuerzas y momentos. Estos efectos son importantes comprenderlos ya que es muy difícil calcularlo y afectarán a nuestras condiciones de equilibrio.

Estela de la hélice (Prop Wash):

La hélice no va a empujar el aire horizontalmente hacia la parte trasera, sino que al proceder de un movimiento rotatorio como es el de las palas de las hélices, y el avance de la aeronave, se va a crear una forma helicoidal alrededor del fuselaje del avión (figura 8). Esto tendrá como consecuencia, que el aire que se trasiega hacia detrás, una vez llegue al empenaje vertical de la aeronave, golpeará esta por el lado izquierda, lo que provocará que este guiñe hacia la izquierda (variará el ángulo ψ).

Para conseguir contrarrestar este efecto, deberemos corregirlo con el timón de cola δr (el "rudder"). También es importante explicar, que a cuanta mas velocidad vaya la aeronave, este efecto será más despreciable a mayor velocidad de vuelo ya que el aire trasegado tenderá a una forma más horizontal.

Par de torsión de la hélice:

Según la Ley de Newton, para cada acción hay una reacción igual y opuesta. Como la hélice, que gira en sentido horario, es decir que el motor hace una fuerza para que esta gire, por la misma ley de newton, crea una fuerza reactiva que gira en sentido anti horario. Esto va a provocar que el avión gire en el eje X del avión, es decir se producirá un alabeo. Así pues deberemos corregir este alabeo lógicamente con los alerones δa, pero esto producirá una guiñada adversa, por lo que deberemos reajustar con el timón de cola δr. (Figura 9)

P-Factor:

Este fenómeno ocurre y es mayor con el ángulo de ataque del avión. Partimos de la premisa que las palas de la hélice del avión, tienen un perfil alar, es decir tienen una curvatura. Esto quiere decir que como sabemos, si aumentas dicho ángulo de ataque del perfil, aumentarás la sustentación en dicho perfil.

Pues bien, cuando un avión aumenta su ángulo de ataque, en el momento del giro de las palas, la pala que en ese momento este descendiendo (estará en el lado derecho en ese momento si gira en sentido horario), tendrá mayor ángulo de ataque que la pala que este ascendiendo (Figura 10). Así pues se producirá una guiñada a la derecha porque por la sustentación de los perfiles, habrá mayor empuje en el lado derecho de la hélice. Para corregir este efecto, deberemos hacer uso del timón de cola δr.

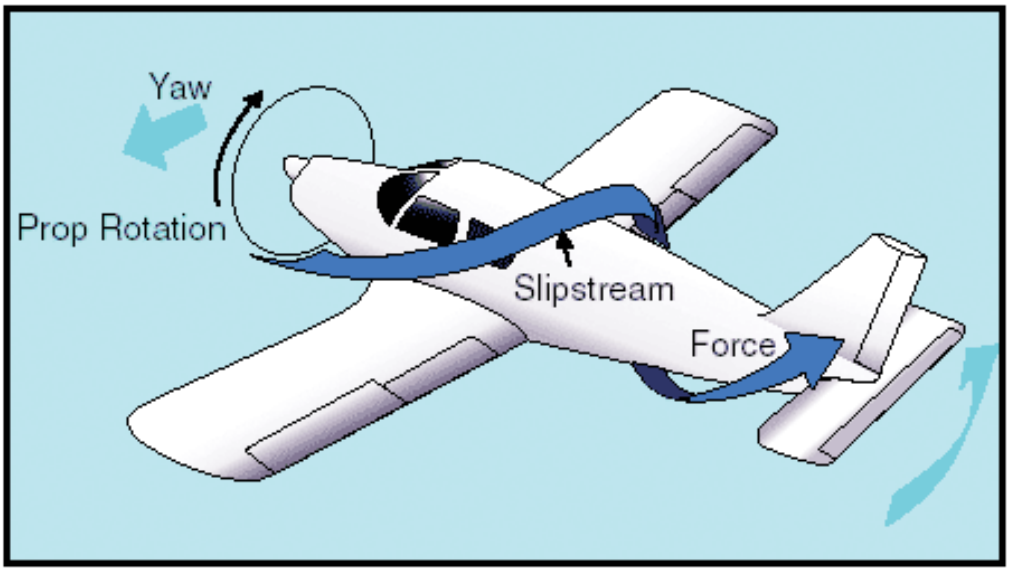


Figura 8: Dibujo del efecto PropWash de una hélice

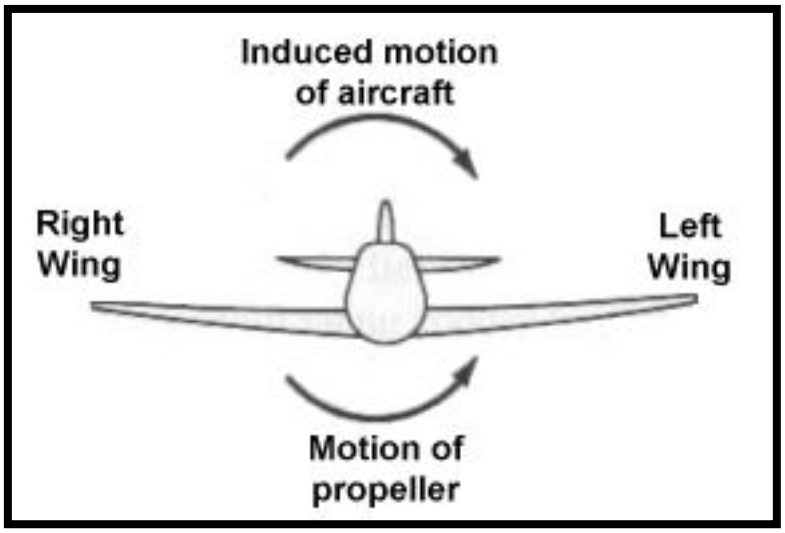


Figura 9: Dibujo del efecto Torsión en un avión.

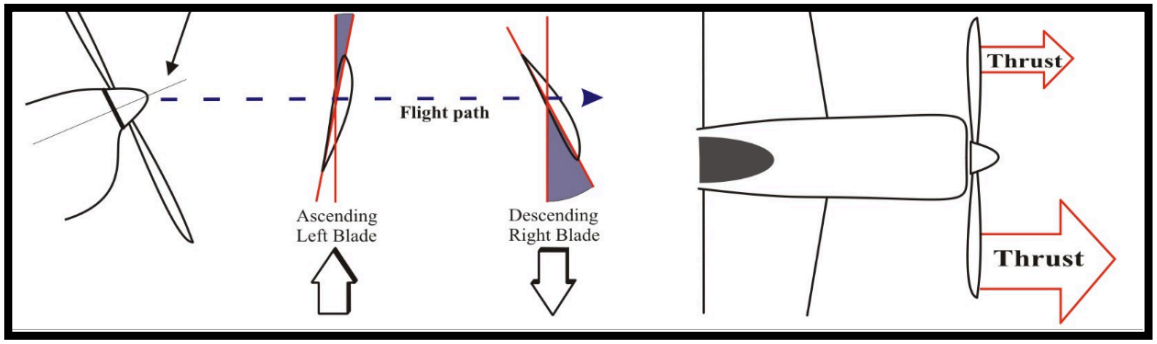


Figura 10: Esquema del efecto P-Factor en un avión.

5. Plataforma de simulador de vuelo FlightGear

5.1 Descripción de la plataforma.

FlightGear es el simulador de vuelo más completo que existe de código abierto. Esto es muy importante ya que lógicamente al ser un software libre, no se necesita de ninguna licencia para funcionar, y puesto que una de nuestras motivaciones es su uso en las aulas de prácticas, es un gran aliciente para la Universidad. Además posee la GNU GENERAL PUBLIC LICENSE, cualquier persona tiene permitido copiar y distribuir esa licencia, pero sin cambiarla.

El programa es multiplataforma, está disponible para todos los sistemas operativos lo que nos permite una alta flexibilidad a la hora de programar. Este programa aunque sea de código abierto, lo único que le tiene que envidiar a los simuladores de vuelo comerciales son los gráficos. Sin embargo a nivel de realismo de los controles y la física de las aeronaves (tanto a niveles Aerodinámicos, como propulsivos, como electrónicos etc.), FlightGear se encuentra al mismo nivel, o incluso a un nivel superior de los simuladores comerciales. Por ello lo hacen perfecto en este proyecto ya que los gráficos no nos interesan tanto como observar el comportamiento del avión a lo largo de un desarrollo del diseño y validación de controladores.

5.2 Características principales.

- Modelos Dinámicos de Vuelo: En FlightGear nos dejan la opción de elegir entre tres modelos dinámicos de vuelo según el realismo y las características de nuestra aeronave. Nosotros usaremos el genérico: JSBSim que explicaremos más adelante.
- La extensa base de datos de escenarios del mundo: con más de 20 000 aeropuertos reales incluidos, FlightGear nos devuelve los escenarios mas realistas (incluyendo a la meteorología). Esto se debe a que al ser un programa creado por voluntarios de todo el mundo, su extensión se vuelve increíble.
- Las propiedades internas están expuestas: Esto va a ser un punto muy favorable en este proyecto, ya que no solo podemos ver las variables de nuestro avión de una manera muy dinámica, sino que también podemos manipularlas a nuestros antojo.
- Múltiples opciones de conectividad: Imprescindible, ya que a lo largo de este proyecto el programa principal del controlador es Matlab, asi que necesitaremos una buena conectividad para poder tanto enviar como recibir información.

5.3 Funcionamiento y conectividad.

Nosotros en este proyecto no vamos a crear ningún modelo nuevo sino que vamos a utilizar el avión (Cessna 172) que viene por defecto en el simulador. Pero sin embargo nos interesa saber como funcionan importantes como el acceso a las propiedades del sistema, la conectividad y la escritura de variables.

El árbol de propiedades de FlightGear ("Property Tree"):
El árbol de propiedades esta considerado como el corazón de este simulador y por el cual es tan maleable y reconocido. Este sistema se dedica a proporcionar de una manera jerárquica y a tiempo real el valor de cada variable tanto del avión como del entorno. Además no solo permite observar las variables sino controlarlas, lo que parece perfecto para nuestro sistema. Antes de entender como preparar nuestro controlador, parece necesario aprender como poder controlar este árbol de propiedades.

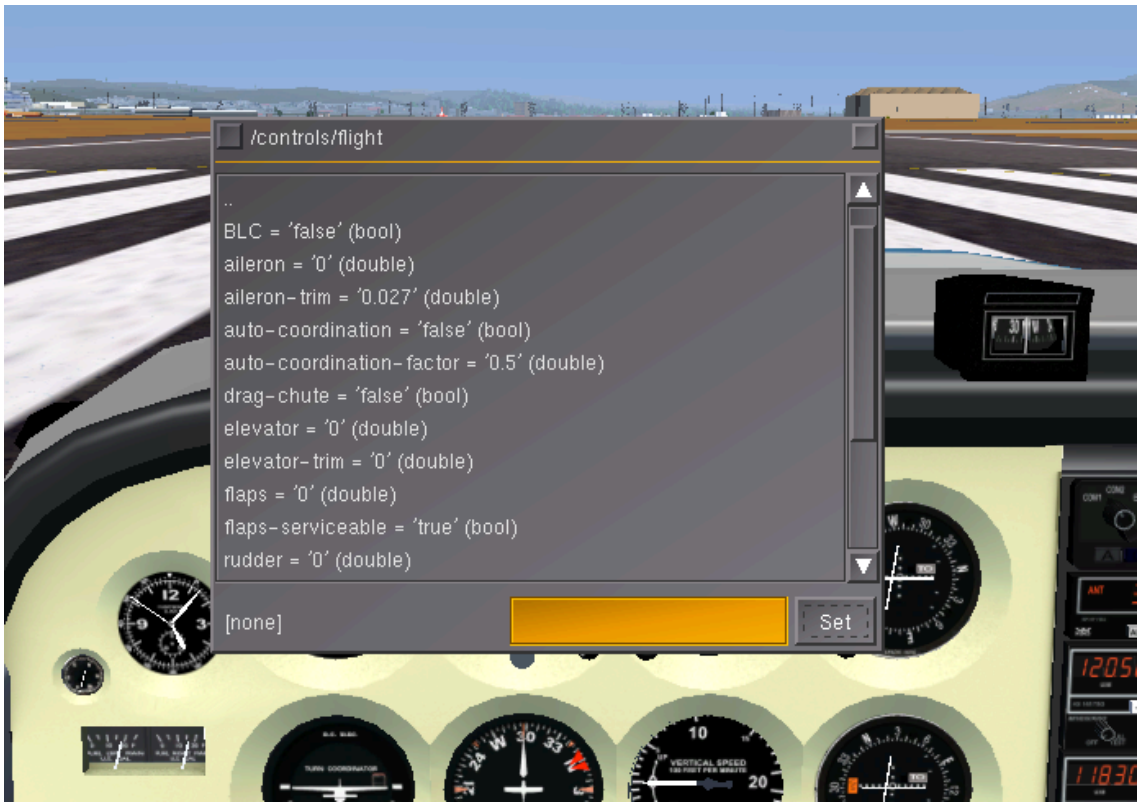


Figura 11: Visualización del árbol de propiedades desde el simulador

Existen muchas maneras de acceder a este árbol de propiedades, pero necesitamos una manera de poder acceder a ellos desde la herramienta Matlab. Se puede realizar por Web Server (servidor web), que conecta al simulador vía http a cualquier navegador que este conectada al mismo modem que donde se haya inicializado el simulador. Por otro lado están los protocolos, en el cual nosotros sólo hemos hecho usos del protocolo UDP (User Datagram Protocol).
Antes de proseguir, cabe mencionar que a la hora de establecer cualquier manera de acceder al árbol de propiedades, deberemos inicializar siempre el programa mediante la terminal del ordenador por comandos, para decirle al sistema los atributos de dicha conexión que explicaremos ahora.

Conexión por Web Server (http):

La manera de inicializar el programa para establecer una conexión de este tipo, se realiza de la manera siguiente :

```
./fgfs --httpd="puerto"
```

Figura 12 : Comandos para conexión web server

Donde "puerto" debe ser sustituido por el numero del puerto que el usuario quiera usar. Y cuando el programa se inicialice, desde nuestro navegador ya podemos acceder al árbol de propiedades metiéndonos en el servidor del programa. Para ello deberemos especificar la IP de donde se este ejecutando el programa, el puerto y decirle que queremos ir a las propiedades como muestra en la siguiente captura de pantalla:

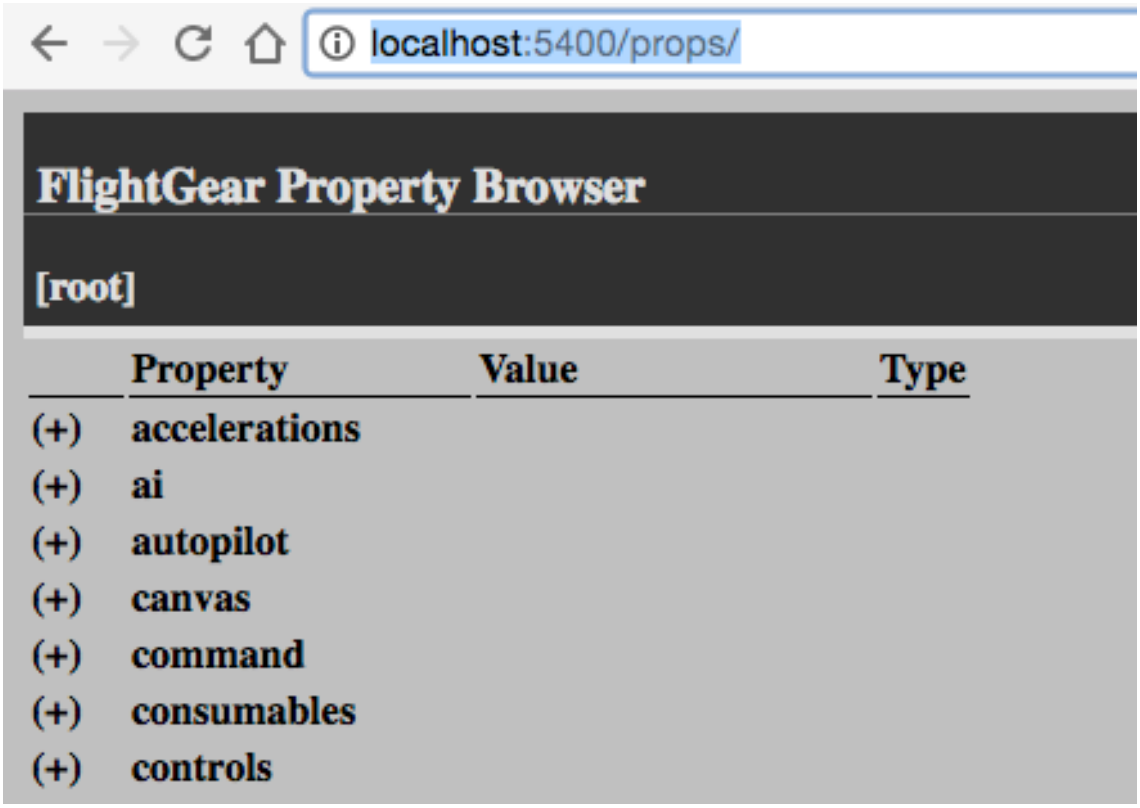


Figura 13: Navegador del árbol de propiedades vía web server.

En nuestro caso la IP será "localhost" por que nos conectamos desde el mismo ordenador que está ejecutando el simulador.

Ventajas de la conexión web server:

- Sus ventajas son su manejabilidad y su fácil acceso. Con este sistema, el explorador del los arboles de propiedades nos permite tanto explorar como modificar las variables con una gran sencillez.
- También su fácil acceso desde Matlab, ya que lo único que necesitamos es el comando urlread para que nos ayude a recoger el contenido. Lo mismo para

escribir. Dichas funciones estarán más especificadas en los apartados de programación.

Desventajas de la conexión web server:

- Su robustez: estás todo el rato dependiendo del servidor (modem), lo que significa que la velocidad de emisión y recepción puede variar.
- Los periodos de muestreo no son constantes (figura 14) , por lo que nos devuelve una respuesta deteriorada que puede afectar a nuestro análisis y nos obliga a usar métodos de interpolación para ajustar las curvas.

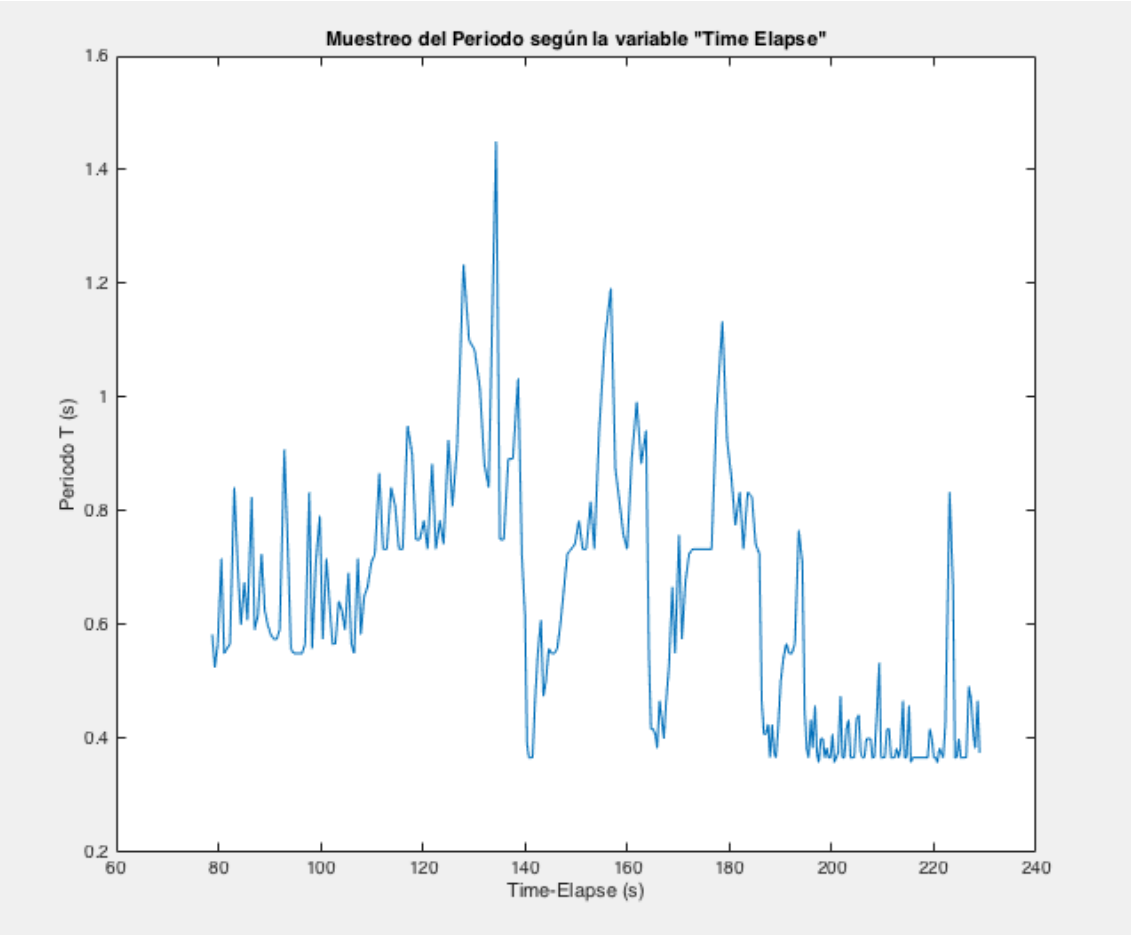


Figura 14: Muestreo del Periodo según el tiempo recogido por el simulador

Como podemos observar, tenemos desviaciones de hasta mas de un segundo, lo cual para esto sistemas puede ser bastante crucial.

Conexión por UDP:

La manera de inicializar el simulador mediante esta conexión es un poco más compleja que la anterior, deberemos establecer una comunicación por socket:

```
./fgfs --generic=socket,dir,hz,machine,port,style,protocol
```

Figura 15: Comandos para conexión por UDP

- Socket: designa el concepto que define que dos programas puedan interactuar intercambiando cualquier flujo de datos de una manera fiable (el socket depende de las características del protocolo en el que se implementa, suele ser TCP o UDP en este caso).
- dir: indica la dirección de conexión. Tres opciones: in (recibiendo), out (emitiendo) o bi (bidireccional)
- hz: Es la frecuencia con la que se procesa el canal.
- machine: es la IP de donde se esta ejecutando el programa (en nuestro caso siempre localhost)
- port: Puerto de conexión
- style: Es el estilo de protocolo que hemos mencionado antes, en nuestro caso será UDP para que el socket recoja las características de este.
- Protocol: Este se trata de un archivo .xml, que recogerá los datos que este le indique y lo enviará mediante UDP. Procedemos a explicar más detalladamente estos protocolos.

Esta manera de transmitir vía UDP con el simulador, la llaman "Generic Communication". Con esta opción nos da la posibilidad de transmitir unos datos pre configurados, los cuales están definidos en un archivo .xml. Estos datos son enviados en ASCII string o secuencia binaria.

Lo primero a especificar en estos archivos XML es si son de entrada o de salida bajo la forma: *<output> CONTENIDO </output> o <input> CONTENIDO </input>*, según la inicialización que se le ha dado anteriormente.

Acto seguido, se definen los parámetros de dichas instrucciones.

- Si queremos que emitan o reciban en ASCII o en binario (por defecto está en ASCII y así lo dejaremos),
- Saber que elemento va a separar cada paquete de datos (nosotros usaremos *<line_separator>newline</line_separator>,* por que así recogeremos un vector de una columna)
- Y por último que elemento va a separar cada dos varables (nosotros le aplicaremos la coma *<var_separator>,</var_separator>*).

Por último, queda definir que variables queremos transmitir o recibir para ello usaremos la clase *<chunk> parámetros de las variables </chunk>,* cuyas variables serán:

- <name> : Darle un nombre para usarlo de manera más sencilla, **no se transmite.**
- <node> : Lugar donde se encuentra en el árbol de propiedades.
- <type> : Tipo de variable (puede ser string, integer, float o bolean).
- <format> : Formato para guardar la varibale (como en un fprintf), con %s para string, %f para float y %d para integer.

Para ver un ejemplo real mirar la figura de nuestro input_protocol.xml implementado para este proyecto que encuentra en el anexo.

Ventajas:

- La gran ventaja de este sistema es su robustez y la capacidad para enviar los datos en periodos de tiempo más constantes.
- No dependes de ningún modem ni altas conexiones a internet.
- Emite paquetes de datos enteros. Esto es muy importante porque recibes o envías todos los datos a la vez. Por http tienes que enviarles o recibirlo uno detrás del otro, lo que provoca que los tiempos de cada variable no sean los mismos.

Desventaja:

- Su gran desventaja es la poca maleabilidad que tiene, no tienes una interfaz para visualizar las variables o cambiarlas de manera más sencilla. Siempre hay que crear un protocolo o archivos XML para poder recogerlas. Es poco útil si se desea contrastar algo instantáneamente, ya que el árbol de propiedades de FlighGear es inmenso.

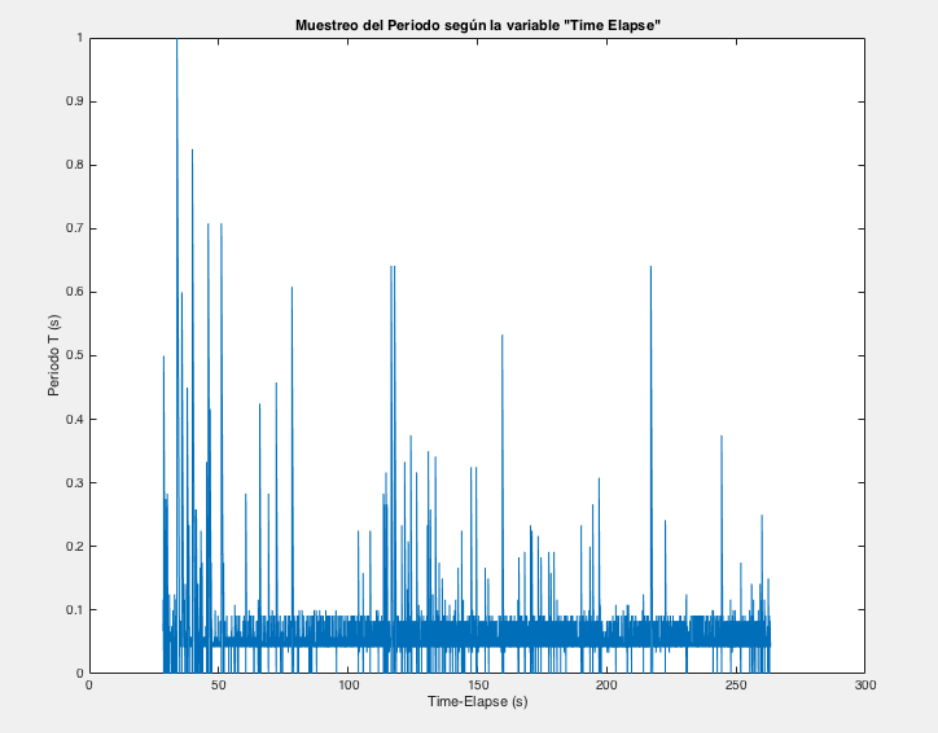


Figura 16: Muestreo de la variación del periodo según el tiempo del simulador para T=0.05

Como podemos observar el envío de datos es a mucha mayor velocidad (ya no dependemos tanto del modem) y se ajusta bastante al periodo introducido T=0.05. Los picos se deben que a veces la herramienta Matlab, intercambia mas rápido que lo que le da tiempo a procesar el simulador, por lo que se repiten tiempos.

Conclusión:

Para intentar concluir este desarrollo de conexiones, el simulador nos deja inicializar mas de una conexión a la vez, así que por comodidad (http para visualizar el gestor de propiedades) y por precisión temporal (UDP para envío de paquetes de datos con periodo fijo), vamos a inicializar los dos tipos de conexiones a la vez.

6. El modelo dinámico de vuelo JSBSim

Como hemos citado antes, habían tres tipos de modelos dinámicos en este simulador. Así que nosotros hemos hecho uso de uno de ellos: el JSBSim. Es importante conocer bien como funciona este sistema para poder proceder al análisis de nuestra aeronave.

6.1 Presentación:

JSBSim no pertenece a FlightGear ni mucho menos, igual que el, es de código abierto y se usa en muchos ámbitos ya sean simuladores de vuelo como base de cálculos para programas sin interfaz gráfica. El JSBSim proporciona esencialmente los modelos físico matemáticos que definen el movimiento de una aeronave, de los sistemas propulsivos, y su interacción con el entorno. Su librería está programada en un lenguaje en C++, por lo que para entenderla va a hacer falta conocimientos de este mismo.

6.2 Fuerzas y momentos Aerodinámicos del modelo

Para poder establecer el comportamiento de una aeronave, lo principal son las fuerzas y los momentos que lo envuelven. Lo más importante en el desarrollo de la dinámica del avión que pone en relación a las fuerzas con los actuadores (nuestras variables controlables), son los coeficientes aerodinámicos.

La manera en la que JSBSim calcula dichos coeficientes, es utilizando datos reales de ensayos de cada perfil alar que están disponibles en la web de la NASA (http://ntrs.nasa.gov/) y los introducen en el archivo XML de nuestro avión (ver figura 18). No calcula los coeficientes uno a uno y luego calcula la fuerza total, sino que calcula por partes la fuerza que da cada coeficiente.

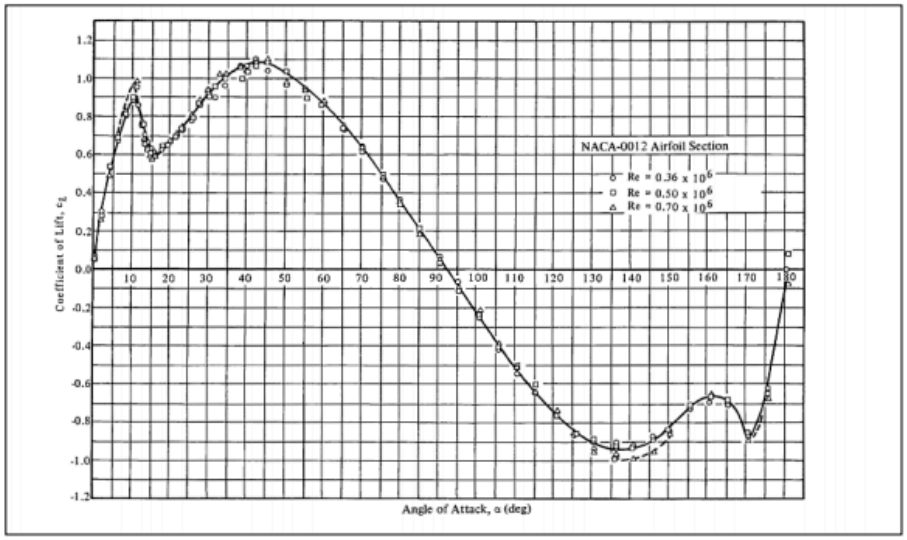


Figura 17: Gráfica de los valores de C_L según los ángulos de ataque α de un perfil NACA 0012

```
<function name="aero/coefficient/CDwbh">
    <description>Drag_due_to_alpha</description>
    <product>
        <property>aero/qbar-psf</property>
        <property>metrics/Sw-sqft</property>
        <property>aero/function/kCDge</property>
          <table>
              <independentVar lookup="row">aero/alpha-rad</independentVar>
              <independentVar lookup="column">fcs/flap-pos-deg</independentVar>
              <tableData>
                    0.0000  10.0000 20.0000 30.0000
                -0.0873   0.0041  0.0000  0.0005  0.0014
                -0.0698   0.0013  0.0004  0.0025  0.0041
                -0.0524   0.0001  0.0023  0.0059  0.0084
                -0.0349   0.0003  0.0057  0.0108  0.0141
                -0.0175   0.0020  0.0105  0.0172  0.0212
                0.0000    0.0052  0.0168  0.0251  0.0299
                0.0175    0.0099  0.0248  0.0346  0.0402
                0.0349    0.0162  0.0342  0.0457  0.0521
                0.0524    0.0240  0.0452  0.0583  0.0655
                0.0698    0.0334  0.0577  0.0724  0.0804
                0.0873    0.0442  0.0718  0.0881  0.0968
                0.1047    0.0566  0.0874  0.1053  0.1148
                0.1222    0.0706  0.1045  0.1240  0.1343
                0.1396    0.0860  0.1232  0.1442  0.1554
                0.1571    0.0962  0.1353  0.1573  0.1690
                0.1745    0.1069  0.1479  0.1708  0.1830
                0.1920    0.1180  0.1610  0.1849  0.1975
                0.2094    0.1298  0.1746  0.1995  0.2126
                0.2269    0.1424  0.1892  0.2151  0.2286
                0.2443    0.1565  0.2054  0.2323  0.2464
                0.2618    0.1727  0.2240  0.2521  0.2667
                0.2793    0.1782  0.2302  0.2587  0.2735
                0.2967    0.1716  0.2227  0.2507  0.2653
                0.3142    0.1618  0.2115  0.2388  0.2531
                0.3316    0.1475  0.1951  0.2214  0.2351
                0.3491    0.1097  0.1512  0.1744  0.1866
              </tableData>
          </table>
    </product>
</function>
```

Figura 18: Extracto del archivo XML de la aeronave Cessna 172 con la fuerza de resistencia Aerodinámica debida al ángulo α.

En la tabla de la figura 18 por ejemplo, el coeficiente depende tanto del ángulo de ataque como de la posición de los flaps. La primera fila indica la posición de los flaps y la primera columna los ángulos de ataque (en radianes). En el caso en el que el valor calculado por el programa no sea exactamente el que aparece en la columna, este buscará el valor del coeficiente interpolando entre los dos puntos más próximos.

La clase <product>, define que se va a multiplicar todos los elementos que estén dentro de este. En el ejemplo de la figura 18 la ecuación quedaría tal que:

$$CDwbh = q * Sw * kCDge * CD_{\alpha} \quad (x.x)$$

Donde:

- $q = \frac{1}{2} * \rho * V^2$ → Presión Dinámica
- Sw → Superficie alar
- kCDge → Cambio de resistencia debido al efecto suelo
- CD_{α} → Coeficiente extraído de la tabla

Estas maneras de calcular coeficientes, se aplica en todo el archivo "aeronave.xml", tanto para fuerzas como para momentos.

6.3 Modelos propulsivos.

JSBSim también proporciona varios modelos propulsivos para el cálculo del empuje del avión. Existen 5 tipos de motores dentro de este modelo:

- Motores de Pistón
- Rocket
- De Turbina
- Turboprop
- Motores eléctricos

Los parámetros de dichos motores vienen definidos en otro .xml dentro de la carpeta de la aeronave (c172p en nuestro caso).

En ese .xml, de la misma forma que las fuerzas Aerodinámicas, se ha introducido un coeficiente, en este caso el coeficiente del empuje (Thrust Coeficient), que es una función directa del *Advance Ratio.*

Así pues, este modelo calculará el empuje de la siguiente manera:

$$T = Ct * \rho * n^2 * D^4$$

Donde:

- T → Empuje (en N)
- Ct → Coeficiente de empuje
- ρ → Densidad del aire (kg/m^3)
- n → Revoluciones por segundo (s^{-1})
- D → Diámetro de las hélices (m)

6.4 Calculo de los coeficiente aerodinámicos del Cessna 172 según JSBSim

En la mecánica de vuelo convencional las expresiones de las fuerzas y momentos viene expresadas en función de unos coeficientes aerodinámicos tal que:

$$Drag = \frac{1}{2}\, Sw\, \rho\, V^2\, CD$$
$$Lift = \frac{1}{2}\, Sw\, \rho\, V^2\, CL$$
$$Side = \frac{1}{2}\, Sw\, \rho\, V^2\, CY$$
$$L = \frac{1}{2}\, Sw\, bw\, \rho\, V^2\, Cl$$
$$M = \frac{1}{2}\, Sw\, cw\, \rho\, V^2\, CM$$
$$N = \frac{1}{2}\, Sw\, bw\, \rho\, V^2\, CN$$

Es en estos coeficientes donde se establece la relación entre las fuerzas y los actuadores (excepto la palanca de gases).

En la Cessna existen 4 actuadores, las cuales serán nuestras variables de control o de entrada:

- $\delta a \rightarrow$ alerones ("aileron"), controlarán sobre todo el alabeo del avión.
- $\delta e \rightarrow$ elevador o timón de profundidad ("elevator"), controlará sobre todo el cabeceo.
- $\delta r \rightarrow$ timón de cola ("rudder"), controlará sobre todo la guiñada del avión.

Como hemos explicado anteriormente, el modelo JSBSim no usa coeficientes fijos, sino que usa valores experimentales de datos sacados de la NASA según el perfil alar de la aeronave, para algunos coeficientes. Para aquellos valores que vienen de datos experimentales de tablas, hemos hecho uso de aproximaciones por mínimos cuadrados para aproximarlo a un polinomio, ya que necesitamos dichas funciones en función de los parámetros tanto controlables como de estado.

Coeficientes aerodinámicos del Drag:

Nomenclatura	Valor	Descripción
CD_0	0.27	Coeficiente de Resistencia parásita
CD_{flaps}	0	Coeficiente de Resistencia debida a los flaps (hipótesis 8 no hay flaps)
CD_α	$-153\alpha^5 + 64.14\alpha^4 - 9.82\alpha^3 + 2.141\alpha^2 + 0.304\alpha + 0.0058$	Coeficiente de Resistencia debida al ángulo de ataque. Aproximación por mínimos cuadrados
$CD_{\delta e}$	0	Coeficiente de Resistencia debida a la deflexión del elevador
CD_β	0.17β	Coeficiente de Resistencia debida al ángulo de deslizamiento
CD	$CD_0 + CD_{flaps} + CD_\alpha + CD_{\delta e} + CD_\beta$	Coeficiente de Resistencia total.

Tabla 4: Coeficientes de la resistencia aerodinámica

Coeficientes aerodinámicos del Lift:

Nomenclatura	Valor	Descripción
CL_α	$772.76\alpha^5 + 440.72\alpha^4 - 85.622\alpha^3 + 0.5757\alpha^2 + 6.2077\alpha + 0.2476$	Coeficiente de sustentación debida al ángulo de ataque. Aproximación por mínimos cuadrados
CL_{flaps}	0	Coeficiente de Sustentación debida a los flaps (hipótesis 8 no hay flaps)
$CL_{\delta e}$	$0.43\delta e$	Coeficiente de Sustentación debida a la deflexión del elevador
$CL_{\dot\alpha}$	$1.7\ \dot\alpha\ \frac{cw}{2V}$	Coeficiente de Sustentación debido a la velocidad angular del ángulo de ataque
CL_q	$3.9\ q\ \frac{cw}{2V}$	Coeficiente de sustentación debido a la velocidad angular q
CL	$CL_\alpha + CL_{flaps} + CL_{\delta e} + CL_{\dot\alpha} + CL_q$	Coeficiente de sustentación total.

Tabla 5: Coeficientes de la sustentación aerodinámica

Coeficientes aerodinámicos del Side:

Nomenclatura	Valor	Descripción
CY_p	$-0.7447\alpha - 0.075$	Coeficiente de Side debido al roll (velocidad angular en p)
$CY_{\delta r}$	$0.1870\ \delta r$	Coeficiente de Side debido al timón de cola.
CY_r	$0.5638\alpha + 0.214$	Coeficiente de Side debido al Yaw (velocidad angular en r)
$CY_{\delta a}$	0	Coeficiente de Side debido a los alerones.
CY_β	-0.3926β	Coeficiente de Side debido al ángulo de deslizamiento
CY	$CY_p + CY_{\delta r} + CY_r + CY_{\delta a} + CY_\beta$	Coeficiente de Side total.

Tabla 6: Coeficientes aerodinámicos del Side

Coeficientes aerodinámicos del Roll:

Nomenclatura	Valor	Descripción
Cl_p	$-0.4840\ p\ \frac{cw}{2\,V}$	Coeficiente de roll debido a la velocidad angular en p
$Cl_{\delta r}$	$0.0147\ \ \delta r$	Coeficiente de Roll debido al timón de cola.
Cl_r	$(1.1394\alpha + 0.0798)\ r\ \frac{cw}{2\,V}$	Coeficiente del Roll debido a la velocidad angular en r
$Cl_{\delta a}$	$0.229\ \delta a$	Coeficiente del Roll debido a los alerones.
Cl_β	-0.0923 β	Coeficiente de Roll debido al ángulo de deslizamiento
Cl	$Cl_p + Cl_{\delta r} + Cl_r + Cl_{\delta a} + Cl_\beta$	Coeficiente del Roll total.

Tabla 7: Coeficientes aerodinámicos del Roll

Coeficientes aerodinámicos del Pitch

Nomenclatura	Valor	Descripción
CM_α	$-1.8 \sin\alpha$	Coeficiente del Pitch debido al ángulo de ataque
CM_0	0.1	Coeficiente del Pitch parásito
$CM_{\delta e}$	-1.122 δe	Coeficiente del Pitch debido a la deflexión del elevador
$CM_{\dot\alpha}$	-7.27 $\dot\alpha\ \frac{cw}{2\,V}$	Coeficiente del Pitch debido a la velocidad angular del ángulo de ataque
CM_q	-12.4 $q\ \frac{cw}{2\,V}$	Coeficiente del Pitch debido a la velocidad angular q
CM	$CM_\alpha + CM_0 + CM_{\delta e} + CM_{\dot\alpha} + CM_q$	Coeficiente del Pitch total.

Tabla 8: Coeficientes aerodinámicos del Pitch

Coeficientes aerodinámicos del Yaw:

Nomenclatura	Valor	Descripción
CN_p	$-0.0278\ p\ \frac{cw}{2\ V}$	Coeficiente del Yaw debido a la velocidad angular en p
$CN_{\delta r}$	-0.043 δr	Coeficiente del Yaw debido al timón de cola.
CN_r	$-0.0937\ r\ \frac{cw}{2\ V}$	Coeficiente del Yaw debido a la velocidad angular en r
$CN_{\delta a}$	-0.0053 δa	Coeficiente del Yaw debido a los alerones.
CN_β	0.0587 β	Coeficiente del Yaw debido al ángulo de deslizamiento
CN	$CN_p + CN_{\delta r} + CN_r + CN_{\delta a} + CN_\beta$	Coeficiente del Yaw total.

Tabla 9: Coeficientes aerodinámicos del Yaw

7. Implementación de una plataforma en Matlab© para el estudio experimental de los actuadores.

7.1 Introducción.

Ya hemos realizado el análisis teórico del movimiento del avión y la relación entre las ecuaciones diferenciales y los actuadores que iremos a controlar. Sin embargo la motivación principal de este proyecto, es implementar una plataforma para diseñar controladores de una manera experimental. En este apartado explicaremos el desarrollo del estudio experimental de la respuesta de los actuadores en los ángulos de Euler, ya que son la piedra angular de los giros y los más necesarios en un piloto automático.

Lo que queremos buscar en todo esto son las funciones de transferencia que rigen el comportamiento de dicho ángulos.

- Nuestras variables de control serán: $\{\delta a, \delta e, \delta r\}$
- Las variables de estado serán: $\{\phi, \theta, \psi\}$

No nos concentraremos en las velocidades en el cuerpo (también necesarias), ni tampoco controlaremos la palanca de gases del motor δp.

7.2 Condiciones Iniciales.

Para poder conseguir un buen estudio experimental, es necesario que nuestra aeronave se encuentre en equilibrio. Podríamos calcular este punto de funcionamiento con las ecuaciones que habíamos desarrollado en el apartado anterior. Sin embargo hemos visto que en los aviones de hélice, se producen una

serie de efectos que no podemos calcular. Es por culpa de estos efectos que no podremos establecer un punto de equilibrio o de trimado para nuestros actuadores. Por eso mismo hemos tenido que hacerlo de manera experimental nivelando los timones hasta que conseguíamos una respuesta estable. Los parámetros de dicha respuesta se pueden ver en la tabla 11.

7.3 Recogida de datos con la herramienta Matlab.

Como hemos visto anteriormente cuando hemos estudiado las conexiones con el simulador de vuelo, la conexión más fiable para la recogida de datos va a ser la conexión vía UDP.

Así pues crearemos una interfaz simulink para recoger y enviar la información que queramos.

Enviaremos:

Variable	Workspace	Condición inicial
δa	Ua	0.00465
δe	Ue	0.2
δr	Ur	0.00782
δp	Up	0.8

Tabla 10: Variables transmitidas al simulador

Recibiremos :

Variable	Workspace	Condición inicial
ϕ	rollident	0º
θ	pitchident	0.85º
ψ	yawident	24º
p	No guardaremos en el workspace	0º/s
q	No guardaremos en el workspace	0º/s
r	No guardaremos en el workspace	0º/s
α	No guardaremos en el workspace	1º
β	No guardaremos en el workspace	0º
V (airSpeed)	Vident	93.89 kt
V_{ground}	No guardaremos en el workspace	m/s
Z (altitud)	altident	6570 ft
Latitud	No guardaremos en el workspace	37.44 º
Longitud	No guardaremos en el workspace	-121.86 º

Tabla 11: Variables recibidas del simulador de vuelo

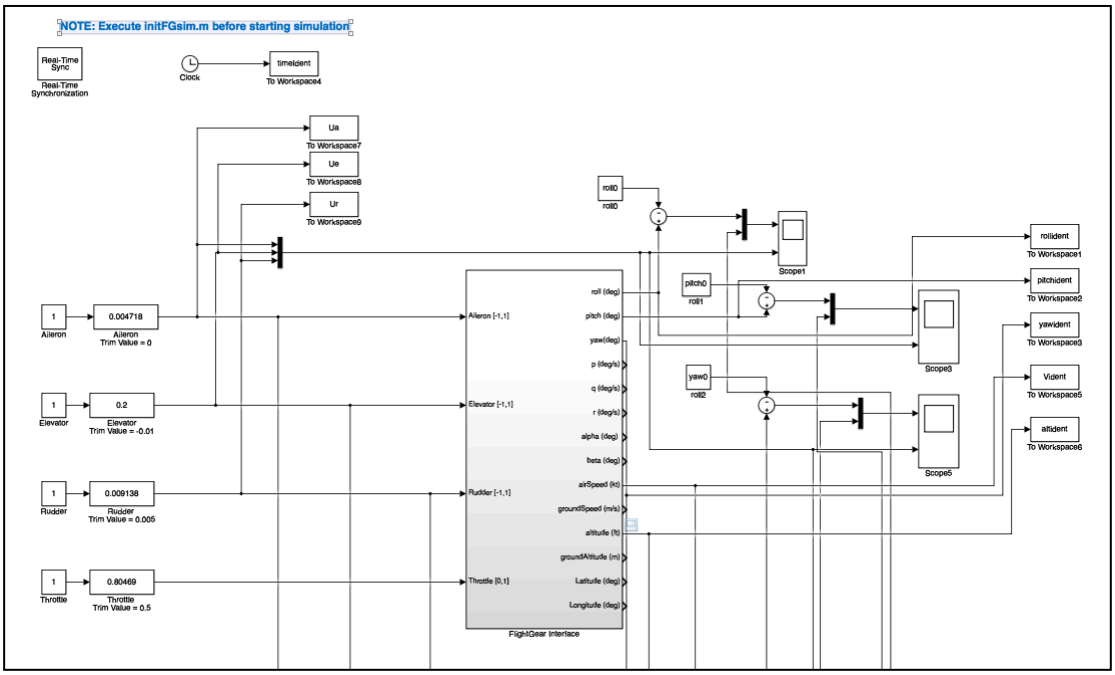


Figura 19: Vista general del sistema de emisión y recepción de variables mediante simulink.

7.4 Respuesta experimental de la entrada en escalón de los alerones δa

Vamos a visualizar de una manera practica desde el Workspace las salidas que tendría una entrada escalón en el actuador de los alerones.

La idea es establecer a ojo el tipo de función de transferencia que se trataría de analizar para cada respuesta.

- Respuesta de ϕ ($figura$ 20): Parece que se trata de una función de transferencia de primer orden, sin inercia y sin retraso, hay pequeñas oscilaciones cuando se estabiliza, pero muy leves, no las tendremos en cuenta.
- Respuesta de θ ($figura$ 21): En el cabeceo se producen muchas oscilaciones por que depende de la velocidad. Es decir cuando la aeronave tiene un ángulo de cabeceo positivo la velocidad disminuirá, por esto mismo tenderá a bajar el morro, por lo que la velocidad aumentará y tenderá a subir. Buscaremos aproximar esta respuesta a un primer orden sin oscilaciones porque intentaremos controlar la velocidad.
- Respuesta de ψ ($figura$ 22): En la guiñada la aproximaremos a una función de transferencia de primer orden con integrador. La respuesta habrá que manipularla ya que el sistema nos devuelve el valor de la guiña de entre 0 a 360, por ello aparecen esos picos, pero en teoría debería seguir subiendo.

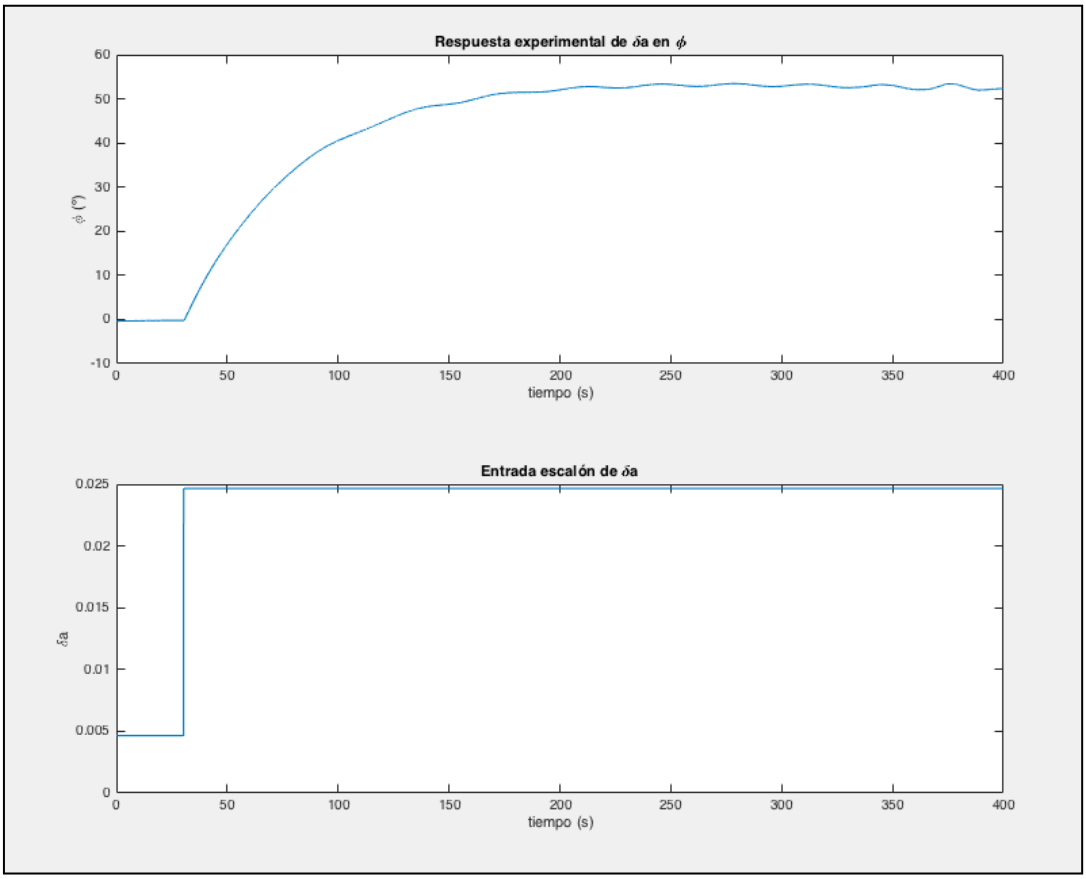


Figura 20: Gráfica de la respuesta experimental del ángulo de alabeo de una entrada en escalón de los alerones.

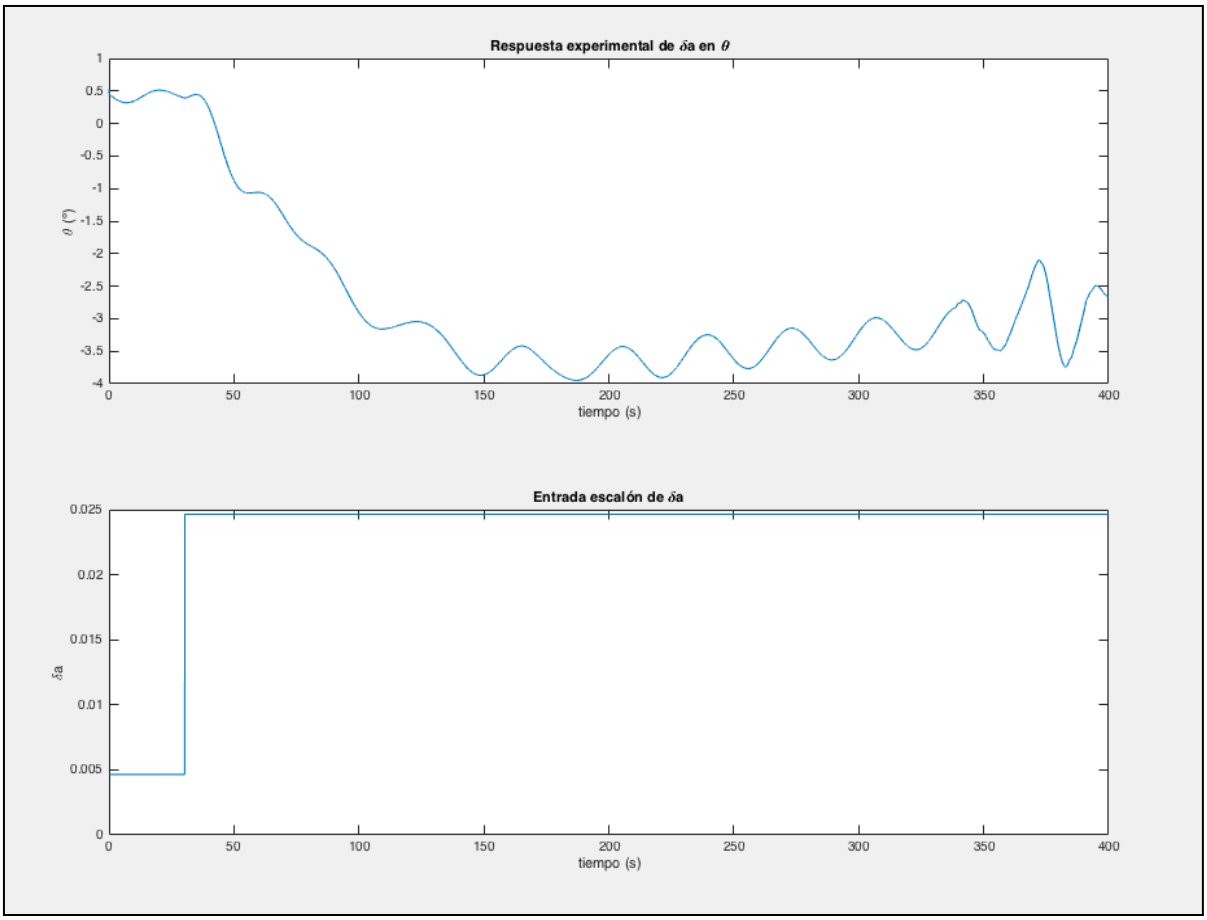


Figura 21: Gráfica de la respuesta experimental del ángulo de cabeceo de una entrada en escalón de los alerones.

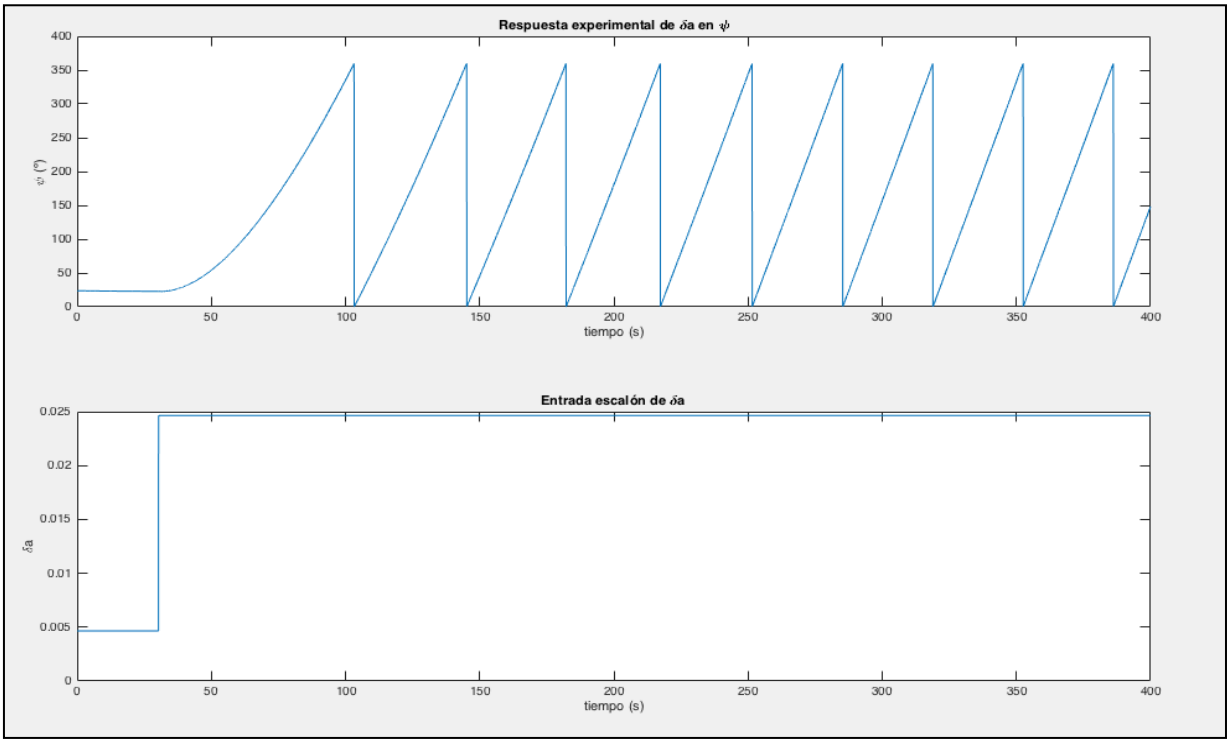


Figura 22: Gráfica de la respuesta experimental del ángulo de guiñada de una entrada en escalón de los alerones.

7.5 Respuesta experimental de la entrada en escalón de los elevadores δe

Ahora vamos a ver que respuestas dan de los mismos ángulos los alerones.

- Respuesta de ϕ $(figura\ 23)$: En esta caso vemos que parece un sistema de primer orden, pero sin embargo con el largo tiempo (a los 300 segundos), vemos que va perdiendo (por los efectos de las hélices explicados anteriormente). Como no vamos a provocar giros tan largos, lo aproximaremos al primer orden.
- Respuesta de θ $(figura\ 24)$: Aquí lo aproximaremos a un sistema de segundo orden dado que parece que se estabiliza al final aunque aun existan unas pequeñas oscilaciones, pero bastante largar en el tiempo.
- Respuesta de ψ $(figura\ 25)$: En este caso, en la figura x.x, ya hemos convertido el sistema de valores de 0 a 360 a $[-\infty, +\infty]$ para poder analizar sin problemas. Vemos que podemos aproximarlo a un sistema de primer orden con integrador.

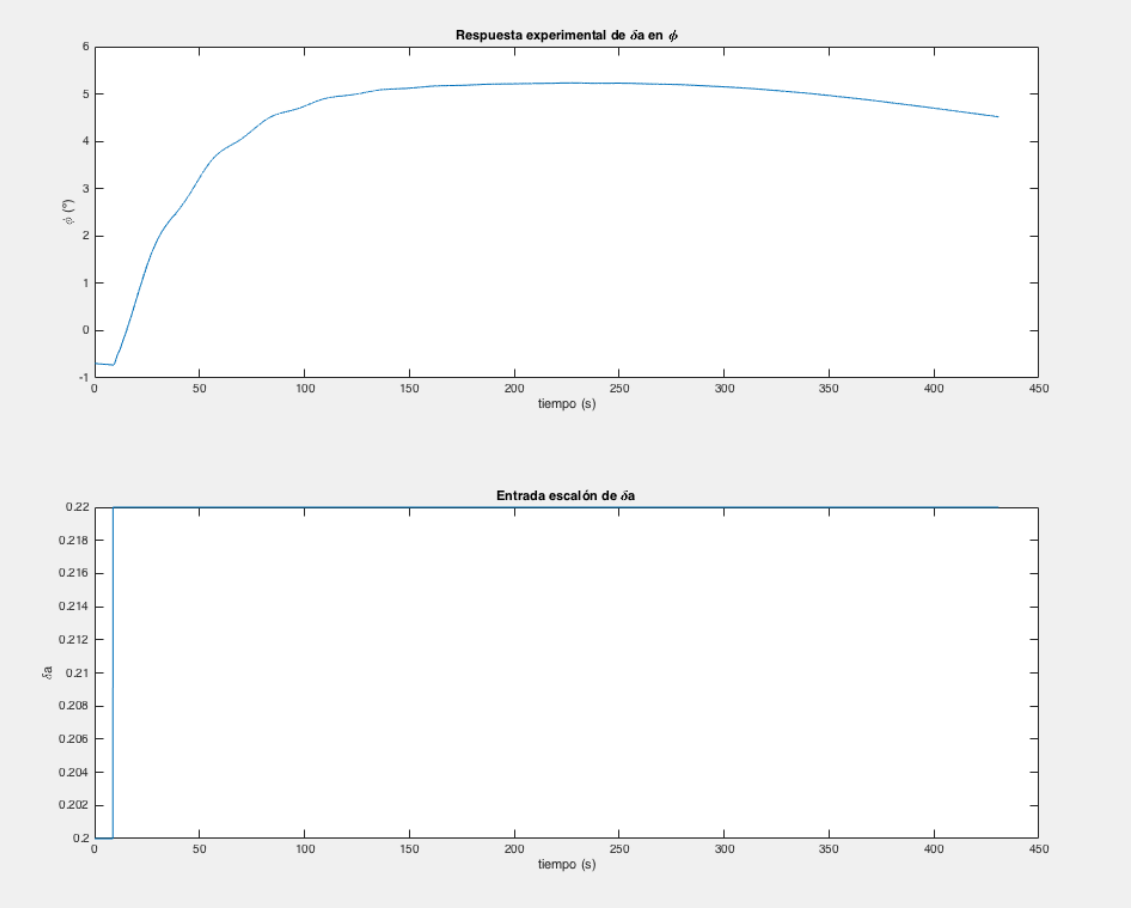


Figura 23: Gráfica de la respuesta experimental del ángulo de alabeo de una entrada en escalón de los elevadores.

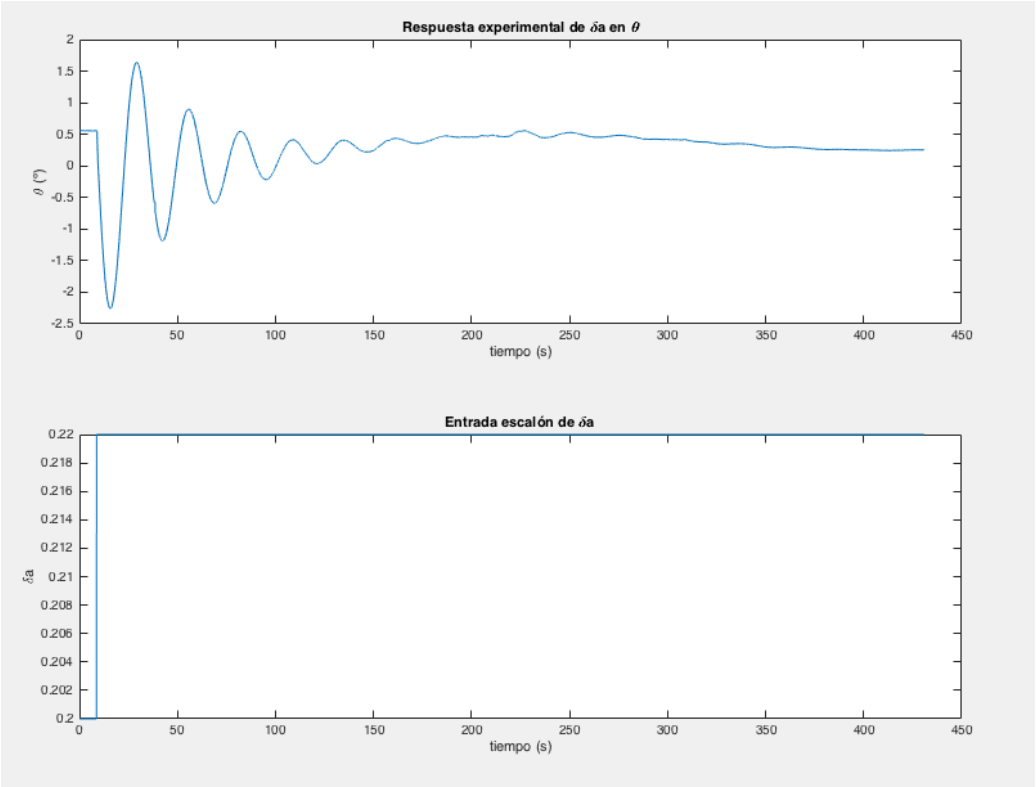


Figura 24: Gráfica de la respuesta experimental del ángulo de cabeceo de una entrada en escalón de los elevadores.

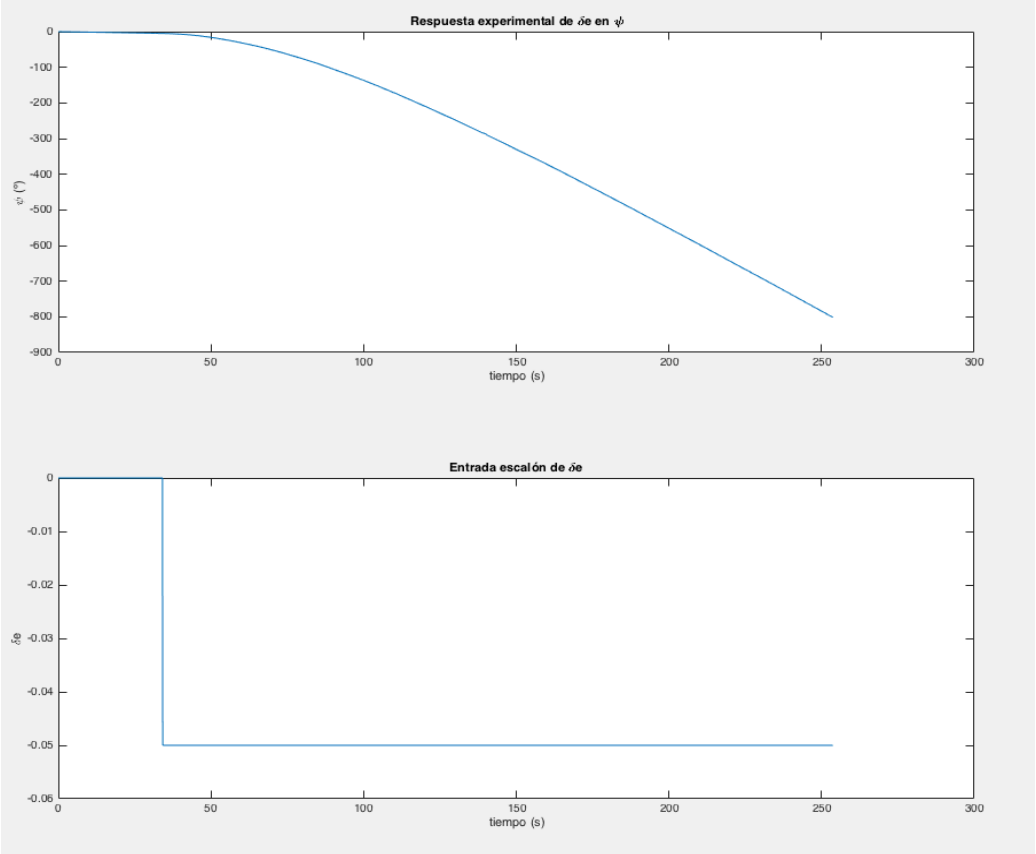


Figura 25: Gráfica de la respuesta experimental del ángulo de guiñada de una entrada en escalón de los elevadores.

7.6 Respuesta experimental de la entrada en escalón del timón de cola δr

Antes de nada hay que observar que este muestreo se ha tomado en un intervalo de tiempo muy grande (40 minutos)

- Respuesta de ϕ ($figura$ 26): Aquí sigue pareciendo una función de transferencia de primer orden aunque pasa como en el caso anterior pero con pequeñas oscilaciones. Obviaremos estas oscilaciones y lo dejaremos en una aproximación de primer orden.
- Respuesta de θ ($figura$ 27): En este caso presenciamos muchísimas oscilaciones y parece imposible de modelar, sin embargo si nos fijamos en la escala del eje, en realidad el ángulo está variando cerca de 0.5 grados en 40 minutos, por lo que parece que el efecto del pitch en este caso es muy leve. Aun así lo aproximaremos a una función de segundo orden para este leve cambio.
- Respuesta de ψ ($figura$ 28): Esta parece la respuesta mas estable del timón de cola. Aproximaremos a una función de primer orden con integrador. Sobra decir que falta ajustar los valores al intervale de $[-\infty, +\infty]$.

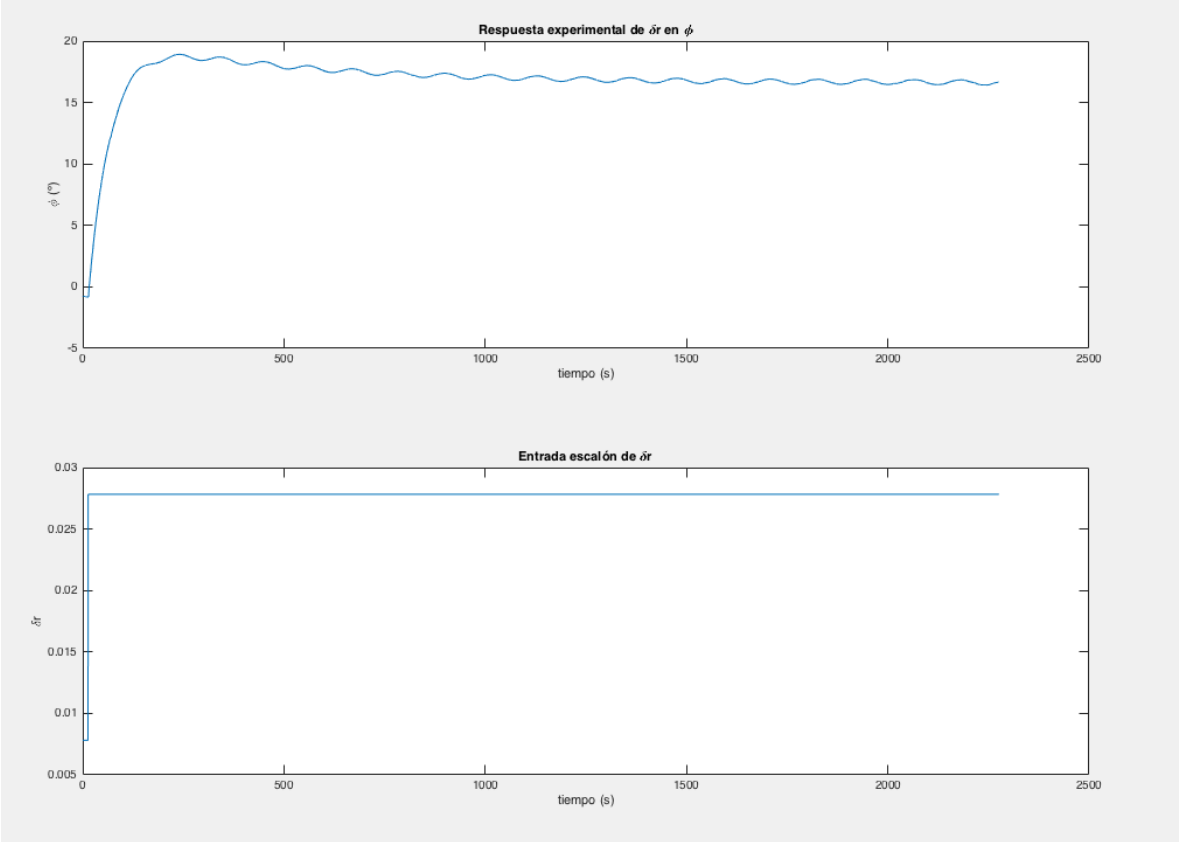


Figura 26: Gráfica de la respuesta experimental del ángulo de alabeo de una entrada en escalón del timón de cola.

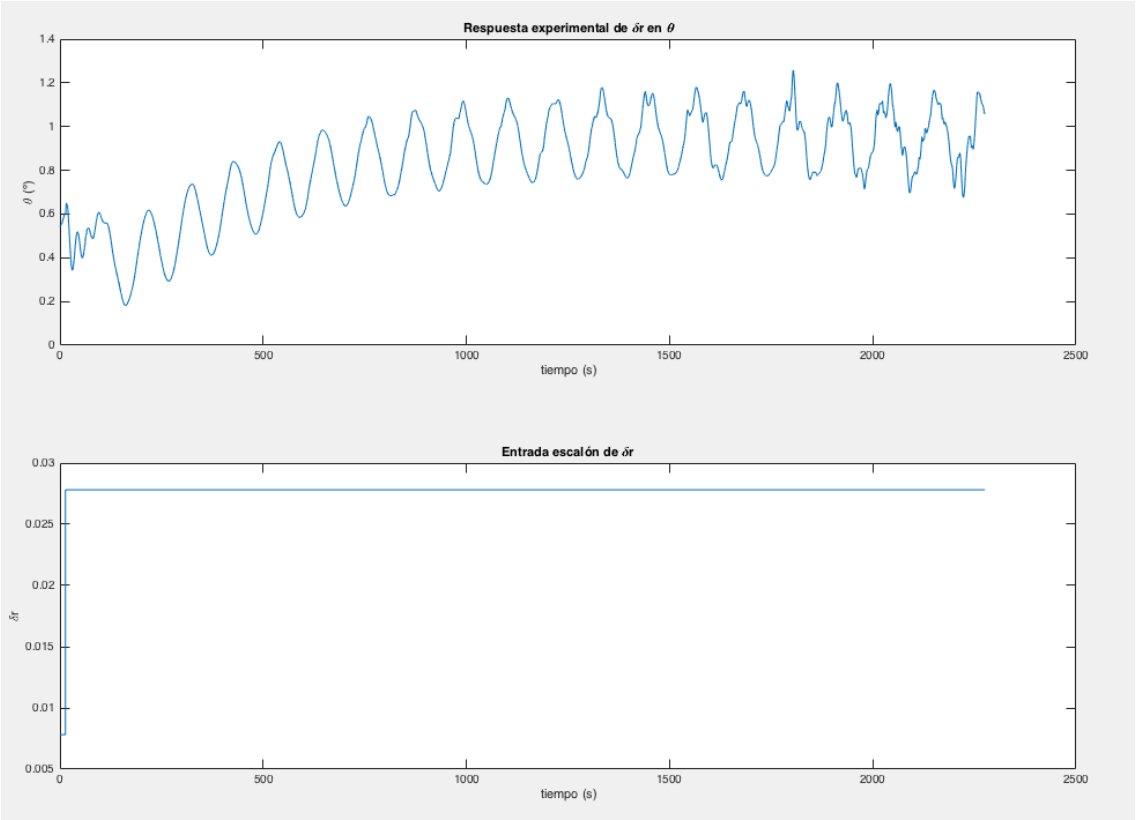


Figura 27: Gráfica de la respuesta experimental del ángulo de cabeceo de una entrada en escalón del timón de cola.

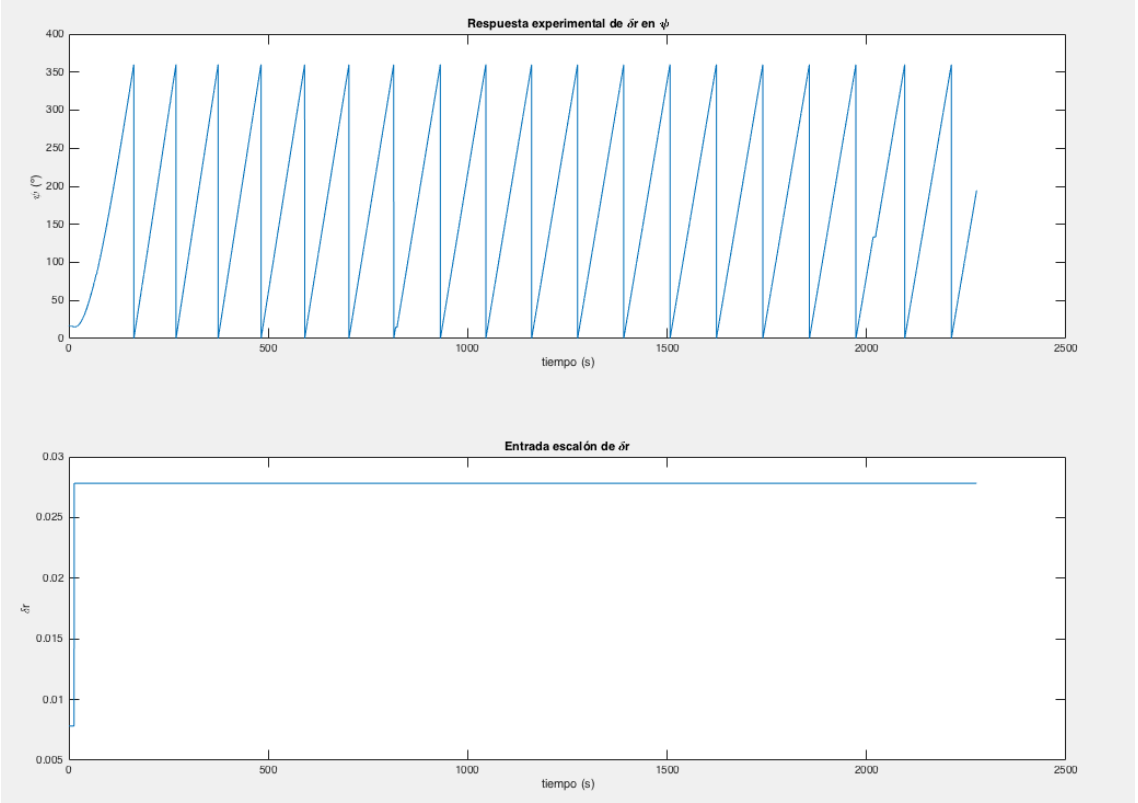


Figura 28: Gráfica de la respuesta experimental del ángulo de guiñada de una entrada en escalón del timón de cola.

7.7 Modelado del sistema.

Ahora que tenemos las respuestas a nuestros actuadores, y tenemos una idea de que aproximación vamos a realizar (1er orden, 2do, con integrador etc.), llega el momento de encontrar los valores de las aproximaciones para establecer funciones de transferencia que responda de una manera similar a la realidad mediante entradas escalón. Para encontrar dichos modelos hemos hecho uso de dos técnicas, una proporcionada por la herramienta Matlab©, y otra implementada por nosotros con la ayuda de algunas funciones facilitadas por el departamento de control de la UPV.

System Identification Toolbox™:

Este Toolbox nos proporciona una serie de funciones para la herramienta Matlab© y de bloques de Simulink© cuya finalidad es construir modelos matemáticos de sistemas dinámicos a partir de datos experimentales de entradas y salidas del sistema.

Las técnicas con las que funciona este sistema son de máxima verosimilitud, es decir usando algoritmos de predicción de mínimos errores (PEM prediction-error minimization).

Se puede cargar la función "ident", la cual te devuelve una interfaz con la que es muy fácil e intuitiva tanto cargar las funciones reales como modelarlas.

1. *Ventajas:*

- Su interfaz (figura 29), nos permite de una manera mucho más dinámica la obtención del modelado.
- Para funciones cuya función de errores no es muy compleja viene perfecto ya que calcula de una manera bastante rápida los modelos.
- La rapidez de cálculo

2. *Desventajas:*

- A causa del dinamismo y la rapidez del sistema muchas veces la minimización de los errores puede entrar en mínimos locales cuando la función de errores es más compleja y nos devuelve modelos que no se asemejan en nada con el sistema real.
- Se tienen muchos problemas para acotar los parámetros a calcular para el modelado.

Así pues este sistema lo estableceremos como una primera aproximación para observar si el tipo de modelado que vamos a querer usar para ese sistema va a dar buenos frutos o no, pero necesitaremos de otro método que elimine estas desventajas.

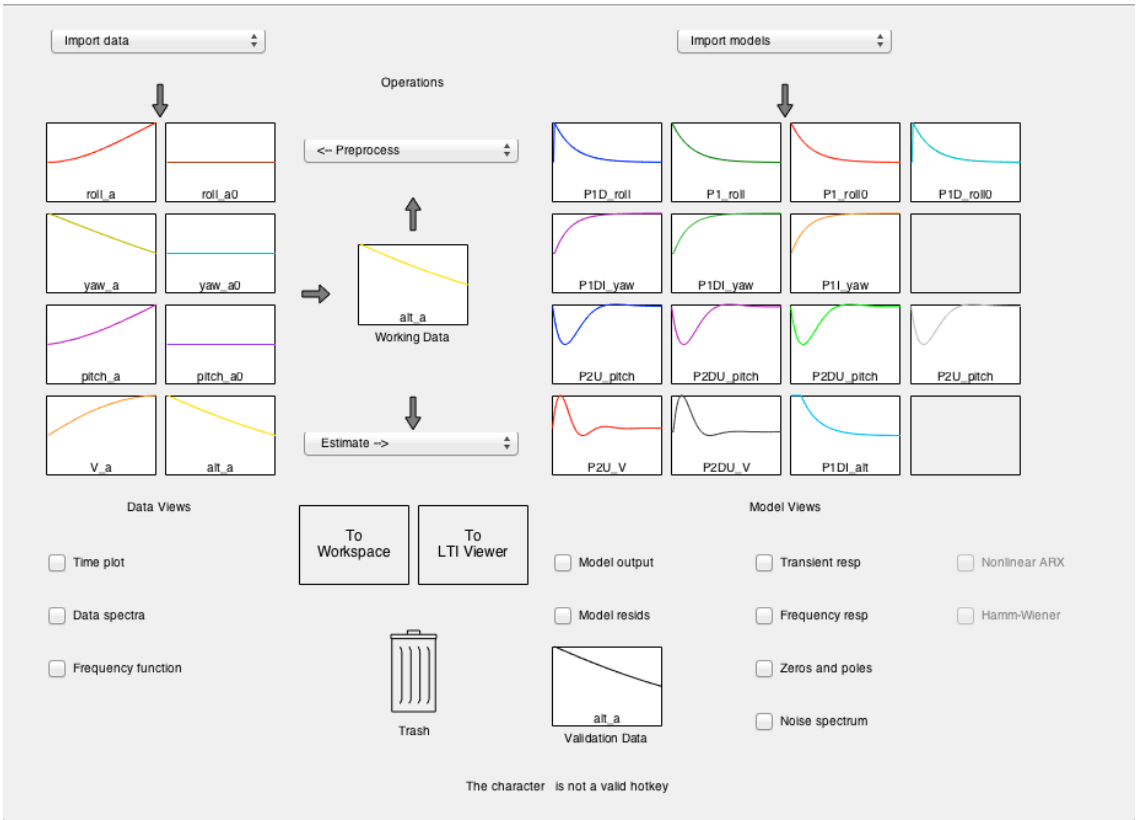


Figura 29: Interfaz de la herramienta ident para las respuestas de los alerones

Algoritmos genéticos:

De la herramienta anterior surje la gran desventajas de entrar en mínimos locales. Para evitar este error, se ha buscado implementar el mismo sistema de busqueda de error minímo, pero usando algoritmos genéticos.

Los algoritmos genéticos son una técnica de optimización basada en la analogía con la teoría de la evolución de las especias donde una población evoluciona para adaptarse al entorno. En esta analogía se establecen un entorno y unos individuos donde:

$$Individuo \rightarrow Posible\ solución$$
$$Entorno \rightarrow Función\ objetivo$$

Para comprenderlo mejor, vamos a por un ejemplo que se ha aplicado al modelado del ángulo ϕ en respuesta a los alerones (figura 20). Como hemos dicho en el apartado anterior, esta respuesta la podemos aproximar a una funcion de transferencia de primer orden que tendrá la forma de: $G(s) = \frac{K}{\tau * s + 1}$
Así pues en este caso, cada individuo tendrá dos parámetros que serán f(K, τ), y los valores que les demos a estos parámetros serán posibles soluciones.
Por otro lado nuestra función objetivo será la integral del valor absoluto de la diferencia entre los valores del ángulo ϕ experimental y los valores de nuestra función G(s) para las entradas de nuestro escalón experimental, en el intervalo de tiempo de los datos experimentales. Nosotros haremos uso de Simulink© para sacar estos valores (Figura 30).

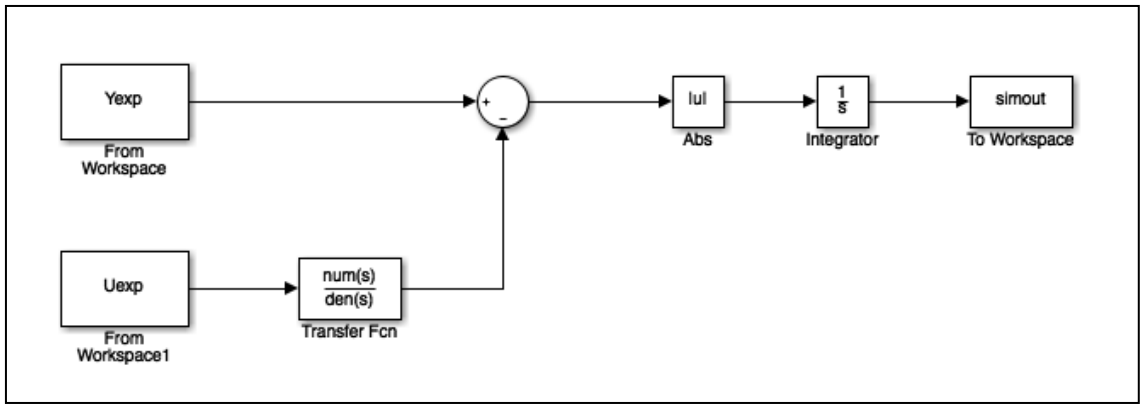


Figura 30: Diagrama de bloques de la función objetivo del algoritmo genético

Se pueden usar otras funciones objetivo para minimizar errores, pero con esta se intenta minimizar a lo largo de toda la función al recoger el aerea que existe entre las dos funciones.

Con esto claro, en primer lugar se establece una población inicial con x individuos aleatoriamente repartidos entre unas condiciones de contorno. Por ejemplo en nuestro caso repartiremos 30 individuos para un contorno de $K \in [-2000; 2000]$ y $\tau \in [0.1\ 50]$ (estos valores los pondremos después de haber realizado una primera aproximación con la herramienta de Matlab© anterior). Estos individuos estarán uniformamente repartidos por toda la función objetivo, y acto seguido se buscará obtener una nueva generación mediante tres tipos de operadores genéticos:

- **Por selección**: En este operador genético se van a elegir que individuos seguirán formando parte de la generación siguiente. Estos individuos según el resultado que tienen de la función objetivo, se les dará mayor o menor probabilidad de sobrevivir, por ejemplo en nuestro ejemplo si un individuo con cromosomas (K, τ)=(200,2) tiene como resultado de la función objetivo un valor menor que un individuo con cromosomas (K, τ)=(100,30), entonces el primer individuo tendrá mayor probabilidad de sobrevivir . Luego tiraremos lo que llaman de un punto de vista metafórico "una ruleta". A esta "ruleta" evitaremos poner solo un puntero, ya que aunque sea poca, existe una probabilidad de que solo salga el individuo que no queremos que sobreviva. Así pues usaremos una ruleta "Stochastic Universal Sampling" (Figura 31). Con esta ruleta elegiremos de manera aleatoria los individuos que seguirán la generación siguiente.
- **Por cruce:** A partir de los individuos que han sobrevivido, se usa sus cromosomas para crear unos individuos "hijos" haciendo una recombinación lineal de los ambos (figura 32). Dichos nuevos individuos tendrán tambien una probabilidad de supervivencia para la siguiente iteración. De dos individuos padre, se creará dos individuos hijos.
- **Por mutación:** Aleatoriamente se coge alguno o algunos de los genes de algún cromosoma de algún individuo y se cambia. Esto sirve para evitar entrar en mínimos locales (Figura 33).

Pondremos según los criterios que queramos usar una probabilidad de cruce y una probabilidad de mutación para establecer cada cuantos individuos sufren estos operadores genéticos.

Con estos algoritmos conseguimos encontrar una solución valida en un número de iteraciones mucho más reducidas. Al final la iteración parará según la condición que establezcamos como de finalización (figura 34), hasta entonces el proceso anterior seguirá repitiéndose. Lo que conseguimos con esto es que al final haya un gran cumulo de individuos en un mismo espacio muy reducido y que al final todos tengan unos cromosomas muy parecidos, esa será nuestra solución para obtener los valores mínimos de nuestra función objetivo.

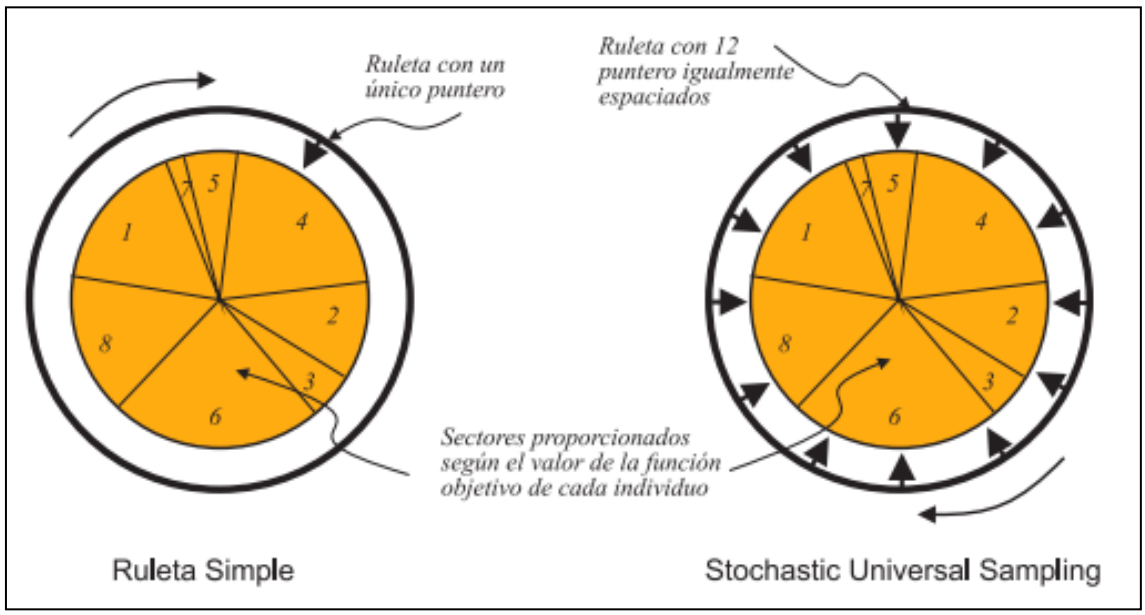


Figura 31: Esquema del operador genético de selección de individuos

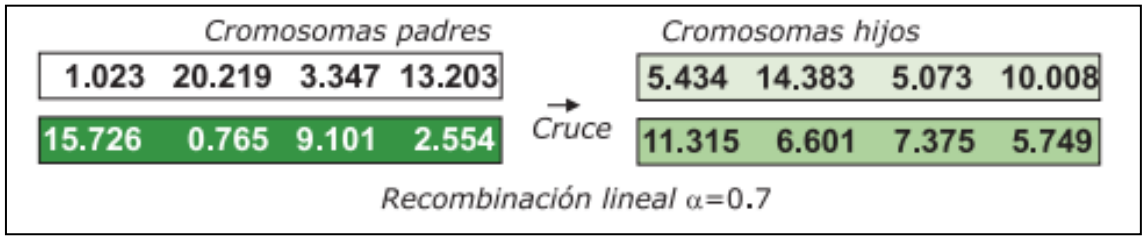


Figura 32: Ejemplo de cruce para individuos con 4 cromosomas (4 variables)

Figura 33: Ejemplo de mutación para inviduos con 4 cromosomas (4 variables)

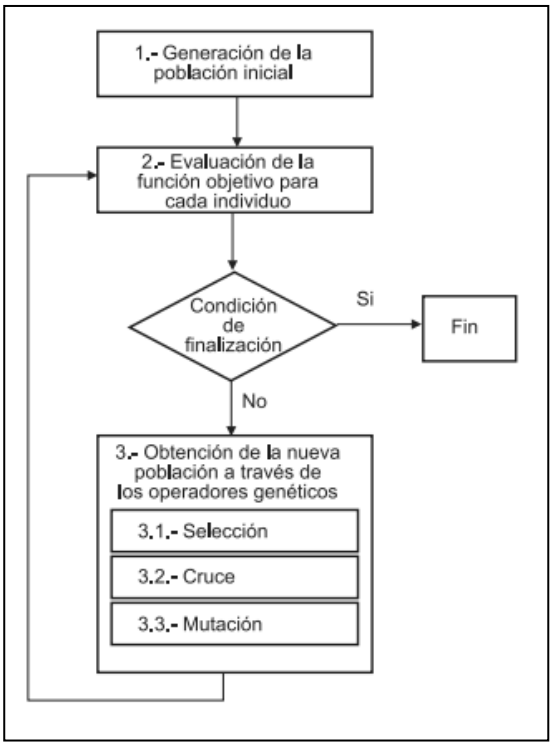


Figura 34: Esquema general de un algoritmo genético

7.8 Modelado de la respuesta de los alerones en los ángulos de Euler

Hemos seguido los modelos que habíamos previsto en el apartado anterior, y haciendo uso de los algoritmos genéticos hemos hallado las siguientes funciones de transferencia para cada ángulo:

Ángulo	Función de transferencia	Observaciones
ϕ	$\frac{2701}{49.62\ s + 1}$	Figura 35: 96.83% de semejanza con los valores reales.
θ	$\frac{-303e4\ s - 121.9}{1.153e5\ s^3 + 1.087e04\ s^2 + 183.3\ s + 1}$	Figura 36: 74.75% de semejanza con los valores reales. Esto es así porque hemos aproximado la función eliminando las oscilaciones.
ψ	$\frac{544.5}{59.61\ s^2 + s}$	Figura 37: 99.7% de semejanza con los valores reales.

Tabla 12: Funciones de transferencia de los alerones

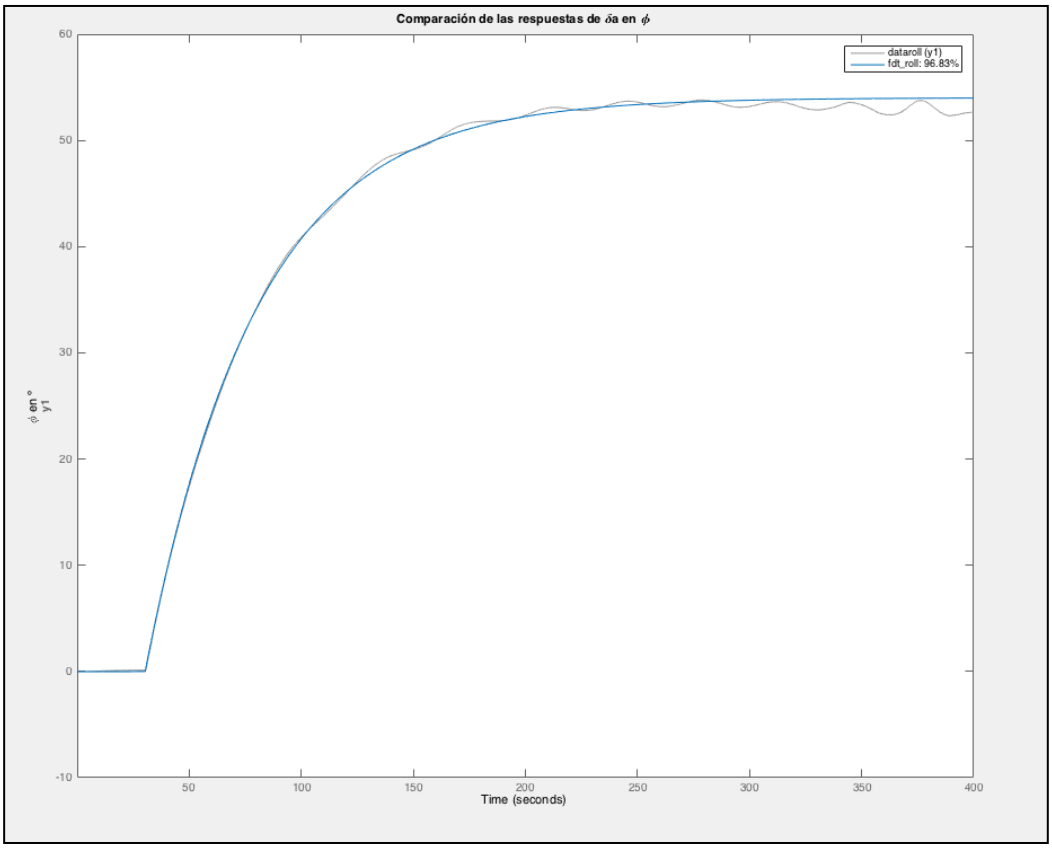


Figura 35: Comparación entre los valores reales y la respuesta de la función de transferencia en ϕ de los alerones

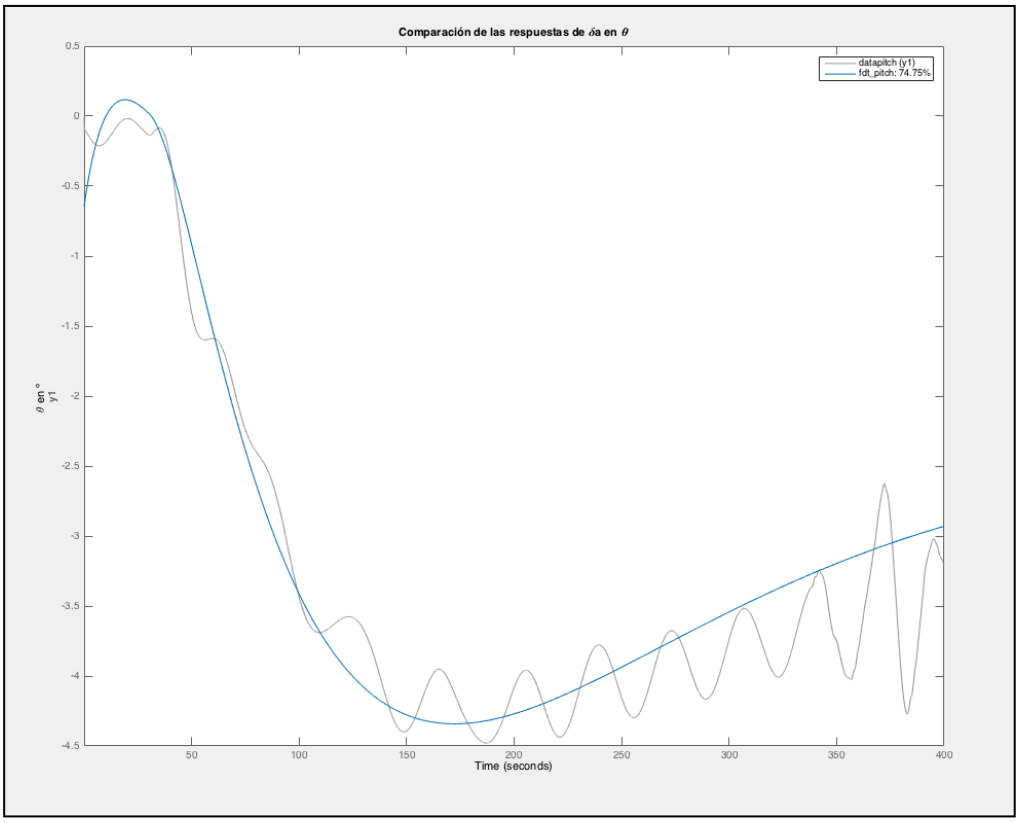


Figura 36: Comparación entre los valores reales y la respuesta de la función de transferencia en θ de los alerones

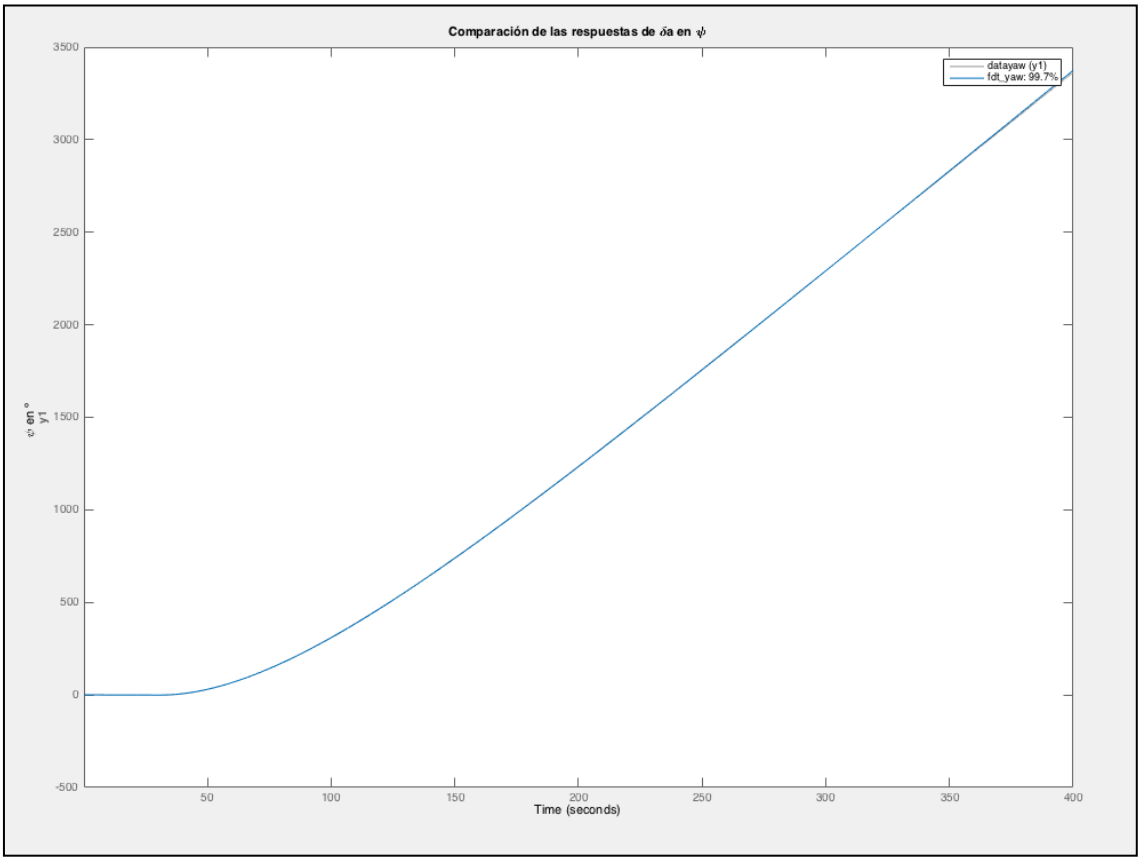


Figura 37: Comparación entre los valores reales y la respuesta de la función de transferencia en ψ de los alerones

7.9 Modelado de la respuesta del elevador en los ángulos de Euler

Ángulo	Función de transferencia	Observaciones
ϕ	$\frac{296.4}{36.95\ s + 1}$	Figura 38: 82.58% de semejanza con los valores reales. Esto es debido por que al final pierde.
θ	$\frac{-608.4\ s - 13.56}{18.09\ s^2 + 0.9274s + 1}$	Figura 39: 64.67% de semejanza con los valores reales. Es la mejor aproximación que se podía conseguir dada su altas oscilaciones, aunque en realidad los valores varían de 1.5 a -2.5 grados.
ψ	$\frac{94.27}{50.95s^2 + s}$	Figura 40: 99.29% de semejanza con los valores reales.

Tabla 13: Funciones de transferencia del elevador

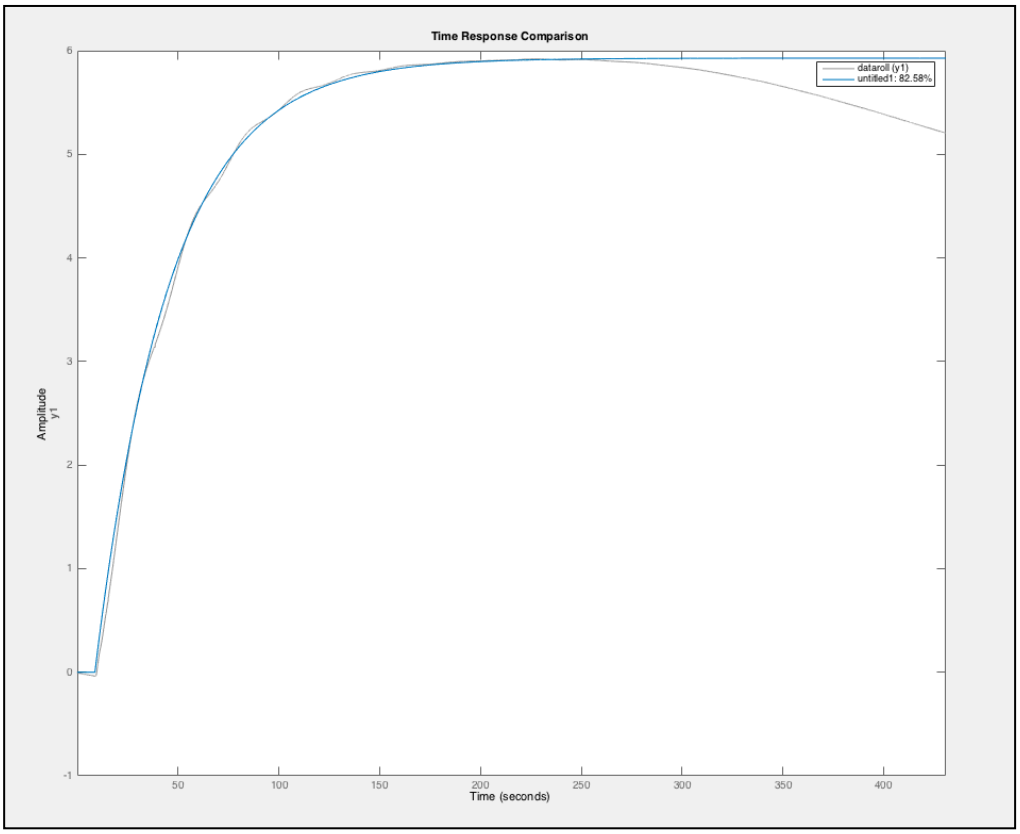


Figura 38: Comparación entre los valores reales y la respuesta de la función de transferencia en ϕ de los elevadores

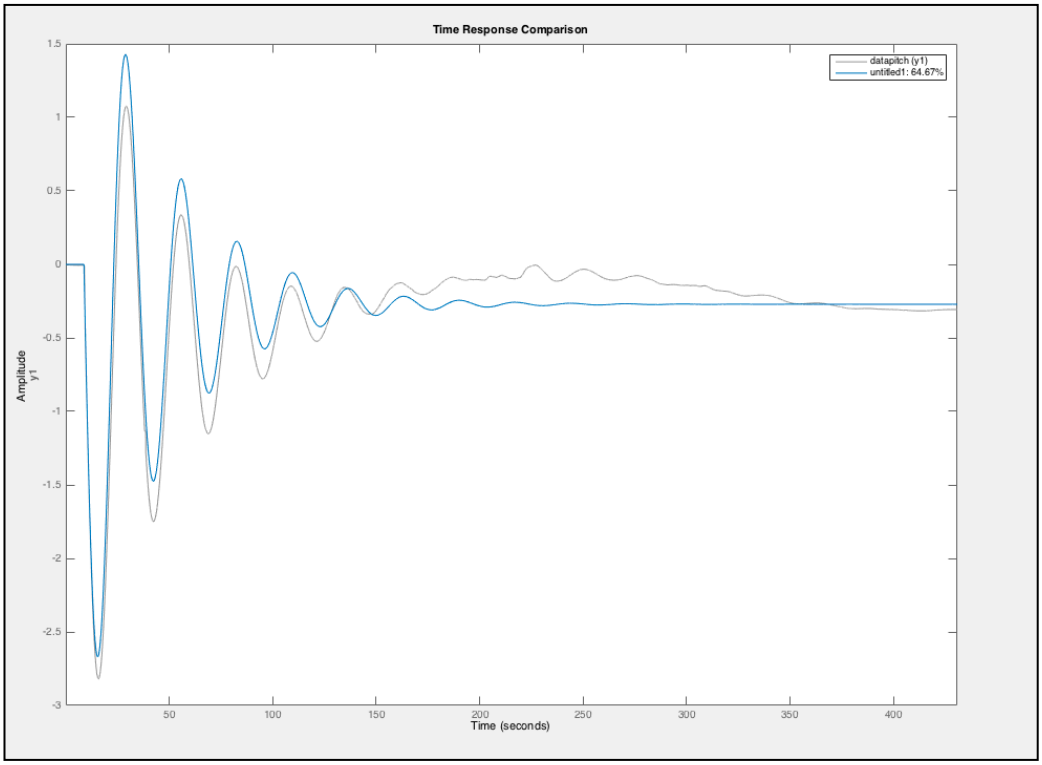


Figura 39: Comparación entre los valores reales y la respuesta de la función de transferencia en θ de los elevadores

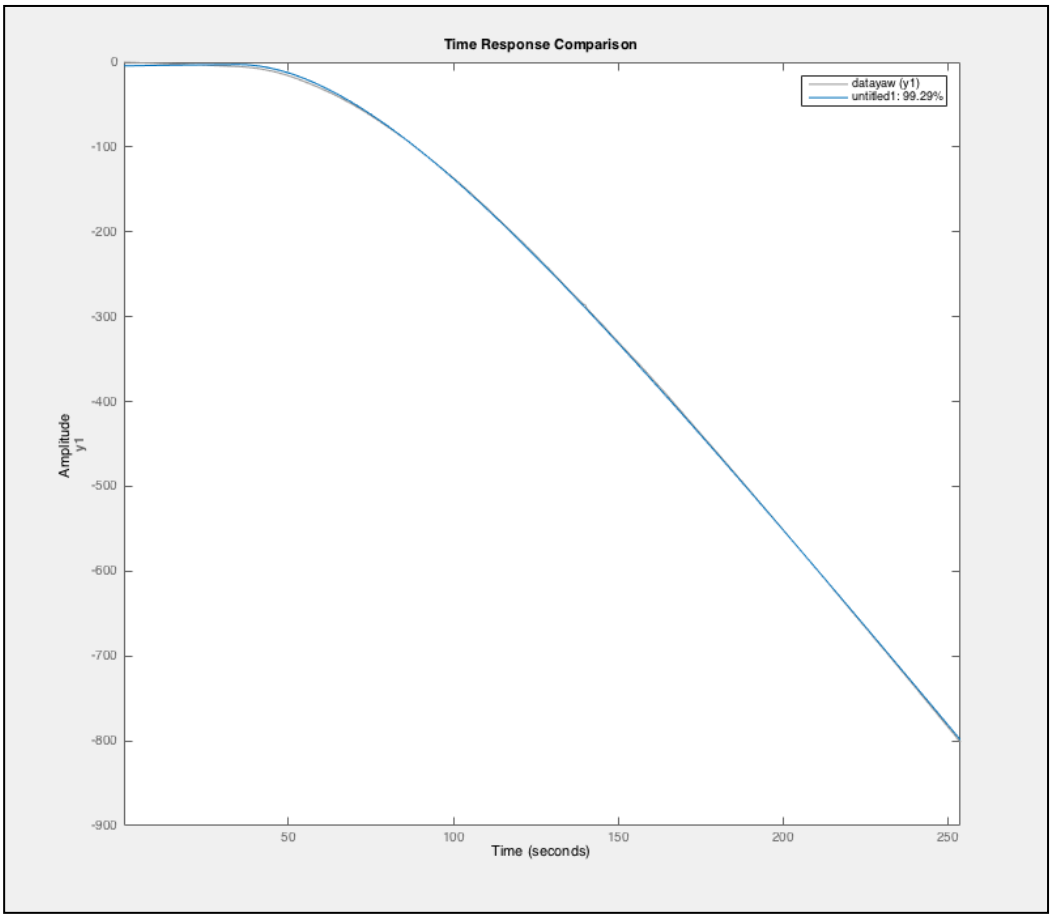


Figura 40: Comparación entre los valores reales y la respuesta de la función de transferencia en ψ de los elevadores

7.10 Modelado de la respuesta del timón de cola en los ángulos de Euler

Ángulo	Función de transferencia	Observaciones
ϕ	$\frac{942.6}{48.26\ s + 1}$	Figura 41: 52.39% de semejanza con los valores reales. Este muestreo puede ser el más complicado por que se necesitaba mucho tiempo para tomar buenos valores.
θ	$\frac{-5563\ s + 20.49}{1.467e6\ s^3 + 47450\ s^2 + 448.5s + 1}$	Figura 42: 40.03% de semejanza. En esta hay que tener mucho cuidado porque en realidad parece que oscila mucho pero no, en realidad podría ser casi despreciable porque varía como mucho 0.5 grados, lo que es muy poco.
ψ	$\frac{160}{53.42s^2 + s}$	Figura 43: 96.99% de semejanza con los valores reales.

Tabla 14: Funciones de transferencia del timón de cola

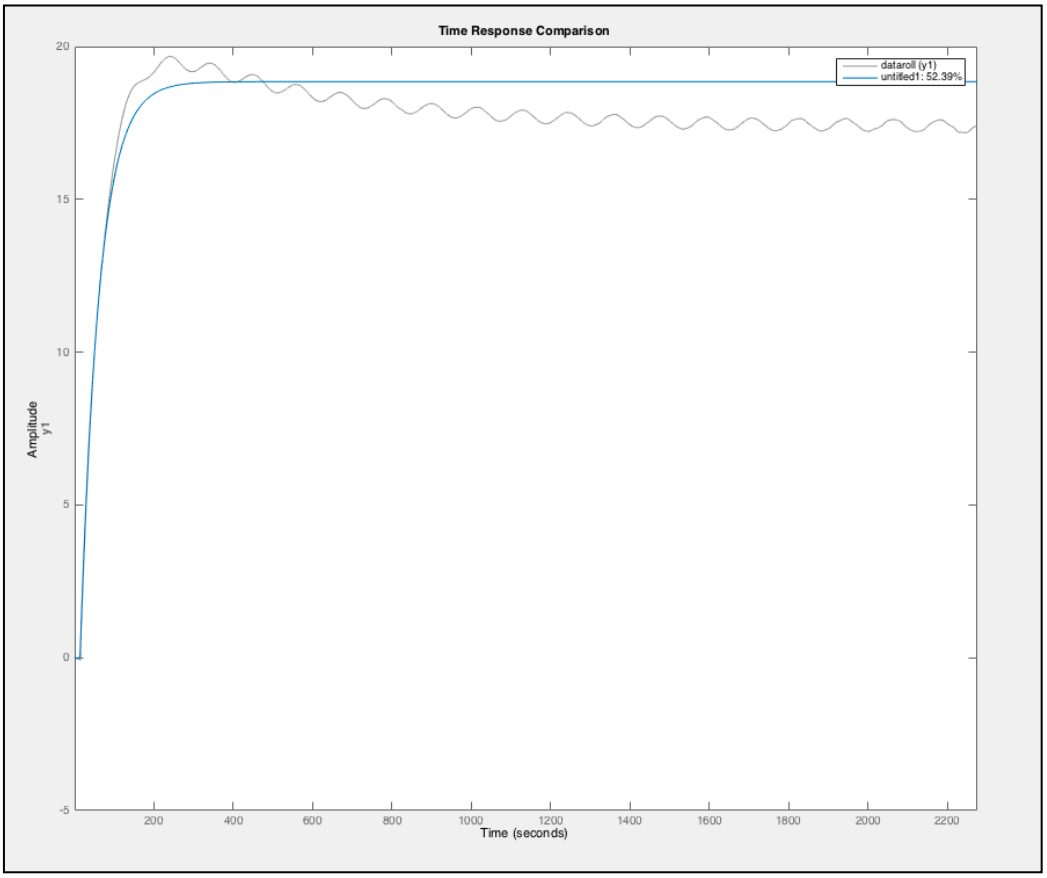


Figura 41: Comparación entre los valores reales y la respuesta de la función de transferencia en ϕ del timón de cola

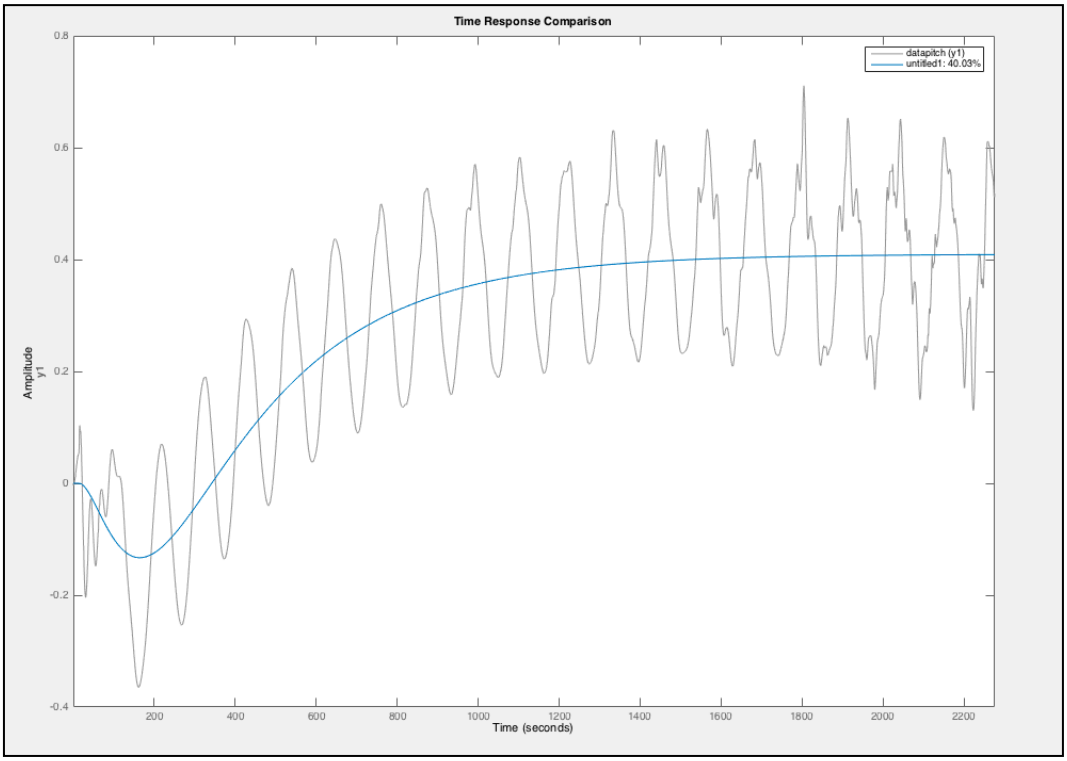


Figura 42: Comparación entre los valores reales y la respuesta de la función de transferencia en θ del timón de cola

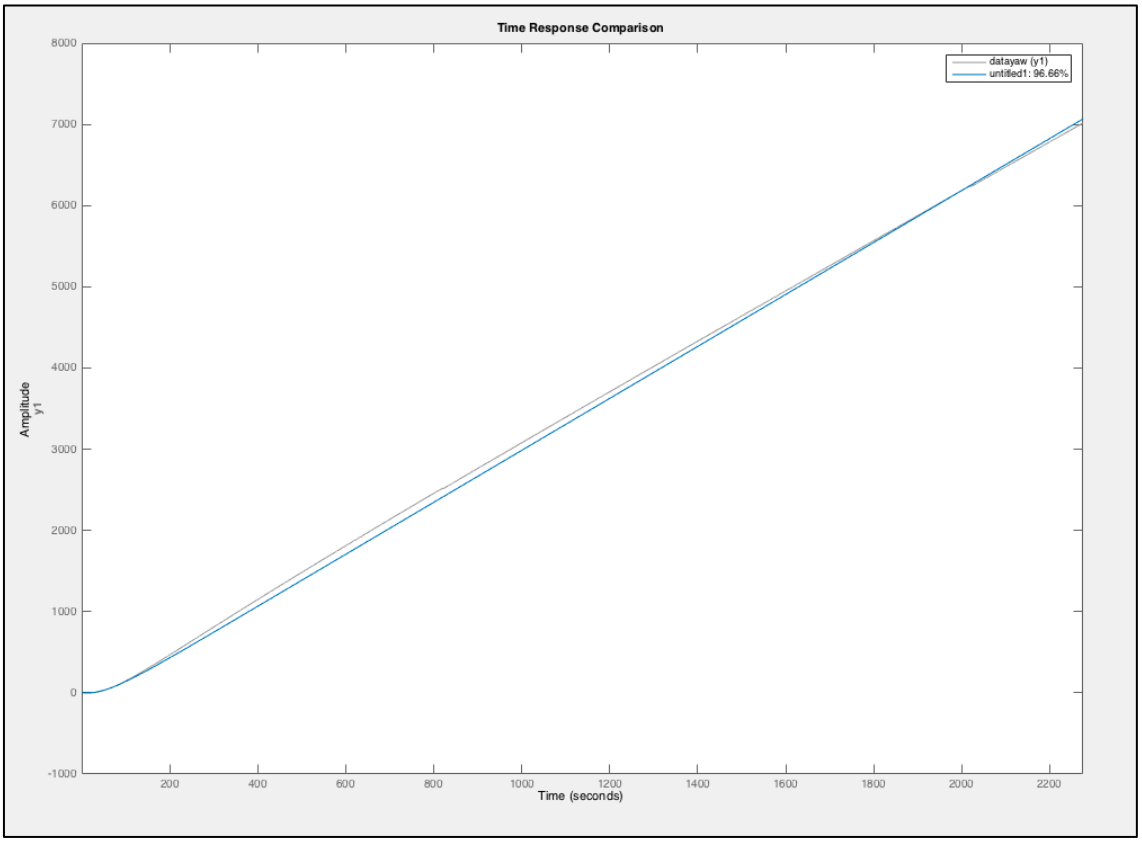


Figura 43: Comparación entre los valores reales y la respuesta de la función de transferencia en ψ del timón de cola

Ahora que hemos aproximado estas respuestas a un sistema de funciones de transferencia, vamos a ver como interactúan entre ellas y vamos a compararla con el sistema real.

7.11 Validación del modelado del sistema

Ahora que tenemos todas las funciones de transferencia, vamos a usar los principios de superposición para establecer un sistema de funciones de transferencias que nos de los valores de los ángulos de Euler cuando usemos los actuadores. Superpondremos las funciones de transferencia como aparece en el diagrama de bloques de la figura 45. Lo haremos en el mismo archivo simulink que el de recogida de datos para poder comparar en tiempo real si nuestro sistema lineal es válido.

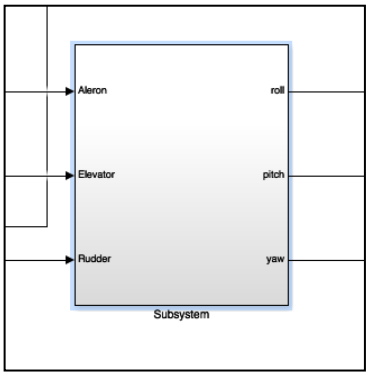


Figura 44: Bloque simulink del modelo lineal del sistema

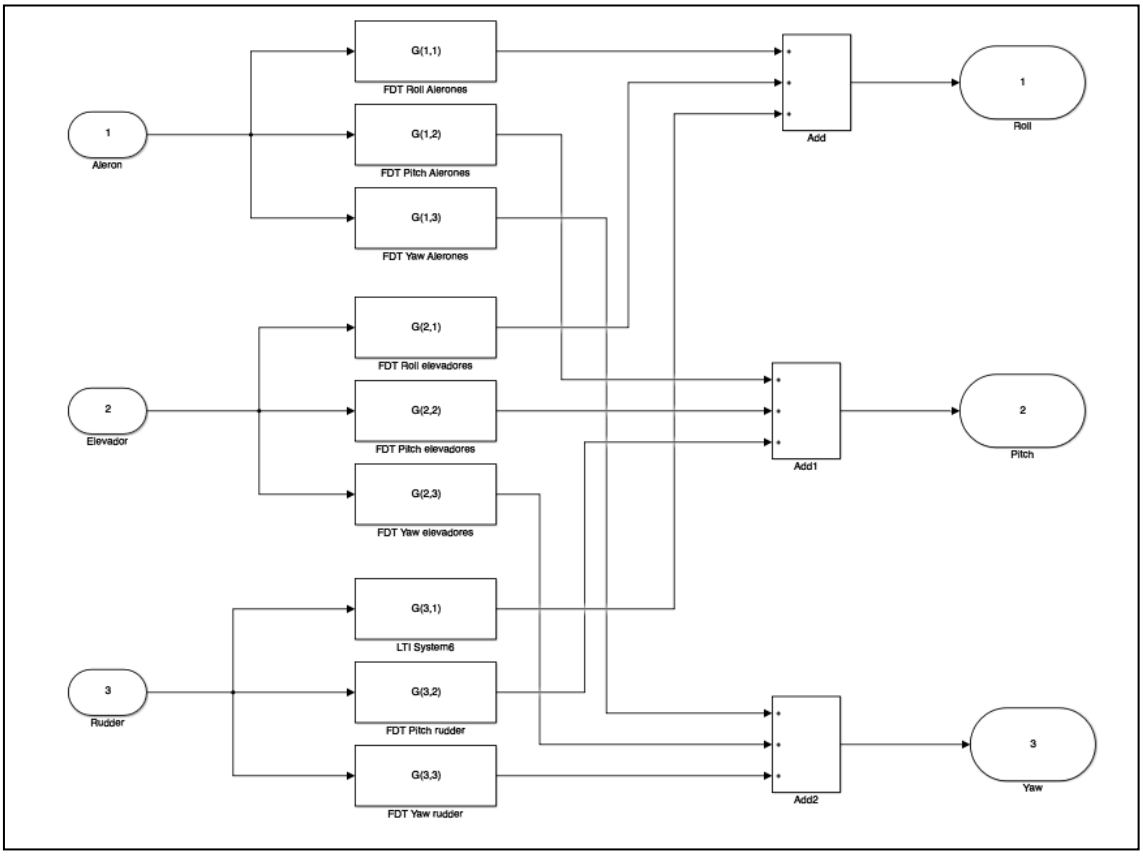


Figura 45: Diagrama de bloques del modelo del sistema por superposición.

Para ver si el sistema entero funciona correctamente, hemos hecho una experiencia en la cual vamos a ir manipulando los actuadores a nuestro gusto y vamos a observar si la respuesta de nuestro sistema lineal de funciones de transferencia se parece a lo que de verdad calcula el simulador.

Encontramos para cada ángulo los siguientes resultados:

- En ϕ $(roll, figura\ 46)$: En esta validación podemos comprobar que nuestro sistema se asemeja bastante a los datos reales, el único que se va un poco es cuando usamos el timón de cola, que los valores se separan un poco. También podemos observar que cuando nos empezamos a poner en ángulos más bruscos, el sistema tiene aún más error, así que solo lo usaremos para pequeños giros.
- En θ $(Pitch, figura\ 47)$: Este es el modelo que más nos preocupaba, ya que habíamos aproximado sin muchas oscilaciones y es lo que ocurre en la primera parte de la comparación, que existe bastante diferencia (relativa), pero aún así sigue siendo una diferencia de 1 grado. De todas formas, se puede observar que cuando usamos el actuador del elevador, el error es mínimo, así que parece que existe una buena interacción entre estos dos.
- En ψ $(Yaw, figura\ 48)$: Cuando estábamos modelando, las respuesta del Yaw eran las que más se ajustaban a nuestro modelo lineal, y esto se puede observar muy firmemente en la gráfica. Vemos que se asemejan bastante, lo que pasa es que los valores reales están acotados de 0 a 360, por eso ocurre ese salto, pero vemos que siguen la misma curva.

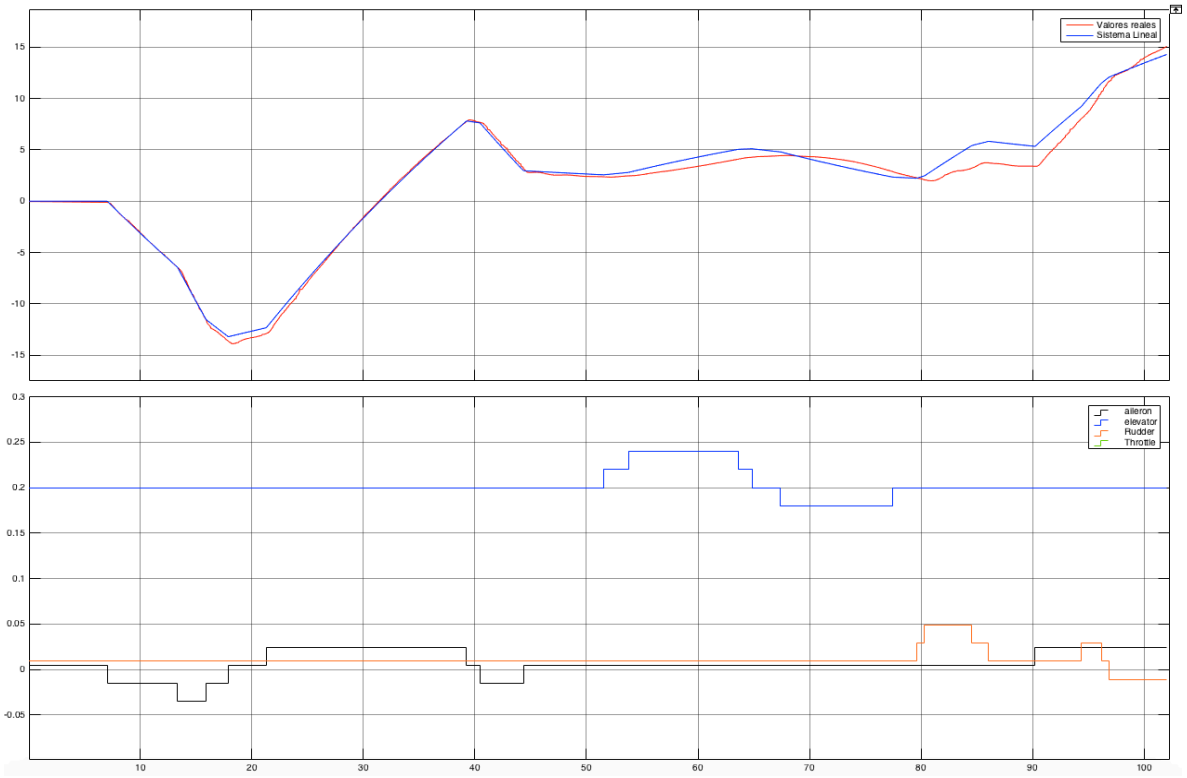


Figura 46: Gráficas de la validación del sistema lineal en el ángulo ϕ (X=tiempo en s, Y=ángulo en grados.

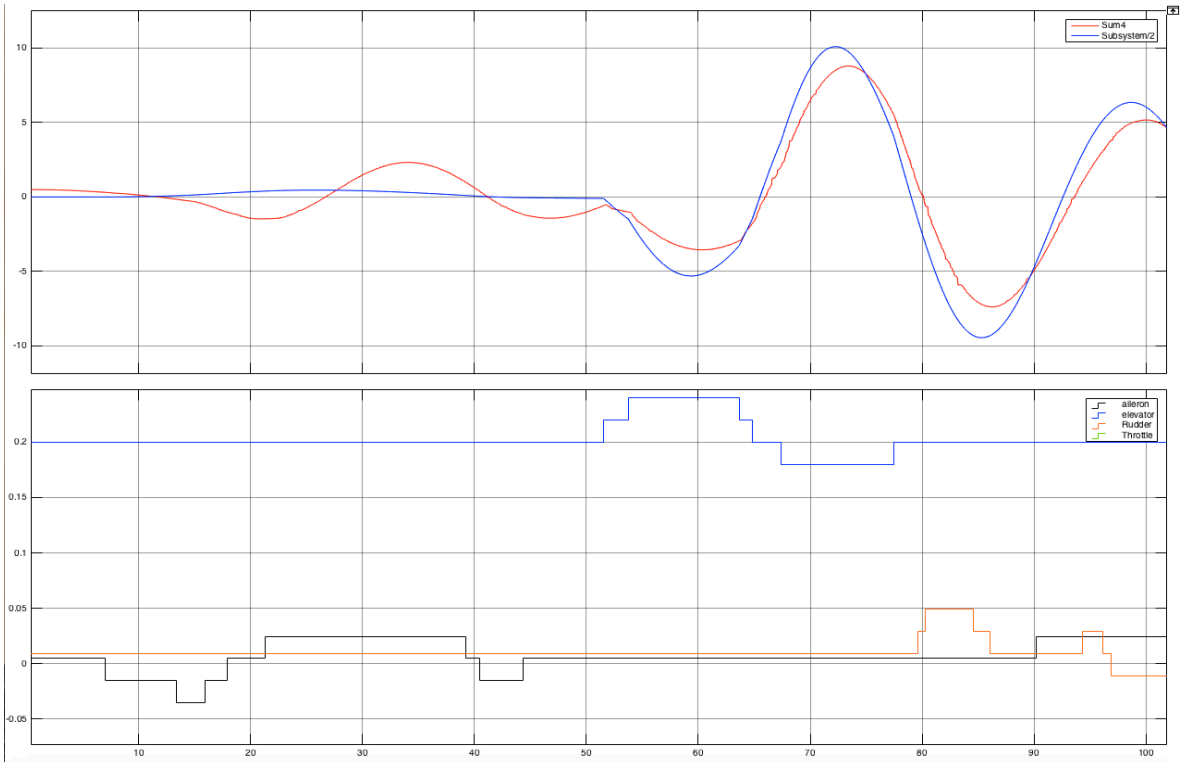


Figura 47: Gráficas de la validación del sistema lineal en el ángulo θ (X=tiempo en s, Y=ángulo en grados.

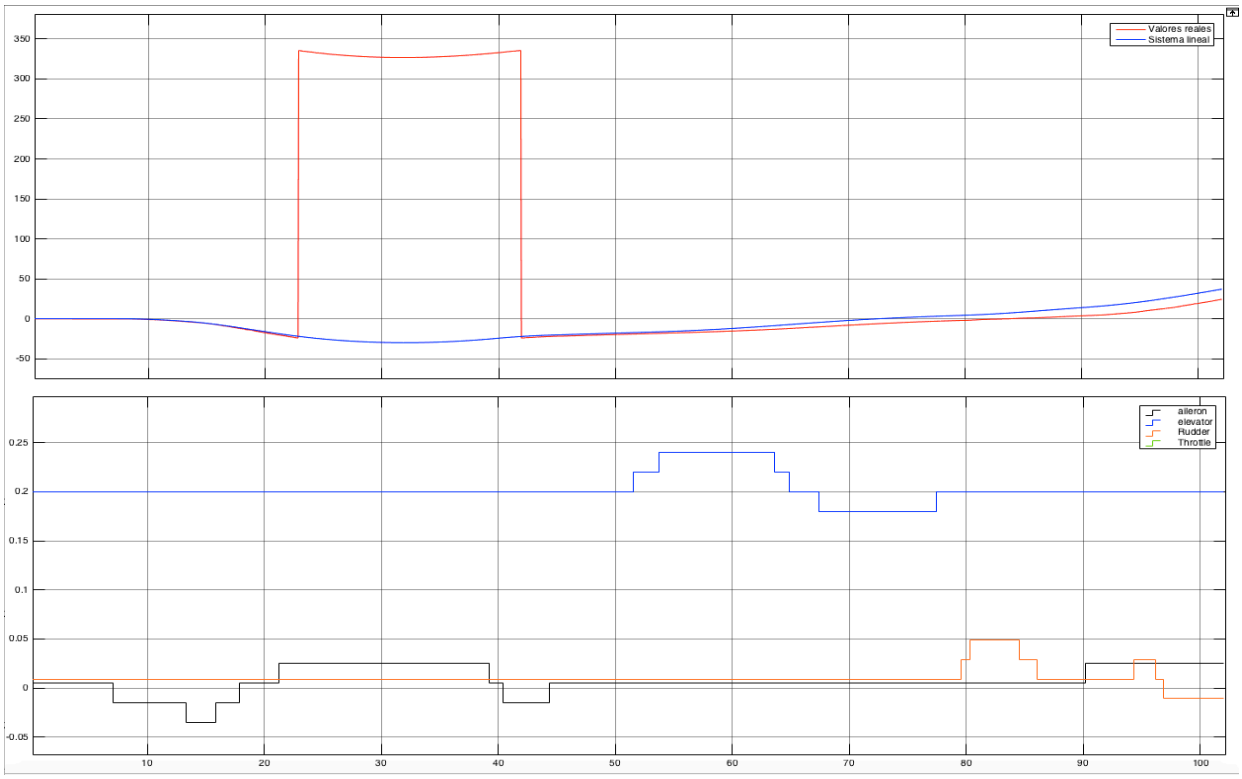


Figura 48: Gráficas de la validación del sistema lineal en el ángulo ψ (X=tiempo en s, Y=ángulo en grados.

8. Diseño de controladores.

8.1 Conceptos básicos de los controladores.

El objetivo de un controlador, es conseguir que un proceso tenga el comportamiento que se desee, el cual describiremos con un conjunto de objetivos que queremos cumplir:

- Especificaciones del sistema: La velocidad, la calidad (oscilaciones).
- Las restricciones de seguridad y protección medio ambiental (residuos etc)
- Objetivos económicos: la eficiencia.

Así pues, dichos controladores se encargaran de tomar las decisiones de cómo afectar a cada variable manipulada para que la variable controlada cumpla las especificaciones.
Queremos que los reguladores eliminen en todo lo posible el efecto de las perturbaciones que puedan afectar a nuestro sistema y disminuir el efecto de variaciones de los parámetros del proceso sobre la variable controlada (en otras palabras la sensibilidad). Por eso mismo para hacer un controlador óptimo, usaremos lo que se llama un control bucle cerrado (figura 49) el cual nos proporcionará una menor sensibilidad a los errores de modela y una alta capacidad de rechazar perturbaciones.

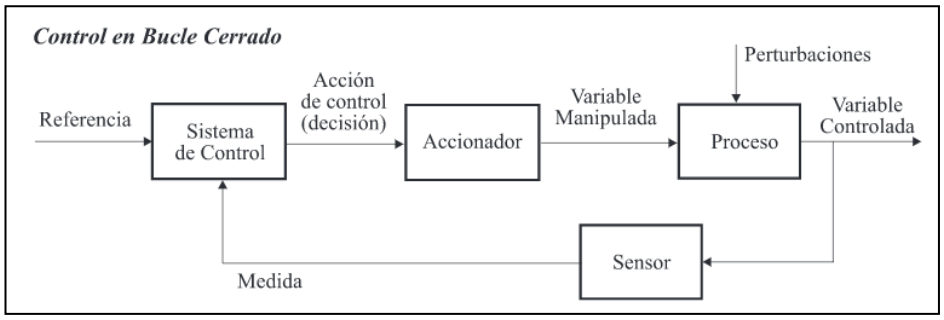


Figura 49: Diagrama de bloques de un control bucle cerrado

La función de transferencia del Bucle Cerrado será la siguiente (figura 50):

$$G_{BC}(s) = \frac{G_r(s)G(s)}{1 + G_r(s)G(s)H(s)}$$

Donde:

$$G_r(s): Función\ de\ transferencia\ del\ controlador$$
$$G(s): Función\ de\ transferencia\ del\ proceso$$
$$H(s): Función\ de\ transferencia\ del\ sensor$$
$$1 + G_r(s)G(s)H(s) = 0: Ecuación\ caracteristica\ del\ sistema$$
$$E(s): Error\ entre\ la\ referencia\ y\ la\ variable\ controlada$$

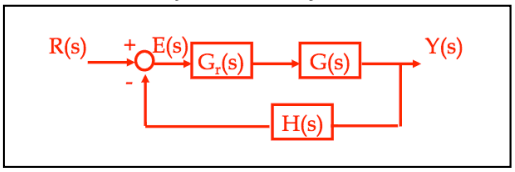


Figura 50: Diagrama de bloques de un control bucle cerrado

Este error entre la referencia y la variable controlada pueden ser calculado en régimen estacionario. según el tipo de sistema en el que estamos trabajando. El tipo de sistema será definido por el número de integradores que presente el bucle en abierto (es decir como si no estuviera cerrado). Cuanto mayor sea el tipo, mayor calidad tendrá la respuesta en régimen estacionario. Además existen tres tipos de errores también:

- Error de posición: Es el error haciendo que s tienda al infinito de una entrada en escalón del bucle abierto.
- Error de velocidad: Lo mismo pero con una entrada rampa unitaria.
- Error de aceleración: Es lo mismo pero ante una entrada parábola unitaria.

Los errores pueden ser calculados rápidamente mediante la siguiente tabla 15, que reúne el tipo de sistema y el tipo de error.

Tipo \ Error	E_P	E_v	E_a
0	$\frac{1}{1+K_P}$	∞	∞
1	0	$\frac{1}{K_v}$	∞
2	0	0	$\frac{1}{K_a}$

Tabla 15: Relaciones de los errores con el sistema

Una vez tenemos claro esto, tenemos que introducir el concepto de lugar de las raíces.

El lugar de las raíces se define como el lugar geométrico de los puntos del plano complejo que son solución de la ecuación características para cualquier valor de K. Donde K serán las ganancias proporcionadas por el controlador. Esto es de gran ayuda ya que de una manera bastante sencilla podemos encontrar el controlador que queramos para nuestras especificaciones sin arriesgarnos a convertirlo en un sistema inestable.
Para analizar el sistema en el lugar de las raíces usaremos la función de Matlab© *rltool,* que nos permite visualizar el lugar de las raíces para cualquier función de transferencia dada.

8.2 Conceptos sobre reguladores PID.

La estructura de control que vamos a utilizar consta de 3 acciones básicas.

- P: Proporcional → Es instantánea, es una amplificación de error y depende del presente.
- I: Integral → Depende del pasado, es un cumulo del error anterior.
- D: Derivada → Va a intentar predecir el error futuro (viene bien para el régimen transitorio).

Combinando adecuadamente estas tres acciones, encontraremos la familia que se conoce como PIDs, y son los que usaremos nosotros.

Lo que pasa es que no usaremos las acciones aisladas, sino que las combinaremos con la proporcional (P), así que existirán 4 combinaciones cuyas funciones de transferencia serán:

$$P \rightarrow Gr(s) = Kc$$
$$PI \rightarrow Gr(s) = Kc\ \frac{(s+a)}{s}$$
$$PD \rightarrow Gr(s) = Kc\ (s+b)$$
$$PID \rightarrow Gr(s) = Kc\ \frac{(s+a)(s+b)}{s}$$

Y siempre intentaremos establecer el controlar más sencillo posible que cumpla las especificaciones del sistema en el orden de P → PI → PD → PID.

8.3 Elección de los controladores.

Nuestro modelo se trata de un sistema multivariable, donde cada una de las entradas no solo afecta a las tres salidas sino que su interacción puede provocar el cambio de otro controlador. Por eso tenemos que introducir el concepto de RGA (Relative Gain Array).

Nuestro sistema multivariable será:

$$\begin{pmatrix}\phi\\ \theta\\ \psi\end{pmatrix} = \begin{pmatrix}G(1,1) & G(1,2) & G(1,3)\\ G(2,1) & G(2,2) & G(2,3)\\ G(3,1) & G(3,2) & G(3,3)\end{pmatrix}\begin{pmatrix}\delta a\\ \delta e\\ \delta r\end{pmatrix}$$

La matriz RGA nos va indicar a que actuador puede controlar que ángulo según menor sea el efecto de la interacción de los otros controladores y para ello nos haremos uso de las ganancias estáticas de las funciones de transferencia. Para ello recogeremos las ganancias de nuestro sistema en una matriz, cuyo resultado será el siguiente:

$$K = \begin{bmatrix} 2701 & 296.4 & 942.6\\ -121.9 & -13.6 & 20.5\\ 544.5 & 94.3 & 160\end{bmatrix}$$

Ahora la matriz RGA la sacaremos usando la función:

$$RGA = K \otimes (K^{-1})^T$$

Donde el operador ⊗ es la multiplicación elemento a elemento, no matricial.

Al final nos queda que:

$$RGA = \begin{bmatrix} 1.8905 & -1.5508 & 0.6603\\ 0.8617 & -0.1877 & 0.3260\\ -1.7522 & 2.7384 & 0.0138\end{bmatrix}$$

¿Esto de que nos sirve para elegir los controladores?

Pues bien lo que nos indica cada elemento λ en la matriz RGA es la relación entre el efecto directo de una actuador sobre la variable controlaba y el efecto de las interacciones de las demás variables controladas de modo que:

$$\lambda_{ij} = \frac{efecto_{directo}}{efecto_{directo} + efecto_{interacción}}$$

Esto quiere decir que cuanto más próximo sea λ de 1, menos interacción harán el resto de variables y mejor podremos controlar (1 es el caso ideal). Los emparejamiento irán según las recomendaciones de la siguiente tabla:

Valor de λ	Explicación	Recomendación
$\lambda_{ij} = 1$	$efecto_{interacción} = 0$	Es el caso ideal, se empareja uj con yi
$\lambda_{ij} = 0$	$efecto_{interacción} = \infty$ $efecto_{directo} = 0$	En este caso el control es imposible no emparejar
$0 < \lambda_{ij} < 1$	Los dos efectos van en el mismo sentido	Si $\lambda_{ij} < 0.5$ no emparejar por que significa que el efecto de interacción es mayor que el directo
$\lambda_{ij} > 1$	Efecto directo mayor que el valor absoluto de interacción y son de sentido contrario.	Hay que evitar demasiado elevados por que la interacción acaba cancelando el efecto directo Para $\lambda_{ij} > 10$ no emparejar
$\lambda_{ij} < 0$	Son de sentido contrario pero esta vez es la interacción mayor que el efecto directo	No emparejar ya que hay peligro de que si el resto de bucles se abren, este control se quedaría inestable.
$\lambda_{ij} = \infty$	Efecto directo = -efecto interacción	Control imposible

Tabla 16: Emparejamientos según el valor de λ_{ij}

Es aquí donde nos damos cuenta de varias cosas en nuestro planteamiento del control automático de la aeronave.

- En primer lugar ver que el emparejamiento Elevadores-Cabeceo, no debería hacerse ya que sale negativo, y eso quiere decir que solo podríamos controlarlo si el resto de controles estuviesen activados (por si solo no se valdría).
- Acto seguido vemos que en el punto (2,1), que sería un buen emparejamiento, este serían los alerones que controlasen el cabeceo (Pitch), lo cual desde un punto físico es imposible o como poco muy complejo de controlar.
- Por último vemos que según nuestra matriz, los alerones no deberían poder controlar la guiñada (rumbo/yaw), sin embargo sabemos de buena tinta que cuando producimos un giro, lo primero que usamos son los alerones.

Hemos constatado que surge un gran problema en nuestro sistema, pero buscamos el por que de todo esto. Al final hemos constatado varias hipótesis.

Problemas:

- Desde un principio supusimos mal al creer que debíamos controlar estos ángulos, ya que desde un punto de vista físico, estos sistemas están totalmente acoplados. Por ejemplo tu a la aeronave no puedes ponerle una referencia de rumbo de 60 grados y roll de 5 grados, por que cuando el controlador llegue al rumbo de 60 le dirá al otro controlador que el roll no puede seguir a 5 grados para parar de girar, pero este tiene la referencia en 5 y se estorbarán, por lo que no podemos hacer un control de este tipo.
- Cuando estudiamos las respuestas de los actuadores, estábamos dejando al avión demasiados grados de libertad por lo que algunos controladores no iban a hacer efecto (como es el del cabeceo), ya que sabemos a ciencia cierta, que lo que maneja el cabeceo de una aeronave son los elevadores y la palanca de gases, pero de ningún modo el timón de cola o los alerones.
- Vemos con todo esto, que de una manera experimental (que era el objetivo de este proyecto), no podemos juntarnos demasiado con las ecuaciones cinemáticas, debemos encontrar otro método para analizar estos comportamientos.

Soluciones:

- Como hemos comentado, hay ángulos que no podemos controlar a la vez, así que desde un punto de vista practico, estableceremos como el ángulo que queremos controlar el rumbo, ya que en la Aeronavegación es el más importante para llegar al punto deseado con un piloto automático. Así que no necesitaremos controlar el roll, sino que el roll irá relacionado directamente con nuestro controlador en el rumbo (Yaw).
- En realidad nos damos cuenta que lo único que necesitamos después de controlar el rumbo para obtener el modelo de un piloto automático, es controlar la altitud y la velocidad. Así que tendremos que atribuirle dos actuadores a estas dos.
- Cuando se esta en vuelo crucero, el actuador que controlar la altitud es el elevador y cuando se esta despegando es la palanca de gases.
- Cuando estamos en vuelo crucero, será la palanca de gases la que controle la velocidad.
- Como hemos visto anteriormente con los efectos de las hélices (causa por la que la matriz RGA no deja que los elevadores controlen el cabeceo), necesitamos obtener el modelo de los elevadores con la altitud de vuelo pero esta vez con el controlador de los alerones y el rumbo, de este modo si que podremos obtener un buen modelo.
- Por último nos queda el timón de cola, el cual en las aeronaves convencionales, cuando se produce un giro, se utiliza para suavizar el efecto de los alerones, así que lo pondremos como un complemento del controlador de estos.

8.4 Diseño del controlador de los alerones para el rumbo

La función de transferencia que emparejaba los alerones con el rumbo (o Yaw ya que habíamos establecido el eje X horizonte local en el norte) era la siguiente:

$$\frac{544.5}{s\,(59.61\,s+1)}$$

Estableceremos las especificaciones siguientes para el sistema:

- Error de posición 0 ya que vamos a tener entradas en escalón.
- Tiempo de establecimiento en 60 segundos. Una aeronave ligera como esta debe girar los 360º en dos minutos para que no sean giros muy bruscos. Por eso establecemos ese tiempo por el peor de los casos en el que tenga que hacer 180 grados.
- Sobreoscilaciones queremos las mínimas posibles.

Para empezar el sistema es de tipo 1 ya que posee un integrador, así que no nos tenemos que preocupar por el error de posición.

Usando la función *rltool* de Matlab© (figura 51) observamos que para que se cumplan la especificación del tiempo de establecimiento (línea vertical en negrita), es necesario establecer un controlador derivado, es decir un PD. Ya que podemos, el cero de este controlador cancelará el polo de la función de transferencia, así que lo situamos en -1/60. Al final vemos en el lugar de las raíces que con este controlador si que podemos obtener dicha especificación. Además, como una función de transferencia no puede tener más ceros que polos, decidimos poner en el controlador un polo 100 veces mayor que el cero establecido (figura x.x), para que así este no predomine. Al final el controlador nos quedaría de la forma:

$$Gr(s) = 0.0001\,\frac{(60\,s+1)}{(0.6\,s+1)}$$

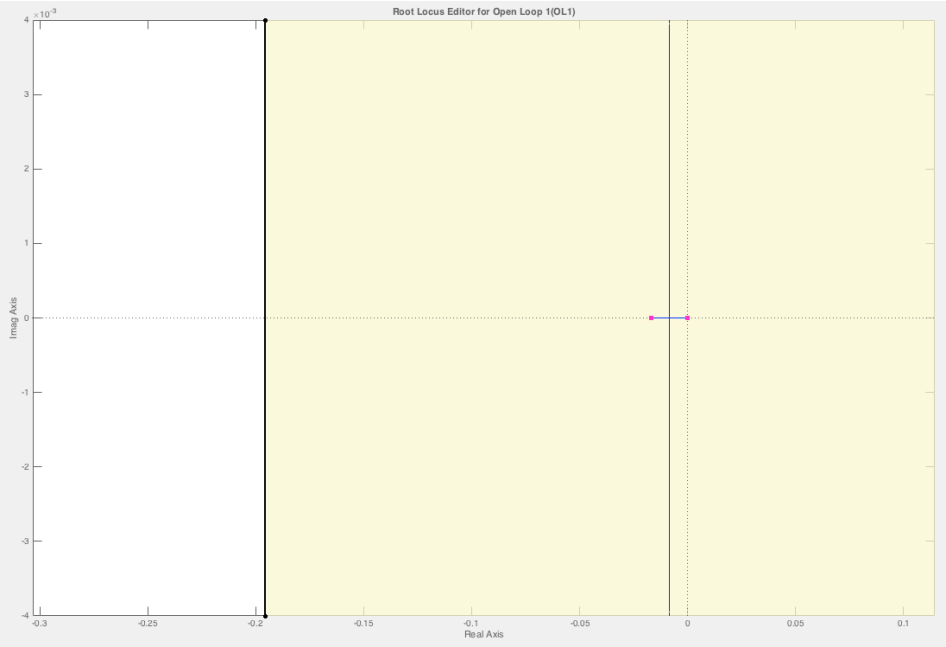


Figura 51: Lugar de las raíces de la función de transferencia Alerón-Rumbo sin el controlador

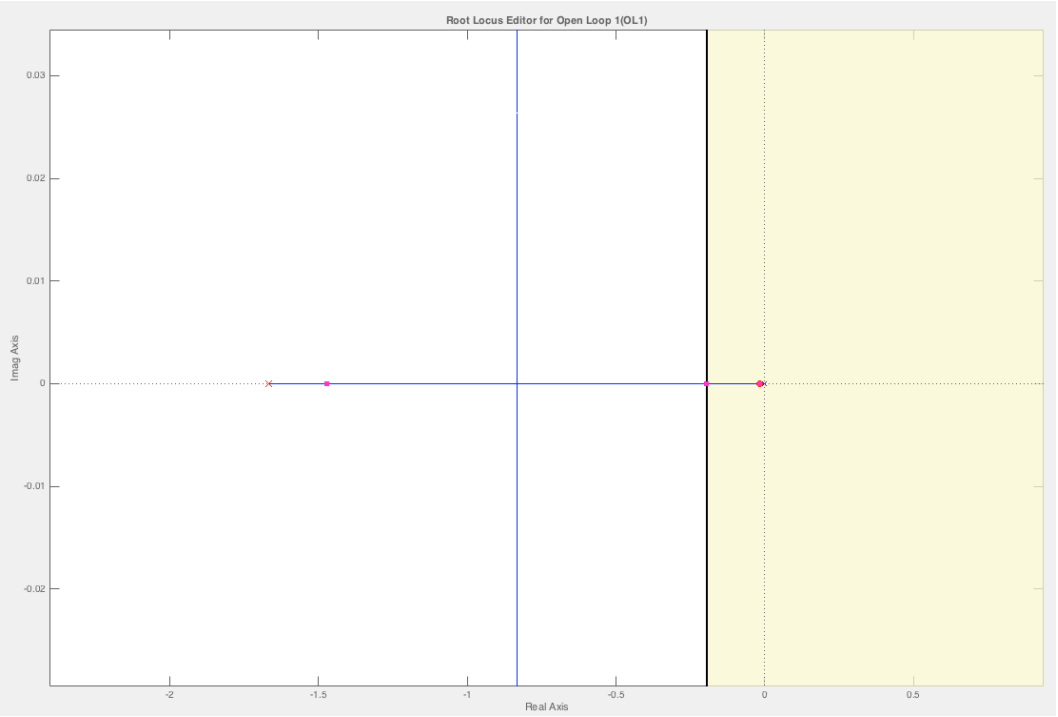


Figura 52: Lugar de las raíces de la función de transferencia Alerón-Rumbo con el controlador

8.5 Validación del controlador Alerón-Rumbo

Antes de poner el controlador en práctica con el simulador de vuelo, vamos a hacer una pequeña validación poniendo el modelo lineal en simulink©, y así veremos como responde de manera teórica este sistema.

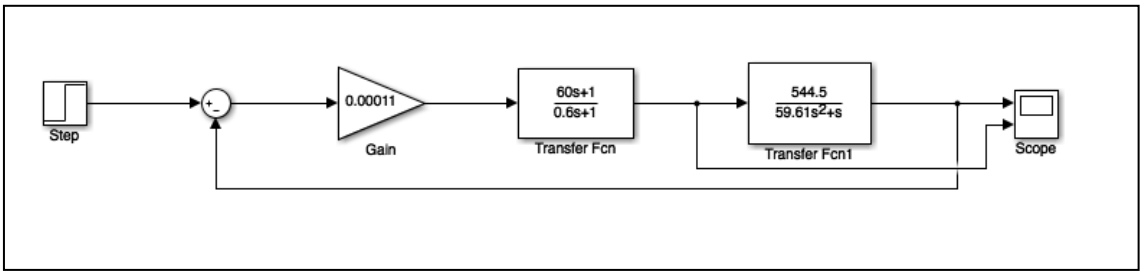


Figura 53: Bloques de validación del controlador del sistema lineal Alerones-Rumbo

En la figura 54 vemos que los resultados son coherentes para una referencia de una variación de rumbo de 50º, el único inconveniente que nos encontraremos es la alta variación de alerones que seguramente en algún momento si la variación del rumbo es más elevada, nos puede llevar a ser inestable por saturación, ya que recordemos que los alerones van de -1 a 1 (en este ejemplo ya es brusco porque ponemos los alerones a 0.5). Tendremos que ver como lo resolveremos. Vemos que la especificación del tiempo se cumple sin ningún problema.

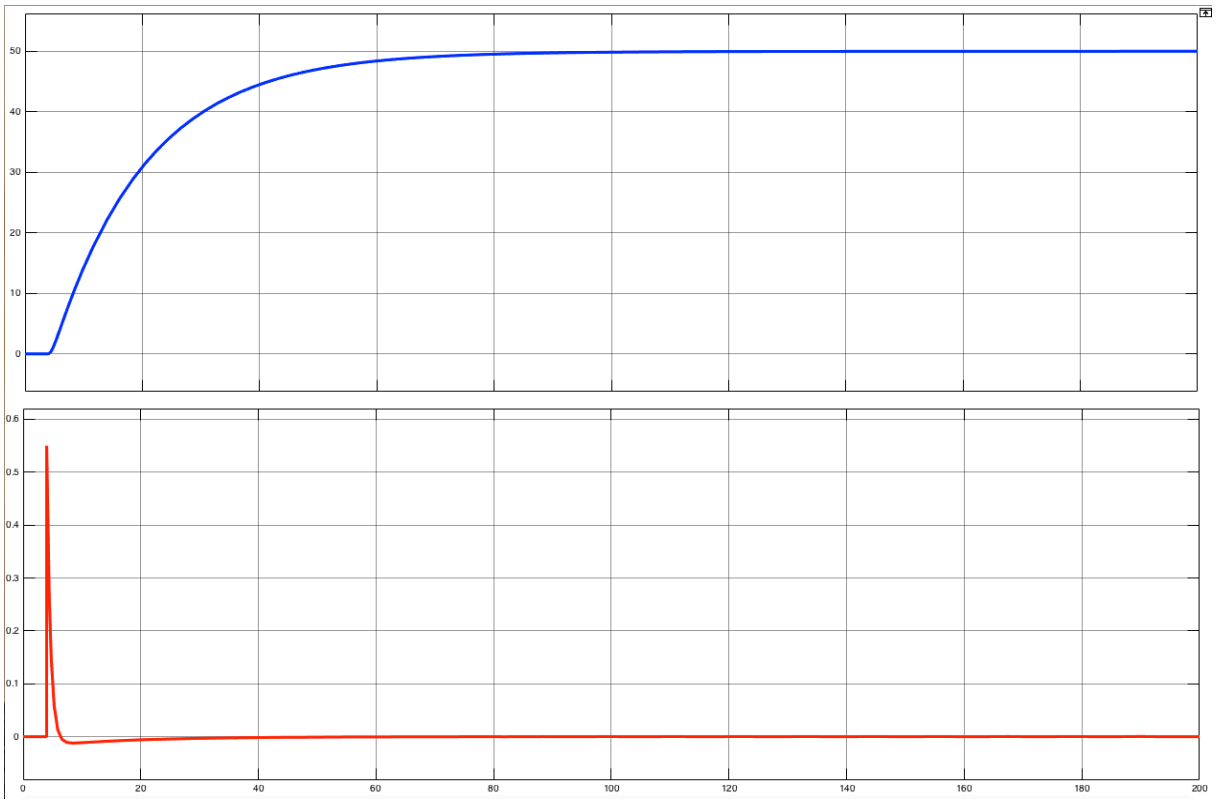


Figura 54: Respuesta del controlador Alerones-Rumbo en el modelo lineal. (X=tiempo en s, Y=ángulo en grados.)

Así como para grandes giros el movimiento de los alerones será demasiado brusco, para giros muy pequeños ocurrirá todo lo contrario, tan pequeño será a veces la variación del actuador del controlador que el simulador no hará caso. Por ello nos proponemos implementar un sistema. Este sistema tratará de darle el tiempo de establecimiento cada 5 grados girados, así podremos obtener una buena exactitud en el tiempo que deseamos.

Si en 360º tardamos 4 minutos, en 10 grados tardaremos que hacerlo en 6.66 segundos así que pondremos el nuevo tiempo de establecimiento en 7 segundos. El nuevo controlador será:

$$Gr(s) = 0.0002\ \frac{(60\ s + 1)}{(0.6\ s + 1)}$$

Esto podremos ir variándolo según la sensibilidad que queramos, en este caso la sensibilidad será 10 grados. Esto dará la figura 56.

Sin embargo existe otra manera para poder implementar el controlador principal que queríamos sin que de giros demasiado bruscos, y esto se consigue poniendo un filtro de saturación, es decir poner un bloque de saturación diciéndole que no le deje al controlador pasar de un rango, que nosotros pondremos de -0.2 a 0.2 para los alerones. Esto nos permitirá dar giros suaves y afectivo, ya que en algunos momentos nuestro controlador anterior no giraba porque pensaba que siempre estaba a 10 grados y le estábamos "engañando". En la figura 58 podemos ver la comparativa de las dos trayectorias.

8.6 Piloto automático

Ahora ya tenemos un modelo válido para los controladores pero además queremos probarlo no para escoger un rumbo sino para llegar al waypoint que deseemos. Por eso mismo vamos a implementar un bloque en simulink© (figura 55) que según la posición en la que estemos y las coordenadas a las que queramos llegar, nos vaya calculando que rumbo tenemos que coger (como referencia).

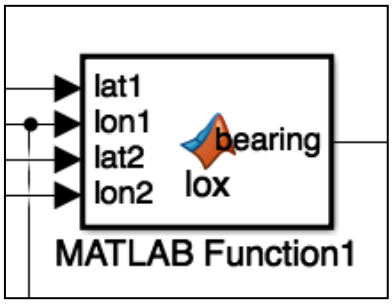


Figura 55: Bloque con la formula del rumbo inicial

Para la obtención de dicho rumbo, haremos uso de la formula que indica el rumbo inicial a seguir con el arco del círculo máximo que pasará entre los dos puntos. En teoría no va a hacer falta un calculo loxodrómico directo ya que en realidad estamos recalculando cada vez dicho rumbo en cada iteración, ya que sabemos en todo momento la situación de nuestra aeronave. La fórmula para el calculo de dicho rumbo será:

$$\theta = atan2(sin\Delta\lambda \cos\varphi_2, cos\varphi_1\ sin\varphi_2 - sin\,\varphi_1\ cos\varphi_2\ cos\Delta\lambda)$$

Este será nuestro parámetro de referencia de nuestro controlador.

Luego hemos implementado dos bloques más que podrán observar su código en el anexo que del final. La finalidad de los dos es uno que lea los waypoints por los que deseamos pasar para enviárselos a la función que nos calcula el bearing, y el otro un pequeño script que definirá hacia que lado tiene que girar la aeronave para que sea más corto el giro y que sensibilidad tendrá la aeronave (en nuestro caso 10 grados). Recordamos que esto último es por que el tiempo de establecimiento tenemos que intentar definirlo para un giro concreto (10 grados en este caso) y de ahí la aeronave vaya girando de 10 en 10 hasta llegar al objetivo. Si no hiciéramos esto, la aeronave pegaría giros demasiado bruscos.

En la figura 56 podemos observar la ruta que ha seguido el avión con el controlador habiendo definido 3 waypoints. Podemos observar que ha sido bastante exitoso, aunque a veces hay un pequeño error. Este error lo ponemos nosotros ya que hemos calculado que cuando la distancia entre el waypoint y la aeronave es menor de 1km, el sistema defina que hemos llegado a ese punto y comience a girar hacia el siguiente. La distancia se ha calculado con la formula de Haversine que es la siguiente:

$$a = sin^2\left(\frac{\Delta\varphi}{2}\right) + cos\varphi_1\, cos\varphi_2\, sin^2\left(\frac{\Delta\lambda}{2}\right)$$

$$c = 2\, atan2(\sqrt{a}, \sqrt{(1-a)})$$

$$d = R * c$$

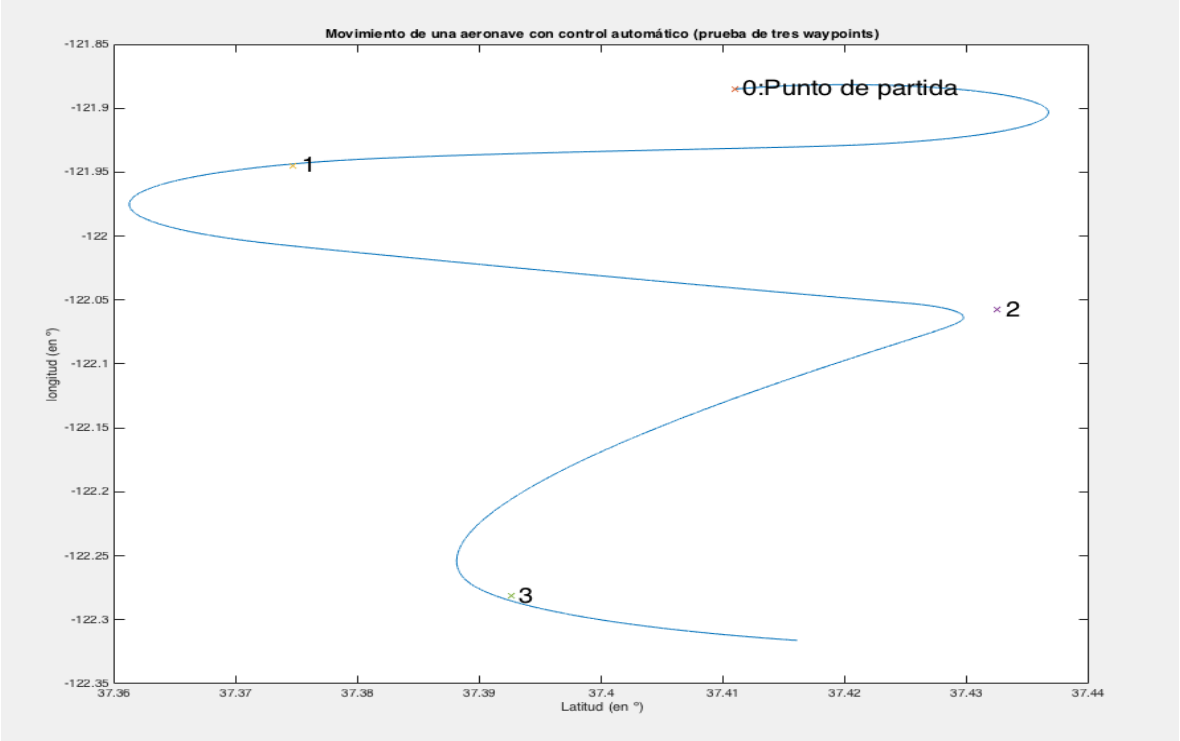


Figura 56: Recorrido de la aeronave con el piloto automático a través de 3 waypoints.

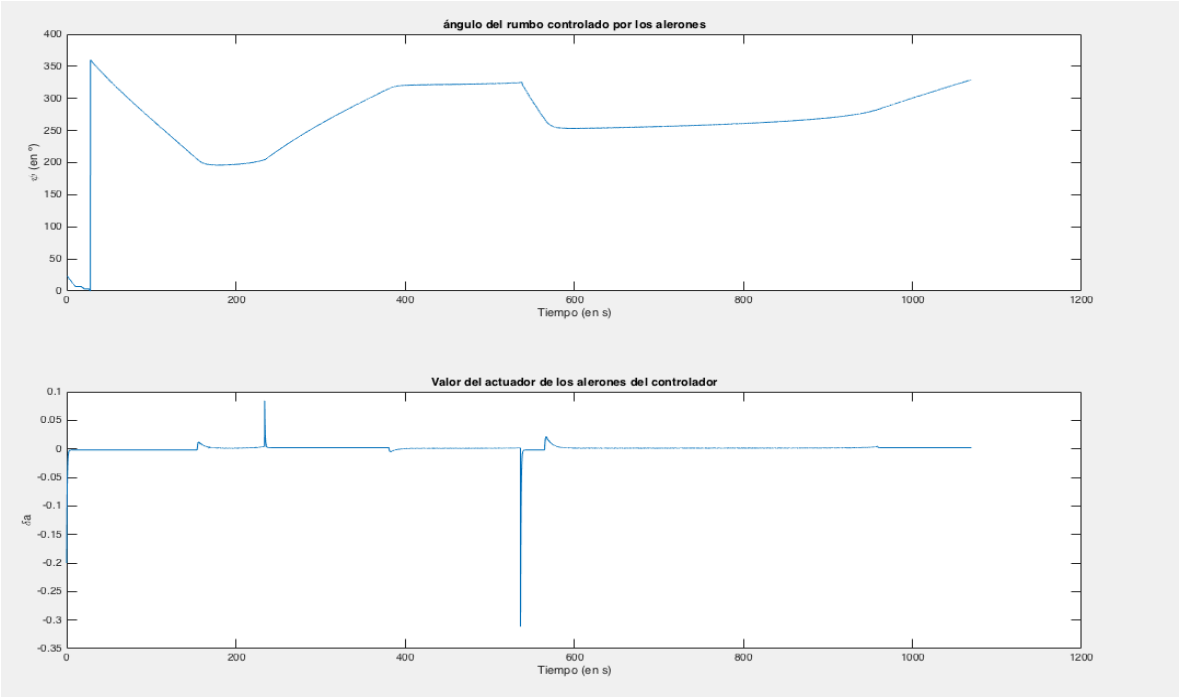


Figura 57: Gráfica del comportamiento del rumbo con el controlador de los alerones.

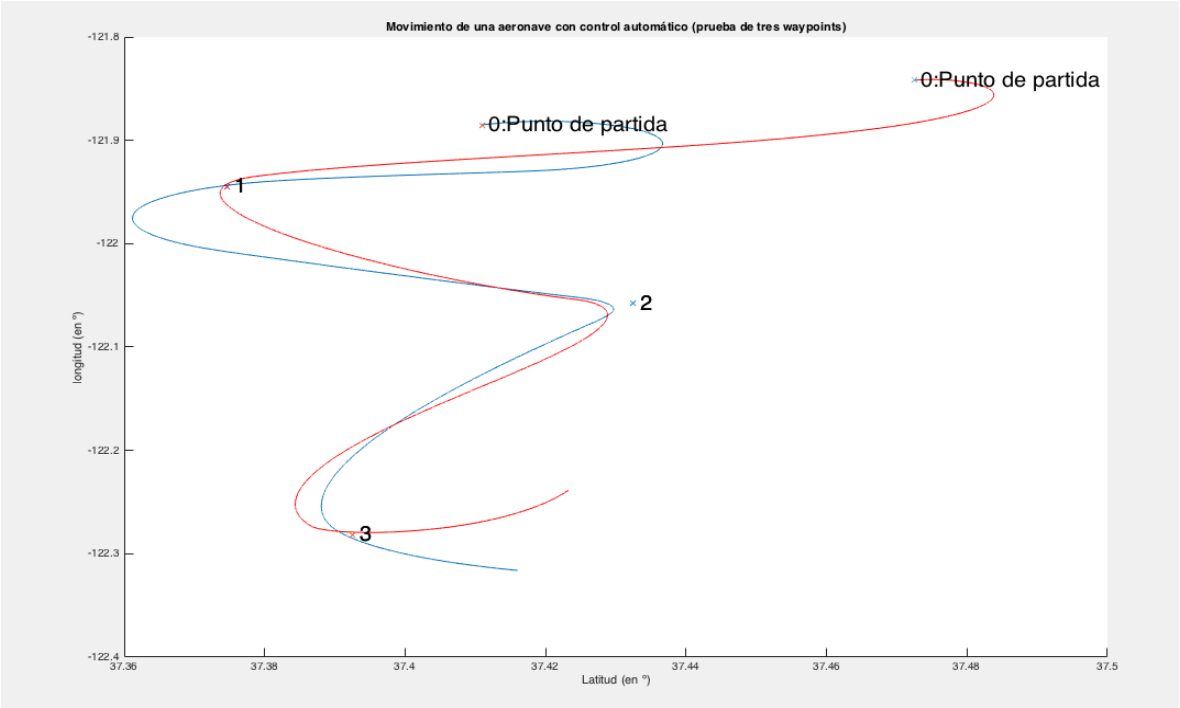


Figura 58: Comparación de las dos maneras de filtrar el controlador. Rojo con saturación, Azul manera implementada por nosotros.

Como podemos observar, los giros usando el antiwindup (azul), son más efectivos que sin este. Eso lo podemos observar perfectamente cuando pasan por el Waypoint 1.

9. Nivel de prestaciones obtenidos y conclusiones.

A lo largo de este proyecto se ha obtenido la plataforma que se deseaba, es decir un entorno en el cual podamos analizar el comportamiento de una aeronave en base a los impulsos que le damos a los actuadores. Sin embargo, solo se ha podido controlar el rumbo, lo que es suficiente para que una aeronave llegue a un punto en el plano XY. Esto demuestra que si que es posible establecer un piloto automático a partir del análisis experimental.

De todo el proyecto hemos constatado varias cosas:

- Para un mejor análisis del comportamiento de los ángulos, no hay que separarlos en los tres de Euler como hicimos en una primera instancia, sino que hay que separarlos dinámica longitudinal y transversal como se puede observar en las ecuaciones no lineales, ya que no podemos separar la guiñada del alabeo, pero si el cabeceo de las dos anteriores.

- En las prestaciones no ha sido posible controlar la velocidad ni la altitud por una simple razón, el P-Factor que habíamos mencionado desequilibra nuestro avión y no nos permite establecer lo que sería un control real de los elevadores. La solución que se nos ocurre sobre esto para un futuro proyecto, es mantener fijo "a la fuerza", es decir sin un controlador sino controlando directamente la variable, el alabeo del avión para poder analizar el comportamiento del cabeceo como si estuviese en un túnel de viento. Este no era el propósito de este proyecto ya que queríamos diseñar controladores dándole todos los grados de libertad a la aeronave.

- Hemos visto que el simulador de vuelo complementa las ecuaciones no lineales y las vuelve más realistas, ya que si estableciéramos directamente los controladores de estas, no hubiéramos observado los efectos que se crean en realidad a la hora del acoplamiento entre las variables.

- FlightGear ha pasado la prueba de simulador competente con unos modelos dinámicos bastante viables que nos han servido para darle un sentido físico y observable a nuestros controladores, creemos que puede ser una buena herramienta para la validación de ellos. Esto puede ser muy útil para la docencia ya que le da un toque de realismo, muy útil para la formación.

- Por otro lado, la plataforma de Matlab© Simulink© ha respondido correctamente a las prestaciones que se le pedía. Enviando y recibiendo ordenes con un periodo de tiempo fijo, nos ha facilitado el análisis y el modelado.

10. Bibliografía.

[1] http://wiki.flightgear.org/

[2] http://www.flightgear.org/

[3] http://jsbsim.sourceforge.net/JSBSimReferenceManual.pdf

[4] http://ntrs.nasa.gov/

[5] http://cessna.txtav.com/en/piston/cessna-skyhawk

[6] http://es.slideshare.net/junio_oliveira/manual-cessna-172-jornaldoarblogspotcom

[7] http://wiki.flightgear.org/Understanding_Propeller_Torque_and_P-Factor

[8] Xavier Blasco Ferragud : "Control predictivo basado en modelos mediante técnicas de optimización heurística. Aplicación a procesos no lineales y multivariables."

[9] Jose Pedro Magraner: "11889_2015_TEORIA_TEMA_11_BRYAN_EQUATIONS" – Mecánica de vuelo.

[10] Xavier Blasco Ferragud: "Estructuras de control Multi-Bucle para Sistemas Multivariables" – UD3 Control Industrial.

Trabajo Fin de Grado
ETSID
UNIVERSIDAD POLITECNICA DE VALENCIA

DISEÑO Y VALIDACIÓN DE SISTEMAS DE CONTROL PARA AERONAVES BASADO EN LAS HERRAMIENTAS SOFTWARE FLIGHTGEAR Y MATLAB©. APLICACIÓN AL DISEÑO DE PILOTOS AUTOMÁTICOS

2. Manual de programación.

Autor: **Pablo Brusola Fernández-Portolés**
Director de proyecto: **Xavier Blasco Ferragud**
Especialidad: **Ingeniería de Sistemas Automáticos**
Valencia, 5 de Septiembre de 2016

Índice:

Índice de figuras:

1. Introducción

En este documento se va a explicar más detalladamente la programación realizada durante el desarrollo del proyecto "DISEÑO Y VALIDACIÓN DE SISTEMAS DE CONTROL PARA AERONAVES BASADO EN LAS HERRAMIENTAS SOFTWARE FLIGHTGEAR Y MATLAB©. APLICACIÓN AL DISEÑO DE PILOTOS AUTOMÁTICOS".

Para ello no hablaremos de todo el código que hemos ido implementando, sino de la parte que consideramos más útil para comprender el funcionamiento de la plataforma para futuros usos del usuario. Explicaremos la programación en las 4 unidades que hemos implantado este proyecto:

- Unidad 1: Interfaz FlightGear-Matlab©
- Unidad 2: Plataforma para el modelado experimental.
- Unidad 3: Plataforma para el diseño y validación de los controladores.
- Unidad 4: Plataforma para el piloto Automático.

2. Interfaz FlightGear-Matlab

En el documento de la memoria hemos explicado las dos maneras que teníamos de conectar el simulador a nuestro entorno de programación, aquí vamos a tratar el código y las funciones que hemos implementado en Matlab© para poder tener una prospera conectividad con el simulador de vuelo.

2.1. Condiciones iniciales.

Las condiciones iniciales podríamos enviarlas a través de Matlab© (solo en conexión por UDP), pero pensamos que una manera más práctica era usar la interfaz de FLightGear directamente. Para ello creamos lo que parece ser un "Tutorial", aunque en realidad lo único que hace es poner la aeronave en las condiciones iniciales y una vez hecho eso "termina" el Tutorial. Para ello hemos tenido que programar un archivo xml con las condiciones iniciales como muestra la figura 1 de manera general.

En el ejemplo de la figura 2 vemos más precisamente con que atributos tenemos que caracterizar los elementos o variables que queremos darle un valor. Hay algunos valores que se pueden configurar como "preset" que es antes de inicializar, y otros con "set", justo cuando se inicializa la simulación. Gracias a su forma de tutorial, podemos seleccionarlo cada vez que lo deseemos en el menú principal del simulador.

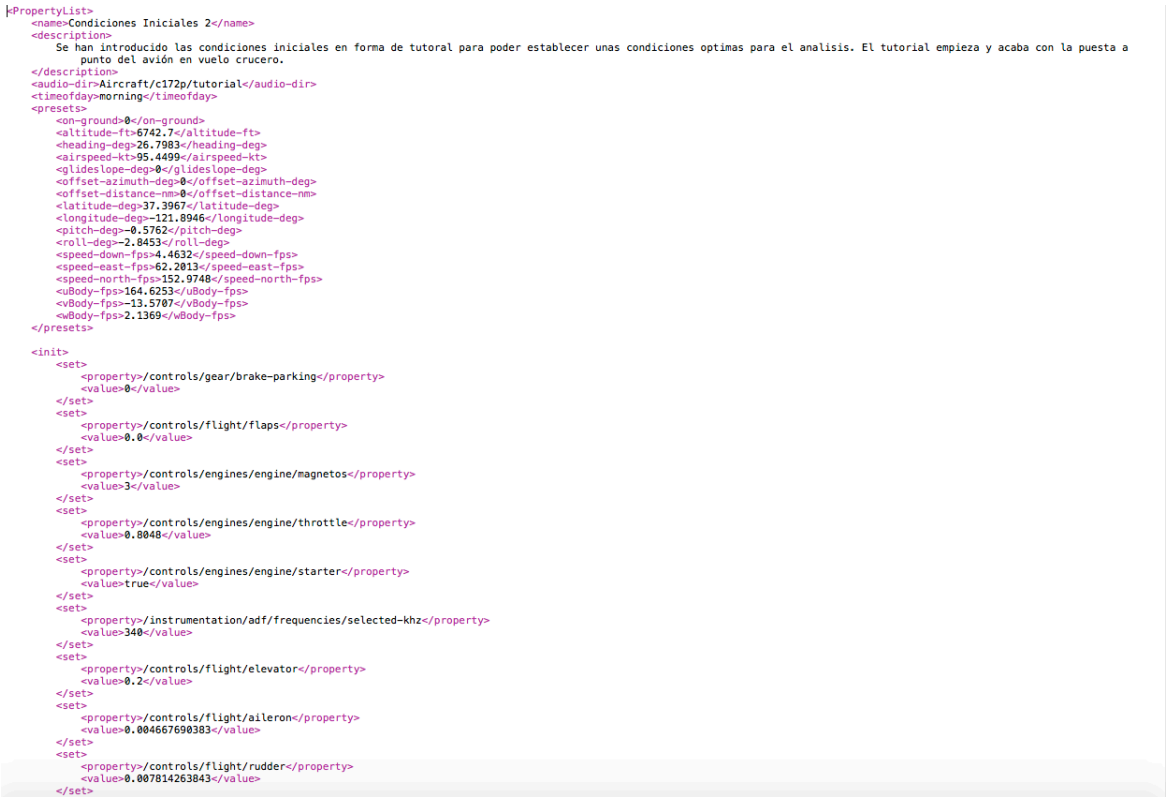


Figura 1: Archivo XML con las condiciones iniciales.

Figura 2: Ejemplo de <presets> y <set>

2.2 Conexión Web Server (http)

Como vimos, esta es una de las dos conexiones que nos disponía a través del navegador una forma sencilla y eficaz de acceder al árbol de propiedades del sistema. Esta interfaz nos permitía tanto leer como modificar las variables del simulador a nuestro antojo. Ahora les mostraremos que funciones hemos implementado para poder tener esta libertad desde Matlab.

```
function [ dato ] = leer_html( url_datos ,var)
%esta función devuelve la variable en forma de double a partir
% de la url donde está situado el dato enviado por http
%url_dato hay que ponerlo en forma de string con llaves (con {' '}).

url=[url_datos var];
str = urlread(url);

for i=1:length(str)
    analizar = str(i:i+6);
    if(analizar=='Value: ')
        n= i+7;
        for j=i+7:length(str)
            analizar2 = str(j:j+1);
            if(analizar2=='</')
               m=j-1;
               break
            end

        end
        dato=str(n:m);
        break
    end

end

dato = str2double(dato);
end
```

Figura 3: Script de la función leer_html.m

Con la función de la figura 3 lo que buscamos es leer desde la url del navegar al cual el navegador está enviando la información el valor del árbol de propiedades que nosotros le indiquemos. La entrada de la función es la url donde se encuentre esa propiedad y el nombre de la variable del árbol de propiedades que queramos recoger. La salida de la función será el valor de dicha propiedad.

Por otro lado, en la figura 4, hemos implementado la función para escribir una variable. Partiendo del mismo principio que la anterior, esta vez se busca mediante la url (que recordamos nos permite modificar un valor) escribir o cambiar el valor de alguna variable en el simulador. Los parámetros de entrada son los mismos, añadiendo el valor a modificar de la variable. No hay salidas para esta función.

```
function write_html( url, propiedad,v )
%En esta función escribiremos en la url los parametros para  modificarlos
%que queremos modificar en el simulador FlightGear.

%Parametros:
%url --> (string) es la url del arbol de propiedades donde esta nuestro
%parametro a controlar
%propiedad --> (string) es la pripiedad especifica dentro de ese arbol que
%nos interesa cambiar el parametro.
%v --> (double) es el valor que vamos a introducir en nuestro parametro


u = [url '?submit=set&' propiedad '%5B0%5D=' num2str(v)];
urlread(u);

end
```

Figura 4: Script de la función write_html.m

Estas dos funciones van a ser el pilar del resto, ya que van a ser las responsables de intercambiar la información con el simulador. A partir de aquí estableceremos funciones solo para los actuadores (alerones, elevadores y timón de cola) como *set_params.m (*figura 5). También crearemos un script para leer todos las variables iniciales que deseemos y la guardemos en una struct de variables iniciales (figura 6)

```
function set_params( host,puerto,aleron,rudder,elevator )
%% Entradas
% TODO VALORES ENTRE -1 y 1
% - aileron (alerones)
% - Rudder (timón de cola)
% - Elevator (timón de profundidad)
%% Datos de conexión
url=['http://',host,':',puerto,'/props/'];
%% Variables a escribir
%% Elementos de controlVV
%En el arbol de propiedades controls/flight/
url_control_vuelo=[url,'controls/flight/'];
write_html(url_control_vuelo,'aileron',aleron);
write_html(url_control_vuelo,'rudder',rudder);
write_html(url_control_vuelo,'elevator',elevator);

end
```

Figura 5: Función set_params.m

```
function [var]=read_ini(host,puerto)

%% Analisis de condiciones de contorno
% Aquí procedemos a hacer un vuelo manual para intentar recoger los datos
% de un vuelo crucero para poder luego hacer el analisis

%% Datos de conexión
url=['http://',host,':',puerto,'/props/'];
%% Variables a analizar
%% Elementos de control
%En el arbol de propiedades controls/flight/
url_control_vuelo=[url,'controls/flight/'];
% - aileron (alerones)
% - Rudder (timón de cola)
% - Elevator (timón de profundidad)
[var.aleron]=leer_html(url_control_vuelo,'aileron');
[var.rudder]=leer_html(url_control_vuelo,'rudder');
[var.elevator]=leer_html(url_control_vuelo,'elevator');

% %En el arbol en controls/engines/engine
 url_control_motor=[url,'controls/engines/engine/'];
%
% % - throttle (palanca de gases)
[var.throttle]=leer_html(url_control_motor,'throttle');

%% Elementos de posición

%En el arbol /position
url_pos=[url,'/position/'];

% - Queremos la altitud de crucero
[var.alt]=leer_html(url_pos,'altitude-ft'); % en pies
[var.lat]=leer_html(url_pos,'latitude-deg');
[var.long]=leer_html(url_pos,'longitude-deg');
%% Velocidades
%en el arbol /velocities
url_vel=[url,'velocities/'];
[var.airspeed]=leer_html(url_vel,'airspeed-kt');
[var.uBody]=leer_html(url_vel,'uBody-fps');
[var.vBody]=leer_html(url_vel,'vBody-fps');
[var.wBody]=leer_html(url_vel,'wBody-fps');
[var.down]=leer_html(url_vel,'speed-down-fps');
[var.east]=leer_html(url_vel,'speed-east-fps');
[var.north]=leer_html(url_vel,'speed-north-fps');
```

Figura 6: Función read_ini.m

Teniendo esto claro, hay que ver como ejecutamos el software para que envie y reciba datos lo más periódico posible y además poder establecer tiempos coherentes con el momento en el que cogemos el dato. Para ello usaremos la función *analizar.m* que nos va a permitir tanto leer como escribir, recogiendo el tiempo de la variable 'elapsed-sec' que es el tiempo del simulador (ver figura 7). Esto tendrá el pequeño error que será lo que tarde la función leer_html en ejecutarse, pero es sencilla y rápida asi que lo obviamos. Con esta función sabremos con bastante exactitud en que instante estamos recogiendo el valor de la variable.

```
%% Lectura de orientación
for n=1:length(var_string)
    [var_r(i,n)]=leer_html(url_orient,var_string{n});
end
time_orient=leer_html(url_time,'elapsed-sec');
%% Lectura de velocidad
[var_a(i,1)]=leer_html(url_V,'airspeed-kt');
time_V=leer_html(url_time,'elapsed-sec');
%% Lectura de altitud
[var_a(i,2)]=leer_html(url_h,'altitude-ft');
time_h=leer_html(url_time,'elapsed-sec');
i=i+1;

if i>length(var_r)
   evalin('base','stop(t)');
end
end
```

Figura 7: Extracto de lectura de variables y tiempos.

Por otro lado tendremos que ocuparnos que esta función se ejecute en un tiempo periódico, y si en teoría la función tarda siempre lo mismo en ejecutarse, conseguiremos una respuesta bastante firme para la recogida de datos. Para ello vamos a usar la función Timer de Matlab© que permite ejecutar funciones en el periodo que tu le mandes (figura 8). La función se terminará cuando las variables ocupen todo el espacio de sus vectores. Tendremos tres matrices de salida:

- var_r: Matriz de tres filas con los ángulos de Euler (roll,pitch,yaw)
- var_C: Matriz de tres filas con los valores de los actuadores (alerones, elevadores, timón de cola)
- var_a: Matriz de dos filas con la velocidad y la altitud.

```
%% Este script lo usaremos para sacar gráficas de los parámetros
clc;clear all;

host='localhost';
puerto='5480';
url=['http://',host,':',puerto,'/props/sim/time/'];
variable='elapsed-sec';
%% Timer para que sea periodico la recogida de datos
i=1;
j=1;
var_r=zeros(800,3);
var_c=var_r;
var_a=zeros(800,2);
t=timer;
set(t,'period',0.2); %perido de ejecucion
set(t,'ExecutionMode','fixedrate'); % ejecucion a un ratio constante fijado por 'period'

t.StartFcn='[t0]=leer_html(url,variable);'; %funcion que se ejecuta al arrancar el timer.
t.TimerFcn='[var_r,time_orient(i),var_c,time_V(i),time_h(i),var_a,t_step(i),i,j]=analizar(i,var_r,j,var_c,var_a);';

start(t); %arranco el timer

load handel
sound(y,Fs)
%para acabar ejecutar:
%stop(t)
```

Figura 8: Script para la lectura y escritura periódica de las variables

2.3 Conexión UDP

Así como la conexión mediante http la hemos hecho con el workspace normal de Matlab©, para la conexión UDP vimos muchos más fácil implementar la interfaz mediante Simulink©. De este modo podríamos controlar y observar las variables de una manera más dinámica. El subsistema que se encarga de controlar todas estas acciones es el que vemos en la figura 9. En el cual se puede observar perfectamente cuales son las variables que escribimos (actuadores) y cuales recibimos para observar su comportamiento.

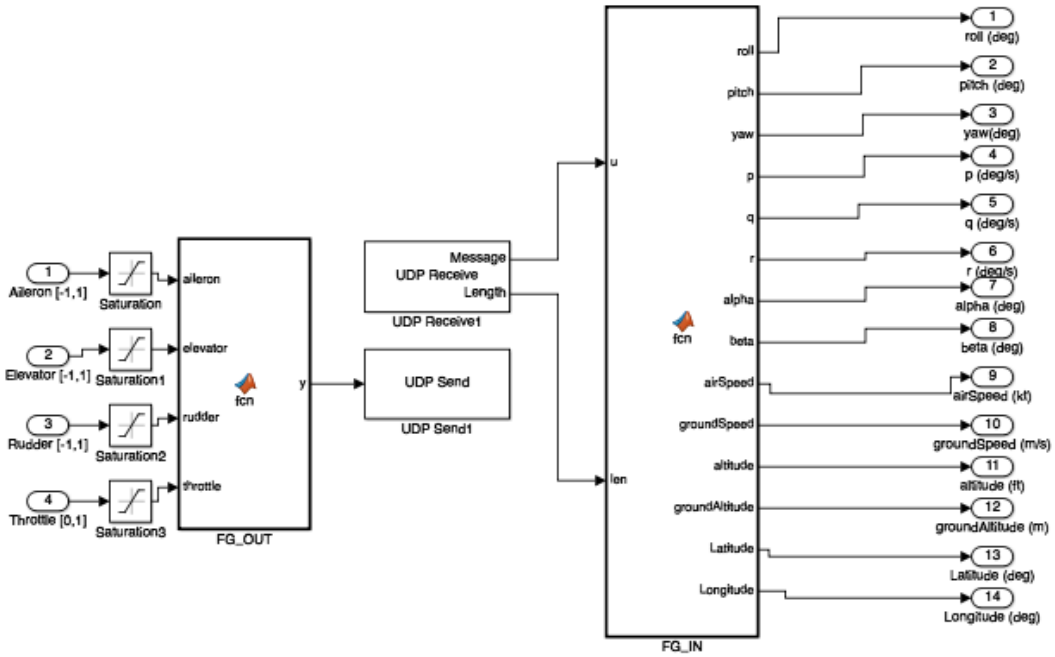


Figura 9: Bloques Simulink© de conexión UDP

De estos bloques cabe destacar la saturación de los actuadores ya que no podemos dejar que envíe un valor que no este comprendido entre el máximo y el mínimo del sistema (de -1 a 1 para todos excepto para la palanca de gases que va de 0 a 1).

También es interesante saber configurar los bloques de envío y recibo de mensajes UDP (que recordamos esta explicado su funcionamiento en la memoria). En la figura 10 podemos ver que configuración le hemos dado nosotros, pero no es la única, todo dependerá de los parámetros que le hayamos dado al simulador a la hora de iniciarlo como indica en la memoria.

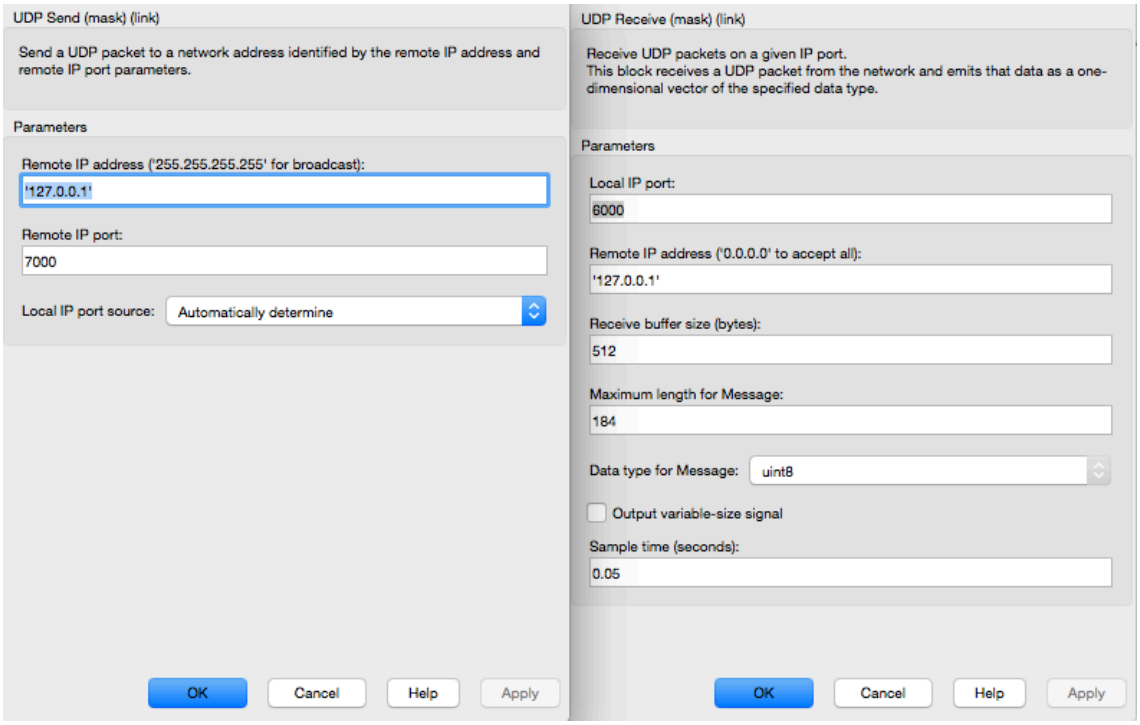


Figura 10: Configuración para la emisión y recepción de mensajes UDP

Como vimos anteriormente, para que el simulador sepa que variables queremos enviar o recibir, hay que escribir una serie de Protocolos, en los cuales especificaremos las variables, los separadores y el lenguaje. Habrá que crear un protocolo input (para enviar desde Simulink©) y un output (para recibirlos en Simulink©). Podemos ver que atributos hay que darles en las figuras 11 y 12.

```
<?xml version="1.0"?>
<PropertyList>
<generic>

   <input>
      <line_separator>newline</line_separator>
      <var_separator>,</var_separator>

      <chunk>
         <name>/controls/flight/aileron</name>
         <node>/controls/flight/aileron</node>
         <type>float</type>
         <format>%+1.5f</format>
       </chunk>

      <chunk>
         <name>/controls/flight/elevator</name>
         <node>/controls/flight/elevator</node>
         <type>float</type>
         <format>%+1.5f</format>
       </chunk>

      <chunk>
         <name>/controls/flight/rudder</name>
         <node>/controls/flight/rudder</node>
         <type>float</type>
         <format>%+1.5f</format>
       </chunk>

      <chunk>
         <name>/controls/engines/engine/throttle</name>
         <node>/controls/engines/engine/throttle</node>
         <type>float</type>
         <format>%+1.5f</format>
       </chunk>

   </input>

</generic>
</PropertyList>
```

Figura 11: Protocolo XML input (actuadores)

```
<?xml version="1.0"?>
<PropertyList>
<generic>
   <output>
      <line_separator>newline</line_separator>
      <var_separator>,</var_separator>

      <chunk>
         <name>/orientation/roll-deg</name>
         <node>/orientation/roll-deg</node>
         <type>float</type>
         <format>%+010.5f</format>
      </chunk>

      <chunk>
         <name>/orientation/pitch-deg</name>
         <node>/orientation/pitch-deg</node>
         <type>float</type>
         <format>%+010.5f</format>
      </chunk>

      <chunk>
         <name>/orientation/heading-deg</name>
         <node>/orientation/heading-deg</node>
         <type>float</type>
         <format>%+010.5f</format>
      </chunk>

      <chunk>
         <name>/orientation/p-body</name>
         <node>/orientation/p-body</node>
         <type>float</type>
         <format>%+012.5f</format>
      </chunk>
```

Figura 12: Protocolo XML output (variables observables)

3. Plataforma de modelado experimental.

3.1 Implementación del modelado.

Después de establecer la plataforma para el envió y recibo de las variables que deseamos del simulador, y después de establecer algunos ensayos de las respuestas de los actuadores de los ángulos de Euler, se ha buscado implementar una plataforma que consiga modelar de una manera dinámica las dichas respuestas para aproximarlas a funciones de transferencia (funciones lineales). Este proceso va a ser mucho más ergonómico que implementar las funciones no lineales directamente (figura 13), y permitirá un mejor diseño de los controladores.

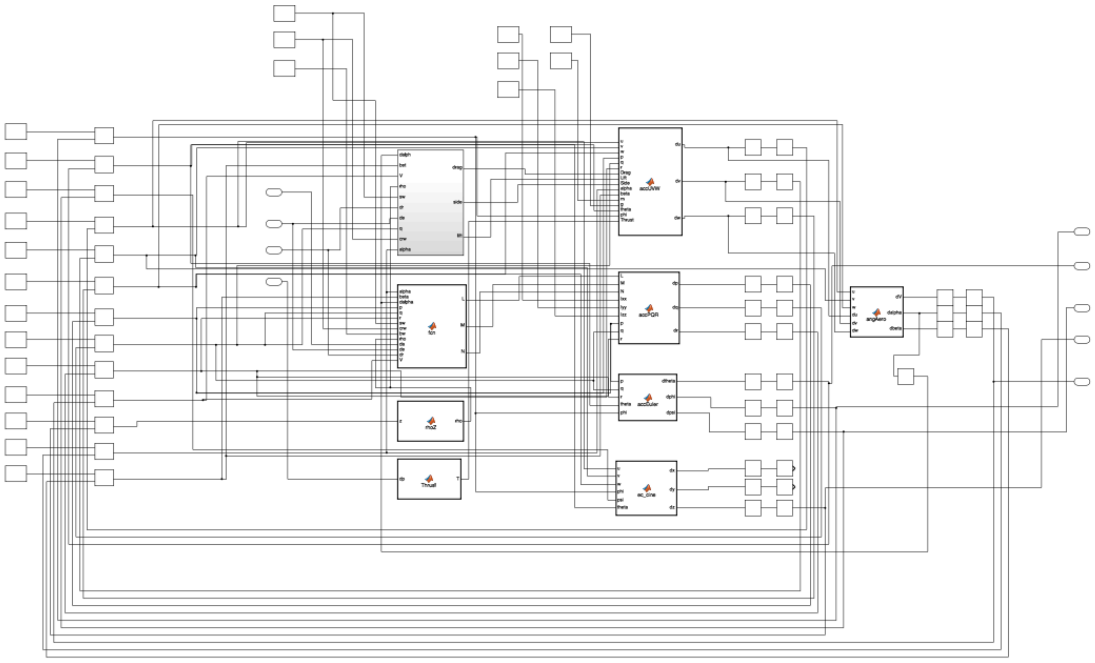


Figura 13: Vista general del diseño del modelo no lineal en Simulink©

Como esta manera de proceder no era viable para las prácticas docentes de la asignatura, buscamos una manera más didáctica y a la vez dinámica de aproximar las funciones. Como hemos mencionado en la memoria, existía la herramienta "ident" para una primera aproximación. Sin embargo, lo que nosotros hemos programado ha sido el modelado mediante algoritmos genéticos.

Ya hemos explicado como estos funcionan, pero cabe destacar que la función motora de dichos algoritmos ha sido facilitada por el departamento de cotrol de la Universidad Politécnica de Valencia, más precisamente del Grupo de Control Predictivo y Optimización Heurística (CPOH) (figura 14). Lo que hace esta función son las iteraciones necesarias para unas condiciones de contorno y una función objetivo dadas. De ahí podremos sacar los mínimos.

Así pues nosotros debemos implementar nuestra función objetivo que será *compara.m* en la cual según los parámetros que le indiquemos nos situará en primer lugar en la función de transferencia deseada (1er orden, 2do, con

integrador, con inercia etc., figura 15) y en segundo lugar inicializará un modelo en Simulink© el cual trazará la función objetivo (figura 16). Este mecanismo ha sido explicado en la memoria en el apartado de algoritmos genéticos asi que no nos repetiremos.

```
function gaDat=ga(g)
%
% Algoritmo genético básico
%
%    gaDat=ga(gaDat)
%    gaDat : estructura de datos utilizada en el algoritmo.
%
% Estructura de datos
% Parametros obligatorios seleccionables por usuario
% gaDat.FieldD=[lb; ub];
% gaDat.Objfun=;
% Parametros seleccionables por usuario que tienen valor por defecto
% gaDat.MAXGEN={NVAR*5+5};
% gaDat.NIND={NVAR*20} ;
% gaDat.alfa={0};
% gaDat.Pc={0.9};
% gaDat.Pm={0.1};
% gaDat.ObjfunPar={[]};
% gaDat.indini={[]};
%
% Grupo de Control Predictivo y Optimización Heurística - CPOH
% Universidad Politécnica de Valencia.
% http://cpoh.upv.es
% (c) CPOH  1995 - 2012
```

Figura 14: Algoritmo genético básico proporcionado por el CPOH

```
function [ J ] = compara(theta,param)

if strcmp(param.modelo,'1er orden')==1
K=theta(1);
tau=theta(2);
num=K;
den=[tau 1];
indice=2;
elseif strcmp(param.modelo,'2do orden')==1
K=theta(1);
wn=theta(2);
tau=theta(3);
num=K*wn^2;
den=[1 2*tau*wn wn^2];
indice=3;
if param.zero>0
    for i=1:param.zero
       num=conv(num,[theta(indice+i) 1]);
    end
    indice=3+param.zero;
end
elseif strcmp(param.modelo,'3er orden')==1
K=theta(1);
wn=theta(2);
tau=theta(3);
z=theta(4);
t=theta(5);
num=K*wn^2*[z 1];
den=conv([1 2*tau*wn wn^2],[t 1]);
indice=5;
end
if param.inercia==1
   T=theta(indice+1);
   den=conv(den,[T 1]);
end
if param.integrator==1
   den=[den 0];
end
if param.delay==1
   td=theta(end);
else
    td=0;
end


experimento=param.experimento;

yexp=[experimento(:,1) experimento(:,3)];
uexp=[experimento(:,1) experimento(:,2)];

texp=experimento(end,1)-experimento(1,1);

    assignin('base','td',td);
    assignin('base','num',num);
    assignin('base','den',den);
    assignin('base','Yexp',yexp);
    assignin('base','Uexp',uexp);
    assignin('base','texp',texp);

a = sim(param.simulink,'SimulationMode','normal');
b = a.get('simout');
J = sum(b.Data.^2);
%J=sum(b)/length(b)+max(b);

end
```

Figura 15: Función objetivo para algoritmos genéticos

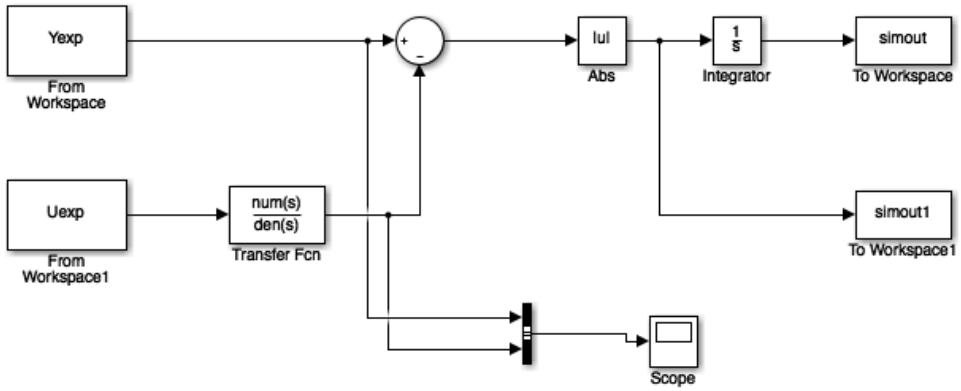


Figura 16: Simulink© ejecutado en la función objetivo del error.

Una vez tenemos clara la función objetivo y los parámetros que van a servir como individuos (parámetro theta en figura 15), es el momento de inicializar los algoritmos genéticos para encontrar el mínimo más óptimo para nuestra función. Introduciremos los limites superior e inferior de cada variable y los parámetros de la función objetivo para conseguir la mejor aproximación posible. Un script ejemplo para el modelador del roll se puede observar en la figura 17.

```
%% Modelado del roll
%Datos experimentales
rollident=rollident-rollident(1);
 parAlgoritmo.experimento=[timeIdent Ue rollident];
 parAlgoritmo.simulink='f_error';
 parAlgoritmo.modelo='1er orden';
 parAlgoritmo.integrator=false;
 parAlgoritmo.delay=false;
 parAlgoritmo.inercia=false;

 gaDat.FieldD=[0 0.1;
               4000  500];     % Cod. Real: Limites Inferior y superior de cada variable
 gaDat.Objfun='compara'; % Función a minimizar
 gaDat.ObjfunPar=parAlgoritmo; % parametros adicionales funciona a minimizar

 gaDat=ga(gaDat);
 solucion=gaDat.xmin;

%Validación del modelo
dataroll=iddata(rollident,Ue,0.05);
s=tf('s');
fdtroll=solucion(1)/((solucion(2)*s+1));
compare(dataroll,fdtroll,opt);
```

Figura 17: Ejemplo del modelado del roll con algoritmos genéticos

En este ejemplo no sale pero viendo los parámetros que podemos introducirle a nuestro algoritmo genético, podemos modificar probabilidades, número de individuos iniciales, número de iteraciones que se van a hacer. Sin embargo hay algunas configuraciones por defecto.

3.2 Validación del modelado.

Por último, para comprobar que nuestro modelo es correcto, se ha establecido una interfaz Simulink©, que compare los ángulos de Euler que nos da el simulador de vuelo, con los ángulos que nos dan las funciones de transferencia que hemos encontrado mediante los algoritmos genéticos. Para ello se establecen exactamente las mismas entradas para los dos y se observa su comportamiento como se ha indicado en la memoria. La interfaz simulink que se ha implementado es la que aparece en la figura 18.

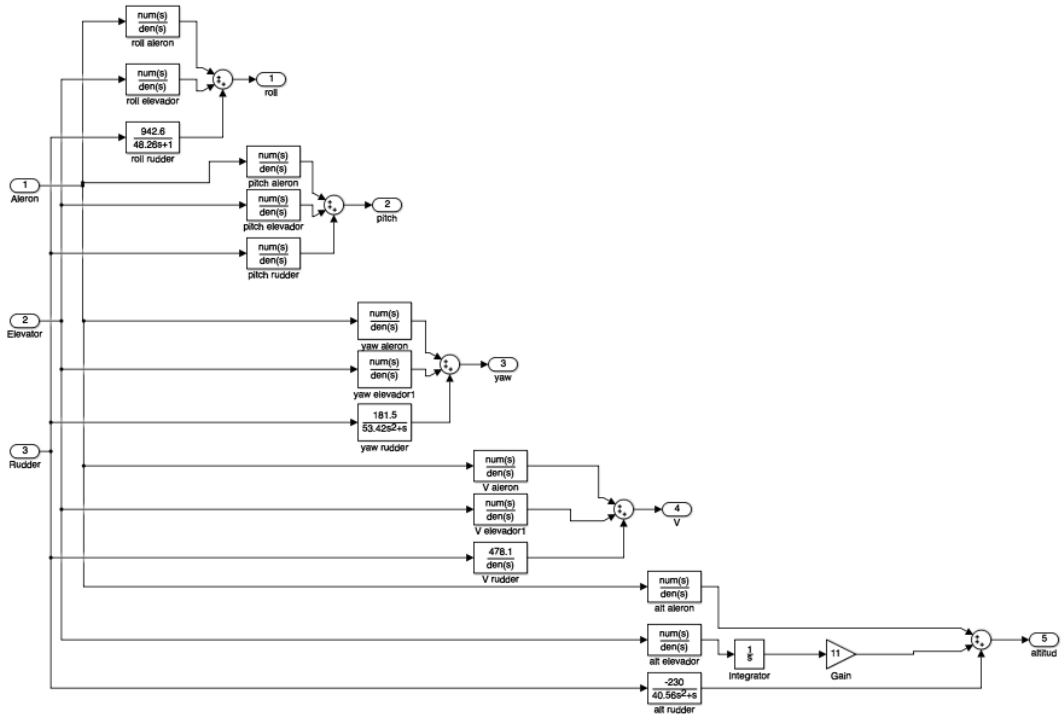


Figura 18: Interfaz Simulink© del sistema de funciones de transferencia.

4. Plataforma del diseño y validación de controladores.

En este apartado posiblemente sea la parte más sencilla del código, pero creemos que es necesario ver como se ha implementado. Es sencilla ya que la mayoría de este viene por defecto ya que usamos la herramienta de Matlab© *rtool*, que nos permite un gran análisis del lugar de las raíces y poder jugar mucho con el diseño de los controladores, ya sea viendo de una manera más dinámica las especificaciones o variando las posibilidades de control.

4.1 Diseño de controladores.

Para el diseño de los controladores como bien se especifica en la memoria, se ha usado el calculo de la matriz RGA para establecer el emparejamiento, que luego veremos que no es plausible. Vemos en la figura 19 como hemos metido las funciones de transferencia en una sola matriz, y con una simple operación hemos calculado el RGA.

```
G(1,1)=tf(numrolla,denrolla);
G(2,1)=tf(numpitcha,denpitcha);
G(3,1)=tf(numyawa1,denyawa1);

G(1,2)=tf(numrolle,denrolle);
G(2,2)=tf(numpitche,denpitche);
G(3,2)=tf(numyawe1,denyawe1);

G(1,3)=tf(numrollr,denrollr);
G(2,3)=tf(numpitchr,denpitchr);
G(3,3)=tf(numyawr1,denyawr1);

%Es la K equivalente
K=dcgain(G)
RGA=K.*inv(K)'
```

Figura 19: Calculo de la matriz RGA

Todo lo que necesitamos saber acerca de los resultados y razonamientos de esta matriz, esta situado en la memoria.

4.2 Validación de controladores.

Por otro lado, siempre es necesario antes de la implementación de un controlador, validar de una forma lineal su diseño para saber si la respuesta que queremos va a ser la deseada.

Así pues nosotros hemos decidido hacerla mediante la plataforma simulink© ya que de una manera muy sencilla y sobre todo visible, podemos deducir rápidamente si nuestro controlador va a ser lo suficiente fiable o no. Dicha estructura de bloques se ve reflejada en la figura 20.

Figura 20: Bloques Simulink© de validación de controladores.

5. Plataforma para el piloto automático.

Esta plataforma ha sido desarrollada de nuevo en Simulink©, es el entorno más favorable para este tipo de controladores. Lo que hemos usado de base como siempre para enviar y recibir información del simulador, es lo que hemos descrito en el apartado 2. Pero le hemos añadido los controladores y funciones necesarias para poder llevar a nuestro avión a cualquier waypoint o incluso crear una ruta de ellos (figura 21)

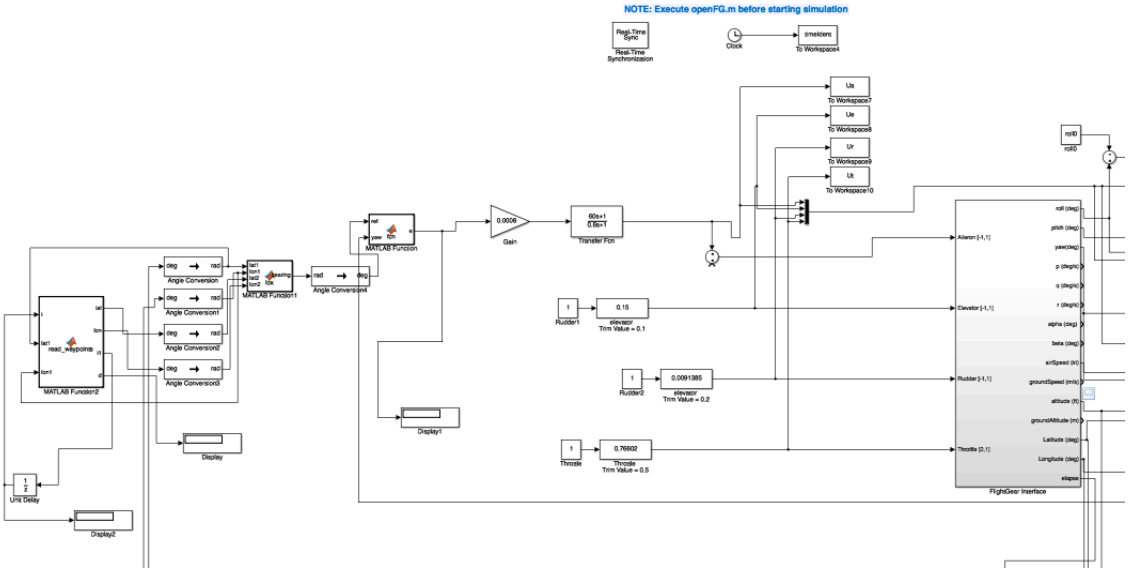


Figura 21: Vista general del piloto automático basado en Simulink©

5.1 Controlador del heading.

Como podemos ver, lo que coordina el giro para establecer un rumbo correcto es el controlador (figura 22). Como se ha explicado, como te ha explicado en la memoria se trata de un PD.

Para que este funcione correctamente se le tiene que aplicar como entrada el error entre la referencia y la señal del rumbo recibida del simulador.

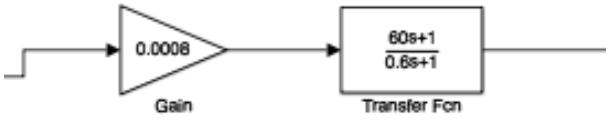


Figura 22: Bloques del controlador de Heading.

5.2 Referencia del rumbo.

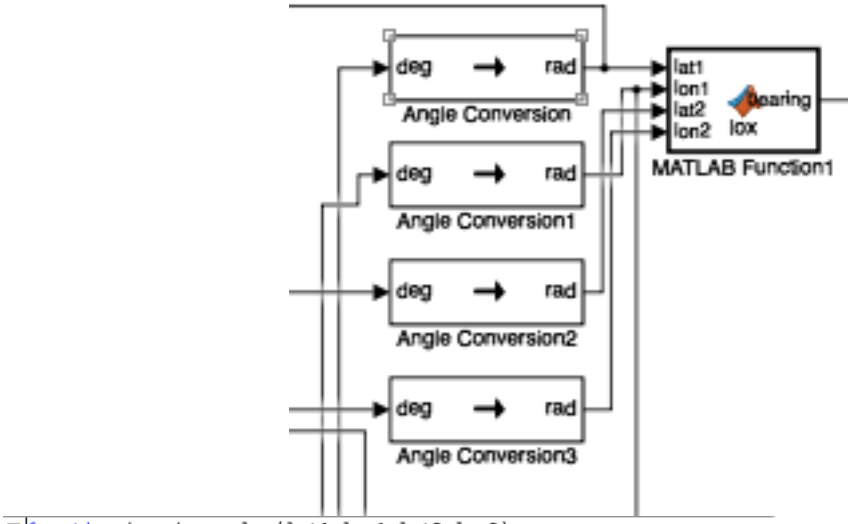
Lo que hemos buscado en este bloque (figura 23), es que al controlador le llegue la referencia de un giro poco brusco como explicamos en la memoria, y además el giro más corto posible. Es decir que cuando tu estas en 30 grados y la referencia en 330, el controlador le va a decir que aumente hasta 330. Sin embargo, sería más productivo si girase -60 en sentido contrario. De eso se ocupa la función de este bloque. En este ejemplo, hemos puesto la sensibilidad en 5 grados, por lo que el error siempre le dirá que esta a 5 grados a la izquierda o a la derecha, para no provocar giros bruscos hasta que se estabilice y se alinee con el rumbo que debe.


```
function e = fcn(ref,yaw)
%#codegen
sensibilidad=5;
if ref<0
    ref=ref+360;
end
if ref>yaw
   if ref-yaw<180
       if ref-yaw>sensibilidad
           e=sensibilidad;
       else
           e=ref-yaw;
       end
   else
       if yaw-(ref-360)>sensibilidad
           e=-sensibilidad;
       else
           e=-(yaw-(ref-360));
       end
   end
else
    if yaw-ref<180
       if yaw-ref>sensibilidad
           e=-sensibilidad;
       else
           e=-(yaw-ref);
       end
   else
       if ref-(yaw-360)>sensibilidad
           e=sensibilidad;
       else
           e=(ref-(yaw-360));
       end
   end
end
```

Figura 23: Bloque y función del error del controlador.

5.3 Cálculo del rumbo inicial.

Como dijimos en la memoria, sabiendo la posición actual de nuestra aeronave y la posición final a la que queremos que llegue, podemos calcular el rumbo inicial ortodrómico, que si vamos actualizando a cada instante de tiempo, conseguimos una navegación loxodrómica. Pues eso mismo hace el bloque de la función *bearing.* Usando la ecuación del rumbo inicial, obtenemos la referencia que será la entrada del bloque anterior (Figura 24) .




```
function bearing = lox(lat1,lon1,lat2,lon2)
%#codegen
if lat2==0 && lon2==0
    bearing=0;
else
bearing=atan2(sin(lon2-lon1)*cos(lat2),cos(lat1)*sin(lat2)-sin(lat1)*cos(lat2)*cos(lon2-lon1));
end
```

Figura 24: Bloque y script para el calculo de Bearing.

5.4 Lectura de Waypoints.

En este bloque lo que se busca es introducir una serie de waypoints por los que la aeronave tendrá que seguir una ruta. Es lo que va a probar que nuestro piloto automático funciona perfectamente.

Lo que le diremos al programa es calculando la distancia al waypoint destino, que cuando este a menos de 1km de el, ya cambie al siguiente waypoint y comience a girar. Esta sensibilidad se puede cambiar pero 1 km nos parecía correcto. La manera de calcular las distancias aparece en la memoria y los waypoints elegidos han sido a conciencia para estén más o menos cerca del punto inicial de la aeronave. También se han escogido con virajes más y menos pronunciados para ver cual era su efecto. Resultados en la memoria.

El resultado de este bloque y del script de la función se puede observar en la figura 25 y 26.

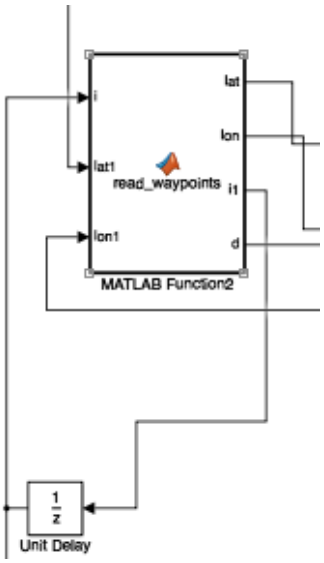


Figura 25: Bloque de lectura de waypoints

```
function [lat,lon,i1,d] = read_waypoints(i,lat1,lon1)
%#codegen
%% Coordenadas
w=[ 37+22.48/60 -121-56.68/60;
    37.4325 -122.0577;
    37+23.55/60 -122-16.88/60 ;
    37+22.48/60 -121-56.68/60];
if i>length(w)-1
    lat=0;
    lon=0;
    i1=i;
    d=0;
else

R=6371000; %radio de la tierra en metros
lat=w(i+1,1);
lon=w(i+1,2);
lat2=lat*pi/180;
lon2=lon*pi/180;
%% distancia haversine
a=sin((lat2-lat1)/2)^2+cos(lat1)*cos(lat2)*sin((lon2-lon1)/2)^2;
c=2*atan2(sqrt(a),sqrt(1-a));
d=R*c;

if d<1000
    i1=i+1;
else
    i1=i;
end
end
```

Figura 26: Script de lectura de Waypoints

Trabajo Fin de Grado
ETSID
UNIVERSIDAD POLITECNICA DE VALENCIA

DISEÑO Y VALIDACIÓN DE SISTEMAS DE CONTROL PARA AERONAVES BASADO EN LAS HERRAMIENTAS SOFTWARE FLIGHTGEAR Y MATLAB©. APLICACIÓN AL DISEÑO DE PILOTOS AUTOMÁTICOS

3. Presupuesto

Autor: **Pablo Brusola Fernández-Portolés**
Director de proyecto: **Xavier Blasco Ferragud**
Especialidad: **Ingeniería de Sistemas Automáticos**
Valencia, 5 de Septiembre de 2016

Índice:

1. Introducción.

Este documento tiene como objetivo detallar la información del corte de la ejecución del proyecto "DISEÑO Y VALIDACIÓN DE SISTEMAS DE CONTROL PARA AERONAVES BASADO EN LAS HERRAMIENTAS SOFTWARE FLIGHTGEAR Y MATLAB©. APLICACIÓN AL DISEÑO DE PILOTOS AUTOMÁTICOS".

En primer lugar vamos a definir los precios de la mano de obra (ya que en este campo existe mucho debate) y materiales que hemos usado para este proyecto. Luego se detallará cada parte que hemos realizado del proyecto dividiéndolas en 4 partes:

- Programación de la interfaz Matlab©-FlightGear
- Programación para la Plataforma del modelado
- Programación para la Plataforma del diseño de controladores.
- Programación para la Plataforma del piloto automático.

2. Mano de obra.

En este apartado puede existir mucho debate ya que en la situación que ocupa actualmente nuestro país tenemos que plantearnos si tenemos que poner un salario acorde con lo que cobra un ingeniero hoy en día en España, o calcular cuanto creemos que deberíamos cobrar como profesionales de la ingeniería que estamos suponiendo que somos. El colegio oficial de Ingenieros Técnicos Industriales de Bizkaia sacó en 2014 un articulo que se titulaba "¿Sabes calcular el precio por hora de tu trabajo?" (http://www.coitibi.net/ventanilla-unica/noticias/sabes-calcular-precio-por-hora-de-tu-trabajo) , en el que calculaban el precio por ahora que debía cobrar un trabajo para que no se limitase solo a cubrir las demandas de un proyecto.

En el mencionan que "no hay que olvidar que si se cobra poco, no faltará el trabajo pero siempre faltará dinero y no es tan importante trabajar, sino que nuestro trabajo nos permita vivir". Así pues proceden a hacer el cálculo de la siguiente manera:

Calculo de horas anuales:
40 horas semanales x 52 semana=2080 horas al año
21 días de vacaciones y 14 de festivos =280 horas/año

Total 1800 horas/año

Si fuésemos autónomos, deberíamos estimar las horas facturables ya que se ocupa mucho tiempo en búsquedas de información, consultas, formación, actividades

comerciales etc. Vamos a suponer que estamos para una gran empresa y las horas facturables son el 100% de las horas.

Calculo del ratio precio/horas base al año
Desde el principio hemos puesto un salario de 50.000€ netos al año.
Entre 1800 horas facturables al año obtenemos.

Total: 28€/hora

Gastos.
Según el articulo:
http://albertojoven.blogcanalprofesional.es/cuantopagaempresanomina/
La empresa aporta a la seguridad social un 30.90%
Además el trabajador de su nómina aporta un 6.35%

Total: 37.25% de impuestos

Si lo restamos de nuestro salario nos quedaría que deberíamos cobrar:

50 000*0.3725=18 625€
18 625€ / 1800 horas trabajadas= 11€/hora

Salario total: 40€/hora

Así que este sería nuestro mínimo, ya que hemos puesto que el 100% de nuestras horas son facturables. Así que nos quedamos en este mínimo que como se puede observar, se aleja mucho de la realidad.

Resumen:

Mano de obra				
Salario anual neto	Horas Laborables anuales	Ratio base al año	Gastos de seguridad social 37.25%	Salario final por hora de trabajo
50000€	1800 horas	28€/hora	18 625€	40€/hora

3. Precios de los materiales.

Aquí presentamos los precios de los materiales y licencias que hemos necesitado para este proyecto.

Material	Coste/Unidad	Periodo de utilización	Coste/hora
Mac Mini Core i5 de doble núcleo a 1,4 GHz	549€	3 años/5400 horas	0.10€/hora
2xMonitor LED de 20" LG Dual Smart Solution	2x90€	4 años/7200 horas	0.02€/hora
Licencia de software FlightGear	0€	-	0€
Licencia de software Matlab© for Academic use	500€	1 año/1800 horas	0.28€/hora
Subtotal			0.4€/hora

Como podemos ver en este proyecto ha habido dos ahorros importantes.

Ahorros:

- Simulador de vuelo X-Plane → 60€/ordenador
- Aerospace $Toolbox^{TM}$ → 200€/ordenador

Esto para un ordenador individual no es mucho gasto, pero en la sala pueden haber muchos ordenadores y estos gastos incrementarían notablemente.

4. Precios por unidades de obra

En este apartado vamos a indicar el tiempo que hemos invertido por cada tarea dentro de cada una de las unidades de obra que hemos puesto en la introducción y desglosaremos en base al precio de la mano de obra, el precio por tarea en este proyecto.

4.1 Unidad 1: Programación de la interfaz FlightGear-Matlab©

Tarea	Días	Horas	Mano de obra	Importe total
Estudio del funcionamiento del simulador	3	24	40	960€
Plataforma de conexión vía web server	2	16	40	640€
Plataforma Simulink© de conexión vía UDP	2	16	40	640€
Programación de protocolos del simulador	3	24	40	960€
Total	10	80	40	3200€

4.2 Unidad 2: Programación para la plataforma del modelado

Tarea	Días	Horas	Mano de obra	Importe total
Desarrollo del modelo no lineal	5	40	40	1600€
Primera aproximación con "ident"	2	16	40	640€
Modelado con algoritmos genéticos	4	32	40	1280€
Plataforma Simulink© de validación de modelos	2	16	40	640€
Total	13	102	40	4080€

4.3 Unidad 3: Programación para la plataforma del diseño de controladores

Tarea	Días	Horas	Mano de obra	Importe total
Especificaciones de los controladores	1	8	40	320€
Emparejamientos	3	24	40	960€
Diseño de controladores *rltool*	4	32	40	1280
Validación de controladores Simulink©	1	8	40	320€
Total	9	72	40	2880€

4.4 Unidad 4: Programación para la plataforma de piloto automático

Tarea	Días	Horas	Mano de obra	Importe total
Programación cálculo de rumbo	1	8	40	320€
Programación giro más corto	2	16	40	640€
Plataforma simulink© de controlador piloto automático	3	24	40	960€
Adquisición de waypoints	1	8	40	320€
Total	7	56	40	2240€

5. Presupuesto Total

Asunto	Horas	Coste/Hora	Coste total
UNIDADES			
Unidad 1	80	40	3200€
Unidad 2	102	40	4080€
Unidad 3	72	40	2880€
Unidad 4	56	40	2240€
Materiales			
Mac Mini Core i5 de doble núcleo a 1,4 GHz	310	0.10€	31€
2xMonitor LED de 20" LG Dual Smart Solution	310	0.02€	6.2€
Licencia de software FlightGear	310	0€	0€
Licencia de software Matlab© for academic use	310	0.28€	86.8€
TOTALIDAD			
Coste total			**12 524€**

Vemos que el coste de este proyecto sería de 12 524€, y ocuparía en total un tiempo de 310 horas de trabajo. Observamos que en los materiales es donde más dinero hemos ahorrado ya que el propósito principal era ese: establecerlo en las aulas docentes en varios equipos, lo que puede llegar a costar mucho dinero.