Escola d'Enginyeria de Telecomunicació i Aeroespacial de Castelldefels
UNIVERSITAT POLITÈCNICA DE CATALUNYA

# TRABAJO FINAL DE GRADO

**TÍTULO DEL TFG: Desarrollo de software para planificación de vuelos VFR**

**TITULACIÓN: Grau en Enginyeria de Sistemes Aeroespacials**

**AUTOR: Noelia Carrasquilla García**

**DIRECTOR: José Antonio Castán Ponz**

**FECHA: 25 de enero del 2022**

**Título:** Desarrollo de software para planificación de vuelos VFR

**Autor:** Noelia Carrasquilla García

**Director:** José Antonio Castán Ponz

**Fecha:** 25 de enero del 2022

## Resumen

La planificación de rutas de vuelos visuales a mano supone una dedicación de tiempo y una cantidad de cálculos considerable. Por ello, con el tiempo han ido surgiendo aplicaciones o recursos web a modo de herramienta para los pilotos. Estos programas pueden hacer los cálculos de manera inmediata y proporcionan documentos útiles de cara a la navegación.

Pese a esto, sigue siendo necesario el uso de múltiples softwares para poder tener acceso a todas las funcionalidades necesarias para la fase previa al vuelo. De esta necesidad nace FlightTrack, el aplicativo de escritorio desarrollado con la finalidad de unificar el máximo de herramientas útiles para el piloto. Su código ha sido programado desde cero en lenguaje C# con Visual Studio.

En el inicio del proyecto, se han analizado los softwares actuales determinando su grado de utilidad, para así poder crear una aplicación capaz de superarlos. De esta manera, se optimizará el tiempo del piloto dando la posibilidad usar únicamente FlightTrack como herramienta para planificar el vuelo.

Para la creación del programa, ha sido necesario el estudio de los espacios aéreos en España y la normativa de vuelos visuales, así como la aplicación de los cálculos relativos a las rutas.

Se puede concluir que FlightTrack ha superado en grado de utilidad a los softwares existentes en la actualidad. El programa facilita gran cantidad de cálculos (como velocidades, tiempos, combustible, pesos, etc.) y da la opción de descargar los datos en documentos PDF automáticamente. Con su interfaz sencilla y visual basada en mapas, proporciona un manejo intuitivo y rápido.

**Title:** VFR flight planning software development

**Author:** Noelia Carrasquilla García

**Director:** José Antonio Castán Ponz

**Date:** January 25 th 2022

## Overview

The visual flight route planning by hand involves a considerable deal of time and calculation. Therefore, over time applications or web resources have emerged as a tool for pilots. These programmes can make calculations immediately and also provide useful documents for the navigation.

Despite this, it is still necessary the use of multiple software to be able to access all of the required functions for the pre-flight phase. FlightTrack stems from this need. It is a desktop application developed in order to unify the maximum number of useful tools for the pilot. Its code has been programmed from scratch in C# with Visual Studio.

In the beginning of the project, the actual software has been analysed determining its level of usefulness, in order to create an application capable of overcome them. This way, the pilot's time will be optimized giving the possibility of using only FlightTrack as a flight planning tool.

In order to create the programme, it has been necessary the study of the Spanish airspaces and the visual flight normative, so as the application of the route calculations.

It might be concluded that FlightTrack has overcome the actual existing software in the usefulness degree. The program provides great number of calculations (speeds, times, fuel, weights, etc.) and gives the option to download the data automatically in PDF documents. With its simple and visual interface based on maps, it provides an intuitive and fast use.

# ÍNDICE


**INTRODUCCIÓN ..... 1**

**CAPÍTULO 1. PRESENTACIÓN DEL PROYECTO ..... 2**

**1.1. Estado del arte ..... 2**
1.1.1. VFR Flight ..... 6
1.1.2. SkyDemon ..... 8

**1.2. Objetivo ..... 9**

**CAPÍTULO 2. PLANIFICACIÓN DE VUELO VISUAL ..... 10**

**2.1. Espacios aéreos en España ..... 10**

**2.2. Normativa vuelos VFR en territorio español ..... 12**
2.2.1. Condiciones de visibilidad y distancia de nubes ..... 12
*2.2.1.1. Condiciones meteorológicas visuales (VMC)* ..... 12
*2.2.1.2. Techo de nubes y visibilidad en aeródromos* ..... 13
2.2.2. Alturas de vuelo ..... 14
2.2.3. Velocidades ..... 14

**2.3. Cálculo de rutas VFR ..... 15**
2.3.1. Limitación de la aeronave ..... 15
2.3.2. Limitación de los aeródromos ..... 16
2.3.3. Cálculo de parámetros ..... 17
2.3.4. Meteorología: el METAR ..... 18
2.3.5. El NavLog y el guion de vuelo ..... 19

**CAPÍTULO 3. FLIGHTTRACK ..... 21**

**3.1. Creación del software ..... 21**
3.1.1. Entorno de programación ..... 21
*3.1.1.1. GMap.NET* ..... 23
3.1.2. Obtención de datos ..... 23
*3.1.2.1. Archivos cargados* ..... 23
*3.1.2.2. APIs* ..... 24
3.1.3. Estructura de la solución ..... 26
3.1.4. Características y funcionalidades ..... 28

**3.2. Manual de usuario ..... 29**
3.2.1. Main ..... 29
3.2.2. NavLog ..... 31
*3.2.2.1. Comentarios* ..... 33
*3.2.2.2. Fecha y hora* ..... 34
*3.2.2.3. Combustible* ..... 35
*3.2.2.4 Espacios Aéreos* ..... 36
3.2.3. Peso y Balance ..... 37
3.2.4. METAR ..... 38
3.2.5. Ayuda ..... 39
3.2.5. Biblioteca ..... 40

**CAPÍTULO 4. PARTIDA ECONÓMICA ..... 42**

**4.1. Sueldo del ingeniero/desarrollador ..... 42**

**4.2. Precio del hardware y las credenciales ..... 43**

**4.3. Gastos de oficina ..... 44**

**CAPÍTULO 5. CONCLUSIONES Y LÍNEAS DE CONTINUIDAD ..... 46**

**BIBLIOGRAFÍA ..... 48**

**ANEXO A. CÓDIGO DE FLIGHTTRACK ..... 50**

**A.1. Formularios ..... 50**
A.1.1. Main ..... 50
A.1.2. Inicio ..... 67
A.1.3. NavLog ..... 67
A.1.4. Espacios Aéreos ..... 79
A.1.5. Fecha y hora ..... 81
A.1.6. Desglose de combustible ..... 83
A.1.7. Comentarios ..... 84
A.1.8. Peso y balance ..... 85
A.1.9. Metar: principal ..... 90
A.1.10. Metar: detalle ..... 92
A.1.11. Metar: otro aeródromo ..... 94
A.1.12. Metar: decodificación ..... 96
A.1.13. Ayuda ..... 97
A.1.14. Biblioteca ..... 98

**A.2. Clases ..... 101**
A.2.1. AD ..... 101
A.2.2. Avion ..... 104
A.2.3. Balance ..... 111
A.2.4. EspacioAer ..... 112
A.2.5. KML ..... 116
A.2.6. Metar_ ..... 117
A.2.7. Peticiones ..... 120
A.2.8. Radioayuda ..... 121
A.2.9. RutaAer ..... 123

# ÍNDICE DE TABLAS Y FIGURAS

**Tabla 1.1. Listado de aplicativos para planificar vuelos VFR ..... 2**

**Tabla 1.2. Grado de utilidad de los programas de planificación de vuelos VFR ..... 5**

**Tabla 1.3. Ranking de utilidad de los programas de planificación de vuelos VFR ..... 6**

**Fig. 1.1. Pantalla de VFR Flight ..... 6**

**Fig. 1.2. Área de información meteorológica de meteo.pl ..... 7**

**Fig. 1.3. Pantalla de SkyDemon ..... 8**

**Fig. 2.1. División del espacio aéreo ..... 10**

**Fig. 2.2. Clases de espacio aéreo ..... 11**

**Tabla 2.1. Clasificación y descripción del espacio aéreo ..... 12**

**Tabla 2.2. Condiciones meteorológicas visuales ..... 13**

**Fig. 2.3. Altitudes de vuelo en vuelos VFR ..... 14**

**Fig. 2.4. Peso y Balance de Cessna 172N ..... 16**

**Tabla 2.3. Distancia de despegue para Cessna 172N ..... 17**

**Tabla 2.4. Rendimiento en crucero para Cessna 172N ..... 18**

**Tabla 2.5. Estructura de METAR ..... 19**

**Tabla 2.6. Estructura de NavLog ..... 20**

**Fig. 2.5. Extracto de guion de vuelo ..... 20**

**Fig. 3.1. Pantalla de carga de FlightTrack ..... 21**

**Fig. 3.2. Pantalla de Visual Studio con el formulario Main de FlightTrack ..... 22**

**Fig. 3.3. Archivo de carga de la aeronave Cessna N ..... 24**

**Fig. 3.4. Métricas de Maps Elevation API en Google Maps Platform ..... 25**

**Fig. 3.5. Proyectos de la solución FlightTrack ..... 26**

**Fig. 3.6. Pantalla principal de FlightTrack ..... 29**

**Fig. 3.7. Creación de Ruta ..... 30**

**Fig. 3.8. Pantalla NavLog ..... 31**

**Fig. 3.9. Descargable: guion de vuelo ..... 32**

**Fig. 3.10. Descargable: NavLog ..... 33**

**Fig. 3.11. Agregar comentario ..... 33**

**Fig. 3.12. Modificar fecha y hora** ..... **34**

**Fig. 3.13. NavLog después de modificar fecha y hora** ..... **35**

**Fig. 3.14. Espacios Aéreos** ..... **36**

**Fig. 3.15. Ventana de peso y balance** ..... **37**

**Fig. 3.16. Ventana principal de METAR** ..... **38**

**Fig. 3.17. METAR de aeropuerto de Seve Ballesteros-Santander** ..... **38**

**Fig. 3.18. METAR decodificado de aeropuerto de Seve Ballesteros-Santander** ..... **39**

**Fig. 3.19. Selección de METAR de cualquier aeropuerto** ..... **39**

**Fig. 3.20. Ventana de Ayuda** ..... **40**

**Fig. 3.21. Ventana de Biblioteca** ..... **40**

**Fig. 4.1. Salario anual de un desarrollador en Estados Unidos.** ..... **42**

**Fig. 4.2. Salario anual de un desarrollador en España** ..... **43**

**Tabla 4.1. Precio al mes por petición de la API Google Maps Elevation** ..... **44**

**Tabla 4.2. Gastos medios mensuales por español** ..... **44**

**Tabla 5.1. Grado de utilidad de los programas de planificación de vuelos VFR, incluyendo FlightTrack** ..... **46**

# ÍNDICE DE ACRÓNIMOS

**AGL.** Del inglés: *Above Ground Level* (“Sobre el nivel del suelo”).

**ALT.** Del inglés: *Altitude* (“Altitud”).

**API.** Del inglés: *Application Programming Interface* (“Interfaz de programación de aplicaciones”).

**ATC.** Del inglés: *Air Traffic Control* (“Control del tráfico aéreo”).

**ATS.** Del inglés: *Air Traffic Services* (“Servicios de tránsito aéreo”).

**ATZ.** Del inglés: *Aerodrome Traffic Zone* (“Zona de tránsito de aeródromo”).

**AWY.** Del inglés: *Airway* (“Aerovía”).

**CAVOK.** Del inglés: *Ceiling And Visibility OK* (“Techo y visibilidad bien”).

**CTA.** Del inglés: *Control Area* (“Área de control”).

**CTR.** Del inglés: *Control Zone* (“Zona de control”).

**DME.** Del inglés: *Distance Measuring Equipment* (“Equipo medidor de distancia”).

**ETA.** Del inglés: *Estimated Time of Arrival* (“Hora estimada de llegada”).

**ETE.** Del inglés: *Estimated Time Enroute* (“Tiempo estimado en ruta”).

**FIR.** Del inglés: *Flight Information Region* (“Región de información de vuelo”).

**FIS.** Del inglés: *Flight Information Service* (“Servicio de información de vuelo”).

**FL.** Del inglés: *Flight Level* (“Nivel de vuelo”).

**GPS.** Del inglés: *Global Positioning System* (“Sistema de posicionamiento global”).

**GS.** Del inglés: *Ground Speed* (“Velocidad en tierra”).

**HTML.** Del inglés: *HyperText Markup Language* (“Lenguaje de etiquetas de hipertexto”).

**ICAO.** Del inglés: *International Civil Aviation Organization* (“OACI: Organización de Aviación Civil Internacional”).

**IDE.** Del inglés: *Integrated Development Environment* (“Entorno de desarrollo integrado”).

**IFR.** Del inglés: *Instrumental Flight Rules* (“Reglas de vuelo instrumental”).

**ILS.** Del inglés: *Instrumental Landing System* (“Sistema de aterrizaje instrumental”).

**JSON.** Del inglés: *JavaScript Object Notation* (“Notación de objeto de JavaScript”).

**KIAS.** Del inglés: *Knots Indicated Airspeed* (“Velocidad indicada, en nudos”).

**KML.** Del inglés: *Keyhole Markup Language* (“Lenguaje de marcado de ojo de cerradura”).

**KMZ.** Del inglés: *Keyhole Markup language Zipped* (“Lenguaje de marcado de ojo de cerradura comprimido”).

**KTAS.** Del inglés: *Knots True Airspeed* (“Velocidad verdadera, en nudos”).

**LDA.** Del inglés: *Landing Distance Available* (“Distancia de aterrizaje disponible”).

**METAR.** Del francés: *Météorologique Aviation Réguilière* (“Informe meteorológico aeronáutico de rutina”).

**MLW.** Del inglés: *Maximum Landing Weight* (“Peso máximo de aterrizaje”).

**MTOW.** Del inglés: *Maximum Take-Off Weight* (“Peso máximo al aterrizaje”).

**NavLog.** Del inglés: *Navigation Log* (“Registro de navegación”).

**NDB.** Del inglés: *Non-Directional Beacon* (“Baliza no direccional”).

**NOTAM.** Del inglés: *Notice To Airmen* (“Aviso a navegantes”).

**OGC.** Del inglés: *Open Geospatial Consortium* (“Consorcio abierto geoespacial”).

**PDF.** Del inglés: *Portable Document Format* (“Formato de documento portable”).

**POH.** Del inglés: *Pilot’s Operating Handbook* (“Manual de vuelo”).

**RAM.** Del inglés: *Random Access Memory* (“Memoria de acceso aleatorio”).

**SFC.** Del inglés: *Surface* (“Superficie”).

**SO.** Sistema Operativo.

**TAF.** Del inglés: *Terminal Aerodrome Forecast* (“Pronóstico de tiempo de aeropuerto”).

**TMA.** Del inglés: *Terminal Manouvering Area* (“Área de control terminal”).

**TORA.** Del inglés: *Take-Off Run Available* (“Distancia de despegue disponible”).

**TXT.** Del inglés: *Textfile* (“Archivo de texto”).

**UIR**. Del inglés: *Upper Information Region* (“Región superior de información”).

**UNTLD**. Del inglés: *Unlimited* (“Sin límite”).

**URL**. Del inglés: *Uniform Resource Locator* (“Localizador de Recursos Uniforme”).

**Va.** Velocidad de maniobra.

**VFE.** Del inglés: *Maximum Flap Extended Speed* (“Velocidad máxima con flaps extendidos”).

**VFR.** Del inglés: *Visual Flight Rules* (“Reglas de vuelo visual”).

**VMC.** Del inglés: *Visual Meteorological Conditions* (“Condiciones meteorológicas visuales”).

**VNE.** Del inglés: *Never Exceed Velocity* (“Velocidad de nunca exceder”).

**VNO.** Máxima velocidad estructural en crucero.

**VOR.** Del inglés: *Very High Frequency Omnidirectional Range* (“Radiofaro omnidireccional de muy alta frecuencia”).

**WP.** Del inglés: *waypoint* (“Punto en ruta”).

# INTRODUCCIÓN

Cuando un piloto decide realizar un vuelo visual, primero ha de hacer una planificación del mismo. El uso de software como herramienta para planificación implica una disminución del tiempo y esfuerzo por parte del usuario. El principal inconveniente es que es necesario el uso de varios programas para poder revisar todos los aspectos que requiere el vuelo, ya que cada programa se centra en ciertas funcionalidades.

La aplicación FlightTrack se ha creado con el propósito de unificar el máximo de funcionalidades con una interfaz fácil y visual. Su finalidad es superar en utilidad a los softwares comerciales que hay en la actualidad.

El proyecto ha sido creado desde cero, usando el lenguaje C# y el IDE Microsoft Visual Studio. Se han implementado también dos APIs y el control Gmap.NET para la visualización en mapas.

La memoria está dividida en cinco capítulos. El primero es la presentación del proyecto. En ella se expone el estado del arte, donde se analizan los programas actuales que sirven para la planificación de vuelos VFR, y se clasifican según la utilidad. También se fijan los objetivos del proyecto.

En el segundo capítulo se muestran los diferentes aspectos necesarios para la planificación de vuelos visuales, tanto el funcionamiento de los espacios aéreos como las normativas, así como los cálculos y documentos de la fase previa del vuelo.

En el tercer capítulo se presenta la aplicación de FlightTrack. La primera parte hace referencia a todos los aspectos relativos a su creación, y la segunda es su manual de usuario.

El cuarto capítulo es la partida económica, donde se hace un cálculo del presupuesto para desarrollar FlightTrack.

El quinto capítulo recoge las conclusiones y propone líneas de continuidad del proyecto.

Finalmente, en el anexo se encuentra el código del programa, separado en formularios y clases.

# CAPÍTULO 1. PRESENTACIÓN DEL PROYECTO

Para el desarrollo de FlightTrack, primero se hizo un estudio sobre el estado del arte en el ámbito de los softwares de ayuda para la planificación de vuelos VFR.

Una vez analizados los programas existentes, se clasificaron y se estudiaron los más completos en detalle.

Finalmente, se establecieron unos objetivos a cumplir a la hora de crear FlightTrack. En este capítulo se abarcan estos temas.

## 1.1. Estado del arte

Actualmente hay diferentes opciones a la hora de planificar un vuelo visual.

La opción tradicional es hacer uso de mapas y cartas físicos y calcular a mano todo lo necesario para emprender el vuelo. Tanto las distancias a recorrer como los tiempos de ruta, combustible requerido, etc. Estos cálculos pueden ser tediosos y llevar al piloto algo de tiempo.

Por ello, la opción más cómoda y popular actualmente es sacar provecho de las diferentes aplicaciones que hay hoy en día para facilitar los cálculos y obtener datos más precisos o simplemente para verificar los resultados obtenidos a mano.

En total hay 14 aplicaciones con funcionalidades de ayuda para la planificación de vuelos VFR. Estas aplicaciones son las mostradas en la tabla 1.1.

**Tabla 1.1**. Listado de aplicativos para planificar vuelos VFR

| |
|---|
| Skyvector |
| FileStar VFR Planning Software |
| VFR Flight |
| Plan G |
| Flight Planner 3000 |
| OIR Flight Planner |
| Little Navmap |
| Flylog |
| Aeroplus Aviation Flight Plan App |
| ForeFlight |
| Easy VFR |
| RocketRoute |
| Skydemon |

Para valorar el grado de utilidad de estas aplicaciones, se han tenido en cuenta una lista de criterios. Estos criterios son los siguientes:

- La aplicación contiene datos de España. Dado que ésta es el área de interés, se ha verificado que haya información tanto de los terrenos como de los aeropuertos o información meteorológica de la región.

- La aplicación tiene mapas con imágenes de satélite. Algunas de las aplicaciones no contaban con una interfaz de mapas, y otras no incluían mapas con imágenes de satélite. La opción de contar con mapas satélite es muy útil en vuelos VFR ya que se basan en referencias visuales. De esta manera se puede tener una mejor idea del terreno que se verá cuando se ejecute el vuelo.

- La aplicación permite selección de WP (*waypoints*) visuales. Siguiendo con el criterio anterior, es necesaria la opción de seleccionar un punto de ruta en cualquier sitio. Este criterio no lo cumplirán las aplicaciones que solo permitan seleccionar como punto de ruta uno IFR o las que requieran introducir las coordenadas manualmente.

- La aplicación permite el cálculo de peso y centrado. Se ha verificado que se puedan introducir los pesos del piloto, copiloto, pasajeros y equipajes. La aplicación ha de ser capaz de calcular si la disposición es válida para un vuelo estable.

- La aplicación permite el cálculo de combustible. Según la ruta y el modelo de la aeronave, la aplicación ha de poder calcular el combustible requerido y verificar si excede los límites del avión.

- La aplicación permite la creación y descarga de NavLog/Flightplan. Un rellenado automático de los formularios con todos los datos necesarios es un criterio muy importante a la hora de verificar la utilidad de la aplicación.

- La aplicación ofrece datos meteorológicos en ruta. Dado que el viento y otras condiciones meteorológicas pueden alterar la ruta y sus parámetros, se ha tenido en cuenta este criterio para el análisis.

- La aplicación permite consultar TAF/METAR/NOTAM. Se ha considerado el acceso a la información meteorológica específica transmitida en estos informes como otro punto a considerar.

- La aplicación muestra el perfil vertical de la ruta. Una representación visual de la ruta ayuda al usuario a visualizar mejor el recorrido y a comprobar que se cumplen todos los requerimientos con el terreno.

- La aplicación tiene función GPS para vuelo. Ciertas aplicaciones presentaban una funcionalidad para utilizar durante el vuelo: el seguimiento de la ruta por GPS.

- La aplicación presenta una interfaz sencilla y visual. Se ha valorado la facilidad para interactuar con el programa, así como la visualización de los resultados.

- La aplicación calcula las altitudes mínimas seguras. El conocimiento del terreno y de los espacios aéreos es esencial a la hora de hacer la ruta. Por ello, es importante saber cuál es la altura mínima en la que la aeronave puede volar en cada punto del recorrido.

- La aplicación se puede usar en fase *pre-planning*. Ciertos aplicativos sólo contaban con un seguimiento a tiempo real de la ruta. Dichos programas no contaban con la opción de planificar la ruta con anterioridad para así generar los documentos pertinentes o hacer un estudio de las zonas atravesadas. Por ello, en los criterios también se ha tenido en cuenta el hecho de poder planificar con antelación el vuelo.

- La aplicación tiene un simulador de vuelo. Algunas aplicaciones tenían la funcionalidad añadida de poder simular el vuelo en el propio programa. Esto es muy útil, ya que el piloto puede tener más claro cómo se desarrollará el vuelo.

- La aplicación tiene la opción de importar a KML. La extensión de archivo .kml indica que el archivo contiene datos geográficos. Estos datos pueden ser abiertos por cualquier otro programa que sea capaz de leer KML, como por ejemplo Google Earth. El poder importar la ruta a otras aplicaciones permite que la experiencia de planificar la ruta no se acabe en el aplicativo donde se genera. Además, permitiría a pilotos compartir entre ellos sus rutas.

- La aplicación tiene un *logbook*. Un logbook es un diario. En él, se puede ver el registro de los vuelos realizados con las informaciones más relevantes (como la fecha, la duración, etc.).

- La aplicación permite planificar los procesos de despegue y aterrizaje. Estos procesos requieren de datos específicos sobre los aeropuertos. Se han de saber los datos meteorológicos, el número y orientación de las pistas, así como su longitud, etc. De esta manera se puede comprobar si una aeronave puede despegar o aterrizar en un aeródromo, ya que puede ser que la limitación del peso o los vientos lo impidan. Además, cada aeropuerto cuenta con sus procedimientos de aproximación y despegue, que condicionan la salida y la llegada del vuelo.

- La aplicación proporciona información sobre las ayudas a la navegación (*navaids*). Es importante saber qué radioayudas están en un rango cercano en cada punto del recorrido. En caso de necesitar ayuda en el guiado durante la ruta, saber la ubicación y frecuencia del navaid puede ser vital. Ejemplos de estas ayudas son el VOR, DME, ILS, NDB, etc.

Estos criterios suman un total de 18 puntos a tener en cuenta a la hora de valorar las diferentes opciones de programas que hay actualmente para planificar un vuelo VFR.

En la tabla 1.2 se muestran los resultados de la valoración de grado de utilidad de los programas de la tabla 1.1 según los criterios mencionados.

**Tabla 1.2**. Grado de utilidad de los programas de planificación de vuelos VFR

| | Skyvector | FileStar | VFR Flight | Plan G | Flight Planner 3000 | OIR Flight Planner | Little Navmap | Flylog | Aeroplus Aviation | ForeFlight | Easy VFR | RocketRoute | Skydemon |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Contiene datos de España | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Mapa satélite | | | ✓ | | | | | | | | | | |
| Selección de WP visuales | | ✓ | ✓ | | | ✓ | ✓ | | | | | | ✓ |
| Cálculo peso y centrado | | ✓ | ✓ | | ✓ | | | | | | ✓ | | ✓ |
| Cálculo combustible | | | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ |
| Creación y descarga de Navlog / Flightplan | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Datos meteorológicos en ruta | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| TAF/METAR/NOTAM | ✓ | | ✓ | ✓ | | ✓ | | | | ✓ | | | ✓ |
| Perfil vertical de ruta | | ✓ | ✓ | | | | ✓ | | | | ✓ | | ✓ |
| Función GPS para vuelo | | ✓ | | | | | | ✓ | ✓ | | ✓ | | ✓ |
| Interfaz sencilla y visual | ✓ | | | ✓ | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Altitudes mínimas seguras | | ✓ | ✓ | | | | ✓ | | | | | | ✓ |
| Pre-planning | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ |
| Simulador de vuelo | | | ✓ | ✓ | | | ✓ | | | | | | |
| Importación kml | | | ✓ | ✓ | | | | | | | | | |
| Logbook | | | | | | | ✓ | ✓ | | | | | |
| Procedimientos despegue/aterrizaje | | | | ✓ | | | ✓ | | | ✓ | | | |
| Información navaids | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | | ✓ |
| **TOTAL (%)** | **39** | **50** | **72** | **61** | **33** | **44** | **67** | **33** | **39** | **50** | **50** | **33** | **72** |

Ahora podemos clasificar las aplicaciones según su grado de utilidad conforme los criterios anteriormente descritos. Los resultados se muestran en la tabla 1.3.

**Tabla 1.3**. Ranking de utilidad de los programas de planificación de vuelos VFR

| | |
|---|---|
| VFR Flight | 72 % |
| Skydemon | 72 % |
| Little Navmap | 67 % |
| Plan G | 61 % |
| FileStar VFR Planning Software | 50 % |
| ForeFlight | 50 % |
| Easy VFR | 50 % |
| OIR Flight Planner | 44 % |
| Skyvector | 39 % |
| Aeroplus Aviation Flight Plan App | 39 % |
| Flight Planner 3000 | 33 % |
| Flylog | 33 % |
| RocketRoute | 33 % |

Con estos datos, podemos afirmar que VFR Flight y SkyDemon son las aplicaciones más completas en la actualidad, con un 72% de los criterios cumplidos. Por ello, vamos a ver de una manera más detallada cómo son estos dos programas.

### 1.1.1. VFR Flight

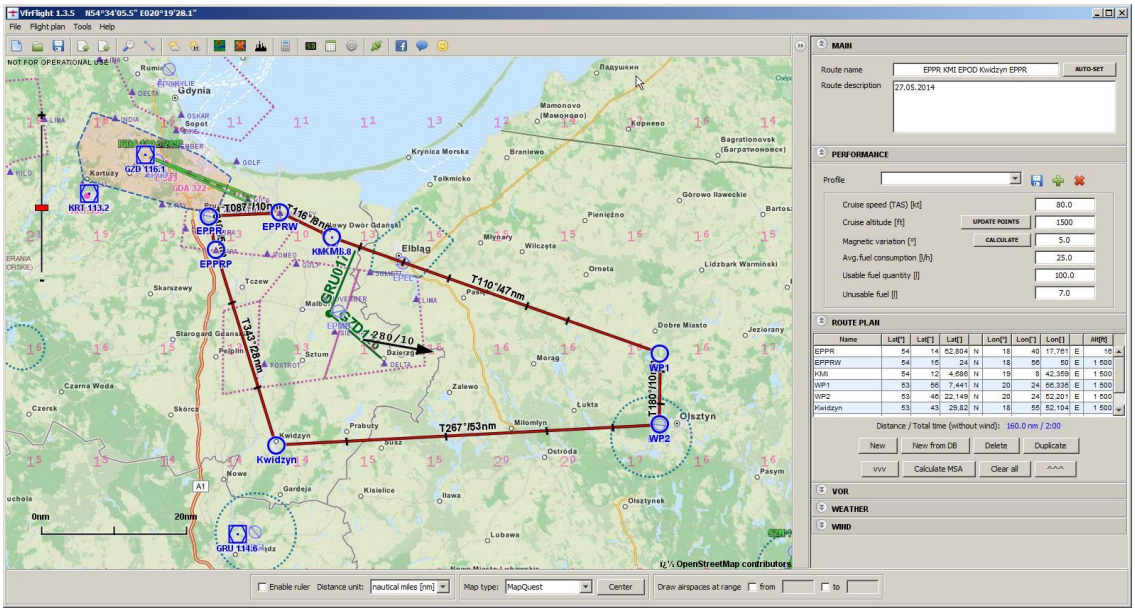



**Fig. 1.1.** Pantalla de VFR Flight. Fuente: vfrflight.org

VFR Flight es un programa gratuito descargable para ordenador. Entre sus funcionalidades destacan:

- Soporte para simuladores de vuelo
- Diferentes mapas para seleccionar
- Altitudes mínimas seguras
- Plano vertical
- TAF/METAR
- Meteorología en ruta
- Descarga de planes de vuelo
- Búsqueda de VORs próximos
- Cálculo de peso y balance
- Exportación KML
- Selección de WP visuales

Aun así, cabe destacar que para zonas de la región de España no se dispone de TAF/METAR ni meteorología en ruta, ya que se usa el servicio de meteo.pl, que únicamente se centra en los países mostrados en la figura 1.2.

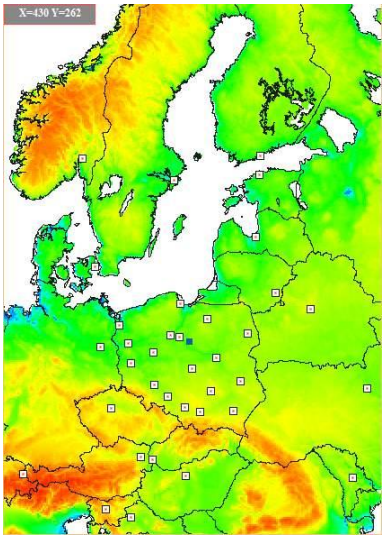



**Fig. 1.2.** Área de información meteorológica de meteo.pl. Fuente: vfrflight.org

Por ello, pese a sus muchas cualidades, también tiene varios defectos:

- No ofrece información meteorológica de España
- No hace cálculos de combustible
- No tiene una interfaz sencilla y visual
- No tiene datos para el despegue y aterrizaje
- No tiene registro de vuelos (logbook)
- No se puede usar durante la navegación, es sólo para fase pre-planning

### 1.1.2. SkyDemon

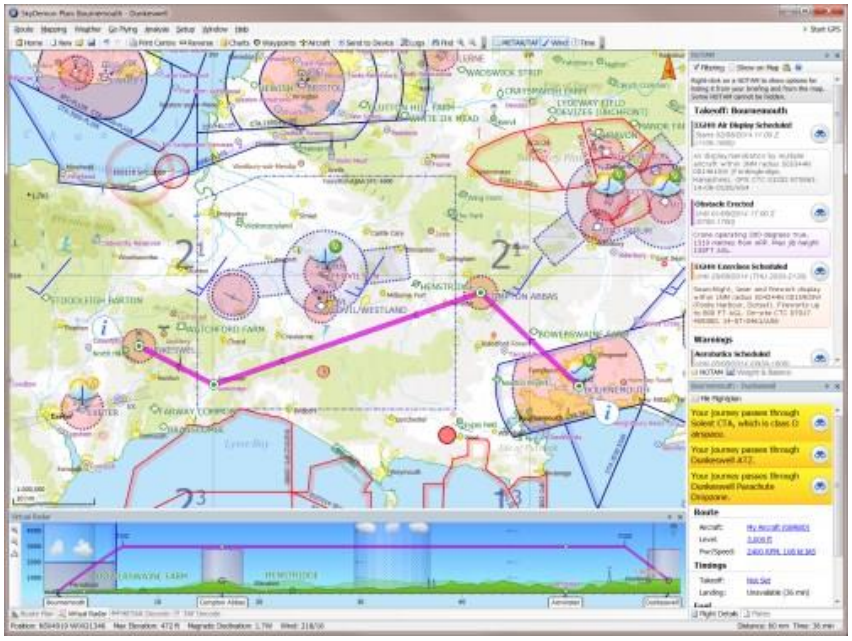


**Fig. 1.3.** Pantalla de SkyDemon. Fuente: skydemon.aero

SkyDemon es un software multiplataforma que ayuda tanto en la planificación del vuelo como en la navegación durante la ruta. Es necesaria una suscripción para acceder a sus servicios, así como un pago extra por si se requieren cartas específicas adicionales.

Entre sus funcionalidades destacan:

- Cartas dinámicas, con datos meteorológicos de Europa, Estados Unidos y Sudáfrica
- Cálculo de altitudes mínimas seguras
- Muestra radioayudas
- NOTAM, TAF y METAR
- Perfil vertical
- Datos meteorológicos en ruta
- Plan de vuelo
- Cálculo de peso y centrado
- Cálculo de combustible
- Ayuda durante la navegación, función GPS
- Selección de WP visuales
- Interfaz sencilla y visuales

Aunque es muy completa, también presenta defectos:

- No hay opción de mapas satélite

- No hay simulador de vuelo
- No se puede importar en KML
- No hay logbook

## 1.2. Objetivo

Del estudio del grado de utilidad de los programas que hay actualmente para la planificación de vuelos VFR podemos sacar como conclusión que es probable que los pilotos utilicen varios recursos a la hora de calcular una ruta.

Si un piloto quisiera, por ejemplo, un programa que llevara el registro de sus vuelos y a la vez calculara el peso y centrado de la aeronave tendría que usar dos aplicativos diferentes. Eso implica tener que trazar la ruta dos veces o importar los archivos, pero ¿y si los programas no tienen la opción de importar KML? La pérdida de tiempo innecesaria sería obligatoria.

Por ello, el objetivo de este trabajo será crear un software para ordenador que tenga mayor grado de utilidad en comparación con los programas actuales, según los criterios establecidos. De esta manera se ahorrará el tiempo de los pilotos, brindando la posibilidad de usar únicamente una aplicación a la hora de planificar un vuelo.

Dicho software se llamará Flightrack y permitirá el trazado de rutas en el territorio español a través de la selección de puntos de ruta visuales en un mapa. Las rutas propuestas seguirán las mínimas altitudes seguras y cumplirán con el reglamento de los espacios aéreos de España.

Con FlightTrack se pretende que el usuario pueda acceder a los datos útiles de manera intuitiva, fácil y rápida, así como obtener documentos en PDF de manera automática y de dar la opción de guardar rutas en caso de querer consultarlas más adelante. De esta manera el proceso previo a realizar un vuelo visual será mucho menos tedioso que comparando diferentes mapas o usando diferentes aplicativos con funcionalidades más específicas.

# CAPÍTULO 2. PLANIFICACIÓN DE VUELO VISUAL

Este capítulo se centra en la planificación del vuelo visual.

Para el desarrollo de un vuelo visual, primero hemos de entender cómo son los espacios aéreos en España y cuál es la normativa específica en esta región.

Posteriormente, podemos proceder a realizar los cálculos relativos a la ruta. Para ello hemos de tener en cuenta ciertas limitaciones y la meteorología.

Finalmente, se mostrarán un par de documentos usados en aviación que recogen los datos de la planificación del vuelo.

## 2.1. Espacios aéreos en España

Un espacio aéreo se define como la porción de atmósfera que queda por encima de un territorio terrestre (tanto si es tierra como agua). Cada país se encarga de regular sus espacios aéreos.

De esta manera, el mundo está dividido en 9 zonas llamadas FIR. España se divide, a su vez, en otras tres FIR: Barcelona, Madrid y Canarias. Una FIR se extiende desde el suelo hasta el FL 245. Una UIR comprende alturas mayores a FL 245.

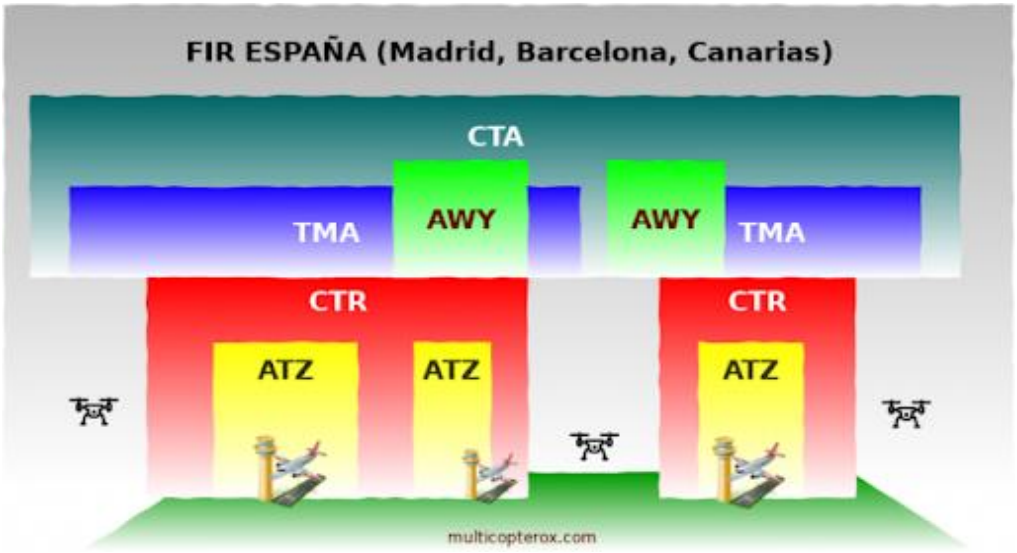



**Fig. 2.1.** División del espacio aéreo. Fuente: multicopterox.es

Los FIR contienen los siguientes tipos de espacio aéreo:

- ATZ (Zona de Tránsito de Aeródromo). Es el espacio aéreo controlado asociado a un aeródromo. Controla las inmediaciones de dicho aeropuerto. En este espacio la Torre de Control controla y protege los vuelos VFR.

- CTR (Zona de Control). Es el espacio aéreo asociado a un aeródromo que tiene como objetivo controlar las entradas y salidas IFR.

- CTA (Área de Control). Es el espacio aéreo que tiene por objetivo proteger las aeronaves hasta que entran en ruta. También están en las proximidades de los aeropuertos.

- TMA (Área Terminal de Maniobras). Es el espacio aéreo designado para áreas con mucho tráfico (donde pasen varias aerovías o haya varios aeropuertos, por ejemplo)

- AWY (Aerovía). Es el espacio aéreo controlado con forma de corredor.

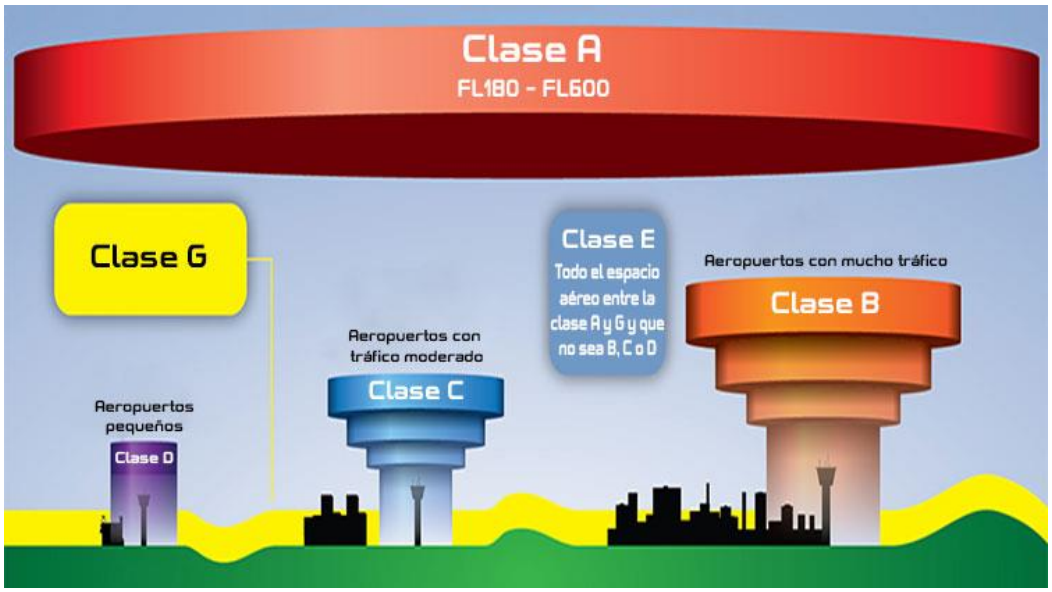



**Fig. 2.2.** Clases de espacio aéreo. Fuente: hispadrones.es

También se puede dividir el espacio aéreo en controlado y no controlado. En el espacio aéreo controlado se facilita servicio de tránsito aéreo (ATS) por Enaire. Estos servicios son: Control de Tráfico Aéreo (ATC), Servicios de Información de Vuelo (FIS) y Servicios de Alerta.

Los espacios aéreos controlados se clasifican, a su vez, en clases: A, B, C, D y E. Los espacios aéreos no controlados se clasifican en clases F y G. Las zonas peligrosas o restringidas se consideran espacio aéreo no clasificado (ver [1]).

**Tabla 2.1**. Clasificación y descripción del espacio aéreo. Fuente: elaboración propia a partir de datos de aip.enaire.es

| Clase | Tipo de vuelo | Separación proporcionada | Servicios suministrados | Radiocomunicación obligatoria | Sujeto a autorización ATC |
|---|---|---|---|---|---|
| A | Solo IFR | Todas aeronaves | ATC | Sí | Sí |
| B | IFR / VFR | Todas aeronaves | ATC | Sí | Sí |
| C | IFR | IFR de IFR | ATC | Sí | Sí |
| | VFR | VFR de IFR | ATC para separación IFR, Información VFR/VFR | Sí | Sí |
| D | IFR | IFR de IFR | ATC, información tránsito VFR | Sí | Sí |
| | VFR | Ninguna | ATC, información tráfico IFR/VFR y VFR/VFR | Sí | Sí |
| E | IFR | IFR de IFR | ATC, información vuelos VFR | Sí | Sí |
| | VFR | Ninguna | Información de tránsito | No | No |
| F | IFR | IFR de IFR siempre que sea factible | Servicio de información de vuelo | Sí | No |
| | VFR | Ninguna | Servicio de información de vuelo | No | No |
| G | IFR | Ninguna | Servicio de información de vuelo | Sí | No |
| | VFR | | | No | No |

## 2.2. Normativa vuelos VFR en territorio español

En este apartado se enumerarán y describirán las diferentes normativas para vuelos VFR en territorio español, clasificadas según la naturaleza de los condicionantes (ver [2]).

### 2.2.1. Condiciones de visibilidad y distancia de nubes

#### *2.2.1.1. Condiciones meteorológicas visuales (VMC)*

Antes de iniciar un vuelo VFR se han de comprobar los informes meteorológicos y pronósticos más recientes.

Dichos vuelos tendrán que cumplir con las condiciones de visibilidad y distancia de las nubes iguales o superiores a los de la tabla 2.2.

**Tabla 2.2**. Condiciones meteorológicas visuales. Fuente: guiavfr.enaire.es

Salvo cuando operen con carácter de vuelos VFR especiales, los vuelos VFR/OVFR se realizarán de forma que la aeronave vuele en condiciones de visibilidad y distancia de las nubes iguales o superiores a las indicadas en la tabla siguiente:

| TABLA SERA S5-1 DE CONDICIONES DE VISIBILIDAD Y DISTANCIA DE NUBES DE VUELOS VFR | | | | |
|---|---|---|---|---|
| ALTITUD | CLASES DE ESPACIO AÉREO | VISIBILIDAD DE VUELO | DISTANCIA DE NUBES | |
| | | | HORIZONTAL | VERTICAL |
| A 3050 m (10000 ft) AMSL o por encima (*) | A(**) B C D E F G | 8 km | 1500 m | 300 m (1000 ft) |
| Por debajo de 3050 m (10000 ft) AMSL y por encima de 900 m (3000 ft) AMSL, o por encima de 300 m (1000 ft) sobre el terreno, de ambos valores el mayor | | 5 km | | |
| A 900 m (3000 ft) AMSL o por debajo, o a 300 m (1000 ft) sobre el terreno, de ambos el mayor | A(**) B C D E | | | |
| | F G | 5 km (***) | Libre de nubes y con la superficie a la vista. | |

Si las condiciones meteorológicas se ven deterioradas hasta quedar por debajo de las VMC, el vuelo VFR que se realice como vuelo controlado deberá solicitar una autorización que le permita continuar hasta el destino o salir del espacio aéreo dentro del cual se necesita autorización ATC.

En caso de que no sea posible, se notificará a la dependencia ATC las medidas tomadas o se solicitará continuar como vuelo VFR especial o como vuelo IFR.

### *2.2.1.2. Techo de nubes y visibilidad en aeródromos*

Antes de ejecutar el vuelo VFR también se ha de comprobar la meteorología de los aeropuertos de origen y destino, así como alternativos (ver subapartado 2.3.4).

Las condiciones mínimas para despegar, aterrizar o entrar en la zona de tránsito del aeródromo especifican que el techo de nubes sea 1500 pies y que la visibilidad en tierra sea 5km.

Si no se alcanzan estos mínimos, se necesitará una autorización VFR especial para operar el vuelo.

### 2.2.2. Alturas de vuelo

La altura mínima de vuelo deberá ser:

- De 1000 pies por encima del terreno cuando se sobrevuelan lugares habitados, edificios, pueblos, etc. sobre el obstáculo más alto situado en un radio de 600m desde la aeronave.

- De 500 pies por encima del terreno en cualquier otra parte sobre el obstáculo más alto situado dentro de un radio de 150m desde la aeronave.

La altura máxima quedará limitada por el nivel de vuelo 195, a no ser que sea explícitamente aprobado por la autoridad ATS responsable o se haya reservado un espacio aéreo reservado por el Estado Español.

En lo correspondiente a los niveles de vuelo, la altitud queda delimitada por el rumbo de la aeronave y las normas de vuelo. En este caso, al tratarse de vuelos VFR, las altitudes (en pies) siempre deberán acabar en 500. Para rumbos de 270º a 89º empezarán por un número par, y de 90º a 269º por uno impar, tal como se muestra en la figura 2.3.

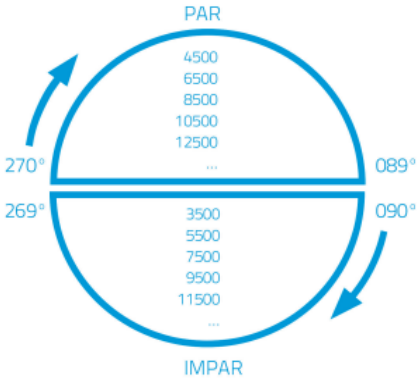



**Fig. 2.3.** Altitudes de vuelo en vuelos VFR. Fuente: guiavfr.enaire.es

### 2.2.3. Velocidades

Por debajo del FL100, la aeronave no podrá superar los 250 KIAS excepto:

- para la separación de tránsito, por orden de las dependencias ATS

- si las especificaciones del modelo de la aeronave requieren velocidad mayor para la seguridad en el funcionamiento

- que sea imprescindible por motivos de seguridad

- que haya una velocidad más alta permitida en la clase de espacio aéreo en el que se está

- que se tenga permiso de la Autoridad Aeronáutica Militar.

## 2.3. Cálculo de rutas VFR

En esta sección se hará referencia todo lo que hay que tener en cuenta a la hora de hacer los cálculos relativos a las rutas VFR.

En primer lugar, se han de tener en cuenta las limitaciones, tanto las de la aeronave como las de los aeródromos implicados en la ruta. Para estas limitaciones es importante tener acceso al Manual de Vuelo (POH).

En segundo lugar, se ha de trazar la ruta. A la hora de trazarla se deben hacer cálculos entre tramos, teniendo en cuenta el rendimiento de la aeronave, los rumbos, etc.

En tercer lugar, hay que revisar a meteorología. En concreto, el subapartado 2.3.4 trata sobre la lectura de METARs.

Por último, se deben rellenar ciertos documentos. En esta sección se mostrarán el NavLog y el guion de vuelo.

### 2.3.1. Limitación de la aeronave

El modelo de la aeronave condiciona el vuelo. Es por ello que hace falta consultar el POH para consultar sus límites:

- Las velocidades límites. Entre ellas figuran:
  - VNE (*Never Exceed Velocity*)
  - VNO (Velocidad máxima estructural en crucero)
  - Va (Velocidad de maniobra)
  - VFE (*Maximum Flap Extended Speed*)

- Las limitaciones de peso. Serán diferentes para categoría utilitaria o categoría normal. Entre ellas figuran:
  - MTOW: peso máximo al despegue
  - MLW: peso máximo al aterrizaje
  - Máximo peso de equipaje. Puede estar distribuido en diferentes áreas.
  - Peso del combustible. Dependen del número de tanques y del tamaño de estos. De la capacidad total de combustible, una parte especificada será utilizable y la otra no.

- Balance. A parte de las capacidades máximas de peso, se ha de tener en cuenta dónde se ubican todas las masas. La finalidad de esto es conseguir asegurar una estabilidad en la aeronave. Para ello el centro de masas ha de situarse por delante del centro de sustentación. Los POH incluyen una gráfica con la que, habiendo calculado previamente el peso y momento, se puede situar la aeronave y ver si coincide dentro de su categoría (ver Fig. 2.4)

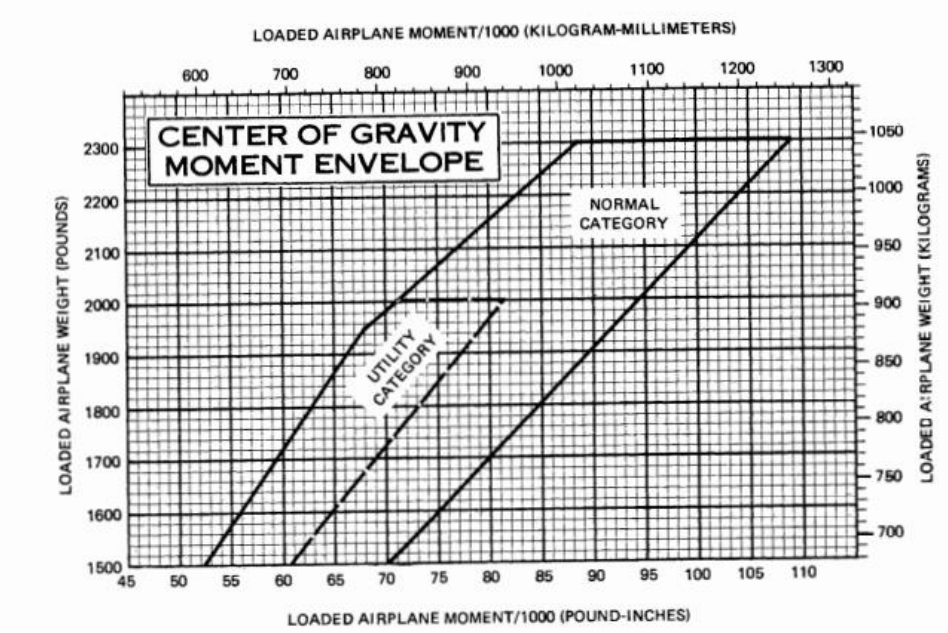



**Fig. 2.4.** Peso y Balance de Cessna 172N. Fuente: POH Skyhawk Cessna Model 172N

### 2.3.2. Limitación de los aeródromos

Los aeródromos han de cumplir los requisitos de visibilidad según el reglamento para los vuelos VFR en España. Además, deben ser aptos para el despegue o aterrizaje teniendo en cuenta los fenómenos meteorológicos del momento (ver apartado 2.3.4).

En la fase de planificación de la ruta, hay que tener en cuenta además las limitaciones de sus pistas.

La longitud de la pista ha de ser apta para el modelo de aeronave que se pretende volar. Las longitudes de despegue y aterrizaje se calculan con las distancias TORA y LDA facilitadas por el aeropuerto, con unos factores de corrección. Estos factores de corrección (ecuaciones 2.1, 2.2, 2.3) son para corregir la elevación, la temperatura y la pendiente de las condiciones estándar en las distancias TORA y LDA (ver [3]).

$$F_h = 1 + \frac{0{,}007\, h_{ref}}{300} \quad \textbf{(2.1)}$$

$$F_T = 1 + 0{,}01\left(T_{ref} - T_{SH}\right) \quad \textbf{(2.2)}$$

$$F_p = 1 + 0{,}1 \cdot pendiente(\%) \quad \textbf{(2.3)}$$

$$L_{despegue} = \frac{TORA}{F_h \cdot F_T \cdot F_p} \quad \textbf{(2.4)}$$

$$L_{aterrizaje} = \frac{LDA}{F_h \cdot F_T \cdot F_p} \quad \textbf{(2.5)}$$

Las distancias calculadas han de ser mayores que las mínimas especificadas en la tabla de Distancias de despegue/aterrizaje mostradas en el POH (ver tabla 2.3). En caso contrario, no será posible que la aeronave opere en ese aeródromo.

**Tabla 2.3**. Distancia de despegue para Cessna 172N. Fuente: POH Skyhawk Cessna Model 172N

| WEIGHT LBS | TAKEOFF SPEED KIAS | | PRESS ALT FT | 0°C | | 10°C | | 20°C | | 30°C | | 40°C | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | LIFT OFF | AT 50 FT | | GRND ROLL | TOTAL TO CLEAR 50 FT OBS | GRND ROLL | TOTAL TO CLEAR 50 FT OBS | GRND ROLL | TOTAL TO CLEAR 50 FT OBS | GRND ROLL | TOTAL TO CLEAR 50 FT OBS | GRND ROLL | TOTAL TO CLEAR 50 FT OBS |
| 2300 | 52 | 59 | S.L. | 720 | 1300 | 775 | 1390 | 835 | 1490 | 895 | 1590 | 960 | 1700 |
| | | | 1000 | 790 | 1420 | 850 | 1525 | 915 | 1630 | 980 | 1745 | 1050 | 1865 |
| | | | 2000 | 865 | 1555 | 930 | 1670 | 1000 | 1790 | 1075 | 1915 | 1155 | 2055 |
| | | | 3000 | 950 | 1710 | 1025 | 1835 | 1100 | 1970 | 1185 | 2115 | 1270 | 2265 |
| | | | 4000 | 1045 | 1880 | 1125 | 2025 | 1210 | 2175 | 1300 | 2335 | 1400 | 2510 |
| | | | 5000 | 1150 | 2075 | 1240 | 2240 | 1335 | 2410 | 1435 | 2595 | 1540 | 2795 |
| | | | 6000 | 1265 | 2305 | 1365 | 2485 | 1475 | 2680 | 1585 | 2895 | 1705 | 3125 |
| | | | 7000 | 1400 | 2565 | 1510 | 2770 | 1630 | 3000 | 1755 | 3245 | 1890 | 3515 |
| | | | 8000 | 1550 | 2870 | 1675 | 3110 | 1805 | 3375 | 1945 | 3670 | 2095 | 3990 |

## 2.3.3. Cálculo de parámetros

A la hora de trazar la ruta en un mapa, el primer paso es familiarizarse con la geografía de la zona por la cual se volará. Es útil identificar ríos, autopistas, montañas, etc. porque pueden servir como referencia visual. Tanto mapas físicos, como topográficos o mapas de fotos de satélite pueden ser útiles.

Una vez se ha examinado la geografía, se han de seleccionar los segmentos de vuelo. La idea es establecer puntos de ruta con referencias visuales del aeródromo de origen hacia al de destino. Estos puntos serán la división entre segmentos.

Con estos segmentos establecidos, se ha de comprobar la orografía del terreno y los límites del espacio aéreo para escoger las altitudes. Nuestro vuelo ha de cumplir con los reglamentos descritos anteriormente en el apartado 2.1 y 2.2.

Después, para cada segmento se deberá calcular:

- La longitud
- El rumbo de la aeronave
- La velocidad (KTAS). Ésta se extrae de las tablas de rendimiento del POH. En caso de saber el viento, también se puede calcular la GS.
- El tiempo que se tarda en recorrer el segmento

Una vez definida la ruta se ha de escoger un aeródromo alternativo. También se calcularán todos los parámetros enunciados anteriormente.

Con todos estos datos y las tablas de rendimiento del POH (ver Fig. 2.4), se puede hacer el cálculo del combustible. Para ello, se multiplica el tiempo en cada tramo por el valor de GPH del POH. El desglose del cómputo total de combustible requerido es:

- Combustible de arranque y rodadura
- Combustible requerido para ascenso y descenso
- Combustible en fase crucero
- Combustible requerido para ir al aeropuerto alternativo
- Combustible extra de 45 minutos en fase crucero

**Tabla 2.4**. Rendimiento en crucero para Cessna 172N. Fuente: POH Skyhawk Cessna Model 172N

| PRESSURE ALTITUDE FT | RPM | 20°C BELOW STANDARD TEMP | | | STANDARD TEMPERATURE | | | 20°C ABOVE STANDARD TEMP | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | % BHP | KTAS | GPH | % BHP | KTAS | GPH | % BHP | KTAS | GPH |
| 2000 | 2500 | --- | --- | --- | 75 | 116 | 8.4 | 71 | 115 | 7.9 |
| | 2400 | 72 | 111 | 8.0 | 67 | 111 | 7.5 | 63 | 110 | 7.1 |
| | 2300 | 64 | 106 | 7.1 | 60 | 105 | 6.7 | 56 | 105 | 6.3 |
| | 2200 | 56 | 101 | 6.3 | 53 | 100 | 6.1 | 50 | 99 | 5.8 |
| | 2100 | 50 | 95 | 5.8 | 47 | 94 | 5.6 | 45 | 93 | 5.4 |

Cabe recordar que el peso del combustible ha de tenerse en cuenta en el cómputo del cálculo de peso y balance.

### 2.3.4. Meteorología: el METAR

Se han de revisar antes del vuelo los fenómenos meteorológicos que puedan afectar a la ruta (mapas significativos), así como los vientos

(mapas de vientos). Estos parámetros pueden tener influencia para escoger por qué zonas pasará el vuelo y si se puede llevar a cabo.

Otro factor importante son las condiciones meteorológicas en los aeródromos escogidos. El METAR es un ejemplo de mensaje codificado que se emite para dar a conocer estas condiciones.

**Tabla 2.5**. Estructura de METAR. Fuente: elaboración propia a partir de METAR de FlightTrack

| 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|
| LEZG | 111630Z | 31015KT | CAVOK | 14/05 | Q1024 |

Los METAR tienen la siguiente estructura, ejemplificada en la tabla 2.5:

- 1. Identificador ICAO. Ej. LEZG: A. de Zaragoza)
- 2. Fecha y hora de observación. Ej. 11/01, 16:30Z
- 3. Vientos. Ej. 15 kt a 310º
- 4. Cobertura del cielo. Ej. CAVOK (techo y visibilidad OK)
- 5. Temperatura y temperatura de rocío. Ej. 14ºC y 5ºC
- 6. Presión. Ej. 1024 hPa

### 2.3.5. El NavLog y el guion de vuelo

Un par de documentos que sirven como herramienta a la hora de planificar nuestro vuelo son el navigation log (o NavLog) y el guion de vuelo.

El NavLog contiene en una tabla los valores relevantes sobre cada tramo del vuelo. La tabla 2.6 es un ejemplo de NavLog obtenido en FlightTrack. Sus campos son:

- WP: representa un tramo entre dos puntos de ruta
- Course: rumbo de la aeronave en ese tramo (en grados)
- Altitude: altitud en el tramo (en pies)
- Distance: la longitud del tramo (en millas náuticas)
- ETE: tiempo estimado en ruta para ese tramo (en minutos)
- ETA: tiempo estimado de llegada al siguiente tramo (en minutos)
- Fuel: combustible consumido en el tramo (en galones)

**Tabla 2.6**. Estructura de NavLog. Fuente: elaboración propia con FlightTrack

| WP | Course (°) | Altitude (ft) | Distance (NM) | ETE (min) | ETA (min) | Fuel (gal) |
|---|---|---|---|---|---|---|
| 0 - 1 | 86.81 | 2500 | 15.33 | 3.48 | 10:03 | 0.97 |
| 1 - 2 | 161.38 | 1500 | 6.39 | 8.76 | 10:12 | 0.41 |
| 2 - 3 | 194.83 | 3500 | 4.22 | 3.65 | 10:15 | 0.26 |
| 3 - 4 | 118.95 | 3500 | 4.47 | 2.41 | 10:18 | 0.28 |
| 4 - 5 | 88.87 | 2500 | 6.37 | 2.55 | 10:20 | 0.40 |
| 5 - 6 | 107.47 | 3500 | 7.19 | 3.64 | 10:24 | 0.44 |
| 6 - 7 | 51.38 | 2500 | 6.46 | 4.11 | 10:28 | 0.41 |
| 7 - 8 | 32.89 | 2500 | 2.45 | 3.69 | 10:32 | 0.15 |

El guion de vuelo sirve para detallar las diferentes referencias visuales. De esta manera, se escribe por cada tramo un encabezado resumiendo los valores de los campos más relevantes del NavLog y debajo se describen las diferentes referencias visuales (ver Fig. 2.5). Así se consigue preparar el vuelo y crear un documento que puede servir como guía a la hora de ejecutarlo.

**Tramo 0-1 - 15.33 NM - 3.48 min - 2500 ft - 86.81°**
Sobrevolar Marina de Cudeyo y seguir en ese rumbo hasta la Reserva Natural de las marismas de Santoña y Noja

**Tramo 1-2 - 6.39 NM - 8.76 min - 1500 ft - 161.38°**
Seguir el curso de las marismas y cruzar la A-8. Seguir ese rumbo hasta Ampuero

**Tramo 2-3 - 4.22 NM - 3.65 min - 3500 ft - 194.83°**
Seguir el curso del río por el valle hasta Ramales de la Victoria

**Tramo 3-4 - 4.47 NM - 2.41 min - 3500 ft - 118.95°**
Volar con la CA-150 a la izquierda hasta llegar a Ambasaguas.

**Tramo 4-5 - 6.37 NM - 2.55 min - 2500 ft - 88.87°**
Seguir con la CA-152 a mano izquierda, pasando a la derecha de Villanueva y el Valle de Villaverde hasta llegar a la alutra de Artzentales

**Fig. 2.5.** Extracto de guion de vuelo. Fuente: elaboración propia con FlightTrack

# CAPÍTULO 3. FLIGHTTRACK

En este capítulo se detallarán, en primer lugar, los aspectos relativos a la creación del software FlightTrack. Primero se expondrá el entorno de programación y el origen de los datos usados. Finalmente se explicará la estructura de la solución y las características y funcionalidades de la aplicación.

En segundo lugar, se presentará el manual de usuario del programa.

## 3.1. Creación del software

La aplicación FlightTrack podrá usarse tras su instalación como aplicativo de escritorio en ordenadores con un sistema operativo Windows.

A continuación, se detallan los aspectos relativos a su programación y el resultado final de la solución.


**Fig. 3.1.** Pantalla de carga de FlightTrack

### 3.1.1. Entorno de programación

El objetivo es crear un programa basado en formularios (Windows Forms). Para ello, se ha usado el IDE Microsoft Visual Studio.

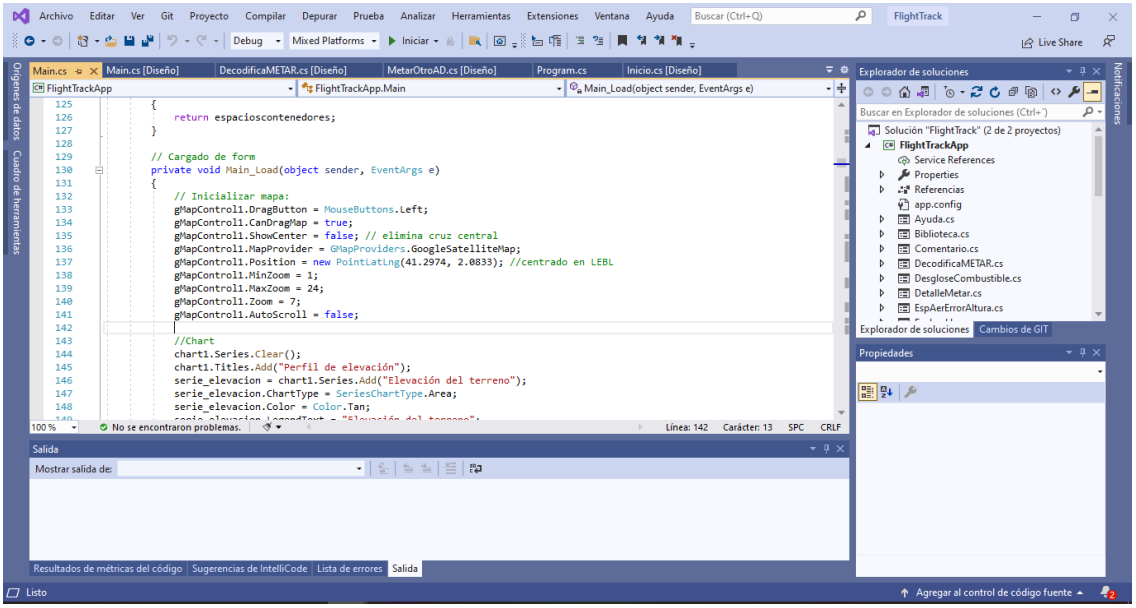



**Fig. 3.2.** Pantalla de Visual Studio con el formulario Main de FlightTrack

Este entorno de desarrollo integrado es compatible con diferentes lenguajes de programación. Además, permite la instalación de paquetes NuGet. Estos paquetes son unidades de código reutilizable que otros desarrolladores comparten para su uso en otros proyectos.

En concreto, FlightTrack ha sido programado en C# con Microsoft Visual Studio 2019, con la versión de .NET Framework 4.8.

A continuación, se detallan los paquetes de NuGet instalados y su utilidad en el proyecto:

- Conjunto perteneciente a Google.Apis. En la programación de FlightTrack se ha hecho uso de una interfaz de programación de aplicaciones (API) de Google. Para poder hacer uso de ella, era necesaria la instalación de los paquetes de Google.Apis. En el apartado 3.1.2.2 se detalla en qué consiste esta API.

- HtmlAgilityPack.NetCore. Este paquete NuGet permite el parseo de archivos HTML (para decodificar, por ejemplo, respuestas a peticiones de bases de datos).

- iTextSharp. Este paquete permite la creación y edición de archivos PDF en la programación de la aplicación.

- Newtonsoft.Json. JSON (JavaScript Object Notation) es un formato para guardar e intercambiar datos entre un servidor y un cliente. Con el paquete Newtonsoft.Json se pueden serializar y deserializar peticiones y

respuestas. De esta manera podemos obtener información de bases de datos externas, concretamente de las APIs del apartado 3.1.2.2.

- SharpKml. Este paquete es una implementación del estándar OGC KML 2.2 con la función de leer y escribir archivos KML y KMZ.

#### *3.1.1.1. GMap.NET*

Gmap.NET es un control .NET gratuito, multiplataforma y de código abierto que permite implementar en Windows Forms funciones de planificación de rutas, geocodificación y visualización de mapas de Google, Yahoo, Bing, OpenStreetMap, entre otros.

Para este proyecto únicamente se ha usado la funcionalidad de visualización de mapas y creación de rutas. Cabe remarcar que la creación de rutas de la clase GMapRoute ha sido únicamente usada para la visualización en el mapa, ya que los puntos del recorrido representado debían ser del tipo LatLong y no permitían contener más datos (como la altitud).

Por ello, se creó la clase RutaAer, donde se aprovechaba la lista de LatLong de GMapRoute y se añadían tanto las altitudes como otros parámetros útiles como el combustible en cada punto, por ejemplo. Esta era la clase usada para todas las operaciones realizadas en el programa.

### 3.1.2. Obtención de datos

En este subapartado se detalla el origen de los datos usado en FlightTrack: desde archivos cargados y desde APIs.

#### *3.1.2.1. Archivos cargados*

En la carga del programa, es necesario que se lean en primer lugar varios archivos con los datos necesarios para la ejecución de la aplicación:

- Archivo sobre las características de la aeronave. Este documento es un TXT con los datos extraídos manualmente del Pilot's Operating Handbook de la aeronave. En este caso, se ha escogido Cessna N. En caso de seguir desarrollando el programa, se podrían rellenar más ficheros sobre otras aeronaves (siguiendo el mismo formato) y dar al usuario la opción de escoger cual usar.

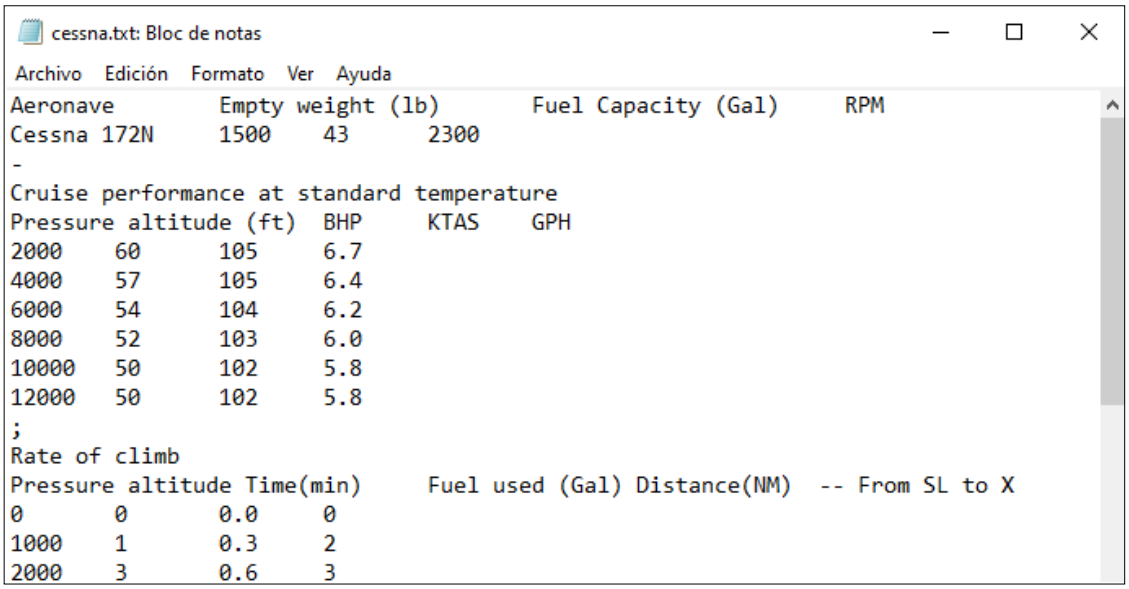



**Fig. 3.3.** Archivo de carga de la aeronave Cessna N. Fuente: elaboración propia a partir de datos del POH.

- Archivo sobre los espacios aéreos de España. Para la representación en el mapa de los diferentes espacios aéreos, se ha parseado el archivo spain.sua.kml (ver [4]).

- Archivo sobre los aeródromos VFR de España. Para la representación y recopilación de información relevante de los aeródromos que admiten vuelos VFR de España, se ha creado manualmente un archivo TXT con un listado de los no privados y de uso no exclusivo para servicios contraincendios. Estos datos se han extraído de la guía VFR de Enaire (ver [2]). En este TXT cada línea indica el nombre del aeropuerto, las coordenadas (longitud, latitud y elevación) y su identificador ICAO, separados por tabulación.

- Archivo con las radioayudas en España. Para la representación y extracción de datos de las radioayudas se ha parseado un KML proporcionado previamente por el profesor Albert Prades para la asignatura Projectes en Gestió del Trànsit Aeri.

#### *3.1.2.2. APIs*

Una API es un conjunto de definiciones y protocolos que permite la comunicación entre dos aplicaciones. De esta manera, se puede acceder a bases de datos y a información actualizada con solo una petición HTML.

En FlightTrack era necesario saber las elevaciones del terreno a lo largo de las rutas creadas para así poder calcular las altitudes mínimas seguras, asegurando una distancia de 1000 pies por encima del terreno.

Se recurrió a la API de Google Maps Elevation para la obtención de estos datos. Fue necesario crear una cuenta en Google Cloud y confirmar la identidad mediante tarjeta de crédito. Sólo así se podía obtener una clave API válida para hacer las peticiones al servidor.

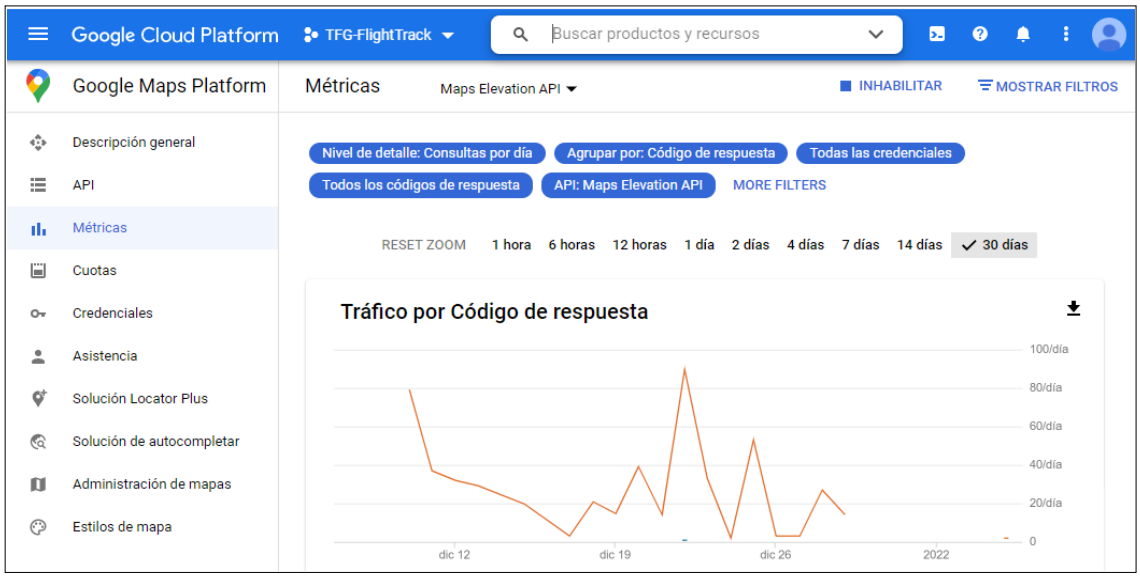



**Fig. 3.4.** Métricas de Maps Elevation API en Google Maps Platform

Se optó por pedir la elevación de todos los puntos muestreados de la ruta en una misma petición por eficiencia y coste, usando el método PathElevation. En este método se introduce el número de muestras y se hace una petición a un enlace compuesto por una URL base seguido de las coordenadas latitud y longitud separadas por “,” entre ellas y “|” con las otras y finalmente el número de muestras y la clave API.

La respuesta JSON es decodificada en el método GetCoordenadasTerreno y se añade esa altitud a la lista de coordenadas del terreno.

De esta manera posteriormente se busca por tramos entre puntos de ruta la altitud máxima del terreno y se añaden los 1000 pies, para así asegurar la distancia reglamentaria entre aeronave y suelo. Esta altitud puede ser modificada por la aplicación después teniendo en cuenta el rumbo del avión y las altitudes permitidas para vuelos VFR.

Además de esta interfaz de programación de aplicaciones, también se ha hecho uso de Aviation Weather API (ver [5]), de BluePony Technlogies. De la misma manera que en Google Cloud, también ha sido creada una cuenta en checkwxapi.com para obtener la clave API.

En la versión gratuita de esta API, se puede acceder a los METAR de todos los aeropuertos y a su decodificación. En FlightTrack, se puede consultar el último METAR emitido por el aeródromo (si es que tiene) y se da la opción de decodificar el mensaje.

### 3.1.3. Estructura de la solución

Para llevar a cabo la programación de FlightTrack, se crearon dos proyectos dentro de la solución. Uno recoge todos los formularios de la aplicación y el otro es una librería de clases utilizadas por el proyecto de formularios.

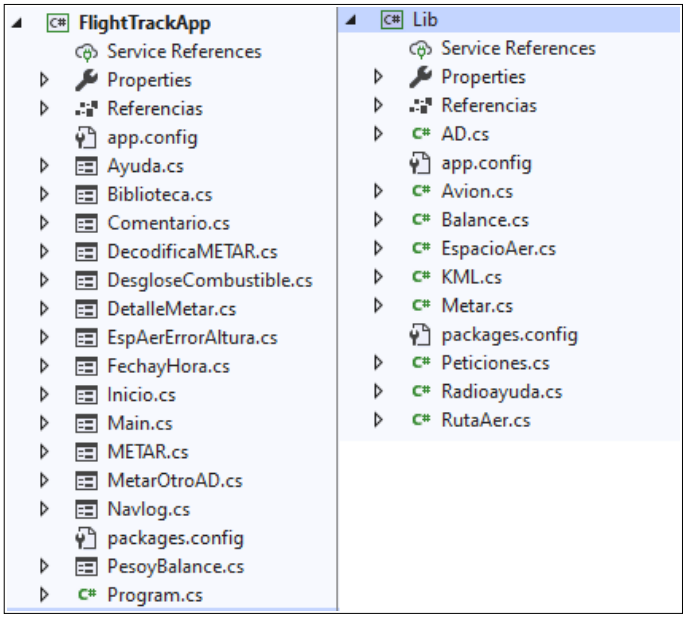



**Fig. 3.5.** Proyectos de la solución FlightTrack

El aspecto y contenido de los formularios se puede ver en el Manual de usuario (ver 3.2).

Las clases creadas para la librería de clases Lib, son brevemente explicadas a continuación:

- Clase *AD*. Esta clase contiene los atributos de los aeródromos. Estos atributos son nombre, coordenadas e identificador ICAO. Además de los métodos de set y get para la manipulación de atributos, tiene los métodos de leer la lista de aeródromos facilitada por el fichero y de buscar un aeródromo en la lista para obtener sus coordenadas o ICAO.

- Clase *Avion*. Esta clase inicializa los objetos de tipo *Avion* a partir de la lectura del fichero con el modelo. Los datos de este fichero se almacenan dentro del objeto. Por ello los atributos son todas las características que contiene el TXT. Por ello, si en un futuro desarrollo de la aplicación se usaran otros modelos diferentes del Cessna 172N, se podría usar esta clase para decodificarlos y usarlos.

- Clase *Balance*. Esta clase es exclusivamente usada por el formulario PesoyBalance. Tiene métodos para obtener la representación gráfica del momento y peso del avión cargado usando el espacio de nombres System.Drawing.

- Clase *EspacioAer*. Esta clase contiene los atributos de los diferentes espacios aéreos. Estos atributos son: el nombre, una breve descripción, la clase y los límites superior e inferior (tanto el tipo como la cifra). Los tipos de límite pueden ser SFC, UNTLD, FL, AGL y ALT. SFC indica que el espacio aéreo empieza a nivel del terreno, UNTLD se usa cuando no tiene límite superior, FL cuando es del tipo Flight Level (nivel de vuelo), AGL cuando se da la altura sobre el suelo y ALT cuando es la altitud sobre el nivel medio del mar.
  Para la decodificación del KML sobre los espacios aéreos del estado español se usa esta clase, resultando de ello los espacios aéreos con todas sus características en una lista.

- Clase *KML*. Esta clase es usada para la codificación en KML usando el paquete NuGet SharpKml.

- Clase *Metar*. En esta clase se inicializa el constructor con un vector JSON, que contiene la respuesta a la petición formulada sobre el METAR. También se ha implementado un método para la decodificación de la cobertura de nubes a partir del código de letras obtenido.

  El objeto obtenido podrá tener o no los atributos de presión, cobertura de nubes, temperaturas, vientos, etc. extraídos del METAR. Este METAR será el más reciente emitido por el aeródromo seleccionado. Al usuario se le da la opción de leer el METAR sin modificar o de decodificarlo.

- Clase *Peticiones*. Esta clase es usada específicamente para las peticiones a la API de elevaciones de Google.

- Clase *Radioayuda*. Esta clase contiene los atributos de las radioayudas de España. Estos atributos son su identificación, nombre, tipo de radioayuda, frecuencia y coordenadas de ubicación. Contiene un método para leer el fichero KML y inicializar las radioayudas, con sus respectivos métodos de set y get.

- Clase *RutaAer*. Esta clase es la usada para la creación, modificación, cargado y guardado de las rutas aéreas. En ella también se hacen las peticiones de elevación del terreno, se calculan las mínimas altitudes

seguras y se comprueba la contención de puntos de la ruta dentro de espacios aéreos, entre otros.

### 3.1.4. Características y funcionalidades

Finalmente, se ha generado FlightTrack, la aplicación de escritorio para la planificación de rutas de vuelos VFR. Sus características y funcionalidades son:

- Planificación de rutas en vuelos VFR en cualquier punto del territorio español
- Ejecutable en ordenadores con SO Windows
- Varios mapas disponibles para seleccionar
- Cálculo de altitudes mínimas seguras
- Visualización de perfil vertical de la ruta, con las elevaciones del terreno
- Visualización en el mapa de:
  - Espacios aéreos
  - Radioayudas. Identificador y frecuencia
  - Puntos en ruta
- Opción de ver lista de espacios aéreos atravesados en ruta y sus límites
- Cálculo automático de combustible. Opción de ver desglose.
- Programación de fecha y hora para el vuelo, con formato ZULU, hora local de península u hora local en Islas Canarias.
- Relleno automático de NavLog. Opción a descarga en PDF.
- Opción de añadir comentarios a los tramos de la ruta para la creación de guion de vuelo y descarga en PDF.
- Cálculo de peso y balance para categoría Normal y representación gráfica
- Decodificación de METARs de aeropuertos seleccionados
- Importación KML de la ruta
- Ventana de ayuda con pasos
- Biblioteca (logbook). Registro de vuelos. Opción de guardar/cargar/borrar. Opción de modificar y guardar una ruta cargada. Opción de puntuar las rutas.

## 3.2. Manual de usuario

En esta sección, se explicará en detalle cómo usar FlightTrack. Se detallará qué contiene cada formulario (o ventana) y cómo se accede a cada uno, así como sus funcionalidades.

### 3.2.1. Main

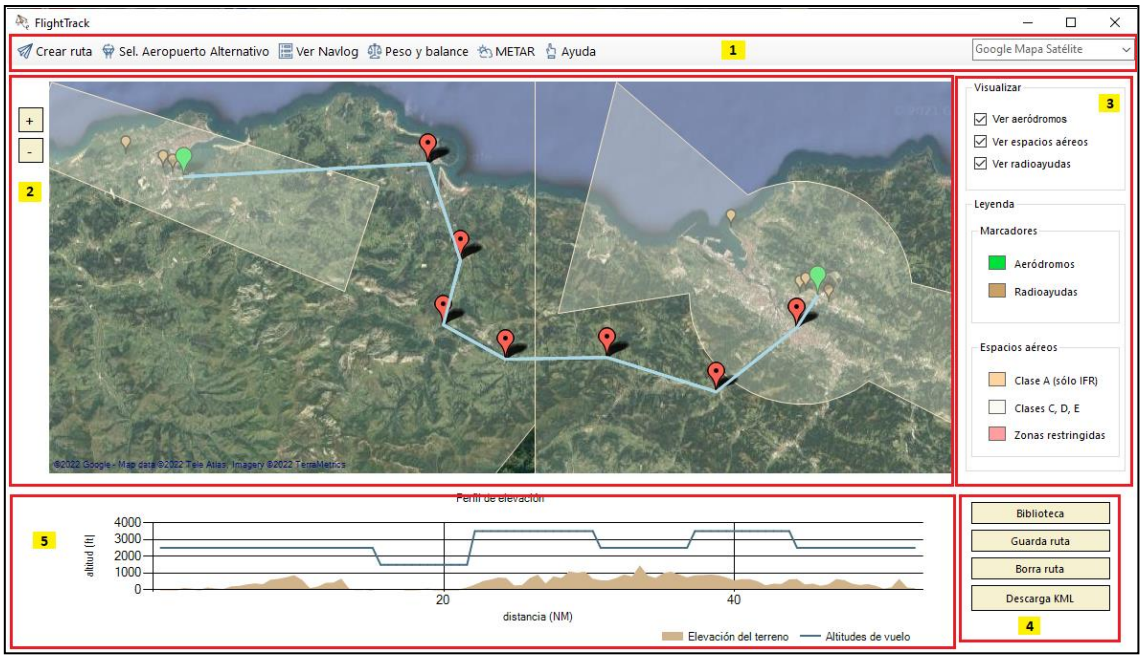



**Fig. 3.6.** Pantalla principal de FlightTrack

Main es el formulario principal. Consta de una barra de herramientas superior (1), un mapa a la izquierda (2), unas cajas de leyenda a la derecha (3) y tres botones en la parte inferior derecha (4). Una vez se crea o carga una ruta, aparece el perfil vertical de ésta en la parte inferior (5).

En la barra de herramientas superior se encuentran los botones principales y un desplegable que se puede usar para cambiar el tipo de mapa. Por defecto, aparece el Mapa de Google Satélite. También se pueden usar varios mapas topográficos y con detalles del terreno.

Los botones alineados a la izquierda en la barra de herramientas están ordenados por orden de uso. El primero, sirve para crear una ruta desde cero.

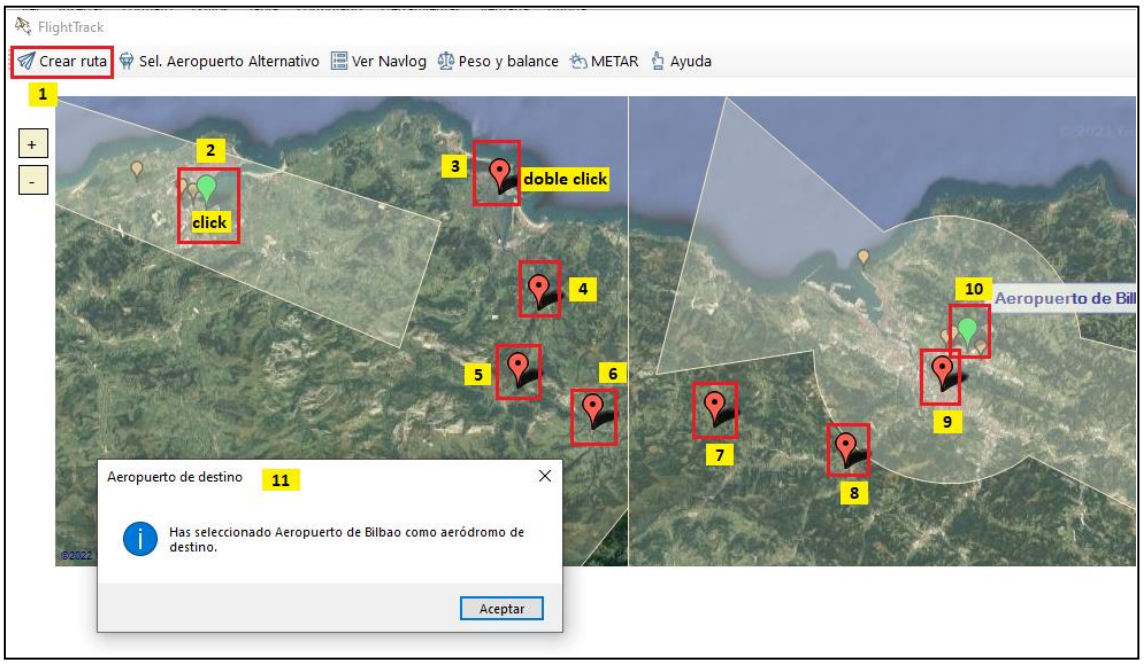



**Fig. 3.7.** Creación de Ruta

Para ello, se hace clic a ese botón que inicia el evento de crear ruta (1). Entonces se hace clic sobre el marcador del aeródromo que queremos como origen (2). Para comprobar que se ha marcado bien, sale una ventana emergente diciéndonos cuál se ha escogido. Después se marcan sobre el mapa los waypoints (3-9) con un doble clic y finalmente se vuelve a escoger el aeródromo (10), esta vez de destino, con un clic. Nuevamente aparecerá una ventana emergente si hemos marcado bien el aeropuerto (11).

Una vez se ha escogido el aeropuerto de destino, se dibuja la ruta. A partir de este momento, ya se puede guardar en la Biblioteca o importar en KML con los botones de la parte inferior derecha.

Es posible que la altitud mínima calculada de la ruta exceda la altitud máxima para un vuelo visual. En ese caso, el programa lo notificará con una ventana emergente, y preguntará al usuario si desea ver qué espacios aéreos está cruzando y cuáles son sus límites, para así plantear mejor la ruta. Este formulario, Espacios Aéreos, es accesible también desde el formulario NavLog.

El siguiente paso sería seleccionar un aeropuerto alternativo. Se puede empezar a hacer el NavLog sin este parámetro, pero no se podrá rellenar la columna de combustible ni hacer el cálculo de combustible total, ya que para ello se necesita tener en cuenta el combustible necesario para llegar al aeropuerto alternativo. Por lo tanto, es mejor seleccionar el aeropuerto alternativo desde un principio. Para hacerlo, simplemente se hace clic en el botón de la barra de herramientas “Sel. Aeropuerto Alternativo” y otro clic en el aeropuerto que elegimos. De nuevo saldrá una ventana emergente confirmando nuestra elección.

Ahora ya se pueden obtener todos los parámetros en el NavLog o en Peso y Balance. Para acceder a estos formularios, sólo hace falta hacer clic en sus respectivos botones de la barra de herramientas. En caso de querer consultar cualquier METAR o ver la ventana de ayuda, del mismo modo sólo hay que clicar sus correspondientes botones.

En lo que al uso del mapa se refiere, se puede tanto usar el ratón como los botones a la izquierda para hacer zoom. Se mueve la zona haciendo clic y arrastrando. Además, cuando se dibuja la ruta automáticamente se centra y hace zoom en la zona de interés.

A la derecha se encuentran las leyendas. En la leyenda superior se puede marcar o desmarcar la visualización de los espacios aéreos, radioayudas y aeródromos.

Finalmente, abajo a la derecha se encuentran cuatro botones. El primero abrirá la ventana de Biblioteca, donde se almacenan los vuelos. El segundo (Guardar ruta) permite guardar la ruta en la Biblioteca. El tercer botón (Borrar ruta) borra todo lo que se haya hecho en este formulario principal para poder crear o cargar otra ruta. El último botón es para importar la ruta en formato KML a cualquier directorio del ordenador.

### 3.2.2. NavLog

Para acceder a la pantalla del NavLog, hay que hacer clic sobre el botón “Ver NavLog” de la barra de herramientas de la pantalla principal.

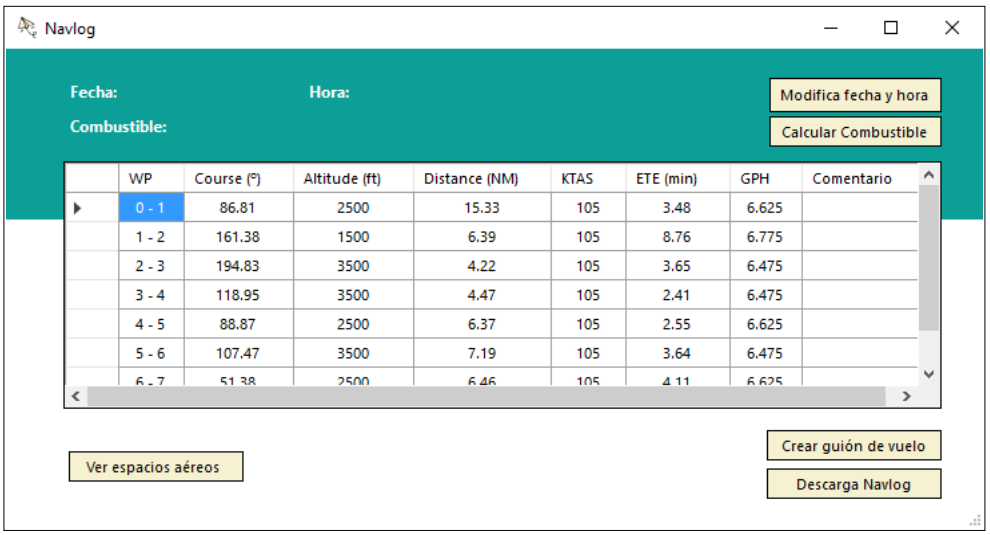



**Fig. 3.8.** Pantalla NavLog

El formulario consta de un texto arriba a la izquierda a modo de información, cinco botones y una tabla en el centro.

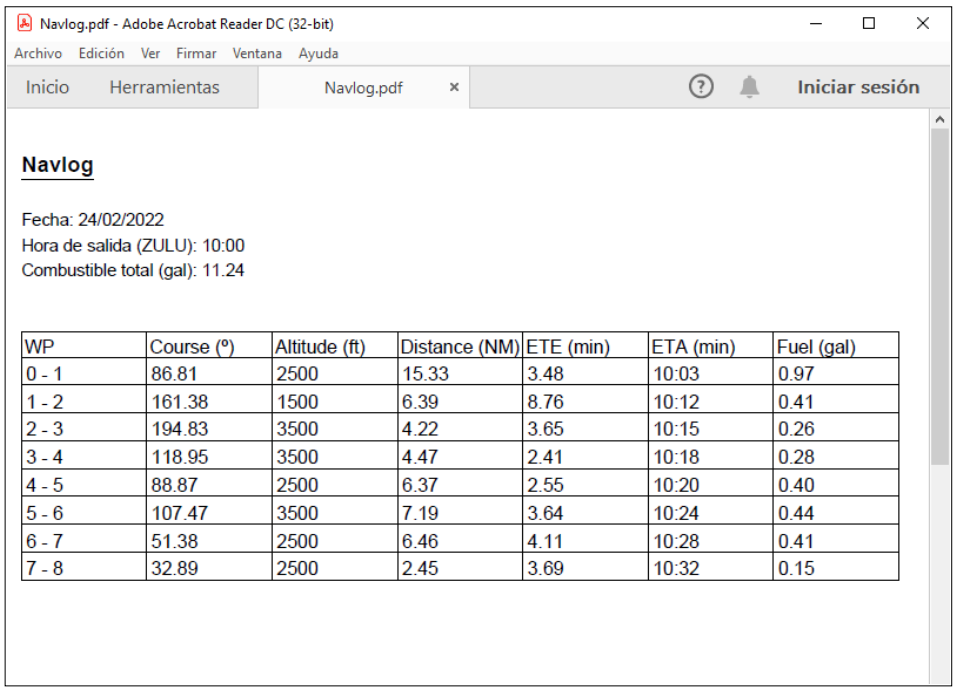
Cuando se crea una ruta nueva, los campos fecha, hora y combustible aparecen vacíos. Para rellenarlos, hay que acceder a los formularios de Fecha y Hora (clicando en el botón “Modifica Fecha y Hora”) y al de Combustible (clicando el botón “Calcular Combustible”). En el caso de tratarse de una ruta cargada, estos campos saldrán rellenados automáticamente, dando la opción de modificar la fecha y la hora, pero no la de volver a calcular el combustible (este botón no aparece en este caso particular).

La tabla central contiene la información del NavLog. Cada fila representa un tramo entre puntos de ruta, siendo 0-1 el tramo desde el aeropuerto de origen hasta el waypoint 1, por ejemplo. Los campos que aparecen en la tabla se van modificando y añadiendo conforme se concretan ciertas informaciones. Cuando se añade la hora de inicio de ruta, se añade una columna llamada ETA a la derecha de la columna ya existente ETE. Una vez el combustible es calculado, se cambia la columna de GPH (galones por hora en cada tramo) por la de Fuel, siendo esta la de combustible total consumido en cada tramo. La columna de Comentarios será explicada en el apartado 3.2.2.1.

Si se quiere consultar por qué espacios aéreos se cruza, se puede hacer clic al botón “Ver Espacios Aéreos”. Esta pantalla se explicará con detalle en el apartado 3.2.2.4.

Finalmente, se pueden descargar dos documentos: el NavLog (haciendo clic en el botón “Descarga NavLog” y el guion de vuelo (haciendo clic en el botón “Crear guion de vuelo”). Los comentarios de la columna Comentarios serán las explicaciones que aparezcan en el documento del guion de vuelo. Ambos archivos son descargados en formato PDF.


**Fig. 3.9.** Descargable: guion de vuelo




| WP | Course (°) | Altitude (ft) | Distance (NM) | ETE (min) | ETA (min) | Fuel (gal) |
|---|---|---|---|---|---|---|
| 0 - 1 | 86.81 | 2500 | 15.33 | 3.48 | 10:03 | 0.97 |
| 1 - 2 | 161.38 | 1500 | 6.39 | 8.76 | 10:12 | 0.41 |
| 2 - 3 | 194.83 | 3500 | 4.22 | 3.65 | 10:15 | 0.26 |
| 3 - 4 | 118.95 | 3500 | 4.47 | 2.41 | 10:18 | 0.28 |
| 4 - 5 | 88.87 | 2500 | 6.37 | 2.55 | 10:20 | 0.40 |
| 5 - 6 | 107.47 | 3500 | 7.19 | 3.64 | 10:24 | 0.44 |
| 6 - 7 | 51.38 | 2500 | 6.46 | 4.11 | 10:28 | 0.41 |
| 7 - 8 | 32.89 | 2500 | 2.45 | 3.69 | 10:32 | 0.15 |

**Fig. 3.10.** Descargable: NavLog

### *3.2.2.1. Comentarios*

Para abrir la ventana emergente de Comentarios, hacer clic sobre la celda de la columna Comentarios en cuya fila se quiera añadir el texto (1).

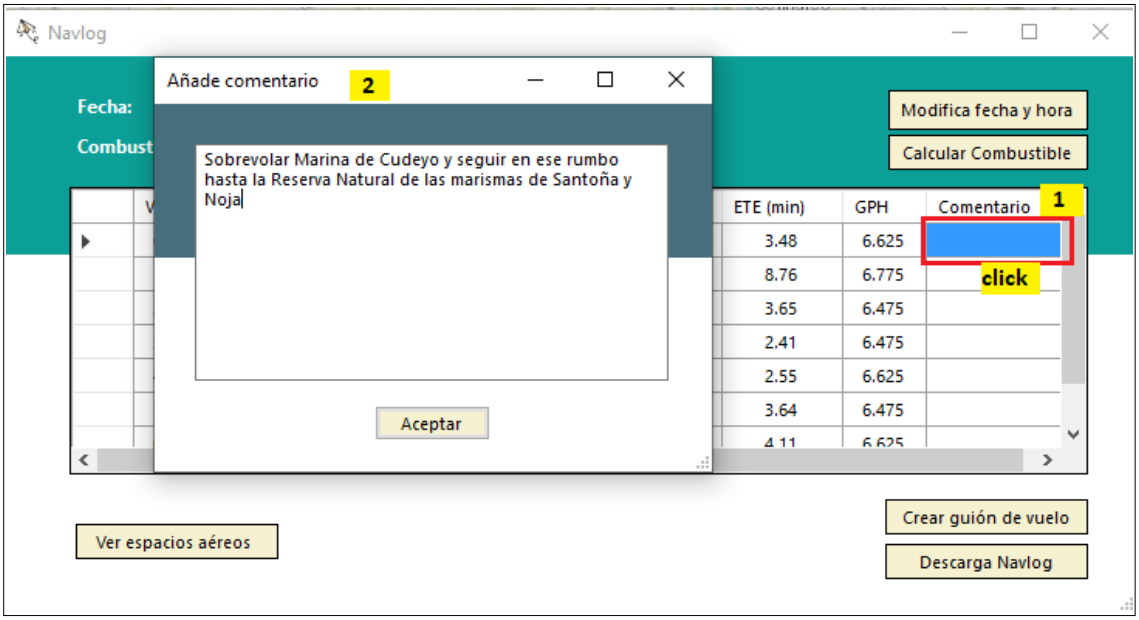



**Fig. 3.11.** Agregar comentario

Una vez escrito el comentario (2), hacer clic en el botón Aceptar. Como comprobación de que la explicación ha sido guardada, en la celda aparecerá el texto “Contiene comentario”. Si se quisiera modificar, bastaría con hacer clic en la celda de nuevo y repetir el proceso.

Una vez se hayan escrito todos los comentarios deseados, se puede descargar el guion de vuelo en la pantalla NavLog.

#### *3.2.2.2. Fecha y hora*

Para la modificación de fecha y hora, hacer clic en el botón “Modifica fecha y hora” de la ventana NavLog.

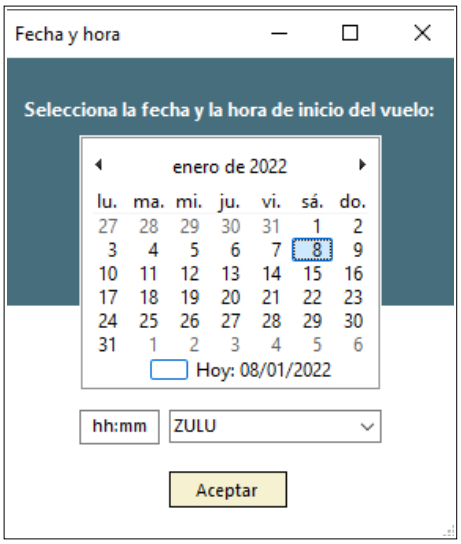
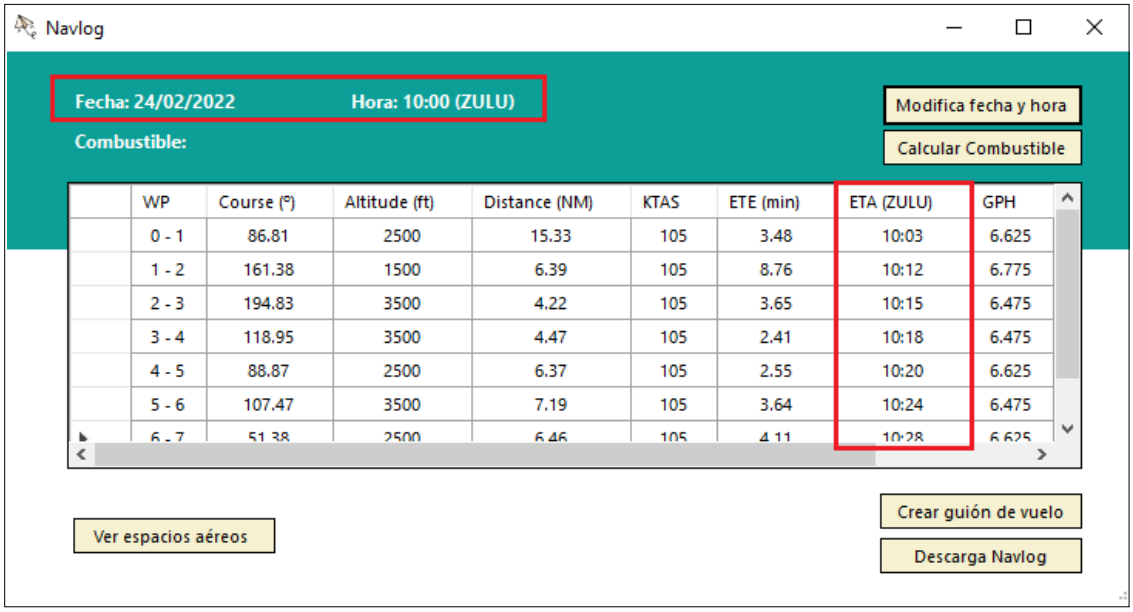



**Fig. 3.12.** Modificar fecha y hora

La ventana muestra un calendario y una caja donde escribir la hora, así como un desplegable para especificar el formato de la hora introducida. Esto es necesario ya que en el NavLog la hora utilizada será la hora ZULU, así que si se introduce la hora local peninsular, se sumará una hora.

Es importante también introducir la hora con el formato hh:mm, contemplando desde las 00:00 hasta las 23:59. Si la hora no siguiera las especificaciones de formato, se notificaría un error y no dejaría introducirlas en el NavLog.

Una vez aceptados los datos, se rellenarían los campos fecha y hora en la ventana de NavLog, y se añadiría a la tabla la columna ETA, como se ha explicado anteriormente.




| | WP | Course (°) | Altitude (ft) | Distance (NM) | KTAS | ETE (min) | ETA (ZULU) | GPH |
|---|---|---|---|---|---|---|---|---|
| | 0 - 1 | 86.81 | 2500 | 15.33 | 105 | 3.48 | 10:03 | 6.625 |
| | 1 - 2 | 161.38 | 1500 | 6.39 | 105 | 8.76 | 10:12 | 6.775 |
| | 2 - 3 | 194.83 | 3500 | 4.22 | 105 | 3.65 | 10:15 | 6.475 |
| | 3 - 4 | 118.95 | 3500 | 4.47 | 105 | 2.41 | 10:18 | 6.475 |
| | 4 - 5 | 88.87 | 2500 | 6.37 | 105 | 2.55 | 10:20 | 6.625 |
| | 5 - 6 | 107.47 | 3500 | 7.19 | 105 | 3.64 | 10:24 | 6.475 |
| | 6 - 7 | 51.38 | 2500 | 6.46 | 105 | 4.11 | 10:28 | 6.625 |

**Fig. 3.13.** NavLog después de modificar fecha y hora

### *3.2.2.3. Combustible*

Para el cálculo del combustible, hacer clic en el botón “Calcular Combustible” de la ventana NavLog. Es necesario haber seleccionado un aeropuerto alternativo para llevar a cabo este paso.

Si el aeropuerto alternativo está seleccionado, se abrirá una ventana emergente que dice cuál es el combustible total requerido y pregunta si se quiere ver el desglose.

En caso de hacer clic en “Sí” aparecerá la ventana de “Desglose de Combustible”, donde aparecen los combustibles por fases (ascenso, descenso, crucero, arranque y rodadura, 45 minutos extra en fase crucero y combustible para llegar a aeropuerto alternativo).

Una vez cerradas estas ventanas y de vuelta a la de NavLog, el campo “Combustible” habrá sido rellenado con la cantidad total y ya no se podrá volver a calcular el combustible, así que el botón no es visible.

### *3.2.2.4 Espacios Aéreos*

La última funcionalidad de la ventana NavLog. Haciendo clic en el botón “Ver espacios aéreos”, accedemos al formulario “Espacios Aéreos”.

Esta ventana contiene una lista de todos los espacios aéreos atravesados por la ruta, con su nombre, clase y límites. Así, en caso de tener que replantear la ruta, podría verse con más detalle qué zonas evitar para cumplir con los límites de altitud para vuelo visual.

Seleccionando la fila que se desea consultar con un clic, se coloreará en amarillo en el mapa de la pantalla principal ese espacio aéreo.

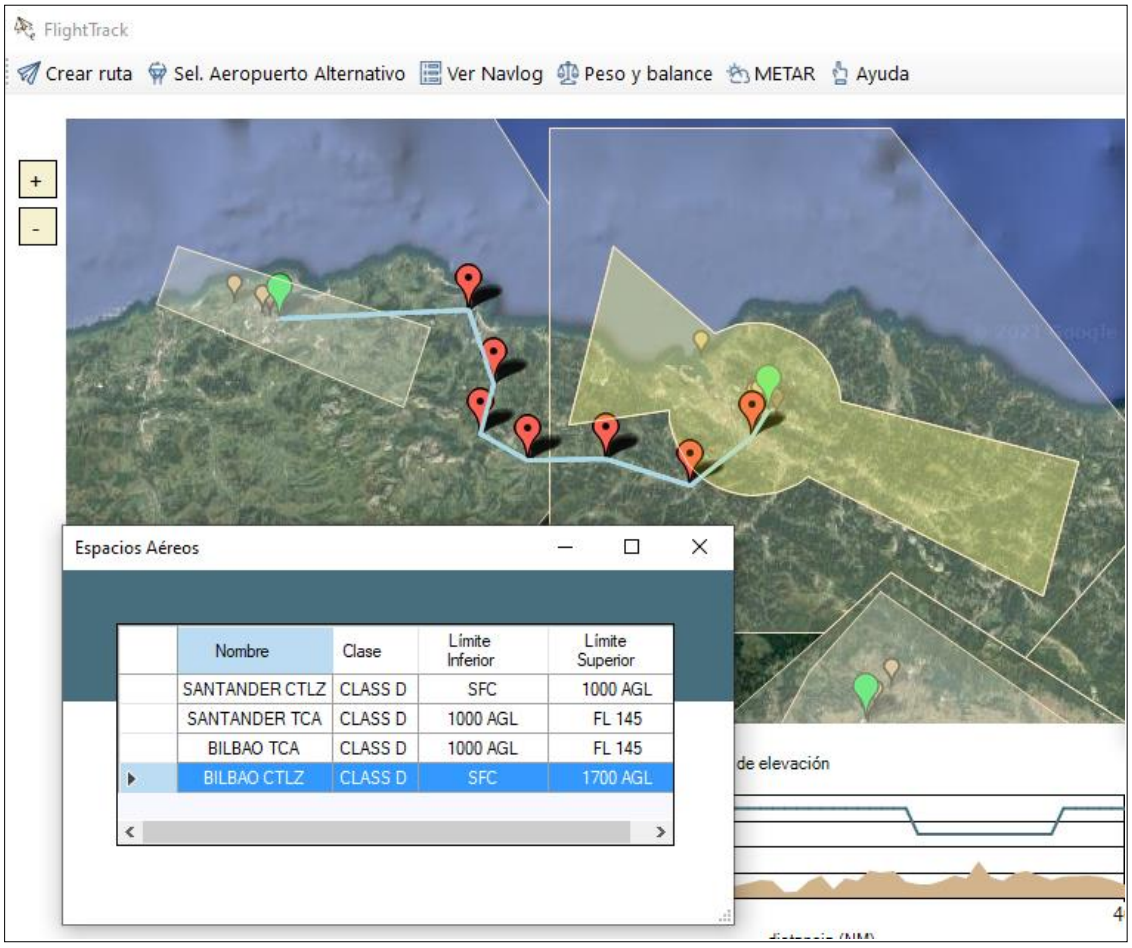



**Fig. 3.14.** Espacios Aéreos

### 3.2.3. Peso y Balance

Para acceder a la pantalla de Peso y balance, hay que hacer clic sobre el botón “Peso y Balance” de la barra de herramientas de la pantalla principal.

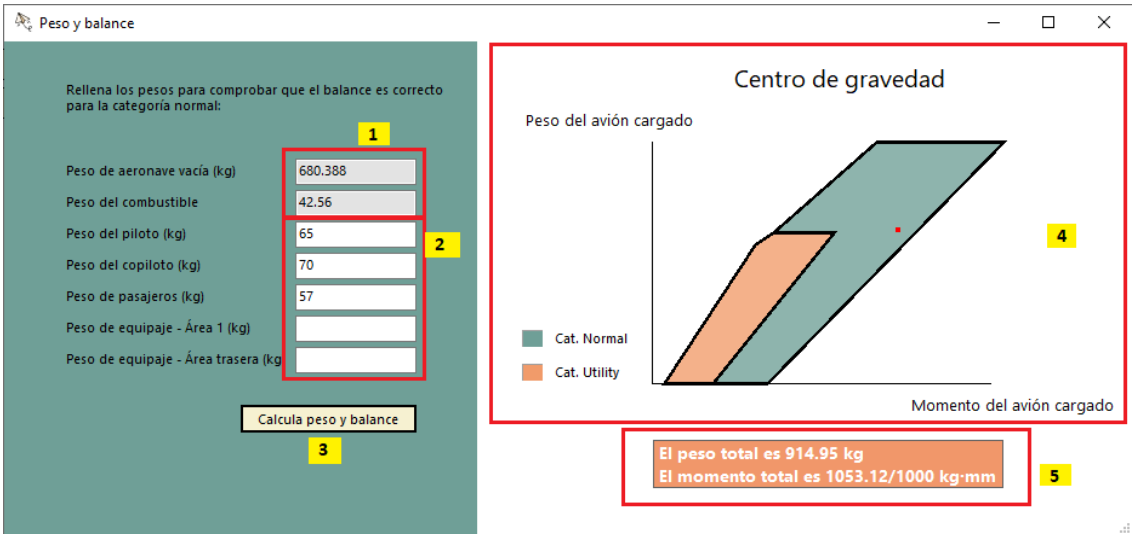



**Fig. 3.15.** Ventana de peso y balance

A la izquierda, están los pesos a rellenar para poder calcular el peso total y el momento. El peso de la aeronave y el combustible salen automáticamente rellenados (1). Los otros, se han de introducir (2). Todos los pesos han de estar en kilogramos. Si una casilla tiene valor nulo se puede poner 0 o dejar en blanco.

A la derecha está el gráfico correspondiente del Manual del piloto sobre la posición del centro de gravedad (2).

El siguiente paso una vez rellenadas las casillas de la izquierda es calcular el peso y el balance. Para ello, hacer clic en el botón homónimo (3). En caso de exceder algún límite (como el peso del equipaje) o dejar nulo el peso del piloto, el programa lo notificará. El resultado de los cálculos saldrá en una caja (5) que aparece tras pulsar el botón (3). El resultado también se pinta en forma de punto rojo en el gráfico (4).

En caso de estar dentro de los parámetros de la categoría normal, el punto estará dentro del polígono azul. De lo contrario, el programa lo notificará para así poder modificar los pesos.

### 3.2.4. METAR

Para acceder a la pantalla de METAR, hay que hacer clic sobre el botón “METAR” de la barra de herramientas de la pantalla principal.

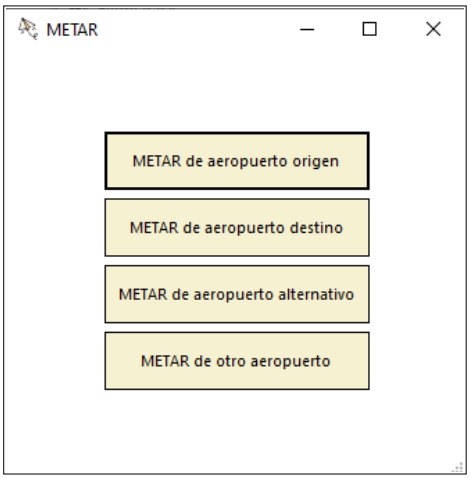



**Fig. 3.16.** Ventana principal de METAR

En esta ventana aparecen cuatro opciones. Las tres primeras son para los aeropuertos específicos de la ruta. Como es posible que algún aeródromo no emita METARs, hay una cuarta opción para consultar METAR de cualquier aeropuerto. De esta manera se puede consultar el METAR de un aeropuerto cercano.

Haciendo clic en cualquiera de los tres primeros botones, accedemos al METAR (Fig. 3.17).

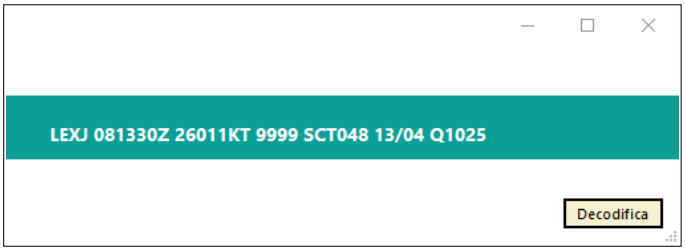



**Fig. 3.17.** METAR de aeropuerto de Seve Ballesteros-Santander

Clicando en decodificar accedemos a METAR Decodificado (Fig. 3.18).

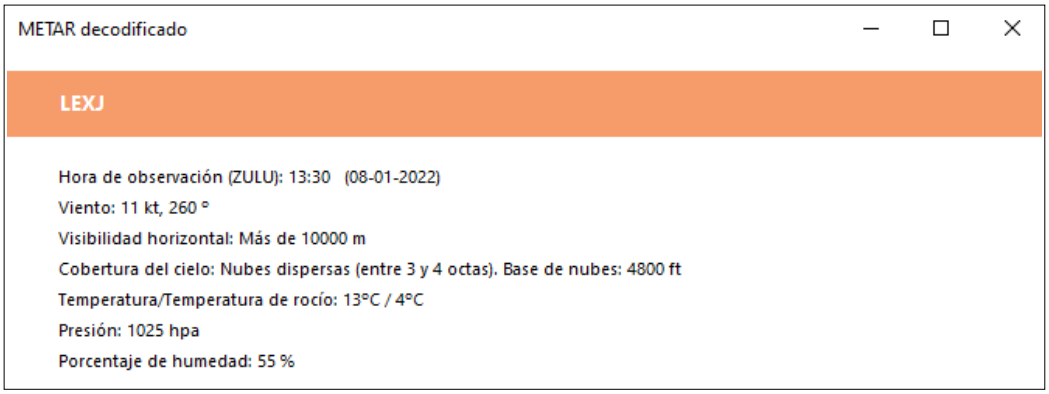



**Fig. 3.18.** METAR decodificado de aeropuerto de Seve Ballesteros-Santander

En caso de seleccionar la opción de “METAR” de otro aeropuerto, accedemos a la ventana mostrada en la Fig. 3.18 En ella hay un desplegable con todos los aeródromos. Al seleccionar uno, aparece el METAR en la franja azul. Si se hace clic en Decodifica se accede a la pantalla mostrada anteriormente en la Fig. 3.17.

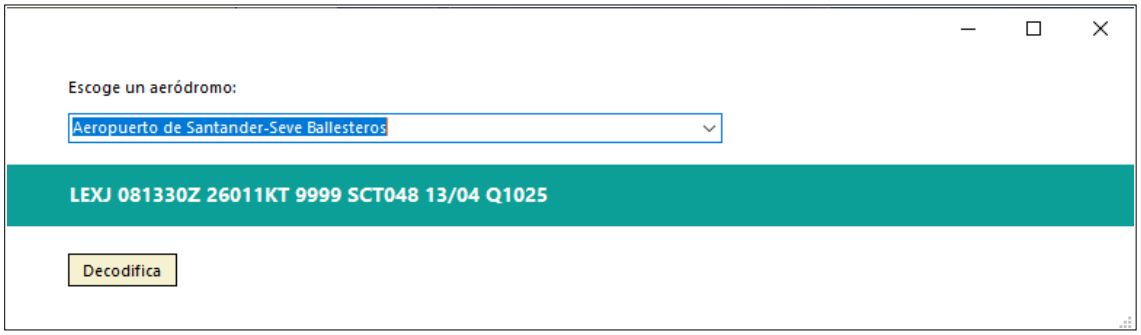



**Fig. 3.19.** Selección de METAR de cualquier aeropuerto

### 3.2.5. Ayuda

Para acceder a la pantalla de Ayuda, hay que hacer clic sobre el botón “Ayuda” de la barra de herramientas de la pantalla principal.

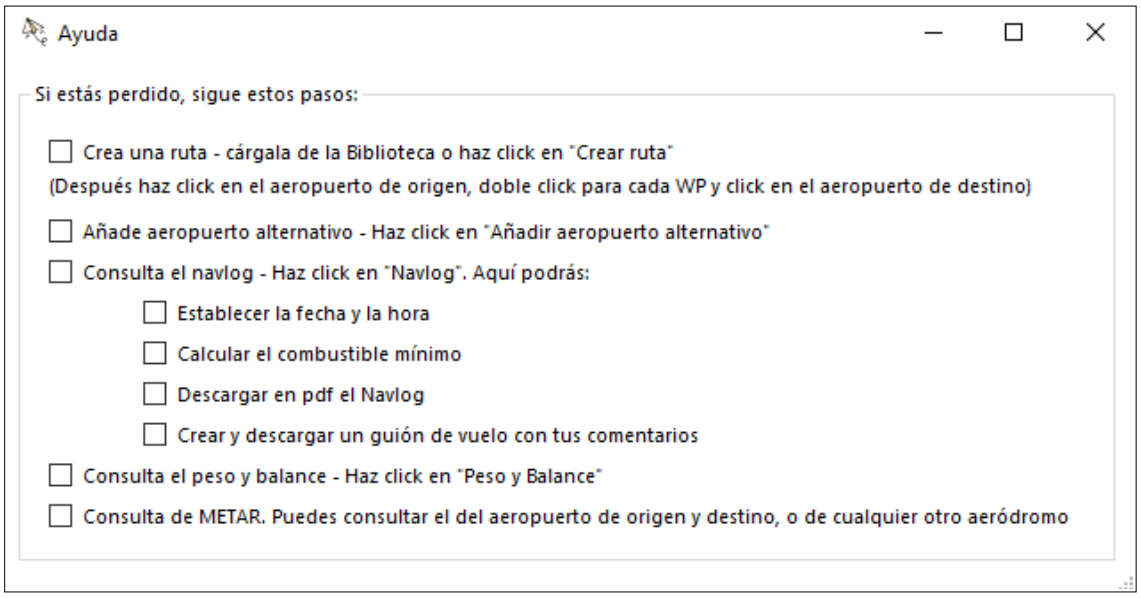



**Fig. 3.20.** Ventana de Ayuda

En esta ventana se muestra un listado de acciones que hacer en FlightTrack. En caso de no saber cuál podría ser el siguiente paso, se puede consultar. Los ítems de la lista se van marcando con un tick conforme se van haciendo, para saber qué se ha hecho y qué no.

### 3.2.5. Biblioteca

Para acceder a la pantalla de Biblioteca, hay que hacer clic sobre el botón “Biblioteca” en la parte inferior derecha de la pantalla principal.

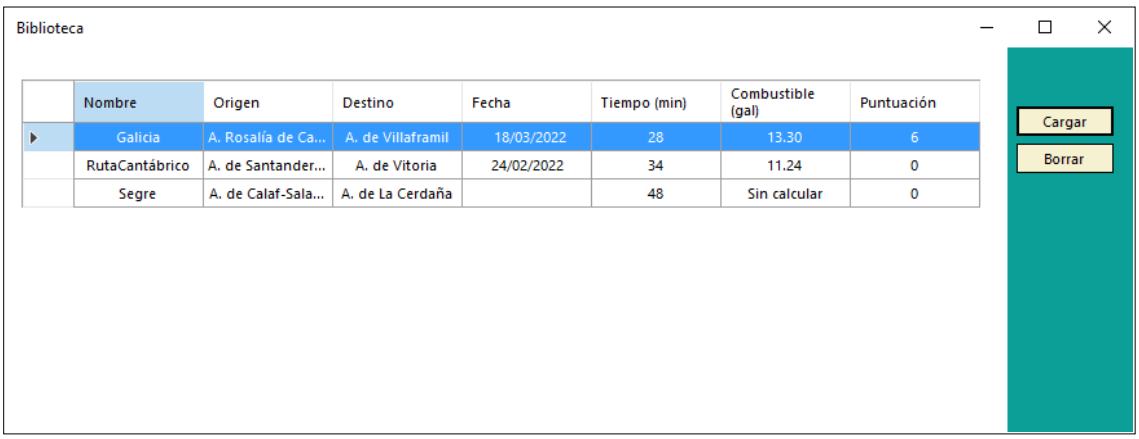



**Fig. 3.21.** Ventana de Biblioteca

En Biblioteca hay un registro de los vuelos guardados. Los campos que siempre aparecen rellenados son el nombre de la ruta (nombre puesto al guardar), los aeropuertos de origen y destino y el tiempo. Si se han establecido la fecha, también aparecerá en la tabla. Si no, esa celda estará vacía. Si el combustible fue calculado, también aparecerá en la tabla. En caso contrario aparecerá “Sin calcular” en la celda correspondiente.

Además, hay una columna extra para ponerle una puntuación a la ruta. Simplemente hay que clicar la celda deseada y escribir la puntuación.

Finalmente, se pueden cargar las rutas a FlightTrack para hacer modificaciones o consultar con detalle con el botón “Cargar”, después de seleccionar la fila deseada. También se pueden borrar permanentemente rutas con el botón “Borrar”, también seleccionando la fila deseada.

# CAPÍTULO 4. PARTIDA ECONÓMICA

Con tal de calcular el precio del desarrollo de FlightTrack, se han de tener en cuenta:

- El sueldo del ingeniero/a desarrollador
- El precio del hardware y de las credenciales de Google para acceder a APIs de bases de datos
- Los gastos de oficina

En este capítulo se analizarán estos aspectos para así llegar al presupuesto necesario para desarrollar el programa.

## 4.1. Sueldo del ingeniero/desarrollador

Para fijar el sueldo del desarrollador de la aplicación, se tienen en cuenta el nivel de estudios (asumiremos que se trata de un ingeniero titulado) y la región donde vive.

El precio por hora cobrado del ingeniero depende directamente de la zona geográfica donde vive. Dicho precio oscila mucho según este parámetro. Los siguientes datos son extraídos de la página arc.dev.

En Estados Unidos, el salario medio anual es de 96999$ (85990 €). Teniendo en cuenta una jornada laboral de 8h/día y trabajando 5 días a la semana, obtenemos un sueldo de 41,34€/h. El rango de salarios va de 38€/h hasta más de 84€/h.

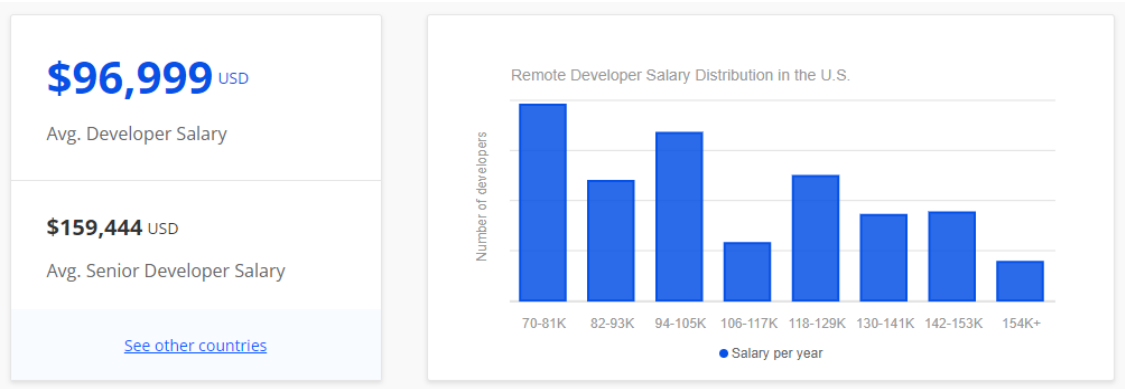



**Fig. 4.1.** Salario anual de un desarrollador en Estados Unidos. Fuente: arc.dev

Si, por el contrario, analizamos los salarios de un país menos desarrollado (la India) vemos que el salario anual medio es de 48918$ (55277 €), lo que equivale a 26,58€/h.

En nuestro caso, el ingeniero es de España.

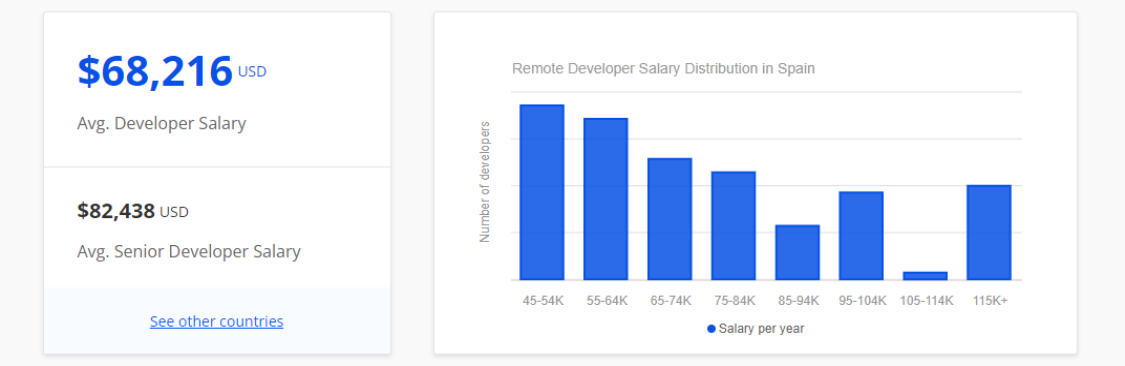



**Fig. 4.2.** Salario anual de un desarrollador en España. Fuente: arc.dev

El salario medio aquí es de 68216$ (77084 €). Por hora equivale a 37,06€/h. Los rangos de salario van de 25€/h hasta más de 62€/h.

De esta manera, estableceremos el salario del ingeniero conforme a la media de España, redondeando a 40€/h.

Con tal de calcular el precio total de contratar al ingeniero, se han contado las horas de trabajo hasta la publicación del aplicativo FlightTrack. En total han sido 644h, lo que equivale a 29.760€.

## 4.2. Precio del hardware y las credenciales

El precio del hardware requerido para desarrollar FlightTrack requiere tener en cuenta el uso de un ordenador con velocidad de procesador y capacidad de RAM suficientes para poder ejecutar y programar con VisualStudio.NET Framework 4.8. El precio de dicho ordenador es aproximadamente 800€.

Con tal de poder acceder a las bases de datos de Google, se ha hecho uso de Maps Elevation API, la API de Google que permite obtener la elevación del terreno (en metros) en un punto dado con sus coordenadas.

El servicio de APIs de Google funciona de la siguiente manera: un desarrollador puede solicitar una clave API para incorporarla en su programa. Dentro del programa, se hacen peticiones a una dirección que contiene esta clave, y Google almacena la cuenta de las peticiones que se han hecho.

Dependiendo de la API que se esté usando, el precio de las peticiones varía. Para Maps Elevation API, los precios los mostrados en la tabla 4.1.

**Tabla 4.1**. Precio al mes por petición de la API Google Maps Elevation. Fuente: console.cloud.google.com

| MONTHLY VOLUME RANGE (Price per REQUEST) | | |
|---|---|---|
| 0–100,000 | 100,001–500,000 | 500,000+ |
| 0.005 USD per each (5.00 USD per 1000) | 0.004 USD per each (4.00 USD per 1000) | Contact Sales for volume pricing |

Otro factor a tener en cuenta es que Google nos ofrece 200$ gratuitos al mes. De esta manera, a la hora de desarrollar un software, no se acaba cobrando nada, ya que se deberían exceder las 4000 peticiones en un mes para tener que pagar, lo que resulta muy improbable. La razón es que se pueden agrupar múltiples elevaciones en una petición. Por ello, cada vez que se ejecuta una ruta en FlightTrack se hace únicamente una petición a Maps Elevation API.

Este método está pensado para que cuando una aplicación se comercialice, se tenga que pagar a Google en función del uso que se hace de ella. Por ejemplo: si FlightTrack fuera utilizado por 1000 usuarios y dichos usuarios hicieran una media de 30 peticiones al mes (es decir, que consultaran 30 rutas diferentes), se debería pagar 1300$ (1500$ - 200$ gratuitos). En cambio, si fuera utilizado por 100 usuarios que hicieran el mismo número de peticiones, sería gratuito (150$ - 200$ gratuitos).

## 4.3. Gastos de oficina

Teniendo en cuenta que el ingeniero trabaja como freelance en casa, los gastos de oficina se hacen a partir de un tanto por ciento de los gastos del hogar.

Para el cálculo de gastos, se ha tenido en cuenta los datos de la media española por persona.

**Tabla 4.2**. Gastos medios mensuales por español. Fuentes: tarifaluzhora.es (ver [6]), shbarcelona.es (ver [7])

| | |
|---|---|
| Luz | 45,00 € |
| Agua | 10,00 € |
| Internet y móvil | 40,00 € |
| Alquiler (Barcelona) | 900,00 € |
| TOTAL | 995,00 € |

Como se ha definido una jornada de 8h/día, los gastos de oficina serán un 30% de los gastos totales mensuales.

Finalmente, como se ha tardado 4 meses en crear el software, el gasto final de oficina será: 1194€.

Sumando el precio de todos los factores enunciados, podemos concluir que el precio de desarrollar Flightrack es de 31.754€

# CAPÍTULO 5. CONCLUSIONES Y LÍNEAS DE CONTINUIDAD

El objetivo de este proyecto era crear un software de planificación de vuelos visuales que superara en grado de utilidad a los ya existentes.

Mediante un desarrollo de código con Visual Studio en lenguaje C# y un uso de las APIs de Google, se ha acabado obteniendo un programa de escritorio capaz de competir con los softwares comerciales del mercado.

Retomando la tabla 1.2 (ver apartado 1.1), podemos añadir el software resultante a la lista de programas (ver tabla 5.1).

**Tabla 5.1**. Grado de utilidad de los programas de planificación de vuelos VFR, incluyendo FlightTrack. Fuentes: elaboración propia

| | Skyvector | FileStar | VFR Flight | Plan G | Flight Planner 3000 | OIR Flight Planner | Little Navmap | Flylog | Aeroplus Aviation | ForeFlight | Easy VFR | RocketRoute | Skydemon | FlytTrack |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Contiene datos de España | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Mapa satélite | | | ✓ | | | | | | | | | | | ✓ |
| Selección de WP visuales | | ✓ | ✓ | | | ✓ | ✓ | | | | | | ✓ | ✓ |
| Cálculo peso y centrado | | ✓ | ✓ | | ✓ | | | | | | ✓ | | ✓ | ✓ |
| Cálculo combustible | | | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Creación y descarga de Navlog / Flightplan | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Datos meteorológicos en ruta | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| TAF/METAR/NOTAM | ✓ | | ✓ | ✓ | | ✓ | | | | ✓ | | | ✓ | ✓ |
| Perfil vertical de ruta | | ✓ | ✓ | | | | ✓ | | | | ✓ | | ✓ | ✓ |
| Función GPS para vuelo | | ✓ | | | | | | ✓ | ✓ | | ✓ | | ✓ | |
| Interfaz sencilla y visual | ✓ | | | ✓ | | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| Altitudes mínimas seguras | | ✓ | ✓ | | | | ✓ | | | | | | ✓ | ✓ |
| Pre-planning | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | ✓ | ✓ | ✓ |
| Simulador de vuelo | | | ✓ | ✓ | | | ✓ | | | | | | | |
| Importación kml | | | ✓ | ✓ | | | | | | | | | | ✓ |
| Logbook | | | | | | | ✓ | ✓ | | | | | | ✓ |
| Procedimientos despegue/aterrizaje | | | | ✓ | | | ✓ | | | ✓ | | | | |
| Información navaids | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | | ✓ | ✓ | | ✓ | ✓ |
| **TOTAL (%)** | **39** | **50** | **72** | **61** | **33** | **44** | **67** | **33** | **39** | **50** | **50** | **33** | **72** | **78** |

Con un 78% de grado de utilidad de acuerdo con los criterios enunciados en el apartado 1.1, FlightTrack se coloca a la cabeza de los programas de planificación de vuelo visual.

Contando con más tiempo, se podría seguir desarrollando la aplicación. Un aspecto ya contemplado podría ser acabar de hacer una base de datos de diferentes modelos de aeronaves, puesto que la aplicación ya fue programada para poder usar cualquier archivo con los datos del avión que siguiera el formato establecido. De esta manera, sólo se tendría que añadir una opción para el usuario de cambiar de aeronave para así cambiar de fichero.

Otra posible mejora, sería la implementación de los procedimientos de despegue y aterrizaje. Para ello se tendría que crear una gran base de datos de todos los aeródromos con las características de las pistas y una adecuación de las rutas en esos tramos.

Si se contara con un presupuesto, se podría implementar también el cálculo de velocidades teniendo en cuenta los vientos. Como ya se ha comentado anteriormente, el acceso a estas bases de datos es de pago.

Finalmente, también se podría plantear usar otro lenguaje para poder usar el software en móviles o tabletas, y así poder usarlo tanto en fase de planificación como en fase de navegación, haciendo uso del GPS y guiando durante el vuelo.

En definitiva, FlightTrack es un programa de interfaz sencilla y visual que acaba ofreciendo una experiencia muy completa al usuario, con opción a hacer futuras mejoras dando paso así a la posibilidad de unificar todo lo relativo a los vuelos visuales en una sola aplicación.

# BIBLIOGRAFÍA

[1] “Clasificación y descripción del espacio aéreo”. Enaire. https://aip.enaire.es/AIP/contenido_AIP/ENR/LE_ENR_1_4_en.pdf (acceso: Sep. 20, 2021).

[2] “Guía VFR ENAIRE”. Enaire. https://guiavfr.enaire.es/ (acceso: Sep. 20, 2021).

[3] “Tema 2. Los aviones y sus actuaciones”, apuntes de clase de EA. EETAC, UPC, 2019.

[4] “Espacio Aéreo de España”. RealSpain. http://vfrflightsimulator.blogspot.com/2010/01/espacio-aereo-de-espana.html (acceso: Sep. 20, 2021)

[5] “Aviation Weather API.” https://www.checkwxapi.com/ (acceso: Dic. 12, 2021).

[6] “Consumo medio de luz en España”. Tarifaluzhora. https://tarifaluzhora.es/info/consumo-mensual-luz (acceso: Oct. 4, 2021)

[7] “Vivienda: coste de gastos e Internet. Shbarcelona. https://www.shbarcelona.es/blog/es/vivienda-coste-de-gastos-e-internet/#:~:text=Agua%2C%20gas%20y%20electricidad,-Cuando%20se%20mude&text=A%20ello%20se%20le%20debe,%E2%82%AC%20a%20final%20del%20mes (acceso: Oct. 4, 2021)

[8] “ForeFlight” https://foreflight.com/europe/ (acceso: Sep. 10, 2021).

[9] “Aeroplus Aviation.” https://www.aeroplusaviation.com/flight-plan-app (acceso: Sep. 10, 2021).

[10] “FlyLog.” https://es.flylog.io/ (acceso: Sep. 10, 2021).

[11] “Little Navmap.” https://albar965.github.io/littlenavmap.html (acceso: Sep. 10, 2021).

[12] “OIR Flight Planner.” https://sourceforge.net/projects/oirfp/ (acceso: Sep. 10, 2021).

[13] “Flight Planner 3000.” https://www.champagnepcservices.com.au/ (acceso: Sep. 10, 2021).

[14] “FileStar VFR Planning software.” http://shop.pilotwarehouse.co.uk/product6580023catno2020023.html (acceso: Sep. 10, 2021).

[15] “Plan-G – VFR Flight Planing for flight simulators.” https://www.tasoftware.co.uk/ (acceso: Sep. 10, 2021).

[16] “VfrFlight.” http://vfrflight.org/en/index.html (acceso: Sep. 10, 2021).

[17] “Servicios de tránsito aéreo (ATS).” https://www.enaire.es/servicios/gestion_del_transito_aereo/servicios_de_transito_aereo (acceso: Nov. 30, 2021).

[18] “Lista de radioayudas para la navegación (AIP España).” https://aip.enaire.es/aip/contenido_AIP/GEN/LE_GEN_2_5_en.html (acceso: Nov. 24, 2021).

[19] “SimBrief.com - Virtual Flight Planning Solutions.” https://www.simbrief.com/home/ (acceso: Sep. 18, 2021).

[20] “SkyDemon, VFR Flight Planning Software and GPS Navigation.” https://www.skydemon.aero/ (acceso: Sep. 18, 2021).

[21] “OIR Flight Planner.” https://sourceforge.net/projects/oirfp/ (acceso: Sep. 18, 2021).

[22] “SkyVector.” https://skyvector.com/ (acceso: Sep. 18, 2021).

[23] “RocketRoute.” https://rocketroute.com/ (acceso: Sep. 10, 2021).

[24] “EasyVFR 4.” https://easyvfr4.aero/ (acceso: Sep. 10, 2021).

[25] “Pilot’s Operating Handbook. Skyhawk Cessna 172N”. Cessna. https://wayman.edu/files/Cessna-172N-POH.pdf (acceso: Sep. 20, 2021).

# ANEXO A. CÓDIGO DE FLIGHTTRACK

En este anexo se encuentra el código que forma parte de la aplicación FlightTrack separado en dos apartados, según el proyecto al que pertenece. En el subapartado A.1 está todo el código relativo a los formularios. En el A.2 se encuentran las diferentes clases dentro de la librería de clases.

Se ha mantenido el resaltado de sintaxis para facilitar la lectura. Todos los métodos contienen comentarios explicativos.

## A.1. Formularios

### A.1.1. Main

```
/*****************************************************************
*  FlightTrack
*  Desarrollo de software para planificación de vuelo visual
*  TFG 2021-2022
*  Por: Noelia Carrasquilla García
*  EETAC (UPC)
*  Última modificación: 01/01/2022
*****************************************************************/

using GMap.NET;
using GMap.NET.MapProviders;
using GMap.NET.WindowsForms;
using GMap.NET.WindowsForms.Markers;
using Lib;
using System;
using System.Collections.Generic;
using System.Device.Location;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using System.Windows.Forms.DataVisualization.Charting;
using System.IO;
using SharpKml.Engine;
using System.Reflection;


namespace FlightTrackApp
{
    public partial class Main : Form
    {
        // Atributos
        bool ADi = false;
        bool ADf = false;
        bool ADalt = false;
```

```
        bool crea = false;
        bool alt = false;
        bool nav = false;
        bool fechayhora = false;
        bool comb = false;
        bool pdf = false;
        bool coment = false;
        bool peso = false;
        bool metar_b = false;
        bool eventoWP;
        int contador;
        double combustibleAlt;
        double combustibleTaxi;
        double combustibleExtra;
        double combtotal;
        double tiempototal;
        string fecha;
        string hora;
        Avion avion = new Avion();
        AD aerodromo = new AD();
        EspacioAer espaer = new EspacioAer();
        Radioayuda radioayuda = new Radioayuda();
        List<double> dist = new List<double>();
        List<double> tCrucero = new List<double>();
        List<double> GPH = new List<double>();
        List<double> performance_asc = new List<double>();
        List<double> performance_desc = new List<double>();
        List<double> vectortiempotot = new List<double>();
        List<double> GS = new List<double>();
        List<double> comblist;
        List<PointLatLng> WP = new List<PointLatLng>();
        List<GeoCoordinate> coordenadasterreno = new List<GeoCoordinate>();
        List<GeoCoordinate> RecorridoAvion = new List<GeoCoordinate>();
        List<EspacioAer> EspacioAerSpain;
        List<AD> ADspain;
        List<AD> AeropuertosSeleccionados = new List<AD>();
        List<Radioayuda> navaids;
        List<List<EspacioAer>> espacioscontenedores = new
List<List<EspacioAer>>();
        GMapOverlay markerAD = new GMapOverlay("markers");
        GMapOverlay espacioaereo = new GMapOverlay("polygons");
        GMapOverlay markerWP = new GMapOverlay("markers");
        GMapOverlay rutas = new GMapOverlay("route");
        GMapOverlay markerNavaid = new GMapOverlay("markers");
        GMapOverlay Seleccion = new GMapOverlay();
        Series serie_elevacion;
        Series serie_altura;
```

```
// Métodos

public Main()
{
    InitializeComponent();
    Font = new Font("Segoe UI", 8);
    BackColor = Color.White;
    Color colorbotones = Color.FromArgb(246, 241, 209);
    borrar.BackColor = colorbotones;
    guardar.BackColor = colorbotones;
    descargaKML.BackColor = colorbotones;
    Zoomin.BackColor = colorbotones;
    Zoomout.BackColor = colorbotones;
    biblio.BackColor = colorbotones;
}


// Sets y gets
public void SetListaAD(List<AD> a)
{
    ADspain = a;
}
public void SetAvion(Avion a)
{
    avion = a;
}
public void SetEspAer(List<EspacioAer> a)
{
    EspacioAerSpain = a;
}
public void SetRadioayudas(List<Radioayuda> a)
{
    navaids = a;
}
public void AD(AD a)
{
    aerodromo = a;
}
public void SetCoordenadasterreno(List<GeoCoordinate> a)
{
    coordenadasterreno = a;
}
public List<List<EspacioAer>> GetEspContenedores()
{
    return espacioscontenedores;
}

// Cargado de form
private void Main_Load(object sender, EventArgs e)
{
```

```
            // Inicializar mapa:
            gMapControl1.DragButton = MouseButtons.Left;
            gMapControl1.CanDragMap = true;
            gMapControl1.ShowCenter = false; // elimina cruz central
            gMapControl1.MapProvider = GMapProviders.GoogleSatelliteMap;
            gMapControl1.Position = new PointLatLng(41.2974, 2.0833); //centrado
en LEBL
            gMapControl1.MinZoom = 1;
            gMapControl1.MaxZoom = 24;
            gMapControl1.Zoom = 7;
            gMapControl1.AutoScroll = false;

            //Chart
            chart1.Series.Clear();
            chart1.Titles.Add("Perfil de elevación");
            serie_elevacion = chart1.Series.Add("Elevación del terreno");
            serie_elevacion.ChartType = SeriesChartType.Area;
            serie_elevacion.Color = Color.Tan;
            serie_elevacion.LegendText = "Elevación del terreno";
            serie_elevacion.Font = new Font("Segoe UI", 8);

            serie_altura = chart1.Series.Add("Altitudes vuelo");
            serie_altura.ChartType = SeriesChartType.Line;
            serie_altura.Color = Color.FromArgb(71, 110, 125);
            serie_altura.BorderWidth = 2;
            serie_altura.LegendText = "Altitudes de vuelo";
            serie_altura.Font = new Font("Segoe UI", 8);
            chart1.Visible = false;

            // Cargar combobox de selección de mapa
            cambiarmapa.Items.Add("Google Mapa Satélite");
            cambiarmapa.Items.Add("Google Mapa Terreno");
            cambiarmapa.Items.Add("ArcGIS Mapa Topográfico");
            cambiarmapa.Items.Add("OpenCycle Mapa Topográfico");
            cambiarmapa.SelectedItem = cambiarmapa.Items[0];

            // Cargar avión
            SetAvion(avion.LeerFichero("cessna.txt"));

            // Estilos de boxes leyenda y visualizar
            leyendacolornav.BackColor = Color.FromArgb(204, 161, 102);
            leyendacoloraer.BackColor = Color.FromArgb(0, 225, 60);
            leyendacolorA.BackColor = Color.FromArgb(95, Color.DarkOrange);
            leyendacolorcde.BackColor = Color.FromArgb(95, Color.Beige);
            leyendacoloresrest.BackColor = Color.FromArgb(95, Color.Red);
        }


        //-MAPA-//
        // Cargado de mapa
```

```
        private void gMapControl1_Load(object sender, EventArgs e)
        {
            checkBoxAD.Checked = true;
            checkBoxEspAer.Checked = true;
            checkBoxNavaids.Checked = true;
            eventoWP = false;
            ADi = false;
            ADf = false;

            // OVERLAYS
            // Radioayudas
            SetRadioayudas(radioayuda.LeerRadioayudas("Radioayudas.kml"));
            for (int i = 0; i < navaids.Count(); i++)
            {
                PointLatLng p = new
PointLatLng(navaids[i].GetCoorNavaid().Latitude,
navaids[i].GetCoorNavaid().Longitude);
                string tipo = navaids[i].GettipoNavaid();
                GMapMarker marker = new GMarkerGoogle(p,
GMarkerGoogleType.brown_small);
                marker.ToolTipText = navaids[i].GettipoNavaid() + " " +
navaids[i].GetIDNavaid() + "\n " + navaids[i].GetfrecNavaid() + " MHz";
                markerNavaid.Markers.Add(marker);
            }
            gMapControl1.Overlays.Add(markerNavaid);

            // Aeródromos
            SetListaAD(aerodromo.LeerAD("AD.txt"));
            for (int i = 0; i < ADspain.Count(); i++)
            {
                PointLatLng p = new PointLatLng(ADspain[i].GetCoorAD().Longitude,
ADspain[i].GetCoorAD().Latitude);
                GMapMarker marker = new GMarkerGoogle(p,
GMarkerGoogleType.green);
                marker.ToolTipText = ADspain[i].GetNombreAD();
                marker.Tag = i;
                markerAD.Markers.Add(marker);
            }
            gMapControl1.Overlays.Add(markerAD);

            // Espacios aéreos
            SetEspAer(espaer.LeerEspacioAer("spain.sua.kml"));
            foreach (EspacioAer a in EspacioAerSpain)
            {
                if (a.GetCoordenadas() != null)
                {
                    GMapPolygon pol = espaer.GetPoligonoMapa(a);
                    espacioaereo.Polygons.Add(pol);
                    if (a.GetClase() == "CLASS X") // espacios restringidos
                    {
```

```
                    pol.Fill = new SolidBrush(Color.FromArgb(60, Color.Red));
                    pol.Stroke = new Pen(Color.Black, 1);
                }
                else if (a.GetClase() == "CLASS A")
                {
                    pol.Fill = new SolidBrush(Color.FromArgb(60,
Color.DarkOrange));
                    pol.Stroke = new Pen(Color.Black, 1);
                }
                else
                {
                    pol.Fill = new SolidBrush(Color.FromArgb(70, Color.Beige));
                    pol.Stroke = new Pen(Color.Bisque, 1);
                }
            }
        }
        gMapControl1.Overlays.Add(espacioaereo);
    }

    // Cambiar tipo de mapa
    private void cambiarmapa_SelectedIndexChanged(object sender,
EventArgs e)
    {
        if (cambiarmapa.SelectedIndex == 0)
            gMapControl1.MapProvider = GMapProviders.GoogleSatelliteMap;
        if (cambiarmapa.SelectedIndex == 1)
            gMapControl1.MapProvider = GMapProviders.GoogleTerrainMap;
        if (cambiarmapa.SelectedIndex == 2)
            gMapControl1.MapProvider =
GMapProviders.ArcGIS_World_Topo_Map;
        if (cambiarmapa.SelectedIndex == 3)
            gMapControl1.MapProvider =
GMapProviders.OpenCycleLandscapeMap;
    }

    // Zooms del mapa
    private void Zoomin_Clic(object sender, EventArgs e)
    {
        gMapControl1.Zoom += 1;
    }
    private void Zoomout_Clic(object sender, EventArgs e)
    {
        gMapControl1.Zoom -= 1;
    }

    // Doble clic para agregar marcadores WP si se ha clicado "Crear ruta" y
no hay una ruta ya creada
    private void gMapControl1_MouseDoubleClic(object sender,
MouseEventArgs e)
    {
```

```
        if (eventoWP && ADi && !ADf)
        {
            Point clic =
gMapControl1.PointToClient(System.Windows.Forms.Cursor.Position);
            PointLatLng p = gMapControl1.FromLocalToLatLng((int)clic.X,
(int)clic.Y);
            WP.Add(p);
            GMapMarker marker = new GMarkerGoogle(p,
GMarkerGoogleType.red_dot);
            marker.ToolTipText = Convert.ToString(contador);
            markerWP.Markers.Add(marker);
            contador++;
        }

    }

    // Con un clic se guardan los aeropuertos si se ha clicado a "Crear ruta" y
si crea el recorrido si el aeropuerto es el de destino
    private void gMapControl1_OnMarkerClic(GMapMarker item,
MouseEventArgs e)
    {
        if (markerAD.Markers.Contains(item))
        {
            if (eventoWP)
            {
                if (!ADi)
                {

AeropuertosSeleccionados.Add(ADspain[Convert.ToInt32(item.Tag)]);
                    WP.Add(item.Position);
                    ADi = true;
                    MessageBox.Show("Has seleccionado " +
AeropuertosSeleccionados[0].GetNombreAD() + " como aeródromo de origen.",
"Aeropuerto de origen", MessageBoxButtons.OK,
MessageBoxIcon.Information);
                }

                else if (ADi && !ADf)
                {

AeropuertosSeleccionados.Add(ADspain[Convert.ToInt32(item.Tag)]);
                    WP.Add(item.Position);
                    ADf = true;
                    MessageBox.Show("Has seleccionado " +
AeropuertosSeleccionados[1].GetNombreAD() + " como aeródromo de
destino.", "Aeropuerto de destino", MessageBoxButtons.OK,
MessageBoxIcon.Information);

                    // Se crea el recorrido
                    CrearRecorrido();
```

```
                    // si la altitud calculada excede los límites permitidos no se
representa en la chart, y se notifica al usuario
                    bool erroraltura = false;
                    foreach (GeoCoordinate a in RecorridoAvion)
                    {
                        if (a.Altitude > 19500)
                        {
                            erroraltura = true;
                            serie_altura.Points.Clear();
                        }
                    }
                    if (erroraltura)
                    {
                        // Se da la opción de ver los espacios aéreos atravesados en
la ruta y sus límites
                        DialogResult resultado = MessageBox.Show("La altitud de la
ruta excede los límites permitidos para vuelo visual." + "\n" + "¿Desea ver los
detalles de los espacios aéreos que se atraviesan durante la ruta?" + "\n" +
"(puede volver a consultarlos dentro del NavLog más adelante)", "Error de
altitud", MessageBoxButtons.YesNo, MessageBoxIcon.Warning);
                        if (resultado == DialogResult.Yes)
                            DetallesEspAer();
                    }

                }
            }
            // Si se selecciona el aeropuerto alternativo, se hace el cálculo del
combustible requerido
            if (ADi && ADf && ADalt)
            {

AeropuertosSeleccionados.Add(ADspain[Convert.ToInt32(item.Tag)]);
                RutaAer rutaalt = new RutaAer();
                List<double> performancealt = new List<double>();
                GeoCoordinate ADlast =
AeropuertosSeleccionados[AeropuertosSeleccionados.Count - 2].GetCoorAD();
                GeoCoordinate ADalter =
AeropuertosSeleccionados[AeropuertosSeleccionados.Count - 1].GetCoorAD();
                double dextra = ADlast.GetDistanceTo(ADalter);
                performancealt = rutaalt.CalculaPerformanceAlt(avion,
AeropuertosSeleccionados);
                double vel = performancealt[1];
                double gph = performancealt[2];
                double talt = dextra / (vel * 1852 / 3600);
                combustibleAlt = (gph * talt / 3600);
                combustibleExtra = gph * 45 / 60;
                combustibleTaxi = avion.GetCombustibleTaxi();
                alt = true;
```

```
                MessageBox.Show("Has seleecionado " +
Convert.ToString(AeropuertosSeleccionados.Last().GetNombreAD() + " como
aeropuerto alternativo"), "Aeropuerto Alternativo", MessageBoxButtons.OK,
MessageBoxIcon.Information);
            }
        }
    }


    //-VISUALIZACIÓN EN MAPA-//
    // Mostrar/Ocultar aeródromos
    private void checkBoxAD_CheckedChanged(object sender, EventArgs e)
    {
        if (!checkBoxAD.Checked)
            markerAD.IsVisibile = !Visible;
        else
            markerAD.IsVisibile = Visible;
    }

    // Mostrar/Ocultar espacios aéreos
    private void checkBoxEspAer_CheckedChanged(object sender, EventArgs
e)
    {
        if (!checkBoxEspAer.Checked)
            espacioaereo.IsVisibile = !Visible;
        else
            espacioaereo.IsVisibile = Visible;
    }

    // Mostrar/Ocultar radioayudas
    private void checkBoxNavaids_CheckedChanged(object sender,
EventArgs e)
    {
        if (!checkBoxNavaids.Checked)
            markerNavaid.IsVisibile = !Visible;
        else
            markerNavaid.IsVisibile = Visible;
    }

    //-TOOLSTRIP-//
    // Crear ruta
    private void addWP_Clic(object sender, EventArgs e)
    {
        if (RecorridoAvion.Count == 0)
        {
            eventoWP = true;
            contador = 1;
            gMapControl1.Overlays.Add(markerWP);
        }
        else
```

```
            MessageBox.Show("Antes borra la ruta actual", "",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }

        // Abrir formulario de NavLog
        private void verNavLog_Clic(object sender, EventArgs e)
        {
            if (RecorridoAvion.Count == 0)
                MessageBox.Show("Primero has de crear una ruta", "",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
            else
            {
                NavLog n = new NavLog();
                n.SetRecorrido(RecorridoAvion);
                n.SetDist(dist);
                n.SetGPH(GPH);
                n.SetVel(GS);
                n.Sett(vectortiempotot);
                n.SetPerformance_Asc(performance_asc);
                n.SetPerformance_Desc(performance_desc);
                n.SetCombExtra(combustibleAlt, combustibleExtra, combustibleTaxi);
                n.SetAvion(avion);
                if (fecha != null)
                    n.Setfecha(fecha);
                if (hora != null)
                    n.SetHora(hora);
                if (combtotal != 0)
                {
                    n.SetCombtot(combtotal);
                    n.SetComblist(comblist);
                }
                n.ShowDialog();
                combtotal = n.GetComb();
                comblist = n.GetComblist();
                hora = n.GetHora();
                fecha = n.GetFecha();
                nav = true;
                if (!fechayhora)
                    fechayhora = n.GetFyH();
                coment = n.GetComent();
                pdf = n.Getpdf();
                if (!comb)
                    comb = n.GetCombBool();
                Seleccion.Clear();
            }
        }

        // Botón para seleccionar aeropuerto alternativo
        private void SeleccionaAerAlt_Clic(object sender, EventArgs e)
        {
```

```
            if (AeropuertosSeleccionados.Count != 0)
            {
                ADi = true;
                ADf = true;
            }
            else
                MessageBox.Show("Primero añade la ruta", "",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
            ADalt = true;
        }

        // Abrir formulario de peso y balance
        private void weightandbalance_Clic(object sender, EventArgs e)
        {
            if (combtotal != 0)
            {
                PesoyBalance p = new PesoyBalance();
                p.SetAvion(avion);
                p.SetComb(combtotal);
                p.ShowDialog();
                peso = true;
            }
            else
                MessageBox.Show("Primero calcula el combustible en 'Ver NavLog'",
"", MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }

        // Abrir formulario de Biblioteca
        private void biblio_Clic_1(object sender, EventArgs e)
        {
            Biblioteca b = new Biblioteca();
            b.SetADspain(ADspain);
            b.ShowDialog();
            List<RutaAer> lista = b.GetLista();
            // Actualizar en caso de cambio
            if (b.GetCambio())
            {
                foreach (RutaAer a in lista)
                    if (a.GetPuntuacion() != 0)
                    {
                        RutaAer r = new RutaAer();
                        string fichero = a.Getnombrefichero();
                        File.Delete(fichero);
                        r.SetPuntuacion(a.GetPuntuacion());
                        r.GuardarRuta(fichero, a.GetWP(), a.GetADsel(), a.GetFecha(),
a.GetComb(), a.Gett(), a.GetPuntuacion(), a.GetComblist(), a.GetHora());
                    }
            }

            // Borrar archivo seleccionado
```

```
            if (b.GetListaBorrar() != null)
                foreach (RutaAer a in b.GetListaBorrar())
                    File.Delete(a.Getnombrefichero());

            // Cargar archivo seleccionado
            if (b.GetCarga())
            {
                BorrarRuta();
                RutaAer r = b.GetRuta();
                AeropuertosSeleccionados = b.GetAerSel();
                if (AeropuertosSeleccionados.Count == 3)
                    alt = true;
                fecha = r.GetFecha();
                hora = r.GetHora();
                if ((fecha != null) && (hora != null))
                    fechayhora = true;
                combtotal = r.GetComb();
                if (combtotal != 0)
                    comb = true;
                if (comb || fechayhora)
                    nav = true;
                comblist = r.GetComblist();
                WP = r.GetWPLatLon();
                CrearRecorrido();
                gMapControl1.Overlays.Add(markerWP);
                contador = 0;
                foreach (PointLatLng p in WP)
                {
                    if ((contador != 0) && (contador != WP.Count - 1))
                    {
                        GMapMarker marker = new GMarkerGoogle(p,
GMarkerGoogleType.red_dot);
                        marker.ToolTipText = Convert.ToString(contador);
                        markerWP.Markers.Add(marker);
                    }
                    contador++;
                }
                ADi = true;
                ADf = true;
                crea = true;
            }
        }

        // Abrir formulario de METAR
        private void metar_Clic(object sender, EventArgs e)
        {
            if (AeropuertosSeleccionados != null)
            {
                METAR m = new METAR();
                List<string> icaos = new List<string>();
```

```
            foreach (AD a in AeropuertosSeleccionados)
                icaos.Add(a.GetICAO());
            m.SetICAOS(icaos);
            m.SetADs(ADspain);
            m.ShowDialog();
            metar_b = true;
        }
        else
            MessageBox.Show("Antes crea una ruta", "",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
    }

    // Abrir formulario de Ayuda
    private void Ayuda_Clic(object sender, EventArgs e)
    {
        Ayuda a = new Ayuda();
        a.SetBool(crea, alt, nav, fechayhora, comb, pdf, coment, peso,
metar_b);
        a.ShowDialog();
    }


    //-BOTONES-//
    // Guardar ruta para posterior carga en FlightTrack
    private void guardar_Clic(object sender, EventArgs e)
    {
        RutaAer r = new RutaAer();
        SaveFileDialog nuevo = new SaveFileDialog();
        nuevo.Filter = "TXT|*.txt";
        nuevo.InitialDirectory =
Path.GetDirectoryName(Assembly.GetEntryAssembly().Location) + "\\Rutas";
        DialogResult dialog = nuevo.ShowDialog();
        if (dialog == DialogResult.OK)
        {
            string fichero = nuevo.FileName;
            int i = r.GuardarRuta(fichero, RecorridoAvion,
AeropuertosSeleccionados, fecha, combtotal, tiempototal, 0, comblist, hora);
            if (i == 0)
                MessageBox.Show("La ruta se ha guardado correctamente.", "",
MessageBoxButtons.OK, MessageBoxIcon.Information);

            else
                MessageBox.Show("No se ha podido guardar el fichero.", "",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    // Borrar ruta
    private void borrar_Clic(object sender, EventArgs e)
    {
```

```
            BorrarRuta();
        }

        // Descargar ruta en extensión .kml
        private void descargaKML_Clic(object sender, EventArgs e)
        {
            if (RecorridoAvion.Count != 0)
            {
                KML k = new KML(RecorridoAvion);
                SaveFileDialog nuevo = new SaveFileDialog();
                nuevo.Filter = "KML|*.kml";
                DialogResult dialog = nuevo.ShowDialog();
                if (dialog == DialogResult.OK)
                {
                    string fichero = nuevo.FileName;
                    KmlFile res = k.CrearKML(fichero);
                    if (res != null)
                        MessageBox.Show("El archivo KML se ha guardado
correctamente.", "", MessageBoxButtons.OK, MessageBoxIcon.Information);
                    else
                        MessageBox.Show("No se ha podido guardar el fichero KML.",
"", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
                else
                    MessageBox.Show("No se ha podido guardar el fichero KML.", "",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
            else
                MessageBox.Show("Antes crea una ruta", "",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
        }


        //-MÉTODOS INVOCABLES-//
        // Método invocable para crear la ruta con las "minimum safe altitudes"
        private void CrearRecorrido()
        {
            //Pintar ruta en mapa
            GMapRoute ruta = new GMapRoute(WP, "Ruta");
            ruta.Stroke.Width = 3;
            ruta.Stroke.Color = Color.LightBlue;
            rutas.Routes.Add(ruta);
            gMapControl1.Overlays.Add(rutas);
            gMapControl1.ZoomAndCenterRoute(ruta);

            try
            {
                // Creación de ruta de la clase "RutaAer" para añadir elevaciones y
las coordenadas de cada punto
                RutaAer miruta = new RutaAer(WP);
```

```
            string muestras = Convert.ToString(Convert.ToInt32(ruta.Distance));
//muestas cada km
            string resultado = miruta.PathElevation(muestras); // elevaciones del
terreno (Google Elevation API)
            SetCoordenadasterreno(miruta.GetCoordenadasTerreno(resultado));

            // Cálculo de distancias
            dist = miruta.CalculaDist(); // en metros

            // Creación de recorrido del vuelo. Comprueba los espacios aéreos y
los rumbos para determinar altitud
            RecorridoAvion =
miruta.GetWPconRumbo(miruta.GetWPconAltMin(), miruta.CalculaRumbo());
            espacioscontenedores =
miruta.GetEspaciosAerContenedores(coordenadasterreno, EspacioAerSpain);
            RecorridoAvion = miruta.GetWPconEspAer(espacioscontenedores);

            // Cálculo de velocidades y consumo de combustible a partir de las
performances del avión
            List<List<double>> performances_total =
miruta.CalcularPerformances(avion, RecorridoAvion,
AeropuertosSeleccionados); ;
            performance_asc = performances_total[0]; //0: alt, 1: t, 2: GPH, 3: d
            performances_total.Remove(performances_total[0]);
            performance_desc = performances_total.Last(); //0: t, 1: GPH, 2: dist
            performances_total.Remove(performances_total.Last());
            // Las que no son primera ni última (crucero) => 0: alt, 1: GS, 2: GPH
            foreach (List<double> l in performances_total)
            {
                GS.Add(l[1]);
                GPH.Add(l[2]);
            }

            // Cálculo de tiempos
            tCrucero = miruta.CalculaTiempo(dist, GS);
            double tiempoAscenso = performance_asc[1] * 60;
            vectortiempotot.Add(tiempoAscenso);
            foreach (double item in tCrucero)
                vectortiempotot.Add(item);
            double tiempoDescenso = performance_desc[1] * 60;
            vectortiempotot.Add(tiempoDescenso);
            tiempototal = vectortiempotot.Sum(); //en segundos


            // Representación de datos en chart
            int i = 0;
            int j = 0;
            double d = dist[j] / 1852;
        List<GeoCoordinate> elevaciones =
miruta.GetCoordenadasTerreno(resultado);
```

```
            while (i < elevaciones.Count - 1)
            {
                double posicion = (i + 1) / 1.852;
                if (d < posicion)
                {
                    if (j == dist.Count - 1)
                        break;
                    else
                    {
                        d += dist[j + 1] / 1852;
                        j++;
                    }
                }
                serie_elevacion.Points.AddXY(posicion, elevaciones[i].Altitude *
3.28084); //X: distancia en NM, Y: altitud del terreno en ft
                serie_altura.Points.AddXY(posicion, RecorridoAvion[j].Altitude);
                i++;
            }
        }
        catch
        {
            MessageBox.Show("Ha habido un error con la descarga de
elevaciones del terreno", "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
        chart1.Visible = true;
        crea = true;
    }

    // Método invocable para acceder a formulario con detalles sobre los
espacios aéreos atravesados
    private void DetallesEspAer()
    {
        EspAerErrorAltura e = new EspAerErrorAltura();
        e.SetEsp(espacioscontenedores);
        e.ShowDialog();
        Seleccion.Clear();
    }

    // Método invocable para colorear el espacio aéreo seleccionado desde el
form EspAerErrorAltura
    public void MostrarSeleccion(EspacioAer e)
    {
        GMapPolygon pol = espaer.GetPoligonoMapa(e);
        if (!Seleccion.Polygons.Contains(pol))
        {
            gMapControl1.Overlays.Remove(Seleccion);
            Seleccion.Clear();
            pol.Fill = new SolidBrush(Color.FromArgb(40, Color.Yellow));
            pol.Stroke = new Pen(Color.Bisque, 1);
```

```
            Seleccion.Polygons.Add(pol);
            gMapControl1.Overlays.Add(Seleccion);
            gMapControl1.Zoom += 1;
            gMapControl1.Zoom -= 1;
        }
    }

    // Método invocable para borrar ruta
    private void BorrarRuta()
    {
        eventoWP = false;
        ADi = false;
        ADf = false;
        ADalt = false;
        crea = false;
        alt = false;
        nav = false;
        fechayhora = false;
        comb = false;
        pdf = false;
        coment = false;
        peso = false;
        metar_b = false;
        combustibleAlt = 0;
        combustibleTaxi = 0;
        combustibleExtra = 0;
        combtotal = 0;
        contador = 0;
        tiempototal = 0;
        fecha = "";
        hora = "";
        AeropuertosSeleccionados = new List<AD>();
        WP = new List<PointLatLng>();
        RecorridoAvion = new List<GeoCoordinate>();
        comblist = new List<double>();
        dist = new List<double>();
        tCrucero = new List<double>();
        GPH = new List<double>();
        performance_asc = new List<double>();
        performance_desc = new List<double>();
        vectortiempotot = new List<double>();
        GS = new List<double>();
        coordenadasterreno = new List<GeoCoordinate>();
        markerWP.Clear();
        rutas.Clear();
        serie_altura.Points.Clear();
        serie_elevacion.Points.Clear();
        chart1.Visible = false;
    }
}
```

```
}
```

### A.1.2. Inicio

```
using System;
using System.Windows.Forms;

namespace FlightTrackApp
{
   public partial class Inicio : Form
   {
      public Inicio()
      {
         InitializeComponent();
      }

      private void Inicio_Load(object sender, EventArgs e)
      {
         timer1.Start();
         timer1.Interval = 3500;
      }

      private void timer1_Tick(object sender, EventArgs e)
      {
         Close();
      }
   }
}
```

### A.1.3. NavLog

```
using Lib;
using System;
using System.Collections.Generic;
using System.Device.Location;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using System.IO;
using iTextSharp.text.pdf;
using iTextSharp.text;

namespace FlightTrackApp
{
   public partial class NavLog : Form
```

```
{
    bool comb = false;
    bool coment = false;
    bool fechayhora = false;
    bool pdf = false;
    double totalCombustible;
    double alter;
    double taxi;
    double extra;
    string fecha;
    string hora;
    string[] comentarios;
    Avion avion;
    List<GeoCoordinate> RecorridoAvion = new List<GeoCoordinate>();
    List<double> dist = new List<double>();
    List<double> GPH = new List<double>();
    List<double> t = new List<double>();
    List<double> vel = new List<double>();
    List<double> performance_asc = new List<double>();
    List<double> performance_desc = new List<double>();
    List<double> combustiblelist = new List<double>();

    public NavLog()
    {
        InitializeComponent();
        Font = new System.Drawing.Font("Segoe UI", 8);
        BackColor = Color.White;
        Color colorbotones = Color.FromArgb(246, 241, 209);
        modifica.BackColor = colorbotones;
        CalculaComb.BackColor = colorbotones;
        guionvuelo.BackColor = colorbotones;
        NavLogpdf.BackColor = colorbotones;
        Espaercont.BackColor = colorbotones;
        fondo.BackColor = Color.FromArgb(11,159,151);

    }

    // Sets y Gets
    public void SetAvion(Avion a)
    {
        avion = a;
    }

    public void SetRecorrido(List<GeoCoordinate> a)
    {
        RecorridoAvion = a;
    }

    public void SetDist(List<double> a)
    {
```

```
        dist = a;
    }

    public void SetGPH(List<double> a)
    {
        GPH = a;
    }

    public void Sett(List<double> a)
    {
        t = a;
    }

    public void SetVel(List<double> a)
    {
        vel = a;
    }

    public void Setfecha(string a)
    {
        fecha = a;
    }

    public void SetHora(string a)
    {
        hora = a;
    }

    public void SetCombtot(double c)
    {
        totalCombustible = c;
    }

    public void SetComblist(List<double> c)
    {
        combustiblelist = c;
    }

    public void SetPerformance_Asc(List<double> a)
    {
        performance_asc = a;
    }

    public void SetPerformance_Desc(List<double> a)
    {
        performance_desc = a;
    }

    public void SetCombExtra(double alter, double extra, double taxi)
    {
```

```
        this.alter = alter;
        this.extra = extra;
        this.taxi = taxi;
    }

    public double GetComb()
    {
        return totalCombustible;
    }

    public List<double> GetComblist()
    {
        return combustiblelist;
    }

    public string GetFecha()
    {
        return fecha;
    }

    public string GetHora()
    {
        return hora;
    }

    public bool GetFyH()
    {
        return fechayhora;
    }

    public bool GetCombBool()
    {
        return comb;
    }

    public bool GetComent()
    {
        return coment;
    }

    public bool Getpdf()
    {
        return pdf;
    }

    // Carga de NavLog
    private void NavLog_Load(object sender, EventArgs e)
    {
        if (fecha != null)
            fechalabel.Text = "Fecha: " + fecha.ToString();
```

```
        if (hora != null)
            horalabel.Text = "Hora: " + hora.ToString();
        comentarios = new string[RecorridoAvion.Count - 1];

        // Añadir columnas
        dataGridView.Columns.Add("wp", "WP");
        dataGridView.Columns.Add("rumbo", "Course (º)");
        dataGridView.Columns.Add("alt", "Altitude (ft)");
        dataGridView.Columns.Add("dist", "Distance (NM)");
        dataGridView.Columns.Add("vel", "KTAS");
        dataGridView.Columns.Add("Estimated time Enroute", "ETE (min)");
        dataGridView.Columns.Add("fuel", "GPH");
        dataGridView.Columns.Add("comentario", "Comentario");
        int i = 0;
        while (i < RecorridoAvion.Count() - 1)
        {
            string wp = Convert.ToString(i) + " - " + Convert.ToString(i + 1);
            string r =
Convert.ToString(Decimal.Round(Convert.ToDecimal(RecorridoAvion[i].Course)
, 2));
            string alt = Convert.ToString(RecorridoAvion[i].Altitude);
            string d =
Convert.ToString(Decimal.Round(Convert.ToDecimal((dist[i] / 1852)), 2));
//pasado de m a NM y redondeado
            string v = Convert.ToString(vel[i]);
            string time =
Convert.ToString(Decimal.Round(Convert.ToDecimal((t[i] / 60)), 2));
            string gph = Convert.ToString(GPH[i]);
            if ((RecorridoAvion[i].Altitude > 19500) || (RecorridoAvion[i].Altitude >
avion.GetTecho()))
            {
                string causa = "";
                alt = "Ninguna altitud posible";
                if (RecorridoAvion[i].Altitude > 19500)
                    causa = "el máximo permitido para vuelo visual";
                if ((RecorridoAvion[i].Altitude > avion.GetTecho()) &&
(RecorridoAvion[i].Altitude < 19500))
                    causa = "el techo de vuelo de la aeronave";
                time = "Error en altura";
                gph = "Error en altura";
                MessageBox.Show("La aeronave no puede pasar por la zona " +
wp + "porque la altitud excede " + causa, "", MessageBoxButtons.OK,
MessageBoxIcon.Warning);
            }

            dataGridView.Rows.Add(wp, r, alt, d, v, time, gph);
            i++;
        }

        if (combustiblelist.Count != 0)
```

```
        {
            dataGridView.Columns.Remove("fuel");
            dataGridView.Columns.Add("fuel", "Fuel (gal)");
            dataGridView.Columns[dataGridView.Columns.Count -
1].DisplayIndex = dataGridView.Columns.Count - 2;
            i = 0;
            while (i < RecorridoAvion.Count() - 1)
            {
                dataGridView.Rows[i].Cells["fuel"].Value =
Decimal.Round(Convert.ToDecimal(combustiblelist[i]), 2);
                i++;
            }
            comblab.Text = "Combustible: " +
Decimal.Round(Convert.ToDecimal(totalCombustible), 2).ToString() + " gal";
            CalculaComb.Visible = false;

        }

        if (hora != "" && hora != null)
            SetETA(hora, dataGridView);
    }

    // Establecer/modificar fecha y hora
    private void modifica_Clic(object sender, EventArgs e)
    {
        FechayHora f = new FechayHora();
        f.ShowDialog();
        hora = f.GetHora();
        fecha = f.GetFecha();
        if (f.GetFecha() != null)
            fechalabel.Text = "Fecha: " + fecha.ToString();
        if ((f.GetHora() != "") && f.GetFormatoH())
        {
            horalabel.Text = "Hora: " + hora.ToString() + " (ZULU)";
            SetETA(hora, dataGridView);
            fechayhora = true;
        }
    }

    // Calcular combustible
    private void CalculaComb_Clic(object sender, EventArgs e)
    {
        List<double> combustible = new List<double>();
        int i = 0;
        int j = 1;
        while (i < GPH.Count)
        {
            combustible.Add((GPH[i] / 60) * (t[j] / 60));
            i++;
            j++;
```

```
        }

        double combustible_asc = performance_asc[2] * performance_asc[1] /
3600;
        double combustible_desc = performance_desc[2] *
performance_desc[1] / 3600;
        totalCombustible = combustible.Sum() + alter + taxi + extra +
combustible_asc + combustible_desc;
        string total_redondeado =
Convert.ToString(Decimal.Round(Convert.ToDecimal(totalCombustible), 2));
        if (totalCombustible > avion.GetCapacidad())
        {
            MessageBox.Show("El combustible requerido excede el límite de
combustible del avión","", MessageBoxButtons.OK, MessageBoxIcon.Warning);
            comblab.Text = "Combustible: " + total_redondeado + " gal";
            comblab.ForeColor = System.Drawing.Color.Red;
        }

        // Si el combustible no supera las capacidades del avión, se pregunta si
se desea ver el desglose (abre DesgloseCombustible)
        if ((totalCombustible <= avion.GetCapacidad()) && (alter != 0))
        {
            string mensaje = "El combustible total son " +
total_redondeado.ToString() + " gal. ¿Quiere ver el desglose de esa cantidad?";
            DialogResult Combustible = MessageBox.Show(mensaje, "Desglose
de combustible", MessageBoxButtons.YesNo, MessageBoxIcon.Information);
            if (Combustible == DialogResult.Yes)
            {
                DesgloseCombustible d = new DesgloseCombustible();
                d.SetCombustibles(combustible.Sum(), combustible_asc,
combustible_desc, alter, taxi, extra, totalCombustible);
                d.ShowDialog();
            }
            comblab.Text = "Combustible: " + total_redondeado + " gal";
            combustible[0] = combustible_asc + combustible[0];
            combustible[combustible.Count - 1] = combustible_desc +
combustible[combustible.Count - 1];
            dataGridView.Columns.Remove("fuel");
            dataGridView.Columns.Add("fuel", "Fuel");
            dataGridView.Columns[dataGridView.Columns.Count -
1].DisplayIndex = dataGridView.Columns.Count - 2;
            i = 0;
            while (i < RecorridoAvion.Count() - 1)
            {
                dataGridView.Rows[i].Cells["fuel"].Value =
Decimal.Round(Convert.ToDecimal(combustible[i]), 2);
                i++;
            }
            combustiblelist = combustible;
        }
```

```
            else
                MessageBox.Show("Antes selecciona un aeropuerto alternativo","",
MessageBoxButtons.OK, MessageBoxIcon.Warning);
            comb = true;

        }

        // Agregar comentario a tramo - se abre formulario Comentario
        private void dataGridView_CellClic(object sender,
DataGridViewCellEventArgs e)
        {
            if (dataGridView.CurrentCell.OwningColumn.Name == "comentario")
            {
                Comentario c = new Comentario();
                string comentarioactual = (string)dataGridView.CurrentCell.Value;
                if (comentarioactual == null)
                    comentarioactual = "";
                if (comentarioactual != null)
                    comentarioactual =
comentarios[dataGridView.CurrentCell.RowIndex];
                c.SetComentario(comentarioactual);
                c.ShowDialog();
                string comentariocargado = c.GetComentario();
                if (comentariocargado != "")
                {
                    comentarios[dataGridView.CurrentCell.RowIndex] =
c.GetComentario();
                    dataGridView.CurrentCell.Value = "Contiene comentario";
                }
            }
        }

        // Crear guion de vuelo - Guarda en pdf un resumen de los tramos con los
comentarios escritos
        private void guionvuelo_Clic(object sender, EventArgs e)
        {
            SaveFileDialog nuevo = new SaveFileDialog();
            nuevo.Filter = "PDF|*.pdf";
            nuevo.FileName = "Guion de vuelo";
            DialogResult dialog = nuevo.ShowDialog();
            if (dialog == DialogResult.OK)
            {
                try
                {
                    using (FileStream stream = new FileStream(nuevo.FileName,
FileMode.Create))
                    {
                        Document pdfDoc = new Document(PageSize.A4, 10f, 20f, 20f,
10f);
```

```
                    PdfWriter.GetInstance(pdfDoc, stream);
                    iTextSharp.text.Font titulotramo =
FontFactory.GetFont(FontFactory.HELVETICA_BOLD, 12);
                    titulotramo.Color = BaseColor.BLUE;
                    iTextSharp.text.Font texto =
FontFactory.GetFont(FontFactory.HELVETICA, 11);
                    pdfDoc.Open();
                    pdfDoc.Add(new Paragraph("Guion de vuelo",
FontFactory.GetFont(FontFactory.HELVETICA_BOLD, 15,
iTextSharp.text.Font.UNDERLINE)));
                    pdfDoc.Add(new Paragraph(Environment.NewLine));
                    if (fecha != null)
                        pdfDoc.Add(new Paragraph("Fecha: " + fecha,
FontFactory.GetFont(FontFactory.HELVETICA, 10)));
                    pdfDoc.Add(new Paragraph(Environment.NewLine));
                    int i = 0;
                    while (i < comentarios.Length)
                    {
                        string dstr =
Convert.ToString(Decimal.Round(Convert.ToDecimal((dist[i] / 1852)), 2));
                        string tstr =
Convert.ToString(Decimal.Round(Convert.ToDecimal((t[i] / 60)), 2));
                        string altstr = Convert.ToString(RecorridoAvion[i].Altitude);
                        string rstr =
Convert.ToString(Decimal.Round(Convert.ToDecimal(RecorridoAvion[i].Course)
, 2));
                        string tit = ("Tramo " + Convert.ToString(i) + "-" +
Convert.ToString(i + 1) + " - " + dstr + " NM  - " + tstr + " min  - " + altstr + " ft  - "
+ rstr + "º");
                        pdfDoc.Add(new Paragraph(tit, titulotramo));
                        string com = comentarios[i];
                        if (com == "")
                            com = "(Sin comentarios)";
                        pdfDoc.Add(new Paragraph(com, texto));
                        pdfDoc.Add(new Paragraph(Environment.NewLine));
                        i++;
                    }
                    pdfDoc.Close();
                    stream.Close();
                }
                MessageBox.Show("Guardado correctamente","",
MessageBoxButtons.OK, MessageBoxIcon.Information);
                coment = true;
            }
            catch
            {
                MessageBox.Show("Error en el guardado del fichero","",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
```

```
        }

        // Descargar NavLog en pdf
        private void NavLogpdf_Clic(object sender, EventArgs e)
        {
            // Se crea un datagridview con las columnas a importar en el pdf
            try
            {
                DataGridView d = new DataGridView();
                int i = 0;
                d.Columns.Add("wp", "WP");
                d.Columns.Add("rumbo", "Course (º)");
                d.Columns.Add("alt", "Altitude (ft)");
                d.Columns.Add("dist", "Distance (NM)");
                d.Columns.Add("Estimated time Enroute", "ETE (min)");
                if (hora != null)
                    d.Columns.Add("Estimated time Arrival", "ETA (min)");
                if (totalCombustible != 0)
                    d.Columns.Add("fuel", "Fuel (gal)");
                while (i < RecorridoAvion.Count - 1)
                {
                    d.Rows.Add(dataGridView.Rows[i].Cells["wp"].Value);
                    d.Rows.Add(dataGridView.Rows[i].Cells["rumbo"].Value);
                    d.Rows.Add(dataGridView.Rows[i].Cells["alt"].Value);
                    d.Rows.Add(dataGridView.Rows[i].Cells["dist"].Value);
                    d.Rows.Add(dataGridView.Rows[i].Cells["Estimated time
Enroute"].Value);
                    if (hora != null)
                        d.Rows.Add(dataGridView.Rows[i].Cells["Estimated time
Arrival"].Value);
                    if (totalCombustible != 0)
                        d.Rows.Add(dataGridView.Rows[i].Cells["fuel"].Value);
                    i++;
                }

                SaveFileDialog sfd = new SaveFileDialog();
                sfd.Filter = "PDF (*.pdf)|*.pdf";
                sfd.FileName = "Output.pdf";
                if (sfd.ShowDialog() == DialogResult.OK)
                {
                    PdfPTable tabla = new PdfPTable(d.Columns.Count);
                    tabla.DefaultCell.Padding = 3;
                    tabla.WidthPercentage = 100;
                    foreach (DataGridViewColumn c in d.Columns)
                    {
                        if (c.Name != "Comentario")
                        {
                            PdfPCell celda = new PdfPCell(new
iTextSharp.text.Phrase(c.HeaderText));
                            tabla.AddCell(celda);
```

```
                }
            }

            foreach (DataGridViewRow r in d.Rows)
            {
                foreach (DataGridViewCell celda in r.Cells)
                {
                    if (celda.Value != null)
                        tabla.AddCell(celda.Value.ToString());
                }
            }
            using (FileStream stream = new FileStream(sfd.FileName,
FileMode.Create))
            {
                Document pdfDoc = new Document(PageSize.A4, 10f, 20f, 20f,
10f);
                PdfWriter.GetInstance(pdfDoc, stream);

                pdfDoc.Open();
                pdfDoc.Add(new Paragraph("NavLog",
FontFactory.GetFont(FontFactory.HELVETICA_BOLD, 14,
iTextSharp.text.Font.UNDERLINE)));
                pdfDoc.Add(new Paragraph(Environment.NewLine));
                if (fecha != null)
                    pdfDoc.Add(new Paragraph("Fecha: " + fecha,
FontFactory.GetFont(FontFactory.HELVETICA, 11)));
                if (hora != null)
                    pdfDoc.Add(new Paragraph("Hora de salida (ZULU): " + hora,
FontFactory.GetFont(FontFactory.HELVETICA, 11)));
                if (totalCombustible != 0)
                    pdfDoc.Add(new Paragraph("Combustible total (gal): " +
Decimal.Round(Convert.ToDecimal(totalCombustible), 2),
FontFactory.GetFont(FontFactory.HELVETICA, 11)));
                pdfDoc.Add(new Paragraph(Environment.NewLine));
                pdfDoc.Add(new Paragraph(Environment.NewLine));
                pdfDoc.Add(tabla);
                pdfDoc.Close();
                stream.Close();
            }
            MessageBox.Show("Guardado correctamente", "",
MessageBoxButtons.OK, MessageBoxIcon.Information);
            pdf = true;
        }
    }
    catch
    {
        MessageBox.Show("Error. No se ha podido guardar", "",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }
```

```
        }

        // Ver los espacios aéreos cruzados - Abre formulario EspAerErrorAltura
        private void Espaercont_Clic(object sender, EventArgs e)
        {
            EspAerErrorAltura es = new EspAerErrorAltura();
            Main master = (Main)Application.OpenForms["Main"];
            es.SetEsp(master.GetEspContenedores());
            es.ShowDialog();
        }

        // Método invocable para añadir columna ETA en datagridview una vez
añadida la hora
        private void SetETA(string hora, DataGridView dataGridView)
        {
            string[] zulu = hora.Split(':');
            decimal h = Convert.ToDecimal(zulu[0]);
            if (h == 24)
                h = 0;
            double min = Convert.ToDouble(zulu[1]);
            int n = 8;
            if (dataGridView.Columns.Count == n)
            {
                n = dataGridView.Columns.Add("Estimated time Arrival", "ETA
(ZULU)");
                dataGridView.Columns[dataGridView.Columns.Count -
1].DisplayIndex = 6;
            }

            int i = 0;
            while (i < RecorridoAvion.Count() - 1)
            {
                min += (t[i] / 60);
                if (min > 60)
                {
                    min -= 60;
                    h++;
                    if (h > 23)
                        h = 0;
                }
                string sh = Convert.ToString(h);
                string smin = Convert.ToString(Math.Truncate(min));
                if (sh.Count(char.IsDigit) < 2)
                    sh = "0" + sh;
                if (smin.Count(char.IsDigit) < 2)
                    smin = "0" + smin;
                string eta = (Convert.ToString(sh + ":" + smin));
                dataGridView.Rows[i].Cells[n].Value = eta;
                i++;
            }
```

```
        }
    }
}
```

### A.1.4. Espacios Aéreos

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using Lib;

namespace FlightTrackApp
{
    public partial class EspAerErrorAltura : Form
    {
        int i;
        EspacioAer sel;
        List<EspacioAer> lista = new List<EspacioAer>();
        List<List<EspacioAer>> e = new List<List<EspacioAer>>();

        public EspAerErrorAltura()
        {
            InitializeComponent();
            BackColor = Color.White;
            fondo.BackColor = Color.FromArgb(71,110,125);
        }

        // Sets y Gets
        public void SetEsp(List<List<EspacioAer>> e)
        {
            this.e = e;
            foreach (List<EspacioAer> l in e)
                foreach (EspacioAer es in l)
                    if (!lista.Contains(es))
                        lista.Add(es);
        }

        public void SetEspSeleccionado(int i)
        {
            try
            {
                sel = lista[i];
            }
            catch { }

        }
        public EspacioAer GetEspSel()
        {
```

```
            return sel;
        }

        // Cargar espacios aéreos
        private void EspAerErrorAltura_Load(object sender, EventArgs e)
        {
            dataGridView1.Columns.Add("nombre", "Nombre");
            dataGridView1.Columns.Add("Clase", "Clase");
            dataGridView1.Columns.Add("liminf", "Límite Inferior");
            dataGridView1.Columns.Add("limsup", "Límite Superior");

            foreach (EspacioAer es in lista)
            {
                string n = es.GetNombre();
                string c = es.GetClase();
                string limabajo = Convert.ToString(es.GetLimAbajo());
                string tipoabajo = es.GetTipoAbajo();
                string liminf = limabajo + " " + tipoabajo;
                if (limabajo == "0")
                    liminf = tipoabajo;
                if (tipoabajo == "FL")
                {
                    limabajo = Convert.ToString(es.GetLimAbajo() / 100);
                    liminf = tipoabajo + " " + limabajo;
                }
                string limarriba = Convert.ToString(es.GetLimArriba());
                string tipoarriba = es.GetTipoArriba();
                string limsup = limarriba + " " + tipoarriba;
                if (tipoarriba == "FL")
                {
                    limarriba = Convert.ToString(es.GetLimArriba() / 100);
                    limsup = tipoarriba + " " + limarriba;
                }
                dataGridView1.Rows.Add(n, c, liminf, limsup);
            }
        }

        // Haciendo clic, se muestra en el mapa el espacio aéreo en otro color
        private void dataGridView1_CellClic(object sender,
DataGridViewCellEventArgs e)
        {
            i = dataGridView1.CurrentCell.RowIndex;
            SetEspSeleccionado(i);
            Main master = (Main)Application.OpenForms["Main"];
            master.MostrarSeleccion(sel);
        }
    }
}
```

### A.1.5. Fecha y hora

```
using System;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;

namespace FlightTrackApp
{
   public partial class FechayHora : Form
   {

      string fechacalendario;
      string fechaguardada = "";
      string hora;
      string horaguardada = "";
      bool formato = true;

      public FechayHora()
      {
         InitializeComponent();
         Font = new System.Drawing.Font("Segoe UI", 8);
         BackColor = Color.White;
         Color colorbotones = Color.FromArgb(246, 241, 209);
         button1.BackColor = colorbotones;
         fondo.BackColor = Color.FromArgb(71, 110, 125);
      }

      // Gets
      public string GetHora()
      {
         return horaguardada;
      }

      public string GetFecha()
      {
         return fechaguardada;
      }

      public bool GetFormatoH()
      {
         return formato;
      }

      // Cargar formulario
      private void FechayHora_Load(object sender, EventArgs e)
      {
         comboBox1.Items.Add("ZULU");
         comboBox1.Items.Add("Hora local (península)");
         comboBox1.Items.Add("Hora local (Canarias)");
```

```
            comboBox1.SelectedIndex = 0;
        }

        // Seleccionar fecha
        private void monthCalendar1_DateSelected(object sender,
DateRangeEventArgs e)
        {
            fechacalendario = e.Start.ToShortDateString();
        }

        // Guarda la fecha y la hora, detecta errores de formato y cierra el form
        private void button1_Clic(object sender, EventArgs e)
        {
            hora = textBox1.Text;
            if (hora == "hh:mm")
                hora = "";
            formato = true;
            bool mensaje = false;
            try
            {
                string[] comprobacion = hora.Split(':');
                if (comprobacion.Count() != 2)
                    formato = false;
                if (Convert.ToDouble(comprobacion.First()) >= 24)
                    formato = false;
                if (Convert.ToDouble(comprobacion.Last()) >= 60)
                    formato = false;
                if (Convert.ToDouble(comprobacion[1].Count()) != 2)
                    formato = false;
            }
            catch
            {
                MessageBox.Show("Error en el formato de la hora", "",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                mensaje = true;
            }
            if (formato)
            {
                string[] zulu;
                decimal h;
                if (String.Equals(comboBox1.SelectedItem.ToString(), "Hora local
(península)"))
                {
                    zulu = hora.Split(':');
                    h = Convert.ToDecimal(zulu[0]) + 1;
                    if (h == 24)
                        h = 0;
                    string hs = Convert.ToString(h);
                    if (h == 0)
                        hs = "00";
```

```
                hora = hs + ":" + zulu[1];
            }
            zulu = hora.Split(':');
            fechaguardada = fechacalendario;
            horaguardada = hora;
            this.Close();
        }
        if (!formato && !mensaje)
            MessageBox.Show("Error en el formato de la hora", "",
MessageBoxButtons.OK, MessageBoxIcon.Error);
    }

  }
}
```

### A.1.6. Desglose de combustible

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace FlightTrackApp
{
  public partial class DesgloseCombustible : Form
  {
    double combustibleCrucero;
    double combustibleAsc;
    double combustibleDesc;
    double combustibleAeropuertoAlt;
    double combustibleTaxi;
    double combustibleExtra;
    double total;

    public DesgloseCombustible()
    {
      InitializeComponent();
      Font = new Font("Segoe UI", 8);
      BackColor = Color.White;
    }

    // Set combustibles
    public void SetCombustibles(double crucero, double asc, double desc,
double alt, double taxi, double extra, double total)
    {
      combustibleCrucero = crucero;
      combustibleAsc = asc;
      combustibleDesc = desc;
      combustibleAeropuertoAlt = alt;
      combustibleTaxi = taxi;
```

```
            combustibleExtra = extra;
            this.total = total;
        }

        // Cargar desglose combustible
        private void DesgloseCombustible_Load(object sender, EventArgs e)
        {
            string t = Convert.ToString(Decimal.Round(Convert.ToDecimal(total),
2));
            string ccru =
Convert.ToString(Decimal.Round(Convert.ToDecimal(combustibleCrucero), 2));
            string casc =
Convert.ToString(Decimal.Round(Convert.ToDecimal(combustibleAsc), 4));
            string cdes =
Convert.ToString(Decimal.Round(Convert.ToDecimal(combustibleDesc), 4));
            string cex =
Convert.ToString(Decimal.Round(Convert.ToDecimal(combustibleExtra), 2));
            string calt =
Convert.ToString(Decimal.Round(Convert.ToDecimal(combustibleAeropuertoAlt
), 2));
            label1.Text = "TOTAL: " + t + " gal";
            label3.Text = "Combustible de la fase crucero: " + ccru + " gal";
            label4.Text = "Combustible en ascenso: " + casc + " gal";
            label5.Text = "Combustible en descenso: " + cdes + " gal";
            label6.Text = "Combustible para arranque y rodadura: " +
Convert.ToString(combustibleTaxi) + " gal";
            label7.Text = "Combustible para 45min extra en fase crucero: " + cex + "
gal";
            label8.Text = "Combustible llegar a aeropuerto alternativo: " + calt + "
gal";
        }
    }
}
```

### A.1.7. Comentarios

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace FlightTrackApp
{
    public partial class Comentario : Form
    {
        string comentario;

        public Comentario()
        {
            InitializeComponent();
```

```
            Font = new System.Drawing.Font("Segoe UI", 8);
            BackColor = Color.White;
            Color colorbotones = Color.FromArgb(246, 241, 209);
            ok.BackColor = colorbotones;
            fondo.BackColor = Color.FromArgb(71, 110, 125);
        }

        // Sets y Gets
        public void SetComentario(string c)
        {
            comentario = c;
        }

        public string GetComentario()
        {
            return comentario;
        }

        // Cargar comentario
        private void Comentario_Load(object sender, EventArgs e)
        {
            textBox1.Text = comentario;
        }

        // Guardar comentario
        private void ok_Clic(object sender, EventArgs e)
        {
            comentario = textBox1.Text;
            Close();
        }
    }
}
```

### A.1.8. Peso y balance

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using Lib;


namespace FlightTrackApp
{
    public partial class PesoyBalance : Form
    {
        int escala;
        int offsetX;
```

```
int offsetY;
double totalcomb;
bool iniciado = false;
bool esNormal = false;
Avion avion;
Graphics g;
Point puntodibujo;
List<Point> Utility = new List<Point>();
List<Point> Normal = new List<Point>();
readonly Balance balance = new Balance();

public PesoyBalance()
{
    InitializeComponent();
    Font = new Font("Segoe UI", 8);
    BackColor = Color.White;
    Color colorbotones = Color.FromArgb(246, 241, 209);
    calcular.BackColor = colorbotones;
    coloruti.BackColor = Color.FromArgb(241, 155, 106);
    colornormal.BackColor = Color.FromArgb(111, 159, 151);
    // Color recuadro
    Color azul = Color.FromArgb(111, 159, 151);
    fondo.BackColor = azul;
    explicacionlab.BackColor = azul;
    lab2.BackColor = azul;
    lab3.BackColor = azul;
    lab4.BackColor = azul;
    lab5.BackColor = azul;
    lab6.BackColor = azul;
    lab7.BackColor = azul;
    lab8.BackColor = azul;

}

// Sets
public void SetAvion(Avion a)
{
    this.avion = a;
}

public void SetComb(double c)
{
    totalcomb = c;
}

// Cargar PesoyBalance. Contempla que el avión es de categoría normal
private void PesoyBalance_Load(object sender, EventArgs e)
{
    Utility = avion.GetUtilityCategory();
    Normal = avion.GetNormalCategory();
```

```
            balance.SetCategorias(Utility, Normal);
            avionBox.Text = Convert.ToString(avion.GetPeso() * 0.453592);
            comlab.Text =
Decimal.Round(Convert.ToDecimal(totalcomb/0.26417),2).ToString();
            // Establecer offsets para gráfica picturebox
            offsetX = Utility[0].X - 10;
            offsetY = Utility[0].Y + 1000;
            // Explicación
            explicacionlab.Text = "Rellena los pesos para comprobar que el balance
es correcto " + "\n" + "para la categoría normal:";
        }


        // Pintar en picturebox
        private void pictureBox1_Paint(object sender, PaintEventArgs e)
        {
            g = e.Graphics;
            escala = 20;
            PointF[] poligono_u = balance.GetDibujoPoligono(Utility, escala);
            PointF[] poligono_n = balance.GetDibujoPoligono(Normal, escala);
            g.ScaleTransform(5, 5);

            // Definir pens y brushes
            Pen newPen = new Pen(Color.Black, 2.5f / 5); // los pens no se verán
escalados
            Pen ejes = new Pen(Color.Black, 1.5f / 5);
            SolidBrush bn = new SolidBrush(Color.FromArgb(200,111,159,151));
            SolidBrush bu = new SolidBrush(Color.FromArgb(200,241,155,106));

            // Dibujar polígonos
            g.FillPolygon(bn, poligono_n);
            g.FillPolygon(bu, poligono_u);
            g.DrawPolygon(newPen, poligono_u);
            g.DrawPolygon(newPen, poligono_n);
            // Dibujar ejes
            g.DrawLine(ejes, new Point((int)poligono_u[0].X - 2,
(int)poligono_u[0].Y), new Point((int)poligono_n[4].X - 2, (int)poligono_u[0].Y));
//eje X
            g.DrawLine(ejes, new Point((int)poligono_u[0].X - 2,
(int)poligono_u[0].Y), new Point((int)poligono_u[0].X - 2, (int)poligono_n[4].Y));
//eje Y


            // Pintar el punto resultante y comrpobar si está dentro de la categoría
normal
            if (iniciado)
            {
                g.FillEllipse(Brushes.Red, puntodibujo.X, puntodibujo.Y, 1, 1);
                iniciado = false;
                RutaAer r = new RutaAer();
```

```
                esNormal = r.PuntoEstaContenido(puntodibujo, poligono_n);
            }
        }

        // Calcular peso y balance. Si las casillas se dejan en blanco se interpretan
como 0 kg
        private void calcular_Clic(object sender, EventArgs e)
        {
            List<string> errores = new List<string>();
            // Obtener pesos típicos de la aeronave
            List<double> equipaje = avion.GetEquipaje();
            List<double> b = avion.GetBrazo();
            double empty = avion.GetPeso();
            // Obtener los pesos introducidos por el usuario
            double piloto = 0;
            if (pilotobox.Text != "")
                piloto = Convert.ToDouble(pilotobox.Text);
            if (piloto == 0)
                errores.Add("El peso del piloto no puede ser nulo");
            if (piloto < 0)
                errores.Add("El peso del piloto no puede ser negativo");
            double copiloto = 0;
            if (copilotobox.Text != "")
                copiloto = Convert.ToDouble(copilotobox.Text);
            if (copiloto < 0)
                errores.Add("El peso del copiloto no puede ser negativo");
            double pilotoycopiloto = piloto + copiloto;
            double comb = totalcomb / 0.26417;
            double pax = 0;
            if (pasajerobox.Text != "")
                pax = Convert.ToDouble(pasajerobox.Text);
            if (pax < 0)
                errores.Add("El peso de los pasajeros no puede ser negativo");

            // Obtener pesos de los equipajes
            double area1 = 0;
            if (area1box.Text != "")
                area1 = Convert.ToDouble(area1box.Text);
            if (area1 < 0)
                errores.Add("El peso del equipaje (área 1) no puede ser negativo");
            if (area1 > equipaje[0])
                errores.Add("El peso del equipaje en el área 1 excede el peso
máximo " + "(" + equipaje[0] + ")");
            double area2 = 0;
            if (area2box.Text != "")
                area2 = Convert.ToDouble(area2box.Text);
            if (area2 < 0)
                errores.Add("El peso del equipaje (área 2) no puede ser negativo");
            if (area2 > equipaje[1])
```

```
            errores.Add("El peso del equipaje en el área 2 excede el peso
máximo " + "(" + equipaje[1] + ")");
        if (area1 + area2 > equipaje[2])
            errores.Add("El peso del equipaje total no puede exceder los " +
equipaje[2] + " kg");

        // Calcular el peso total y el momento
        double pesototalkg = empty * 0.453592 + pilotoycopiloto + comb + pax +
area1 + area2;
        double momento = empty * 0.453592 * b[0] + comb * b[1] +
pilotoycopiloto * b[2] + pax * b[3] + area1 * b[4] + area2 * b[5]; //en kg·m

        double MTOW = balance.GetMTOW();
        if (pesototalkg * 2.20462 > MTOW)
            MessageBox.Show("Demasiado peso. Excede MTOW", "",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        // Si no hay error, mostrar los resultados. Se pide modificación de pesos
en caso de no estar en categoría normal
        if (errores.Count == 0)
        {
            // Resultados numéricos
            resultado.Text = "El peso total es " +
Decimal.Round(Convert.ToDecimal(pesototalkg), 2) + " kg" + "\n" + "El
momento total es " + Decimal.Round(Convert.ToDecimal(momento), 2) +
"/1000 kg·mm";
            resultado.BackColor = Color.FromArgb(241,151,106);
            resultado.BorderStyle = BorderStyle.FixedSingle;
            resultado.Location = new Point(535, 330);
            resultado.Font = new Font("Segoe UI", 10, FontStyle.Bold);
            resultado.ForeColor = Color.White;


            // Punto en gráfica
            Point posicion = new Point(Convert.ToInt32(momento * 86.7962 /
1000), Convert.ToInt32(pesototalkg * 2.20462)); //Punto: (Momento/1000 lb·in,
peso cargado lb)
            puntodibujo = new Point(posicion.X - offsetX, -(posicion.Y - offsetY) /
escala);
            iniciado = true;
            pictureBox1.Refresh();

            // Categoría
            if (esNormal)
                MessageBox.Show("Peso y balance correcto", "",
MessageBoxButtons.OK, MessageBoxIcon.Information);
            else
                MessageBox.Show("Modifique valores de peso para un correcto
balance", "", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        // Mostrar los errores
```

```
            else
            {
                string mensaje = "";
                foreach (string s in errores)
                    mensaje = mensaje + "-" + s + "\n";
                MessageBox.Show("No se puede calcular momento debido a los
siguientes errores:" + "\n" + mensaje, "", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            }
        }
    }
}
```

### A.1.9. Metar: principal

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.Windows.Forms;
using Lib;

namespace FlightTrackApp
{
    public partial class METAR : Form
    {
        List<AD> ADspain;
        List<string> ICAOs;

        public METAR()
        {
            InitializeComponent();
            Font = new Font("Segoe UI", 8);
            BackColor = Color.White;
            Color colorbotones = Color.FromArgb(246, 241, 209);
            morigen.BackColor = colorbotones;
            mdest.BackColor = colorbotones;
            malt.BackColor = colorbotones;
            motro.BackColor = colorbotones;
        }

        // Sets
        public void SetICAOS(List<string> i)
        {
            ICAOs = i;
        }

        public void SetADs(List<AD> a)
        {
```

```
            ADspain = a;
        }

        // Cargar METAR
        private void METAR_Load(object sender, EventArgs e)
        {
        }

        // Botón origen - Inicia form DetalleMetar
        private void morigen_Clic(object sender, EventArgs e)
        {
            if (ICAOs.Count != 0)
            {
                DetalleMetar m = new DetalleMetar();
                m.SetICAO(ICAOs[0]);
                m.ShowDialog();
            }
            else
                MessageBox.Show("No existe ningún aeropuerto de origen", "",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

        // Botón destino - Inicia form DetalleMetar
        private void mdest_Clic(object sender, EventArgs e)
        {
            if (ICAOs.Count >= 2)
            {
                DetalleMetar m = new DetalleMetar();
                m.SetICAO(ICAOs[1]);
                m.ShowDialog();
            }
            else
                MessageBox.Show("No existe ningún aeropuerto de destino", "",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

        // Botón alternativo - Inicia form DetalleMetar
        private void malt_Clic(object sender, EventArgs e)
        {
            if (ICAOs.Count == 3)
            {
                DetalleMetar m = new DetalleMetar();
                m.SetICAO(ICAOs[2]);
                m.ShowDialog();
            }
            else
                MessageBox.Show("No hay ningún aeropuerto alternativo
seleccionado", "", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
```

```
        // Botón otro aeropuerto - Inicia form MetarOtroAD
        private void motro_Clic(object sender, EventArgs e)
        {
            MetarOtroAD m = new MetarOtroAD();
            m.SetAD(ADspain);
            m.ShowDialog();
        }
    }
}
```

### A.1.10. Metar: detalle

```
using System;
using System.Windows.Forms;
using Newtonsoft.Json.Linq;
using System.Net;
using System.IO;
using Lib;
using System.Drawing;

namespace FlightTrackApp
{
    public partial class DetalleMetar : Form
    {
        string icao;
        string baseurl = "https://api.checkwx.com/metar/";
        string apikey = [redacted];
        Metar_ m;

        public DetalleMetar()
        {
            InitializeComponent();
            Font = new Font("Segoe UI", 8);
            BackColor = Color.White;
            Color colorbotones = Color.FromArgb(246, 241, 209);
            decode.BackColor = colorbotones;
            fondo.BackColor = Color.FromArgb(11, 159, 151);
        }

        // Set
        public void SetICAO(string i)
        {
            icao = i;
        }

        // Carga de Detalle Metar
        private void DetalleMetar_Load(object sender, EventArgs e)
        {
            try
            {
```

```
            string url = baseurl + icao + apikey;
            HacerRequest(url);
            mtexto.Text = m.GetMetar();
            mtexto.Font = new Font("Segoe UI", 10, FontStyle.Bold);
            mtexto.ForeColor = Color.White;
        }
        catch
        {
            mtexto.Text = "Este aeropuerto no proporciona informes METAR. " +
"\n" + "Prueba a buscar un aeródromo cercano en 'METAR de otro aeropuerto'";
        }
    }

    // Método invocable para hacer petición de METAR a API CheckWX
    private void HacerRequest(string url)
    {
        var httpRequest = (HttpWebRequest)WebRequest.Create(url);
        var httpResponse = (HttpWebResponse)httpRequest.GetResponse();
        using (var streamReader = new
StreamReader(httpResponse.GetResponseStream()))
        {
            var resultado = streamReader.ReadToEnd();
            JObject j = JObject.Parse(resultado);
            JArray datos = (JArray)j["data"];
            if (datos.ToString() != "{[]}")
            {
                Metar_ met = new Metar_(datos);
                m = met;
            }
        }
    }

    // Botón para decodificar - abre formulario DecodificaMETAR
    private void decode_Clic(object sender, EventArgs e)
    {
        if (m == null)
            MessageBox.Show("No se dispone de METAR para decodificar");
        else
        {
            DecodificaMETAR d = new DecodificaMETAR();
            d.SetMetar(m);
            d.SetICAO(icao);
            d.ShowDialog();
        }
    }
  }
}
```

### A.1.11. Metar: otro aeródromo

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using Lib;
using System.Net;
using System.IO;
using Newtonsoft.Json.Linq;
using System.Drawing;

namespace FlightTrackApp
{
    public partial class MetarOtroAD : Form
    {
        string baseurl = "https://api.checkwx.com/metar/";
        string apikey = [redacted];
        string icao;
        Metar_ m;
        List<AD> ADspain;

        public MetarOtroAD()
        {
            InitializeComponent();
            Font = new Font("Segoe UI", 8);
            BackColor = Color.White;
            Color colorbotones = Color.FromArgb(246, 241, 209);
            decode.BackColor = colorbotones;
            fondo.BackColor = Color.FromArgb(11, 159, 151);
        }

        // Set AD
        public void SetAD(List<AD> a)
        {
            ADspain = a;
        }

        // Cargar MetarOtroAD
        private void MetarOtroAD_Load(object sender, EventArgs e)
        {
            foreach (AD a in ADspain)
                comboBox1.Items.Add(a.GetNombreAD());
        }

        // Selección de aeropuerto
        private void comboBox1_SelectedIndexChanged(object sender, EventArgs
e)
        {
            try
            {
```

```
            icao = ADspain[comboBox1.SelectedIndex].GetICAO();
            string url = baseurl + icao + apikey;
            HacerRequest(url);
            mlabel.Text = m.GetMetar();
            mlabel.Font = new Font("Segoe UI", 10,FontStyle.Bold);
            mlabel.ForeColor = Color.White;
        }
        catch
        {
            mlabel.Text = "Este aeropuerto no proporciona informes METAR. " +
"\n" + "Prueba a buscar un aeródromo cercano";
        }
    }

    // Botón de decodificar - Abre form DecodificaMETAR
    private void decode_Clic(object sender, EventArgs e)
    {
        if (m == null)
            MessageBox.Show("No se dispone de METAR para decodificar", "",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        else
        {
            DecodificaMETAR d = new DecodificaMETAR();
            d.SetMetar(m);
            d.SetICAO(icao);
            d.ShowDialog();
        }
    }

    // Método invocable para hacer petición
    private void HacerRequest(string url)
    {
        var httpRequest = (HttpWebRequest)WebRequest.Create(url);

        var httpResponse = (HttpWebResponse)httpRequest.GetResponse();
        using (var streamReader = new
StreamReader(httpResponse.GetResponseStream()))
        {
            var resultado = streamReader.ReadToEnd();
            JObject j = JObject.Parse(resultado);
            JArray datos = (JArray)j["data"];
            if (datos.ToString() != "{[]}")
            {
                Metar_ met = new Metar_(datos);
                m = met;
            }
        }
    }
  }
}
```

### A.1.12. Metar: decodificación

```
using System;
using System.Drawing;
using System.Windows.Forms;
using Lib;

namespace FlightTrackApp
{
   public partial class DecodificaMETAR : Form
   {
      Metar_ m;
      string icao;

      public DecodificaMETAR()
      {
         InitializeComponent();
         Font = new Font("Segoe UI", 8);
         BackColor = Color.White;
         fondo.BackColor = Color.FromArgb(246, 155, 106);
         ICAOlabel.BackColor = Color.FromArgb(246, 155, 106);
         ICAOlabel.Font = new Font("Segoe UI", 10, FontStyle.Bold);
      }

      // Sets y Gets
      public void SetMetar(Metar_ met)
      {
         m = met;
      }

      public void SetICAO(string i)
      {
         icao = i;
      }

      // Decodificar metar
      private void DecodificaMETAR_Load(object sender, EventArgs e)
      {
         ICAOlabel.Text = icao;
         horalab.Text = "Hora de observación (ZULU): " + m.GetHora() + "  (" +
m.GetFecha() + ")";
         if (m.GetHaceviento())
            vientolab.Text = "Viento: " + m.GetVientoVel() + " kt, " +
m.GetVientoDir() + " º";
         else
            vientolab.Text = "Viento: sin viento";
         string vis = m.GetVis().ToString();
         if (vis == "9999")
```

```
            vis = "Más de 10000";
        vislab.Text = "Visibilidad horizontal: " + vis + " m";
        string basenub = "";
        if (m.GetBaseNubes() != null)
            basenub = ". Base de nubes: " + m.GetBaseNubes() + " ft";
        coberturanubeslab.Text = "Cobertura del cielo: " +
m.GetNubesDecodificado(m.GetNubes()) + basenub;
        templab.Text = "Temperatura/Temperatura de rocío: " +
m.GetTemp().ToString() + "ºC / " + m.GetRocio().ToString() + "ºC";
        presionlab.Text = "Presión: " + m.GetPresion().ToString() + " hpa";
        humlab.Text = "Porcentaje de humedad: " + m.GetHum().ToString() + "
%";
      }
   }
}
```

### A.1.13. Ayuda

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace FlightTrackApp
{
   public partial class Ayuda : Form
   {
      bool crea;
      bool alt;
      bool nav;
      bool fechayhora;
      bool comb;
      bool pdf;
      bool com;
      bool peso;
      bool metar;

      public Ayuda()
      {
         InitializeComponent();
         Font = new Font("Segoe UI", 8);
         BackColor = Color.White;
      }

      public void SetBool(bool crea, bool alt, bool nav, bool fechayhora, bool
comb, bool pdf, bool com, bool peso, bool metar)
      {
         this.crea = crea;
         this.alt = alt;
```

```
            this.nav = nav;
            this.fechayhora = fechayhora;
            this.comb = comb;
            this.pdf = pdf;
            this.com = com;
            this.peso = peso;
            this.metar = metar;
        }

        private void Ayuda_Load(object sender, EventArgs e)
        {
            if (crea)
                crealab.Checked = true;
            if (alt)
                altlab.Checked = true;
            if (nav)
                navlab.Checked = true;
            if (fechayhora)
                fechalab.Checked = true;
            if (comb)
                comblab.Checked = true;
            if (pdf)
                pdfnavlab.Checked = true;
            if (com)
                guionlab.Checked = true;
            if (peso)
                pesolab.Checked = true;
            if (metar)
                metlab.Checked = true;
        }
    }
}
```

### A.1.14. Biblioteca

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.IO;
using System.Reflection;
using Lib;
using System.Drawing;

namespace FlightTrackApp
{
    public partial class Biblioteca : Form
    {
```

```
int i;
bool deseacarga = false;
bool cambio = false;
List<AD> ADspain;
List<RutaAer> lista = new List<RutaAer>();
List<RutaAer> listaborrar = new List<RutaAer>();
List<List<AD>> AerSel = new List<List<AD>>();

public Biblioteca()
{
    InitializeComponent();
    Font = new System.Drawing.Font("Segoe UI", 8);
    BackColor = Color.White;
    Color colorbotones = Color.FromArgb(246, 241, 209);
    carga.BackColor = colorbotones;
    borra.BackColor = colorbotones;
    fondo.BackColor = Color.FromArgb(11, 159, 151);
}

// Sets y Gets
public void SetADspain(List<AD> a)
{
    ADspain = a;
}

public RutaAer GetRuta()
{
    return lista[i];
}

public List<RutaAer> GetLista()
{
    return lista;
}

public List<RutaAer> GetListaBorrar()
{
    return listaborrar;
}

public bool GetCarga()
{
    return deseacarga;
}

public bool GetCambio()
{
    return cambio;
}
```

```
        public List<AD> GetAerSel()
        {
            return AerSel[i];
        }


        // Cargar biblioteca
        private void Biblioteca_Load(object sender, EventArgs e)
        {
            BackColor = Color.White;
            dataGridView1.RowsDefaultCellStyle.Alignment =
DataGridViewContentAlignment.MiddleCenter;
            dataGridView1.Columns.Add("nombre", "Nombre");
            dataGridView1.Columns.Add("o", "Origen");
            dataGridView1.Columns.Add("d", "Destino");
            dataGridView1.Columns.Add("f", "Fecha");
            dataGridView1.Columns.Add("t", "Tiempo (min)");
            dataGridView1.Columns.Add("c", "Combustible (gal)");
            dataGridView1.Columns.Add("p", "Puntuación");
            // Cargar archivos de carpeta "Rutas"
            string directorio =
Path.GetDirectoryName(Assembly.GetEntryAssembly().Location);
            string[] archivos = Directory.GetFiles(directorio + "\\Rutas");
            foreach (string s in archivos)
            {
                RutaAer r = new RutaAer();
                string nombre = s.Replace(directorio + "\\Rutas\\", "").Replace(".txt",
"");
                List<AD> AeropuertosSeleccionados = r.CargarRuta(s, ADspain);
                r.SetADsel(AeropuertosSeleccionados);

                string inicio = AeropuertosSeleccionados.First().GetNombreAD();
                //Sustituir Aeródromo y Aeropuerto por "A."
                inicio = inicio.Replace("Aeropuerto", "A.");
                inicio = inicio.Replace("Aeródromo", "A.");
                string fin = AeropuertosSeleccionados.Last().GetNombreAD();
                fin = fin.Replace("Aeropuerto", "A.");
                fin = fin.Replace("Aeródromo", "A.");

                string fecha = r.GetFecha();
                string t = Decimal.Round(Convert.ToDecimal(r.Gett() / 60)).ToString();
                string comb = Decimal.Round(Convert.ToDecimal(r.GetComb()),
2).ToString();
                if (comb == "0")
                    comb = "Sin calcular";
                string puntuacion = Convert.ToString(r.GetPuntuacion());
                dataGridView1.Rows.Add(nombre, inicio, fin, fecha, t, comb,
puntuacion);
                lista.Add(r);
                AerSel.Add(AeropuertosSeleccionados);
```

```
            }
        }

        // Borrar archivo
        private void borra_Clic(object sender, EventArgs e)
        {
            dataGridView1.Rows.RemoveAt(dataGridView1.CurrentRow.Index);
            listaborrar.Add(lista[i]);
            lista.Remove(lista[i]);
        }

        // Carga archivo
        private void carga_Clic(object sender, EventArgs e)
        {
            deseacarga = true;
            Close();
        }

        // Guardar el número de fila seleccionado
        private void dataGridView1_Clic(object sender, EventArgs e)
        {
            i = dataGridView1.CurrentCell.RowIndex;
        }

        // Cambiar puntuación
        private void dataGridView1_CellValueChanged(object sender,
DataGridViewCellEventArgs e)
        {
            if (dataGridView1.CurrentCell.ColumnIndex == 6)
            {

lista[i].SetPuntuacion(Convert.ToDouble(dataGridView1.CurrentCell.Value));
                cambio = true;
            }
        }
    }
}
```

## A.2. Clases

### A.2.1. AD

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Device.Location;
```

```
namespace Lib
{
   public class AD
   {
      string nombre;
      string icao;
      GeoCoordinate coordenadas;


      //Constructores
      public AD()
      { }

      public AD(string nombre, double X, double Y, double Z)
      {
         this.nombre = nombre;
         coordenadas.Longitude = X;
         coordenadas.Latitude = Y;
         coordenadas.Altitude = Z;  //en ft
      }


      // Sets y Gets
      public void SetNombreAD(string a)
      {
         nombre = a;
      }

      public void SetICAO(string a)
      {
         icao = a;
      }

      public void SetCoorAD(GeoCoordinate c)
      {
         coordenadas = c;
      }

      public string GetNombreAD()
      {
         return nombre;
      }

      public GeoCoordinate GetCoorAD()
      {
         return coordenadas;
      }
```

```
        public string GetICAO()
        {
            return icao;
        }

        public double GetElevacionAD(double lat, double lon, List<AD> lista)
        {
            coordenadas = lista.Find(x => x.GetCoorAD().Latitude == lat &&
x.GetCoorAD().Longitude == lon).coordenadas;
            return coordenadas.Altitude;
        }

        // Leer fichero de aeródromos
        public List<AD> LeerAD(string fichero)
        {
            List<AD> Aerodromos = new List<AD>();
            StreamReader F = new StreamReader(fichero);
            string linea = F.ReadLine();
            while (linea != null)
            {
                AD temp = new AD();
                string[] trozos = linea.Split('\t', ',');
                nombre = trozos[0];
                double X = Convert.ToDouble(trozos[1]);
                double Y = Convert.ToDouble(trozos[2]);
                double Z = Convert.ToDouble(trozos[3]);
                GeoCoordinate c = new GeoCoordinate(X, Y, Z);
                temp.SetNombreAD(nombre);
                temp.SetICAO(trozos[4]);
                temp.SetCoorAD(c);
                Aerodromos.Add(temp);
                linea = F.ReadLine();
            }
            return Aerodromos;
        }

        // Busca aeródromo en lista por el nombre
        public AD BuscaAD(List<AD> lista, string nombre)
        {
            AD Aer = new AD();
            foreach (AD a in lista)
            {
                if (a.GetNombreAD() == nombre)
                {
                    Aer = a;
                }
            }
            return Aer;
        }
    }
```

```
}
```

### A.2.2. Avion

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Drawing;

namespace Lib
{
   public class Avion
   {
      string modelo;
      double peso;
      double peso_empty;
      double capacidad_combustible;
      double alt;
      double KTAS;
      double GPH;
      double techo_vuelo;
      double taxi;
      readonly double[,] crucero_performance = new double[6, 3];
      readonly double[,] climb_performance = new double[11, 4];
      readonly List<Point> Utility = new List<Point>();
      readonly List<Point> Normal = new List<Point>();
      readonly List<double> MTOW = new List<double>();
      readonly List<double> equipaje = new List<double>();
      readonly List<double> brazo_momento = new List<double>();



      //Constructores
      public Avion()
      { }

      public Avion(string modelo, double peso, double peso_empty, double cb,
double tv, double taxi, double[,] cru, double[,] cli, List<Point> u, List<Point> n,
List<double> mtow, List<double> e, List<double> b)
      {
         this.modelo = modelo;
         this.peso = peso;
         capacidad_combustible = cb;
         this.peso_empty = peso_empty;
         techo_vuelo = tv;
         this.taxi = taxi;
         crucero_performance = cru;
```

```
            climb_performance = cli;
            Utility = u;
            Normal = n;
            MTOW = mtow; //lista con MTOW Utility y Normal
            equipaje = e; //lista con peso Area 1, Area 2 y máximo entre 2 áreas
(cat. Normal)
            brazo_momento = b; //lista brazos: peso vacío, combustible, piloto,
pasajeros, área equip. 1, área equip. 2
        }


        // Sets y Gets

        public void SetModelo(string a)
        {
            modelo = a;
        }

        public void SetPeso(double b)
        {
            peso = b;
        }

        public void SetCapacidad(double c)
        {
            capacidad_combustible = c;
        }

        public string GetModelo()
        {
            return modelo;
        }

        public double GetPeso()
        {
            return peso;
        }

        public double GetCapacidad()
        {
            return capacidad_combustible;
        }

        public double GetTecho()
        {
            return techo_vuelo;
        }

        public double GetCombustibleTaxi()
        {
```

```
        return taxi;
    }

    public List<Point> GetUtilityCategory()
    {
        return Utility;
    }

    public List<Point> GetNormalCategory()
    {
        return Normal;
    }

    public List<double> GetMTOW()
    {
        return MTOW;
    }

    public List<double> GetEquipaje()
    {
        return equipaje;
    }

    public List<double> GetBrazo()
    {
        return brazo_momento;
    }

    // Get lista con ALT, BPH, KTAS, GPH interpolado (fase crucero)
    public List<double> GetCruceroPerformance(double alt)
    {
        List<double> performance = new List<double>();
        bool encontrado = false;
        double KTAS = 0;
        double GPH = 0;
        int i = 0;
        while (i < 6)
        {
            if ((crucero_performance[i, 0] > alt) && !encontrado && i != 0)
            {
                double menor_alt = this.crucero_performance[i - 1, 0];
                double mayor_alt = this.crucero_performance[i, 0];
                double menor_KTAS = this.crucero_performance[i - 1, 1];
                double mayor_KTAS = this.crucero_performance[i, 1];
                double menor_GPH = this.crucero_performance[i - 1, 2];
                double mayor_GPH = this.crucero_performance[i, 2];
                encontrado = true;
                // interpolación
                double factor = (alt - menor_alt) / (mayor_alt - menor_alt);
                KTAS = menor_KTAS + (factor * (mayor_KTAS - menor_KTAS));
```

```
                GPH = menor_GPH + (factor * (mayor_GPH - menor_GPH));

            }
            i++;
        }

        if (!encontrado)
        {
            if (alt < this.crucero_performance[0, 0])
            {
                KTAS = this.crucero_performance[0, 1];
                GPH = this.crucero_performance[0, 2];
            }

            if (alt > this.crucero_performance[5, 0])
            {
                KTAS = this.crucero_performance[5, 1];
                GPH = this.crucero_performance[5, 2];
            }

        }
        performance.Add(alt);
        performance.Add(KTAS);
        performance.Add(GPH);

        return performance;
    }

    // Get lista con ALT, BPH, KTAS, GPH interpolado (fases climb)
    public List<double> GetClimbPerformance(double alt)
    {
        List<double> performance = new List<double>();
        bool encontrado = false;
        double time = 0;
        double dist_climb = 0;
        double GPH = 0;
        int i = 0;
        while (i < 11)
        {
            if ((this.climb_performance[i, 0] > alt) && !encontrado)
            {
                double menor_alt = this.climb_performance[i - 1, 0];
                double mayor_alt = this.climb_performance[i, 0];
                double menor_t = this.climb_performance[i - 1, 1];
                double mayor_t = this.climb_performance[i, 1];
                double menor_GPH = this.climb_performance[i - 1, 2];
                double mayor_GPH = this.climb_performance[i, 2];
                double menor_d = this.climb_performance[i - 1, 3];
                double mayor_d = this.climb_performance[i, 3];
                encontrado = true;
```

```
                // interpolación
                double factor = (alt - menor_alt) / (mayor_alt - menor_alt);
                time = menor_t + (factor * (mayor_t - menor_t));
                GPH = menor_GPH + (factor * (mayor_GPH - menor_GPH));
                dist_climb = menor_d + (factor * (mayor_d - menor_d));

            }
            i++;
        }

        if (!encontrado)
        {
            if (alt < this.climb_performance[0, 0])
            {
                KTAS = this.climb_performance[0, 1];
                GPH = this.climb_performance[0, 2];
            }

            if (alt > this.climb_performance[10, 0])
            {
                KTAS = this.climb_performance[10, 1];
                GPH = this.climb_performance[10, 2];
            }

        }
        performance.Add(alt);
        performance.Add(time);
        performance.Add(GPH);
        performance.Add(dist_climb);
        return performance;
    }


    // Leer fichero sobre el avión
    public Avion LeerFichero(string fichero)
    {
        StreamReader F = new StreamReader(fichero);
        string linea;
        while ((linea = F.ReadLine()) != null)
        {

            string[] trozos = linea.Split('\t');
            if (String.Equals(trozos[0], "Aeronave"))
            {
                linea = F.ReadLine();
                string[] t = linea.Split('\t');
                SetModelo(t[0]);
                peso_empty = Convert.ToDouble(t[1]);
                capacidad_combustible = Convert.ToDouble(t[2]);
```

```
            }

            if (String.Equals(trozos[0], "-"))
            {
                F.ReadLine();
                F.ReadLine();

                for (int i = 0; i < 6; i++)
                {
                    linea = F.ReadLine();
                    string[] t = linea.Split('\t');
                    alt = Convert.ToDouble(t[0]);
                    KTAS = Convert.ToDouble(t[2]);
                    GPH = Convert.ToDouble(t[3]);
                    crucero_performance[i, 0] = alt;
                    crucero_performance[i, 1] = KTAS;
                    crucero_performance[i, 2] = GPH;
                }
            }

            if (String.Equals(trozos[0], ";"))
            {
                F.ReadLine();
                F.ReadLine();
                for (int i = 0; i < 11; i++)
                {
                    linea = F.ReadLine();
                    string[] tr = linea.Split('\t');
                    climb_performance[i, 0] = Convert.ToDouble(tr[0]); //alt
                    climb_performance[i, 1] = Convert.ToDouble(tr[1]); //time
                    climb_performance[i, 2] = Convert.ToDouble(tr[2]);
//combustiblelimb
                    climb_performance[i, 3] = Convert.ToDouble(tr[3]);  //dist climb
                }

            }
            if (String.Equals(trozos[0], ";;"))
            {
                linea = F.ReadLine();
                string[] tr = linea.Split(' ');
                techo_vuelo = Convert.ToDouble(tr[1]);
                linea = F.ReadLine();
                tr = linea.Split(' ');
                taxi = Convert.ToDouble(tr[1]);

            }
            if (String.Equals(trozos[0], "---"))
            {
                F.ReadLine();
                linea = F.ReadLine();
```

```
string[] tr = linea.Split('\t');
int i = 1;
while (i < tr.Count())
{
    string[] p = tr[i].Split(',');
    int x = Convert.ToInt32(p[0]);
    int y = Convert.ToInt32(p[1]);
    Point punto = new Point(x, y);
    Utility.Add(punto);
    i++;
}

linea = F.ReadLine();
tr = linea.Split('\t');
i = 1;
while (i < tr.Count())
{
    string[] p = tr[i].Split(',');
    int x = Convert.ToInt32(p[0]);
    int y = Convert.ToInt32(p[1]);
    Point punto = new Point(x, y);
    Normal.Add(punto);
    i++;
}
F.ReadLine();
F.ReadLine();
linea = F.ReadLine();
while (linea != "")
{
    tr = linea.Split('\t');
    MTOW.Add(Convert.ToDouble(tr[1]));
    linea = F.ReadLine();
    i++;
}
F.ReadLine();
linea = F.ReadLine();
while (linea != "")
{
    tr = linea.Split('\t');
    equipaje.Add(Convert.ToDouble(tr[1]));
    linea = F.ReadLine();
    i++;
}

F.ReadLine();
linea = F.ReadLine();
while (linea != null)
{
    tr = linea.Split('\t');
    brazo_momento.Add(Convert.ToDouble(tr[1]));
```

```
                linea = F.ReadLine();
                i++;
            }
        }
    }

    Avion avion = new Avion(modelo, peso_empty, peso_empty,
capacidad_combustible, techo_vuelo, taxi, crucero_performance,
climb_performance, Utility, Normal, MTOW, equipaje, brazo_momento);
    return avion;

  }
 }
}
```

### A.2.3. Balance

```
using System.Collections.Generic;
using System.Drawing;

namespace Lib
{
  public class Balance
  {
    List<Point> Utility = new List<Point>();
    List<Point> Normal = new List<Point>();

    // Constructores
    public Balance()
    { }

    public Balance(List<Point> u, List<Point> n)
    {
      Utility = u;
      Normal = n;
    }

    // Sets y Gets
    public void SetCategorias(List<Point> u, List<Point> n)
    {
      Utility = u;
      Normal = n;
    }

    public PointF[] GetDibujoPoligono(List<Point> l, int escala)
    {
      PointF[] poligono = new PointF[l.Count];
      int i = 0;
```

```
            int offsetX = Utility[0].X - 10;
            int offsetY = Utility[0].Y + 1000;
            while (i < l.Count)
            {
                PointF f = new PointF();
                f.X = l[i].X - offsetX;
                f.Y = -(l[i].Y - offsetY) / escala;
                poligono[i] = f;
                i++;
            }
            return poligono;
        }

        public double GetMTOW()
        {
            double MTOW = 0;
            foreach (Point p in Normal)
                if (p.Y > MTOW)
                    MTOW = p.Y;
            return MTOW;
        }

    }
}
```

### A.2.4. EspacioAer

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.IO;
using System.Device.Location;
using SharpKml.Engine;
using SharpKml.Dom;
using GMap.NET;
using GMap.NET.WindowsForms;



namespace Lib
{
    public class EspacioAer
    {
        string nombre;
        string descripcion;
        string clase;
        string tipolimArriba;
```

```
        double limArriba;
        string tipolimAbajo;
        double limAbajo;
        List<GeoCoordinate> coordenadas;

        // Constructores
        public EspacioAer()
        { }

        public EspacioAer(string nombre, string clase, List<GeoCoordinate>
coordenadas, string tipolimArriba, double limArriba, string tipolimAbajo, double
limAbajo)
        {
            this.nombre = nombre;
            this.clase = clase;
            this.coordenadas = coordenadas;
            this.tipolimArriba = tipolimArriba;
            this.tipolimAbajo = tipolimAbajo;
            this.limArriba = limArriba;
            this.limAbajo = limAbajo;
        }

        // Gets

        public List<GeoCoordinate> GetCoordenadas()
        {
            return coordenadas;
        }

        public string GetNombre()
        {
            return nombre;
        }

        public string GetDescripcion()
        {
            return descripcion;
        }

        public string GetClase()
        {
            return clase;
        }

        public string GetTipoAbajo()
        {
            return tipolimAbajo;
        }
        public string GetTipoArriba()
        {
```

```
            return tipolimArriba;
        }

        public double GetLimAbajo()
        {
            return limAbajo;
        }

        public double GetLimArriba()
        {
            return limArriba;
        }

        public GMapPolygon GetPoligonoMapa(EspacioAer a)
        {
            List<PointLatLng> p = new List<PointLatLng>();
            foreach (var b in a.coordenadas)
            {
                PointLatLng punto = new PointLatLng(b.Latitude, b.Longitude);
                p.Add(punto);
            }
            GMapPolygon pol = new GMapPolygon(p, a.nombre);
            return pol;
        }

        // Leer archivo kml de espacios aéreos
        public List<EspacioAer> LeerEspacioAer(string fichero)
        {
            List<EspacioAer> Spain = new List<EspacioAer>();
            TextReader r = File.OpenText(fichero);
            KmlFile f = KmlFile.Load(r);
            Kml _kml = f.Root as Kml;
            IEnumerable<SharpKml.Dom.Placemark> placemarks =
_kml.Flatten().OfType<SharpKml.Dom.Placemark>();
            foreach (var placemark in placemarks)
            {
                EspacioAer temp = new EspacioAer();
                var carpeta = placemark.GetParent<Folder>();
                string nombrecarpeta = carpeta.Name;
                string[] trozos = nombrecarpeta.Split(' ');
                while (!String.Equals(trozos[0], "CLASS"))
                {
                    carpeta = carpeta.GetParent<Folder>();
                    nombrecarpeta = carpeta.Name;
                    trozos = nombrecarpeta.Split(' ');
                }
                clase = nombrecarpeta;
                temp.clase = clase;
                temp.nombre = placemark.Name;
```

```
            if (placemark.Geometry != null)
            {
                string tipo = placemark.Geometry.GetType().Name;
                if (String.Equals(tipo, "Polygon"))
                {
                    Polygon p = placemark.Geometry as SharpKml.Dom.Polygon;
                    List<GeoCoordinate> clista = new List<GeoCoordinate>();
                    if ((placemark.Description != null) && (p.OuterBoundary != null))
                    {
                        temp.descripcion = placemark.Description.Text;
                        string[] alturas = temp.descripcion.Split('\n', '-');

                        LinearRing lr = p.OuterBoundary.LinearRing as LinearRing;
                        string todolimarriba = alturas[alturas.Length - 1];
                        string todolimabajo = alturas[alturas.Length - 2];
                        string[] conalturaarriba = ConversionAltura(todolimarriba);
                        string[] conalturaabajo = ConversionAltura(todolimabajo);
                        limArriba = Convert.ToDouble(conalturaarriba[0]);
                        tipolimArriba = conalturaarriba[1];
                        limAbajo = Convert.ToDouble(conalturaabajo[0]);
                        tipolimAbajo = conalturaabajo[1];

                        foreach (var a in lr.Coordinates)
                        {
                            GeoCoordinate c = new GeoCoordinate();
                            c.Latitude = a.Latitude;
                            c.Longitude = a.Longitude;
                            clista.Add(c);
                        }
                        temp.coordenadas = clista;
                        temp.limAbajo = limAbajo;
                        temp.limArriba = limArriba;
                        temp.tipolimAbajo = tipolimAbajo;
                        temp.tipolimArriba = tipolimArriba;
                        Spain.Add(temp);
                    }
                }
            }
        }
        return Spain;
    }

    // Decuelve la altura en pies según el código
    public string[] ConversionAltura(string a)
    {
        string[] alt = new string[2];
        if (String.Equals(a, "SFC"))
        {
            alt[0] = "0";
            alt[1] = "SFC";
```

```
            }

            else if (String.Equals(a, "UNLTD"))
            {
                alt[0] = "60000"; // numero mayor que  FL 520
                alt[1] = "SFC";
            }

            else if (a.StartsWith("FL"))
            {
                string sinft = a.Replace("FL", string.Empty);
                alt[0] = Convert.ToString(Convert.ToDouble(sinft) * 100);
                alt[1] = "FL";
            }

            else if (a.Contains("AGL"))
            {
                string sinAGL = a.Replace("AGL", string.Empty);
                alt[0] = sinAGL;
                alt[1] = "AGL"; // por encima del terreno
            }

            else if (a.Contains("ALT"))
            {
                string sinALT = a.Replace("ALT", string.Empty);
                alt[0] = sinALT;
                alt[1] = "ALT";
            }

            return alt;
        }
    }

}
```

### A.2.5. KML

```
using System.Collections.Generic;
using SharpKml.Dom;
using System.Device.Location;
using SharpKml.Base;
using SharpKml.Engine;
using System.IO;

namespace Lib
{
    public class KML
    {
```

```
        readonly List<GeoCoordinate> WP;

        // Constructor
        public KML(List<GeoCoordinate> WP)
        {
            this.WP = WP;
        }

        // Crear fichero kml con la ruta
        public KmlFile CrearKML(string fichero)
        {
            var kml = new Kml();
            var document = new Document();
            List<Vector> c = new List<Vector>();
            foreach (GeoCoordinate g in WP)
            {
                Vector v = new Vector(g.Latitude, g.Longitude, g.Altitude / 3281);
                c.Add(v);
            }
            LineString line = new LineString();
            CoordinateCollection coleccion = new CoordinateCollection(c);
            line.Coordinates = coleccion;
            LineStyle estilo = new LineStyle();
            estilo.Width = 3;
            Style estilos = new Style();
            estilos.Line = estilo;
            document.AddStyle(estilos);

            var placemark = new Placemark
            {
                Name = "Ruta",
                Geometry = line
            };

            kml.Feature = placemark;
            var serializer = new Serializer();
            serializer.Serialize(kml);
            StreamWriter sw = new StreamWriter(fichero);
            sw.Write(serializer.Xml);
            sw.Close();
            KmlFile kmlF = KmlFile.Create(kml, false);
            return kmlF;
        }
    }
}
```

### A.2.6. Metar_

```
using System;
```

```
using Newtonsoft.Json.Linq;

namespace Lib
{
    public class Metar_
    {
        string metar;
        double presion;
        string nubes;
        string basenubes;
        double TpuntorocioC;
        double humedad_porcentaje;
        double T;
        double vism;
        double viento_dir;
        double viento_v;
        string fecha;
        string hora;
        bool viento = true;

        // Constructor a partir de Array JSON
        public Metar_(JArray datos)
        {
            presion = Convert.ToDouble(datos[0]["barometer"]["hpa"]);
            nubes = Convert.ToString(datos[0]["clouds"][0]["code"]);
            if ((nubes == "FEW") || (nubes == "SCT") || (nubes == "BKN") || (nubes
== "OVC"))
                basenubes = Convert.ToString(datos[0]["clouds"][0]["base_feet_agl"]);
            TpuntorocioC = Convert.ToDouble(datos[0]["dewpoint"]["celsius"]);
            humedad_porcentaje =
Convert.ToDouble(datos[0]["humidity"]["percent"]);
            metar = datos[0]["raw_text"].ToString();
            T = Convert.ToDouble(datos[0]["temperature"]["celsius"]);
            vism = Convert.ToDouble(datos[0]["visibility"]["meters_float"]);
            try
            {
                viento_dir = Convert.ToDouble(datos[0]["wind"]["degrees"]);
                viento_v = Convert.ToDouble(datos[0]["wind"]["speed_kts"]);
            }
            catch
            {
                viento = false;
            }
            fecha = Convert.ToString(datos[0]["observed"]);
            string[] f = fecha.Split('T');
            string[] f2 = f[0].Split('-');
            fecha = f2[2] + "-" + f2[1] + "-" + f2[0];
            hora = f[1].Replace('Z', ' ');
        }
```

```
// Gets
public string GetMetar()
{
    return metar;
}

public double GetPresion()
{
    return presion;
}

public string GetNubes()
{
    return nubes;
}

public double GetRocio()
{
    return TpuntorocioC;
}

public double GetHum()
{
    return humedad_porcentaje;
}

public double GetTemp()
{
    return T;
}

public double GetVis()
{
    return vism;
}

public double GetVientoDir()
{
    return viento_dir;
}

public double GetVientoVel()
{
    return viento_v;
}

public string GetFecha()
{
    return fecha;
}
```

```
        public string GetHora()
        {
            return hora;
        }

        public string GetBaseNubes()
        {
            return basenubes;
        }

        public bool GetHaceviento()
        {
            return viento;
        }

        // Decodificar código de nubes
        public string GetNubesDecodificado(string n)
        {
            string decodificado = n;
            if (n == "SKC")
                decodificado = "Cielo despejado de nubes";
            if (n == "FEW")
                decodificado = "Nubes escasas (entre 1 y 2 octas)";
            if (n == "SCT")
                decodificado = "Nubes dispersas (entre 3 y 4 octas)";
            if (n == "BKN")
                decodificado = "Cielo quebradizo, nubosidad abundante (entre 5 y 7
octas)";
            if (n == "OVC")
                decodificado = "Cielo totalmente cubierto";
            if (n == "TCU")
                decodificado = "Desarrollándose Cb";
            if (n == "CB")
                decodificado = "Cumulonimbos";
            if (n == "CAVOK")
                decodificado = "Techo y visibilidad OK";
            return decodificado;

        }

    }
}
```

### A.2.7. Peticiones

```
namespace Lib
{
```

```
    public class Peticiones
    {
        // Peticiones a API de elevación de Google

        public class GoogleElevCodeResponse
        {
            public string status { get; set; }
            public resultsElev[] results { get; set; }
        }

        public class resultsElev
        {
            public string elevation { get; set; }
            public locationElev location { get; set; }
            public string resolution { get; set; }
        }

        public class locationElev
        {
            public string lat { get; set; }
            public string lng { get; set; }
        }
    }
}
```

### A.2.8. Radioayuda

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Device.Location;
using SharpKml.Engine;
using SharpKml.Dom;
using System.IO;


namespace Lib
{
    public class Radioayuda
    {
        string ID;
        string nombre;
        string tipo;
        double frec;
        GeoCoordinate ubicacion;

        // Constructores
        public Radioayuda()
```

```
        { }

        public Radioayuda(string id, string n, string t, GeoCoordinate u, double f)
        {
            ID = id;
            nombre = n;
            tipo = t;
            ubicacion = u;
            frec = f;
        }

        // Gets
        public string GetIDNavaid()
        {
            return ID;
        }

        public string GetNombreNavaid()
        {
            return nombre;
        }

        public string GettipoNavaid()
        {
            return tipo;
        }

        public GeoCoordinate GetCoorNavaid()
        {
            return ubicacion;
        }

        public double GetfrecNavaid()
        {
            return frec;
        }


        // Leer archivo kml de radioayudas
        public List<Radioayuda> LeerRadioayudas(string fichero)
        {
            List<Radioayuda> navaids = new List<Radioayuda>();
            TextReader r = File.OpenText(fichero);
            KmlFile f = KmlFile.Load(r);
            Kml _kml = f.Root as Kml;

            IEnumerable<SharpKml.Dom.Placemark> placemarks =
_kml.Flatten().OfType<SharpKml.Dom.Placemark>();
            foreach (var placemark in placemarks)
            {
```

```
            Radioayuda temp = new Radioayuda();
            temp.ID = placemark.Name;
            Point p = placemark.Geometry as SharpKml.Dom.Point;
            temp.ubicacion = new GeoCoordinate(p.Coordinate.Latitude,
p.Coordinate.Longitude);
            if (placemark.Description != null)
            {
                string descripcion = placemark.Description.Text;
                var doc = new HtmlAgilityPack.HtmlDocument();
                doc.LoadHtml(descripcion);
                var node =
doc.DocumentNode.SelectSingleNode("/html/body/table/tr/td/TIPO");
                var nodes =
doc.DocumentNode.SelectNodes("/html/body/table/tr/td");
                foreach (var a in nodes)
                {
                    if (a.InnerText == "NAME_TXT")
                        temp.nombre = a.NextSibling.InnerText;
                    if (a.InnerText == "TIPO")
                        temp.tipo = a.NextSibling.InnerText;
                    if ((a.InnerText == "FREQUENCY_VAL") &&
(a.NextSibling.InnerText != "Null"))
                        temp.frec =
Convert.ToDouble(a.NextSibling.InnerText.Replace(",", "."));
                }
            }
            navaids.Add(temp);
        }
        return navaids;
    }
  }
}
```

### A.2.9. RutaAer

```
using GMap.NET;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Device.Location;
using System.Linq;
using System.Net;
using System.IO;
using System.Drawing;


namespace Lib
{
```

```
public class RutaAer
{
    string nombrefichero;
    string hora;
    string muestras;
    string fecha;
    double combtot;
    double t;
    double puntuacion;
    List<AD> ADsel;
    List<GeoCoordinate> WP = new List<GeoCoordinate>();
    List<GeoCoordinate> coordenadasterreno = new List<GeoCoordinate>();
    List<double> Dist = new List<double>();
    List<double> comblist = new List<double>();
    // Peticiones
    const string apiKey = [redacted];
    static string baseUrlELP =
"https://maps.googleapis.com/maps/api/elevation/json?path=";
    static string plusUrl = "&key=" + apiKey;


    // Constructores
    public RutaAer()
    { }

    public RutaAer(List<PointLatLng> wp)
    {
        foreach (PointLatLng a in wp)
        {
            GeoCoordinate g = new GeoCoordinate();
            g.Latitude = a.Lat;
            g.Longitude = a.Lng;
            WP.Add(g);
        }

    }

    // Sets y gets
    public void SetPuntuacion(double p)
    {
        puntuacion = p;
    }

    public void SetADsel(List<AD> a)
    {
        ADsel = a;
    }

    public string Getnombrefichero()
    {
```

```
        return nombrefichero;
    }

    public List<AD> GetADsel()
    {
        return ADsel;
    }

    public double GetPuntuacion()
    {
        return puntuacion;
    }
    public string GetFecha()
    {
        return fecha;
    }

    public double GetComb()
    {
        return combtot;
    }

    public double Gett()
    {
        return t;
    }

    public List<GeoCoordinate> GetWP()
    {
        return WP;
    }

    public string GetHora()
    {
        return hora;
    }

    public List<double> GetComblist()
    {
        return comblist;
    }

    public List<GeoCoordinate> GetCoordenadasTerreno(string resultado)
    {
        string[] lista = resultado.Split('\r');
        foreach (string a in lista)
        {
            if (!String.Equals(a, "") && !String.Equals(a, "\n"))
            {
                string[] b = a.Split(' ');
```

```
                double latitud = Convert.ToDouble(b[1]);
                double longitud = Convert.ToDouble(b[3]);
                double elevacion = Convert.ToDouble(b.Last());
                GeoCoordinate g = new GeoCoordinate(latitud, longitud);
                g.Altitude = elevacion;
                coordenadasterreno.Add(g);
            }
        }
        return coordenadasterreno;
    }

    // Devuelve WP con las altitudes mínimas de vuelo
    public List<GeoCoordinate> GetWPconAltMin()
    {
        int j = 0;
        int i = 0;
        double d = Dist[j] / 1852;
        double altmax = 0;
        while (i < coordenadasterreno.Count)
        {
            double posicion = (i + 1) / 1.852;
            double altitud = coordenadasterreno[i].Altitude * 3.28084;
            if (altitud > altmax)
                altmax = altitud;
            if ((d < posicion) && (j < WP.Count - 2))
            {
                WP[j].Altitude = Math.Ceiling((altmax + 1000) / 100) * 100; //
+1000ft requeridos de separacion. Redondeo a centenas
                altmax = 0;
                j++;
                d += Dist[j] / 1852;

            }
            i++;
        }
        WP[j].Altitude = Math.Ceiling((altmax + 1000) / 100) * 100;

        return WP;
    }

    // Devuelve lista con latitudes y longitudes de los WP
    public List<PointLatLng> GetWPLatLon()
    {
        List<PointLatLng> LatLon = new List<PointLatLng>();
        foreach (GeoCoordinate g in WP)
        {
            PointLatLng a = new PointLatLng(g.Latitude, g.Longitude);
            LatLon.Add(a);
        }
        return LatLon;
```

```
        }

        // Devuelve WP con la altura modificada por el rumbo
        public List<GeoCoordinate> GetWPconRumbo(List<GeoCoordinate>
sinrumbo, List<double> rumbo)
        {
            int i = 0;
            while (i < WP.Count - 1)
            {
                char[] cifras = Convert.ToString(sinrumbo[i].Altitude).ToCharArray();
                int unidad = (int)Char.GetNumericValue(cifras[cifras.Length - 1]);
                int decena = (int)Char.GetNumericValue(cifras[cifras.Length - 2]);
                int centena = (int)Char.GetNumericValue(cifras[cifras.Length - 3]);
                int millar = 0;
                if (cifras.Length > 3)
                    millar = (int)Char.GetNumericValue(cifras[cifras.Length - 4]);
                int decenasmillar = 0;
                if (cifras.Length == 5)
                    decenasmillar = (int)Char.GetNumericValue(cifras[cifras.Length -
5]);

                if (centena > 5) // if para poner las centenas a 5 (por arriba) - normas
VFR
                {
                    centena = 5;
                    millar++;
                    if (millar == 10)
                    {
                        millar = 0;
                        decenasmillar++;
                    }
                }
                if (centena < 5)
                    centena = 5;

                if (((rumbo[i] > 90) && (rumbo[i] < 271)) && (millar % 2 == 0)) // si va
de 91º a 270º el millar ha de ser impar
                {
                    millar++;
                    if (millar == 10)
                    {
                        millar = 0;
                        decenasmillar++;
                    }

                }

                if (((rumbo[i] > 270) || (rumbo[i] < 91)) && (millar % 2 != 0)) // si va de
271º a 90º el millar ha de ser par
                {
```

```
                millar++;
                if ((cifras.Length == 5) && (millar == 10))
                {
                    millar = 0;
                    decenasmillar++;
                }
            }

            double vuelorumbo =
Convert.ToDouble(string.Concat(decenasmillar.ToString() + millar.ToString() +
centena.ToString() + decena.ToString() + unidad.ToString()));
            WP[i].Altitude = vuelorumbo;
            WP[i].Course = rumbo[i];
            i++;

        }
        return WP;
    }

    // Duvuelve los espacios aéreos contenedores
    public List<List<EspacioAer>>
GetEspaciosAerContenedores(List<GeoCoordinate> recorrido,
List<EspacioAer> espaertotal)
    {
        List<List<EspacioAer>> lista = new List<List<EspacioAer>>();
        int j = 0;
        int i = 0;
        while (j < WP.Count - 1)
        {
            List<EspacioAer> espaer = new List<EspacioAer>();
            while (i < Convert.ToInt32(muestras) * (j + 1) / WP.Count())
            {
                foreach (EspacioAer e in espaertotal)
                {
                    GeoCoordinate g = new GeoCoordinate(recorrido[i].Latitude,
recorrido[i].Longitude);
                    bool resultado = EstaContenido(g, e.GetCoordenadas());

                    if (resultado)
                        if (!espaer.Contains(e))  // si está en el polígono y no está ya
en el espacioaer se añade a la lista
                            espaer.Add(e);
                }
                i++;
            }
            lista.Add(espaer);
            j++;
        }
        return lista;
    }
```

```
        // Devuelve los WP con las altitudes teniendo en cuenta los espacios
aéreos que se cruzan
        public List<GeoCoordinate> GetWPconEspAer(List<List<EspacioAer>>
espacioscontenedores)
        {
            int i = 0;
            while (i < WP.Count - 1)
            {
                if (espacioscontenedores[i] != null)
                {
                    espacioscontenedores[i].OrderBy(EspacioAer =>
EspacioAer.GetLimArriba());
                    double limabajomax = 0;
                    foreach (EspacioAer a in espacioscontenedores[i])
                    {
                        double limiteabajo = a.GetLimAbajo();
                        double limitearriba = a.GetLimArriba();
                        if (String.Equals(a.GetTipoAbajo(), "AGL"))
                            limiteabajo -= WP[i].Altitude;
                        if (String.Equals(a.GetTipoArriba(), "AGL"))
                            limitearriba -= WP[i].Altitude;
                        if ((WP[i].Altitude > limiteabajo) && (WP[i].Altitude < limitearriba))
                        {
                            limabajomax = limiteabajo;
                            if (String.Equals(a.GetClase(), "CLASS A") ||
String.Equals(a.GetClase(), "CLASS X"))
                            {
                                bool rumbopar = false;
                                char[] cifrasantes =
Convert.ToString(WP[i].Altitude).ToCharArray();
                                int cifrarumbo =
(int)Char.GetNumericValue(cifrasantes[cifrasantes.Length - 4]);
                                if (cifrarumbo % 2 == 0)
                                    rumbopar = true;
                                char[] cifras = Convert.ToString(limitearriba).ToCharArray();
                                int centena =
(int)Char.GetNumericValue(cifras[cifras.Length - 3]);
                                int millar = (int)Char.GetNumericValue(cifras[cifras.Length -
4]);
                                int decenasmillar = 0;
                                if (cifras.Length == 5)
                                    decenasmillar =
(int)Char.GetNumericValue(cifras[cifras.Length - 5]);
                                if (centena != 5)
                                {
                                    centena = 5;
                                    if (((millar % 2 == 0) && rumbopar) || ((millar % 2 != 0) &&
!rumbopar))
```

```
                            millar += 2; //sumando 2 conservamos la altura por
rumbo
                        if (((millar % 2 != 0) && rumbopar) || ((millar % 2 == 0) &&
!rumbopar)) //cuando no coincide el rumbo con la unidad de mil, se suma 1
                            millar++;

                    }
                    if (centena == 5)
                    {
                        if (((millar % 2 == 0) && rumbopar) || ((millar % 2 != 0) &&
!rumbopar))
                            millar += 2; //sumando 2 conservamos la altura por
rumbo
                        if (((millar % 2 != 0) && rumbopar) || ((millar % 2 == 0) &&
!rumbopar)) //cuando no coincide el rumbo con la unidad de mil, se suma 1
                            millar++;
                    }
                    if (millar > 10)
                    {
                        char[] c = Convert.ToString(millar).ToCharArray();
                        millar = Convert.ToInt32(c.Last());
                        decenasmillar++;
                    }
                    double alt =
Convert.ToDouble(string.Concat(decenasmillar.ToString() + millar.ToString() +
centena.ToString() + "0" + "0"));
                    WP[i].Altitude = alt;
                }
            }
        }
    }
    i++;
}
return WP;
}

// Petición de elevaciones del terreno
public string PathElevation(string muestras)
{
    this.muestras = muestras;
    string coordenadastotal = "";
    for (int i = 0; i < WP.Count; i++)
    {
        GeoCoordinate a = WP[i];
        coordenadastotal = coordenadastotal + a.Latitude + "," + a.Longitude
+ "|";
        if (i == WP.Count - 1)
            coordenadastotal = coordenadastotal.TrimEnd('|');
    }
```

```
            var json = new WebClient().DownloadString(baseUrlELP +
coordenadastotal + "&samples=" + muestras + plusUrl);
            Peticiones.GoogleElevCodeResponse jsonResult =
JsonConvert.DeserializeObject<Peticiones.GoogleElevCodeResponse>(json);
            string status = jsonResult.status;
            string geoElevation = String.Empty;
            geoElevation += Environment.NewLine;

            if (status == "OK")
            {
                for (int i = 0; i < jsonResult.results.Length; i++)
                {
                    if (Convert.ToDouble(jsonResult.results[i].elevation) < 0) //en el
mar, modifica las elevaciones negativas por 0
                        jsonResult.results[i].elevation = "0";
                    geoElevation += "Lat/Lng[" + i + "]: " +
jsonResult.results[i].location.lat + " / " + jsonResult.results[i].location.lng + "
Elevation[" + i + "]: " + jsonResult.results[i].elevation + Environment.NewLine;
                }
                return geoElevation;
            }

            else
                return "error";
        }


        // Calcula distancias
        public List<double> CalculaDist()
        {
            int i = 0;
            while (i < WP.Count() - 1)
            {
                double d = WP[i].GetDistanceTo(WP[i + 1]);
                Dist.Add(d);
                i++;
            }
            return Dist;
        }

        // Calcula rumbos
        public List<double> CalculaRumbo()
        {
            List<double> Rumbo = new List<double>();
            int i = 0;
            while (i < this.WP.Count() - 1)
            {
                double lat1 = WP[i].Latitude * Math.PI / 180;
                double lat2 = WP[i + 1].Latitude * Math.PI / 180;
                double lon1 = WP[i].Longitude * Math.PI / 180;
```

```
            double lon2 = WP[i + 1].Longitude * Math.PI / 180;
            double x = Math.Cos(lat2) * Math.Sin(lon2 - lon1);
            double y = Math.Cos(lat1) * Math.Sin(lat2) - Math.Sin(lat1) *
Math.Cos(lat2) * Math.Cos(lon2 - lon1);
            double angulo = Math.Atan2(x, y) * 180 / Math.PI;
            if (angulo < 0)
                angulo = 360 + angulo;
            Rumbo.Add(angulo);
            i++;
        }
        return Rumbo;
    }

    // Se comprueba que el punto esté dentro del espacio aéreo
    public bool EstaContenido(GeoCoordinate g, List<GeoCoordinate>
vertices)
    {
        bool resultado = false;
        var a = vertices.Last();
        foreach (var b in vertices)
        {
            if ((b.Latitude == g.Latitude) && (b.Longitude == g.Longitude))
                resultado = true;

            if ((b.Longitude == a.Longitude) && (g.Longitude == a.Longitude) &&
(a.Latitude <= g.Latitude) && (g.Latitude <= b.Latitude))
                resultado = true;

            if ((b.Longitude < g.Longitude) && (a.Longitude >= g.Longitude) ||
(a.Longitude < g.Longitude) && (b.Longitude >= g.Longitude))
                if (b.Latitude + (g.Longitude - b.Longitude) / (a.Longitude -
b.Longitude) * (a.Latitude - b.Latitude) <= g.Latitude)
                    resultado = !resultado;
            a = b;
        }
        return resultado;
    }

    // Calcula las performances (combustible, velocidades) según la ruta y las
altitudes
    public List<List<double>> CalcularPerformances(Avion avion,
List<GeoCoordinate> RecorridoAvion, List<AD> AeropuertosSeleccionados)
    {
        List<List<double>> performances_total = new List<List<double>>();
        List<double> GPH = new List<double>();
        List<double> GS = new List<double>();
        List<double> performance_ascenso = new List<double>();
        List<double> performance_descenso = new List<double>();
        List<double> performance_crucero = new List<double>();
        int i = 0;
```

```
            while (i < RecorridoAvion.Count)
            {
                if (i == 0)
                {
                    List<double> ascenso =
avion.GetClimbPerformance(RecorridoAvion[i].Altitude);
                    List<double> aerodromo =
avion.GetClimbPerformance(AeropuertosSeleccionados[0].GetCoorAD().Altitud
e);
                    int j = 0;
                    while (j < ascenso.Count)
                    {
                        performance_ascenso.Add(ascenso[j] - aerodromo[j]);
                        j++;
                    }
                    performances_total.Add(performance_ascenso);
                }
                if (i == RecorridoAvion.Count - 1)
                {
                    List<double> descenso =
avion.GetClimbPerformance(RecorridoAvion[i].Altitude);
                    List<double> aerodromo =
avion.GetClimbPerformance(AeropuertosSeleccionados[1].GetCoorAD().Altitud
e);
                    int j = 0;
                    while (j < descenso.Count)
                    {
                        performance_descenso.Add(aerodromo[j] - descenso[j]);
                        j++;
                    }
                    performances_total.Add(performance_descenso);
                }
                else
                {
                    performance_crucero =
avion.GetCruceroPerformance(RecorridoAvion[i].Altitude);
                    RecorridoAvion[i].Speed = performance_crucero[1];
                    GS.Add(RecorridoAvion[i].Speed);
                    GPH.Add(performance_crucero[2]);
                    performances_total.Add(performance_crucero);
                }
                i++;
            }
            return performances_total;
        }

        //Calcular la performance (velocidades, combustible) a aeropuerto
alternativo
        public List<double> CalculaPerformanceAlt(Avion avion, List<AD> ADs)
        {
```

```
            // Consideramos peor caso al aeropuerto alternativo
            List<double> performance = new List<double>();
            double alturaADmax = ADs[ADs.Count() - 1].GetCoorAD().Altitude;
            if (ADs[ADs.Count() - 2].GetCoorAD().Altitude > alturaADmax)
                alturaADmax = ADs[ADs.Count() - 2].GetCoorAD().Altitude;
            performance = avion.GetCruceroPerformance(alturaADmax); //0: alt, 1:
KTAS, 2: GPH
            return performance;

        }

        // Calcula el tiempo entre WP
        public List<double> CalculaTiempo(List<double> Dist, List<double> GS)
        {
            List<double> T = new List<double>();
            int i = 0;
            while (i < WP.Count() - 1)
            {
                double velocidad = GS[i] * 1852 / 3600; //kt = NM/h
                double d = Dist[i];
                double t = d / velocidad;
                T.Add(t);
                i++;
            }

            return T;
        }

        // Calcula si el punto está dentro del polígono de la gráfica de
PesoyBalance. Fuente: https://stackoverflow.com/questions/4243042/c-sharp-
point-in-polygon
        public bool PuntoEstaContenido(Point p, PointF[] vertices)
        {
            var intersects = new List<int>();
            var a = vertices.Last();
            foreach (var b in vertices)
            {
                if (b.X == p.X && b.Y == p.Y)
                    return true;
                if (b.X == a.X && p.X == a.X && p.X >= Math.Min(a.Y, b.Y) && p.Y <=
Math.Max(a.Y, b.Y))
                    return true;
                if (b.Y == a.Y && p.Y == a.Y && p.X >= Math.Min(a.X, b.X) && p.X <=
Math.Max(a.X, b.X))
                    return true;
                if ((b.Y < p.Y && a.Y >= p.Y) || (a.Y < p.Y && b.Y >= p.Y))
                {
                    var px = (int)(b.X + 1.0 * (p.Y - b.Y) / (a.Y - b.Y) * (a.X - b.X));
                    intersects.Add(px);
                }
```

```
            a = b;
        }
        intersects.Sort();
        return intersects.IndexOf(p.X) % 2 == 0 || intersects.Count(x => x < p.X)
% 2 == 1;
    }

    // Guardar ruta para posteriores cargas en FlightTrack
    public int GuardarRuta(string fichero, List<GeoCoordinate> WP, List<AD>
Aeropuertos, string fecha, double combtot, double ttotal, double puntuacion,
List<double> comblist, string hora)
    {
        try
        {
            StreamWriter F = new StreamWriter(fichero);
            nombrefichero = fichero;
            F.WriteLine("***RUTA***");

            if (fecha != null)
                F.WriteLine("Fecha: " + fecha);
            else
                F.WriteLine("Fecha: sin concretar");
            if (hora != null)
                F.WriteLine("Hora: " + hora);
            else
                F.WriteLine("Hora: sin concretar");
            F.WriteLine("Puntuación (sobre 10): " + puntuacion);
            F.WriteLine("Tiempo: " + ttotal / 60 + " min");
            if (combtot != 0)
            {
                F.WriteLine("Combustible requerido: " + combtot);
                foreach (double d in comblist)
                    F.WriteLine(d);
            }
            else
                F.WriteLine("Combustible requerido: no guardado");
            F.WriteLine("WP");
            if (WP.Count == 0)
                return 1;
            foreach (GeoCoordinate g in WP)
                F.WriteLine(Convert.ToString(g.Latitude) + "\t" +
Convert.ToString(g.Longitude) + "\t" + Convert.ToString(g.Altitude));
            F.WriteLine("Aeropuertos");
            foreach (AD a in Aeropuertos)
                F.WriteLine(a.GetNombreAD());
            F.Close();
            return 0;
        }
        catch
        {
```

```
            return 1;
        }
    }

    // Cargar ruta desde fichero guardado
    public List<AD> CargarRuta(string fichero, List<AD> lista)
    {
        List<AD> aeropuertos = new List<AD>();
        StreamReader F = new StreamReader(fichero);
        nombrefichero = fichero;
        F.ReadLine();
        string linea = F.ReadLine();
        string[] trozos = linea.Split(':');
        if (trozos[1] != " sin concretar")
            fecha = trozos[1];
        linea = F.ReadLine();
        trozos = linea.Split(':');
        if ((trozos[1] != " sin concretar") && (trozos[1] != " "))
            hora = trozos[1] + ":" + trozos[2];
        linea = F.ReadLine();
        trozos = linea.Split(':');
        if (Convert.ToDouble(trozos[1]) != 0)
            puntuacion = Convert.ToDouble(trozos[1]);
        linea = F.ReadLine();
        trozos = linea.Split(':');
        t = Convert.ToDouble(trozos[1].Replace("min", "")) * 60;
        linea = F.ReadLine();
        trozos = linea.Split(':');
        if (!String.Equals(trozos[1], " no guardado"))
        {
            combtot = Convert.ToDouble(trozos[1]);
            linea = F.ReadLine();
            while (linea != "WP")
            {
                comblist.Add(Convert.ToDouble(linea));
                linea = F.ReadLine();
            }
            linea = F.ReadLine();
        }
        else
        {
            F.ReadLine();
            linea = F.ReadLine();
        }
        while (linea != "Aeropuertos")
        {

            trozos = linea.Split('\t', ',');
            GeoCoordinate g = new GeoCoordinate(Convert.ToDouble(trozos[0]),
Convert.ToDouble(trozos[1]), Convert.ToDouble(trozos[2]));
```

```
            WP.Add(g);

            linea = F.ReadLine();
        }
        while (linea != "Aeropuertos")
        {
            trozos = linea.Split('\t', ',');
            GeoCoordinate g = new GeoCoordinate(Convert.ToDouble(trozos[0]),
Convert.ToDouble(trozos[1]), Convert.ToDouble(trozos[2]));
            WP.Add(g);

            linea = F.ReadLine();
        }
        linea = F.ReadLine();
        while (linea != null)
        {
            AD a = new AD();
            a = a.BuscaAD(lista, Convert.ToString(linea));
            aeropuertos.Add(a);
            linea = F.ReadLine();
        }
        F.Close();
        return aeropuertos;

    }

  }
}
```