Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Guiado automático de una aeronave B737-800. Fase “en ruta”.

Trabajo de fin de grado de Ingeniería Aeroespacial 2019

Autor: Carlos Escrig Beltrán

Tutor: Juan Antonio Vila Carbó

Agradecimientos

A D. Joan Vila Carbó, por su tutela y ayuda.

A mi familia, a Rodrigo y mi madre por estar cuando se les necesita,

y al resto por estar y no molestar.

Gracias a todos.

Contents

Índice de figuras

Índice de Tablas

1. Objetivos

El objetivo principal de este trabajo es el diseño e implementación de un sistema de guiado para una aeronave Boeing 737-800. Este sistema comprende el guiado vertical y horizontal de la aeronave y se desarrolla sobre un piloto automático estandarizado a aeronaves aviación general con un solo motor. Este sistema será probado sobre un simulador de vuelo profesional, en este caso el simulador *XPlane 10©.*

A través de este trabajo se pretende aumentar la formación del grado, destacando los siguientes aspectos: sistemas de aviónica, gestión del espacio aéreo, fundamentos del vuelo y programación orientada a objetos. Aspectos fundamentales para el desarrollo del sistema de guiado.

A su vez el empleo del software de simulación de vuelo permite que el sistema pueda funcionar en un entorno simulado, generando respuestas similares a las esperadas en una aeronave real, por lo que el sistema adquiere un aspecto práctico que permite su posterior implementación o desarrollo.

2. Introducción

Para poder entender el concepto de navegación aérea es necesario remontarse a la navegación convencional, es decir, a la navegación marítima. Los marineros de antaño empleaban técnicas similares a las actuales para determinar su posición y establecer el rumbo a seguir. La base de aquella navegación se basaba en encontrar puntos de referencia y saber situarse con respecto a ellos, dando lugar a la navegación costera y a la navegación astronómica. Sea un faro en un cabo, la estrella polar o los actuales *waypoints*, hoy en día seguimos empleando puntos de referencia para ubicarnos en el globo.

La navegación aérea actual, en fase “en ruta”, se basa en el seguimiento de estos *waypoints*, con los que podemos establecer un camino entre dos puntos. La precisión y rigurosidad con la que se siguen estos waypoints son factores determinantes en la eficiencia y seguridad del tráfico aéreo, por lo que resulta crucial el desarrollo de sistemas de aviónica que ofrezcan una mejora en la navegación.

Un ejemplo de un sistema de aviónica orientado a la navegación es el piloto automático, responsable de mantener la estabilidad y la orientación correcta durante el vuelo. Los pilotos automáticos primigenios conectaban hidráulicamente los instrumentos de vuelo con las superficies de control de la aeronave, manteniendo así la altitud y la dirección deseada. En la actualidad, estos sistemas se han desarrollado hasta permitir al avión aterrizar casi de manera automática en aeródromos que cuenten con ILS CAT IIIc.

Otro punto a destacar en la evolución de estos sistemas de navegación son los sistemas de guiado, donde los primeros pilotos automáticos no disponían de capacidad para cambiar de forma automática de rumbo al llegar al *waypoint* deseado. Esto suponía que era el piloto el encargado de establecer de manera manual el rumbo al siguiente punto de la ruta, lo que incrementa enormemente el rango de seguridad alrededor de la aeronave. Es por esto por lo que surgen los sistemas de autoguiado, es decir, el guiado de la aeronave sin persona alguna realizando acción sobre el sistema.

3. Estado del arte

3.1. Estado General

En la actualidad existen diferentes formas de implementar un sistema de guiado. Desde los sistemas hidráulicos simples hasta complejos sistemas electrónicos que integran múltiples sensores y actuadores de la aeronave. El uso de sistemas hidráulicos está en desuso en la aviación comercial, por lo que los sistemas dominantes son los sistemas electrónicos que implementan diversos algoritmos para el guiado y el control de la aeronave.

No obstante, todos los sistemas actuales están compuestos de la siguiente manera:

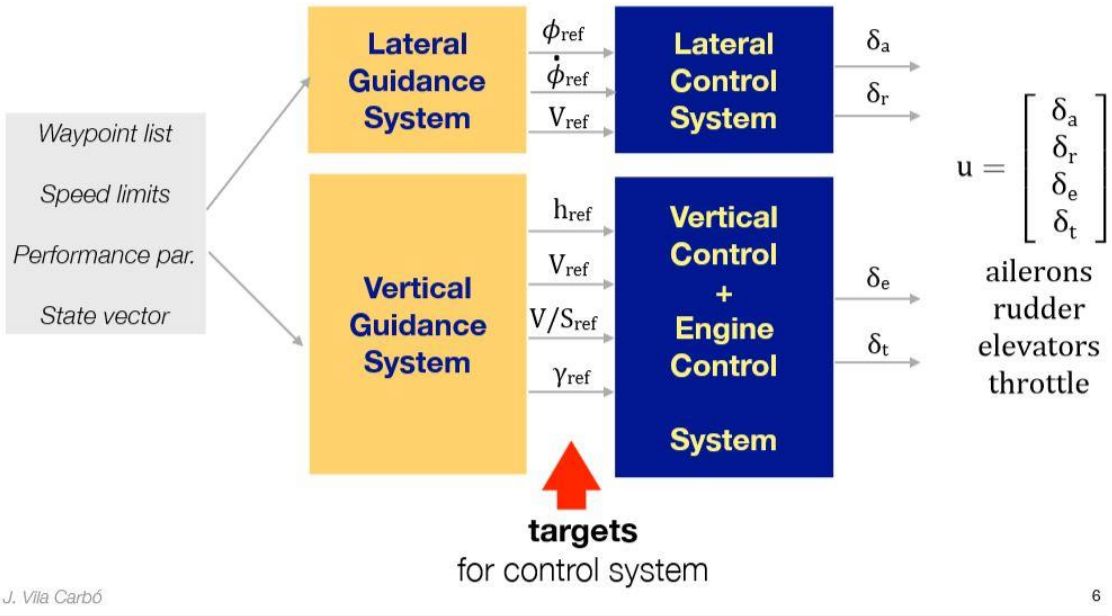

Ilustración 1-Estructura sistema de guiado[7]

Esta estructura describe de manera sencilla el funcionamiento de los sistemas de guiado, los cuales obtienen valores de referencia a partir de datos obtenidos por sensores o programados en la ruta. Estos valores son empleados por los sistemas de control para computar los controles sobre la aeronave, en el caso de la imagen: alerones, timón, elevadores y palanca de gases. Por conveniencia, se tiende a interpretar un sistema de guiado y control como dos sistemas independientes para el guiado vertical y horizontal. Sin embargo, en la realidad, ambos se implementan dentro del mismo sistema de guiado, existiendo un único sistema responsable del guiado vertical y horizontal.

3.2. Guiado Horizontal

A continuación, se describen dos métodos para implementar un guiado horizontal.

3.2.1. Método de Randy Walter

El primer método consiste en calcular el error en el curso de la aeronave y contar con la distancia a la ruta deseada.

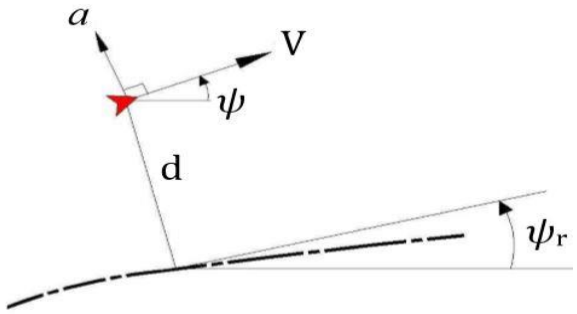

Ilustración 2-Método de Randy Walter para el guiado horizontal [7]

El objetivo del sistema de guiado es el de reducir la distancia (d) y el error en el rumbo (ψ_r-ψ) lo máximo posible. Para ello se computa la aceleración lateral necesaria para corregir estos errores como la suma aritmética del error en el rumbo y el error en la posición:

$$a = k_1 \cdot (\psi_r - \psi) + k_2 \cdot d$$

Este sistema, descrito en [1] y [4] requiere distintos cómputos para su completa realización, entre ellos:

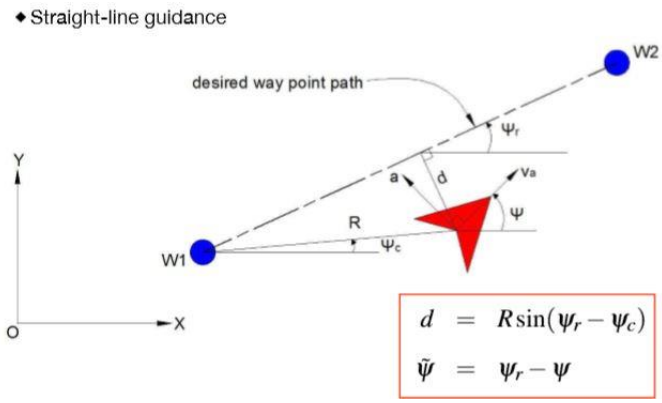

Ilustración 3-Guiado entre Waypoints [7]

Esta figura describe el cálculo de las variables para una traza recta entre dos waypoints.

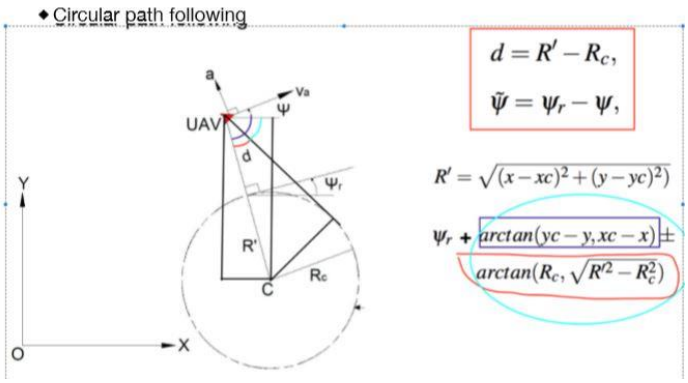

Ilustración 4-Guiado para tramo circular [7]

La figura anterior describe la obtención de las variables necesarias para el guiado en una traza circular.

Por último, se describe la transición entre waypoints de la ruta, ya que se requiere una activación del siguiente tramo. Esta operación requiere un cómputo de una distancia a la que se precisa que se inicie el virado para poder realizar una transición "flyby" entre dos tramos de una ruta.

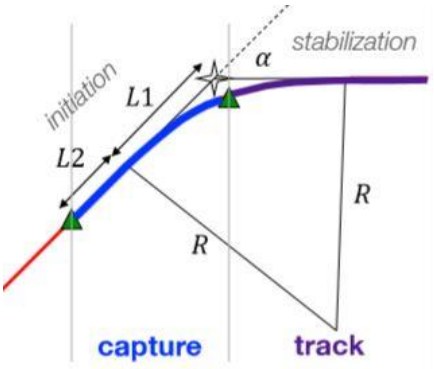

Ilustración 5-Transición entre waypoints [7]

El criterio utilizado para el inicio del viraje para realizar la transición acorde a la figura 5 es:

$$L = L_1 + L_2$$

$$L_1 = R \cdot \tan(\alpha/2)$$

L_2 es una variable que debe ser seleccionada acorde a la velocidad de reacción del alabeo de la aeronave. Esta variable se puede aproximar de la siguiente manera, multiplicando la velocidad de la aeronave por una constante *c* que se acorde a reacción del alabeo de la aeronave.

$$L_2 = c \cdot V$$

$$R = \frac{V^2}{g \cdot \tan(\varphi_{nominal})}$$

Donde:

$$V = Velocidad\ de\ la\ aeronave$$

$$g = aceleración\ gravitatoria$$

$$\varphi_{nominal} = ángulo\ de\ alabeo\ nominal\ de\ viraje$$

3.2.2. Algoritmo de la Zanahoria

El siguiente algoritmo se define como “el algoritmo de la zanahoria” descrito por Ducard GJJ. en “Fault-tolerant Flight Control and Guidance Systems”. Este sistema de guiado se caracteriza porque la aeronave sigue un punto móvil (zanahoria) de la ruta asignada.

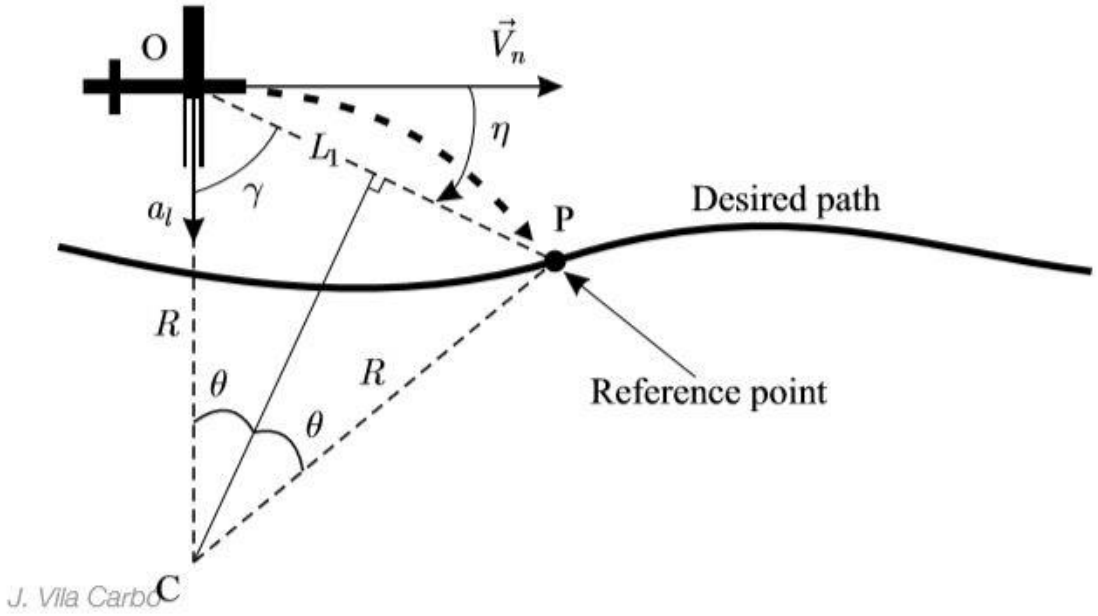

Ilustración 6-Algoritmo de la zanahoria [7]

Tal y como se muestra en la imagen, la aeronave sigue un punto de la ruta a una distancia L_1. Este punto P es la intersección del arco de radio R centrado en C con el camino deseado y η es el ángulo entre el vector velocidad de la aeronave y el punto P. Empleando estos datos se puede computar la aceleración lateral necesaria para guiar el avión al punto P.

$$a_l = \frac{V_n^2}{R}$$

Por geometría se obtiene:

$$\eta = \theta$$

$$L_1 = 2 \cdot R \cdot \sin(\eta)$$

Por lo tanto:

$$a_l = \frac{2 \cdot V_n^2}{L_1} \cdot \sin(\eta)$$

Una vez conocida la aceleración lateral necesaria para guiar el avión al punto P, se puede computar la acción necesaria para realizar el viraje mediante el cálculo del alabeo necesario para dicha acción:

$$\varphi_{acción} = \frac{2 \cdot V_n^2}{L_1 \cdot g} \cdot \sin(\eta)$$

Este algoritmo se puede aproximar de la siguiente manera para valores pequeños de η.

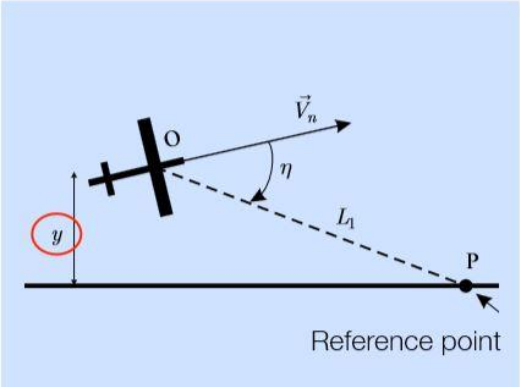

Ilustración 7-Aproximación algoritmo de la zanahoria [7]

En términos del error en la traza y:

$$a_l = \frac{2 \cdot V_n^2}{L_1} \cdot \sin(\eta) \approx 2 \cdot \frac{V_n}{L_1}(y' + \frac{V_n}{L_1} \cdot y)$$

Esta ley de guiado tiene un comportamiento dinámico que puede ser modelado por un controlador PD, en el que la proporción V_n/L_1 es la ganancia del controlador.

En general, este sistema de guiado resulta muy conveniente para tramos circulares ya que la aceleración lateral computada coincide con la aceleración centrípeta asociada a la traza de la ruta circular.

No obstante, resulta también muy conveniente para el seguimiento de una ruta waypoints si se realizan las transiciones entre los distintos tramos de manera apropiada. Es decir, según la implementación realizada de este algoritmo, las transiciones entre tramos de los waypoints no requieren cómputos adicionales, a diferencia del primer método descrito.

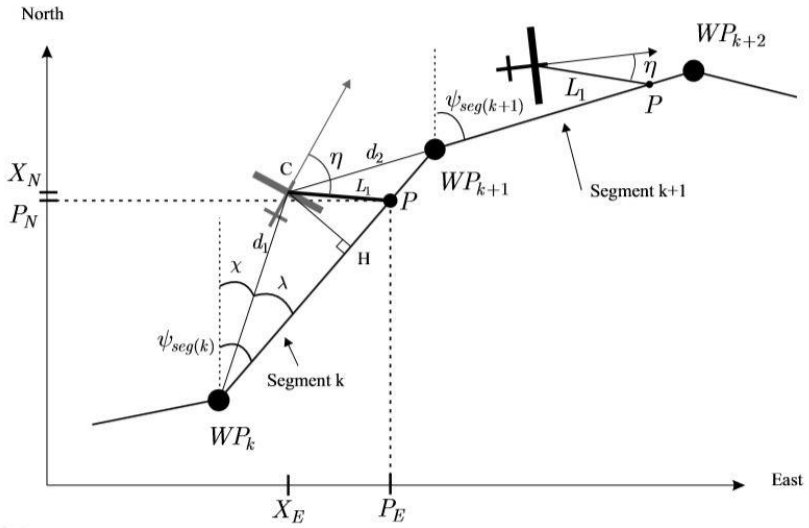

Ilustración 8-Transición entre waypoints [7]

La figura 8 muestra de manera gráfica una ruta compuesta por tramos rectos entre waypoints y cómo la aeronave es capaz de seguir dichos tramos mediante el algoritmo descrito previamente.

El guiado vertical se puede describir como la serie de comandos necesarios sobre el cabeceo y los motores para conseguir los objetivos requeridos, que incluyen: velocidad, empuje, altitud y velocidad vertical.

A diferencia del guiado horizontal, el guiado vertical depende de la fase de vuelo en la que se encuentre la aeronave, puesto que cada una de las fases tiene unos requerimientos verticales distintos. Por ejemplo, el ascenso en rotación suele ser más pronunciado que un ascenso en la fase “en ruta”.

3.3. Guiado Vertical

A continuación se muestran dos propuestas de sistemas de guiado vertical.

3.3.1. Método de Randy Walter

El primer sistema propuesto, descrito en [1] y [6] se compone de distintos modos de control vertical, dependiendo de los requerimientos para cada fase de vuelo. Acorde a estos requerimientos se proponen dos modelos de control:

3.3.1.1. Guiado vertical a velocidad constante

El primer método de control actúa sobre la velocidad de la aeronave, es decir, el objetivo del control es actuar sobre los elevadores para mantener una velocidad constante, por lo tanto, se debe ajustar el ángulo de cabeceo para ajustar la velocidad.

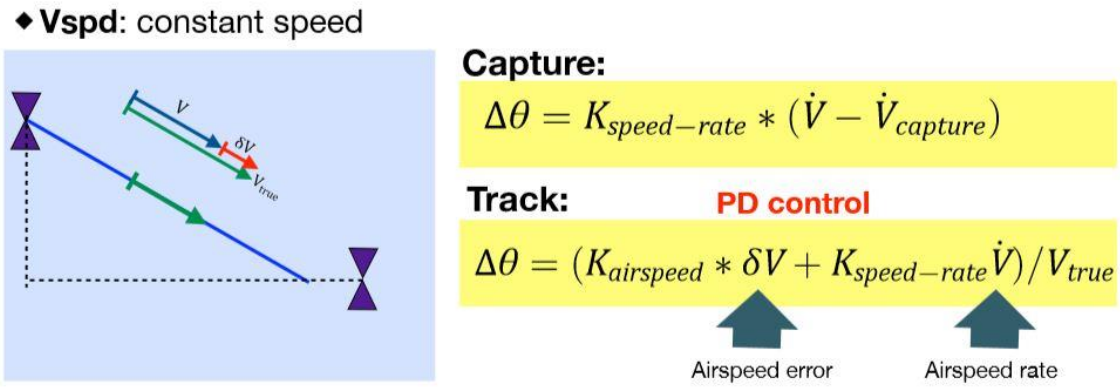

Ilustración 9-Guiado vertical a velocidad constante [7]

Para este método de control se debe computar el error en los cambios de velocidad, y luego controlar este, teniendo en cuenta el viento como error en la velocidad, mediante un controlador PD, como se muestra en la figura 9.

3.3.1.2. Guiado vertical a ángulo constante

El segundo método para el guiado vertical es mantener un ángulo vertical constante, es decir, mantener el ángulo de cabeceo constante. Para ello, se debe ajustar la palanca de gases para proporcionar el empuje adecuado y proporcionar un control sobre los elevadores para mantener estable el cabeceo.

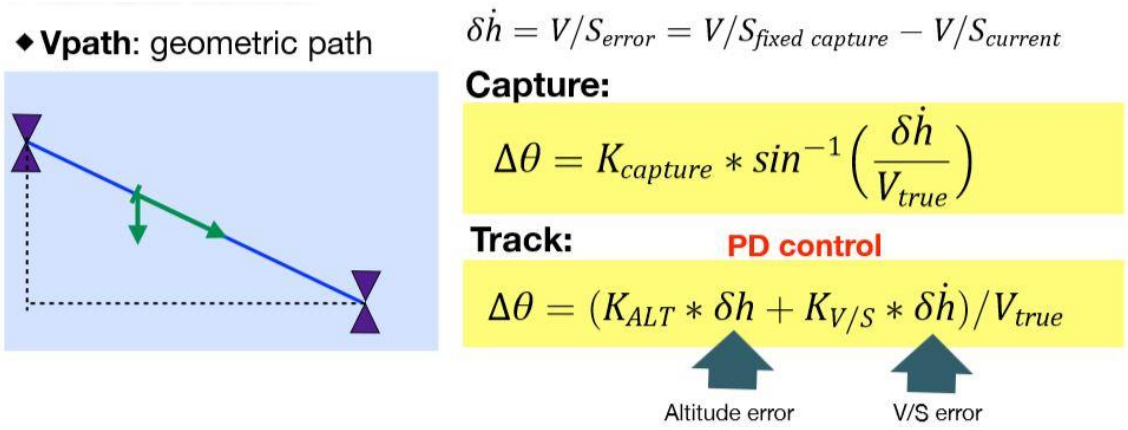

Ilustración 10-Guiado vertical a ángulo constante [7]

De manera similar al método anterior, este método computa un error en el ángulo de cabeceo y controla en función a los cambios de altitud, y la velocidad de la aeronave, los elevadores según un controlador PD.

Se puede observar que un factor determinante en los casos anteriores es el empuje de los motores. No obstante, no es viable realizar un control a tiempo real de la palanca de gases, puesto que una turbina de gas no proporciona una respuesta inmediata y no puede estar sujeta a cambios continuos. Sin embargo, resulta necesario ajustar el empuje para poder realizar correctamente el guiado vertical. Este ajuste no es continuo y se realiza en los momentos en los que resulta coherente este cambio.

El empuje requerido en cada situación se puede modelar del siguiente modo:

$$Thrust = \frac{W \cdot {}^{V}/_{S_{ave}}}{V_{ave}}\left(1 + \frac{V_{ave}}{g} \cdot \frac{dV_{true}}{dh}\right) + D$$

3.3.2. Método de la energía para el guiado vertical

El otro método propuesto para un control vertical se trata de un control por energía. En un momento dado del vuelo la energía mecánica de una aeronave se puede definir por:

$$E_m = E_c + E_p = \frac{1}{2}mv^2 + mgh = m(\frac{1}{2}v^2 + gh)$$

Por simplicidad asumiremos que el cambio en la masa de la aeronave durante la maniobra es despreciable, por lo que un cambio en la energía mecánica de la misma vendrá definido por:

$$\Delta E_m = (\frac{1}{2}v_F^2 + gh_F) - (\frac{1}{2}v_i^2 + gh_i)$$

Como en el método anterior, el guiado vertical se puede realizar a velocidad constante o a ángulo de cabeceo constante.

3.3.2.1. Guiado por energía a velocidad constante

A velocidad constante el cambio de energía mecánica se puede modelar de la siguiente manera:

$$\Delta E_m = gh_F - gh_i = g(h_F - h_i)$$

Como la velocidad no sufre ningún cambio significativo, el control de la energía viene dado por el ritmo al que se cambia la altitud. Por lo tanto, se controlan los elevadores y el empuje para regular la velocidad y el ritmo de cambio de energía.

Esto se muestra en el bucle de control a continuación:

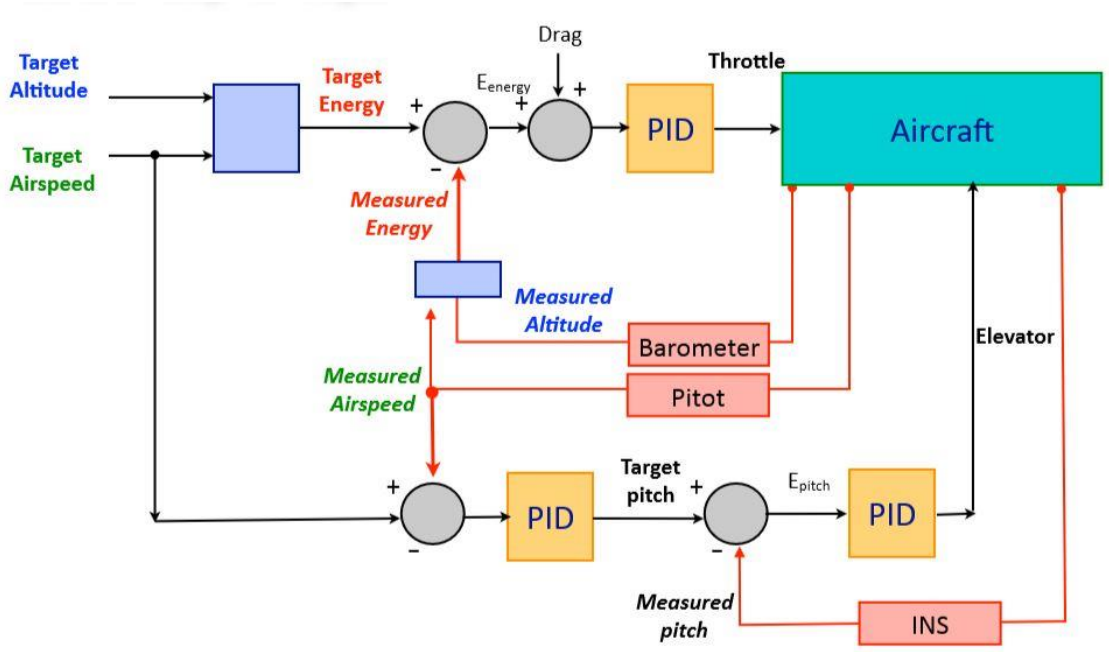

Ilustración 11-Guiado por energía velocidad constante [7]

Se puede apreciar el controlador PID para los elevadores y el PID para el control del empuje, el cual es responsable del cambio de energía, ya que el cambio de energía lo proporciona el motor.

3.3.2.2. Guiado por energía a ángulo constante

Por otro lado, a ángulo de cabeceo constante se modela de distinto modo. Puesto que el ángulo es constante, el ritmo de cambio de altitud se debe de mantener constante, por lo que se debe realizar un control sobre la velocidad.

$$\Delta E_m = \left(\frac{1}{2}v_F^2 + gh_F\right) - \left(\frac{1}{2}v_i^2 + gh_i\right) = \frac{1}{2}\left(v_F^2 - v_i^2\right) + g(h_F - h_i)$$

Este control se realiza sobre la velocidad actuando sobre la palanca de gases y se realiza otro control sobre los elevadores para mantener el ángulo determinado. De manera gráfica:

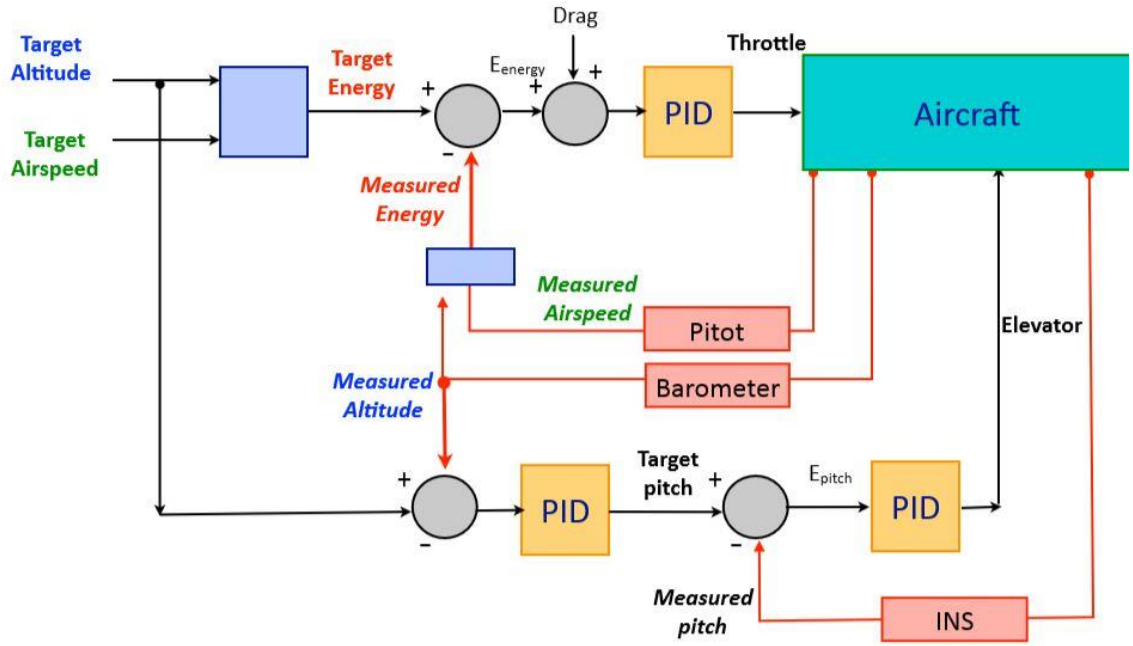

Ilustración 12-Guiado a ángulo constante [7]

Se puede apreciar el cambio en la situación entre el barómetro y el Pitot en ambos bucles, ya que determina sobre qué aspecto del vuelo se realiza cada acción.

4. Herramientas

En este capítulo se describen las principales herramientas empleadas en el desarrollo de este trabajo.

4.1. NetBeans IDE

El sistema de guiado se ha desarrollado en lenguaje Java™ a través del software *NetBeans™ IDE 8.0.2*. Este software se define como: "un entorno de desarrollo integrado (Integrated Development Environment) para escribir, compilar, probar y depurar aplicaciones para la plataforma Java™ y otros entornos. NetBeans IDE incluye un editor de texto completo, herramientas de diseño visual, ayuda al manejo de código fuente, herramientas de integración de bases de datos y muchas otras características"[1].

4.2. XPlane 10

Otra herramienta empleada en el desarrollo de este sistema de guiado es el simulador de vuelo XPlane10™. Este software sirve como interfaz para poder probar el funcionamiento del piloto automático programado en NetBeans. XPlane10 ofrece una simulación muy acertada de la mecánica del vuelo del Boeing 737-800 así como de sus sistemas de vuelo, lo cual permite que se prueben de manera muy certera todas las funciones del piloto automático desarrollado.

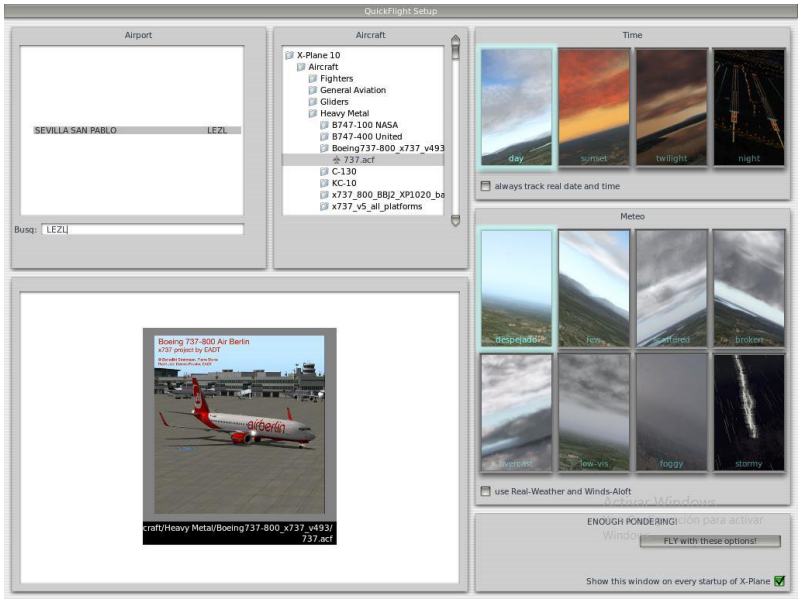

Ilustración 13-Menú inicio XPlane

[1] Traducido del archivo "README" de la versión 8.0.2 de NetBeans™ IDE

Es importante la correcta configuración de XPlane para el correcto funcionamiento de la comunicación con el sistema, empezando por el arranque del software. En la pantalla inicial, como se puede ver en la Figura XXX, se debe seleccionar el B737-800 y el aeropuerto de salida, por nombre o código OACI, según la ruta que se desee completar.

También se pueden variar las condiciones climatológicas y el momento del día en el que se realiza el vuelo con los menús que se encuentran a la derecha de la pantalla de inicio. Por simplicidad, se volará en condiciones normales de día y con climatología favorable para el vuelo, es decir, despejado y durante el día.

Ilustración 14-Pantalla en pista

Una vez iniciado el software de simulación y tras haber seleccionado el origen de la ruta y la aeronave B737-800, se mostrará la pantalla representada en la figura XXX. El simulador se muestra por defecto en vista de cabina, en la que se pueden visualizar todos los instrumentos de la cabina, con cada una de sus funciones. Asimismo, se muestran datos en la esquina superior izquierda (Introducir flecha), en este caso, los datos configurados muestran los tiempos, las velocidades, las posiciones de las superficies de control, los relativos a la posición y orientación de la aeronave, y las estadísticas de ascenso.

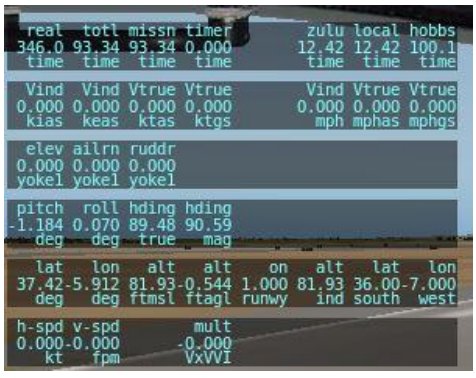

Ilustración 15-Datos Xplane

En la figura anterior se muestran los datos mencionados anteriormente, con sus respectivas unidades. Los datos visualizados en pantalla se pueden configurar mediante el siguiente menú:

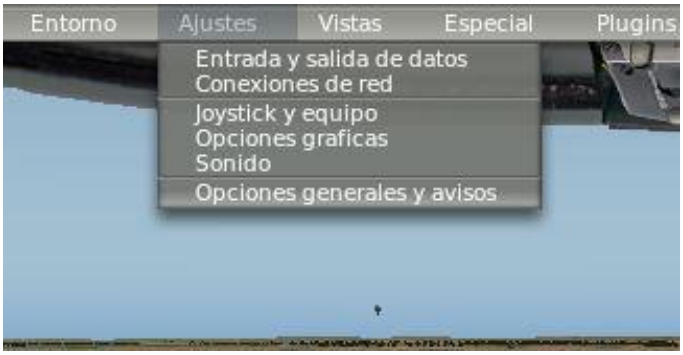

Ilustración 16-Configuración

A través de "Entradas y salidas de datos" se abre el menú de configuración:

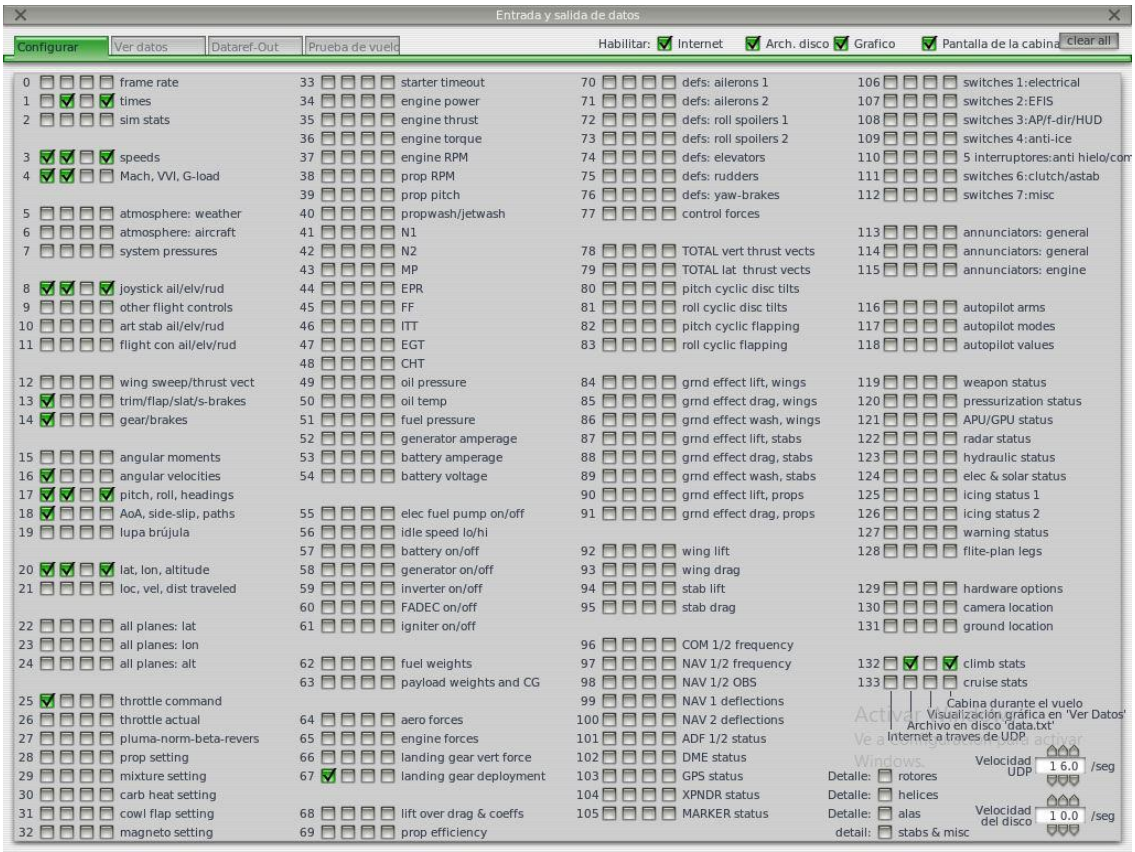

Ilustración 17-Datos XPlane

En este menú se configuran los datos que envía XPlane10 a los diferentes puntos. La casilla de la izquierda envía los datos a través del puerto de salida que se emplearán para conectar NetBeans con XPlane10. La segunda y la tercera casilla envían los datos al archivo de disco y los representa de manera gráfica, respectivamente. La última opción, "Pantalla de la cabina" representa la información de la manera que se ha explicado anteriormente. Se puede observar que las casillas correspondientes a los datos mostrados están seleccionadas, así como otras casillas necesarias para la correcta comunicación con el sistema de guiado, ya que requiere de esos datos.

A continuación, se muestra la configuración del puerto de XPlane para poder comunicar el simulador con el sistema de guiado. Partiendo del menú desplegable de ajustes, mostrado en la figura 14, se selecciona la opción “Conexiones de red”, la cual abre la configuración de entrada y salida de datos del simulador de vuelo.

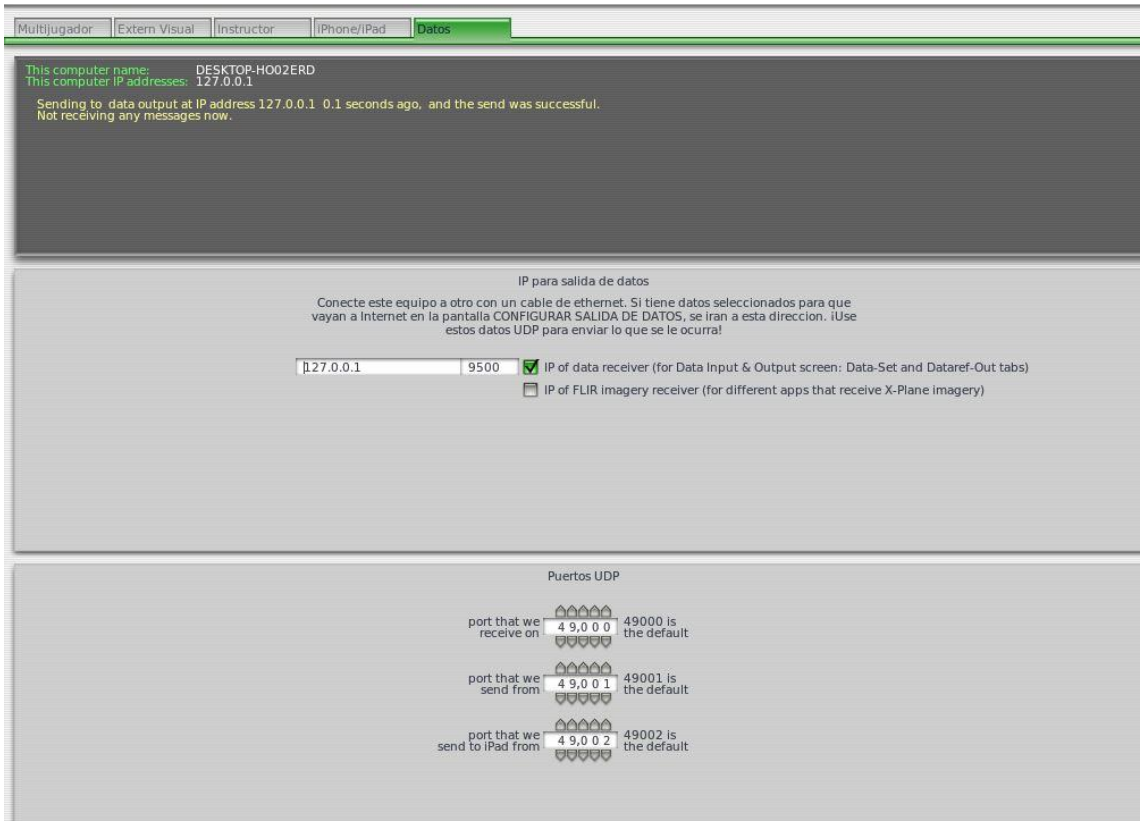

Ilustración 18-Configuración red XPlane

Una vez abierto el menú se selecciona la pestaña “datos” en la esquina superior izquierda, lo cual muestra la pantalla visualizada en la figura XXX. En esta pantalla se configuran los puertos de entrada y salida de datos, por lo que se debe configurar de la manera que se muestra en la figura; asignando el puerto “9500” a la salida de datos a la IP local “127.0.0.1” y el puerto 49000 para los datos de entrada al simulador.

4.3. Otras herramientas

Finalmente, dos herramientas empleadas son Excel y Matlab. Las cuales han sido empleadas para representar los datos mostrados por XPlane. A través del fichero Data.txt generado se emplea Excel para asignar cada dato a una celda y así, con ayuda de la función “xlsread” de Matlab se pueden exportar estos datos y representarlos de forma gráfica.

5. Diseño

Se pretende implementar un sistema de guiado para una aeronave Boeing 737-800. Este sistema debe incluir un guiado vertical y un guiado horizontal, para ello se parte de un piloto automático simple diseñado para una aeronave ligera. El sistema incorpora cuatro de las fases de vuelo, no obstante, el sistema de guiado se ha desarrollado en su totalidad para la fase "en ruta", habiendo ajustado el guiado en el resto de las fases para asegurar su correcto funcionamiento.

5.1. Diseño general

La idea general de este sistema es implementar un sistema que sea capaz de guiar una aeronave a través de una ruta, siguiendo unos waypoints que vendrán definidos por: nombre, latitud, longitud y altitud. Estos datos serán entregados a los programas que se ejecutarán en cada una de las fases de vuelo y que, junto a los datos obtenidos del simulador, se computará la acción a realizar por los controles de la aeronave.

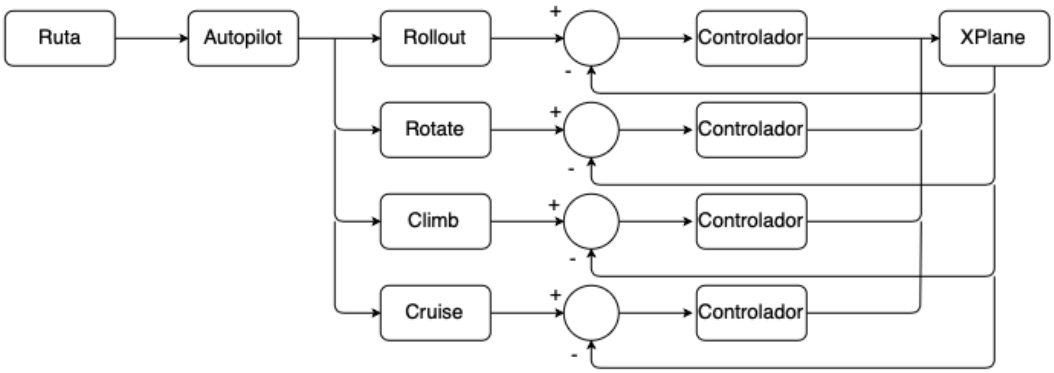

Ilustración 19-Estructura Sistema Guiado

La figura 17 muestra el diagrama de bloques simplificado del sistema de guiado, en el cual se introduce la ruta a seguir y junto a los datos proporcionados por XPlane, se computa a través del controlador la acción a realizar por la aeronave.

Puesto que este sistema tiene que actuar de manera continua sobre la aeronave, se considera un sistema de tiempo real, por lo que se busca la robustez en el código. Por este motivo, y por la posibilidad de ejecutarlo en los principales sistemas operativos comerciales, se emplea la programación orientada a objetos, concretamente el lenguaje Java.

5.2. Guiado Horizontal

A continuación, se describe el diseño del sistema de guiado horizontal empleado en el trabajo. El guiado horizontal diseñado es una interpretación del algoritmo de la zanahoria, aplicando cambios al mismo.

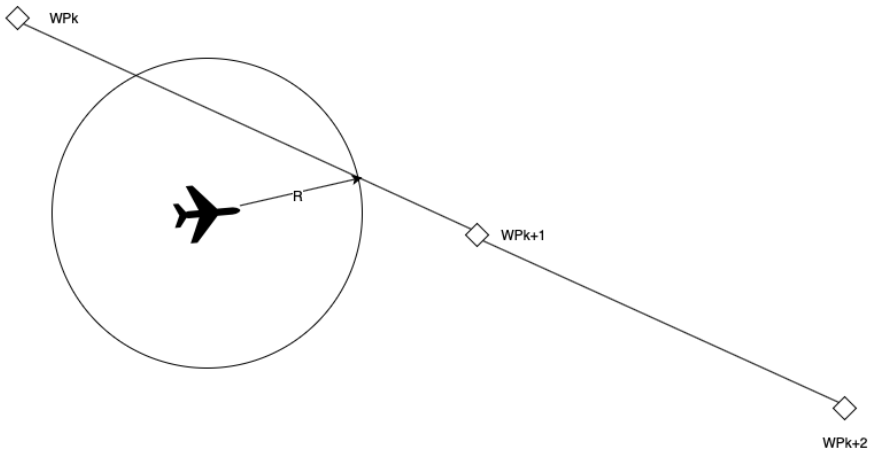

Ilustración 20-Algoritmo de la zanahoria

La figura muestra la aplicación del algoritmo de la zanahoria empleado, en el que se traza una circunferencia centrada en la aeronave con radio R. Acto seguido, se traza el segmento que une los waypoints que definen el tramo de la ruta. El punto de referencia a seguir por la aeronave es la intersección de la circunferencia con el segmento.

Una vez el punto de referencia se encuentra entre unos límites del ultimo waypoint del tramo actual, se activa una función que activa el siguiente tramo de la ruta, de manera que el punto de referencia pasa al siguiente tramo. De esta manera se ejecuta una transición “Flyby” entre los tramos. De modo similar el sistema de guiado permite que se implemente una transición “FlyOver”, ya que requiere un cambio sencillo en el programa, al poner la posición de la aeronave en lugar a la del punto de referencia en la activación del siguiente tramo.

Se puede observar que la intersección de la circunferencia con el tramo recto genera dos puntos en la mayoría de los casos. Mediante el cálculo de la ortodrómica entre esos puntos y el último waypoint del tramo de la ruta se escoge el más cercano a este. Una vez generado el punto de referencia, el sistema de guiado proporciona el rumbo a seguir por la aeronave. Este rumbo es calculado mediante la loxodrómica entre la aeronave y el punto de referencia computado.

Una vez computado el rumbo, el sistema lo devuelve al piloto automático para que este pueda elaborar los algoritmos de control pertinentes para conducir la aeronave a esa dirección.

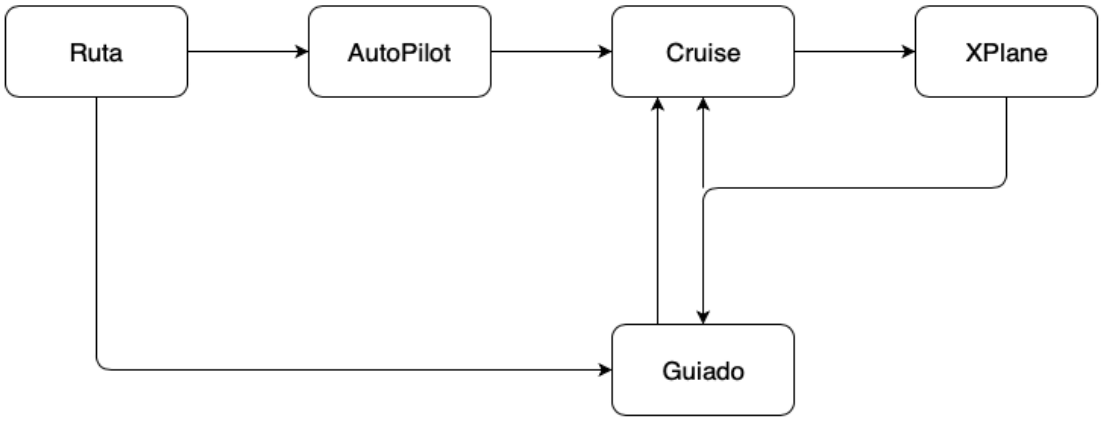

Ilustración 21-Diagrama bloques sistema de guiado

La figura 19 muestra el diagrama de bloques del sistema en cuestión. Los algoritmos de control de la fase “en ruta” o “cruise” toman datos de la ruta a través del autopiloto y, junto a los datos proporcionados por el sistema de guiado y XPlane, computan las acciones a realizar por los elementos de control de la aeronave en el simulador.

Este sistema de guiado proporciona la siguiente trayectoria:

Ilustración 22-Ejemplo funcionamiento guiado horizontal

La figura 22 muestra la ruta seguida por la aeronave tras despegar del aeropuerto de Sevilla (LEZL) siguiendo la ruta programada hacia Valencia (LEVC). El objetivo de la ruta es pasar por los waypoints ROTEX y KUKAL, y como se puede observar, el sistema de guiado horizontal proporciona unos resultados apropiados para un sistema de este tipo.

5.3. Guiado Vertical

A diferencia del guiado horizontal, el guiado vertical se implementa en cada una de las fases de ruta, ya que el mismo afecta directamente a los bucles de control y no se ha implementado en una clase independiente.

En este trabajo se ha diseñado el sistema de guiado vertical en la fase de "en ruta", los sistemas empleados en el resto de las fases de vuelo han sido ajustados a la aeronave Boeing 737-800, pero han sido diseñados e implementados externamente. Sin embargo, el guiado horizontal se aplica en todas las fases del vuelo por igual.

El objetivo del sistema de guiado es llevar a la aeronave de manera eficiente y segura a la altitud determinada en cada uno de los tramos de la ruta o la altitud introducida en pantalla. La idea de esta es simular un de cambio de altitud por orden de control de tráfico aéreo o cualquier otra necesidad.

El sistema de guiado vertical se ha diseñado para su implementación mediante un control por energía con un control proporcional, por lo que el control de la aeronave no es óptimo, como se puede ver a continuación. No obstante, se han empleado los dos métodos explicados para un guiado vertical para seleccionar el que obtuviera mejores resultados.

Para una mejor implementación del sistema de guiado se requiere un mayor ajuste de las ganancias de los controladores y algoritmos más complejos de guiado y control, que, con el tiempo y los recursos disponibles no han sido implementados.

5.3.1. Guiado a velocidad constante

El guiado a velocidad constante se diseña mediante un doble bucle de control en el que se introduce una velocidad de referencia, en este caso 400 kts. Este primer ciclo computa el ángulo de cabeceo necesario para mantener la velocidad de referencia. Acto seguido, el segundo bucle computa la diferencia entre el ángulo computado y el medido por el simulador para enviar una acción a XPlane con el ajuste de elevadores.

El sistema ajusta también el empuje de los motores según se trate de un ascenso o un descenso, ajustando la palanca de gases al 30% en caso de que la altitud objetivo sea menor que la actual y al 80% en caso contrario.

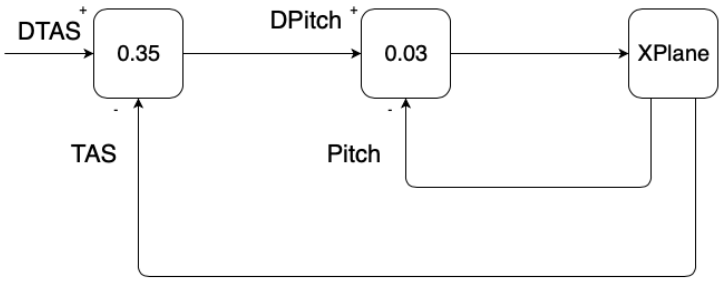

Ilustración 23-Doble bucle de control vertical

La figura 23 muestra el doble bucle de control empleado para el sistema de guiado vertical y las ganancias de los controladores. Es necesario mencionar la existencia de un margen de transición para este guiado. En el caso de que la aeronave se encontrara a menos de 500 pies de la altitud objetivo, el sistema activaba la transición, la cual cambiaba el computo de diferencia de velocidad por la diferencia de altitud entre el objetivo y la actual.

Este método de control generaba los siguientes resultados:

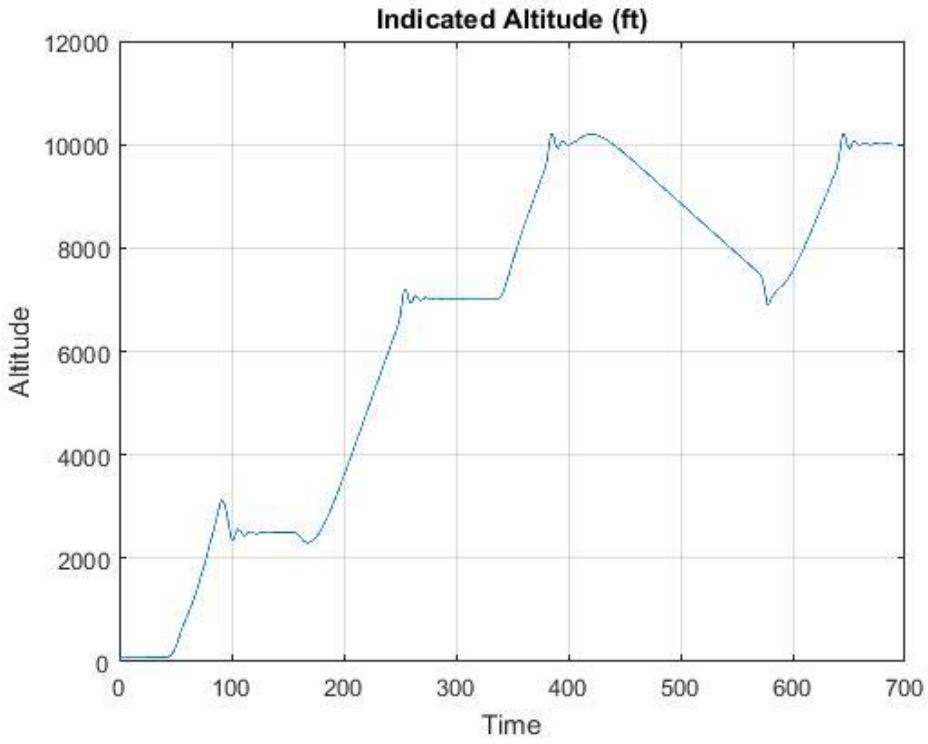

Ilustración 24-Altitud Indicata Vcte

La figura 24 muestra la altitud en pies para los primeros minutos del vuelo. Se pueden observar varias sobre oscilaciones en los cambios de altitud; cabe añadir que, al inicio de los cambios de nivel, se aprecian puntos notables de inflexión. Estos puntos representan el principal error de diseño de este sistema.

Este error era causado por la velocidad de referencia, ya que se había programado que los cambios de altitud se producirían a 400 kts, por lo que al comienzo de la ruta la aeronave ha de reducir altitud para ganar velocidad y después ha de subir ligeramente para perder velocidad.

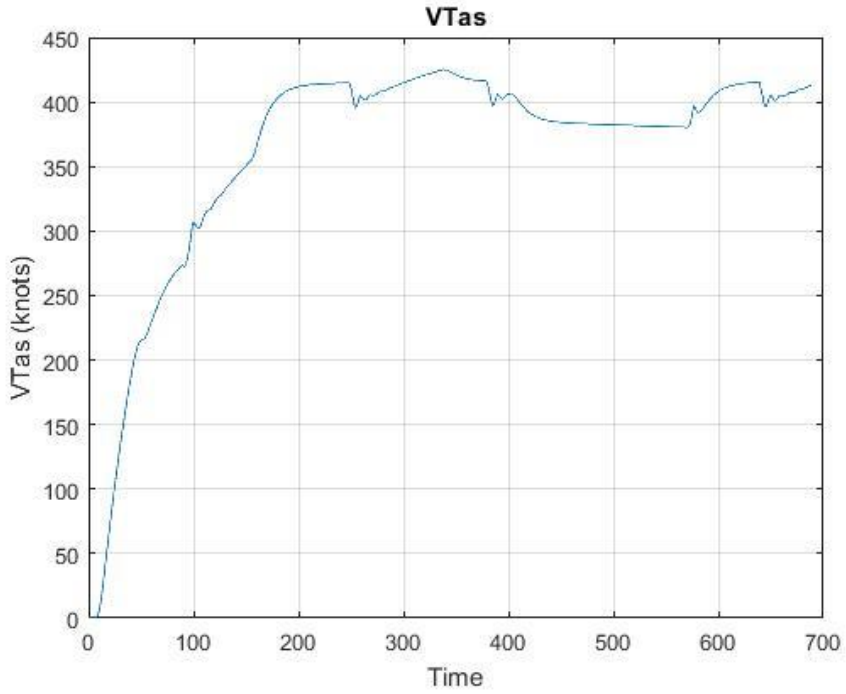

Ilustración 25-VTas a Vcte

La figura 25 muestra la velocidad de la aeronave durante el vuelo. Al igual que en la figura 24 se pueden observar los puntos de cambios de velocidad ya que se representan con un gradiente curvado mostrando el aumento o reducción antes de los ascensos y descensos.

A continuación, se muestran las representaciones de la velocidad vertical, el ángulo de cabeceo y la posición de los elevadores para el segmento de vuelo analizado.

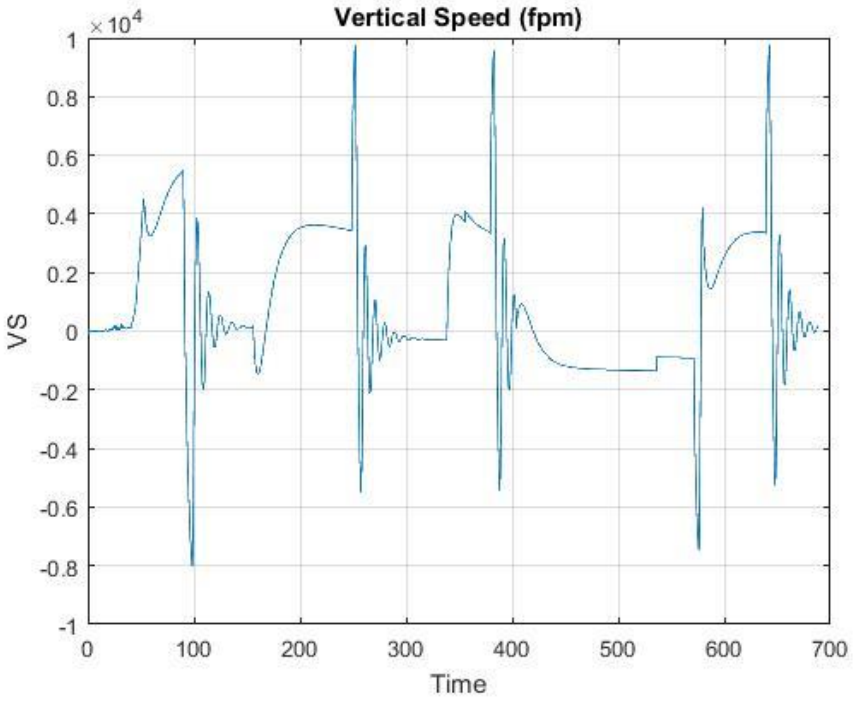

Ilustración 26-Velocidad Vertical a Vcte

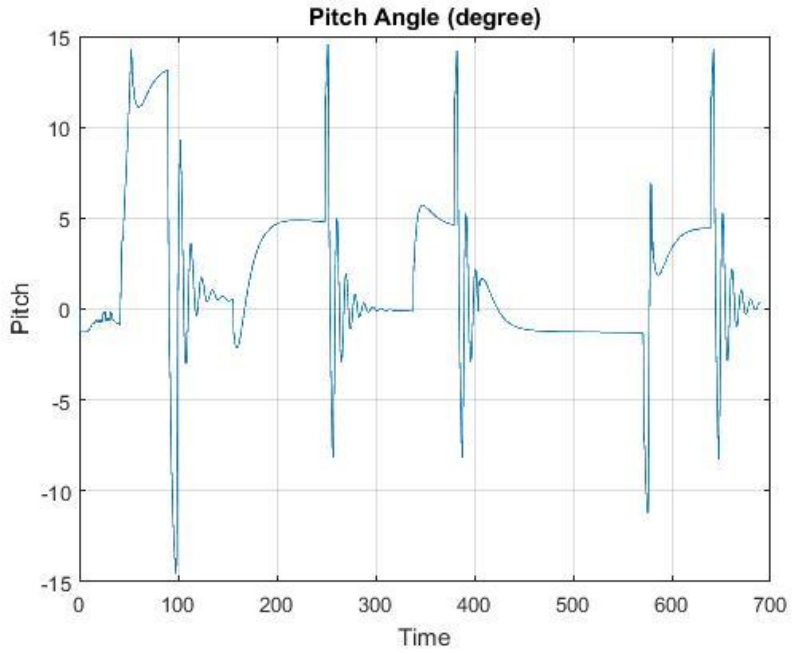

Ilustración 28-Cabeceo a Vcte

Como se puede observar en la figura 24, se producen 5 cambios de altitud en esta fase de vuelo. Esto es representado en las figuras 26, 27 y 28 por las oscilaciones claramente distinguibles. Este tipo de oscilaciones son normales en controles de tipo proporcional como el que se ha implementado, ya que tienden a sobre oscilar.

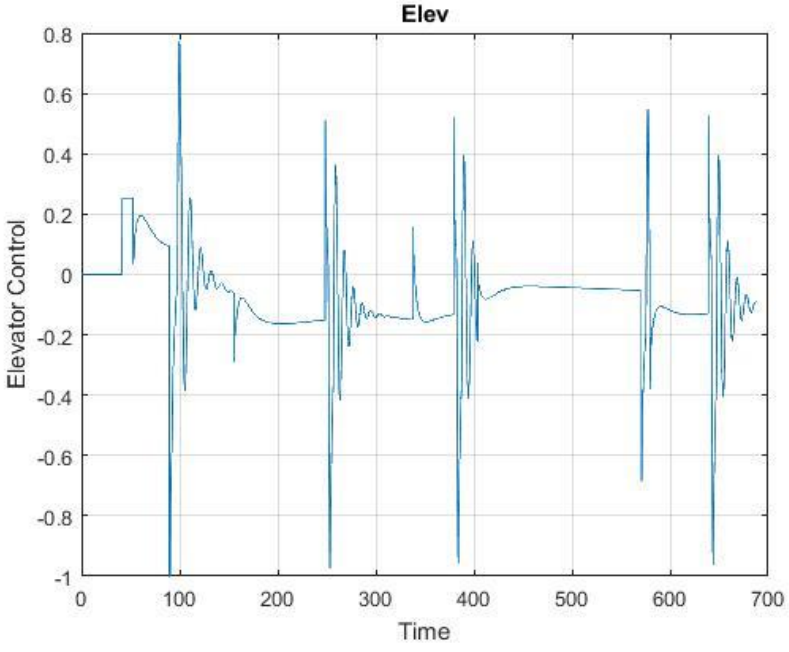

Ilustración 27-Elevadores a Vcte

5.3.2.Guiado Vertical a ángulo constante

A continuación, se explica el sistema de guiado vertical alternativo al explicado en el capítulo 4.3.1.

Este sistema, a diferencia del anterior, se realiza a ángulo de cabeceo constante y no requiere de ningún bucle de control en su etapa principal. No obstante, igual que en el ejemplo anterior, se emplea un controlador proporcional para la correcta nivelación de la aeronave.

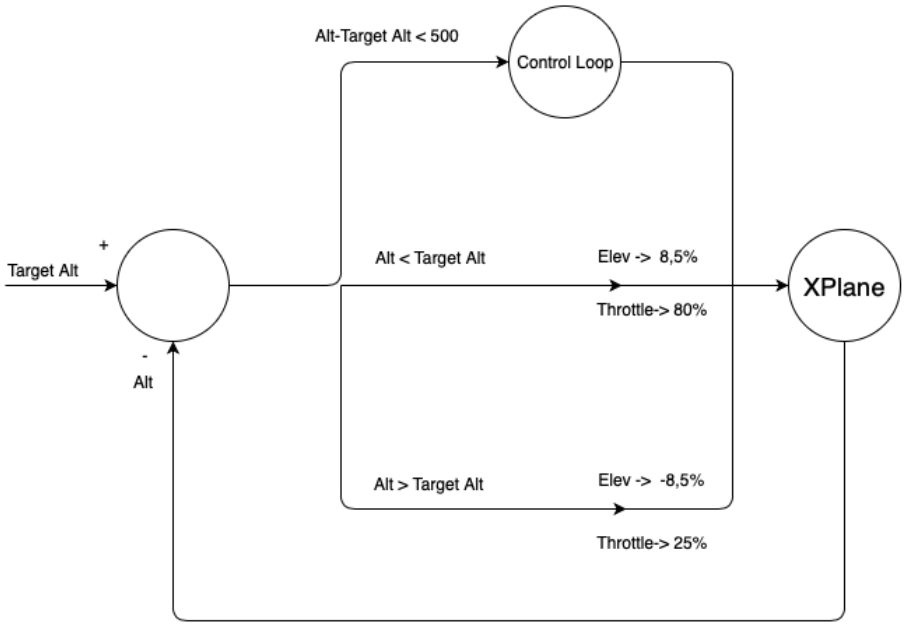

Ilustración 29-Diagrama de bloques guiado vertical cabeceo cte

En el diagrama de bloques mostrado en la figura 26 se muestra la estructura básica del sistema de guiado vertical a ángulo constante. Si la altitud objetivo es mayor que la altitud de la aeronave, el sistema ajusta los elevadores al 8.5% y la palanca de gases al 80%; y en caso contrario, la altitud objetivo es menor que la altitud de la aeronave, se ajustan los elevadores al -8.5% y la palanca de gases al 25%.

En ambos casos, el sistema cuenta con un control adicional que aumenta o disminuye la potencia del motor en función de la velocidad del avión. En otras palabras, si en ascenso la velocidad disminuye de 400 kts, se aumenta la potencia del motor para que opere al 95%; y si en descenso la velocidad aumenta de 400 kts, se reduce la potencia del motor al 15%. De esta manera se mantiene un control de la velocidad ya que el ángulo de cabeceo permanece constante ya que los elevadores permanecen al 8.5% durante todo el proceso.

Puesto que este modelo produce una respuesta inestable cuando se alcanza la altitud objetivo, se implementa el siguiente bucle de control:

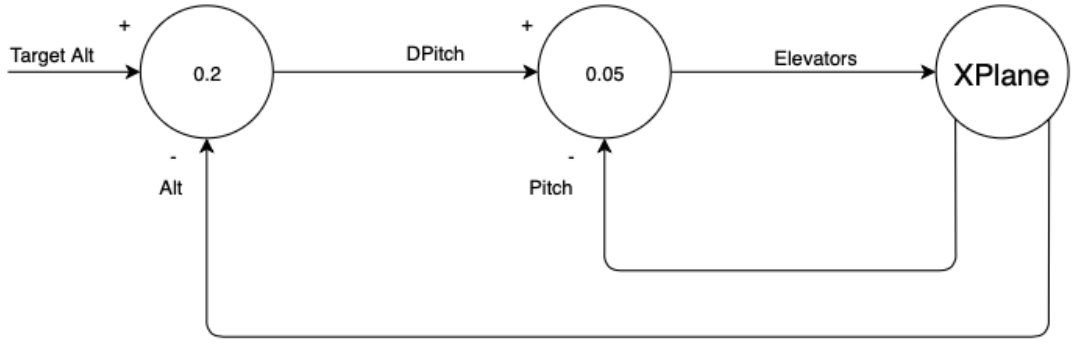

Ilustración 30-Doble bucle control vertical

El bucle representado muestra el mismo bucle de control aplicado a la transición en el capítulo 4.3.1. No obstante, aquel bucle cuenta con las mismas constantes que las mostradas en la figura 20, por lo que este proporciona una respuesta más adecuada, como se demuestra a continuación.

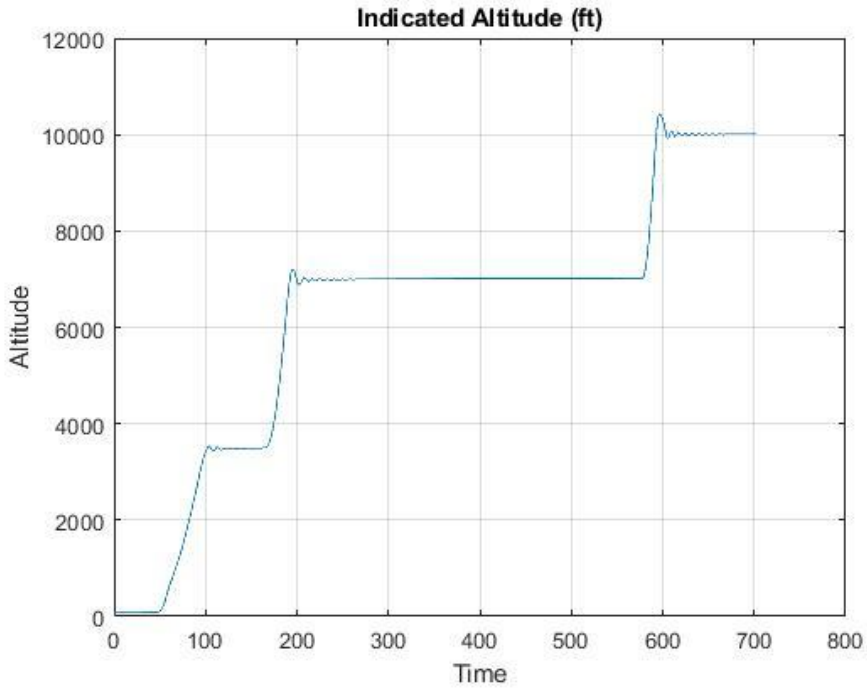

Ilustración 31-Altitud Pitchcte

El gráfico representa la altitud de la aeronave siguiendo el sistema de guiado vertical descrito en este capítulo. A simple vista se aprecian 3 cambios de nivel con una sobre oscilación mucho menor que el caso anterior. Se puede observar también que no hay puntos de inflexión al iniciar un ascenso o un descenso, puesto que el sistema actúa de manera inmediata sobre los elevadores y no requiere un ajuste en la velocidad.

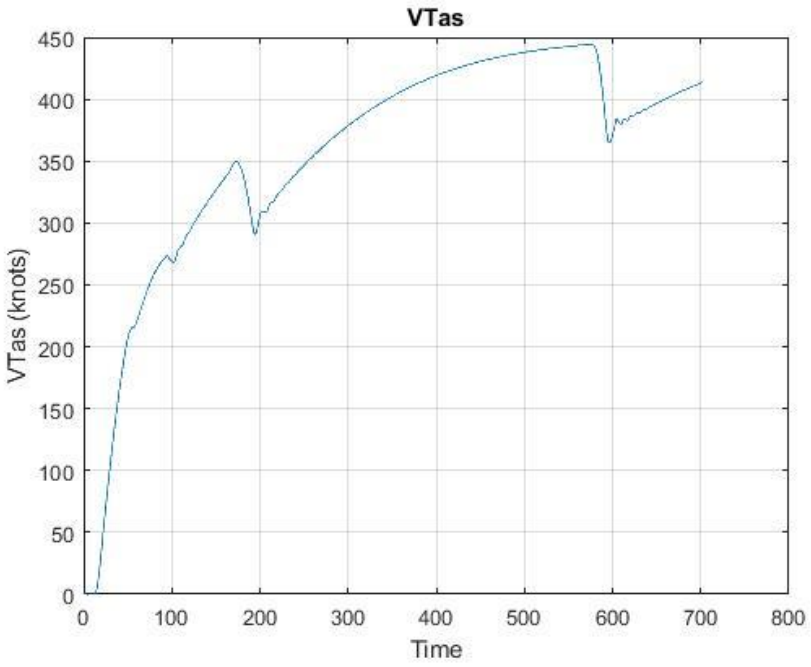

Ilustración 33-Vtas a Pitchcte

En la figura 33 se puede apreciar la velocidad verdadera de la aeronave durante el vuelo. Se puede apreciar que no está sujeta a cambios tan bruscos como en el anterior sistema de guiado, ya que las curvas son mucho más suaves. También se puede observar como la velocidad disminuye al comienzo de cada ascenso, exceptuando el ascenso inicial, pero al producirse el ajuste en la potencia del motor, la velocidad se recupera.

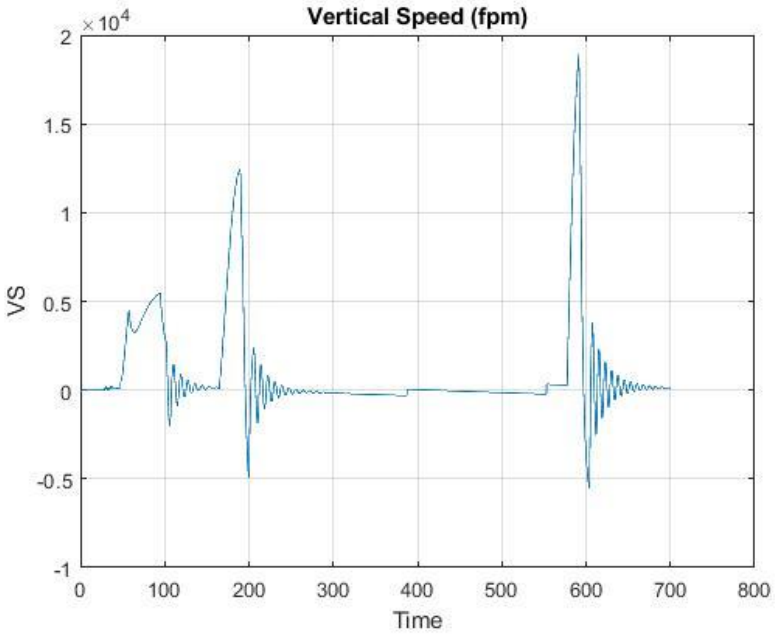

Ilustración 32-VS a Pitchcte

La figura 32 representa la velocidad vertical durante el vuelo. Se puede apreciar un aumento previo a las oscilaciones, representado por los picos en el gráfico. Estas subidas representan los ascensos, puesto que el ángulo es constante y la velocidad verdadera, por lo que la velocidad

vertical también. Se aprecia la diferencia con el modelo anterior en la transición entre picos de oscilaciones, ya que se muestra mucho más estable en este modelo.

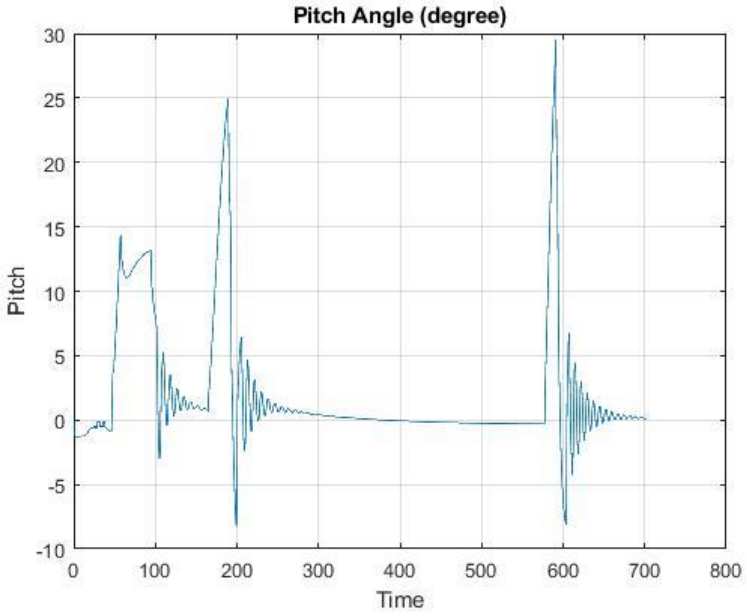

Ilustración 35-Pitch a pitchcte

El gráfico anterior muestra el ángulo de cabeceo de la aeronave. De manera intuitiva, se puede observar que la forma del gráfico coincide aproximadamente con la de la velocidad vertical. Al igual que en la velocidad vertical se observa una transición mucho más suavizada que en el anterior sistema de guiado, aunque los picos muestran valores más elevados.

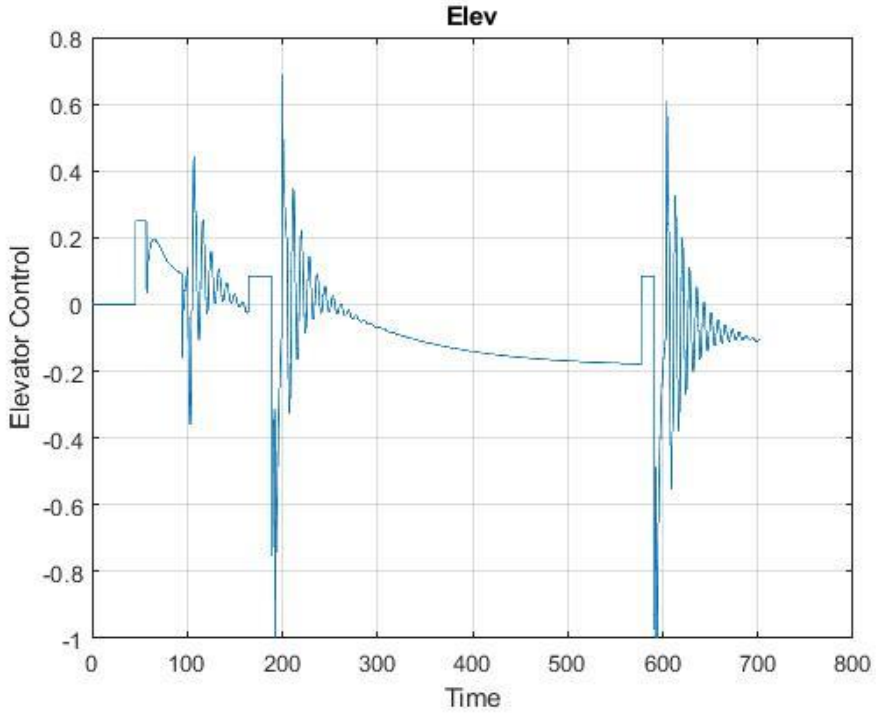

Ilustración 34-Elevadores a pitchcte

Se puede apreciar en el gráfico de la figura 34 los escalones de los elevadores a 0.085 antes de una oscilación, pues es en ese punto donde se realizan los ascensos. Se muestra de manera más

clara en esta imagen la mayor acción requerida para compensar la mayor velocidad vertical adquirida en el ascenso.

En conclusión, el segundo sistema de guiado proporciona una respuesta con menor sobre oscilación, pero necesita más tiempo para estabilizar en la altitud objetivo. Puesto que esto es repercusión de bucle de transición, se puede determinar que se tiene que adquirir un compromiso entre sobre oscilación y tiempo de estabilización. Este problema podría solucionarse implementando un controlador más complejo, no obstante, mediante pruebas empíricas se ha obtenido el sistema de guiado vertical que proporciona, para un control proporcional, una respuesta aceptable.

5.3.3. Guiado en las fases de vuelo

A continuación se detallan los sistemas de guiado vertical y horizontal para cada una de las fases de vuelo incluidas en este trabajo.

5.3.3.1. Fase de Rodadura

Esta fase inicial del vuelo se desarrolla con el tren de aterrizaje sobre la pista, por lo que no conlleva ningún guiado vertical, ya que solo se desplaza en el plano terrestre. Puesto que solo se puede actuar sobre el eje de guiñada en esta fase, la única superficie de control sobre la que actuar es el timón, y requiere un bucle de control simple. Para facilitar el despegue, se configuran los flaps al 50% para esta fase.

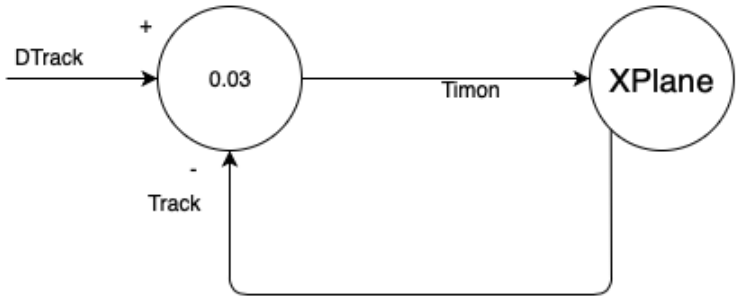

Ilustración 36-Bucle Rodadura

Este bucle de control requiere de un rumbo proporcionado por el sistema de guiado horizontal, al que se le ha programado la ruta a seguir. Esta ruta ha de incluir como primeros waypoints dos puntos alineados con la pista en el sentido a emplear para que la aeronave se mantenga dentro de la pista durante el despegue.

5.3.3.2. Fase de Rotación

La fase de rotación es similar a la fase de rodadura, puesto que no conlleva ningún bucle de control vertical, no obstante, el bucle de guiado y control horizontal es más complejo ya que requiere un computo del alabeo. Por otra parte, el guiado vertical en esta fase se resume en fijar los elevadores al 25% y los flaps al 50% para conseguir un primer ascenso rápido por si se requiriera esquivar algún obstáculo.

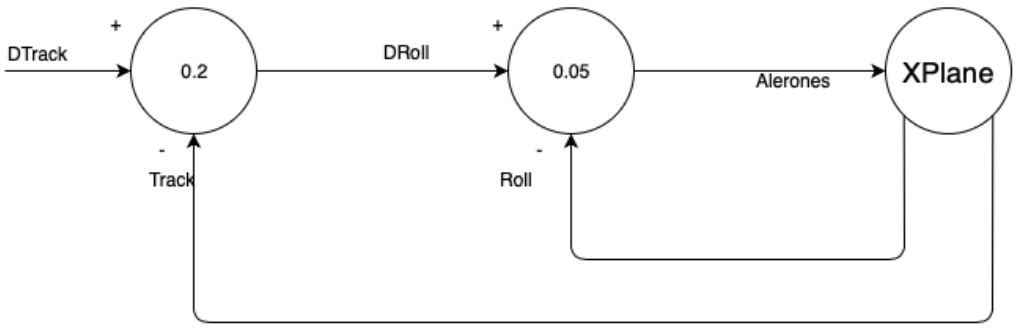

Ilustración 37-Bucle Rotación

5.3.3.3. Fase de Ascenso Inicial

Durante esta fase se emplean ambos sistemas de guiado, el vertical y el horizontal. El sistema de guiado horizontal implementado en esta fase es idéntico al de la fase de rotación que se muestra en la figura 35.

No obstante, el guiado vertical en esta fase funciona a velocidad constante. Sin embargo, puesto que se trata del ascenso inicial, se asume que se realiza con mucha potencia en el motor; en esta simulación se ajusta al 100%, por lo que no se requiere ningún control sobre la palanca de gases. De esta manera, mantener una velocidad constante, en este caso 200kts, es cuestión de ajustar elevadores para variar el ángulo de cabeceo.

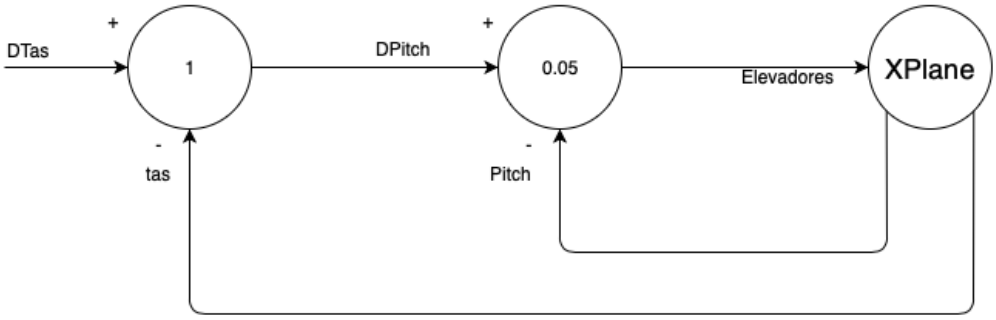

Ilustración 38-Bucle de control ascenso incial

Como función adicional, el piloto automático comanda la recogida el tren de aterrizaje al comienzo de esta fase.

5.3.3.4. Fase “en ruta”

Esta fase contiene el sistema de guiado vertical más complejo, ya que ha de tener en cuenta que el usuario puede introducir una altitud objetivo por pantalla en cualquier momento. Es por ello que al sistema de guiado vertical explicado en el capítulo 4.3.2, que es el empleado en esta fase, hay que añadir una variante. Mientras no se introduzca ningún dato por pantalla, el sistema de guiado funcionará con normalidad, siguiendo la ruta programa, en posición y altitud. No obstante, si se introduce un dato distinto de cero, el sistema de guiado priorizará esta altitud proporcionada y tratará de llevar la aeronave hasta ese nivel. Esta orden se puede revertir introduciendo “0” en la pantalla, de esta manera el sistema de guiado devolverá la aeronave a la ruta original.

Por otro lado, el guiado horizontal en la fase de crucero tiene la estructura de los demás bucles de control lateral, que aplicado a esta fase es:

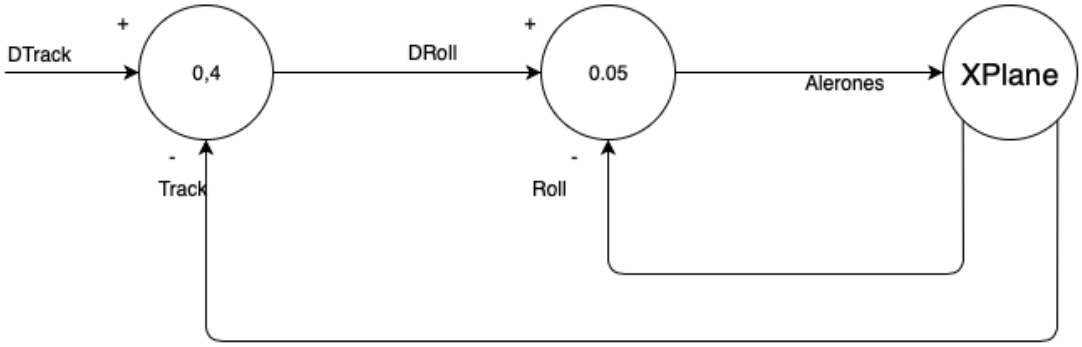

Ilustración 39-Guiado lateral Cruise

6. Implementación

En este capítulo se tratará la implementación de los sistemas de guiado descritos en el capítulo anterior. Por simplicidad en este documento se incluirá solamente la parte del código que se considere necesario o relevante incluir.

La implementación en un lenguaje de programación orientada a objetos conlleva la creación de diversas clases y subclases para poder realizar las acciones programadas.

6.1. Estructura General

En primer lugar, se explica el funcionamiento general de las clases y como se comunican entre ellas.

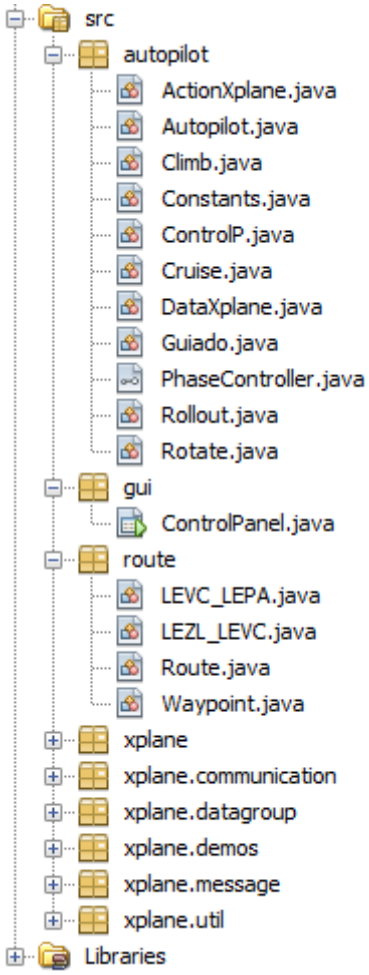

Ilustración 40-Estructura General

Se pueden apreciar 4 paquetes en los que se engloban el resto de las clases: *autopilot*, *gui*, *route* y *xplane*.

A continuación, se muestra la implementación de las clases del programa en cada uno de los paquetes.

6.2. Autopilot

El paquete *autopilot* incluye todas las clases responsables del sistema de guiado, incluyendo la clase *Guiado* responsable del guiado horizontal y vertical; y cada una de las clases responsables de las fases de vuelo.

6.2.1. ActionXplane

La clase *ActionXplane* contiene una serie de *setters* y *getters* con las acciones que se deben realizar en el programa:

public class **ActionXplane**

Constructor
ActionXplane() Inicializa los valores iniciales de los controles del vuelo

Modificador y tipo	Método y descripción
void	setPhase(int phase) Fija la fase al valor introducido
void	setAilerons(float ailerons) Fija el valor de alerones al valor introducido
void	setElevators(float elevators) Fija el valor de elevadores al valor introducido
void	setRudder(float rudder) Fija el valor de rudder al valor introducido
void	setThrottle(float throttle) Fija el valor de throttle al valor introducido
void	setBrake(float brake) Fija el valor de brake al valor introducido
void	setLandingGear(float gear) Fija el valor de landingGear al valor introducido
void	setFlaps(float flaps) Fija el valor de flaps al valor introducido
int	getPhase() Devuelve el valor de phase
float	getAilerons() Devuelve el valor de ailerons
float	getElevators() Devuelve el valor de elevators
float	getRudder() Devuelve el valor de rudder
float	getThrottle() Devuelve el valor de throttle
float	getBrake() Devuelve el valor de brake
float	getLandingGear() Devuelve el valor de landingGear
float	getFlaps() Devuelve el valor de flaps

6.2.2. Autopilot

La clase *Autopilot* es la responsable del funcionamiento del piloto automático. Se realizan los cambios de fases de vuelo, así como el comando de iniciar y terminar el programa.

public class **Autopilot**

extends **Thread**

Constructor
Autopilot(ControlPanel mygui, Route r) Inicializa el programa, la conexión con el simulador y las fases de vuelo

Modificador y tipo	Método y descripción
void	run() Ejecución del programa, envía datos a la interfaz y ejecuta los cambios de fase
void	stopMe() Fija el valor de running a false, parando el programa

6.2.3. Climb

La clase *climb* contiene los algoritmos de control de la fase de ascenso, así como la activación de la siguiente fase.

public class **Climb**

implements **PhaseController**

Constructor
Climb(Route climbRoute) Inicializa los objetos para el sistema de guiado

Modificador y tipo	Método y descripción
ActionXplane	computeAction(DataXplane data) Ejecuta los algoritmos de guiado y control y devuelve las acciones a realizar por la aeronave en forma ActionXPlane

6.2.4. Constants

La clase *Constants* contiene todas las constantes relativas al vuelo empleadas en el sistema de guiado.

public class **Constants**

Constructor
Constants()

Modificador y tipo	Método y descripción

6.2.5. ControlP

La clase *ControlP* actúa como un controlador proporcional, el cual computa una acción a partir de un valor de entrada, una ganancia, y el valor máximo y mínimo de respuesta.

public class **ControlP**

Constructor
ControlP(float Kp, float minv, float maxv) Fija el puntero a los valores introducidos

Modificador y tipo	Método y descripción
float	control(float err) Devuelve la acción proporcional computada si se encuentra dentro de los límites, si no, devuelve el valor máximo o mínimo.

6.2.6. Cruise

La clase *Cruise* contiene los algoritmos de control de la fase "en ruta" así como el guiado vertical de la misma.

public class **Cruise**

implements **PhaseController**

Constructor
Cruise(Route ruta) Inicializa los objetos para el sistema de guiado

Modificador y tipo	Método y descripción
ActionXplane	computeAction(DataXplane data) Ejecuta los algoritmos de guiado y control y devuelve las acciones a realizar por la aeronave en forma ActionXPlane
float	getAltitude() Devuelve la altitud objetivo actual
void	setAltitude(float taltitude) Fija la altitud objetivo

6.2.7. DataXplane

La clase *DataXplane* almacena los datos proporcionados por *XPlane*.

public class **DataXplane**

Modificador y tipo	Método y descripción
void	setLat(float lat) Fija el valor de lat
void	setLon(float lon) Fija el valor de lon
void	setAlt(float alt) Fija el valor de alt
void	setRoll(float roll) Fija el valor de roll
void	setPitch(float pitch) Fija el valor de pitch
void	setHead(float head) Fija el valor de head
void	setAoa(float aoa) Fija el valor de aoa
void	setBeta(float beta) Fija el valor de beta
void	setVpath(float vpath) Fija el valor de vpath
void	setHpath(float hpath) Fija el valor de hpath
void	setIas(float ias) Fija el valor de ias
void	setTas(float tas) Fija el valor de tas
void	setGs(float gs) Fija el valor de gs
void	setVs(float vs) Fija el valor de vs
void	setBrake(float brake) Fija el valor de brake
void	setLandingGear(float gear) Fija el valor de landingGear
float	getLat() Devuelve el valor de lat
float	getLon() Devuelve el valor de lon
float	getAlt() Devuelve el valor de alt
float	getRoll() Devuelve el valor de roll
float	getPitch() Devuelve el valor de pitch
float	getHead() Devuelve el valor de head

float	getAoa() Devuelve el valor de aoa
float	getBeta() Devuelve el valor de beta
float	getVpath() Devuelve el valor de vpath
float	getHpath() Devuelve el valor de hpath
float	getIas() Devuelve el valor de ias
float	getTas() Devuelve el valor de tas
float	getGs() Devuelve el valor de gs
float	getVs() Devuelve el valor de vs
float	getBrake() Devuelve el valor de brake

6.2.8. Guiado

La clase *Guiado* comprende el sistema de guiado horizontal y la altitud objetivo en la fase "en ruta".

public class **Guiado**

Constructor
Guiado(Route ruta) Inicializa la ruta introducida

Modificador y tipo	Método y descripción
float	getAltitude(DataXplane data, float target_altitude) Devuelve target_altitude a no ser que su valor sea 0, en cuyo caso devuelve la altitud del segundo waypoint del segmento actual de la ruta.
float	getDirection(DataXplane data) Computa el algoritmo de la zanahoria para devolver el rumbo a seguir por la aeronave
float	getTAS(DataXplane data) Devuelve el valor de la velocidad introducido en Constants.

6.2.9. Rollout

La clase *Rollout* comprende las acciones ejecutadas en la fase de rodadura.

public class **Rollout**

implements **PhaseController**

Constructor
Rollout(Route rolloutRoute) Inicializa los objetos para el sistema de guiado

Modificador y tipo	Método y descripción
ActionXplane	computeAction(DataXplane data) Ejecuta los algoritmos de guiado y control y devuelve las acciones a realizar por la aeronave en forma ActionXPlane

6.2.10. Rotate

La clase *Rotate* comprende las acciones ejecutadas en la fase de rotación.

public class **Rotate**

implements **PhaseController**

Constructor
Rotate(Route rotateRoute) Inicializa los objetos para el sistema de guiado

Modificador y tipo	Método y descripción
ActionXplane	computeAction(DataXplane data) Ejecuta los algoritmos de guiado y control y devuelve las acciones a realizar por la aeronave en forma ActionXPlane

6.3. Gui

El paquete *gui* incluye la clase principal, por lo que es la clase que se debe ejecutar para iniciar el programa.

6.3.1. ControlPanel

La clase principal *ControlPanel* inicializa el programa y contiene la interfaz gráfica que muestra los datos y permite fijar una altitud durante la ejecución del programa.

public class **ControlPanel**

extends **javax.swing.JFrame**

Constructor
ControlPanel() Inicializa la ruta a seguir y el piloto automático así como la fase “en ruta” del mismo

Modificador y tipo	Método y descripción
void	display(DataXplane dx) Fija los datos del simulador a mostrar por pantalla
void	display(ActionXplane ax) Fija las acciones a mostrar por pantalla
void	formWindowClosing(java.awt.event.WindowEvent evt) Comanda el método StopMe() para detener el programa
void	AltitudeHandler(java.awt.event.ActionEvent evt) Fija la altitud introducida en pantalla en la clase Cruise
static void	main(String args[]) Ejecuta el programa

6.4. Route

El paquete *route* incluye la configuración de las rutas, por lo tanto, la programación de nuevas rutas ha de realizarse dentro de este paquete.

6.4.1. Route

La clase *Route* conforma la lista de Waypoints y la manera de interactuar con la misma.

public class **Route**

Constructor
Route() Crea una nueva ArrayList con objetos de la clase Waypoint
Route(ArrayList<Waypoint> ws) Fija la ArrayList introducida a la ruta.

Modificador y tipo	Método y descripción
void	ponWP(Waypoint W) Añade un Waypoint al último puesto de la lista
Waypoint	firstWP() Devuelve el Waypoint actual según el valor de wp_actual
Waypoint	secondWP() Devuelve el Waypoint que está en la posición wp_actual + 1. En el caso de no existir, devuelve el Waypoint en la posición wp_actual
void	siguienteWP() Aumenta el valor de wp_actual en 1

6.4.2. Waypoint

La clase *Waypoint* conforma la estructura de los waypoints empleados en las rutas.

public class **Waypoint**

Constructor
Waypoint(String name, float lat, float lon, float alt) Crea un waypoint con cada uno de sus atributos

Modificador y tipo	Método y descripción
String	getName() Devuelve la cadena correspondiente al nombre del waypoint
float	getLongitude() Devuelve el valor de longitude
float	getLatitude() Devuelve el valor de latitude
float	getAltitude() Devuelve el valor de altitude
void	setName(String name) Fija la cadena name como el nombre del waypoint
void	setLongitude(float longitude) Fija el valor de longitude
void	setLatitude(float latitude) Fija el valor de latitude
void	setAltitude(float altitude) Fija el valor de altitude

6.4.3. Sub-Clase de Ruta

Se debe generar una sub-clase por cada ruta programada, de esta manera, los waypoints serán declarados en la misma.

A continuación se muestra el ejemplo de la ruta Valencia – Palma de Mallorca.

public class **LEVC_LEPA**

extends **Route**

Constructor
LEVC_LEPA()

```
public class LEVC_LEPA extends Route {
    public LEVC_LEPA() {

        Waypoint RWY12 = new Waypoint("RWY12",39.48361f, -0.46661f, 00172f);
        Waypoint VLCTY = new Waypoint("VLCTY",39.42000f, -0.30241f, 03000f);
        Waypoint MULAT = new Waypoint("MULAT",39.39991f, -0.17991f, 03000f);
        Waypoint COMPI = new Waypoint("COMPI",39.35081f,  0.00787f, 04000f);
        Waypoint NINOT = new Waypoint("NINOT",39.20883f,  0.48333f, 04000f);
        Waypoint GODOX = new Waypoint("GODOX",39.37250f,  1.41083f, 04000f);
        Waypoint RUPIT = new Waypoint("RUPIT",39.45269f,  2.02697f, 03000f);
        Waypoint LEPA  = new Waypoint("LEPA ",39.55254f,  2.73496f, 00150f);

        this.ponWP(RWY12);
        this.ponWP(VLCTY);
        this.ponWP(MULAT);
        this.ponWP(COMPI);
        this.ponWP(NINOT);
        this.ponWP(GODOX);
        this.ponWP(RUPIT);
        this.ponWP(LEPA );
    }
}
```

Ilustración 41-Sub-clase de Ruta

6.5. Xplane

El paquete *Xplane*, con sus respectivos sub-paquetes, contiene la configuración de la comunicación entre el programa y *XPlane*. Puesto que los cambios introducidos no han afectado al protocolo de comunicación, se muestran solo los cambios introducidos en los datos entrantes y salientes del simulador.

6.5.1. XPlaneInputOutput

En la clase *XPlaneInputOutput* se ha generado la modificación que permite el control sobre el tren de aterrizaje para así poder recogerlo o desplegarlo a voluntad. También se añade la modificación que permite actuar sobre el segundo motor de la aeronave, ya que en su diseño original, este piloto automático estaba pensado para aeronaves de un solo motor. Estos cambios implicaban únicamente la adición de tres líneas de código en los métodos *read*() y *write*().

```
53          data.setVpath(xpi.getValue("orientation.vpath"));
54          data.setHpath(xpi.getValue("orientation.hpath"));
55          data.setBrake(xpi.getValue("controls.brake"));
56          data.setLandingGear(xpi.getValue("controls.landingGear"));
57          data.setIas(xpi.getValue("position.ias"));
58          data.setTas(xpi.getValue("position.tas"));
```

Ilustración 42-Método Read()

En el método *read*() se añade la línea 56, mostrada en la figura 42, correspondiente al tren de aterrizaje.

```
69            xpi.setValue("controls.rudderPosition", action.getRudder());
70            xpi.setValue("engine.throttleCommand1", action.getThrottle());
71            xpi.setValue("engine.throttleCommand2", action.getThrottle());
72            xpi.setValue("controls.brake", action.getBrake());
73            xpi.setValue("controls.landingGear", action.getLandingGear());
```

Ilustración 43-Método Write()

Por otro lado, se añaden las líneas 71 y 73, mostrada en la figura 43, representando el control sobre el segundo motor y el tren de aterrizaje respectivamente, en el método *write*().

7. Presupuesto

Este apartado comprende el presupuesto necesario para la realización del proyecto. Este presupuesto englobará los costes directos e indirectos generados durante el periodo de finalización de este. Estos costes incluyen: los equipos necesarios para la realización, las licencias del software empleado, el coste de personal y los costes indirectos.

7.1. Coste de hardware y software

A continuación se enumeran los equipos y el software necesarios para la realización de este proyecto:

Tabla 1-Unidades Equipos

DESCRIPCIÓN	UNIDADES
ORDENADOR DE SOBREMESA	1
ACCESORIOS	1
MICROSOFT OFFICE PROFESSIONAL	1
MATLAB 2019A	1
XPLANE 10	1
NETBEANS IDE 8.0.2	1

7.1.1. Ordenador de Sobremesa y accesorios

El ordenador de sobremesa es una herramienta muy versátil que es empleada en una media de 1 proyecto al año. Si se supone una vida útil de 5 años, se realizan un total de 5 proyectos, por lo que el coste del mismo se puede calcular por proyecto.

De esta manera el precio de los equipos quedaría:

Tabla 2-Coste Equipos

DESCRIPCIÓN	PRECIO UNITARIO	PRECIO POR PROYECTO
ORDENADOR SOBREMESA	1200€	240€
ACCESORIOS	200€	40€
TOTAL		280€

7.1.2.Software

En el caso del software, se emplean las licencias profesionales, por lo que se debe computar el coste de cada una de ellas. En el caso de Microsoft Office Professional, se emplea en el total de los proyectos realizados en el ordenador, y que al cambiarse el hardware se necesitará una renovación de licencia.

En el caso de Matlab 2019a, al ser una herramienta de ingeniería se emplea en el 80% de los proyectos realizados en la vida útil del hardware, es decir, en 4 de los 5 proyectos.

La licencia de Netbeans IDE es gratuita, por lo que no conlleva ningún coste. Sin embargo, XPlane 10 ya no se encuentra disponible para su adquisición, por lo que es necesario adquirir la licencia comercial de XPlane 11. Puesto que este software solo se emplea para la realización de este proyecto, es necesario incluir el importe íntegro de la licencia en el presupuesto.

Tabla 3-Coste Software

DESCRIPCIÓN	PRECIO UNITARIO	PRECIO PROYECTADO
MICROSOFT OFFICE PRO 2019	579€	115,80€
MATLAB 2019ª	2000€	400,00€
XPLANE 11	750$/675€	675,00€
NETBEANS IDE 8.0.2	0€	0€
TOTAL		1190,80€

7.2. Coste de Personal

El coste de personal consiste en computar el precio del personal cualificado necesario para la realización de este proyecto. Para ello se considera una tasa horaria de 20€/hora para el tutor y de 10€/hora para el alumno.

El cómputo de las horas realizadas por el alumno ha sido de 30 horas semanales de media entre junio y la primera quincena de septiembre, por lo que el cómputo total suma 420 horas.

Por otro lado, se computan 25 horas de dedicación por parte del tutor.

Tabla 4-Coste de Personal

DESCRIPCIÓN	HORAS	COSTE/HORA	COSTE TOTAL
ALUMNO	420	12,5€	5250€
TUTOR	25	25€	625€
TOTAL			5875€

7.3. Coste Indirecto

Este proyecto conlleva una serie de costes que no están relacionados de manera directa con la realización de mismo. Ya sea coste de electricidad, servicio de conexión a internet, etc.. Estos costes tienen un carácter arbitrario y no se pueden calcular con exactitud, por lo que se asumirá que son iguales al 2.5% de los costes directos.

Tabla 5-Coste Indirecto

DESCRIPCION	COSTE	APORTACIÓN COSTE INDIRECTO
COSTE EQUIPOS	1470,80€	36,75€
COSTE PERSONAL	5875€	146,88€
COSTE INDIRECTO		183,63€

7.4. Coste Total

Tras realizar todos los cálculos pertinentes, se obtiene un presupuesto total para la realización del trabajo de 7529,43€ (SIETE MIL QUINIENTOS VEINTINUEVE EUROS CON CUARENTA Y TRES CENTIMOS).

Tabla 6-Coste total

DESCRIPCIÓN	COSTE
COSTE EQUIPOS	1470,80€
COSTE PERSONAL	5875€
COSTE INDIRECTO	183,63€
TOTAL	7529,43€

8. Conclusiones

El objetivo de este trabajo es el diseño e implementación de un sistema de guiado en una aeronave Boeing 737-800. A través del análisis de diversas maneras de realizarlo, se ha seleccionado la forma que se ha considerado óptima, por su complejidad de cómputo, implementación y precisión. De esta manera se ha obtenido un sistema de guiado capaz de conducir la aeronave por la ruta asignada durante las primeras fases del vuelo.

Para lograrlo ha sido necesario el estudio de la bibliografía pertinente para obtener conocimientos suficientes para el diseño del sistema. Por otra parte, ha sido imprescindible el aprendizaje de programación orientada a objetos, así como el funcionamiento de la comunicación con el entorno de simulación para poder ajustar el piloto automático inicial a los requisitos de la aeronave Boeing 737-800.

Para poder visualizar y controlar el error producido en el sistema se han programado dos rutas. Estas rutas sobrepasan los waypoints que el simulador nos muestra, por lo que se puede observar de manera fácil e intuitiva el error generado por el sistema.

Al no disponer del modelo dinámico de la aeronave, los ajustes de las ganancias de los controladores se han realizado de manera intuitiva observando los resultados obtenidos. De esta manera se han necesitado muchas horas de simulación para realizar cada ajuste y se han obtenido resultados excelentes en el guiado horizontal y mejorables en el guiado vertical.

En cuanto al guiado horizontal, la implementación del algoritmo de la zanahoria se ha realizado correctamente ya que la aeronave es capaz de seguir la ruta con bastante exactitud. No obstante, en el guiado vertical se pueden observar notables sobre-oscilaciones en los cambios de altitud que no muestran el comportamiento real de una aeronave de este tipo.

Estas sobre-oscilaciones están causadas por un posible ajuste erróneo del controlador, ya que la ganancia puede no haber sido ajustada correctamente o que se debería haber implementado un controlador más complejo, por ejemplo, un PID. Estas sobre-oscilaciones han sido reducidas lo máximo posible en el tiempo que se ha realizado este trabajo.

Queda pendiente la mejora en la integración de la interfaz gráfica del programa así como la implementación de un controlador PID para el guiado vertical y el diseño del sistema de guiado para la fase de descenso y aterrizaje de la aeronave.

9. Bibliografía

1. Randy Walter. *The avionics Handbook*. Chap 15 – "Flight Management Systems". Ed. Cary R. Spitzer Boca Raton, CRC Press LLC. 2001.
2. Maarten Uijt de Haag. EE6900 *Flight Management Systems*. Ohio University course.
3. Ducard GJJ. *Fault-tolerant Flight Control and Guidance Systems*. Ed. Springer. Series: Advances in Industrial Control. 2009.
4. Mangal Kothari. *Algorithms for Motion Planning and Target Capturing*. Ph D. Thesis Univ. Leicester. 2010.
5. Eurocontrol*. User Manual for the Base of Aircraft Data (BADA)* Rev. 3.9. EEC Technical/Scientific Report No. 11/03/08-08.
6. The Boeing Company*. 787-8 Flight Crew Operations Manual*
7. Vila Carbó, J. (2018). *Sistemas de Gestión de Vuelo por Computador*. Ch 8 – "Flight Management Systems"