MASTER UNIVERSITARIO EN INGENIERÍA AERONÁUTICA

TRABAJO FINAL DE MASTER

Simulación y Guiado de Aeronaves utilizando Programación Orientada a Objetos

Autor

Lucas Peris Lozano

Master Universitario en Ingeniería Aeronáutica

Universitat Politècnica de València

Escuela Técnica Superior de Ingeniería del Diseño

2015-2016

Resumen

Los simuladores de vuelo se han convertido hoy en día en una herramienta fundamental tanto para la formación de pilotos como para otras aplicaciones tales como la prueba y validación de nuevos planes de misión y el desarrollo de sistemas de guiado y control. Esto último es de gran utilidad hoy en día, especialmente cuando se aplica a aeronaves no tripuladas (UAVs). La simulación de las misiones de dicho tipo de aeronaves puede llegar a convertirse en una herramienta fundamental en el desarrollo de aeronaves no tripuladas, especialmente en aquellas empresas o instituciones que no disponen del presupuesto necesario para realizar numerosos ensayos de vuelo.

La capacidad de simular la misión permite encontrar fallos tanto en el concepto mismo de la misión, como fallos debidos a las filosofías de guiado y control aplicadas en el vehículo, todo ello de forma sencilla y a un coste muy reducido. Cabe destacar que la correcta planificación de una misión y el diseño del sistema de navegación, guiado y control de una aeronave son temas muy complejos y que requieren de un estudio en profundidad para poder garantizar la seguridad en las operaciones.

El enfoque de este trabajo consiste en la realización de una aplicación gráfica que englobe conceptos tanto de simulación de aeronaves como de guiado y control de las mismas. Además, se ha implementado una serie de instrumentación en la cabina del simulador que permitirá obtener una visión más realista de la simulación, así como familiarizar al lector con distintos conceptos de instrumentación de aeronaves.

Todo esto se ha realizado con el objetivo principal de ofrecer una plataforma modular y ampliable que pueda servir para fines docentes de desarrollo de software para aplicaciones aeronáuticas.

Debido a los temas tan amplios que se abordan en este trabajo, en la implementación del proyecto se ha profundizado en distinta medida en cada uno de los temas especificados anteriormente. Sin embargo, se han abordado todos los temas por igual desde el punto de vista de la documentación, del estudio del estado del arte y del diseño de los sistemas que los componen.

Abstract

Flight simulators have become nowadays a fundamental tool for both pilot training and other applications such as the test of new mission plans and the development of guidance and control systems. The latter is very useful these days, especially when its applied to unmanned aerial vehicles (UAVs). The mission simulation of that kind of aircrafts could become a fundamental element for the development of unmanned aircrafts, specially in companies or institutions that do not have the required budget for an intensive flight test program.

The ability of simulating a complete mission allows the engineers to find errors in the mission concept as well as errors due to the guidance and control philosophies applied to the vehicle, all that in a very simple and cost-effective way. It must be noted that an adequate mission planning procedure and the design of a guidance, navigation and control system of an aircraft are extremely complex subjects, which require an in-depth study in order to guarantee the operational security requirements.

The scope of this work consists on the design and implementation of a graphical application which includes concepts of both aircraft simulation and their guidance and control. Additionally, a number of navigation instruments has been implemented in the simulator's cockpit, which will lead to a more realistic vision of the simulation, as well as familiarize the reader with different concepts related to aircraft instrumentation.

The main objective of this work is to offer a modular and easily expandable that can be used for educational purposes in the fields of software development for aeronautical applications.

Due to the breadth of the subjects seen in this project, the depth of their implementation in this project has been different for each of them. However, all the aspects of this project have been treated equally from the points of view of documentation, study of the state of the art and the design of their systems and components.

MASTER UNIVERSITARIO EN INGENIERÍA AERONÁUTICA

Simulación y Guiado de Aeronaves utilizando Programación Orientada a Objetos

Lucas Peris Lozano

13 de julio de 2016

Índice

Índice de Figuras

Índice de Tablas

Capítulo 1

Introducción

1. Motivación del trabajo

El potencial que tiene la utilización de técnicas de simulación para el desarrollo de sistemas de guiado y control de aeronaves es muy elevado. La constante mejora del hardware de los computadores ha permitido la proliferación de multitud de plataformas comerciales de simulación.

El continuo desarrollo visto en los últimos años en el sector de las aeronaves no tripuladas ha puesto en el punto de mira la mejora de las herramientas de simulación disponibles hoy en día.

La motivación del presente proyecto reside en cuatro aspectos clave:

- La gran utilidad de las técnicas de simulación en las etapas de desarrollo de los sistemas de navegación, guiado y control de las aeronaves. Por tanto, la creación de una plataforma integrada de simulación que permita realizar distintos ensayos de los nuevos sistemas de guiado y control puede convertirse en una herramienta de mucha utilidad en el proceso de desarrollo de los sistemas mencionados anteriormente.
- El amplio abanico de posibilidades que se presenta al combinar técnicas de simulación con algoritmos de control da lugar a que se puedan plantear nuevos tipos de ensayos que llevar a cabo, tales como la verificación de nuevas misiones o incluso pruebas mediante el sistema *Hardware in the Loop*, que permite conectar un autopiloto real (o incluso un UAV completo) al simulador para realizar multitud de tests.
- La implementación de un entorno de simulación integrado con una antena que permite leer el tráfico aéreo dota al sistema de una capacidad de ampliación notable, pudiendo así simular condiciones de tráfico real e implementar sistemas de alerta de tráfico y evasión de colisión (TCAS: *Traffic alert and Collision*

Avoidance System). Esto tiene un interés no solamente desde el punto de vista del desarrollo, sino también desde un punto de vista didáctico, ya que puede permitir que en un futuro otros alumnos experimenten con la herramienta para poder implementar así sistemas más complejos y completos.

- La creación de una plataforma modular que integre los módulos de simulación, autopiloto y tráfico aéreo mediante el lenguaje Java, puede llegar a tener multitud de utilidades en un futuro, siendo fácilmente ampliable y actualizable. Además, la implementación en Java permite que el mismo programa pueda ser utilizado en diversos sistemas operativos sin necesidad de recompilar de nuevo el programa.

2. Objetivos

Con los cuatro aspectos claves anteriores en mente, se plantean los siguientes objetivos para la realización del presente proyecto:

1. Capacitar al alumno para el diseño de herramientas informáticas para la simulación de vuelo y gestión del tráfico aéreo. Esta es una especialización de gran interés para la Ingeniería Aeronáutica y con un gran número de aplicaciones tanto en la industria como en la rama de investigación.

2. La integración en un mismo proyecto de distintas áreas de estudio del Master Universitario en Ingeniería Aeronáutica, tales como:

   - Algoritmos de guiado y control de aeronaves.
   - Instrumentación de aeronaves.
   - Desarrollo de interfaces gráficas.
   - Simulación de vuelo.

3. Afianzar los conocimientos obtenidos en el transcurso del Master, realizando una plataforma sobre la que se pueda realizar multitud de trabajos futuros. Para ello se realizará una implementación de software mediante el uso de un lenguaje orientado a objetos que permita generar un código modular fácilmente adaptable, ampliable y modificable.

4. Estudiar el estado del arte de las disciplinas mencionadas anteriormente, así como los desarrollos futuros que se prevé que se realicen en dichos campos. Para ello se realizará un estudio bibliográfico en el que se consultarán multitud de fuentes, lo que permitirá obtener un mayor conocimiento del estado actual y futuro de las disciplinas de estudio.

El cumplimiento de dichos objetivos se analizará en las conclusiones del presente documento, donde se evaluará el grado de cumplimiento de cada uno de ellos. En caso de no cumplir de forma completa con alguno de ellos, se propondrán una serie de medidas o actuaciones que permitan mejorar dicho grado de cumplimiento.

3. Estudio del estado del arte

Con el propósito de sentar unos antecedentes de los últimos avances realizados en los principales campos de estudio que constituyen este proyecto, se procede a estudiar el estado del arte de cada uno de ellos, citando a la bibliografía correspondiente en cada caso.

3.1. Control de aeronaves

Los algoritmos de guiado y control utilizados en una aeronave son un elemento fundamental para garantizar la seguridad y la operatividad de la misma. Tradicionalmente, dichos algoritmos se implementan en el sistema de Gestión de Vuelo, conocido como FMS por sus siglas en inglés (*Flight Management System*). Como es de esperar, existen multitud de filosofías de control aplicables a las aeronaves. Sin embargo, el presente estudio se centrará en enumerar algunas de las técnicas de control adaptativo por ser el tipo de control en el que se centra la investigación hoy en día. En las siguientes líneas se procederá a enumerar algunas de las más extendidas así como la bibliografía correspondiente.

- ***Gain Scheduling* o Planificación de Ganancias**: Esta técnica es una de las más extendidas en el control de aeronaves y goza de muchos años de uso en el sector aeronáutico. La técnica consiste en la creación de multitud de controladores lineales, ajustados cada uno de ellos para un punto de operación en concreto dentro de la envolvente de vuelo. El sistema se encarga de monitorizar una serie de variables (altitud y velocidad o número de Mach, por ejemplo) para obtener el punto de operación y a partir de ahí elegir el controlador correspondiente. De esta forma, si se consigue realizar un *mapeado* de los distintos puntos de funcionamiento dentro de la envolvente de vuelo, se puede conseguir un controlador que se adapte a las condiciones de operación de cada instante. Se trata, por tanto, de una de las formas más sencillas de implementar un control adaptativo. Los fundamentos teóricos de dicho esquema de control se pueden consultar en [1].

  Las técnicas de *Gain Scheduling* se vienen aplicando desde hace muchos años en aeronáutica debido a la relativa sencillez (en comparación con otros métodos más complejos) con la que se puede obtener un control adaptativo robusto [2]. En los últimos años están apareciendo multitud de nuevas técnicas de control adaptativo, sin embargo, esta técnica sigue gozando de cierta popularidad en el ámbito de la investigación y se siguen desarrollando estudios hoy en día tal y como se puede ver en las siguientes referencias. En [3, 4] se realizan sendos estudios de la estabilidad del sistema de *gain scheduling*. Además [3] profundiza sobre el caso de aeronaves con un comportamiento altamente no lineal y estudia la estabilidad tanto de los

controladores de cada uno de los puntos de operación como del sistema en general que gestiona dichos controladores. Además, se trata de un tema que suscita un cierto interés en el ámbito académico, un ejemplo de esto serían las Tesis [5–7].

- ***Adaptive pole placement***: Este método basa su efectividad en la modificación de la localización de los polos de la función de trasferencia utilizada de forma dinámica. De esta forma se consigue un sistema de control capaz de adaptarse a perturbaciones y a transitorios. Este tipo de control adaptativo no goza de tanta popularidad como el anterior hoy en día; sin embargo, se pueden encontrar numerosas referencias en la literatura de los últimos 20 años ([8–10], por nombrar algunos) y también algunos ejemplos más recientes aplicados a UAVs, como es el caso de [11].

- ***Iterative learning control* (ILC)**: Este método de control consiste en la utilización del error de la repetición anterior para ajustar las ganancias del controlador. Este método es de gran utilidad para procesos repetitivos y tradicionalmente se ha utilizado en el control de brazos robóticos industriales, donde se realiza una misma acción repetidas veces y se necesita un control muy fino. Sin embargo, en los últimos años se está empezando a aplicar este tipo de técnicas en pequeñas aeronaves no tripuladas, en especial en multi-rotores. En [12] se realiza un estudio de las distintas técnicas de control adaptativo y se opta por utilizar técnicas de ILC por su utilidad en el control de cuadricópteros durante despegues, aterrizajes y transiciones. Otro ejemplo sería el de [13], donde lo que se persigue es mejorar el comportamiento de un cuadricóptero al realizar maniobras agresivas.

- ***Model Identification Adaptive Controllers (MIAC)* y *Model Reference Adaptive Controllers (MRAC)*:** El método MIAC se basa en la utilización de un algoritmo de estimación de la función de transferencia del sistema que se recalcula en cada instante basándose en las medidas tomadas instantes antes. De esta forma se ajusta el controlador en cada instante, basándose en la función de transferencia obtenida. En cambio, los métodos MRAC hacen uso de un modelo dinámico inicial y ajustan y corrigen dicho modelo en cada instante de tiempo. El uso de estos tipos de control adaptativo se está convirtiendo en algo bastante extendido a medida que aumentan los recursos computacionales disponibles en los autopilotos, y no son extrañas las investigaciones que comparan las prestaciones de ambos ([14, 15]).

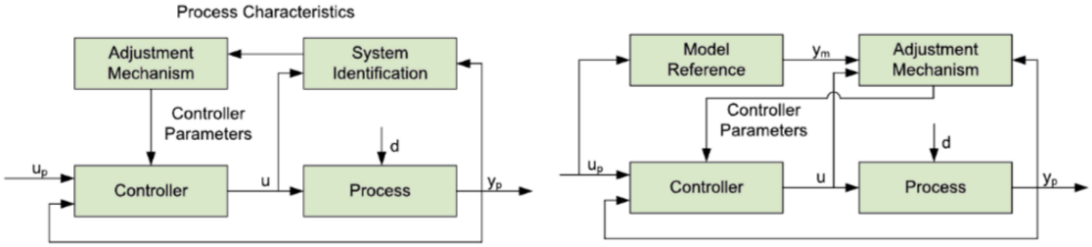


Figura 1.1: Esquema de guiado de un controlador MIAC (izqda) y un controlador MRAC (dcha). Fuente: [14]

El control de aeronaves es un campo de investigación en constante crecimiento y en el que se han realizado numerosos estudios e innovaciones en los últimos años. Prueba de esto son el número de publicaciones que se realizan al año relativas a control adaptativo, así como el número de equipos de investigación dedicados a dicha materia. Además, la integración de sistemas de control adaptativos en aeronaves no tripuladas es un tema de interés por parte de agencias como la NASA no sólo desde el punto de vista del diseño de aeronaves y sus controladores, sino también desde el punto de vista de su certificación [16].

3.2. Guiado de aeronaves

Las estrategias de guiado de aeronaves es otro de los campos en constante desarrollo y que, al igual que en el caso del control de aeronaves, ha visto como la aparición de los UAVs ha propiciado un desarrollo mucho más acusado en los últimos años. Las técnicas de guiado aplicadas a este tipo de aeronaves han evolucionado notablemente, pasando de guiados muy simples consistentes en el seguimiento de un rumbo constante a técnicas mucho más complejas. En este apartado el autor se centrará principalmente en las técnicas de guiado aplicadas a UAVs, debido a que las aeronaves no tripuladas son fuente de la mayor parte de los estudios realizados hoy en día en cuanto a técnicas de guiado.

En [17], se presenta una técnica de guiado muy novedosa para permitir el repostaje en vuelo de un UAV. El sistema se basa en la utilización de cámaras que graban a la aeronave que va a realizar la transferencia de combustible y que permiten obtener la posición relativa de dicha aeronave con respecto al UAV que va a realizar el *docking*. De forma similar, en [18] se utilizan cámaras estereoscópicas para calcular la posición relativa de un *target*, con la intención de interceptarlo o seguirlo. Una aplicación más simple es la presentada en [19], donde se presenta una estrategia de guiado que permite compensar los efectos del viento. La gestión de grupos de aeronaves no tripuladas es un aspecto de gran interés, en [20] se realiza un estudio para optimizar las trayectorias de grupos de UAVs de forma que se pueda recopilar la mayor información posible del terreno o de las regiones de interés.

La gran popularidad de la que gozan las aeronaves no tripuladas tanto en el sector profesional como en el de ocio, ha llevado a la creación de nuevas regulaciones en cuanto al guiado de estas aeronaves en el espacio aéreo. Prueba de esto es la regulación creada en 2015 por la *Civil Aviation Authority* británica [21].

3.3. Instrumentación de aeronaves

Los instrumentos de vuelo utilizados en las aeronaves tripuladas han experimentado un gran cambio en las últimas décadas. Como en la mayor parte de los sectores tecnológicos, se ha realizado una transición desde instrumentos analógicos a instrumentos integrados en pantallas digitales que permiten una mejora sustancial de la cantidad y la visibilidad de la información mostrada.

La constante mejora de los instrumentos de navegación es otro punto a tener en cuenta, donde el concepto de Navegación Basada en Prestaciones (PBN, *Performance-Based Navigation*) se muestra como el futuro de la navegación aérea. Una definición del concepto PBN se puede encontrar en el Manual de la OACI [22]. Sin embargo, la Tesis [23] presenta una definición elaborada y clara del concepto de PBN y su utilidad.

El TCAS (*Traffic Alert and Collision Avoidance System*) es sin duda uno de los instrumentos de gran importancia en los *cockpit* de hoy en día. La función de estos sistemas es la de alertar y evitar las posibles colisiones entre aeronaves, algo de vital importancia en los espacios aéreos actuales, donde la ocupación de los mismos es cada vez más alta. Este tipo de sistemas se basan en una naturaleza colaborativa, es decir, necesitan de la colaboración del resto de aeronaves del espacio aéreo, las cuales proporcionan los datos correspondientes de posición y velocidad, entre otros. Puesto que en el presente trabajo se ha implementado un TCAS de tipo I, se procederá a describir en mayor profundidad el sistema en secciones posteriores. En cuanto a los desarrollos presentes y futuros, cabe mencionar que el TCAS de tipo II es el que actualmente se utiliza en la gran mayoría de la aviación comercial. Este tipo de TCAS no solamente advierte del peligro de colisión, sino que da unas directrices al piloto sobre cómo evitar dicha colisión. Pese a tratarse de un sistema que lleva en funcionamiento numerosos años, se siguen desarrollando mejoras al mismo, tal y como se puede observar en [24, 25], donde se muestra información de los cambios en la regulación y en las capacidades del sistema.

Aunque el TCAS II sigue siendo utilizado y desarrollado hoy en día, la intención en un futuro es migrar hacia otro tipo de sistemas cooperativos llamados ADS-B (*Automatic Dependent Surveillance Broadcast*) basados en la distribución de la posición GPS y las trayectorias de las distintas aeronaves. Esta red de distribución la

componen tanto las aeronaves que se encuentran en el espacio aéreo como los controladores de tráfico aéreo. A partir de dichas posiciones y trayectorias y mediante una lógica muy similar a la empleada en el TCAS tradicional, se realizarían los cálculos correspondientes para evaluar el riesgo de colisión. Esta tecnología se encuentra actualmente en fases de desarrollo, sin embargo se prevé su entrada en servicio en aviación civil en los próximos años. En la tesis [26] se puede encontrar una definición extensa del sistema ADS-B así como un estudio de los beneficios que puede llegar a implicar su implantación. En 2010 la FAA publica en el CFR Part 19 [27] la regulación sobre la cual se rigen los sistemas ADS-B. Dos años antes, en 2008, la misma organización plantea un estudio para evaluar las ventajas de la utilización de sistemas ADS-B, para ello instala el sistema en 12 Boeing 747-400 de United Airlines y en 2010 empieza a probar dicho sistema en condiciones reales. En diciembre de 2015 se publica un informe que recopila los resultados [28]. En este estudio se puedo demostrar que los sistemas ADS-B pueden permitir un ahorro significativo de combustible al permitir una optimización mayor de las rutas. Sin embargo, también se hace hincapié en la necesidad de formación de los pilotos y ATCs para que el sistema se utilice correctamente y se pueda dar dicho ahorro de combustible.

3.4. Desarrollo de interfaces gráficas

La digitalización del *cockpit* y la gran cantidad de instrumentos que se han añadido en los últimos años han propiciado una preocupación por parte de las autoridades reguladoras en cuanto a la seguridad. La cantidad de información que se muestra en las cabinas de una aeronave comercial va en aumento a medida que evolucionan los instrumentos y se vuelven más complejos.

El desarrollo de las interfaces gráficas de los equipos de electrónica de consumo se rige por factores tales como la presentación clara y ordenada de la información, así como la consecución de un diseño que permita la mejor experiencia de usuario posible. Esto último implica que la interfaz con un mejor impacto visual y que puede gozar por tanto de una mejor aceptación por el público general, puede no ser la interfaz que mejor presente la información. Cuando se aplican éstos conceptos al sector de la aeronáutica, la prioridad deja de ser la experiencia de usuario y pasa a ser la seguridad operacional. Una buena interfaz gráfica para ser implementada en un *cockpit* debe de cumplir con una serie de requisitos que garanticen que su uso no va a interferir negativamente en la seguridad. El estudio de cómo puede afectar el diseño de una interfaz gráfica a la seguridad operacional se engloba dentro de la disciplina de factores humanos (*human factors*). En el capítulo 1 de [29] se define a los factores humanos como: *"Un amplio campo que examina la interacción entre personas, máquinas y el ambiente con el propósito de mejorar las prestaciones y reducir errores"*. Los factores humanos se centran en una gran cantidad de factores que pueden alterar la seguridad y las prestaciones, donde el desarrollo de interfaces gráficas coherentes, ordenadas y

seguras es clave en las aeronaves de hoy en día.

En la literatura se pueden encontrar diversos ejemplos del interés que suscitas estos temas en las entidades reguladoras de aviación civil, tanto en conferencias ([30]), como en la literatura ([29, 31]). Además, la comunidad científica también está interesada en profundizar en los desarrollos de las denominadas interfaces hombre-máquina (*Human-Machine Interface*, HMI). En [32] se muestra una introducción a los nuevos conceptos utilizados en factores humanos así como importancia para mejorar la seguridad en la automatización del vuelo. Sin embargo, gran parte de los estudios centrados en el desarrollo de interfaces hombre-máquina se centran hoy en día en las aeronaves no tripuladas. En [33] se estudian los problemas a resolver en las interfaces hombre-máquina de los sistemas FMS para posibilitar la inclusión de UAVs en el espacio aéreo no segregado. Mientras que en [34] se muestran una serie de recomendaciones para el desarrollo de una estación de tierra de control de UAVs de forma que se cumplan los requisitos de seguridad desde el punto de vista de factores humanos.

3.5. Simulación de vuelo

El desarrollo plataformas de simulación de vuelo fiables y completas se justifica en la gran cantidad de aplicaciones que pueden tener, desde entrenamiento de pilotos u operadores, hasta la validación de misiones en UAVs, llegando incluso a conectar autopilotos reales a un simulador de vuelo.

El uso de simuladores como plataforma de entrenamiento de pilotos es una actividad muy extendida hoy en día, ya que permite abaratar en gran medida el entrenamiento de los pilotos y mejorar al mismo tiempo la seguridad en el mismo. Actualmente todos los pilotos de aerolíneas comerciales pasan en algún momento de su entrenamiento por sesiones de pilotaje en el simulador. Entre las ventajas de estos simuladores se encuentran la versatilidad y la gran cantidad de situaciones que se pueden simular sin riesgo alguno: pérdida de un motor, condiciones de viento cruzado, mala visibilidad, etc. En la actualidad son muchas las compañías que se dedican al desarrollo de software o hardware para este tipo de simuladores, donde FlightSafety International (fabricante de simuladores para el Airbus A320 y Boeing 737, entre otros modelos [35]) o Aerosim (fabricante de simuladores para el Airbus A320 y Boeing 767 entre otros [36]) son algunos ejemplos.

En los últimos años, la combinación de simuladores de vuelo con el hardware de los autopilotos se ha convertido un recurso muy utilizado, sobretodo en el sector de los UAVs. Este tipo de técnicas se denominan *Hardware-in-the-Loop* (HIL). El interés en realizar técnicas de simulación se remonta años atrás, si bien hasta la proliferación de las aeronaves no tripuladas, no se había extendido tanto su uso para realizar ensayos

sobre autopilotos completos. Entre las ventajas del HIL se encuentran la facilidad con la que se pueden replicar condiciones de vuelo sobre hardware real, permitiendo así que se pueda ensayar tanto el mismo hardware como el software implementado. Esto permite obtener ventajas potenciales para los largos y complejos procesos de certificación. En [37] se realiza una introducción al concepto HIL y se comparan los resultados obtenidos en ensayos HIL utilizando un pequeño UAV y el simulador de código abierto FlightGear [1]. En [38] su utiliza X-Plane como simulador (el mismo simulador utilizado en el presente trabajo) se realzan las capacidades de dicha técnica para minimizar los riesgos intrínsecos a los test de vuelo de los UAV. Esto permite tener una mayor confianza en la configuración de la aeronave antes de volar, lo que permite aumentar considerablemente la seguridad y facilita enormemente el desarrollo de aeronaves no tripuladas, permitiendo realizar multitud de ensayos con HIL sin riesgo alguno. Esta línea de acción está siendo adoptada por las empresas del sector de las aeronaves no tripuladas, donde se encuentran incluso empresas españolas como Embention, quien realizó sus primeros tests HIL con X-Plane en 2014 y viene utilizando dicha técnica de forma ininterrumpida desde entonces [39].

4. Estructura de la memoria

Una vez introducido el trabajo, se procede a definir la estructura del proyecto en los capítulos siguientes.

El Capítulo 2 se centrará en el proceso de diseño seguido en este proyecto. Para ello se realizará primero una descripción del proyecto y de la arquitectura implementada en el sistema. Posteriormente se hablará de la metodología seguida en el diseño, tanto de desarrollo de software como de la propia implementación gráfica del *cockpit*. A continuación se describirá el modelo seguido para la realización del diseño y la gestión del proyecto. Seguidamente se definirán los requisitos del sistema, dividiéndolos según la naturaleza de los mismos (funcionales, no funcionales y características deseables). Finalmente se enumerarán las principales herramientas utilizadas en el desarrollo del proyecto.

El Capítulo 3 se describirá todo lo relativo a la implementación de los elementos gráficos de la interfaz, donde se hará una mención mucho más detallada de los instrumentos implementados. Para cada uno de los instrumentos implementados se realizará una descripción del mismo, de su utilidad y de su funcionamiento. Además, se hablará de la implementación del código de cada uno de los instrumentos que se incluyen en este trabajo.

El Capítulo 4 habla de las interfaces con los periféricos que se han implementado.

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

Se han implementado dos interfaces con periféricos: la interfaz con el simulador de vuelo X-Plane y la interfaz con la antena de tráfico aéreo. En este capítulo se procede a describir primero la interfaz y la razón por la que se ha implementado para, posteriormente, describir en mayor detalle cómo se ha implementado la interfaz.

El Capítulo 5 trata sobre el autopiloto implementado en este trabajo. Primero se describirá la arquitectura de dicho autopiloto para, posteriormente, pasar a definir los algoritmos de guiado y control que se han implementado. Finalmente se muestra la interfaz realizada para la gestión del autopiloto, describiendo sus principales funcionalidades.

El Capítulo 6 se analizarán los resultados obtenidos para ver cuáles han sido los requisitos que se han cumplido de forma satisfactoria y cuáles no se han podido cumplir o que son sensibles a mejora. Se identificarán las áreas de mejora del proyecto y se tratará de aportar posibles soluciones a las mismas.

Por último, en el Capítulo 7 se mostrarán las conclusiones extraídas durante la realización del presente proyecto. Analizando las mismas y proponiendo una serie de trabajos futuros que puedan complementar al proyecto.

Capítulo 2

Diseño del sistema

1. Descripción del proyecto

Tal y como se ha comentado en secciones anteriores (Sección 2 del Capítulo 1), la intención del presente trabajo es la de crear una plataforma integral de simulación que permita afianzar conceptos relacionados con multitud de contenidos íntimamente relacionados al sector aeronáutico. El principal fin de este trabajo es el de crear una plataforma que sea útil desde un punto de vista didáctico para poder ser utilizada en cursos de Ingeniería Aeronáutica. La intención es que la plataforma sea útil para la enseñanza de temas tan variados como la programación orientada a objetos, la instrumentación aeronáutica o conceptos de navegación, guiado y control.

Con lo objetivos planteados en mente, se decide crear dicha plataforma asumiendo que ya se tiene un motor de simulación ya que el desarrollo del mismo sería una tarea que excedería con creces el objetivo de este proyecto y para la que no se dispone de medios suficientes. Puesto que se va a crear una interfaz gráfica, es deseable que el motor de simulación utilizado carezca de interfaz gráfica o que, al menos, permita que ésta sea desactivada. En este sentido, se ha realizado un estudio de alternativas (Sección 1.1 del Capítulo 4) para elegir de forma razonada el motor de simulación que más se adapte a las necesidades de este trabajo.

Con esto, se procede a describir de forma general la arquitectura del sistema implementado.

1.1. Detalles y estructura

El presente proyecto consiste en la implementación de un panel de instrumentos gráfico, dotado de cierto realismo gracias al aspecto de cabina. Además, el sistema es capaz de interaccionar con un motor de simulación de vuelo (X-Plane en este caso) y con una antena que lee datos de tráfico aéreo en tiempo real. Esto, unido a la implementación de un autopiloto permite conseguir una plataforma integral de

simulación y ensayos que puede ser ejecutada en entornos Windows y Mac OS X.

La idea del autor ha sido explotar al máximo las capacidades de una de las primeras herramientas vistas en Java™: los iconos en los textos estáticos de Java™ *Swing* (JLabel) y un método para rotar imágenes por medio de la generación de una nueva moviendo píxel a píxel. Estas dos sencillas herramientas han servido para preparar 6 instrumentos de navegación básicos en una aeronave de aviación general:

- ADI - *Attitude Director Indicator*: Horizonte artificial.
- ALT - Altímetro: Indicador de altitud.
- ASI - *Airspeed Indicator*: Indicador de la velocidad.
- HI - *Heading Indicator*: Indicador del *heading* de la aeronave.
- T/S TC - Coordinador de giro.
- VSI - *Vertical Speed Indicator*: Indicador de la velocidad vertical.

Además de estos 6 instrumentos, se han implementado un *yoke* o *stick* de control de la aeronave, la palanca de gases o *throttle* y por último un TCAS (*Traffic alert and Collision Avoidance System*), un sistema de alerta y prevención de colisiones, este último con ayuda de la herramienta *Graphics* de Java™.

Con la intención de dotar a esta interfaz del mayor realismo y funcionalidad posibles, se ha implementado una conexión con un motor de simulación para la comunicación e información de la actitud de la aeronave, y una conexión con la antena o con un archivo de trazas, para incluir información de tráfico real en el TCAS.

Por último, con el objetivo de ampliar el alcance del proyecto y poder aplicar así multitud de conceptos de guiado, navegación y control afianzados durante el transcurso del master, se ha implementado un autopiloto. Dicho autopiloto, como se ha comentado anteriormente, será capaz de despegar la aeronave y realizar un procedimiento estándar de salida (SID) simplificado.

Lo comentado anteriormente el tipo de arquitectura a implementar. Dicha arquitectura se muestra de forma esquemática en la Figura 2.1.

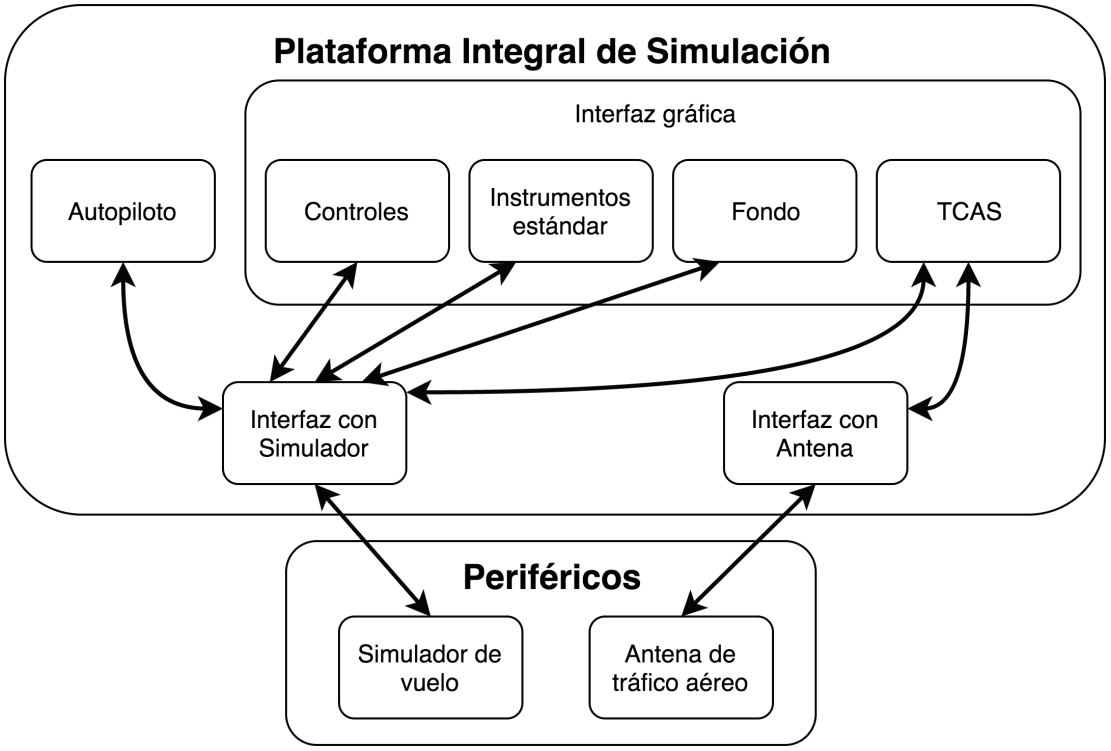


Figura 2.1: Esquema de la arquitectura del sistema

De esta forma, el trabajo se puede estructurar en cinco grandes bloques:

- Interfaz gráfica. Generación de elementos.
- Comunicación con el motor de simulación.
- Comunicación con la antena.
- Interpretación de información de la aeronave tanto en envío como en recepción. Animación de instrumentos correlacionada con información numérica.
- Implementación de un autopiloto con sus distintas fases de vuelo.

2. Metodología de diseño

2.1. Metodología del diseño de software

La metodología empleada en el diseño de software para sistemas de uso aeronáutico es muy estricta y rigurosa, con el objetivo de garantizar en todo momento la mayor seguridad posible. Sin embargo, como es lógico, los objetivos de seguridad que debe cumplir un software concreto serán más o menos exigentes según la criticidad del componente. Por tanto, el software de un sistema crítico para la seguridad del vuelo como un autopiloto tendrá que cumplir unos requisitos mucho más exigentes que un sistema cuyo fallo no afecte a la seguridad de la aeronave (como por ejemplo el

software del sistema de entretenimiento para los pasajeros).

La consideraciones a tener en cuenta en el diseño de software de uso aeronáutico para su certificación vienen recogidas en el estándar DO-178C [40]. Dicho estándar es utilizado tanto por la FAA como por la EASA para la certificación de software embarcado. La normativa DO-178 se ha utilizado en el sector aeronáutico desde hace más de 20 años. La tercera revisión a dicha normativa (revisión C) se publicó en enero de 2012. El estándar DO-178C define 5 niveles de criticidad distintos, especificados como *Design Assurance Level* (DAL). Los niveles DAL se clasifican en función del efecto en la seguridad que tendría un eventual fallo del sistema, utilizando las letras de la A (fallo catastrófico) hasta la E (sin consecuencias de seguridad). Estos niveles, que se pueden observar en la Tabla 2.1, llevan asociados una serie de objetivos a cumplir, que son más numerosos y restrictivos a medida que aumenta el nivel de criticidad. Además de garantizar el cumplimiento de ciertos objetivos, también se pide que se satisfagan algunos de ellos de forma independiente, lo que implica que la verificación del cumplimiento de un objetivo concreto ha de ser llevada a cabo por una persona u organización distinta a la creadora del código.

Tabla 2.1: Niveles de criticidad según el estándar DO-178C

Nivel (DAL)	**Consecuencias del fallo**	**Objetivos**	**Con independencia**
A	*Catastrophic*	71	33
B	*Hazardous*	69	21
C	*Major*	62	8
D	*Minor*	26	5
E	*No Safety Effect*	0	0

Siguiendo la estructura de diseño marcada por la normativa DO-178C se plantean una requerimientos que ha de cumplir el software implementado. Dichos requerimientos se enumeran en la Sección 4.

Puesto que se trata de un sistema no embarcado, no sería necesario garantizar las

objetivos estipulados en el estándar DO-178C. Sin embargo, debido a la capacidad de crecimiento que puede tener la plataforma en un futuro para el desarrollo de autopilotos y de sistemas de evitación de colisiones como el TCAS, es interesante tomar en consideración el estándar. De esta forma, parte del código realizado podría llegar a ser implementado en una aeronave real tras haber verificado su coherencia con la norma DO-178C.

2.2. Metodología del diseño del cockpit

El cockpit implementado en la aeronave se basa en el cockpit de una Cessna 172 Skyhawk, una de las aeronaves de aviación general más conocidas y la aeronave más fabricada de la historia. Al basar el diseño de la cabina en el de una aeronave tan extendida, se garantiza el cumplimiento de multitud de requisitos operacionales relacionados con los factores humanos.

La cabina ha sido modificada convenientemente para añadir el TCAS en la zona central de la misma, así como los mandos correspondientes al *yoke* y a la palanca de gases, los cuales se han implementado en la zona derecha del habitáculo. Para la realización del presente proyecto se ha decidido no implementar todos los instrumentos de la cabina debido a que eso implicaría dotarlos de un tamaño demasiado reducido como para apreciarlos correctamente en la pantalla de un ordenador común. Por tanto, se ha optado por implementar únicamente el TCAS y los 6 instrumentos comúnmente conocidos como *standard six*. En el capítulo correspondiente a la implementación del sistema se analizará en mayor profundidad la utilidad de cada uno de los instrumentos implementados. Las Figuras 2.2a y 2.2b muestran el cockpit estándar de la Cessna 172 y el cockpit utilizado en el presente proyecto, respectivamente. Como se puede observar, ambos son similares en cuando al concepto y la distribución de los instrumentos, lo que dota de un mayor realismo a la solución implementada.

(a) Cabina estándar de una Cessna 172

(b) Distribución de la cabina implementada

Figura 2.2: Comparativa del cockpit de una Cessna 172 y el cockpit implementado

3. Modelo de desarrollo de proyecto

Existe una gran variedad de modelos de desarrollo de proyecto, cada uno con sus ventajas y sus inconvenientes. Para dotar al proyecto de una mayor entidad y ser coherente con la metodología de diseño expuesta en la sección anterior, se ha decidido seguir un modelo de desarrollo de proyecto muy utilizado en la industria aeroespacial. Dicho modelo es el conocido como Método V (*V-Model* en inglés).

3.1. Método V (V-Model)

El método V define un procedimiento de desarrollo uniforme. Dicho modelo se basa en una representación gráfica del ciclo de vida del sistema, en el que se muestran las distintas etapas de diseño. Este modelo se suele representar mediante un gráfico de una V. En el lado izquierdo de dicha V se encuentra la fase conocida como *Validación*, en la que se identifican las distintas necesidades del sistema y se crea la especificación del sistema en base a unos requisitos. La parte baja de la V corresponde a la propia implementación del sistema. Por último, la parte derecha de la V hace referencia a la fase de *Verificación*, en la que se realizan todos los test pertinentes para asegurar el cumplimiento de los requisitos. Lo descrito anteriormente puede observarse en la Figura 2.3.

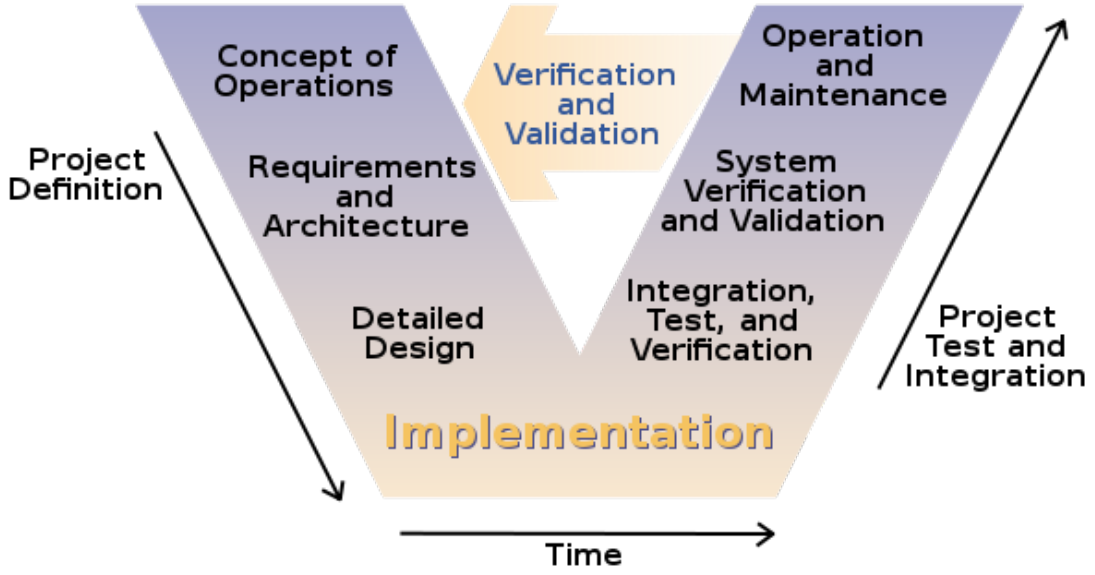


Figura 2.3: Esquema conceptual del Método V (V-Model). Fuente:[1]

A pesar de que la figura anterior muestra el proceso como un procedimiento totalmente lineal, cabe destacar que en un proceso real de diseño nunca se da tal caso. Es común, por tanto, que se realicen distintas iteraciones a lo largo del proceso de

[1] https://en.wikipedia.org/wiki/File:Systems_Engineering_Process_II.svg

diseño, volviendo en algunos casos a fases anteriores de la V.

El método V es un método muy utilizado desde hace años en la industria aeronáutica. De hecho, es el método que se utiliza actualmente en las empresas del grupo Airbus tanto para el diseño de software como para la integración de sistemas. La razón de la popularidad de esta metodología de diseño es que su aplicación permite desarrollar sistemas de una forma muy segura y ordenada, obteniendo unos altos estándares de calidad. Además, durante la realización de este proceso de diseño, necesariamente se genera una serie de documentación (especificaciones, documentos de control de interfaces, documentos de procedimientos de test, etc) que es de gran utilidad en el proceso de certificación del sistema ante las autoridades competentes.

4. Requerimientos del sistema

Una de las parte de un proceso de desarrollo de software es la definición de los requisitos. Una buena definición de los requisitos es fundamental para el desarrollo satisfactorio de un software. Una mala definición de los requisitos o un mal entendimiento de los mismos puede derivar en grandes problemas en el desarrollo, especialmente si el proyecto es de gran magnitud y existe un número importante de personas implicadas en el desarrollo. Para una mejor compresión del procedimiento de desarrollo de software en general y de la definición de los requisitos en particular, se insta al lector a consultar la referencia [41].

Existen diversos tipos de requisitos que se deben de especificar; el presente trabajo definirá requisitos de los siguientes tipos:

- **Requisitos funcionales:** Son aquellos requisitos que definen las funcionalidades requeridas al software. Son por tanto características de funcionalidad necesarias para el sistema.
- **Requisitos no funcionales:** Son todos aquellos requisitos que no se encuentran en la categoría anterior. Se trata de características implícitas del software sobre las cuales los usuarios pueden realizar ciertas suposiciones. En este apartado se incluyen requisitos de calidad, facilidad de uso, mantenimiento, etc.
- **Características deseables o "*nice to have*":** Este grupo engloba todas aquellas características que no forman parte en sí mismo de los requisitos del sistema y que no necesariamente han de ser implementadas. Sin embargo, se consideran características que pueden aportar valor añadido al proyecto y que se desean implementar en la medida de los posible.

4.1. Nomenclatura de los requisitos

Para dotar de una mayor claridad al análisis de requisitos que se realizará en las secciones finales del presente proyecto, se ha decidido implementar una nomenclatura específica y única para distinguir a cada requisito de forma unívoca.

Los requisitos presentados se engloban en dos categorías distintas (funcionales y no funcionales) a las que hay que añadir una categoría extra compuesta por las características deseables del sistema (*nice to have*). Para poder distinguir de forma rápida y eficaz, los requisitos se enumerarán siguiendo la nomenclatura descrita en las siguientes líneas:

- **Requisitos funcionales:** Se definirán mediante el indicador **FR-x**, donde FR son las siglas de *Functional Requirement* y "x" es un número identificador único para cada requisito.
- **Requisitos no funcionales:** En este caso se utilizará el identificador **NFR-x**, donde NFR son las siglas de *Non-Functional Requirement* y "x" es nuevamente un número identificador único.
- **Características deseables o "*nice to have*":** Se utilizará la nomenclatura **NTH-x**, donde NTH son las siglas de *Nice To Have* y "x" es el número identificador.

4.2. Requisitos funcionales

El sistema debe cumplir los requisitos funcionales mostrados a continuación:

FR-1 El sistema ha de poder ser ejecutado en distintas plataformas y sistemas operativos.

FR-2 El sistema ha de tener una interfaz gráfica que permita su manejo.

FR-3 El sistema ha de incluir los siguientes instrumentos de forma funcional:

- ASI
- ADI
- ALT
- T/S
- HI
- VSI
- TCAS

FR-4 Los instrumentos del sistema deben de alimentarse de datos simulados para su funcionamiento.

FR-5 Los datos simulados han de obtenerse en tiempo real de un simulador de vuelo.

FR-6 El sistema ha de poder enviar acciones de control al simulador y actuar sobre el mismo.

FR-7 El TCAS debe de ser capaz de nutrirse de datos de tráfico aéreo real.

FR-8 El TCAS debe de poder estimar el riesgo de colisión siguiendo la metodología usada en instrumentos reales.

FR-9 El TCAS debe de ser, como mínimo, de tipo I.

FR-10 El sistema ha de implementar un autopiloto.

FR-11 El autopiloto debe de ser capaz de controlar, como mínimo un modelo de aeronave.

FR-12 El autopiloto debe de ser capaz, como mínimo, de controlar una aeronave desde la carrera de despegue hasta la de crucero.

FR-13 El autopiloto ha de ser configurable en sus distintas fases.

FR-14 La configuración del autopiloto ha de permitir la inclusión de salidas estándar tipo SID.

FR-15 El sistema debe permitir el control manual de la aeronave en vuelo sin necesidad de que el usuario interactúe con el simulador de vuelo.

4.3. Requisitos no funcionales

Los requisitos no funcionales del sistema se muestran a continuación:

NFR-1 El sistema ha de ser de utilidad para la docencia de diseño de software orientado a objetos en aplicaciones de ingeniería aeronáutica.

NFR-2 El código ha de implementarse mediante un lenguaje orientado a objetos.

NFR-3 El código se ha de implementar de forma modular.

NFR-4 El sistema debe permitir la mejora de sus funcionalidades o la inclusión de nuevas características de forma sencilla.

NFR-5 El sistema debe de ser sencillo de utilizar.

NFR-6 La configuración de la antena o el autopiloto debe de ser simple y debe requerir una una acción mínima por parte del usuario.

4.4. Características deseables o "*nice to have*"

A continuación se enumeran las características deseables del sistema. Como se ha comentado con anterioridad, dichas características presentan un carácter optativo en cuanto su implementación, sin embargo, conviene mencionarlas e intentar conseguir, en la medida de lo posible, cumplir con la mayoría de ellas.

NTH-1 El sistema debe de poder ejecutarse en un ordenador con recursos limitados.

NTH-2 El código Java debe de estar lo más optimizado posible.

NTH-3 El simulador de vuelo debe de consumir el mínimo de recursos posible. A ser posible se utilizará un motor de simulación sin interfaz gráfica.

NTH-4 A ser posible, se dotará al autopiloto de la capacidad de controlar distintos modelos de aeronaves.

NTH-5 El autopiloto debe de poder configurarse mediante archivos de configuración generados con anterioridad.

NTH-6 Los datos de tráfico aéreo deben de poder simularse mediante un archivo de trazas generado con anterioridad para casos en los que no esté disponible la antena.

NTH-7 Los tipos de misión han de ser configurables en el autopiloto.

NTH-8 El autopiloto debe de poder gestionar una misión completa, desde el despegue hasta el aterrizaje.

5. Herramientas utilizadas en el desarrollo

En esta sección se procede a describir las herramientas informáticas necesarias para el desarrollo del presente proyecto. Cabe destacar que el razonamiento seguido en la elección de ciertas herramientas software será descrito en los siguientes capítulos.

5.1. Java™ - Entorno NetBeans

El lenguaje de programación Java™ en el entorno NetBeans es la competencia instrumental específica de la asignatura que enmarca el presente proyecto, por lo que compone la herramienta principal utilizada en el presente proyecto.

Java™ es un lenguaje de programación de propósito general, concurrente, orientado a objetos que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como *WORA*, o *write once, run anywhere*), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser

recompilado para correr en otra. Java™ es, a partir de 2012, uno de los lenguajes de programación más populares en uso, particularmente para aplicaciones de cliente-servidor de web, con unos 10 millones de usuarios reportados.

El lenguaje de programación Java™ fue originalmente desarrollado por James Gosling de Sun® Microsystems (la cual fue adquirida por la compañía Oracle®) y publicado en 1995 como un componente fundamental de la plataforma Java™ de Sun® Microsystems. Su sintaxis deriva en gran medida de C y C++, pero tiene menos utilidades de bajo nivel que cualquiera de ellos. Las aplicaciones de Java son generalmente compiladas a bytecode (clase Java) que puede ejecutarse en cualquier máquina virtual Java™ (JVM) sin importar la arquitectura de la computadora subyacente.

La interfaz del entorno NetBeans se muestra en la Figura 2.4. NetBeans es un entorno de desarrollo integrado libre, desarrollado principalmente para el lenguaje de programación Java™. Existe además un número importante de módulos para extenderlo. NetBeans IDE es un producto libre y gratuito sin restricciones de uso fundado por Sun® MicroSystems.

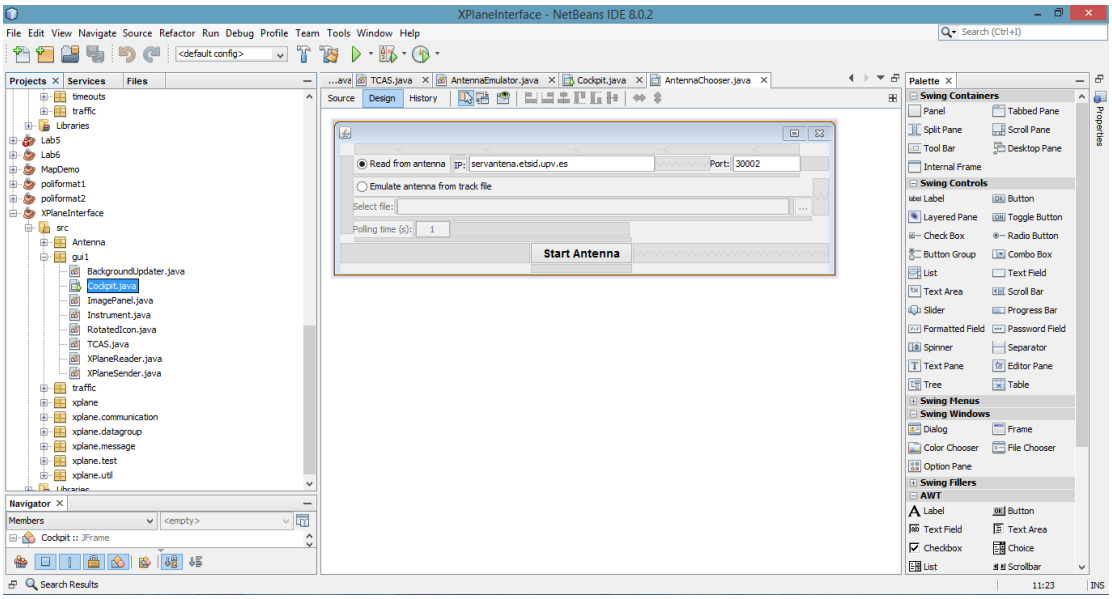


Figura 2.4: Entorno gráfico de NetBeans

5.2. Motor de simulación - X-Plane

Otra de las grandes herramientas necesarias para el desarrollo del proyecto es un motor de simulación de vuelo. Existen numerosas alternativas disponibles y que cumplen con los requisitos impuestos en este proyecto. La intención inicial era la de elegir un motor de simulación sin interfaz gráfica, sin embargo, existen otras variables que pueden afectar a la elección del mejor motor de simulación para este trabajo. Es

por esto que se ha realizado un análisis de dichas alternativas en la Sección 1.1 del Capítulo 4.

El motor de simulación elegido en dicho tras la realización de dicho estudio es el simulador de vuelo X-Plane, con el cual se establece conexión para la recepción y envío de información desde la interfaz implementada en el proyecto.

A continuación se procede a describir brevemente las principales características de dicho simulador.

X-Plane, creado por Austin Meyer, es un simulador de vuelo de aeronaves civiles. Se ha establecido como uno de los principales simuladores de vuelo que son capaces de competir el simulador Flight Simulator de Microsoft®. Según Austin Meyer, el simulador está dotado de una gran precisión, ya que se basa en calcular el efecto del flujo de aire sobre las superficies de los aviones simulados. La clave del realismo de la física de vuelo de X-Plane es la creación de un túnel de viento virtual alrededor del avión, consiguiendo así efectos parecidos a los reales.

Dado que el propósito de este simulador es ofrecer una experiencia de vuelo lo más realista posible, cuenta con una amplia gama de aviones simulados, desde los más sencillos hasta los grandes reactores. Además, es capaz de generar una recreación del planeta tierra con sus accidentes geográficos y alrededor de 18.000 aeropuertos, aeródromos y helipuertos, así como portaaviones en los que realizar sus prácticas de vuelo.

Todas estas características le han servido para que la Administración Federal de Aviación (FAA) de Estados Unidos autorice su uso, junto con hardware específico, para el entrenamiento de pilotos de vuelo instrumental.

5.3. Otro software

Además, para la consecución de este proyecto, se ha hecho uso de otro software como:

- Matlab®
- Adobe™ Photoshop®
- AutoCAD®

Capítulo 3

Implementación de la interfaz gráfica y los instrumentos

1. Interfaz gráfica: *cockpit*. Generación de instrumentos

La interfaz gráfica consta de cuatro grandes elementos:

- **Panel de instrumentos**
- **Instrumentos**
- **Fondo**
- **Controles: Mandos de vuelo y palanca de gases**

Como ya se ha comentado, en este proyecto se decidió exprimir toda la capacidad de dos herramientas muy simples: los iconos de la clase *swing JLabel* y el método `rotateIcon`, que consiste en re-ubicar los píxeles de una imagen, uno a uno, para rotar la misma. Esta forma de implementación requiere un esfuerzo computacional grande y es posible que la implementación con la librería `Graphics2D` de Java™ diese mejor resultado, sin embargo, gran parte del mérito de este proyecto reside en la sencillez de las herramientas utilizadas en la implementación.

Todos estos elementos se cargan por medio del uso del código puro de Java™. Se ha generado un *layeredPane* sobre el que se van creando diversas capas en las que se insertan los distintos elementos gráficos.

1.1. Panel de instrumentos

El panel de instrumentos se extrajo del *cockpit* original de la Cessna C-172 de X-Plane y se realizaron algunas modificaciones para mejorar la visualización de los

instrumentos que se van a implementar. Con este fondo, se le consigue dar un aspecto más realista a la aplicación. La Figura 3.1 muestra dicha imagen.

Figura 3.1: Panel de instrumentos

Como se puede observar hay 6 huecos circulares, donde se situarán los instrumentos, que de izquierda a derecha y de arriba a abajo son: ASI, ADI, ALT, T/S, HI y VSI. El hueco cuadrado corresponde al panel del TCAS.

1.1.1. Localización de los instrumentos

Con el objetivo de promover la modularidad del código y facilitar su adaptabilidad a nuevas disposiciones de la cabina o a nuevas implementaciones de instrumentos, se ha decidido no incluir directamente en el código los parámetros utilizados para la generación del *cockpit*. De esta forma, se ha optado por crear un archivo de configuración, llamado `guiDataFile.txt`, donde se guardará toda la información relevante para la distribución de los objetos de la cabina.

Los instrumentos se añaden al *layeredPane* mediante el método `loadInstruments(String file)`, el cual lee el archivo `guiDataFile.txt`. Este archivo de texto está escrito de una forma interpretable por ese método, de forma que genera el instrumento en la localización indicada, un instrumento que a su vez será de la clase creada `Instrument`, y que estará compuesto de las distintas imágenes cuyas rutas se indican.

El formato se puede ver en la Figura 3.2, donde por ejemplo, en la línea 19 se crea el nuevo instrumento *NewInstrument ASI*, en la localización $\{x, y\} = \{167, 323\}$ y se compone de los *3 JLabels asi_face, asi_hand* y *asi_case*, cuyos respectivos iconos o imágenes se encuentran en las rutas *images/asi/asi_face.png* etcétera.

```
1  ////////////////////////////////////////////////////////////////////////////////
2  //                          X-PLANE GUI DATA FILE                             //
3  // This file contains the information required by the X-Plane GUI to find and //
4  // place the icons.                                                           //
5  //                                                                            //
6  // Author: Lucas Peris Lozano                                                 //
7  //                                                                            //
8  //    Date: 10/02/16                                                          //
9  ////////////////////////////////////////////////////////////////////////////////
10 // Data format
11 // Name     locationX       locationY      Path
12
13 NewInstrument BACKGROUND -7417 -621 1
14 background images/background.jpg
15 // Cockpit
16 cockpit images/cockpit_3.png 0 0
17 NewInstrument ASI 167 323 3
18 asi_face images/asi/asi_face.png
19 asi_hand images/asi/asi_hand.png
20 asi_case images/asi/asi_case.png
21 NewInstrument ADI 322 323 4
22 adi_back images/adi/adi_back.png
23 adi_face images/adi/adi_face.png
24 adi_ring images/adi/adi_ring.png
25 adi_case images/adi/adi_case.png
```

Figura 3.2: Aspecto del fichero `guiDataFile.txt`

Se ha tomado esta decisión ya que es una manera fácil de generar otros paneles de instrumentos en otro orden o con otra geometría o incluso otros instrumentos, todo ello sin necesidad de alterar el código fuente del programa, ya que solo es necesario modificar el fichero `guiDataFile.txt`.

El aspecto final del panel de instrumentos con todos los elementos cargados es el que muestra la Figura 3.3. Como se puede observar incluye el *stick* o *yoke* y la palanca de gases o *throttle*, y ambos permiten la interacción con el usuario y se mueven de una forma bastante realista, incluso el *stick*, con el que se visualiza bastante bien la sensación de aplicar *roll* e incluso *pitch*.

Figura 3.3: Panel de instrumentos `Cockpit`

Hay que recalcar que todos los elementos de los instrumentos son funcionales: desde las distintas agujas, reglas e incluso el regulador de la presión del altímetro y la pequeña esfera del coordinador de giro.

1.2. Fondo

También se ha querido mencionar por separado el fondo, el cual consiste en una imagen panorámica de 360º. Se ha querido implementar para dar sensación de movimiento a la cabina, ya que este fondo se desplazará de acuerdo a la actitud que tenga el avión en cada momento.

La imagen utilizada para el fondo se muestra en la Figura 3.4.

Figura 3.4: Imagen utilizada para el fondo de la interfaz gráfica.

1.3. Controles: Mandos de vuelo y palanca de gases

A pesar de que se ha incluido un autopiloto capaz de despegar la aeronave desde la pista y llevarlo hasta la fase de crucero, se se ha considerado útil añadir la capacidad

de controlar de forma manual la aeronave mediante la interfaz.

Debido a la simplicidad de la aeronave elegida, es posible controlar la misma únicamente mediante los mandos de vuelo básicos (conocido como *yoke*) y una palanca de gases. Con estos dos instrumentos es posible controlar la aeronave en vuelo sin problema alguno. Cabe destacar que debido a que la imagen de fondo es siempre la misma, no se ha considerado útil implementar un control manual del rudder, ya que no sería posible despegar la aeronave desde la pista de forma manual utilizando únicamente la interfaz presentada en este proyecto. Además, el uso del rudder en vuelo para coordinar el giro no sería posible, ya que con el ratón únicamente podríamos actuar sobre un instrumento a la vez, que en este caso sería el del *yoke*.

Los controles implementados se pueden observar en la Figura 3.5.

Figura 3.5: Controles implementados en la interfaz gráfica.

1.3.1. Uso de los mandos de vuelo (yoke)

Los mandos de vuelo implementados están basados en los mandos reales de la aeronave Cessna 175 Skyhawk. El control de los mismos utiliza el mismo mecanismo que el utilizado en el simulador X-Plane. Dicho control se basa en el utilizado por otros simuladores como X-Plane o Microsoft Flight Simulator.

Para actuar sobre los controles, es suficiente con hacer click en la zona en la que se

encuentran los mandos de vuelo y mover el ratón en la dirección deseada. Las direcciones en las que se puede mover el ratón y su efecto se resumen en la Tabla 3.1 y en la Figura 3.6.

Tabla 3.1: Control manual de los mandos de vuelo mediante el ratón

Acción del ratón (arrastre)	**Acción sobre la aeronave**
Arrastre hacia arriba ↑	Cabeceo hacia abajo
Arrastre hacia abajo ↓	Cabeceo hacia arriba
Arrastre hacia la izquierda ←	Alabeo hacia la izquierda
Arrastre hacia la derecha →	Alabeo hacia la derecha

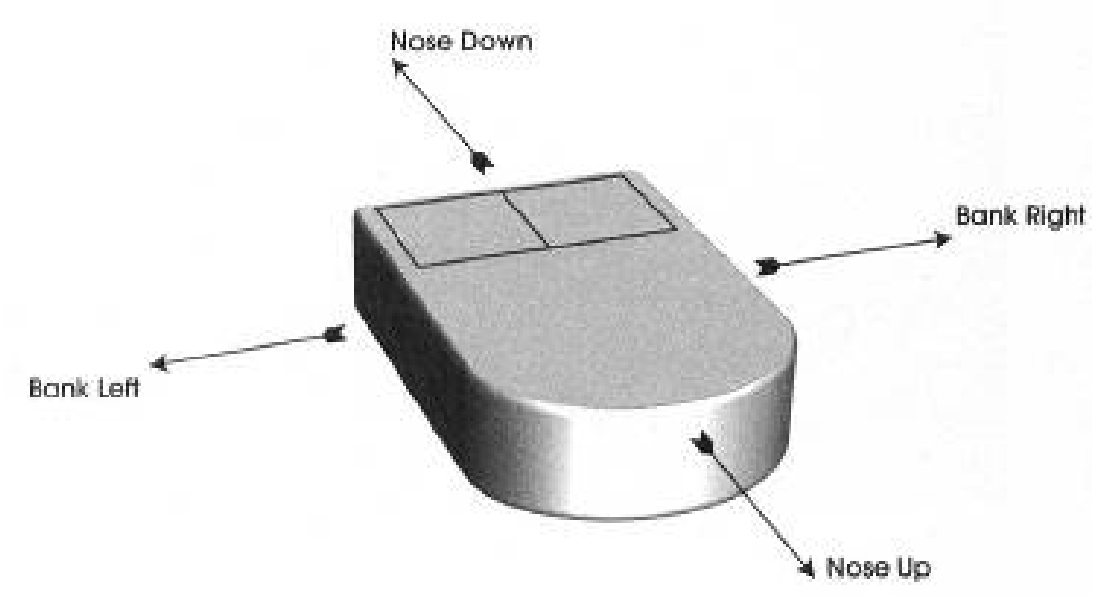


Figura 3.6: Controles implementados en la interfaz gráfica.

Cabe destacar que los mandos de vuelo se mueven en la dirección indicada por el ratón, rotándose los mismos cuando se pretende girar la aeronave y trasladándose en la dirección vertical cuando se pretende realizar un movimiento de cabeceo.

1.3.2. Uso de la palanca de gases

La palanca de gases se puede utilizar simplemente arrastrando la misma hasta la posición deseada. La palanca de gases elegida no es similar en ningún modo a la real utilizada en la Cessna 172, sin embargo, se ha optado por el diseño mostrado en la Figura 3.5 debido a que es mucho más fácil de manejar con el ratón.

2. Implementación de los instrumentos

La generación de instrumentos, como se ha mencionado previamente, se realiza mediante código, sin utilizar el entorno gráfico de NetBeans. Dichos instrumentos están separados por capas en distintos ficheros de imagen, los cuales pueden encontrarse con relativa facilidad en la web o en los propios directorios de X-Plane. Esta Generación por capas es algo muy común en el diseño de todo tipo de interfaces gráficas y de hecho es la aproximación que se ha realizado también desde el simulador X-Plane para la implementación de los instrumentos.

El presente proyecto implementa en la cabina de la interfaz gráfica los siguientes instrumentos: ASI, ADI, ALT, T/S, HI, VSI y TCAS. La representación gráfica de dichos instrumentos, a excepción del TCAS, se ha realizado únicamente mediante la superposición de imágenes que pueden ser rotadas o movidas. En los siguientes párrafos se procede a determinar tanto la utilidad de dichos instrumentos como el camino llevado a cabo por el autor para la implementación de los mismos.

2.1. Clase `instrument` y sus métodos principales

Para llevar a cabo la implementación de los instrumentos, se ha creado la clase `insturment`, la cual engloba todos los tipos de instrumento generados en el presente trabajo, a excepción del TCAS. Debido a la complejidad del TCAS, se ha decidido implementar dicho instrumento mediante una clase dedicada al mismo, sobre la que se hablará en mayor profundidad en la Subsección 2.8.

La clase `instrument` tiene multitud de parámetros de uso interno, entre los que destacan los siguientes:

- Un *Hashmap* en el que se contienen las *JLabel* que componen las distintas capas del instrumento.
- Un nombre identificativo del instrumento (ADI, ALT, etc). Según el nombre con el que se incialice la clase, los distintos métodos que se implementan en la misma tendrán un comportamiento u otro. Adaptándose así al comportamiento deseado para cada uno de los instrumentos.

Esta clase, dedicada a la creación y gestión de los instrumentos, tiene una serie de métodos específicos para cada tipo de instrumento, de los que se hablará en los siguientes apartados para cada uno de los instrumentos implementados. No obstante, en este apartado se considera oportuno presentar los dos métodos principales de la clase `insturment`, sobre los que se apoyan el resto de instrumentos para cambiar su disposición en la interfz y mostrar los datos correspondientes al estado actual de la aeronave. Dichos métodos son:

- `translateLabel`
- `rotateLabel`

2.1.1. Método `translateLabel`

El método `translateLabel` tiene como función desplazar un *JLabel* un determinado número de píxeles en horizontal y en vertical. Este método tiene tres parámetros de entrada:

- El *JLabel* que se pretende desplazar.
- Los píxeles que se debe desplazar el *JLabel* en horizontal, siendo positivos si el desplazamiento es hacia la derecha.
- Los píxeles que se debe desplazar el *JLabel* en vertical, siendo positivos si el desplazamiento es hacia la debajo.
- La posición teórica anterior del *JLabel* (parámetro opcional).

El funcionamiento de este método es muy sencillo, ya que únicamente se trata de cambiar el parámetro `location` del *JLabel* que se pretende desplazar.

Adicionalmente, se ha decidido implementar una serie de filtros lógicos que permitan deducir si es necesario mover el *JLabel*. La intención de interpretar si se debe mover un *JLabel* o si se debe mantener constante se debe a la intención de optimizar el funcionamiento de la interfaz gráfica implementada. Puesto que la frecuencia de refresco de la toma datos del simulador es relativamente alta (20 Hz), muchos datos pueden cambiar de forma muy reducida entre una medida y la siguiente. Esto da lugar a numerosas ocasiones en la se que se demanda que un instrumento mueva un elemento (uno o varios *JLabels*) apenas unos pocos píxeles. En estos casos, se estarían moviendo unas partes de la interfaz de una forma totalmente imperceptible para el usuario, con lo que se estarían consumiendo recursos de forma totalmente innecesaria.

Para evitar el desplazamiento innecesario de elementos de la interfaz, se ha decidido añadir una tolerancia de 2 píxeles para cada movimiento. De esta forma, cuando se llame a la función `translateLabel`, ésta sólo realizará la translación

siempre que esta sea mayor de 2 píxeles en alguna de las dos direcciones (vertical u horizontal). Con esta sencilla comprobación se puede reducir de forma notable el número de veces que se mueve un *Label* sin que el usuario perciba diferencia alguna. El valor de la tolerancia se ha ajustado tras varias pruebas, de forma que los movimientos de todos los objetos sean fluidos y que el usuario no sea consciente de la existencia de dicha tolerancia.

Como parámetro opcional se incluye en algunos casos la posición teórica anterior del *JLabel*, esta posición hace referencia a la localización que el *JLabel* debería de tener antes de realizar el proceso de translación. La razón de ser de este parámetro es que algunos instrumentos se han implementado de forma que no se guarda como parámetro el valor en sí de la magnitud que miden, sino que simplemente de mueven los indicadores a la posición correspondiente. En estos casos, estos instrumentos se han implementado de forma incremental, por lo que para mostrar el valor correcto, es necesario saber la localización que el instrumento debería tener. Al añadir la tolerancia a la translación, puede darse el caso de que un instrumento de este tipo deba desplazarse pocos píxeles y se decida no mostrar ese desplazamiento en la interfaz gráfica. Sin embargo, en estos casos es necesario guardar en una variable la localización que debería de tener el instrumento, ya que el hecho de que no se actualice la interfaz no significa que no se deba actualizar el instrumento en sí. En estos casos, el desplazamiento se realizará añadiendo los valores correspondientes de desplazamiento horizontal y vertical a la localización que debería de tener el instrumento.

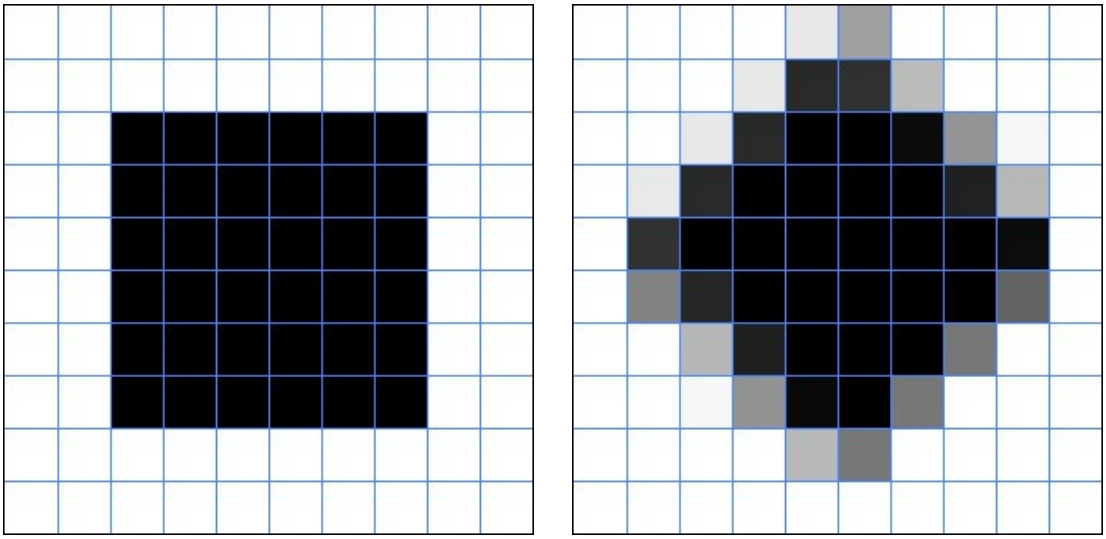
2.1.2. Método `rotateLabel`

El método `rotateLabel` tiene como función rotar un *JLabel* un determinado número de grados. Este método tiene tres parámetros de entrada:

- El *JLabel* que se pretende rotar.
- El número de grados que se pretende rotar el instrumento (en grados), tomando como positivo el sentido a favor de las agujas del reloj.
- Un parámetro booleano que defina si se pretende actualizar una variable de la clase conocida como `currentAngle`.

El método `rotateLabel` rota la imagen que se guarda como icono en el *JLabel* en cuestión. Para rotar un determinado número de grados dicha imagen, se hace uso de la clase `RotatedIcon`. Esta clase ha sido obtenida de las sesiones prácticas de la asignatura *Sistemas de Gestión de Vuelo por Computador* [42]. La clase `RotatedIcon` basa su funcionamiento en recalcular las nuevas coordenadas de cada uno de los píxeles para obtener una imagen que sea el resultado de la rotación deseada. Esta clase se basa en un algoritmo relativamente sencillo, sobre el que se podrían aplicar una serie de mejoras para evitar los efectos de *aliasing* o deformación de la imagen en el proceso de

rotación. Sin embargo, debido al reducido tamaño de los elementos que se van a rotar en los instrumentos, no es necesario implementar un algoritmo más complejo, ya que esto supondría también un coste en cuanto a rendimiento. En la Figura 3.7 se muestra un ejemplo del proceso de rotación implementado ligeramente más complejo (ya que incluye una corrección del efecto de *aliasing*):



(a) Imagen original antes de la rotación (b) Imagen rotada 45º

Figura 3.7: Ejemplo de la rotación de una imagen. Imagen obtenida de:[1]

De una forma similar al caso de la clase `translateLabel`, se han implementado una serie de filtros que impiden que se rote la imagen de forma imperceptible. En este caso se ha implementado una tolerancia de 0.5º. Cabe destacar que el ahorro computacional de filtrar los movimientos imperceptibles en la interfaz gráfica es mucho más significativo que en el caso anterior, ya que en este caso además de actualizar la interfaz con la nueva imagen rotada, se ha de calcular la rotación en sí. Además, el coste computacional del cálculo de la nueva imagen rotada aumenta de forma exponencial con el número de píxeles totales de la imagen.

Por último, el parámetro booleano opcional denominado que indica si la variable `currentAngle` debe actualizarse, tiene su utilidad únicamente en algunos de los instrumentos implementados. La razón de ser de este parámetro es la misma que la del parámetro opcional del método `translateLabel`. Es decir, la variable `currentAngle` se deberá actualizar en aquellos instrumentos en los que sea necesario guardar el valor real del ángulo del instrumento (que puede no coincidir con el valor que se muestra en la interfaz).

[1] `http://hipertextual.com/archivo/2011/08/rotar-fotografias-pecado-o-no`

2.2. ADI - Attitude Director Indicator: Horizonte artificial

2.2.1. Descripción

El ADI, conocido comúnmente como horizonte artificial, es un instrumento cuya función principal es la de dotar al piloto de una referencia al horizonte cuando éste no es visible. Existen multitud de ADIs en el mercado, donde en cada uno se puede llegar a mostrar un conjunto de parámetros distintos. Si bien, algo que todo horizonte artificial tiene en común es la habilidad de indicar la actitud de la aeronave mediante los ángulos de *pitch* y *bank* en relación a la superficie terrestre.

El mecanismo interno que permite funcionar ADI es quizás uno de los más complejos mecánicamente dentro del *cockpit*. Este instrumento utiliza un giroscopio para establecer una plataforma inercial. En el caso de tratarse de una ADI analógico, el giroscopio se conecta mecánicamente a la pantalla, la cual tiene dos grados de libertad, lo que le permite representar los ángulos de *pitch* y de *bank*. En el caso de tratarse de un ADI digital, los datos del giroscopio serán interpretados por el instrumento para representar correctamente la información en la pantalla. La complejidad mecánica de este tipo de instrumentos se puede observar en la Figura 3.8, en la que se muestra un ejemplo de un horizonte artificial analógico.

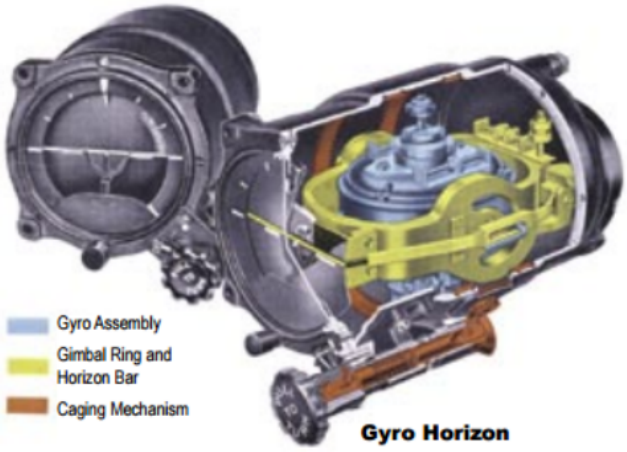


Figura 3.8: Esquema de un ADI. Fuente: [43]

2.2.2. Generación por capas

En el caso que ocupa este proyecto, se ha implementado un horizonte artificial analógico, muy similar al que equipan las aeronaves Cessna 172 Skyhawk. Como ya se ha comentado, la implementación del ADI se ha realizado combinando distintas imágenes a modo de capas. La descomposición en capas del ADI puede verse en la Figura 3.9. Estas imágenes se van cargando en el orden adecuado dentro del *layeredPane* hasta generar el instrumento final. Mediante esta generación por capas, es

posible mover cada una de la forma adecuada para simular el funcionamiento real del instrumento.

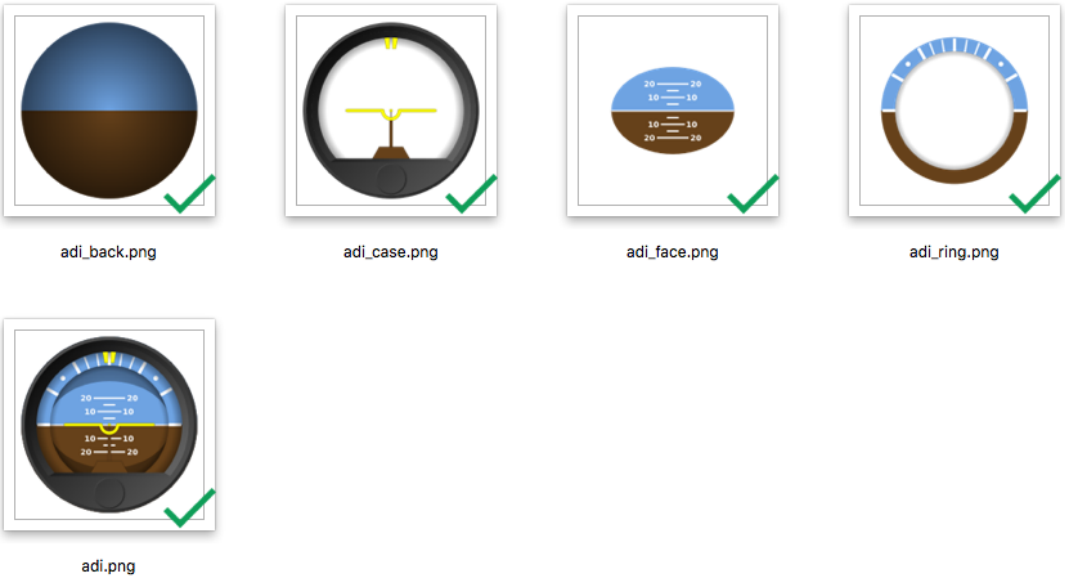


Figura 3.9: Descomposición del ADI en capas

Las capas se han superpuesto siguiendo el siguiente orden, donde la numeración ascendente implica que la capa con el número más alto se trata de la capa más superficial. Además, se ha seguido la notación de la Figura 3.9 para una mayor claridad:

1. adi_back
2. adi_face
3. adi_ring
4. adi_case

2.2.3. Implementación del código

El horizonte artificial es sin duda el instrumento más complejo mecánicamente del grupo denominado como *standard six*. Esto conlleva que su implementación sea también algo más compleja que la de otros instrumentos.

Pese a su complejidad, el horizonte artificial tiene únicamente dos capas móviles, dichas capas, siguiendo la notación de la Figura 3.9, son:

- **adi_face:** Esta pieza móvil es la encargada de dar información acerca del ángulo de *pitch* de la aeronave.
- **adi_ring:** Esta capa es la encargada de dar información acerca del ángulo de *roll* de la aeronave.

Por tanto, en el código implementado se deberán de tener en cuenta tanto el ángulo de *pitch* como el de *roll*.

Representación del ángulo de *pitch*:
Para ser capaces de representar el ángulo de *pitch* de forma correcta, se han tenido que contar los píxeles a los que corresponden las marcas de $+10^{o}$ y -10^{o} de la imagen *adi_face*. Una vez calibrado correctamente, se ha simplemente se llama al método `translateLabel` con el número de píxeles correspondiente.

Representación del ángulo de *roll*:
El procedimiento seguido para la representación del ángulo de *roll* más sencillo que en el caso anterior. En este caso, simplemente se ha de rotar la imagen *adi_ring* el ángulo correspondiente llamando al método `rotateLabel`.

2.3. ALT - Altímetro: Indicador de altitud

2.3.1. Descripción

El altímetro es un instrumento indispensable para garantizar una vuelo seguro. La función del altímetro no es otra que la de indicar al piloto la altura a la que se está volando. Los altímetros utilizados en aviación basan su medición en la presión estática. A partir de la presión estática se puede obtener la altura a la que se encuentra la aeronave mediante el uso de modelos de atmósfera estándar (ISA).

El principio de operación de un altímetro barométrico se basa en la compresión de una serie de cilindros en cuyo interior se ha realizado un vacío parcial. La presión atmosférica a la que están sometidas las caras del cilindro deforman al mismo y esta deformación es la que mueve a la aguja del altímetro en mayor o menor medida. La morfología del mecanismo que compone el altímetro se puede ver en la Figura 3.10.

En este caso, el altímetro implementado es el conocido como altímetro de tres agujas, el mismo tipo que se muestra en la Figura 3.10. En este tipo de altímetros, la aguja de grosor intermedio usa una escala de 100 pies en la esfera. La aguja gruesa y corta utiliza una escala de 1000 pies, mientras que la aguja larga y más fina (terminada en un triangulo invertido) tiene una escala de 10000 pies. La suma de la lectura de las tres agujas será la lectura de altura total.

El altímetro utilizado implementa también una rueda de regulación de la presión de referencia, para ajustar así la altura mostrada por el altímetro en función de la presión.

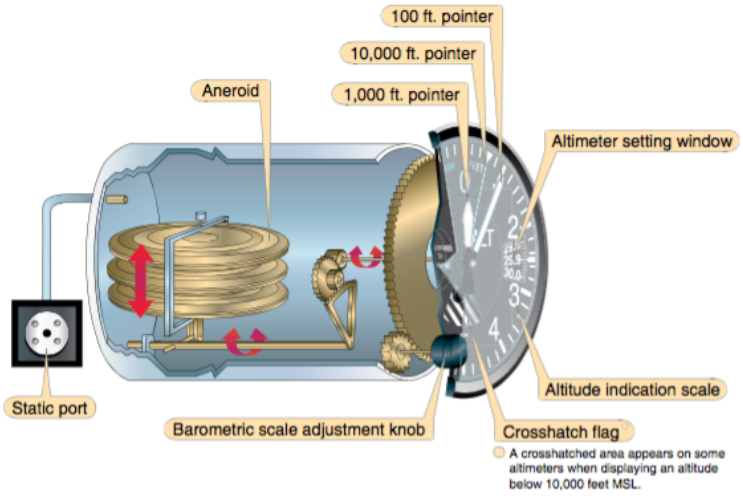


Figura 3.10: Esquema del mecanismo de un altímetro barométrico. Fuente: [29]

2.3.2. Generación por capas

Las imágenes utilizadas en el *layeredPane* para la recreación gráfica del altímetro se muestran en la Figura 3.11. En este caso se puede observar cómo, debido a la multitud de escalas presentes en este instrumento, el número de imágenes superpuestas necesarias es mayor que en otros instrumentos.

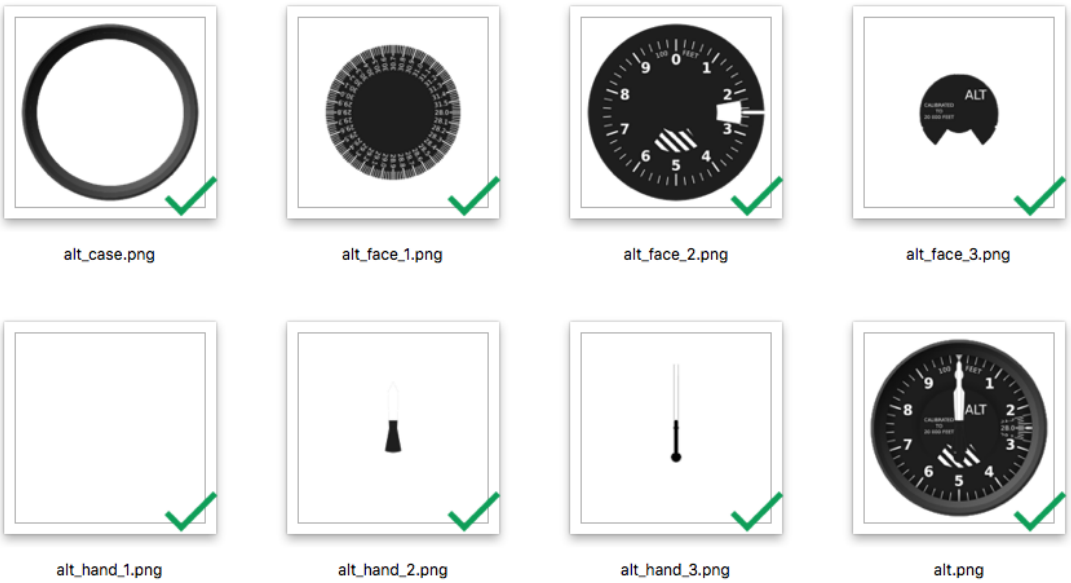


Figura 3.11: Descomposición del Altímetro (ALT) en capas

Las capas se han superpuesto siguiendo el siguiente orden, donde la numeración ascendente implica que la capa con el número más alto se trata de la capa más superficial.

Además, se ha seguido la notación de la Figura 3.11 para una mayor claridad:

1. alt_face_1
2. alt_face_2
3. alt_face_3
4. alt_hand_1
5. alt_hand_2
6. alt_hand_3

2.3.3. Implementación del código

En este caso, las piezas móviles son las siguientes Labels:

- **alt_hand_1:** Indica las décimas de millar de pies. Cada marca del instrumento equivale a 10000 pies. En este caso, el ángulo de rotación de la aguja será:

$$\theta_{hand1} = \frac{360}{10000} Altitud = 0{,}0036 Altitud \tag{3.1}$$

- **alt_hand_2:** Esta aguja indica los millares de pies, por lo que el ángulo que se deberá de rotar será una décima parte de la aguja anterior:

$$\theta_{hand2} = \frac{360}{1000} Altitud = 0{,}036 Altitud \tag{3.2}$$

- **alt_hand_3:** Indica los centenares de pies. Para escalar el ángulo que se debe de rotar esta aguja es suficiente con aplicar el factor de conversión mostrado en (3.3):

$$\theta_{hand3} = \frac{360}{100} Altitud = 0{,}36 Altitud \tag{3.3}$$

Por tanto, para mover las distintas agujas del altímetro, se deberá de llamar al método `rotateLabel` tres veces (una para cada aguja), con los valores obtenidos según las expresiones (3.1), (3.3) y (3.3).

2.4. ASI - Airspeed Indicator: Indicador de velocidad

2.4.1. Descripción

El *Airspeed Indicator* o ASI es un instrumento fundamental para la garantizar la seguridad en el vuelo. La necesidad de obtener una medida real de la velocidad del viento en la aeronave reside en el hecho de que el piloto debe garantizar en todo momento que está volando a la velocidad adecuada para la maniobra que esté

realizando.

El ASI basa su medición en la comparación entre la presión de parada y la presión estática para obtener la presión dinámica, para lo que se utiliza un tubo de Pitot. El funcionamiento del tubo de pitot se basa en la ecuación de Bernouilli:

$$p_t = p_s + q = p_s + \frac{1}{2}\rho u^2 \tag{3.4}$$

Donde, conociendo la presión de parada (p_t) y la presión estática (p_s) gracias a las mediciones del tubo de Pitot, se puede obtener fácilmente el valor de la velocidad en función de dichas presiones y de la densidad del aire:

$$u = \sqrt{\frac{2(p_t - p_s)}{\rho}} \tag{3.5}$$

Sin embargo, esto implica que se debe conocer la densidad del aire en cada instante para poder obtener el valor de la velocidad. Dicha densidad se puede obtener en función de la altura de vuelo, utilizando el modelo de atmósfera estándar ISA. Sin embargo, dependiendo de la desviación de la temperatura y de la presión del momento en concreto con la temperatura y presión del modelo estándar, la velocidad medida deberá ser corregida para obtener un valor real. Existen por tanto los siguientes tipos de velocidad aerodinámica:

- **IAS - *Indicated Airspeed*: Velocidad Indicada**
  La velocidad indicada es la que se obtiene aplicando la Ecuación (3.5) directamente con el valor de densidad correspondiente a nivel del mar de la atmósfera estándar ISA, sin correcciones de ningún tipo.

- **CAS - *Calibrated Airspeed*: Velocidad Calibrada**
  La velocidad calibrada es una velocidad corregida a partir de la IAS. En este caso se realizan correcciones tanto por posición (altura) como por posibles errores del instrumento. Normalmente los instrumentos que muestran la CAS tienen una serie de tablas que les permiten corregir la velocidad según los parámetros que apliquen en cada momento.

- **EAS - *Equivalent Airspeed*: Velocidad Equivalente**
  La velocidad equivalente se obtiene al corregir la velocidad calibrada (CAS) por los efectos de la compresión del aire dentro del tubo de Pitot. A nivel del mar, CAS y EAS son idénticas, sin embargo, a medida que aumenta la altura la velocidad CAS se vuelve mayor de lo que la EAS, razón por la que se aplica la corrección.

- **TAS - *True Airspeed*: Velocidad Verdadera**
  La velocidad verdadera se obtiene al corregir la velocidad calibrada (CAS) por las diferencias de presión y temperatura con respecto a la atmósfera estándar ISA. La velocidad TAS será por tanto la velocidad aerodinámica real de la aeronave.

En el caso del Airspeed Indicator, éste muestra datos de velocidad indicada (IAS), esto se debe a que las actuaciones de la aeronave dependen en gran medida de la densidad del aire. Por tanto, la IAS es mucho más relevante que la TAS a la hora de pilotar una aeronave, ya que parámetros como la velocidad de entrada en pérdida serán constantes a cualquier altura siempre que se utilice en términos de velocidad indicada (IAS). En el caso de utilizar la velocidad verdadera, la entrada en pérdida se produciría a distintas velocidades en función de la altura.

La Figura 3.12 muestra la estructura conceptual de un indicador de velocidad. En ella se puede observar cómo el ASI requiere de dos valores de entrada: la presión total y la presión estática.

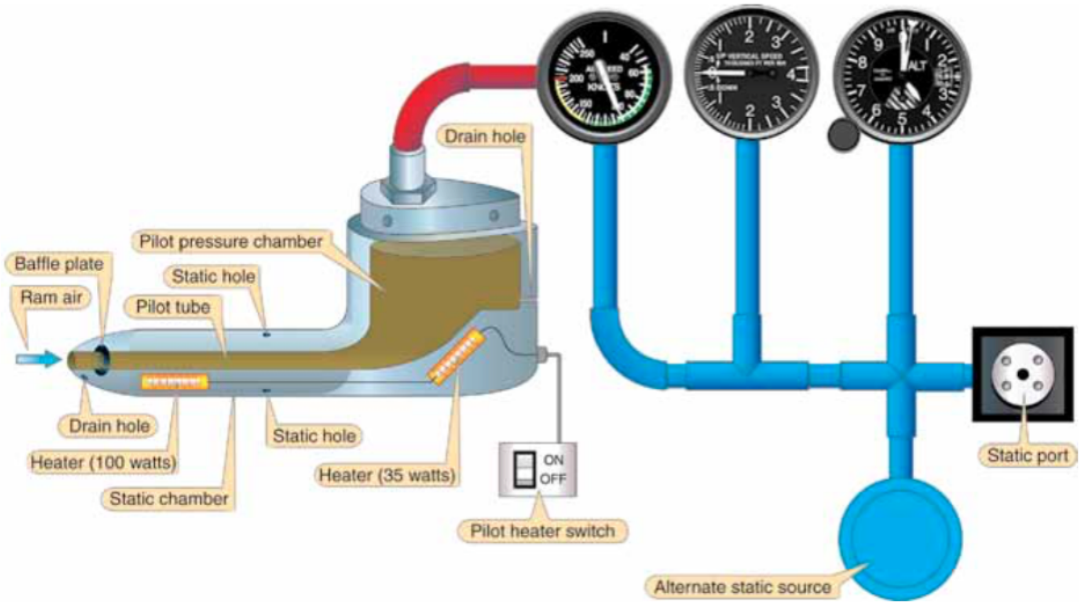


Figura 3.12: Estructura conceptual de un indicador de velocidad. Fuente: [29]

2.4.2. Generación por capas

El ASI implementado en este proyecto, con sus correspondientes capas, se puede observar en la Figura 3.13. Se puede ver que en este caso se trata de un instrumento muy sencillo a nivel visual, razón por la que el número de capas necesario para su representación es muy reducido.

En la Figura 3.14 se pueden observar distintas zonas delimitadas con colores en la circunferencia del ASI. Éstos arcos corresponden a zonas de operación de la aeronave y se explican a continuación:

- **Arco blanco:** Se trata del rango de operación de la aeronave con flaps extendidos. Su límite inferior marca la velocidad de entrada en pérdida con flaps extendidos, mientras que su límite superior indica la máxima velocidad permitida con flaps

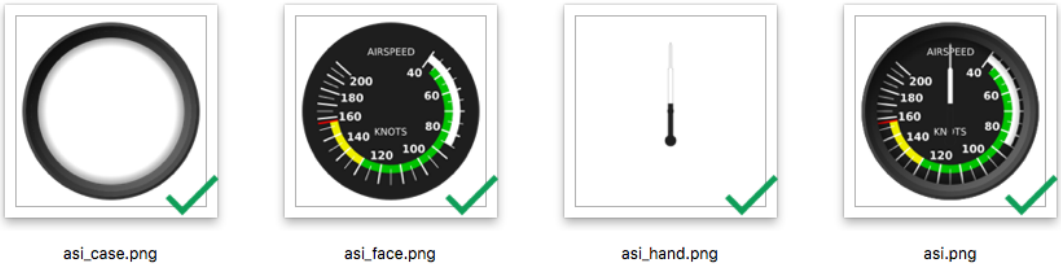


Figura 3.13: Descomposición del *Airspeed Indicator* (ASI) en capas

extendidos.

- **Arco verde:** Se trata del rango normal de operación de la aeronave. Su límite inferior marca la velocidad de entrada en pérdida sin flaps, mientras que su límite superior indica la máxima velocidad recomendada en casos de viento turbulento.
- **Arco amarillo:** Se trata del rango de operación en el que hay peligro de daño estructural. Su límite inferior marca la máxima velocidad recomendada en casos de viento turbulento mientras que su límite superior marca la velocidad máxima que nunca debería de ser rebasada (*Never-exceed airspeed*.
- **Línea roja:** La línea roja situada en el límite superior del arco amarillo indica a velocidad máxima que nunca debería de ser rebasada (*Never-exceed airspeed*.

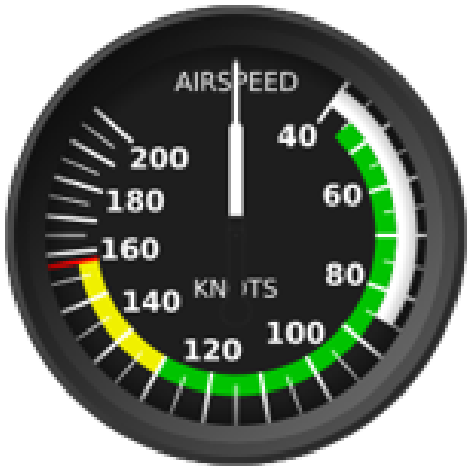


Figura 3.14: Rangos de velocidad del *Airspeed Indicator* (ASI)

Las capas se han superpuesto siguiendo el siguiente orden, donde, como en casos anteriores, la numeración ascendente implica que la capa con el número más alto se trata de la capa más superficial. Además, se ha seguido la notación de la Figura 3.13 para una mayor claridad:

1. asi_face
2. asi_hand
3. asi_case

2.4.3. Implementación del código

Para mostrar correctamente la velocidad en el ASI es suficiente con rotar la aguja los grados pertinentes (que en este caso se corresponde con la capa *asi_face*). Sin embargo, en este caso en concreto, se puede observar que la escala del instrumento varía según la velocidad. Esto se puede observar fácilmente en la Figura 3.15, donde se ha importado la imagen del instrumento a Autocad® y se ha medido el ángulo que define cada una de las tres escalas del instrumento.

Figura 3.15: Obtención de las escalas del Airspeed Indicator mediante Autocad®

Tal y como se puede ver en la Figura 3.15, existen tres escalas distintas en el instrumento. Por tanto, la lógica que se debe de seguir para obtener el ángulo de rotación de la aguja es la siguiente:

$$\left.\begin{array}{lll} \theta_{rot} = \dfrac{35}{40}V & si & V \leq 40\ KIAS \\ \theta_{rot} = 35 + \dfrac{18}{10}(V-40) & si & 40 \leq V \leq 160\ KIAS \\ \theta_{rot} = 35 + \dfrac{18}{10}(160-40) + \dfrac{12}{10}(V-160) & si & V \leq 160\ KIAS \end{array}\right\} \quad (3.6)$$

Por tanto, para mostrar de forma gráfica la velocidad indicada, se llamará al método `rotateLabel` con el ángulo obtenido de (3.6).

2.5. HI - Heading Indicator: Indicador del heading de la aeronave

2.5.1. Descripción

El indicador del *heading* de la aeronave es un instrumento básico para la orientación del piloto, sobretodo si no se dispone de radio-ayudas cercanas que nos permitan utilizarlas para seguir una ruta.

Este tipo de instrumento basa su funcionamiento en la utilización de un giroscopio. El mecanismo interno es similar al visto en el ADI, sin embargo, en este caso lo que se mide es la rotación sobre el eje vertical de la aeronave.

Cabe destacar que la mayoría de indicadores de *heading* no tienen capacidad para saber dónde se encuentra el norte, ya que su funcionamiento se basa únicamente en el giroscopio. Es por esto que el piloto debe ayudarse de la rueda de calibración para hacer coincidir el norte del HI con el norte magnético, para lo que se necesitará la ayuda de una brújula. La rueda de calibración así como el mecanismo giroscópico de un HI se pueden observar en la Figura 3.16.

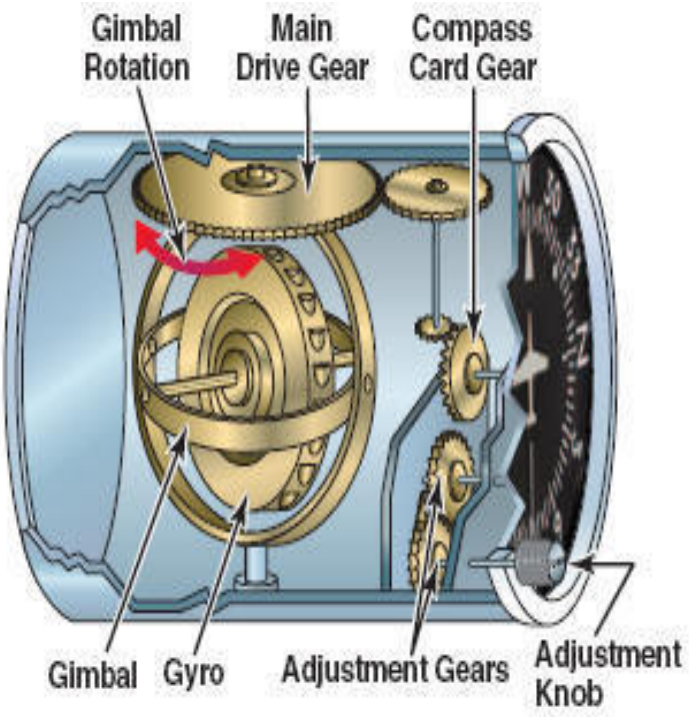


Figura 3.16: Estructura conceptual de un *Heading Indicator* (HI). Fuente: [44]

2.5.2. Generación por capas

La descomposición por capas del *Heading Indicator* implementado se muestra en la Figura 3.17. Cabe destacar que en este caso el instrumento implementado no tiene rueda de calibración. Esto no es un problema ya que las medidas simuladas de lo sensores se obtienen directamente desde el simulador X-Plane, por lo que el instrumento indicará el *heading* de forma correcta. De la misma forma que en el caso de indicador de velocidad (ASI), el HI es un instrumento sencillo en cuanto a su representación gráfica. Esto es por lo que tres capas son suficientes para representar de forma correcta todas las funcionalidades del instrumento.

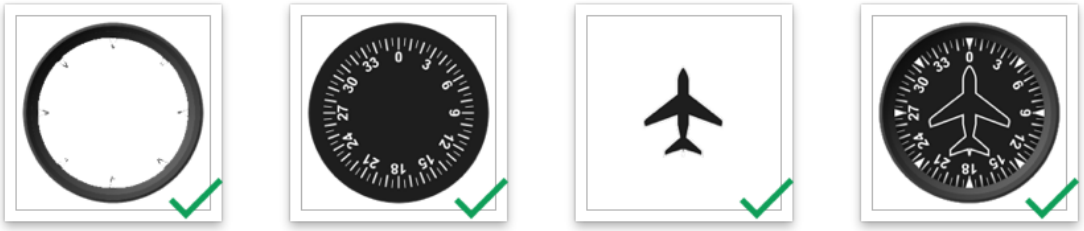


Figura 3.17: Descomposición del *Heading Indicator* (HI) en capas

En este caso, el orden de las capas, siguiendo la notación de la Figura 3.17, será el siguiente:

1. hsi_face
2. hsi_plane
3. hsi_case

2.5.3. Implementación del código

La implementación del *Heading Indicator* es probablemente la más sencilla de los instrumentos presentados en este proyecto. En este caso la capa que se rota es la capa *hsi_face*. El avión representado en el instrumento se mantiene estático y solamente se rota la regla del instrumento.

Además, para la implementación de este instrumento no es necesario escalar el ángulo, por lo que será suficiente con llamar al método `rotateLabel` para rotar el label *hsi_face* el ángulo correspondiente al rumbo de la aeronave.

2.6. T/S TC - Coordinador de giro

2.6.1. Descripción

El coordinador de giro es un instrumento cuya función es la de dar al piloto información sobre dos parámetros: el ratio de giro y la relación entre el ángulo de *bank* y el ángulo de *yaw* [29].

Para presentar información acerca del ratio de giro, el coordinador de giro utiliza un giroscopio. En este caso, el instrumento en cuestión se sirve del fenómeno físico de la precesión giroscópica para calcular el ratio de giro. La precesión giroscópica es el fenómeno que produce que al aplicar una fuerza a un objeto en rotación, los efectos de la aplicación de la misma se producen con un desfase de 90^{o} con respecto al punto de aplicación de la fuerza. La Figura 3.18 muestra lo comentado acerca de la precesión.

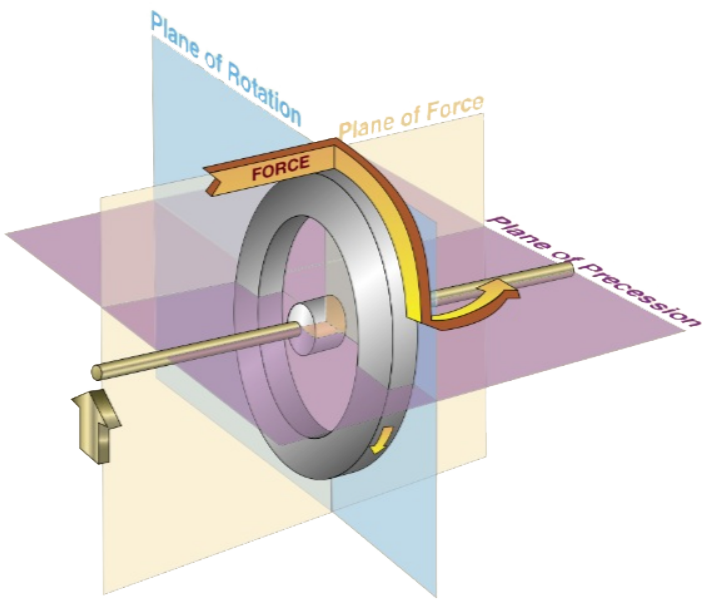


Figura 3.18: Descripción del fenómeno de precesión giroscópica. Fuente: [29]

El giroscopio que incluye el coordinador de giro está inclinado alrededor de 30º con respecto al eje longitudinal de la aeronave. Con esto se consigue medir tanto el ángulo de *roll* como el de *yaw*. En los instantes iniciales de un giro, el símbolo de la aeronave del instrumento muestra el ratio de alabeo. Cuando el ángulo de alabeo se estabiliza, el coordinador de giro pasa a mostrar el ratio de giro de la aeronave, que es realmente el valor que interesa al piloto. Este tipo de instrumentos suele incluir una serie de marcas blancas, concretamente dos a cada lado de la esfera. Las marcas superiores indican la posición en la que el ratio de giro es nulo y por tanto la aeronave sigue una trayectoria rectilínea. Las marcas inferiores indican el ratio de giro correspondiente a un giro de 2 minutos, esto es, el giro para el que se tarda 2 minutos en dar una vuelta completa. Estas marcas son de gran ayuda, ya que numerosos procedimientos incluyen giros de 2 minutos. La Figura 3.19 muestra la estructura interna de un coordinador de giro. En dicha figura se puede observar claramente el giroscopio inclinado que se ha descrito anteriormente.

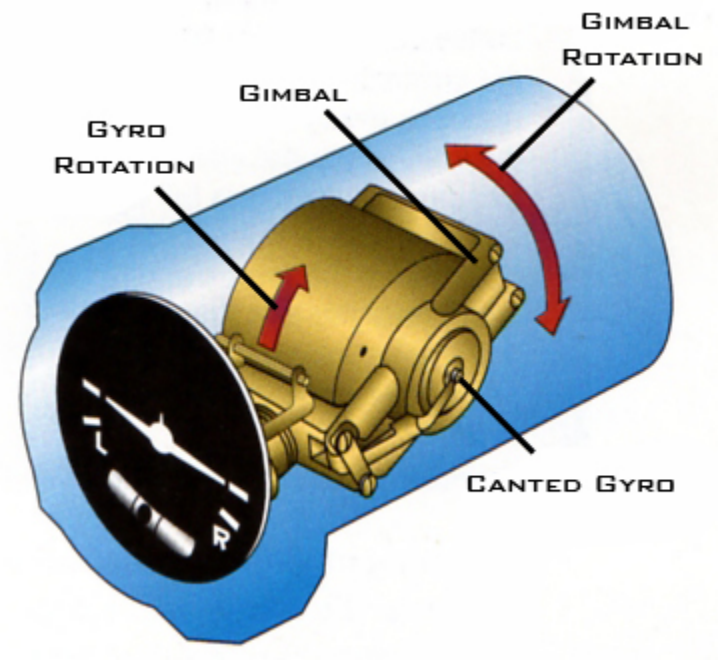


Figura 3.19: Estructura conceptual de un coordinador de giro. Fuente: [45]

La pequeña esfera que se incluye en el coordinador de giro tiene la función de indicar al piloto acerca de la coordinación del giro. Se define un giro coordinado como aquel en el que la nariz de la aeronave apunta en todo momento a la tangente de la trayectoria seguida, sin que se produzca resbalamiento o derrape. Esto implica que el ángulo de resbalamiento (*sideslip* en inglés) es nulo en un giro coordinado. El ángulo de resbalamiento se define como el ángulo que existe entre el vector de velocidad relativa del viento y el eje longitudinal de la aeronave, tal y como se muestra en la Figura 3.20

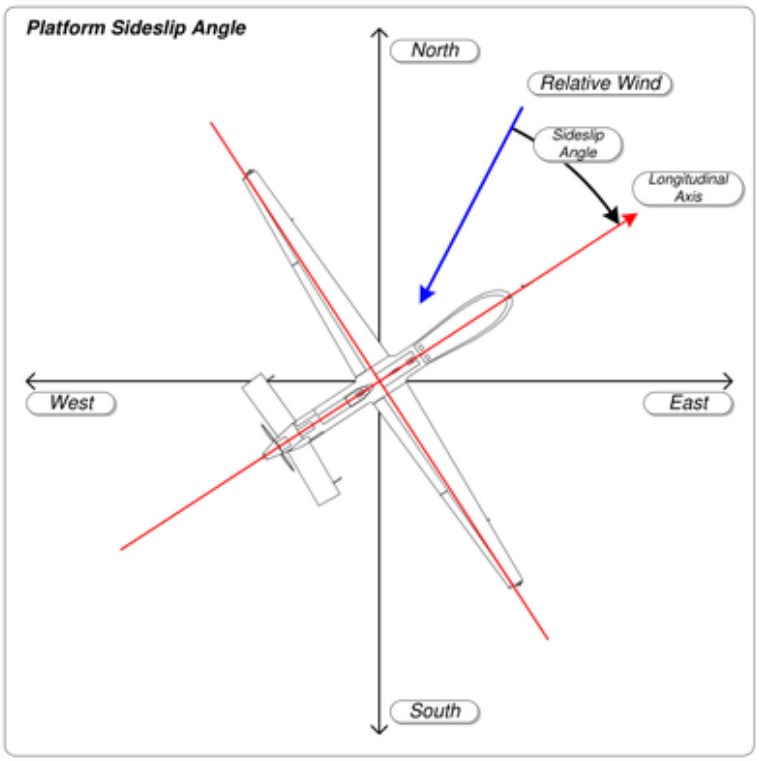


Figura 3.20: Representación gráfica del ángulo de resbalamiento en una aeronave. Imagen obtenida de: [2]

En un giro coordinado, el piloto y todos los objetos que se encuentren dentro de la aeronave únicamente sentirá fuerzas en el eje longitudinal (aceleraciones y deceleraciones) y en el eje vertical (su propio peso y posibles aceleraciones debidas a la maniobra realizada). Por tanto, en un giro perfectamente coordinado, no existirán aceleraciones laterales, siempre tomando como referencia los ejes cuerpo de la aeronave. La esfera del coordinador de giro consiste en una pequeña bola inmersa en un líquido dentro del compartimento del instrumento.

Cuando se realiza un giro coordinado, la aceleración lateral de la aeronave en ejes cuerpo será nula, por lo que la esfera se mantendrá en el centro. Sin embargo, si se realiza un giro no coordinado, las aceleraciones laterales en ejes cuerpo ocasionadas por la falta de coordinación moverán la esfera, indicando así no coordinación el giro. Cabe destacar que todas las aceleraciones referidas anteriormente se definen en los ejes cuerpo. Durante un giro, necesariamente se producirán aceleraciones laterales con respecto a un sistema de referencia fijo como el NED (*Nort-East-Down*), sin embargo, en el sistema de referencia dinámico de los ejes cuerpo, sí que se puede realizar un giro sin que los ocupantes de la aeronave sientan aceleraciones laterales.

[2] `https://en.wikipedia.org/wiki/Slip_(aerodynamics)`

Para controlar la coordinación del giro, el piloto hace uso del timón de cola o *rudder*. La dirección a aplicar el *input* de rudder es siempre la misma en la que se desplaza la esfera. Por tanto, en un giro a derechas en el que la bola se haya desplazado hacia la izquierda (derrape), el piloto deberá aplicar una corrección de *rudder* pisando el pedal izquierdo.

Lo explicado en los párrafos anteriores se puede ver de forma resumida en la Figura 3.21.

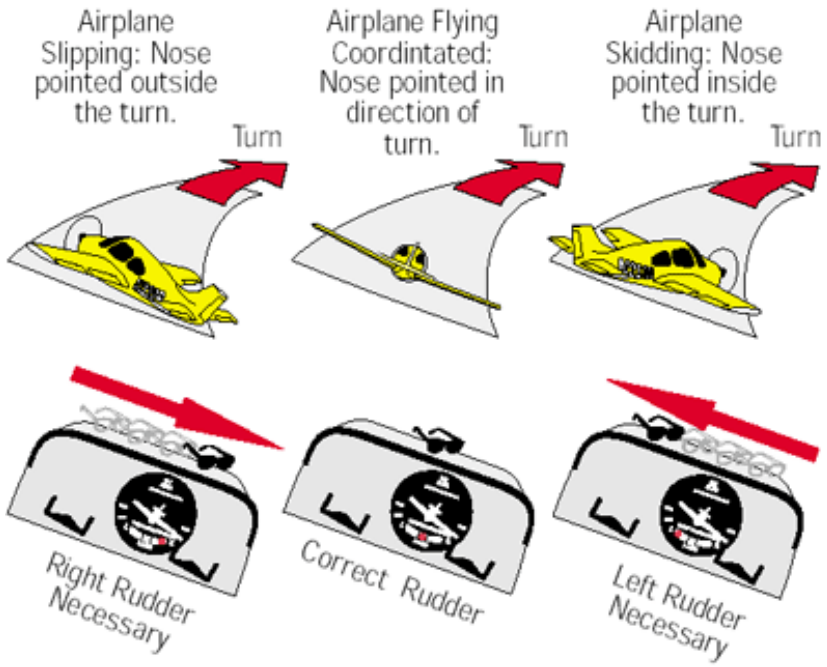


Figura 3.21: Descripción de la coordinación de un giro. Fuente [46]

2.6.2. Generación por capas

La descomposición por capas del coordinador de giro se muestra en la Figura 3.22. En ella se pueden ver todas las capas necesarias para la formación gráfica del instrumento, que en este caso es bastante sencilla.

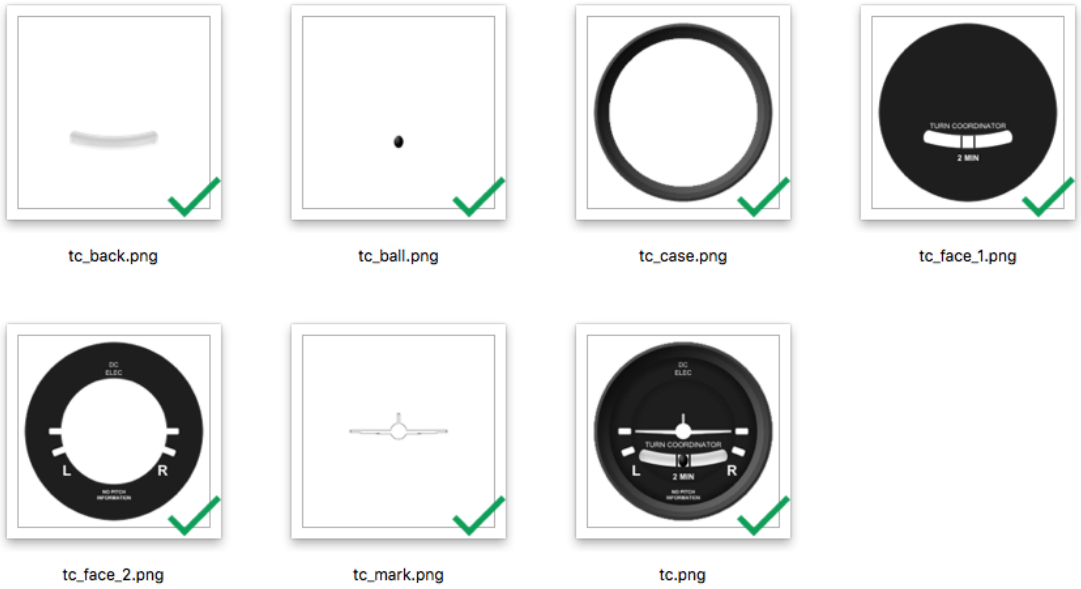


Figura 3.22: Descomposición por capas del coordinador de giro

En el caso del coordinador de giro, el orden de las capas será el siguiente:

1. tc_back
2. tc_ball
3. tc_face_1
4. tc_face_2
5. tc_mark
6. tc_case

2.6.3. Implementación del código

Este instrumento presenta dos capas móviles:

- *tc_ball:* La esfera que indica la coordinación del giro.
- *tc_mark:* La marca del avión que indica el ratio de giro.

Movimiento de la esfera (capa tc_ball)

Como se ha comentado con anterioridad, la esfera del coordinador de giro se mueve debido a las cargas laterales a las que se expone la aeronave. Estas cargas laterales están relacionadas con el ángulo de *side-slip*. Este ángulo,

La aproximación que se ha seguido en este trabajo para caracterizar el movimiento de la esfera ha sido la de comparar el comportamiento de la esfera en el simulador

X-Plane® con el ángulo de *sideslip* en cada momento. Después de estos vuelos en los que se han recopilado datos sobre el comportamiento de la bola, se ha podido determinar que su movimiento se puede modelar de una forma razonablemente precisa asumiendo que la bola se mueve de forma proporcional al cambio del ángulo de *sideslip*. El autor es consciente de que no se trata de la forma más exacta, sin embargo, teniendo en cuenta la poca exactitud del instrumento, se ha considerado que no era necesario optar por realizar un desarrollo teórico. Esto, unido a la falta de parámetros disponibles para medir correctamente los esfuerzos laterales desde X-Plane (los datos que X-Plane envía por el protocolo UDP son limitados), justifica la aproximación empírica por la que se ha optado en este trabajo.

Durante los vuelos realizados con el Simulador X-Plane, se ha observado que la esfera del coordinador de giro llega al extremo del instrumento aproximadamente a $\pm 7^{o}$ de *sideslip*. De esta forma, se ha obtenido la siguiente fórmula empírica para obtener el desplazamiento horizontal de la esfera en función del ángulo de *sideslip* (β):

$$x = \mathsf{Round}\left(-\mathsf{Min}\left(\left|\frac{\beta}{7}33\right|, 33\right)\right) \cdot \frac{\beta}{|\beta|} \tag{3.7}$$

Donde x es el número de píxeles de desplazamiento en la dirección horizontal y el signo negativo dentro del redondeo y la fracción final tienen la función de garantizar que el movimiento de la esfera se realiza siguiendo el criterio de signos asociado al sistema de referencia de la interfaz gráfica. El redondeo realizado se debe al hecho de que los desplazamientos que se pueden aplicar son píxeles y por tanto se han de obtener desplazamientos con números enteros. Además, la inclusión del mínimo es debido a que el máximo desplazamiento lateral de la esfera es de 33 píxeles en la interfaz implementada. Obviamente, este valor variará si se altera la interfaz, por lo que en realidad se debe de implementar como un valor proporcional al tamaño en píxeles del instrumento.

La Ecuación (3.7) muestra el desplazamiento horizontal de la esfera. Sin embargo, a medida que se desplaza en horizontal, la esfera también se desplaza en la dirección vertical. Este desplazamiento se ha modelado mediante la ecuación (3.8), basándose en la geometría del instrumento implementado.

$$\left.\begin{array}{lll} y = 0 & si & |x| \leq 15 \\ y = -1 & si & 15 \leq x \leq 22 \\ y = -2 & si & 22 \leq x \leq 25 \\ y = -3 & si & 25 \leq x \leq 31 \\ y = -4 & si & x \geq 31 \end{array}\right\} \tag{3.8}$$

Una vez obtenidos los desplazamientos horizontal y vertical en píxeles, se llama al método `translateLabel` con estos valores para mover la esfera a la posición deseada.

Movimiento del indicador de ratio de giro (capa tc_mark)
El movimiento del indicador de ratio de giro es mucho más sencillo de modelar que el de la esfera del coordinador de giro. En este caso, es suficiente con rotar el label *tc_mark* un determinado ángulo, proporcional al ratio de giro.

Desgraciadamente, el simulador de vuelo X-Plane no exporta el ratio de giro de la aeronave mediante su interfaz UDP. Debido a esta limitación, el ratio de giro (ω) se ha tenido de calcular como la derivada discreta del ángulo de *track* de la aeronave.

$$\omega^n + 1 = \frac{trk^{n+1} - trk^n}{t^{n+1} - t^n} \tag{3.9}$$

Donde el ratio de giro obtenido anteriormente tendrá las unidades de grado por segundo (^{o}/s).

Una vez calculado el ratio de giro instantáneo de la aeronave, se debe escalar dicho ratio para obtener el ángulo de rotación de la capa correspondiente. Esto, como en otros instrumentos, se ha realizado mediante el uso del software Autocad® y el resultado se muestra en la Figura 3.23.

Figura 3.23: Obtención de la escala del indicador de ratio de giro mediante Autocad®

Con esto se puede calcular el ángulo de rotación de la capa *tc_mark* teniendo en cuenta que las marcas de la Figura 3.23 delimitan un giro de 2 minutos, o lo que es lo mismo, un giro con un ratio de giro de $\omega = 3^o/s$. El ángulo de rotación del Label se calcula según la expresión (3.10). Con dicho valor se llamará al método `rotateLabel` para que el instrumento indique correctamente el ratio de giro.

$$\theta_{rot} = \frac{21}{3}\omega \tag{3.10}$$

2.7. VSI - Vertical Speed Indicator: Indicador de la velocidad vertical

2.7.1. Descripción

El indicador de velocidad vertical es un instrumento de gran utilidad en ascensos y en descensos, ya que permite al piloto tener una noción de su velocidad vertical en dichas maniobras. De esta forma se consigue que el piloto realice ascensos y descensos mucho más controlados y, por tanto, seguros. Sin embargo, la utilidad de un VSI no se limita únicamente a maniobras de ascenso y descenso. El VSI es también un instrumento de gran utilidad para mantener una altitud constante, debido a que su precisión es mucho mayor que la de un altímetro. Si un piloto de sirve únicamente de un altímetro para mantener su altura, es posible que su altura se reduzca más de un centenar de pies sin que el piloto se dé cuenta. Por el contrario, sirviéndose de un VSI, el piloto puede controlar su velocidad vertical fácilmente y conseguir mantener su altura con pocos pies de error.

El funcionamiento de un VSI es similar al de un indicador de velocidad (ASI). Sin embargo, en este caso no se comparan valores de presión de parada y presión estática, sino que se implementa una orificio de entrada de presión estática y un pequeño orificio

de salida. Lo que se pretende medir con el VSI son los cambios de presión estática, lo que puede extrapolarse a cambios de altura. En este caso, al existir un pequeño orificio de salida, cuando se produce un cambio de presión estática, ésta termina por igualarse dentro del instrumento gracias al orificio de salida. Sin embargo, debido al reducido tamaño del orificio, la presión tardará unos instantes en igualarse, por lo que se puede obtener una medida de la derivada de la presión atmosférica y con ello el ratio de cambio de altura (es decir, la velocidad vertical). En la Figura 3.24 se muestra la estructura interna que compone el VSI, la cual se basa en la deformación de un cilindro debido a la presión aplicada en sus caras, algo que es común a otros instrumentos como el altímetro o el indicador de velocidad.

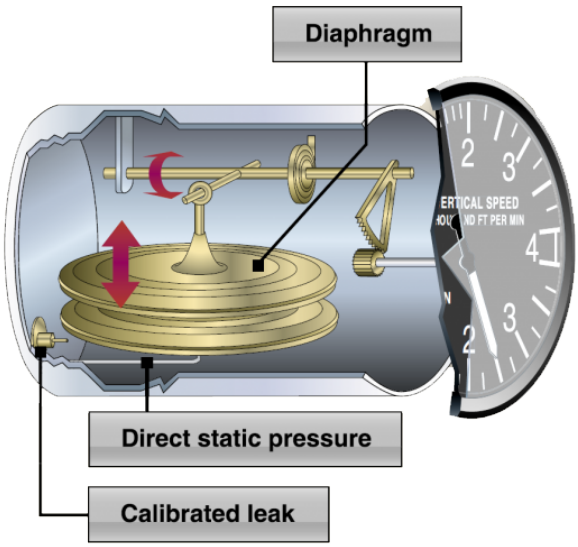


Figura 3.24: Estructura conceptual de un indicador de velocidad vertical (VSI). Fuente: [47]

Uno de los inconvenientes del uso de los VSI tradicionales como el de la Figura 3.24 es el retraso existente entre el instante en el que se produce un cambio de velocidad vertical y el instante en el que se muestra dicho cambio en el instrumento. Esto es debido al tiempo que se tarda en que se modifiquen las presiones dentro del instrumento lo suficiente como para que se deformen los cilindros internos. Para resolver este problema, existen otro tipo de indicadores de velocidad vertical, más complejos y precisos; son los denominados: indicadores de velocidad vertical instantáneos (*Instantaneous Vertical Speed Indicators*, IVSI). En este caso se añaden dos acelerómetros que se activan por el cambio de pitch de la aeronave y mueven al mecanismo hacia la posición indicada. Una vez han dejado de actuar los acelerómetros y el pitch ha dejado de variar, el tiempo de retraso de la medida del VSI tradicional ya ha pasado, por lo que el instrumento ya es capaz de medir los cambios de presión correctamente.

En el instrumento que ocupa a este trabajo, el retraso en la medida de la velocidad vertical no existe debido a que dicha velocidad se toma directamente de los valores dados por el simulador X-Plane. Por tanto, el único retraso existente es el producido por la frecuencia de adquisición de datos del simulador: 0,05 segundos (la adquisición de datos se realiza a 20 Hz).

2.7.2. Generación por capas

En la Figura 3.25 se muestra la descomposición por capas del VSI implementado. Debido a que el instrumento posee únicamente una parte móvil en su exterior (la aguja), 3 capas son suficientes para su representación gráfica.

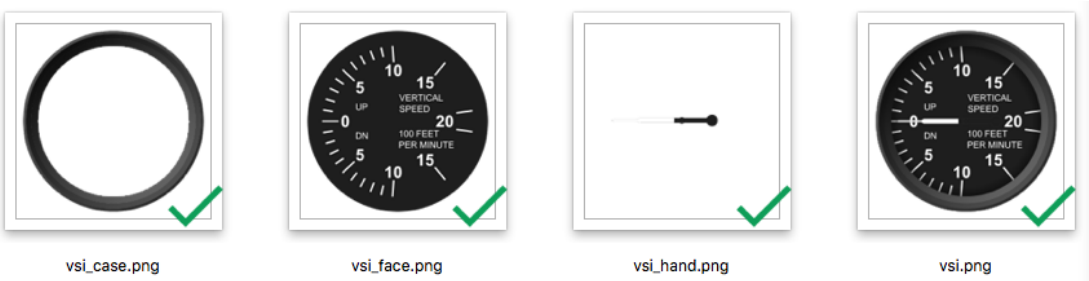


Figura 3.25: Descomposición por capas del indicador de velocidad vertical (VSI)

Las capas se han superpuesto siguiendo el siguiente orden, donde, como en casos anteriores, la numeración ascendente implica que la capa con el número más alto se trata de la capa más superficial.

1. vsi_face
2. vsi_hand
3. vsi_case

2.7.3. Implementación del código

La implementación de este instrumento es sencilla en comparación con otros más complejos como el coordinador de giro. Para implementar el indicador de velocidad vertical es suficiente rotar la aguja el ángulo correspondiente, midiendo el ángulo mediante Autocad®, de la misma forma que en instrumentos comentados anteriormente.

En este caso, cada una de las marcas de 500 pies/minuto se encuentran a 43º con respecto a la horizontal. Por tanto es inmediato ver que el ángulo que se debe de rotar la aguja se obtendrá según la Ecuación (3.11).

$$\theta_{rot} = \frac{43}{500} VS \tag{3.11}$$

Donde VS es la velocidad vertical en pies por minuto.

El único requisito adicional es el de impedir que la aguja se mueva más allá de las marcas de 2000 y -2000 ft/min. Algo trivial en este caso.

Con el valor del ángulo obtenido se llama al método `rotateLabel` para girar la aguja y que ésta apunte en la dirección adecuada.

2.8. TCAS

2.8.1. Descripción

El TCAS (*Traffic Alert and Collision Avoidance System*) se ha descrito anteriormente en la Sección 3.3 del Capítulo 1. Como se ha comentado anteriormente, el TCAS es un instrumento fundamental para la navegación aérea a día de hoy.

Debido a la gran densidad de tráfico aéreo que circula por ciertas zonas del mundo, el uso del TCAS se ha vuelto indispensable para garantizar la seguridad aérea y ayudar a los controladores de tráfico aéreo en su labor. Esta congestión de los espacios aéreos ha llegado al punto en el que en zonas de Europa central es obligatoria la inclusión de un sistema TCAS para el vuelo de aeronaves de más de 5700 Kg y 19 pasajeros [48].

La importancia de este instrumento en la navegación actual, así como la complejidad de su implementación son los dos aspectos clave por los que se ha decidido incluir este instrumento en el presente proyecto.

2.8.2. Generación por capas

El TCAS merece una mención a parte ya que es el único elemento que utiliza una mezcla de implementaciones: por un lado el fondo y la regla móvil que indica el *heading* que lleva el avión consiste en un *JLabel*, mientras que sobre ellos se define un panel de dibujo sobre el que se utiliza la librería `Graphics2D` con la que se pintan los elementos.

El TCAS está compuesto de un fondo negro con tres reglas fijas y una regla móvil, cuyo centro es la posición del avión, representado con forma de flecha. Las reglas indican radio de distancia a la aeronave de 10, 20, 30 y 40 Millas Náuticas, de menor a mayor distancia, siendo la última la regla móvil que indica el *heading* de la aeronave.

2.8.3. Closest Point of Approach (CPA)

Antes de proceder a explicar el funcionamiento del TCAS conviene introducir el concepto del *Closest Point of Approach* o CPA por sus siglas en inglés. El CPA se define como la distancia mínima a la que se encontrarán dos cuerpos al seguir sus respectivas trayectorias. Por tanto, en el contexto de la navegación aérea, se puede definir el CPA como la distancia mínima a la que se encontrarán dos aeronaves.

Este concepto es de gran ayuda para evaluar el peligro de colisión entre dos aeronaves. Debido a la gran velocidad a la que se mueven los aviones, imponer un sistema de alerta que se base únicamente en la distancia entre dos cuerpos no sería seguro ni eficiente. La importancia en la detección reside en ser capaz de saber si en el punto de distancia mínima (CPA) la distancia entre los dos aviones será suficientemente grande como para que no haya riesgo de colisión.

Aunque hay distintas formas de calcular la distancia del CPA, una de las formas más comunes y sencillas de hacerlo es mediante la utilización de un doble producto vectorial. Tomando r_1 como el vector posición de la aeronave 1 (en 3 dimensiones), y c_2 como la velocidad relativa de la aeronave 2 con respecto a la aeronave 1, el CPA se puede calcular como sigue:

1. Primero se normaliza el vector posición de la aeronave 1:

$$\overrightarrow{u_{c2}} = \frac{\overrightarrow{c_2}}{|\overrightarrow{c_2}|} \tag{3.12}$$

2. Seguidamente se calcula el siguiente doble producto vectorial:

$$\overrightarrow{r_m} = \overrightarrow{u_{c2}} \times (\overrightarrow{r_1} \times \overrightarrow{u_{c2}}) \tag{3.13}$$

3. Por último, la distancia del CPA se calcula como el módulo del vector r_m:

$$D_{CPA} = |\overrightarrow{r_m}| \tag{3.14}$$

Esta distancia del CPA se puede observar de forma gráfica en la Figura 3.26.

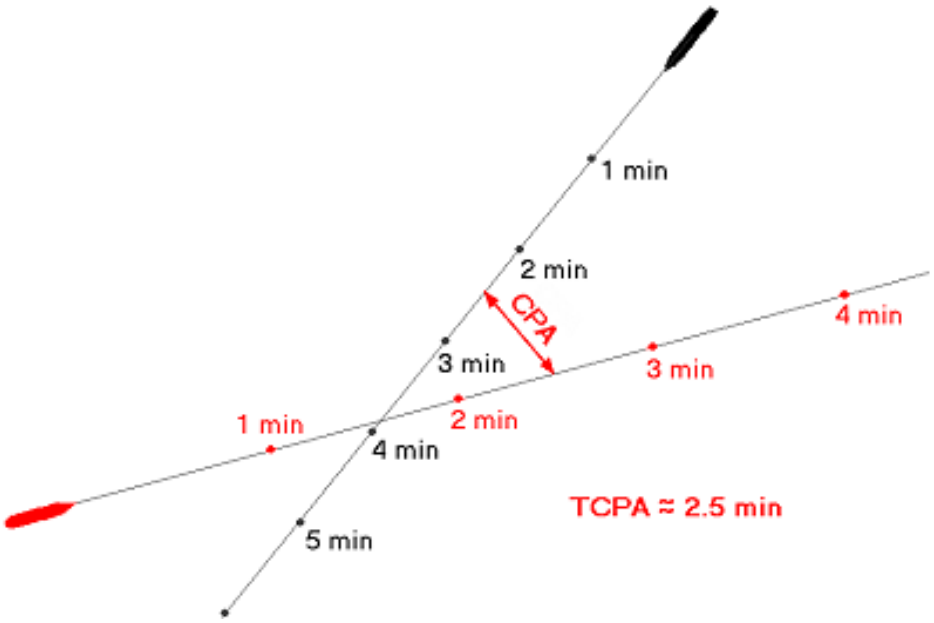


Figura 3.26: Representación gráfica del *Closest Point of Approach*

La distancia mínima a la que se encontrarán dos aeronaves es un parámetro crítico, pero en sí misma no es suficiente para evaluar el riesgo de colisión, ya que dos aeronaves con un CPA muy pequeño y situadas en dos puntos muy lejanos pueden cambiar de trayectoria en un momento determinado y aumentar notablemente su CPA. Es por esto que además de la distancia mínima se utiliza también el parámetro τ, que mide el tiempo restante para que se alcance la condición del *Closest Point of Approach*. En la Figura 3.26 se muestra también el parámetro τ (TCPA en la imagen).

Por tanto, los parámetros utilizados en los algoritmos de detección de amenazas de tráfico aéreo son tanto el CPA como el tiempo τ. Para un mayor detalle en cuanto a la utilización de estos dos parámetros, se insta al lector a consultar la referencia [49]. En la Figura 3.27 se muestra un esquema de las zonas de distintos niveles de alerta del TCAS, basadas en el tiempo de colisión τ.

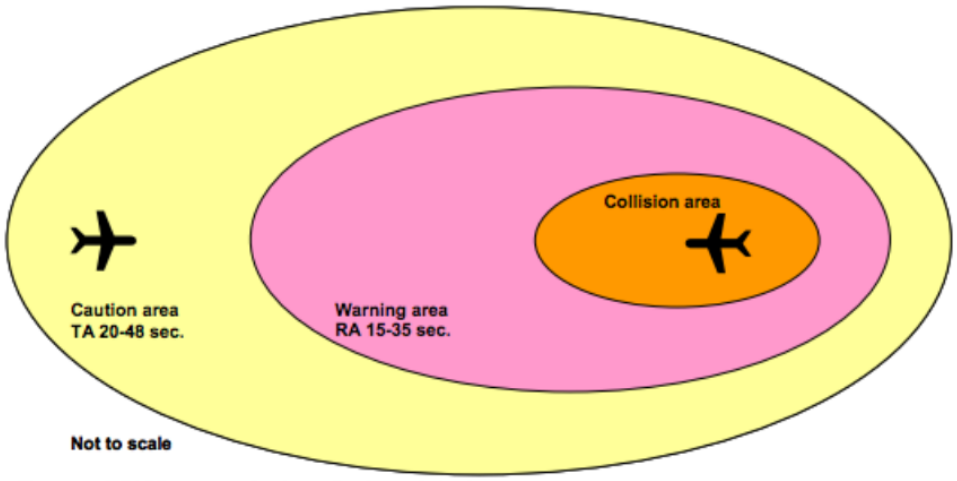


Figura 3.27: Niveles de alerta del TCAS en función del tiempo τ. Fuente: [49]

2.8.4. Funcionamiento del TCAS

En el presente proyecto se ha optado por implementar un sistema TCAS de tipo I, un sistema de TCAS que, si bien es uno de los más básicos que existen, es el que se suele montar en aeronaves de aviación general, como es el caso del *cockpit* que ocupa este proyecto. El TCAS de tipo I da la información sobre las aeronaves que se encuentran en las inmediaciones y evalúa el peligro de colisión con cada una de dichas aeronaves. Además, este tipo de TCAS suele tener un rango de operación de alrededor de 40 millas náuticas.

Cabe destacar que al contrario que en sistemas TCAS más avanzados como el TCAS II, III o IV, el TCAS de tipo I no da sugerencias u órdenes al piloto sobre cómo actuar cuando hay peligro de colisión. En este caso el TCAS simplemente se limita a evaluar el peligro de colisión y en caso de haberlo muestra la aeronave de un color distinto en el radar y hace saltar una señal sonora diciendo: "*traffic, traffic*". En este caso el sistema simplemente avisa de un posible riesgo de colisión y queda al criterio del piloto decidir cómo evitarla (normalmente asistido por los controladores de tráfico aéreo.

Como se ha comentado anteriormente,se ha querido dotar al sistema TCAS del mayor realismo posible, por lo que se ha implementado también una interconexión entre el sistema TCAS y los datos de tráfico aéreo real que provienen de la antena SACTA[3].Debido a que el uso de la antena SACTA está restringido por la disponibilidad de una conexión de internet y de una conexión VPN en caso de encontrarse el ordenador fuera de la red de la UPV, se ha decidido implementar también la posibilidad de utilizar un archivo de trazas reales para leer el tráfico aéreo.

[3]La interfaz de comunicación con la antena SACTA será tratada en la Sección 2 del Capítulo 4

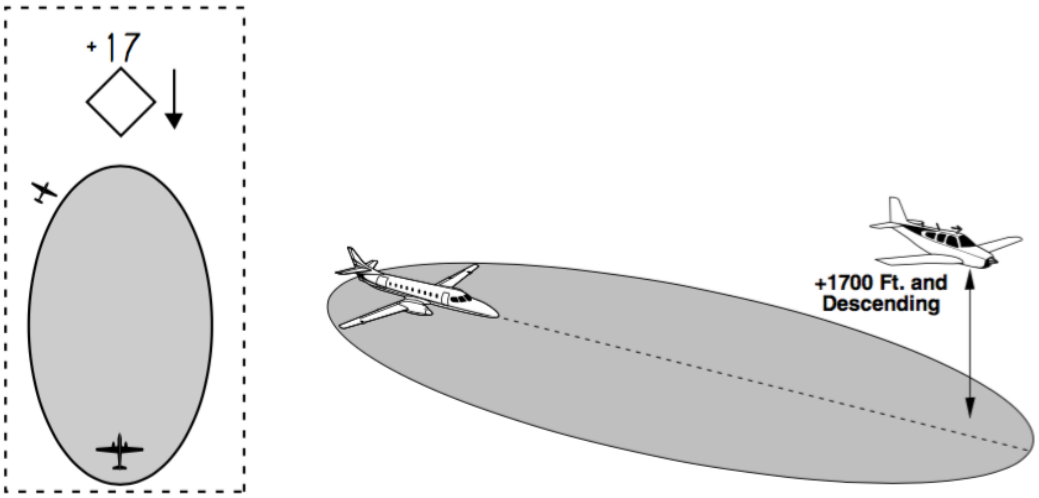
Hay que señalar que los datos referentes a los vuelos están limitados en este caso al área circundante de Valencia que es capaz de leer la antena SACTA, si bien siempre se podría utilizar un archivo de trazas de otra zona geográfica utilizando el mismo formato.

Se ha de aclarar que el TCAS es un instrumento que no se incluye en la configuración del *cockpit* real de la Cessna 172S. Se trata por tanto de un instrumento extra en la aeronave, ya que en el panel de instrumentos original no está incluído.

Con el objetivo de implementar un sistema TCAS de tipo I que sea lo más fiel posible a la realidad, se ha consultado la guía de uso de un TCAS real, utilizado en aeronaves de aviación general. Dicho TCAS es el denominado como *CAS 66A*, desarrollado por Honeywell Inc [50].

En las siguientes líneas de procede a describir de forma resumida el comportamiento del TCAS *CAS 66A*:

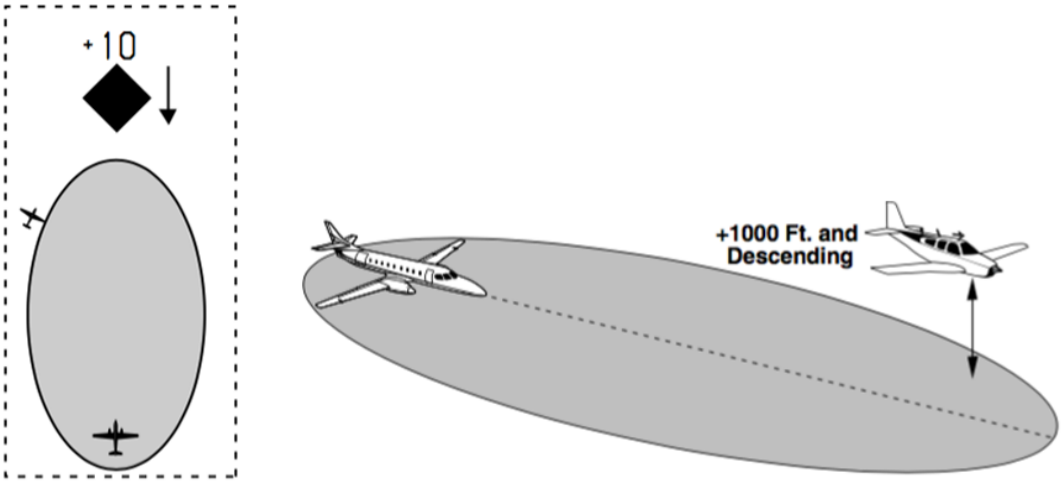
- **Tráfico sin riesgo:** Cuando una aeronave se encuentra en un rango mayor o igual a 5 NM, o con una diferencia de altitud mayor o igual a ±1500 ft, el TCAS muestra el icono de dicho avión con un rombo blanco y hueco. Una ilustración se muestra en la Figura 3.28.



(a) Vista en planta (b) Vista en perspectiva

Figura 3.28: Representación de una situación de tráfico sin riesgo. Fuente: [50]

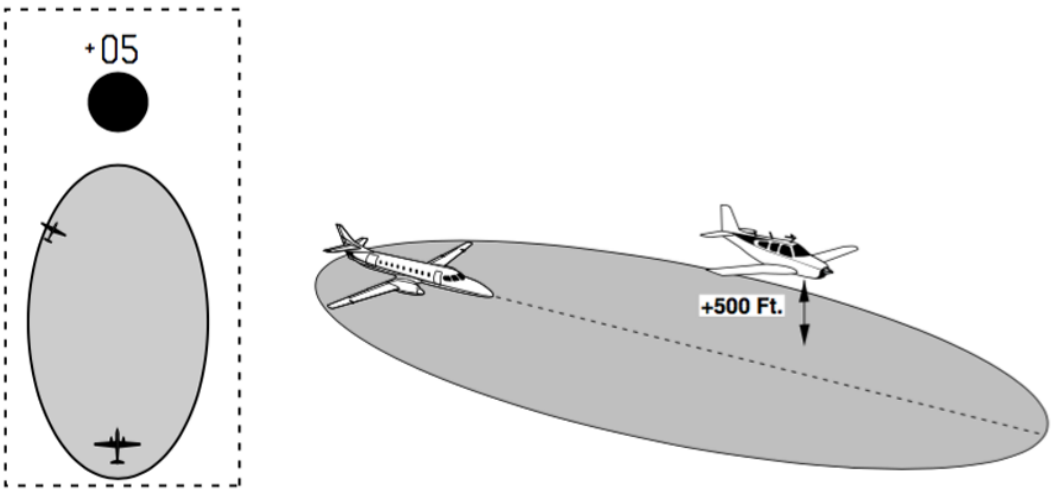
- **Tráfico con proximidad de intrusión:** Cuando una aeronave se encuentra en un rango menor o igual a 5 NM, o con una diferencia de altitud que se encuentra dentro de ±1500 ft, el TCAS muestra el icono de dicho avión con un rombo relleno amarillo. Un ejemplo de este tipo de tráfico se muestra en la Figura 3.29.



(a) Vista en planta (b) Vista en perspectiva

Figura 3.29: Representación de una situación de tráfico con proximidad de intrusión. Fuente: [50]

- **Aviso de tráfico (TA):** Cuando una aeronave se encuentra dentro del rango de proximidad, y tiene el *Closest Point of Approach* a menos de 30 segundos (o 15 dependiendo de la sensibilidad del sistema), se muestra un icono redondo y rojo, con una alerta de "*Traffic, Traffic*" en rojo sobre un fondo amarillo en la pantalla del TCAS. Además el sistema emite una alerta sonora para el piloto mediante el aviso "*Traffic, Traffic*". La situación de aviso de tráfico se esquematiza en la Figura 3.30.



(a) Vista en planta (b) Vista en perspectiva

Figura 3.30: Representación de una situación de aviso de tráfico. Fuente: [50]

Los números que se muestran encima de cada icono de cada avión muestran la

diferencia de altitud que existe entre dicho avión y el avión pilotado, medida en centenares de pies. Una flecha hacia arriba indica que dicha aeronave lleva una velocidad de ascenso superior a 500 ft/min, y una flecha hacia abajo indica que lleva una velocidad de descenso de más de 500 ft/min.

2.8.5. Implementación del código

La implementación del código se basa en los datos obtenidos mediante la antena de tráfico aéreo SACTA, los datos de posición del simulador X-Plane y el uso de las librerías de OpenMaps para el cálculo de distancia entre dos puntos. Además de esto, se hace uso del algoritmo de CPA presentado en la Sección 2.8.3.

Para la representación de las aeronaves que se encuentran en el arco de visión del TCAS, se ha implementado un algoritmo capaz de transformar las coordenadas en el sistema LLA de las aeronaves cercanas a unas coordenadas locales basadas en la representación gráfica de la interfaz. Con esto, de traducen los datos obtenidos de longitud, latitud y altura a un sistema de referencia local basado en los píxeles del display. Una vez convertidas las coordenadas de las aeronaves detectadas por la antena, se evalúa si dichas aeronaves se encuentran en el área de detección del TCAS. En caso afirmativo, se mostrará la aeronave siguiendo las reglas mostradas en la Sección 2.8.4 y que se basan en el manual del TCAS real *CAS 66A*, desarrollado por Honeywell Inc [50].

El resultado de dicha implementación se muestra en las siguientes figuras para cada uno de los estados del TCAS: tráfico sin riesgo, tráfico con proximidad de intrusión y aviso de tráfico.

- **Tráfico sin riesgo:** En la Figura 3.31 se muestra un ejemplo de cómo muestra el TCAS implementado una situación de tráfico sin riesgo basado en datos de vuelos reales.

Figura 3.31: Ilustración de tráfico sin riesgo en el TCAS

- **Tráfico con proximidad de intrusión:** En este caso, tal y como se indica en el manual del *CAS 66A* [50], el TCAS muestra el icono del avión con proximidad de intrusión mediante un rombo relleno amarillo. Esto se muestra en la Figura 3.32.

Figura 3.32: Ilustración de tráfico con proximidad en el TCAS

- **Aviso de tráfico (TA):** En este caso se muestra un icono redondo y rojo, con una alerta de "*Traffic, Traffic*.[en] rojo sobre un fondo amarillo en la pantalla del TCAS. En la realidad el sistema lo emite además con una voz, pero en el presente proyecto se ha implementado haciendo sonar un "*beep*.[en] el equipo. Esto se muestra en la Figura 3.33.

Figura 3.33: Ilustración de tráfico con riesgo de colisión en el TCAS

Capítulo 4

Implementación de interfaces con periféricos

1. Interfaz de comunicación con X-Plane

En el presente trabajo se ha implementado una interfaz de comunicación con el simulador X-Plane. El motivo de dicha implementación es el de poder alimentar de datos a los instrumentos y al autopiloto.

1.1. Elección del simulador

Durante la realización del proyecto se han analizado distintas alternativas en cuanto a motores de simulación que se podrían utilizar. Cabe destacar que en los requerimientos del presente proyecto no exigen la inclusión de un motor gráfico, ya que el proyecto que ocupa esta memoria ya implementa un entorno gráfico.

Las alternativas más conocidas de simuladores comerciales multiplataforma más conocidas son las siguientes:

- **Microsoft™ Flight Simulator X®(FSX):** Se trata de uno de los simuladores de vuelo comerciales más vendidos y utilizados en el mundo. A pesar de se un simulador muy utilizado tanto en el sector de ocio como en el sector del entrenamiento de pilotos, la incompatibilidad del mismo con el uso de distintos sistemas operativos desaconseja su uso en el presente trabajo.
- **X-Plane 10®:** Este simulador de vuelo se ha convertido en los últimos años en una opción muy utilizada en multitud de sectores como el ocio (en plataformas tradicionales y móviles), la formación de pilotos, y también en el sector de la investigación. La facilidad con la que se pueden implementar interfaces de comunicación UDP con el simulador y la capacidad de crear multitud de *plug-ins*

han convertido a este simulador en la opción preferida para multitud de aplicaciones profesionales relacionadas con autopilotos, UAVs e investigación.

- **FlightGear®:** Se trata de una opción de código libre multiplataforma muy completa y que utiliza el motor de simulación JSBSim. Los gráficos generados en FlightGear no gozan de la misma calidad que en los simuladores anteriores. Aunque a priori puede parecer una buena opción debido a su naturaleza de código abierto, la posibilidad de utilizar únicamente su motor de simulación sin necesidad de utilizar los recursos gráficos del simulador parece una opción mejor, tal y como se describe en el siguiente ítem.
- **JSBSim®:** Este motor de simulación de código libre es capaz de correr en multitud de sistemas operativos (Windows, Macintosh OS X y Linux, entre otros). La principal ventaja de la utilización de este motor de simulación es la eliminación de la capa gráfica presente en los simuladores presentados anteriormente. Con ello se puede conseguir un comportamiento más fluido de la interfaz implementada en el presente proyecto, así como reducir los requerimientos de *hardware* mínimos para la ejecución de la aplicación.

En la Tabla 4.1 se presenta un resumen de las principales virtudes y defectos de cada uno de los simuladores enumerados anteriormente con respecto al proyecto que ocupa esta memoria:

Tabla 4.1: Resumen de las características de los simuladores evaluados

Simulador	**Plataformas**	**Interfaz gráfica**	**Interfaz UDP**
FSX	Sólo Windows	Siempre activa	Dificultad moderada
X-Plane 10	Windows, Mac OS, Linux	Siempre activa	Menor dificultad
FlightGear	Windows, Mac OS, Linux	Siempre activa	Dificultad moderada
JSBSim	Windows, Mac OS, Linux	Inactiva	Mayor dificultad

En la tabla anterior se puede observar que hay dos claros candidatos para la implementación de la interfaz con el simulador, éstos son X-Plane y JSBSim. La opción más eficiente y óptima es sin duda la utilización del motor de simulación JSBSim, puesto que se prescinde de la interfaz gráfica y se reducen drásticamente los

requisitos de *hardware* de la aplicación creada en este proyecto.

Sin embargo, la mayor complejidad de la implementación de una interfaz UDP unido a la gran cantidad de desarrollos ya implementados con licencia GNU ha decantado la balanza en favor de la implementación de una interfaz de comunicación UDP con el simulador de vuelo X-Plane.

1.2. Desarrollo de la interfaz de comunicación con X-Plane

Antes de realizar un desarrollo completo de una interfaz de comunicación UDP con el simulador X-Plane, se ha realizado una búsqueda de alternativas ya implementadas disponibles en la red y en los desarrollos realizados anteriormente en el departamento de Informática de Sistemas y Computadores de la UPV.

En el trabajo final de carrera *Banco de pruebas para el diseño de autpilotos* [51] se implementó en Matlab®una interfaz que permitía obtener los datos de vuelo del simulador X-Plane para poder utilizarlos en el desarrollo de autopilotos. Si bien el código de dicho trabajo se podría portar al lenguaje de programación Java, se optó por intentar buscar herramientas ya implementadas en el lenguaje utilizado en el proyecto que ocupa esta memoria.

Tras una búsqueda en la red, se encontró una alternativa ya implementada en Java. Esta librería fue desarrollada por el Dr. Luiz Cantoni en su tesis doctoral: *Avaliaçao do uso da linguagem pddl no planejamento de missoes para robôs aéreos* [52]. El código se encuentra publicado en GitHub [1] bajo licencia GNU, lo que permite la utilización y modificación del código en este trabajo.

Puesto que la interfaz desarrollada por en [52] cumple con todos los requerimientos impuestos en este trabajo, se ha optado por utilizar dicha interfaz, implementando las correcciones necesarias para su correcta integración en la plataforma desarrollada en este proyecto. Con esto se consigue dedicar una mayor cantidad de tiempo al desarrollo de los instrumentos, la interfaz gráfica y el TCAS, elementos que diferencian a este proyecto de otros realizados en el pasado.

1.3. Implementación del código

En este apartado se trata con más detalle la implementación seguida en el proceso de comunicación con X-Plane. Implementación, que tal y como se ha comentado con anterioridad, está basada en los trabajos presentados en [52], sobre los que se han

[1] Código disponible en: `https://github.com/luizcantoni/x-pi`

realizado una serie de modificaciones.

La comunicación con X-Plane en tiempo real se realiza mediante la clase `XPlaneInterface`, la cual necesita la información de la IP, los puertos de envío y recepción de información por UDP y una hoja de información `DATAGroupConfig.xml`. Este documento contiene la información necesaria para extraer o enviar la información que X-Plane puede comunicar mediante UDP, y tiene el aspecto que muestra la Figura 4.1.

```
<DATAGroups>
    <DATAGroup name="simulation">
        <Entries>
            <DATA name="visRatio" message="0" parameter="3" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="grndRatio" message="0" parameter="5" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="flitRatio" message="0" parameter="6" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
        </Entries>
    </DATAGroup>

    <DATAGroup name="position">
        <Entries>
            <DATA name="ias" message="3" parameter="0" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="eas" message="3" parameter="1" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="tas" message="3" parameter="2" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="gs" message="3" parameter="3" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="lat" message="20" parameter="0" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="lon" message="20" parameter="1" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="altMsl" message="20" parameter="2" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="altAgl" message="20" parameter="3" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="altInd" message="20" parameter="5" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="X" message="21" parameter="0" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="Y" message="21" parameter="1" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="Z" message="21" parameter="2" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="vX" message="21" parameter="3" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="vY" message="21" parameter="4" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="vZ" message="21" parameter="5" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="P" message="16" parameter="0" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="Q" message="16" parameter="0" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="R" message="16" parameter="0" readOnly="true" xplaneName="" unit="" convertTo="" desc=""></DATA>
            <DATA name="distNm" message="21" parameter="7" readOnly="false" xplaneName="" unit="" convertTo="" desc=""></DATA>
        </Entries>
    </DATAGroup>
```

Figura 4.1: Aspecto del fichero `DATAGroupConfig.xml`

La clase `XPlaneInterface` arranca dos *threads*, uno para envío y otro para recepción de información, entre otros muchos procesos, que permiten con éxito la comunicación por UDP. Por otro lado, las clases que permiten interpretar la información UDP procedente de X-Plane y cuantificarla en un valor numérico, y viceversa, son las clases `XPlaneReader` y `XPlaneSender`.

La clase `XPlaneReader` consiste en un *thread* que lee la información de X-Plane mediante el método `getValue` de la interfaz `XPlaneInterface`. Desde este *thread* es desde donde se llaman a los distintos métodos implementados en la clase `Instrument` que permiten animar los instrumentos, el fondo, el *stick* y el *throttle* con la información procedente de X-Plane.

Por otro lado, la clase `XPlaneSender` es la que usa los métodos implementados en la clase `Instrument` para el *yoke* y para el *throttle* e interpreta la información según la interacción gráfica del usuario con estos elementos, para posteriormente enviarla por UDP al simulador.

2. Interfaz de comunicación con la antena de tráfico aéreo

Un instrumento como el TCAS solo tiene sentido con la presencia de más tráfico aéreo a parte de la aeronave que se está manejando, y de ahí que se haya implementado una comunicación con la una antena capaz de leer datos del transpondedor de las aeronaves. Dicha antena será la encargada de generar tráfico alrededor del avión manejado en el simulador.

2.1. Descripción

La antena de tráfico aéreo utilizada en el presente proyecto basa su funcionamiento en el uso de los datos enviados por el transpondedor que equipan las aeronaves que sobrevuelan el espacio aéreo. El transpondedor es un equipo que se encarga de difundir datos de interés de la aeronave, tales como su posición (latitud y longitud), altitud, velocidad, rumbo, etc.

Los transpondedores son un sistema fundamental para la gestión del tráfico aéreo por parte de los ATCs [53], y además son indispensables también para el correcto funcionamiento de sistemas de evitación de colisiones como el TCAS.

Con el objetivo de ser capaces de obtener datos en tiempo real sobre el tráfico aéreo en el área de Valencia, la Escuela Técnica Superior de Ingeniería del Diseño de la Universidad Politécnica de Valencia ha instalado una antena capaz de leer los datos difundidos por los transpondedores de las aeronaves. La utilización de los datos reales de tráfico aéreo que se obtienen tiene multitud de aplicaciones, siendo una de ellas la que ocupa el presente proyecto: la obtención de tráfico real para la implementación de un sistema de evitación de colisiones TCAS.

Puesto que la antena se encuentra en la Escuela Técnica Superior de Ingeniería del Diseño de la Universidad Politécnica de Valencia, hay que señalar que el rango del tráfico aéreo que detecta está centrado en esa zona, por lo que para que usar X-Plane junto con la lectura de la antena y tener algo de información en el TCAS es necesario que la simulación del vuelo en X-Plane se lleve a cabo dentro del rango de cobertura de la antena. Dicha antena está conectada a un equipo que comparte la información que recibe dentro de la red UPVNET, también accesible mediante VPN.

2.2. Menú de configuración

Para permitir la correcta configuración de la antena, se ha decidido implementar un menú gráfico de configuración. En la interfaz principal `Cockpit` se ha incluido el menú Setup » Select Antenna, el cuál abre el panel de selección de la antena como muestra la

Figura 4.2. Este panel es la interfaz `AntennaChooser`. En dicho panel se presentan dos opciones:

- Elegir la antena real, para lo que hay que introducir la IP y el Puerto.
- Leer una traza de archivos, para lo que hay un selector de archivos y una casilla en la que introducir el tiempo de intervalo de lectura.

Figura 4.2: Panel de selección de la antena

Seleccionada la antena se puede hacer clic en el botón *Start Antenna*, el panel se cierra y se inicia la lectura de la antena.

2.3. Implementación del código

La interfaz con la antena de tráfico aéreo se ha implementado siguiendo las directrices indicadas en la asignatura de *Sistemas de Gestión de Vuelo por Computador* [42], se ha implementado un código capaz de conectarse a la antena e interpretar los mensajes recibidos por ésta. En el presente proyecto se ha utilizado un código basado en el que se pone a disposición de los alumnos en [42], al que se le han aplicado diversos cambios para optimizarlo.

El código de la antena se basa en la implementación de un *Listener* que lee los datos de los vuelos. Los vuelos y sus datos se guardan en un *HashMap* utilizando el número del vuelo como identificador. Por cada vuelo leído, se analiza si dicho vuelo se encuentra ya en el *HashMap* y en caso afirmativo, se procede a sobrescribir únicamente aquella información que ha sufrido cambios. En caso de que el vuelo no se encuentre en la base de datos, se procede a incluirlo en la misma. El programa es capaz de saber los datos que han variado en un vuelo gracias al *Label* del tipo de mensaje que se recibe.

En cuanto a la estructura de los datos leídos por la antena, es relativamente sencillo de analizar. Los datos enviados por el transpondedor de una aeronave siguen una estructura estandarizada. Esta estructura se puede encontrar en diversos ejemplos en la literatura, sin embargo, el presente proyecto se ha basado en los datos presentados en [54] para que el código sea capaz de interpretar los datos leídos por la

antena de tráfico aéreo.

Las Figuras 4.3, 4.4 y 4.5 muestran parte de la estructura de datos seguida en el envío de datos mediante el transpondedor. Estas figuras se muestran a modo de ejemplo y no contienen todo lo necesario para interpretar la estrucutra completa de los datos, si bien son útiles para ilustrar al lector el proceso seguido en este proyecto. Para ver la estructura completa de los datos del transpondedor, se insta al lector a consultar la referencia [54].

ID	Type		Description
MSG,1	ES Identification and Category	DF17 BDS 0,8	
MSG,2	ES Surface Position Message	DF17 BDS 0,6	Triggered by nose gear squat switch.
MSG,3	ES Airborne Position Message	DF17 BDS 0,5	
MSG,4	ES Airborne Velocity Message	DF17 BDS 0,9	
MSG,5	Surveillance Alt Message	DF4, DF20	Triggered by ground radar. Not CRC secured. MSG,5 will only be output if the aircraft has previously sent a MSG,1, 2, 3, 4 or 8 signal.
MSG,6	Surveillance ID Message	DF5, DF21	Triggered by ground radar. Not CRC secured. MSG,6 will only be output if the aircraft has previously sent a MSG,1, 2, 3, 4 or 8 signal.
MSG,7	Air To Air Message	DF16	Triggered from TCAS. MSG,7 is now included in the SBS socket output.
MSG,8	All Call Reply	DF11	Broadcast but also triggered by ground radar

Figura 4.3: Tipos de mensajes recibidos. Fuente [54].

Field 1:	Message type	(MSG, STA, ID, AIR, SEL or CLK)
Field 2:	Transmission Type	MSG sub types 1 to 8. Not used by other message types.
Field 3:	Session ID	Database Session record number
Field 4:	AircraftID	Database Aircraft record number
Field 5:	HexIdent	Aircraft Mode S hexadecimal code
Field 6:	FlightID	Database Flight record number
Field 7:	Date message generated	As it says
Field 8:	Time message generated	As it says
Field 9:	Date message logged	As it says
Field 10:	Time message logged	As it says

Figura 4.4: Mensajes estándar, incluidos en todos los tipos de mensajes. Fuente [54].

Field 11:	**Callsign**	An eight digit flight ID - can be flight number or registration (or even nothing).
Field 12:	**Altitude**	Mode C altitude. Height relative to 1013.2mb (Flight Level). Not height AMSL..
Field 13:	**GroundSpeed**	Speed over ground (not indicated airspeed)
Field 14:	**Track**	Track of aircraft (not heading). Derived from the velocity E/W and velocity N/S
Field 15:	**Latitude**	North and East positive. South and West negative.
Field 16:	**Longitude**	North and East positive. South and West negative.
Field 17:	**VerticalRate**	64ft resolution
Field 18:	**Squawk**	Assigned Mode A squawk code.
Field 19:	**Alert (Squawk change)**	Flag to indicate squawk has changed.
Field 20:	**Emergency**	Flag to indicate emergency code has been set
Field 21:	**SPI (Ident)**	Flag to indicate transponder Ident has been activated.
Field 22:	**IsOnGround**	Flag to indicate ground squat switch is active

Figura 4.5: Mensajes con información específica de la aeronave. Fuente [54]

Capítulo 5

Implementación del autopiloto

1. Descripción

El autopiloto es uno de los pilares básicos de la implementación del presente proyecto debido tanto a desarrollo como a su capacidad de crecimiento y su importancia en posibles trabajos futuros. Hoy en día las técnicas de control y guiado aplicadas a las aeronaves están al orden del día, no sólo en aplicaciones autónomas como los UAVs, sino en la práctica totalidad de aeronaves modernas.

El nivel de automatización de las aeronaves va constantemente en aumento y como se ha comentado con anterioridad, el desarrollo de aeronaves no tripuladas no hace más que recalcar la importancia que tiene un sistema de autopiloto hoy en día.

El diseño de un autopiloto sería un tema que podría ocupar perfectamente la longitud de un proyecto como el que se expone, es por esta razón que el autopiloto implementado en este trabajo tiene un importante margen de mejora. Si bien es cierto que se consiguen cumplir de forma exitosa las funciones deseadas y que se plantearon durante la fase de diseño conceptual del proyecto.

2. Arquitectura del autopiloto

Este autopiloto estará basado en la utilización de controladores PID en cascada. Esta forma de afrontar el problema es muy común en la industria, siendo utilizada en numerosas aplicaciones de UAVs. El uso de controladores en cascada permite ajustar de forma independiente los bucles de guiado y de control de la aeronave, lo que facilita sobremanera la obtención de una ley de guiado óptima. Un controlador en cascada no es más que la inclusión de distintos controladores PID en serie, tal y como se muestra en la Figura 5.1.

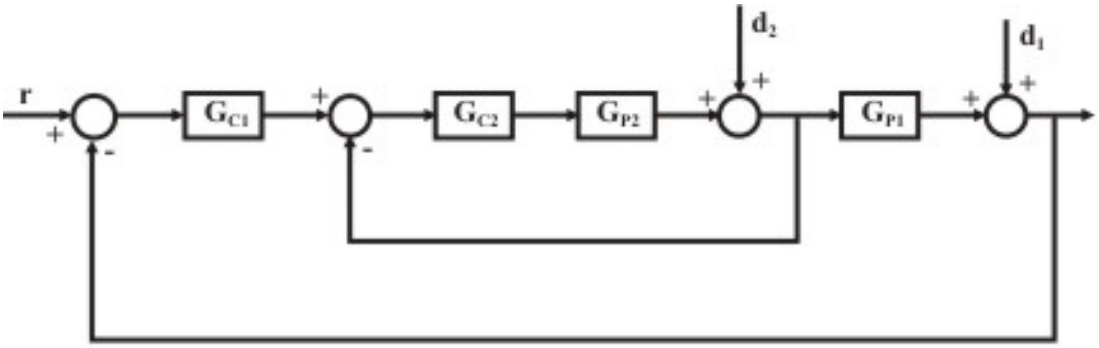


Figura 5.1: Topología de un esquema de control en cascada.

El guiado de la aeronave se realiza de forma distinta según la fase de vuelo de la aeronave; por ejemplo, durante fases de rodadura en tierra no se pueden utilizar los alerones para girar la aeronave, sino que se utiliza el timón de profundidad y la rueda delantera. Es por esto que el autopiloto implementado cambia su funcionamiento en función de la fase de vuelo a la que se encuentre la aeronave. De esta forma la lógica de guiado será distinta para cada fase de vuelo, así como los controladores PID utilizados.

Por simplicidad y para permitir la finalización del proyecto en los plazos de tiempo establecidos, se ha decidido implementar controladores lo más simples posibles ya que de esta forma se disminuyen las probabilidades de fallo, consiguiendo así un sistema más seguro. Sin embargo, no es menos cierto que con una lógica de control más compleja se podría conseguir un comportamiento más preciso de la aeronave, aunque esto se deja para trabajos futuros.

3. Implementación de los algoritmos de guiado y control

Tal y como se ha comentado en la introducción, el objetivo principal de este proyecto es la implementación de los distintos algoritmos de guiado y control que permitan operar de forma segura la aeronave Cessna C-172 Skyhawk en el simulador de vuelo X-Plane.

Con la intención de dotar de un mayor realismo al trabajo y siguiendo las directrices del profesorado, se han implementado los algoritmos de guiado y control correspondientes a las siguientes fases de vuelo:

- *Rollout* o fase de rodadura en pista de despegue
- Rotación en pista hasta elevar la aeronave del suelo
- Fase de ascenso (*Climbing*)
- Fase de vuelo en crucero

Cada una de las fases expuestas anteriormente tendrá asociadas unas acciones de control sobre los siguientes sistemas de la aeronave:

- Palanca de gases o *Throttle*
- Timón de cola o *Rudder*
- Alerones
- Timón de profundidad o elevador
- Frenos
- Flaps

En las siguientes páginas se procederá a detallar tanto la lógica de fases y transiciones entre fases como la filosofía de guiado y control utilizada en cada una de las fases para cada uno de los canales de control de la aeronave.

A modo de ejemplo, se procede a detallar la logica implementada utilizando como referencia el procedimiento de salida por instrumentos SID de la pista 12 del aeropuerto de Valencia.

La ruta a seguir por la aeronave en el presente proyecto se muestra resaltada en rojo en la Figura 5.2. Como se puede ver, dicha ruta se corresponde con el procedimiento de salida estándar SID desde la pista 12 del aeropuerto de Valencia (LEVC). Por tanto la ruta consistirá de una fase de rollout y de rotación siguiendo el rumbo de la pista (119^o) para posteriormente variar el rumbo a 124^o al entrar en la fase de climbing. Una vez alcanzada una altura sobre le nivel del mar de 2000 pies se transicionará hacia la fase de crucero, donde en este caso se continuará con un rumbo de 124^o.

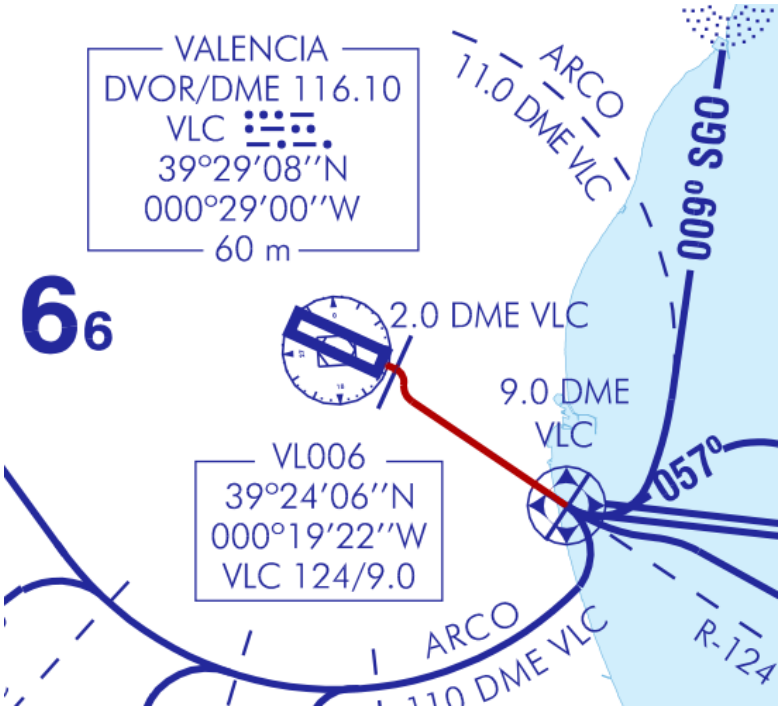


Figura 5.2: Ruta seguida por la aeronave

3.1. Fase de rollout

La fase de *rollout* consiste en la etapa de de rodadura en pista para el despegue de la aeronave. En este caso, y siguiendo los consejos del profesorado, se ha considerado que dicha fase comprende el intervalo de rodadura en pista desde el momento inicial hasta que la aeronave alcanza la velocidad de rotación $V_2 = 80\,\text{KIAS}$. Como se puede ver, dicha velocidad se expresa como velocidad indicada al nivel del mar, algo a tener en cuenta en cuando se trate de despegar en condiciones atmosféricas distintas de la atmósfera ISA o en aeródromos situados a una altitud importante. Una vez alcanzada la velocidad de rotación, se transicionará a la fase de rotación.

El guiado y control implementado en esta fase asume que la aeronave se encuentra en cabecera de pista, con el motor en funcionamiento y con el freno de aparcamiento accionado.

3.1.1. Guiado y control

En este caso se ha actuado sobre los canales de control de la siguiente forma:

Throttle

Fijo a un valor de $Thr = 1$, es decir, máxima potencia

Rudder

El *rudder* se utiliza en este caso para el guiado horizontal de la aeronave sobre la pista, ya que acoplada a acción del timón de cola se encuentra la rueda del tren principal, lo que permite controlar la aeronave en tierra. En este caso se han realizado distintas pruebas para la correcta implementación de las acciones de control. En primer lugar, el parámetro utilizado para el control es el *heading* o rumbo de la aeronave, de esta forma, se comanda un rumbo objetivo que será comparado con el rumbo actual de la aeronave para obtener el error de rumbo y posteriormente realizar la corrección correspondiente mediante un controlador de tipo proporcional.

En este caso, debido a la marcada tendencia que tiene la Cessna C-172 a desviarse hacia la izquierda, se ha planteado la implementación de controlador de tipo proporcional-derivativo (controlador PD). El objetivo era intentar suavizar la respuesta de la aeronave para así conseguir que se realizase la fase de *rollout* sin desviarse del eje de la pista. Desgraciadamente, debido a la falta de tiempo, dicho controlador no pudo ser ajustado convenientemente.

Sin embargo, después del intento fallido de ajustar correctamente el controlador PD, se ha implementado un nuevo enfoque basado en la utilización de un controlador proporcional (controlador P) que realizase un offset en la acción de control, de forma que la acción de control comandada sería equivalente a lo mostrado en la ecuación (5.1). De esta forma se consigue contrarrestar la tendencia a desviarse hacia la izquierda que tiene la aeronave utilizada en este proyecto, con lo que se facilita sobremanera el ajuste del controlador y se consigue realizar íntegramente la fase de *rollout* sin desviarse del eje de la pista.

$$\delta_r = (\mathrm{Hdg}_{com} - \mathrm{Hdg})K_p + \mathrm{offset} \tag{5.1}$$

Alerones

En esta fase, debido a que la aeronave se encuentra en tierra, no se utiliza el canal de control de los alerones, por lo que la acción sobre los mismos será nula.

Elevador

Debido a que en la fase de *rollout* no se pretende separarse del suelo, el canal de control de los alerones no se utilizará en la misma.

Frenos

Los frenos se soltarán al inicio de la fase y se mantendrán sin accionar durante el resto de la misión implementada.

Flaps

En este caso no se han utilizado los flaps para el despegue, sin embargo, se podrían utilizar sin problema alguno indicando el valor deseado en la ventana de configuración del autopiloto.

3.2. Fase de rotación

La fase de rotación comienza cuando se alcanza $V_2 = 80\,\text{KIAS}$ y se prolonga hasta que la aeronave se encuentra a 15 pies de la pista. La altura sobre la pista se puede obtener fácilmente mediante la lectura del parámetro de AGL que envía X-Plane. Una vez alcanzados los 15 pies sobre nivel de la pista la aeronave transiciona a la fase de ascenso o *climbing*.

3.2.1. Guiado y control

La lógica de control utilizada en este caso es la siguiente:

Throttle

Fijo a un valor de $Thr = 1$, es decir, máxima potencia

Rudder

En este caso se deja de actuar sobre el rudder por seguridad, ya que la fase de rotación abarca la transición entre el control de la aeronave en tierra y en el aire. Además, debido a que el intervalo de tiempo entre el incio de la fase de rotación y el despegue de la aeronave es pequeño, la desviación sobre el umbral de pista en tierra es mínima.

Alerones

En este caso los alerones son los encargados del control lateral de la aeronave. En este caso se seguirá una estructura de dos bucles anidados, donde se podrán diferenciar claramente el bucle de guiado (el más externo) y el bucle de control (el más interno).

La función del bucle de guiado será la de calcular el ángulo de alabeo requerido para corregir un error en el rumbo de la aeronave. La función de guiado implementada asume que se pretende realizar un giro estándar de 2 minutos y con ello calcula el ángulo de alabeo necesario para realizar dicho giro que corregirá el error de rumbo. Esto se realiza mediante un controlador proporcional común. Además, en esta fase es

conveniente indicar unos ángulos de saturación adecuados, que no permitan que se comanden grandes ángulo de alabeo que puedan propiciar que la punta de una de las alas impacte con la pista.

El bucle de control se encargará de garantizar que la aeronave obtenga el ángulo de alabeo requerido por el bucle de guiado. En este caso se ha utilizado un controlador de tipo P.

Elevador

Durante esta fase se realiza un *input* positivo de elevador que permita que la aeronave se eleve. Debido a que la transición a la case de ascenso se produce a los pocos instantes, no es necesario realizar un control adaptativo de la deflexión del elevador, ya que dicho control se realizará a los pocos instantes una vez se haya transicionado a la fase de ascenso.

Frenos

En este caso, como en el anterior, los frenos se mantienen desactivados.

Flaps

Los flaps en la fase de rotación han de estar necesariamente en la misma posición en la que se encuentran en la fase de rollout, por lo que en esta fase su deflexión es también nula.

3.3. Fase de ascenso

La aeronave accede a la fase de ascenso una vez que se encuentra a 15 pies sobre la superficie de la pista y seguirá en dicha fase hasta que alcance una altitud configurable, que en el caso del procedimiento SID de la pista 12 de Aeropuerto de Valencia es de 2000 pies sobre el nivel del mar. Siguiendo también el procedimiento normalizado del aeropuerto de Valencia, se ha comandado que la aeronave siga un rumbo de 124 grados para alcanzar la radioayuda correspondiente en el procedimiento. En caso de tratarse de otro procedimiento de salida o de otro aeropuerto dicho rumbo también se puede configurar.

3.3.1. Guiado y control

La lógica de control utilizada para la fase de ascenso es la que se muestra a continuación:

Throttle

Fijo a un valor de $Thr = 1$, es decir, máxima potencia

Rudder

En este caso el rudder no se controla y se mantiene en su posición central. Cabe destacar que si bien el timón de cola no se utiliza para realizar cambios de rumbo, sí que tiene su utilidad en casos de viento lateral o para la realización de giros coordinados. La implementación de un controlador de giro coordinado no es excesivamente complicada, sin embargo, no se trata en este proyecto debido a que no se trata de uno de los objetivos principales del mismo.

Alerones

De igual manera que en la fase anterior, los alerones son los encargados de corregir el rumbo de la aeronave. En este caso, sin embargo, se ha optado por ir un paso más allá e implementar una lógica de guiado más compleja que tenga en cuenta la distancia a la que se encuentra la aeronave de la trayectoria rectilínea que debe seguir.

La intención de este algoritmo es obtener unos errores tanto de rumbo como de error lateral (*cross-track error*) lo más cercanos a 0 posible. Para ello se aplica la siguiente ley de control, basada en la inclusión de dos ganancias proporcionales:

$$a = K_1(\Psi_r - \Psi) + K_2 d \tag{5.2}$$

Donde la definición de los parámetros anteriores se puede observar en la Figura 5.3.

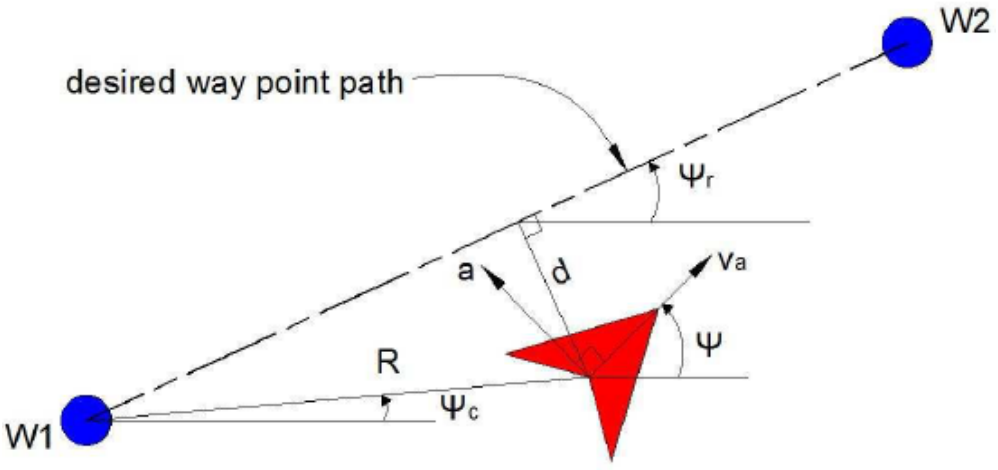


Figura 5.3: Definición de los parámetros del algoritmo de guiado lateral. Fuente: [42]

El parámetro d que indica la distancia entre la aeronave y la trayectoria a seguir, se puede estimar fácilmente mediante la ecuación (5.3).

$$d = R\sin(\Psi_r - \Psi_c) \tag{5.3}$$

Donde R es la distancia entre la aeronave y el punto inicial que define la ruta.

Cabe mencionar que en el caso que nos ocupa es realmente sencillo calcular la distancia d, ya que se obtienen las coordenadas exactas de la posición de la aeronave mediante X-Plane. Sin embargo, en el caso de una aplicación real, esto puede no ser tan sencillo, ya que no necesariamente se tendrían las coordenadas de la aeronave mediante sistemas GNSS por lo que podría ser necesario recurrir a distintas radioayudas para posicionar los puntos de la ruta y poder calcular la posición relativa de la aeronave con respecto a dichos puntos.

En este caso, tanto la distancia R como el ángulo Ψ_c se obtienen a partir de las coordenadas del punto de inicio de la ruta (W1 en la Figura 5.3) y de la aeronave. Para ello se hace uso de la librería de código libre OpenMap™ y concretamente del método `distance()`, el cual hace uso de dos puntos dados por longitud y latidud para devolver la distancia entre ellos. Para obtener dichos resultados es necesario guardar en memoria la coordenada del punto WP1 en el que empieza la ruta definida, algo que en la implementación de este trabajo se realiza la primera vez que se transiciona a la fase de ascenso.

Una vez obtenidos todos los parámetros correspondientes, el procedimiento a seguir es muy similar a lo realizado en la fase anterior: primero se calcula la el ángulo de alabeo correspondiente y después se calcula la deflexión del alerón δ_a mediante un controlador proporcional.

Elevador

El elevador en este caso controla la velocidad verdadera o *True Airspeed*, con una velocidad objetivo de $V_c = 90\,\text{KTAS}$. En este caso, si la velocidad es menor a la comandada, la aeronave reducirá su ángulo de cabeceo, lo que le permitirá ganar velocidad y en caso de que la velocidad se mayor a la comandad se procederá de modo inverso.

El control del elevador se subdivide en dos bucles: un bucle de guiado que utiliza la información del error de velocidad para comandar un ángulo de pitch objetivo y un bucle de control que comanda la deflexión de elevador δ_e para alcanzar dicho pitch.

Frenos

Al tratarse de una fase en vuelo, es obvio que los freno se econtrarán inutilizados en este caso.

Flaps

La deflexión de los flaps en ascenso será nula en este caso, ya que como se ha comentado con anterioridad, no se han utilizado en este proyecto.

3.4. Fase de crucero

La transición a la fase de crucero se realizará al llegar a una altura sobre el nivel del mar de 2000 pies. En este caso, la fase de crucero permite la inclusión de distintos parámetros tales como la velocidad de vuelo (que en este caso es $V = 100KTAS$), el rumbo a seguir y la altura del waypoint de destino.

Cabe destacar que en esta fase se han desacoplado el control de velocidad y el control de altura, siendo el primero llevado a cabo por el motor y el segundo por el elevador. El autor es plenamente consciente de que existen diversas filosofías de control más eficientes que acoplan el control de la velocidad y la altura y que se basan en un balance energético. Sin embargo, debido a las limitaciones de tiempo y a la mayor complejidad de dicho modelo energético, se ha optado por realizar una aproximación más simple que consiste en el desacople del control de velocidad y altura.

3.4.1. Guiado y control

La lógica de control utilizada en la fase de crucero es la siguiente:

Throttle

En este caso, tal y como se ha comentado, el control sobre la palanca de gases permitirá controlar la velocidad de la aeronave. Dicho control se puede realizar fácilmente mediante un controlador proporcional al que se le ha incluído un offset, ya que la posición requerida en la palanca de gases para una velocidad de $V = 100KTAS$ será del orden del 60-70 % del total. De esta forma se consigue un buena respuesta dinámica a los cambios en la velocidad comandada.

Rudder

Del mismo modo que en el caso de la fase de ascenso, no se ha implementado control alguno sobre la superfície del timón de cola por los motivos expuestos anteriormente.

Alerones

El control de alerones en la fase de crucero es totalmente análogo al control de los mismos en la fase de ascenso.

Elevador

El control sobre el elevador se utiliza en este caso para conseguir que la aeronave sea capaz de controlar su altura de vuelo. En este caso, el sistema de guiado mide el error de altura y comanda un ángulo de cabeceo proporcional a dicho error. El bucle de control se encarga de dar el *input* de alerón necesario para alcanzar el ángulo de cabeceo deseado.

En este caso se ha querido aportar una implementación algo distinta a lo visto anteriormente y se ha optado por utilizar un control Proporcional-Integral (control PI) que permite un mejor ajuste del bucle de control. Dicho controlador PI se ha implementado en el bucle de control, ya que en los casos en los que se comandan ángulos de cabeceo grandes, la dinámica del sistema deja de ser lineal, lo que dificulta el ajuste del controlador proporcional. La ventaja del uso de un controlador PI reside en le hecho de que el error acumulado en el tiempo (la integral del error cometido) se multiplica por una segunda ganancia K_i y su efecto se suma al del controlador proporcional. De esta forma se ha conseguido eliminar el error de altura en el estacionario sin necesidad de recurrir a esquemas más complejos como los basados en un alance de energía. La integral del error se puede realizar numéricamente de forma bastante sencilla mediante el uso del método Newton-Cotes, donde únicamente es necesario saber el paso temporal en cada iteración del bucle de control, el estado anterior de la variable y su estado actual. Dicho método de cálculo de las integrales se puede ver en la Figura 5.4 Sin embargo, cabe destacar que ésta forma de calcular las integrales, si bien es rápida y sencilla, es menos exacta que otras, por lo que se remarca que en un futuro podría ser interesante sustituir dicho cálculo por la regla trapezoidal o mejor aún, por la regla se Simpson.

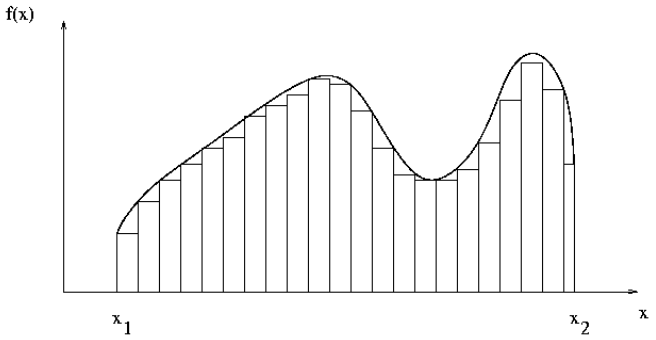


Figura 5.4: Método de integración de Newton-Cotes

La inclusión del controlador PI en el bucle de control ha permitido ajustar razonablemente bien el control de altura, sin embargo, dicho control no es lo suficientemente fino como para poder llegar a implementarlo en un caso real y debería de trabajarse en mayor profundidad para reducir su tiempo de estabilización y a la vez conseguir un comportamiento más suave.

Frenos

En la fase de crucero, los frenos están desactivados al igual que en fases anteriores.

Flaps

En las fases anteriores se ha indicado que los flaps no se han utulizado pero que se podría haber hecho. En la fase de crucero, sin embargo, nunca se llegaría a utilizar los flaps y éstos deberían de desactivarse de forma progresiva bien al entrar en la fase de crucero, o bien a partir de un cierto punto en la fase de ascenso.

4. Menú de configuración

Para facilitar la configuración del autopiloto y poder configurar fácilmente el mismo para la realización de distintos procedimientos SID, se ha implementado un menú gráfico de configuración similar al implementado en la antena.

El menú se encuentra en la pestaña Setup » Set Autopilot y al acceder al mismo se muestra la ventana que aparece en la Figura 5.5.

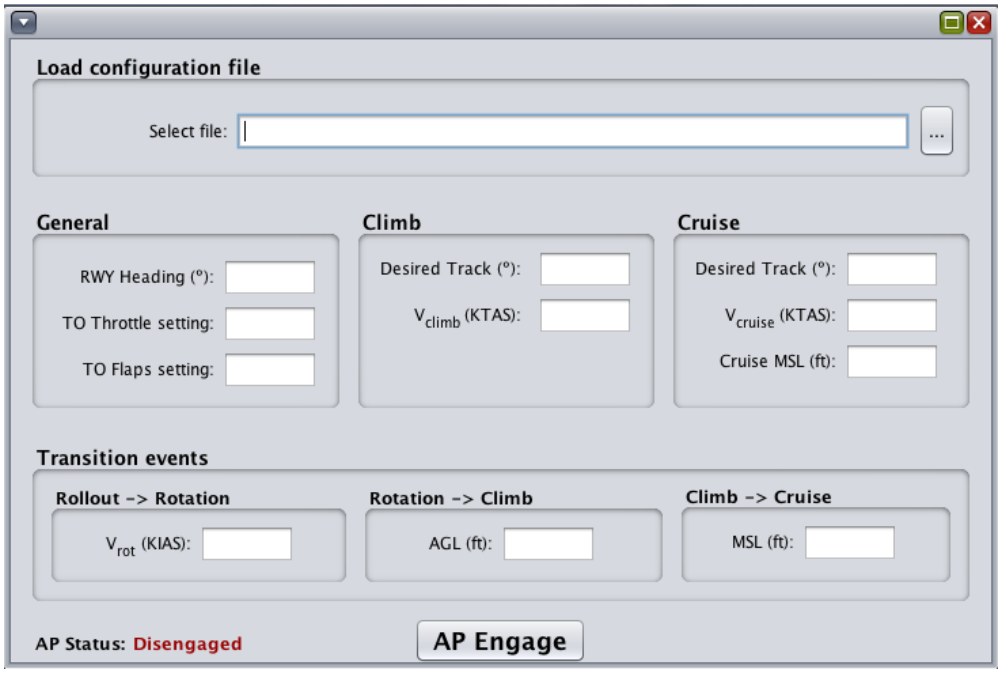


Figura 5.5: Menú de configuración del autopiloto

4.1. Parámetros de configuración

En la parte central del menú mostrado en la Figura 5.5 se encuentran los datos relativos al control de las distintas fases de vuelo, ordenados de la siguiente forma:

- **General**
  - **RWY Heading:** Se trata del rumbo de la pista desde la que se va a iniciar el despegue
  - **TO Throttle Setting:** Se trata del setting de la palanca de gases que se va a utilizar durante el despegue hasta llegar a la fase de crucero.
  - **TO Flaps Setting:** Se trata del setting de flaps que se utilizará durante el despegue.
- **Climb**
  - **Desired Track:** Es el ángulo de track que debe seguir la aeronave al iniciar la fase de ascenso.
  - $\mathbf{V_{climb}}$: Es la velocidad verdadera (*True Airspeed*) a la que se pretende que la aeronave realice el ascenso (en nudos).
- **Cruise**
  - **Desired Track:** Es el ángulo de track que debe seguir la aeronave al iniciar la fase de crucero (en nudos).

- **V_{cruise}**: Es la velocidad verdadera (*True Airspeed*) a la que se pretende que la aeronave viaje en su fase de crucero.
- **Cruise MSL:** Es la altura sobre el nivel del mar a la que se debe de realizar el vuelo de crucero (en pies)

Además de dichos parámetros, se incluyen los valores utilizados para las transiciones entre fases, situados en la parte inferior del menú mostrado en la Figura 5.5.Dichos parámetros se muestran con el siguiente orden:

- **Rollout → Rotation**
  - **V_{rot}**: Velocidad de rotación; es la velocidad a la que se inicia la maniobra de rotación y se utiliza el valor de la velocidad indicada en nudos.
- **Rotation → Climb**
  - **AGL:** Es la altura sobre el nivel de pista a la que se transiciona desde la fase de rotación a la fase de ascenso.
- **Climb → Cruise**
  - **MSL:** Es la altitud sobre el nivel del mar a la que se realiza la transición entre la fase de ascenso y la fase de crucero. Para una transición más suave se recomienda utilizar un valor cercano a la altitud de crucero, aunque ligeramente inferior.

4.2. Indicador de estado y botón Engage/Disengage

En la parte inferior del menú mostrado en la Figura 5.5 se pueden ver dos elementos: el indicador de estado (parte inferior izquierda de la imagen) y un botón de paro o de puesta en marcha.

El indicador de estado tiene como función mostrar al usuario el estado del autopiloto. El indicador muestra uno de los dos estados disponibles:

- **AP Status: Disengaged**
- **AP Status: Engaged**

La función del botón Engage/Disengage es justamente la de ejecutar o detener el autopiloto. El botón implementado cambia el mensaje impreso sobre el mismo según el estado actual del autopiloto, tal y como se puede observar en la Figura 5.6

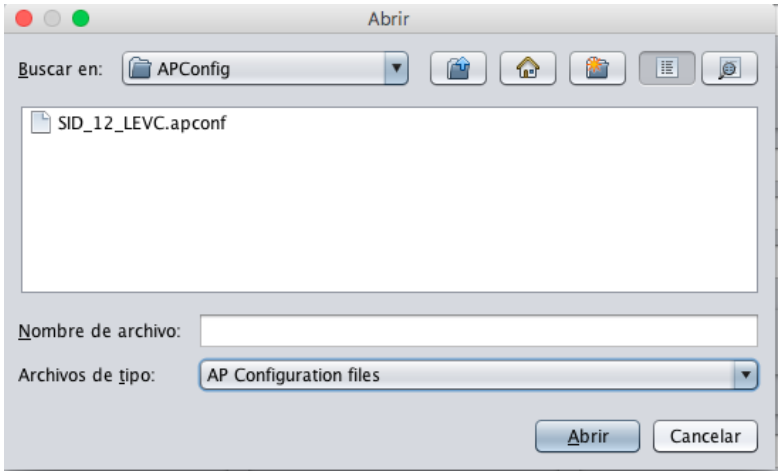
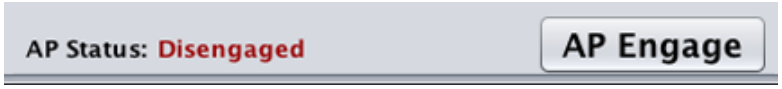


Figura 5.7: Selector de archivos de configuración del autopiloto



(a) Autopiloto sin ejecutar

(b) Autopiloto en funcionamiento

Figura 5.6: Indicador de estado y botón Engage/Disengage del autopiloto según el estado del mismo

4.3. Archivos de configuración del autopiloto

Para facilitar la utilización del autopiloto y para permitir que un usuario poco experimentado sea capaz de ejecutar el autopiloto de forma satisfactoria se ha decidido incluir la opción de cargar la configuración del autopiloto mediante un archivo. Este archivo de configuración contiene en su interior todos los parámetros significativos para la puesta en marcha del autopiloto.

Para cargar el archivo de configuración se puede escribir directamente la ruta al mismo o bien se puede presionar sobre el botón [...]. En este caso se abrirá una ventana con un selector de archivos, tal y como se puede ver en la Figura 5.7. La ruta por defecto en la que se abrirá este selector de archivos es la correspondiente al directorio **APConfig**, que es la carpeta por defecto en la que se guardan los archivos de configuración del autopiloto.

El tipo de archivo utilizado en este caso es un archivo con la extensión `*.apconf`. Como se puede ver la Figura 5.5, se ha seleccionado por defecto dicha extensión en el selector de archivos, para que únicamente se muestren los archivos válidos para ser

utilizados en la configuración del autopiloto.

4.3.1. Formato del fichero de configuración

El formato del fichero `*.apconf` es el que se muestra en la Figura 5.8. Todas las líneas precedidas por "//" serán ignoradas por el lector de archivos de configuración, al ser consideradas como comentarios. Además, para el corrector funcionamiento del autopiloto es esencial que se incluyan los datos en el orden descrito en la Figura 5.8.

```
////////////////////////////////////////////////////////////////////////////////
//                          AUTOPLOT CONFIGURATION FILE                       //
// This file contains the information required by the autopilot to run        //
//                                                                            //
// Author: Lucas Peris Lozano                                                 //
//                                                                            //
//    Date: 10/02/16                                                          //
//                                                                            //
// Data format: The AP variables must be writtten in the following order:     //
//              1. hdg_rwy : Runway heading                          [deg]    //
//              2. Vrot    : Rotation speed                          [KIAS]   //
//              3. aglCmb  : Above Ground Level alt. tans. ROT->CMB  [ft]     //
//              4. VtCmb   : Target Climb True Airspeed              [KTAS]   //
//              5. DTK_CMB : Des. Track for climb phase              [deg]    //
//              6. mslCr   : Des. cruise Mean Sea Level altitude     [ft]     //
//              7. VtCr    : Target Cruise True Airspeef             [KTAS]   //
//              8. DTK_CR  : Des. Track for cruise phase             [deg]    //
//              9. TO_throt: Throttle setting for takeoff (range->0-1) [-]    //
//             10. TO_flaps: Flaps setting for takeoff (range->0-1)    [-]     //
//                                                                            //
// ALL THE DATA MUST BE PRECEDED BY THE STATEMENT "value"                     //
////////////////////////////////////////////////////////////////////////////////
// CONFIGURATION FILE FOR: SID LEVC RWY 12

// hdg_rwy
value 116.15
// Vrot
value 80
// aglCmb
value 15
// VtCmb
value 90
// DTK_CMB
value 124
// mslCr
value 2000
// VtCr
value 100
// DTK_CR
value 124
// TO_throt
value 1
// TO_flaps
value 0
```

Figura 5.8: Ejemplo de un fichero de configuración `*.apconf`

El requerimiento de la inclusión de la palabra "value" antes de cada valor obedece a

razones de capacidad de crecimiento. Si en un futuro se plantea el rediseño de la interfaz de configuración del autopiloto, se puede querer incluir cadenas de texto en el archivo de configuración. Esto permitiría indicar en el menú de configuración el nombre de la salida SID cuya configuración se ha cargado, entre otras cosas. Con la inclusión de la palabra "value" se especifica que los dígitos siguientes corresponden a un valor numérico y no a una cadena de caracteres.

4.4. Mensajes de aviso y error

En el menú de configuración se han añadido distintos mensajes de aviso y error para prevenir posibles fallos del programa debido a una mala adición de datos.

Los tipos de error contemplados en la ventana de configuración del autopiloto son:

- **Errores en la escritura de los datos.**
- **Errores en el formato del archivo de configuración.**

4.4.1. Errores en la escritura de los datos

Esta categoría engloba todos los errores debidos bien a la falta de datos o bien a la una inclusión errónea de los mismos (valores negativos, letras, caracteres extraños, etc).

El aviso de error se muestra al pulsar el botón de Engage AP y su aspecto es el que se muestra en la Figura 5.9.

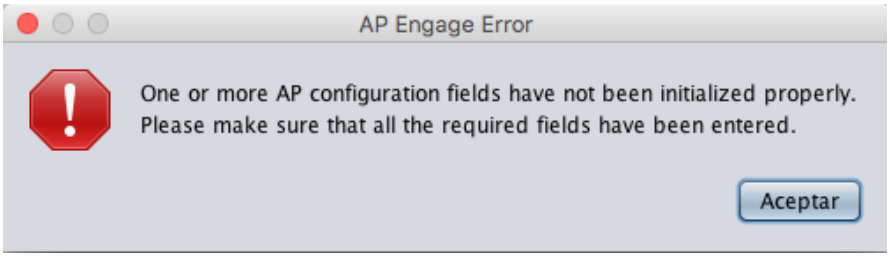


Figura 5.9: Ventana de error en la escritura de datos de configuración del autopiloto

4.4.2. Errores en el formato del archivo de configuración

Esta categoría engloba todos los errores derivados de la selección de un archivo de configuración corrupto o con un formato erróneo. El aviso saltará al seleccionar el archivo en cuestión. Cuando se da este error, el menú de configuración del autopiloto no carga ningún valor en las casillas correspondientes a los datos de entrada del autopiloto.

El aviso que se muestra es el que se puede ver en la Figura 5.10.

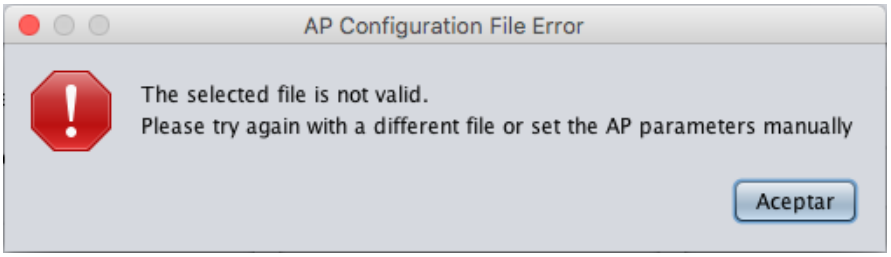


Figura 5.10: Ventana de error en el formato de archivo de configuración del autopiloto

Capítulo 6

Resultados

1. Procedimiento de análisis de los resultados

En un proyecto de desarrollo de software como el que ocupa el presente trabajo, los resultados obtenidos se consideran satisfactorios en mayor o menor medida según el cumplimiento de los requisitos. Esta es una forma objetiva de analizar si el software cumple con los objetivos marcados en las fases iniciales del desarrollo.

De esta forma, se pretende analizar qué requisitos se han cumplido de forma total, cuáles se han cumplido parcialmente y cuáles no se han podido cumplir. Para demostrar el cumplimiento de los requisitos, se propondrán una serie de casos de prueba (*Test Cases*), los cuales se trazarán a cada uno de los requisitos correspondientes.

Una vez realizado en análisis de cumplimiento de requisitos y trazados los mismos a los *Test Case* correspondientes, se realizará un resumen de los resultados, analizando de forma general el grado de cumplimiento del proyecto y sus áreas susceptibles a mejoras.

2. Diseño de los casos de prueba o *Test Cases*

El diseño de los casos de prueba es una parte fundamental del desarrollo de un diseño. Se debe de diseñar una serie de *Test Case* que sea capaz de demostrar de forma objetiva el cumplimiento de los requisitos del sistema. Si bien esto puede parecer simple, es en realidad una de las tareas más complejas del proceso de desarrollo. Un buen programa de Test debe de tener en cuenta una gran cantidad de casos posibles en los que podrían dejar de cumplirse ciertos requisitos. Además, es importante realizar una planificación óptima de la campaña de Tests, de forma que se minimicen los recursos necesarios para acometerla, así como los test a realizar. Para más información sobre qué son los casos de prueba y cómo se deben realizar de forma correcta, se insta al lector a consultar las referencias [38, 55]

La realización del diseño y planificación exhaustivos de los casos de prueba es algo que escapa al objetivo de este trabajo, por lo que se dejará como tarea a futuro o como algo a realizar en futuros trabajos fin de Máster. Sin embargo, para poder realizar un análisis completo de los resultados en este proyecto, se ha decidido preparar una serie de *Test Cases* simples que engloben todo lo necesario para verificar el cumplimiento de los requisitos. Debido a que el software implementado en este proyecto no va a ser ejecutado en una aeronave (no se trata de software embarcado), no es necesario realizar un análisis tan exhaustivo del cumplimiento de requisitos y se considera aceptable la decisión tomada.

Es muy poco probable que se pretenda implementar la totalidad del código realizado en este trabajo de forma embarcada en una aeronave en algún momento. Sin embargo, es mucho más probable que se lleguen a implementar ciertos módulos del mismo (TCAS, Autopiloto, etc). En ese caso, se debería de realizar un ejercicio de definir de forma mucho más metódica todos los *Test Case* necesarios para validar los requerimientos del software, ya que es una parte crítica del proceso de certificación del software.

Una vez aclarados los conceptos necesarios relativos al diseño de los casos de prueba, se procede a enumerar los *Test Case* que se han definido para la validación de los requisitos del sistema.

2.1. Definición del entorno de pruebas

Antes de la definición de los casos de prueba, es fundamental definir el entorno de pruebas en el que se validarán los requisitos. En el caso concreto que ocupa a este proyecto, dicho entorno no es especialmente complejo, sin embargo, es conveniente especificar el entorno de pruebas sobre el que se va a trabajar.

2.1.1. Pre-requisitos de Hardware

El hardware esencial para las pruebas a realizar se enumera a continuación:

- 1 Ordenador[1] con los siguientes requisitos mínimos de hardware[2]
  - Procesador de doble núcleo a 2.5 + GHz
  - 4 GB de memoria RAM

[1] Se podrá utilizar un ordenador siempre que éste disponga de la posibilidad de ejecutar al menos dos sistemas operativos distintos. En caso contrario se deberá utilizar un mínimo de 2 ordenadores distintos.

[2] Estos son los requisitos mínimos que requiere el simulador de vuelo X-Plane para su correcto funcionamiento: `http://www.x-plane.com/es/escritorio/requerimientos-del-sistema/`

- Una tarjeta de vídeo con 1 GB de memoria de video (VRAM)

- 1 Ordenador con recursos más limitados que los recursos mínimos recomendados para las pruebas de optimización del software.
- Se necesitará una conexión a Internet con acceso a la VPN de la UPV para las pruebas de la antena.

2.1.2. Pre-requisitos de Software

El software necesario para poder realizar la campaña de test se enumera a continuación:

- 1 licencia de sistema operativo Windows XP o superior[3].
- 1 licencia de sistema operativo Mac OSX Snow Leopard o superior[4].
- 2 licencias del Simulador de vuelo X-Plane 10, una para cada sistema operativo (el modo Demo que proporcionan de forma gratuita en su web es suficiente).
- 2 licencias del entorno de desarrollo de software NetBeans, una para cada sistema operativo.
- 1 copia del Software desarrollado en este proyecto.

Cabe destacar que, por simplicidad, se utilizarán únicamente dos sistemas operativos en las pruebas (Windows y MAC OSX). Se considera suficiente probar la aplicación en estos dos entornos para justificar el cumplimiento del requisito que solicita que el sistema sea capaz de ser ejecutado en distintos sistemas operativos.

2.2. Casos de prueba implementados

En este apartado se presentarán los casos de prueba implementados, para ello, se ha optado por seguir un formato típico en estos casos, muy similar al utilizado en compañías del sector aeronáutico como Airbus y Airbus Defence & Space.

[3]En los tests realizados se ha utilizado Windows 7.
[4]En los tests realizados se ha utilizado Mac OSX El Capitan.

TC-1. Interfaz, instrumentos y simulador

Entorno de test	Ver Sección 2.1	
Requisitos cubiertos	FR-2, FR-3, FR-4, FR-5, FR-6, FR-15	
Pre-requisitos	▪ El ordenador debe de estar encendido y con una sesión de usuario iniciada. ▪ El simulador debe de estar iniciado, con la aeronave Cessna 172S en la pista 12 del aeropuerto de Valencia (LEVC).	
Paso	**Acción a realizar**	**Respuesta esperada**
1	Abrir la aplicación Java mediante Netbeans o mediante el ejecutable *.jar.	La aplicación se abre correctamente y toma los datos del simulador.
2	Comprobar que los instrumentos descritos en FR-3 se encuentran en la interfaz gráfica.	Los instrumentos se encuentran en la interfaz.
3	En X-Plane, seleccionar en el menú superior: Aeronave > Aeronave y situaciones > Vuelo de formación	La aeronave aparece en pleno vuelo en X-Plane.
4	Comprobar que la lectura de los instrumentos de la aplicación Java se corresponde con la de los instrumentos de X-Plane.	Los instrumentos muestran la misma información en las dos aplicaciones.
5	Utilizar los mandos de vuelo y la palanca de gases de la interfaz Java para controlar la aeronave.	Las acciones de control son transmitidas de forma correcta al simulador.
Comentarios:		

TC-2. Ejecución en distintas plataformas

Entorno de test	Ver Sección 2.1	
Requisitos cubiertos	FR-1, FR-2	
Pre-requisitos	▪ Se ha de tener disponibilidad de equipos capaces de ejecutar Windows y Mac OSX.	
Paso	**Acción a realizar**	**Respuesta esperada**
1	Arrancar el PC con una sesión de Windows.	El PC arranca y se inicia la sesión.
2	Abrir el simulador X-Plane, seleccionar la aeronave Cessna 172SP y el aeropuerto de Valencia (LEVC).	Se inicia el simulador y la aeronave aparece preparada en la cabecera de pista.
3	Abrir la aplicación Java mediante Netbeans o mediante el ejecutable *.jar.	La aplicación se abre correctamente y toma los datos del simulador.
4	Cerrar los programas y la sesión en Windows.	El sistema se cierra correctamente.
5	Arrancar el ordenador con Mac OSX.	El PC arranca y se inicia la sesión.
6	Abrir el simulador X-Plane, seleccionar la aeronave Cessna 172 y el aeropuerto de Valencia (LEVC).	Se inicia el simulador y la aeronave aparece preparada en la cabecera de pista.
7	Abrir la aplicación Java mediante Netbeans o mediante el ejecutable *.jar.	La aplicación se abre correctamente y toma los datos del simulador.
Comentarios:		

TC-3. TCAS

Entorno de test	Ver Sección 2.1	
Requisitos cubiertos	FR-2, FR-7, FR-8, FR-9	
Pre-requisitos	▪ El ordenador debe de estar encendido y con una sesión de usuario iniciada. ▪ El simulador debe de estar iniciado, con la aeronave Cessna 172SP en la pista 12 del aeropuerto de Valencia (LEVC).	
Paso	**Acción a realizar**	**Respuesta esperada**
1	Abrir la aplicación Java mediante Netbeans o mediante el ejecutable *.jar.	La aplicación se abre correctamente y toma los datos del simulador.
2	Conectar la aplicación Java a la antena de tráfico aéreo.	La conexión se realiza de forma satisfactoria.
3	Comprobar que el TCAS muestra datos de otras aeronaves.	El tráfico se muestra en el TCAS.
4	Comprobar que la lógica del TCAS es coherente con el manual [50].	El TCAS es coherente con el manual [50].
5	Comprobar que el TCAS muestra los datos de forma acorde al tráfico real (*).	Los datos son representativos del tráfico real.
Comentarios: **(*)**: Para comprobar la validez de los datos de tráfico real se recomienda comparar los datos mostrados en la pantalla del TCAS con los datos proporcionados por la web FlightRadar24: `https://www.flightradar24.com/40.42,-3.68/7`		

TC-4. Autopiloto

Entorno de test	Ver Sección 2.1	
Requisitos cubiertos	FR-2, FR-10, FR-11, FR-12, FR-13, FR-14	
Pre-requisitos	▪ El ordenador debe de estar encendido y con una sesión de usuario iniciada. ▪ El simulador debe de estar iniciado, con la aeronave Cessna 172SP en la pista 12 del aeropuerto de Valencia (LEVC).	
Paso	**Acción a realizar**	**Respuesta esperada**
1	Abrir la aplicación Java mediante Netbeans o mediante el ejecutable *.jar.	La aplicación se abre correctamente y toma los datos del simulador.
2	Abrir el menú del autopiloto en Setup » Set Autopilot.	Se abre el menú de configuración del autopiloto.
3	Rellenar los datos de configuración del autopiloto según la Figura 6.1.	Los datos se quedan rellenados en el menú de configuración.
4	Pulsar el botón AP Engage.	El autopiloto se activa y la aeronave empieza la carrera de despegue en X-Plane.
5	Comprobar que la aeronave despega y sigue el procedimiento marcado (*).	Los datos son representativos del tráfico real.
Comentarios: **(*)**: Para comprobar que el procedimiento seguido por la aeronave se corresponde al de la salida SID, se recomienda monitorizar los instrumentos durante el procedimiento. Además, puede ser de utilidad utilizar el menú Ubicación » Mapa Local para comparar la trayectoria seguida con la mostrada en las cartas de navegación.		

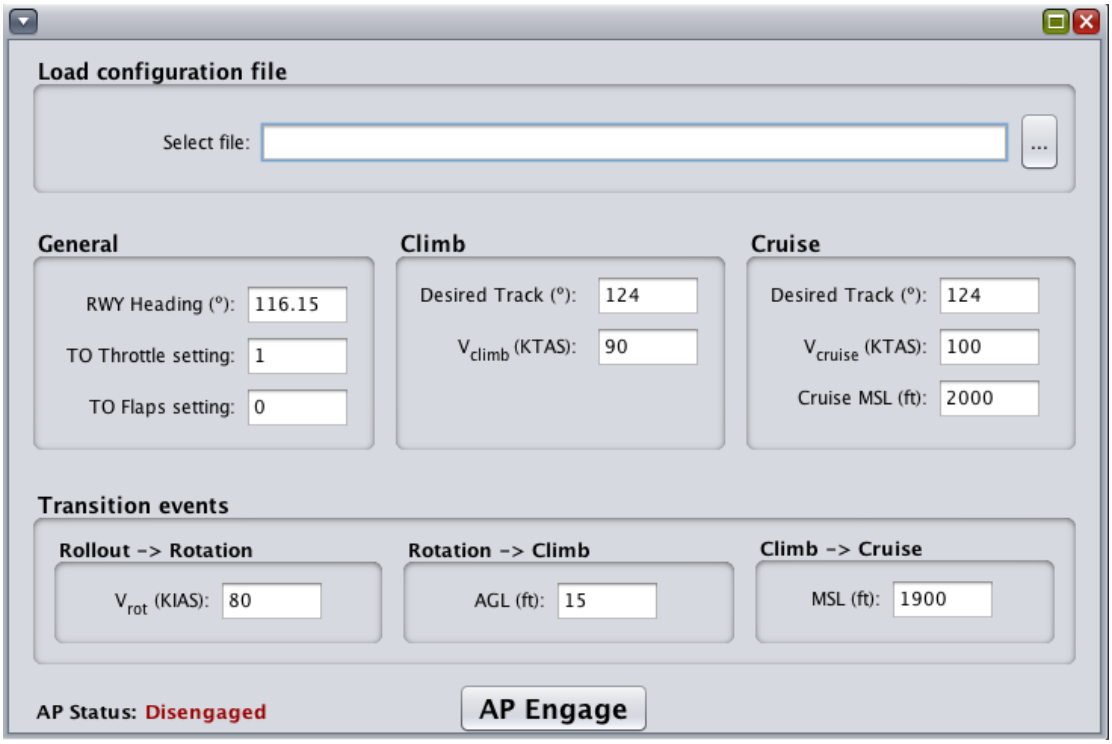


Figura 6.1: Datos de configuración del autopiloto para el Test TC-4

TC-5. Archivos de configuración del autopiloto

Entorno de test	Ver Sección 2.1	
Requisitos cubiertos	NTH-5	
Pre-requisitos	▪ El ordenador debe de estar encendido y con una sesión de usuario iniciada. ▪ El simulador debe de estar iniciado, con la aeronave Cessna 172SP en la pista 12 del aeropuerto de Valencia (LEVC).	
Paso	**Acción a realizar**	**Respuesta esperada**
1	Abrir la aplicación Java mediante Netbeans o mediante el ejecutable *.jar.	La aplicación se abre correctamente y toma los datos del simulador.
2	Abrir el menú del autopiloto en Setup > Set Autopilot.	Se abre el menú de configuración del autopiloto.
3	Seleccionar el archivo de configuración `SID_12_LEVC.apconf`(*)	Los datos se quedan rellenados en el menú de configuración.
4	Cerrar el menú del autopiloto.	Se cierra el menú.
5	Abrir el menú de la antena en Setup > Set Antenna	Los datos son representativos del tráfico real.
Comentarios: **(*)**: El archivo `SID_12_LEVC.apconf` se encuentra en la carpeta *APConfig* del directorio raíz de la aplicación.		

TC-6. Archivos de trazas de tráfico aéreo

Entorno de test	Ver Sección 2.1	
Requisitos cubiertos	NTH-6	
Pre-requisitos	▪ El ordenador debe de estar encendido y con una sesión de usuario iniciada. ▪ El simulador debe de estar iniciado, con la aeronave Cessna 172SP en la pista 12 del aeropuerto de Valencia (LEVC).	
Paso	**Acción a realizar**	**Respuesta esperada**
1	Abrir la aplicación Java mediante Netbeans o mediante el ejecutable *.jar.	La aplicación se abre correctamente y toma los datos del simulador.
2	Abrir el menú del autopiloto en Setup » Select Antenna.	Se abre el menú de configuración de la antena.
3	Seleccionar la opción: *Emulate antenna from track file*	Se habilita el selector de archivos.
4	Seleccionar el archivo de trazas `29-01-15_05.44.ssrtrk`(*)	El archivo se carga satisfactoriamente.
5	Pulsar el botón Start Antenna	La antena empieza a funcionar.
6	Comprobar que el TCAS empieza a mostrar el tráfico del archivo de trazas.	El tráfico se muestra correctamente en el TCAS.
Comentarios: **(*)**: El archivo `29-01-15_05.44.ssrtrk` se encuentra en la carpeta *SSRTracks* del directorio raíz de la aplicación.		

TC-7. Ordenador con recursos limitados

Entorno de test	Ver Sección 2.1	
Requisitos cubiertos	NTH-1	
Pre-requisitos	▪ Se debe de utilizar un ordenador con recursos limitados, menores de los requisitos mínimos de X-Plane. ▪ El ordenador debe de estar encendido y con una sesión de usuario iniciada. ▪ El simulador debe de estar iniciado, con la aeronave Cessna 172SP en la pista 12 del aeropuerto de Valencia (LEVC).	
Paso	**Acción a realizar**	**Respuesta esperada**
1	Abrir la aplicación Java.	La aplicación se abre correctamente y se conecta con el simulador.
2	Abrir el menú del autopiloto en Setup > Select Antenna.	Se abre el menú de configuración de la antena.
3	Conectar la aplicación Java a la antena de tráfico aéreo.	La conexión se realiza de forma satisfactoria.
4	Abrir el menú del autopiloto en Setup > Set Autopilot.	Se abre el menú de configuración.
5	Seleccionar el archivo de configuración `SID_12_LEVC.apconf`(*)	Los datos se quedan rellenados en el menú.
6	Pulsar el botón AP Engage.	El autopiloto se activa.
7	Comprobar que el sistema funciona de forma aceptable.	El sistema funciona correctamente.
Comentarios: **(*)**: El archivo `SID_12_LEVC.apconf` se encuentra en la carpeta *APConfig* del directorio raíz de la aplicación.		

TC-8. Arquitectura

Entorno de test	Ver Sección 2.1	
Requisitos cubiertos	NFR-2, NFR-3, NFR-4	
Pre-requisitos	▪ El ordenador debe de estar encendido y con una sesión de usuario iniciada.	
Paso	**Acción a realizar**	**Respuesta esperada**
1	Abrir el código de la aplicación Java mediante Netbeans.	Netbeans muestra el código.
2	Comprobar que el código se ha realizado siguiendo la filosofía de programación orientada a objetos.	El código está orientado a objetos.
3	Verificar que el existen distintos módulos en el código fuente como: *Antenna* o *Autopilot*	El código implementa distintos módulos.
Comentarios: El hecho de implementar el código de con la filosofía de programación orientada a objetos y de forma modular facilita enormemente la ampliación y mejora de funcionalidades. Por tanto si el resultado de este test es satisfactorio, se valida el requisito NFR-4.		

TC-9. Usabilidad

Entorno de test	Ver Sección 2.1	
Requisitos cubiertos	NFR-1, NFR-5, NFR-6	
Pre-requisitos	▪ El ordenador debe de estar encendido y con una sesión de usuario iniciada. ▪ El simulador debe de estar iniciado, con la aeronave Cessna 172SP en la pista 12 del aeropuerto de Valencia (LEVC). ▪ Un grupo representativo de personas debe estar disponible para evaluar el programa.	
Paso	**Acción a realizar**	**Respuesta esperada**
1	Realizar distintas pruebas de usabilidad con un grupo de gente representativo	El grupo utiliza la aplicación y la evalúa en términos de facilidad de uso, utilidad didáctica y capacidad de mejora.
2	Analizar las evaluaciones realizadas por los usuarios y verificar el cumplimiento de los requisitos.	Las evaluaciones verifican los requisitos.
Comentarios:		

3. Resultados de los casos de prueba

Los test descritos en los casos de prueba del Apartado 2.2 se han realizado siguiendo el procedimiento indicado en los mismos. La Tabla 6.1 resume los resultados obtenidos en cada uno de ellos. En este caso se puede observar que el resultado global de la ejecución de los casos de prueba es muy satisfactorio, con apenas dos tests que se han pasado de forma parcial debido a ciertos matices.

Tabla 6.1: Resultado de la ejecución de los casos de prueba

Caso de prueba	**Resultado del test**
TC-1	**Pasado**
TC-2	**Pasado**
TC-3	**Pasado**
TC-4	**Pasado**
TC-5	**Pasado**
TC-6	**Pasado**
TC-7	**Pasado parcialmente** (*)
TC-8	**Pasado**
TC-9	**Pasado parcialmente** (**)

Comentarios:

(*): El test se considera como pasado parcialmente ya que el sistema se ejecuta de forma satisfactoria únicamente para trabajos de desarrollo. Sin embargo, la experiencia de usuario mejorable.

(**): El test se ha pasado de forma parcial debido a que el número de usuarios que lo han realizado no es suficientemente significativo. En un futuro se deberán realizar tests más exhaustivos para verificar los requisitos de forma definitiva.

4. Trazabilidad de los requisitos

Para demostrar el cumplimiento de los requisitos, se procede a trazar los mismos con los casos de prueba que se han utilizado para validarlos mediante el uso de la matriz de trazabilidad mostrada en la Tabla 6.2. Esta es una forma de proveer al lector de una visión general del procedimiento de test. Así se puede ver de forma visual cuáles son los requisitos validados y cuáles son los que no se han podido cumplir.

Tabla 6.2: Matriz de trazabilidad de los requisitos

	FR-1	FR-2	FR-3	FR-4	FR-5	FR-6	FR-7	FR-8	FR-9	FR-10	FR-11	FR-12	FR-13	FR-14	FR-15	NFR-1	NFR-2	NFR-3	NFR-4	NFR-5	NFR-6	NTH-1	NTH-2	NTH-3	NTH-4	NTH-5	NTH-6	NTH-7	NTH-8
TC-1		X	X	X	X	X									X														
TC-2	X	X																											
TC-3		X					X	X	X																				
TC-4		X								X	X	X	X	X															
TC-5																										X			
TC-6																											X		
TC-7																						X							
TC-8																	X	X	X										
TC-9																X				X	X								

5. Análisis de los resultados

Tal y como se ha podido comprobar mediante los casos de prueba y la matriz de trazabilidad de la Tabla 6.2, los resultados obtenidos en el presente proyecto son muy satisfactorios en cuanto al cumplimiento de requisitos.

Todos los requisitos funcionales presentados en la Sección 4 del Capítulo 2 se han cumplido de forma inequívoca. Esto se ha podido demostrar de forma objetiva mediante la realización de los casos de prueba expuestos anteriormente.

En cuanto a los requisitos no funcionales, éstos se han cumplido también en su totalidad. Sin embargo, debido a la dificultad de diseñar pruebas objetivas que validen requisitos muy subjetivos como NFR-1, NFR-5 y NFR-6. Estos tres requisitos se han validado mediante el análisis de las opiniones de distintos usuarios a los que se ha mostrado la aplicación y que se les ha permitido interactuar con la misma. Sin embargo, debido a los recursos limitados de los que se dispone en este proyecto, no ha sido posible realizar un análisis sobre una población mayor. A pesar de ello, se consideran validados los requisitos, ya que si bien es difícil validarlos de forma totalmente objetiva, parece claro que la aplicación cumple sobradamente dichos requisitos.

Además del cumplimiento total de los requisitos de carácter obligatorio, se ha conseguido también cumplir algunos de los objetivos no obligatorios, denominados como funcionalidades deseables o "*nice to have*". Entre los mismos destacan la capacidad de utilizar archivos de configuración para el autopiloto y de archivos de trazas de tráfico aéreo para emular la antena en casos en los que no se disponga de conexión.

Sin embargo, es conveniente destacar la gran capacidad de crecimiento que tiene la plataforma implementada. Esto se hace patente en las funcionalidades deseables que han quedado por implementar (NTH-2, NTH-3, NTH-5, NTH-7 y NTH-8). En las secciones siguientes se profundizará en los trabajos futuros y los principales aspectos a mejorar del trabajo realizado.

Capítulo 7

Conclusiones y trabajos futuros

1. Conclusiones

El presente proyecto ha tratado el proceso de desarrollo de software aeronáutico mediante una filosofía orientada a objetos. La plataforma desarrollada se ha centrado en la vertiente más académica, con el principal objetivo de servir como plataforma educativa para el aprendizaje de programación orientada a objetos en aplicaciones aeronáuticas. Con este fin se ha desarrollado una plataforma integral que engloba una interfaz gráfica conectada a distintos módulos como son un simulador, un autopiloto, un TCAS con datos de tráfico real y distintos instrumentos de navegación.

El código desarrollado se ha realizado teniendo en mente la necesidad de que sea modular y fácilmente ampliable, con el fin de permitir que los alumnos sean capaces de editarlo en un futuro.

Para dotar al proyecto de una mayor entidad, se ha realizado siguiendo una metodología de gestión de proyectos típica en el desarrollo de software, basada en la ingeniería de requisitos. De esta forma, el proyecto se ha centrado primero en la definición de los requisitos antes de realizar la implementación. Una vez implementado el código, se ha procedido a la fase de verificación, donde se han generado distintos casos de prueba y se han validado los requisitos. Finalmente se ha realizado una matriz de trazabilidad que relacione cada requisito con el caso de prueba que lo valida.

Con esto, se han podido extraer las siguientes conclusiones:

Diseño del sistema

Se ha realizado un diseño del sistema siguiendo la metodología empleada por grandes empresas del sector de la aviación como Airbus Group. Para ello se han enumerado una serie de requisitos funcionales, no funcionales y características deseables.

Esto consiste en un ejercicio representativo del desarrollo real de software de uso aeronáutico, donde la definición de requisitos es una parte fundamental del proceso de diseño (especialmente cuando existen interacciones entre departamentos o proveedores).

Además, se ha intentado seguir la metodología de diseño de software definida por el estándar DO-178C [40], asumiendo en este caso que el software implementado se clasifica como DAL E (*No Safety Effect*) por el hecho de no tratarse de software embarcado.

Implementación de la interfaz gráfica

La implementación de la interfaz gráfica en Java™ ha permitido agrupar todos los instrumentos y funcionalidades del proyecto en una interfaz amigable y fácil de usar.

La interfaz desarrollada cumple con los requerimientos planteados en el trabajo de forma holgada. Sin embargo, como se indica en la sección de resultados, la validación del caso de prueba correspondiente a la usabilidad TC-9 no es lo suficientemente objetiva.Es por esto que se deberán realizar pruebas adicionales para validar de forma más objetiva el cumplimiento de los requisitos de usabilidad de la interfaz.

Instrumentos estándar (*6-pack*)

Los instrumentos estándar implementados se comportan de forma satisfactoria. El tamaño elegido para la realización de los mismos permite una fácil lectura por parte del usuario. Las pruebas realizadas durante el proceso de validación confirman que los instrumentos cumplen con los requerimientos impuestos en el proyecto.

Los instrumentos implementados pueden suponer una muy buena base para dar como ejercicio al alumnado. Al permitir que los alumnos editen, añadan o mejoren el código implementado se puede conseguir que se familiaricen con la programación orientada a objetos y con el funcionamiento de los principales instrumentos de navegación al mismo tiempo.

TCAS

El TCAS es sin duda una de las funcionalidades más completas y novedosas de las implementadas en este proyecto. Se han cumplido con creces los objetivos planteados para el mismo, consiguiendo un producto completo y capaz de emular de forma completa el comportamiento de un TCAS de tipo I real.

La implementación del TCAS ha conllevado el desarrollo de funcionalidades muy interesantes tales como la interconexión con la antena de tráfico aéreo o la

implementación de algoritmos de evaluación del riesgo de colisión tales como el *Closest Point of Approach*.

Simulador de vuelo y sus interfaces

Después de realizar un estudio sobre los motores de simulación disponibles realizado en el Capítulo 4, el simulador de vuelo elegido en este trabajo ha sido X-Plane. Como se ha visto en dicho capítulo, X-Plane se ha considerado la mejor opción a corto plazo debido al reducido desarrollo necesario para la implementación de la interfaz. Sin embargo, la opción con mayores ventajas a largo plazo es la del uso del motor de simulación JSBSim, que carece de interfaz gráfica y necesita muchos menos recursos computacionales. La implementacion de la interfaz con JSBSim se dejará por tanto como una acción a realizar en trabajos futuros.

Antena de tráfico aéreo

La conexión con la antena de tráfico aéreo permite alimentar al TCAS de datos de tráfico real, lo que supone un valor añadido considerable al desarrollo realizado. La posibilidad de utilizar archivos de trazas de tráfico aéreo dota al sistema de una flexibilidad mucho mayor en su uso, ya que permite utilizar el sistema en entornos sin acceso a Internet.

Autopiloto

El autopiloto implementado es capaz de realizar un procedimiento de salida estándar SID completo, desde la carrera de despegue de la aeronave hasta la entrada en la fase de crucero.

Este autopiloto se ha implementado de forma satisfactoria para la operación de una aeronave Cessna 172 Skyhawk. Por motivos restricciones temporales, se deja como tarea para trabajos futuros ampliar el catálogo de aeronaves compatibles con el autopiloto.

El comportamiento del autopiloto es el esperado y cumple con los requisitos planteados en el presente trabajo, obteniéndose una respuesta controlada y suave en las distintas fases de vuelo.

Las funcionalidades del autopiloto se ajustan a lo especificado en los requerimientos del mismo, sin embargo, se considera interesante ampliar sus funcionalidades para permitir su uso durante todas las fases del vuelo.

Resultados y verificación

El el Capítulo 6 se ha realizado un estudio del cumplimiento de los requisitos basado en la filosofía de verificación de ingeniería de requisitos. La utilización de dicho procedimiento ha permitido validar de forma individual todos los requisitos planteados mediante un método objetivo. Dicho método consiste en el diseño y la realización de diversos casos de prueba. Cada uno de los casos de prueba ha sido diseñado con el fin de validar uno o varios requisitos del sistema de la forma más objetiva posible.

El uso de los casos de prueba ha permitido obtener una serie de evidencias que permiten demostrar el cumplimiento de los requisitos. Estas evidencias son fundamentales en caso de querer certificar una o diversas partes del código para su uso en aplicaciones aeronáuticas.

Los requisitos obligatorios planteados en el presente trabajo se han cumplido en su totalidad, por lo que el resultado obtenido a nivel global en el trabajo se puede considerar como satisfactorio. Adicionalmente se han cubierto requisitos opcionales (caracterisiticas deseables o "*nice to have*"), lo que supone una mejora sobre el sistema planteado inicialmente.

2. Trabajos futuros

Si bien el resultado del proyecto ha sido satisfactorio, es evidente que hay ciertas características del sistema que tienen un importante margen de mejora. Debido a la amplitud del proyecto realizado, desde el inicio se ha tenido en cuenta que algunos de los módulos que componen al sistema pueden ser mejorados en un futuro. Cabe destacar que debido al carácter didáctico que se le pretende dar a este proyecto, la existencia de múltiples trabajos futuros da más razón de ser al proyecto en sí. Una gran parte de estos trabajos futuros se podrían realizar bien como parte de otro proyecto o bien como trabajos finales de la asignatura "Sistemas de Gestión de Vuelo por Computador", impartida en el segundo curso del Máster de Ingeniería Aeronáutica de la UPV.

A continuación se procede a enumerar las distintas áreas sobre las que se podría realizar trabajos futuros, detallando los mismos.

Implementación de la interfaz gráfica

La interfaz gráfica desarrollada en este trabajo es muy completa teniendo en cuenta los objetivos iniciales, sin embargo, existe un amplio margen de mejora en algunos aspectos. Los principales trabajos futuros propuestos son los siguientes:

- **Mejora del fondo:** En la solución implementada en este proyecto se ha

utilizado un fondo compuesto por una fotografía panorámica de 360º que da una sensación de movimiento muy aceptable. El problema que tiene esta solución es el sistema no es capaz de simular de forma realista vuelos con ascensos o descensos pronunciados. Sin embargo, se conseguiría un efecto mucho más realista si se implementase un fondo compuesto por una fotografía esférica. De esta forma, sea cual sea la actitud de la aeronave, el fondo siempre se movería de forma realista. Esta mejora requeriría de un esfuerzo considerable por parte de un alumno y podría tratarse de un ejercicio sumamente didáctico para el mismo.

- **Optimización de la interfaz:** La interfaz creada puede ser sin duda mejorada en cuanto a su optimización. Para ello se propone estudiar otras formas de implementación como dejar de utilizar *JLabels* para cargar las imágenes y cargarlas como texturas mediante la librería *Graphics2D*.

Instrumentación

La instrumentación es otro área sobre la que se puede trabajar para mejorar el sistema. Entre las principales líneas de acción se encuentran las siguientes propuestas:

- **Nuevos instrumentos:** Un campo de mejora obvio es la inclusión de un mayor número de instrumentos, más allá de los 6 instrumentos estándar y el TCAS que ya se han integrado en la plataforma.
- **Modernización de los instrumentos:** Otro apartado de mejora interesante sería la modernización de los instrumentos. Los instrumentos implementados son instrumentos electromecánicos simples, propios de aeronaves no muy avanzadas. Las aeronaves actuales intentan basarse en el concepto del *glass cockpit*, basado en la utilización de pantallas multifuncionales que son capaces de mostrar información de distintos tipos de instrumento. Un ejemplo de esto sería la Cessna 172 con el kit G1000 de Garmin. Este trabajo consistiría en implementar unas pantallas multifuncionales sobre las que se mostrase la información de distintos instrumentos, siguiendo la filosofía del *glass cokpit*. Esto supondría un esfuerzo considerable en la realización de las interfaces gráficas de los instrumentos, si bien es cierto que gran parte del código utilizado en los instrumentos analógicos implementados en este proyecto podría ser reutilizado.
- **Mejora del TCAS:** El TCAS implementado es de tipo I, lo que significa que cuando hay riesgo de colisión simplemente se limita a mostrar un aviso para alertar al piloto del peligro. Una línea de trabajo futuro interesante sería sin duda la de mejorar el TCAS implementado y desarrollar la lógica detrás de un TCAS más complejo, como un TCAS de tipo II.

Motor de simulación

Como se ha comentado en diversas ocasiones a lo largo del proyecto, la plataforma de simulación elegida en este proyecto ha sido X-Plane, sin embargo, tal y como se vio en el Capítulo 4, la solución implementada distaba de ser la más óptima en cuanto a rendimiento del sistema. Por ello se propone la siguiente línea de trabajos futuros:

- **Implementación de la interfaz con JSBSim:** JSBSim es sin duda la mejor elección en cuanto a motor de simulación para este proyecto. La capacidad de ser ejecutado sin ninguna interfaz gráfica lo convierte en el simulador idóneo para la aplicación que ocupa a este trabajo. Es por esto que se propone como trabajo futuro la realización de una interfaz con dicho simulador. En caso de realizarse, la plataforma sería capaz de ser ejecutada en ordenadores con recursos mucho más limitados, ofreciendo una mejora significativa en la experiencia de usuario.

Antena de tráfico aéreo

Para el apartado de la antena de tráfico aéreo se propone la siguiente línea de trabajo futuro:

- **Implementación de interfaz con bases de datos online:** La solución implementada con la interfaz de la antena es sin duda una funcionalidad esencial para el funcionamiento del TCAS. Sin embargo, dicha interfaz está limitada por la cobertura de la antena situada en la UPV. Para eliminar dicha restricción se propone implementar una interfaz que extraiga la información de los vuelos de una forma similar a la de la web *FlightRadar24* [1]. Sin embargo, es necesario remarcar que la implementación de dicha interfaz puede ser costosa y compleja, por lo que será necesario realizar un estudio de viabilidad previo antes de embarcarse en el diseño de esta mejora.

Autopiloto

El módulo del autopiloto es posiblemente una de las partes con mayor capacidad de crecimiento de este trabajo. Algunas de las líneas de acción propuestas son las siguientes:

- **Implementación de meas fases de vuelo:** Actualmente el autopiloto soporta las fases de vuelo comprendidas entre la fase de rodadura y la fase de crucero. Como ampliación se propone dotar al mismo de capacidad para realizar una misión completa, desde el despegue en un aeropuerto al aterrizaje en el destino, siguiendo una serie de *waypoints* definidos previamente. Este sería un ejercicio sumamente interesante y que además podría ser reutilizado en multitud de aplicaciones.

[1] `https://www.flightradar24.com/40.42,-3.68/7`

- **Ampliación del número de aeronaves controlables:** El autopiloto implementado únicamente soporta la aeronave Cessna 172. Como trabajo futuro se propone el adaptar el controlador a distintas aeronaves y plataformas.
- **Ajuste de los controladores:** Los controladores implementados proporcionan una respuesta aceptable para el alcance de este proyecto, sin embargo, sería posible mejorar su ajuste para conseguir una respuesta más fina y precisa.
- **Implementación de nuevos guiados:** Los algoritmos de guiado implementados en este trabajo, si bien son eficaces, son también bastante simples. Como ejercicio de ampliación se propone implementar algoritmos de guiado más complejos como el *carrot chasing algorithm* o algoritmo de la zanahoria, el cual proporciona una respuesta mucho más precisa en el seguimiento de trayectorias.
- **Técnicas de control adaptativo:** Otro trabajo futuro planteable sería el de implementar técnicas de control adaptativo sobre la plataforma del autopiloto disponible. Cabe destacar que si bien esta sería una temática sumamente interesante, la complejidad de la misma puede ser muy grande, por lo que sería conveniente estudiar la viabilidad de la realización de este proyecto antes de iniciar su desarrollo.

Verificación

El apartado de verificación del cumplimiento de requisitos y de funcionalidades es susceptible también a mejoras y trabajos futuras. Dichas mejoras se resumen a continuación:

- **Estudio completo de verificación:** Como trabajo futuro se plantea la realización de un estudio más exhaustivo del cumplimiento de los requisitos. Este trabajo se englobaría en el área de la gestión de proyectos y sería tremendamente útil para familiarizar al autor con conceptos muy utilizados en la industria aeronáutica como la realización de ensayos de verificación de funcionalidad. Estos ensayos son fundamentales para la certificación de software embarcado, lo que remarca la utilidad a nivel didáctico que tendría realizar una campaña de ensayos extensa, bien definida y estructurada.

Bibliografía

[1] Wilson J Rugh and Jeff S Shamma. Research on gain scheduling. *Automatica*, 36(10):1401–1425, 2000.

[2] Michael Athans. Nonlinear and adaptive control. Technical report, Massachussets Institute of Technology & Nasa, 1989.

[3] Fany Mendez-Vergara, Ilse Cervantes, and Angelica Mendoza-Torres. Stability of gain scheduling control for aircraft with highly nonlinear behavior. *Mathematical Problems in Engineering*, 2014, 2014.

[4] Bryan P Rasmussen and Young Joon Chang. Stable controller interpolation and controller switching for lpv systems. *Journal of dynamic systems, measurement, and control*, 132(1):011007, 2010.

[5] Wei Guo. Gain scheduling for a passenger aircraft control system to satisfy handling qualities, 2010.

[6] Jianying Gao. Robust control design of gain-scheduled controllers for nonlinear processes, 2004.

[7] Vojtech Veselỳ and Adrian Ilka. Gain-scheduled pid controller design. *Journal of process control*, 23(8):1141–1148, 2013.

[8] KJ Astrom and Björn Wittenmark. Self-tuning controllers based on pole-zero placement. In *IEE Proceedings D (Control Theory and Applications)*, volume 127, pages 120–130. IET, 1980.

[9] Karim Nassiri-Toussi and Wei Ren. Indirect adaptive pole-placement control of mimo stochastic systems: self-tuning results. *Automatic Control, IEEE Transactions on*, 42(1):38–52, 1997.

[10] Francisco C Silva Jr and Aldayr D Araújo. Variable structure adaptive pole placement control. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 2859–2864. IEEE, 2005.

[11] M Zamurad Shah, Raza Samar, and Aamer I Bhatti. Cross-track control of uavs during circular and straight path following using sliding mode approach. In *Control, Automation and Systems (ICCAS), 2012 12th International Conference on*, pages 185–190. IEEE, 2012.

[12] Pong-in Pipatpaibul and PR Ouyang. Application of online iterative learning tracking control for quadrotor uavs. *ISRN robotics*, 2013, 2013.

[13] Oliver Purwin and Raffaello D Andrea. Performing aggressive maneuvers using iterative learning control. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 1731–1736. IEEE, 2009.

[14] Yuriy Brun, Giovanna Di Marzo Serugendo, Cristina Gacek, Holger Giese, Holger Kienle, Marin Litoiu, Hausi Müller, Mauro Pezzè, and Mary Shaw. Engineering self-adaptive systems through feedback loops. In *Software engineering for self-adaptive systems*, pages 48–70. Springer, 2009.

[15] Matthias Schreier. Modeling and adaptive control of a quadrotor. In *Mechatronics and Automation (ICMA), 2012 International Conference on*, pages 383–390. IEEE, 2012.

[16] Siddhartha Bhattacharyya, Darren Cofer, D Musliner, Joseph Mueller, and Eric Engstrom. Certification considerations for adaptive systems. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 270–279. IEEE, 2015.

[17] Daniel B Wilson, Ali H Göktogan, and Salah Sukkarieh. Guidance and navigation for uav airborne docking.

[18] Reuben Strydom, Saul Thurrowgood, Aymeric Denuelle, and Mandyam V Srinivasan. Uav guidance: A stereo-based technique for interception of stationary or moving targets. In *Towards Autonomous Robotic Systems*, pages 258–269. Springer, 2015.

[19] Jack W Langelaan, Nicholas Alley, and James Neidhoefer. Wind field estimation for small unmanned aerial vehicles. *Journal of Guidance, Control, and Dynamics*, 34(4):1016–1030, 2011.

[20] Ferit Çakıcı, Halit Ergezer, Ufuk Irmak, and M Kemal Leblebicioğlu. Coordinated guidance for multiple uavs. *Transactions of the Institute of Measurement and Control*, 2015.

[21] UK Civil Aviation Authority. Unmanned aircraft system operations in uk airspace–guidance, 2012.

[22] International Civil Aviation Organization (ICAO). *Performance based navigation (PBN) manual*, 4th edition edition, 2013.

[23] Hèctor Usach Molina. Integridad y tolerancia a fallos en sistemas de aviónica. Universitat Politècnica de València, 2015-2016.

[24] United States Department of Transportation. Federal Aviation Administration. Introduction to tcas-ii. version 7.1, February 2011.

[25] International Civil Aviation Organization (ICAO). Tcas ii version 7.1. an overview on the pilots and atc on the tcas ii 7.1 euro version. In *Seventh Meeting of the Asia Pacific Regional Aviation Safety Team (APRAST/7)*, 2015.

[26] Edward A Lester. *Benefits and incentives for ADS-B equipage in the national airspace system*. PhD thesis, Massachusetts Institute of Technology, 2007.

[27] United States Department of Transportation. Federal Aviation Administration. 14 cfr part 91 automatic dependent surveillance— broadcast (ads–b) out performance requirements to support air traffic control (atc) service; final rule, May 2010.

[28] Surveillance and Broadcast Services (SBS) Program Office. Automatic dependent surveillance - broadcast in-trail procedures (ads-b itp) operational flight evaluation. Technical report, Federal Aviation Administraton (FAA), 2015.

[29] United States Department of Transportation. Federal Aviation Administration. Instrument flying handbook, 2008.

[30] Civil Aerospace Medical Institute Melchor J. Antuñano. Human factors in cockpit automation, 2011.

[31] United States Department of Transportation. Federal Aviation Administration. Aviation maintenance technician handbook - general, 2008.

[32] Joan Cahill and Tiziana C Callari. A novel human machine interaction (hmi) design/evaluation approach supporting the advancement of improved automation concepts to enhance flight safety.

[33] Luca Damilano, Giorgio Guglieri, Fulvia Quagliotti, and Ilaria Sale. Fms for unmanned aerial systems: Hmi issues and new interface solutions. *Journal of Intelligent & Robotic Systems*, 65(1-4):27–42, 2012.

[34] Alan Hobbs and R Jay Shively. Human factor challenges of remotely piloted aircraft, 2014.

[35] FlightSafety International. Flight simulation training systems. Disponible en: `https://www.flightsafety.com/html/pdf/3739_comm%20sim_bro_pgxpg_final.pdf`.

[36] Aerosim Technologies. Aerosim training centers, 2016. Disponible en: `http://www.aerosim.com/trainingsolutions/trainingcenters.aspx`.

[37] Dongwon Jung and Panagiotis Tsiotras. Modeling and hardware-in-the-loop simulation for a small unmanned aerial vehicle. *AIAA Infotech at Aerospace, AIAA*, pages 07–2763, 2007.

[38] Widyawardana Adiprawita, Adang Suwandi Ahmad, and Jaka Semibiring. Hardware in the loop simulator in uav rapid development life cycle. *arXiv preprint arXiv:0804.3874*, 2008.

[39] Embention Sistemas Inteligentes. Hardware-in-the-loop. hil broadcast. Disponible en: http://www.embention.com/en/hil-broadcast.htm.

[40] Radio Technical Commission for Aeronautics (RTCA). Do-178c. software considerations in airborne systems and equipment certification, January 2012.

[41] TutorialsPoint. Software engineering tutorial, 2014. Disponible en: `http://www.tutorialspoint.com/software_engineering/software_engineering_tutorial.pdf`.

[42] Ángel Rodas Joan Vila. Flight Management Systems - Apuntes de la asignatura. Universitat Politècnica de València, 2015-2016.

[43] Ralph Graves. Adi. attitude directional indicator. *Avionics News*, pages 52–56, September 2005.

[44] CFI-Wiki. Flight Instructor Wiki. Gyroscopic instruments. Disponible en: `http://cfi-wiki.net/w/Gyroscopic_Instruments`.

[45] DUTCHOPS.COM Ground School Articles. Gyroscopic instruments - turn coordinator, 2009. Disponible en: `http://www.dutchops.com/Portfolio_Marcel/Articles/Instruments/Gyroscopic_Instruments/Turn_Coordinator.htm`.

[46] Rod Machado krepelka.com. Lesson 2: How airplanes turn. Disponible en: `http://krepelka.com/fsweb/lessons/student/studentlessons02.htm`.

[47] Hersch Logbook. Aircraft instruments, 2012. Disponible en: `http://herschlogbook.blogspot.com.es/2012/08/aircraft-instruments.html`.

[48] Eurocontrol. Tcas ii version 7.1. Disponible en: `http://www.eurocontrol.int/articles/tcas-ii-version-71`.

[49] Eurocontrol. Acas guide - airborne collision avoidance systems (incorporating tcas ii version 7.0 & 7.1 and introduction to acas x), December 2015. Disponible en: `https://www.eurocontrol.int/sites/default/files/content/documents/nm/safety/ACAS/safety-acas-II-guide.pdf`.

[50] Honeywell International Inc. *CAS 66A Pilot's Guide - Bendix/King® TCAS I Collision Avoidance System*, 2006.

[51] Hèctor Usach Molina. Banco de pruebas para el diseño de autpilotos. Master's thesis, Universitat Politècnica de Valencia, 2012.

[52] Luiz Fernando Abras Cantoni. *Avaliaçao do uso da linguagem pddl no planejamento de missoes para robôs aéreos.* PhD thesis, Universidade General de Minas Gerais, 2010.

[53] Eurocontrol. Transponders in aviation. *NETALERT - the Safety Nets newsletter*, (19), May 2014. Disponible en: `https://www.eurocontrol.int/sites/default/files/publication/files/NetAlert-19.pdf`.

[54] Bones Aviation Page. Sbs basestation. Disponible en: `http://woodair.net/SBS/Article/Barebones42_Socket_Data.htm`.

[55] Cem Kaner. What is a good test case. In *Software Testing Analysis & Review Conference (STAR) East*, 2003.

[56] David Allerton. *Principles of flight simulation.* John Wiley & Sons, 2009.

[57] Alessandro Astolfi, Dimitrios Karagiannis, and Romeo Ortega. *Nonlinear and adaptive control with applications.* Springer Science & Business Media, 2007.

[58] Richard PG Collinson. *Introduction to avionics systems.* Springer Science & Business Media, 2013.

[59] Guillaume JJ Ducard. *Fault-tolerant flight control and guidance systems: Practical methods for small unmanned aerial vehicles.* Springer Science & Business Media, 2009.

[60] Alexander Fradkov and Boris Andrievsky. Combined adaptive controller for uav guidance. *European Journal of Control*, 11(1):71–79, 2005.

[61] C Kaner and REBECCA L Fiedler. Black box software testing. introduction to test design. a survey of test techniques. *BBST Test Design*, 2011.

[62] Mangal Kothari. *Algorithms for Motion Planning and Target Capturing.* PhD thesis, University of Leicester, 2011.

[63] César Munoz, Anthony Narkawicz, and James Chamberlain. A tcas-ii resolution advisory detection algorithm. In *Proceedings of the AIAA Guidance Navigation, and Control Conference and Exhibit*, 2013.

[64] Cary R Spitzer and Cary Spitzer. *Digital Avionics Handbook.* CRC Press, 2000.

[65] Borja Fons Albert. Plataforma para diseño y ejecución de aplicaciones de aviónica. Universitat Politècnica de València, 2013.