UNIVERSITAT POLITÈCNICA DE VALÈNCIA

_ TELECOM ESCUELA TÉCNICA VLC SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería de Telecomunicación

Desarrollo de una aplicación software para la gestión de la separación de aeronaves no tripuladas (UAS) en el espacio aéreo mediante la provisión de servicios U-space

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Telecomunicación

AUTOR/A: Ferrando Navarro, Andrés

Tutor/a: Balbastre Tejedor, Juan Vicente

Cotutor/a: Vico Navarro, Joaquín

CURSO ACADÉMICO: 2022/2023

Agradecimientos

Este trabajo está dedicado a todas aquellas personas especiales que me han ayudado a lo largo del máster y, por supuesto, durante la elaboración de este proyecto.

Como no podría ser de otra forma, gracias a mis padres Juan Andrés e Isabel por todo el apoyo, cariño, ánimo y comprensión que siempre me han demostrado. También me gustaría realizar una mención especial para mi novia Elena por estar siempre a mi lado sufriendo mis horarios de trabajo.

Gracias a mis familiares y amigos por la paciencia infinita y haber estado ahí para apoyarme en los momentos más difíciles. Especialmente, quiero mencionar a Alejandro y Alex por ser los dos grandes compañeros que me han ayudado en todos los trabajos de la universidad.

A mi tío Juan por su increíble e incansable dedicación para enseñar y aconsejarme en las distintas etapas de mi vida académica.

Por último, y súper importante, muchas gracias al todo el grupo de investigación *SNA* por acogerme en su proyecto. En particular, al tutor de este TFM Juan Vicente Balbastre y al cotutor Joaquín Vico, por haber tenido la paciencia de ayudarme a superar los múltiples obstáculos que he encontrado durante el desarrollo de este proyecto.

Muchas gracias de corazón.

Resumen

En este Trabajo Fin de Máster se ha realizado una aplicación Android para el manejo de aeronaves no tripuladas (*UAS*) en el entorno definido por la *SESAR-JU*: *U-space*. En el marco del proyecto europeo *BUBBLES* y a partir del kit de desarrollo *software* de la empresa *DJI*, se ha realizado el diseño de la arquitectura de la interfaz, cuyo objetivo es dotar a la aplicación de robustez y agilidad, a través de la plataforma denominada *U-TraC*. Posteriormente, se ha expuesto la evolución de la aplicación a lo largo de tres versiones, en las cuales se describe el funcionamiento interno y el comportamiento de la interfaz gráfica ante las interacciones del usuario. Además, mediante dos pruebas de vuelo, los pilotos implicados emplearon una serie de formularios para evaluar la plataforma y ofrecer sugerencias de mejora acerca de las características y operatividad de esta. Finalmente, se ha mostrado el resultado definitivo de la aplicación, cuyo diseño gráfico ha sido basado en los Factores Humanos.

Resum

En aquest Treball Fi de Màster s'ha realitzat una aplicació *Android* per al maneig d'aeronaus no tripulades (*UAS*) en l'entorn definit per la *SESAR-JU*: *U-space*. Dins el marc del projecte europeu *BUBBLES* i a partir del kit de desenvolupament *software* de l'empresa *DJI*, s'hi ha realitzat el disseny de l'arquitectura de la interfície, l'objectiu de la qual és dotar a l'aplicació de robustesa i agilitat, a través de la plataforma denominada *U-TraC*. Posteriorment, s'ha exposat l'evolució de l'aplicació al llarg de tres versions, en les quals s'hi descriu el funcionament intern i el comportament de la interfície gràfica davant les interaccions de l'usuari. A més, mitjançant dues proves de vol, els pilots implicats van emprar una sèrie de formularis per avaluar l'aplicació i oferir-ne suggeriments de millora sobre les característiques i operativitat. Finalment, s'hi ha mostrat el resultat definitiu de l'aplicació, el disseny gràfic de la qual ha sigut basat en els Factors Humans.

Abstract

In this Master's Final Project, an Android application has been developed for unmanned aircraft (UAS) management in the SESAR-JU defined environment: U-space. Under BUBBLES european project and based on DJI's software development kit, the design of the application architecture has been carried out, whose objective is to provide the application with robustness and agility, through U-TraC platform. Subsequently, the application's evolution over three versions has been presented, describing internal operation and behaviour to user interactions. Furthermore, by means of two flight tests, pilots involved used a series of forms to evaluate the application and offer suggestions for improvement regarding its features and operability. Finally, application's outcome, whose graphic design has been based on Human Factors, has been shown.

Índice general

Índice de figuras

Índice de tablas

Listado de siglas empleadas

ATM	Air Traffic Management
CPA	Closest Point of Approach
CTR	Controlled Traffic Region
DJI	Da-Jiang Innovations
EASA	European Union Aviation Safety Agency
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
IC	Imminent Collision
IDE	Integrated Development Environment
IMU	Inertial Measurement Unit
IoT	Internet of Things
JSON	JavaScript Object Notation
KML	Keyhole Markup Language
M2M	Machine to Machine
MQTT	Message Queuing Telemetry Transport
MTOW	Maximun Take-Off Weight
NMAC	Near Mid-Air Collision
ODS	Objetivos de Desarrollo Sostenible
QoS	Quality of Service
SAIL	Specific Assurance and Integrity Level
SDK	Software Development Kit

SESAR	Single European Sky ATM Research
SES	Single European Sky
SL	Separation Loss
SORA	Specific Operations Risk Assessment
TC	Tactical Conflict
UAS	Unmanned Aerial System
UE	Unión Europea
UTM	Unmanned Traffic Management
VLOS	Visual Line of Sight
XML	eXtensible Markup Language

Parte I

Memoria

Capítulo 1

INTRODUCCIÓN

1.1. Motivación

La gestión del espacio aéreo está sufriendo una profunda transformación desde el punto de vista de regulación y normativa debido a los desafíos que se derivan de la adopción masiva de los vehículos aéreos no tripulados (*UAS*). Se estima que el tamaño global de este mercado se situó en 3.420 millones de dólares en 2021 y se prevé que aumente a una tasa de crecimiento anual compuesta del 13,8 % entre 2022 y 2030 [1].

De hecho, a finales del año 2022, la Comisión Europea adoptaba la Estrategia Europea de Drones 2.0 [2], donde establece una visión para el futuro desarrollo del mercado europeo de drones. Se basa en el marco de seguridad de la Unión Europea (*UE*) para operar y establecer los requisitos técnicos de los drones, que es el más avanzado del mundo. En efecto, gracias al amplio entorno normativo de la *UE*, los drones han volado con seguridad durante cientos de miles de horas en los cielos europeos, por ejemplo para vigilar infraestructuras, controlar vertidos de petróleo o tomar muestras del suelo.

Uno de los mayores objetivos de la regulación europea aérea es evitar la colisión entre drones. Por tanto, la integración de los drones en el espacio aéreo exige la revisión de las normas de seguridad existentes para tener en cuenta las diferencias con los actuales actores o el desarrollo de normas totalmente distintas diseñadas específicamente para estos nuevos participantes.

Ante la oportunidad que presenta la mejora de las capacidades del espacio aéreo, surge este Trabajo Fin de Máster (TFM). Su propósito reside en desarrollar una interfaz *software* que permita al piloto gestionar el vuelo de los *UAS* en el entorno *U-space* de forma eficiente y automática, a través de la comunicación con la plataforma de gestión de separaciones *U-TraC*.

Esta interfaz se basa en el uso del kit de desarrollo de *software* (*SDK*) de la empresa *DJI*. *Da-Jiang Innovations*, más conocida como *DJI*, es el líder mundial en tecnología de drones, con cerca del 70 % de la cuota de mercado en todo el mundo. Profesionales del cine, agricultura, conservación, búsqueda y rescate, infraestructuras de energía y más confían en los productos de esta compañía para aportar nuevas perspectivas a su trabajo y ayudarles a lograr sus objetivos con mayor rapidez, seguridad y eficiencia [3].

1.2. Contexto

1.2.1. *SESAR Joint Undertaking*

El principal organismo encargado de diseñar el camino hacia la regulación para el despliegue y uso de *UAS* a nivel europeo es la *SESAR Joint Undertaking* (*SESAR-JU*) [4]. Se trata de la colaboración europea de carácter público-privado que gestiona, mediante la incorporación de nuevas tecnologías, la iniciativa de Cielo Único Europeo (*SES*, por sus siglas en inglés) [5]. Su financiación se distribuye del siguiente modo:

- *Horizon Europe*: 600 millones de euros
- *Eurocontrol*: hasta 500 millones de euros, con contribuciones en especie y financieras.
- Industria : 500 millones de euros como mínimo.

Por tanto, la asociación reúne a la Unión Europea, *Eurocontrol* y a más de 50 organizaciones que abarcan toda la cadena de valor de la aviación. En esta unión se pueden encontrar empresas que gestionan los aeropuertos nacionales como *Aena* o *Athens International Airport S.A*, proveedores de servicios de navegación aérea como *ENAIRE*, compañías de transporte aéreo como *Ryanair* o *easyJet*, operadores y proveedores de servicios de drones como *Drone Allince Europe*, industria manufacturera *(Airbus)*, comunidad científica y usuarios del espacio aéreo de todas las categorías.

La visión central de la asociación se apoya en aprovechar las últimas tecnologías digitales (*Soluciones SESAR*) para aumentar los niveles de automatización, intercambio de datos mediante el empleo de técnicas de ciberseguridad y conectividad en la gestión del tráfico aéreo. Además, sus objetivos se centran en permitir la prestación de servicios de tráfico en todos los tipos de espacio aéreo, incluidas las operaciones a muy baja y gran altitud, donde los drones jugarán un papel fundamental.

De este modo, estas tecnologías permiten que el sistema sea más escalable y ágil, al tiempo que refuerzan su resistencia a las perturbaciones, los cambios en la demanda de tráfico y la diversidad de vehículos aéreos. Todos estos atributos son fundamentales para que el sistema esté preparado para el futuro de forma inteligente y sostenible.

El Plan Maestro *ATM* Europeo [6] es la hoja de ruta acordada que conecta las actividades de investigación y desarrollo *ATM* con los escenarios de despliegue para alcanzar los objetivos de rendimiento del Cielo Único Europeo. La versión inicial de este plan, realizada en el año 2009, fue el resultado de la primera fase del proceso de definición del proyecto *SESAR* y constituye la base de las actividades de desarrollo y despliegue del mismo.

Desde entonces, dicho plan se actualiza periódicamente en estrecha colaboración con las partes interesadas para adaptarse a la realidad del contexto de las operaciones, realizándose la primera de ellas en el año 2012, la segunda en el 2015 y la más reciente en el 2020.

En la segunda actualización, a partir de la petición de la Comisión Europea, la *SESAR-JU* empezó a elaborar un proyecto acerca del uso de drones en el espacio aéreo de baja cota con el fin de que el desarrollo de esta nueva zona sea segura y eficiente, dando lugar al nacimiento del concepto *U-space*.

1.2.2. *U-space*

U-space es un conjunto de nuevos servicios basados en un alto nivel de digitalización y automatización de funciones, junto con procedimientos específicos diseñados para soportar el acceso seguro y eficiente al espacio aéreo de un gran número de drones [7].

Se trata de un marco habilitador diseñado para facilitar cualquier tipo de misión rutinaria, en todas las clases de espacio aéreo y en todo tipo de entornos, al tiempo que aborda una interfaz adecuada con la aviación tripulada y el control del tráfico aéreo.

En consecuencia, se deben establecer los pilares fundamentales sobre los que se sostiene dicho marco para cumplir con éxito la visión del proyecto. Estos principios son:

- Garantizar la seguridad de todos los usuarios del espacio aéreo que operan en el marco *U-space*, así como a las personas en tierra.
- Proporcionar un modelo escalable, flexible y adaptable que pueda responder a cambios en la demanda, el volumen, la tecnología, los modelos de negocio, etc.
- Permitir alta densidad de operaciones con múltiples drones automatizados bajo la supervisión de los operadores.
- Garantizar un acceso equitativo y justo al espacio aéreo para todos los usuarios.
- Permitir una prestación de servicios en todo momento, apoyando los modelos de negocio de los operadores de drones y minimizar los costes de despliegue y explotación aprovechando, en la medida de lo posible, los servicios ya existentes en la industria.
- Acelerar el despliegue adoptando tecnologías y normas de otros sectores cuando respondan a las necesidades del *U-space*.

Por otra parte, resulta esencial establecer los mecanismos que permitirán el desarrollo de este conjunto de reglas de forma eficiente. Precisamente, en busca de la eficiencia en la implementación de dicho entorno, se distinguen cuatro fases de desarrollo:

- *U1*: servicios básicos del espacio aéreo no tripulado que abarcan el registro electrónico, la identificación electrónica y la delimitación geográfica.
- *U2*: servicios iniciales *U-space* para la gestión de operaciones con drones, incluida la planificación de vuelos, la aprobación de vuelos, el seguimiento y la interconexión con el control del tráfico aéreo convencional.
- *U3*: servicios avanzados de *U-space* para operaciones más complejas en zonas con alta densidad de drones, como asistencia para la detección de conflictos y funciones automatizadas de detección y evasión.
- *U4*: servicios completos *U-space*, que ofrecen niveles muy altos de automatización, conectividad y digitalización tanto para el dron como para el sistema *U-space*.

1.2.3. *BUBBLES*

En la hoja de ruta propuesta por *SESAR-JU* para desarrollar el *U-space* de una forma eficiente nace el proyecto *BUBBLES* (*BUilding Basic BLocks for a U-space SEparation*). El objetivo principal de este proyecto reside en la definición de un concepto de operaciones (*ConOps*) para la gestión de la separación en el *U-space*, así como la identificación de los requisitos de seguridad y rendimiento de los servicios y sistemas implicados. En consecuencia, comprende la definición de horizontes de conflicto y modos y mínimos de separación aplicables, que se actualizan dinámicamente [8].

El escenario de referencia para el *ConOps BUBBLES* se corresponde a un nivel de desarrollo *U3*, en el que se tiene una densidad relativamente alta de operaciones no tripuladas que hacen necesario mitigar el riesgo de colisión en el aire mediante una serie de medidas de seguridad centradas alrededor de la provisión de separación.

La provisión de separación por parte del *U-space* se basa en sistemas de vigilancia cooperativos, recayendo la principal responsabilidad en los proveedores de servicio de *U-space* o los operadores de los drones.

Las características de los escenarios locales de operaciones en los que se realicen los vuelos pueden ser muy diversos, así como los *UAS* empleados en dichas operaciones, por lo que la gestión de separación es bastante compleja. Con el fin de establecer una taxonomía relacionada con el riesgo *BUBBLES* propone una distinción de hasta 10 clases de tráfico [9], desde *A1* hasta *IFR*. Cada una de las clases presenta unas especificaciones distintas acerca de las distancias mínimas de separación entre aeronaves, peso del dron o riesgo de colisión.

En este sentido, *BUBBLES* también ofrece una distinción entre los rangos posibles de conflicto, a través de la definición de cuatro volúmenes que se corresponden con la severidad del modelo de colisión [9]. El más leve de estos sería la detección táctica de conflicto mientras que el más severo sería el *NMAC* (*Near mid-air collision*). Tanto los volúmenes de separación como la clasificación del tráfico de los drones se detallarán en más profundidad en la arquitectura del proyecto.

A grandes rasgos, el servicio de gestión de la separación se centra en definir cómo debe aplicarse la disposición de separación a nivel táctico en *U-space*, de forma que todos los conflictos que surjan se resuelvan aplicando las mismas reglas y se alcance el nivel de seguridad establecido. En este sentido, la provisión de separación es un proceso iterativo que consta de cuatro pasos [10]:

1. Detección de conflictos.
2. Formulación de la solución.
3. Aplicación de la solución.
4. Supervisión de la solución.

Este proyecto ha sido coordinado por la Universidad Politécnica de Valencia y dirigido por el tutor de este mismo trabajo, Juan Vicente Balbastre Tejedor. El proyecto ha sido desarrollado en el entorno del instituto *ITACA* de la universidad, en concreto en el grupo de investigación *SNA* (*Air Navigation Systems research group*).

Además, se ha contado con el apoyo de la portuguesa *Universidade de Coimbra*, la italiana *Universita degli Studi di Roma La Sapienza*, la organización europea *Eurocontrol* y la multinacional española *Indra Sistemas SA*.

1.2.4. *U-TraC*

U-space Tracking & Conflict detection, *U-TraC*, es el sistema de gestión virtual encargado de centralizar y coordinar digitalmente todos los datos relativos a la operación de drones en una zona determinada. Permite la comunicación e intercambio de información entre los diferentes actores del sistema, así como una interconexión en tiempo real con el resto del sistema para facilitar la toma de decisiones y la resolución de conflictos.

El sistema lo componen principalmente tres grandes bloques: el *tracker*, el detector de conflictos y el servicio de información de tráfico. La operativa de la gestión de conflictos es lineal:

1. En primer lugar, el *tracker* recibe la información de telemetría provista desde la interfaz del piloto, la cual depende tanto de la posición del propio dron como de su situación de vuelo. La calidad de la información recibida dependerá de la degradación derivada del enlace descendente, lo cual puede incluir errores de posicionamiento dependientes de los sensores embarcados en el dron (generalmente *GNSS* + *IMU*) y latencias y pérdida de paquetes generados por el enlace de comunicación.

2. Una vez recibida la información, se transmite hacia la herramienta de monitorización de rendimiento que se encarga de calcular parámetros de desempeño para evaluar los efectos en el desarrollo de la vigilancia y del canal de comunicaciones. A partir de estos valores, el sistema realiza un cálculo de los criterios de separación mínima, enviando el resultado de nuevo hacia el detector de conflictos de *U-TraC*.

3. Tras recibir dicha información, el detector de conflictos se encarga de clasificar y ordenar por orden de severidad los conflictos que presentan los *UAS*, calculando las distancias entre los mismos y comparándolas con el criterio recibido. Si dichas distancias son inferiores, entonces se produce un conflicto y se notifica al servicio de información de tráfico.

4. Con la recepción de los parámetros asociados con el conflicto, el servicio de información se encarga de elaborar un mensaje, que incluye la posición del *UAS* en conflicto y una maniobra en la dimensión horizontal o vertical para resolver el conflicto. Esta alerta se envía a la interfaz del piloto para que pueda actuar en consecuencia.

Así pues, sintetizando el proceso anterior, las principales funcionalidades de la plataforma de gestión de la información *U-TraC* son:

- Monitorización en tiempo real de la situación de vuelo de todos los drones que estén en un mismo rango a través de *tracks* de seguimiento.

- Detección y resolución de conflictos, a partir de la información provista por la interfaz y el cálculo de separación mínima, que depende de varias condiciones.

- Procesión de la información de los conflictos y transmisión de la misma a la interfaz del usuario.
- Gestión de estructuras, datos aéreos y acceso al espacio aéreo, a través de herramientas de registro para su posterior procesado.

1.2.5. Resumen

En resumen, la interfaz ha sido desarrollada en el ámbito de un proyecto europeo que se financió bajo el paraguas de una iniciativa de colaboración público-privada, la cual ha desarrollado un plan para adaptar el espacio aéreo europeo a la operabilidad de los *UAS*.

Una de las partes de dicho plan consiste en gestionar la separación de las aeronaves, lo cual se consigue a través de una herramienta que emplea procesos de inteligencia artificial para realizar todos los cálculos y comunicarse con la interfaz.

La Figura 1.1 describe de forma gráfica el contexto en que se ha desarrollado la interfaz y, por ende, este proyecto.

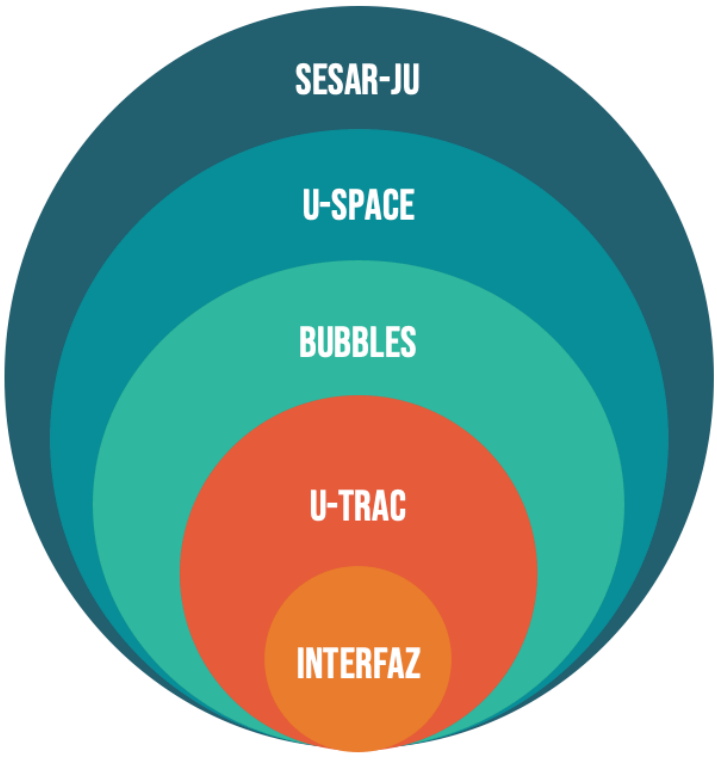


Figura 1.1: Contexto en el que se ha desarrollado la interfaz.

1.3. Organización de la memoria

La memoria se estructura de la siguiente forma:

- En el capítulo 2 se presentan los objetivos que se pretenden conseguir al realizar este TFM.
- En el capítulo 3 se presenta la metodología seguida en el TFM, centrada en los procesos realizados y su distribución temporal, así como la justificación de las herramientas empleadas para cumplir con los objetivos.
- En el capítulo 4 se detalla la arquitectura en la que se enmarca la interfaz.
- En el capítulo 5 se realiza una descripción detallada del método empleado para construir el *software* de la aplicación y se evalúan los resultados tras haber realizado las validaciones y pruebas del sistema.
- En el capítulo 6 se presentan las conclusiones y líneas futuras del TFM.

Capítulo 2

OBJETIVOS

2.1. Objetivo principal

El principal propósito de este Trabajo Fin de Máster es proveer a los pilotos de una interfaz que permita volar su dron en el marco del *U-space* mientras se envían datos de telemetría a la plataforma de gestión de separaciones *U-Trac*, a la vez que se recibe información relevante como tráfico cercano y alertas de conflictos tácticos, de acuerdo con las reglas establecidas en el *ConOps* de *BUBBLES*. La finalidad del sistema reside en transmitir y presentar dicha información de forma segura y clara para que el piloto pueda tomar las decisiones más oportunas dependiendo de la situación de vuelo.

Para conseguirlo, la aplicación deberá presentar características como utilidad y rapidez, permitiendo discernir con total claridad cuándo, dónde y cómo se pueden producir los conflictos de separación. Además, la aplicación deberá ser robusta, a prueba de fallos e imprevistos, puesto que sus operaciones se desarrollan en escenarios de actuación críticos.

2.2. Objetivos específicos

Para cumplir dicha meta, se requieren los siguientes objetivos específicos:

1. Diseñar la aplicación desde el punto de vista de Factores Humanos, con la intención de facilitar la interacción de los pilotos remotos con la aplicación, desde los puntos de vista de la comodidad, la eficiencia y la seguridad.
2. Establecer las conexiones necesarias con el resto de los elementos de la plataforma *U-TraC*, con el fin de recibir y enviar datos de telemetría para una representación fluida de la situación de tráfico.
3. Presentar la información de los conflictos procedente de *U-TraC*, incluyendo la severidad del conflicto, las distancias verticales y horizontales respecto a la posición del *UAS* en cuestión y sugiriendo una maniobra en la dimensión vertical para resolver el conflicto.

4. Configurar la aplicación para permitir tanto el vuelo manual como el vuelo automático de los *UAS*, mediante la carga de un plan de vuelo generado previamente. En determinados escenarios puede resultar de gran interés y utilidad crear un recorrido aéreo específico, de manera que el piloto remoto monitorice el seguimiento de la ruta a través de la interfaz, proporcionando los controles necesarios para pausar, detener y/o reanudar la ejecución.

5. Realizar pruebas de simulación en laboratorio mediante emuladores que permitan reproducir el comportamiento de los drones y su reacción al interaccionar con el *software* desarrollado.

6. Organizar pruebas de vuelo con distintos operadores para verificar el funcionamiento de la aplicación en escenarios reales y obtener retroalimentación acerca del funcionamiento de la interfaz, así como posibles sugerencias de mejora del diseño del sistema.

Capítulo 3

METODOLOGÍA DEL PROYECTO

Para la consecución de los objetivos de este trabajo, ha sido necesario seguir una metodología clara que permita avanzar en el proyecto de forma eficaz. En esta sección de la memoria se procederá a explicar el procedimiento llevado a cabo a partir de la descripción de las tareas realizas junto a su planificación temporal. Además, se justificará la selección de las herramientas empleadas a lo largo del proyecto para construir la interfaz.

3.1. Descripción de las tareas

En primer lugar, como en cualquier proyecto, se debe establecer una definición de las actividades e hitos para cumplir con los objetivos del diseño de la interfaz. A continuación, se describe el desglose de las tareas realizadas:

- Tarea 1: búsqueda bibliográfica y comprensión de la documentación usada, incidiendo en el contexto del proyecto y los lenguajes de programación para el desarrollo del sistema.
- Tarea 2: programación de los códigos fuente del sistema y establecimiento de las conexiones necesarias con el resto de elementos que componen la interfaz.
- Tarea 3: diseño de las escenas necesarias para el correcto funcionamiento de la interfaz, así como la adición de elementos que faciliten el uso del *software*.
- Tarea 4: primeras pruebas de validación del sistema mediante vuelos reales con el fin de probar a fondo las operaciones del sistema.
- Tarea 5: corrección de los posibles errores derivados de las pruebas y adaptación de la interfaz a las sugerencias de los usuarios.
- Tarea 6: realización de segundas pruebas de vuelo con distintos operadores para obtener posibles sugerencias de mejora de la interfaz.
- Tarea 7: implementación de las mejoras sugeridas para ajustar el comportamiento de la herramienta hacia una experiencia más satisfactoria y conseguir los resultados esperados.

- Tarea 8: mejora del diseño gráfico de la interfaz de usuario para hacer que la interacción con la herramienta sea más efectiva.
- Tarea 9: redacción de la memoria.

3.2. Distribución temporal de las actividades

El inicio de este proyecto se remonta al comienzo del primer curso del máster (T_0), momento en el cual se empezó a buscar toda la documentación pertinente para empezar con el desarrollo de la interfaz que se pretendía crear. Durante los primeros meses, se trabajó en una primera versión de la interfaz que se pondría a prueba en el primer ejercicio de validación de vuelo llevado a cabo en el cuarto mes (T_3).

Tras la realización de las pruebas y el parón navideño, se continuó con el desarrollo y mejoras de la interfaz, proceso que se extendió aproximadamente otros cuatro meses. En el mes T_7 se realizaron las segundas pruebas de validación, donde de nuevo se puso a prueba las capacidades de la interfaz ya mejorada. Posteriormente, se siguió con el desarrollo hasta el mes que empezaron las vacaciones de verano.

El proyecto se reanudó al inicio del segundo curso del máster. En este caso, se mejoraron los gráficos de la interfaz y se le dotó de un nuevo diseño, proceso que se alargó varios meses. Por último, una vez terminada interfaz, la redacción de la memoria se retrasó hasta el mes T_{17}, debido a la incorporación del alumno a prácticas en otra empresa.

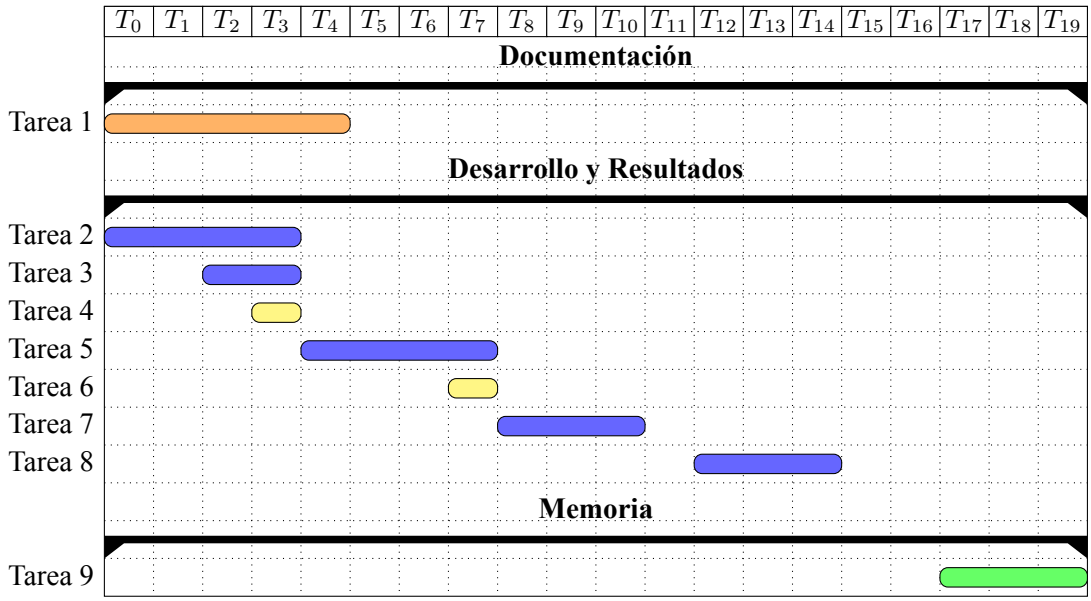


Figura 3.1: Diagrama de *Gantt* sobre la planificación temporal del TFM.

3.3. Herramientas de diseño

En todo proyecto de *software* se requiere de un entorno de trabajo completo que permita al desarrollador crear, diseñar y validar su sistema de forma efectiva. Por ello, fue indispensable para iniciar el proyecto realizar un estudio de las herramientas disponibles para el trabajo requerido. A continuación, se especifica la elección de cada una de ellas.

3.3.1. Kit de desarrollo de *software*

Como se ha comentado con anterioridad la empresa china *DJI* es líder en el mercado mundial de consumo de drones. Este éxito no sólo se debe a la calidad de su extenso catálogo de productos, sino también a la creación de una comunidad a través de su propia plataforma, donde los usuarios pueden crear soluciones mediante la tecnología que se les ofrece.

En este sentido, la elección de desarrollar el proyecto en torno al *Software Development Kit* (*SDK*) del fabricante *DJI* resulta obvia. La posibilidad de acceder al *software* que controla la mayoría de los dispositivos en el mercado es razón más que suficiente para elegir esta herramienta como punto de partida.

Sin embargo, también cabe resaltar el apoyo proporcionado por la empresa mediante una extensa documentación de las funciones provistas en el código, así como los foros facilitados por la comunidad que permiten la interacción con otros usuarios que se encuentran en la misma situación.

3.3.2. Sistema operativo

El sistema operativo escogido para emplear el anterior *SDK* fue *Android*. El *Mobile SDK V4* se provee tanto para el desarrollo en *Android* como en *iOS*, sin embargo, dado que la mayoría de dispositivos de *DJI* funcionan con el sistema *Android*, se ha decidido optar por este sistema operativo.

Igualmente que en el caso de *DJI*, el sistema operativo *Android* desarrollado por *Google* es líder de mercado con una cuota cercana al 70 % [11]. Debido a que se pretende que la aplicación esté disponible para el mayor número de usuarios posibles, permitiendo mejorar el sistema a través de la retroalimentación recibida, no cabe duda que esta es la decisión más plausible.

3.3.3. Entorno de desarrollo

Existen multitud de entornos de desarrollo para programar en *Android* debido a la escala de este sistema, tales como *Eclipse*, *NetBeans*, *IntelliJ*, *Aide* o *Android Studio*.

De entre todos ellos, se ha decidido escoger *Android Studio*, ya que se trata del entorno oficial para la plataforma escogida. Asimismo, esta elección se basa en el hecho que *Android Studio* proporciona un sistema de compilación sólido y flexible, un editor de código inteligente y optimización para todos los dispositivos *Android*, entre otras muchas características que hacen este *IDE* idóneo para el sistema planteado.

3.3.4. Lenguajes

Respecto a los lenguajes de programación, el *IDE Androiud Studio* ofrece la posibilidad de programar tanto en *Kotlin* como en *Java* y *C++*, siendo el primero de estos el preferido por los desarrolladores de *Google*.

Sin embargo, a lo largo del grado y el máster en la escuela, el lenguaje predominante para programar aplicaciones, tanto en *Android* como en otros sistemas operativos, ha sido *Java*, motivo por el cual se ha decidido optar por este lenguaje para la gestión de las acciones de la aplicación.

Por otra parte, *Android Studio* requiere del uso de *XML*, un lenguaje de marcado que proporciona reglas para definir cualquier dato e incluso para diseñar las distintas vistas que pueda requerir la aplicación. Principalmente, en este trabajo, se usa para el dimensionamiento y la posición de los distintos elementos que componen las vistas y la asignación de los comportamientos que deben producirse cuando el usuario interacciona con ellos.

Para la comunicación entre el servidor y la aplicación se estudiaron las posibilidades de uso de distintos lenguajes, pues existe una gran variedad que acometen esta función. Se optó por usar *JSON*, formato que se utiliza para almacenar y transmitir objetos de datos consistentes en pares atributo-valor y *arrays*. La justificación de esta decisión reside en la sencillez y rapidez del lenguaje, así como por el hecho de ser otro de los lenguajes estudiados durante la formación en la escuela.

Con respecto a la programación de vuelos automáticos en los que el dron realiza un a ruta predeterminada previamente por el piloto, es necesario hacer uso del lenguaje *KML*, también desarrollado por *Google*. *KML* usa una estructura basada en etiquetas con elementos y atributos anidados, y se basa en el estándar *XML* [12]. Mediante él, los pilotos se encargarán de emplear marcadores para situar los puntos geográficos por donde quieren que el dron realice su vuelo. En este caso no había posibilidad de escoger otro lenguaje, ya que el ordenador de a bordo de los drones *DJI* sólo emplea este formato para estructurar las rutas de vuelo automático.

3.3.5. Gestión de la telemetría

Una de las piezas fundamentales del sistema es la comunicación con otras entidades de la interfaz. Para la recepción y envío de telemetría entre los diferentes bloques se hizo uso del protocolo *MQTT*. La base de la arquitectura de este protocolo es de publicación/suscripción. Es extremadamente ligero, ideal para conectar dispositivos remotos con una huella de código pequeña y un ancho de banda de red mínimo, por lo cual se emplea en una amplia variedad de sectores, como automoción, fabricación, telecomunicaciones, petróleo y gas [13].

Capítulo 4

ARQUITECTURA DEL PROYECTO

Esta parte de la memoria se dedica a realizar una descripción a alto nivel del funcionamiento de la interfaz, empleando gráficos y diagramas de uso que ayuden a la comprensión del papel que desempeña dentro del funcionamiento básico del proyecto y explorando las distintas interacciones de la misma con el resto de elementos.

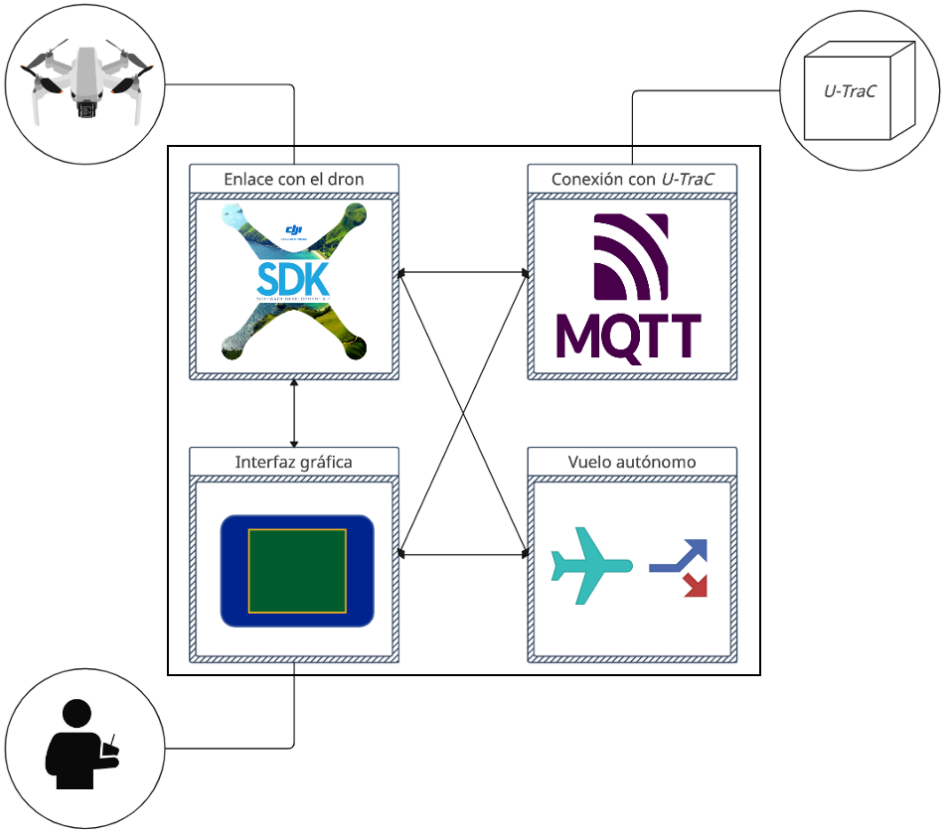


Figura 4.1: Descripción de la arquitectura de la interfaz a alto nivel.

Como se ha explicado en los objetivos, el principal propósito de la interfaz reside en proporcionar a los pilotos una herramienta capaz de presentar la información relevante del dron durante la ejecución del vuelo. Desde el punto de vista del operador, este envía a la interfaz los distintos comandos de vuelo que el dron requiera y obtiene una representación gráfica de la situación de vuelo, así como alertas por riesgos de colisión y sugerencias de maniobras de evitación.

Por su parte, la interfaz se encarga de procesar y transmitir las instrucciones de vuelo del piloto hacia al dron, permitiendo un desarrollo fluido y eficiente de su actividad. Al tiempo, el dron es el encargado de enviar la telemetría con detalles sobre su posición en coordenadas de latitud, longitud y altura, velocidad, dirección, etc. Toda esta información es recopilada por la interfaz para mostrar al piloto los detalles de las operaciones y para comunicarse con la plataforma *U-TraC*.

Por ende, la comunicación de la interfaz con la plataforma *U-TraC* cierra el círculo de la arquitectura, desde el punto de vista de una descripción a alto nivel. Esta última es la encargada de transmitir los detalles acerca de las alertas de conflicto para que la interfaz pueda comunicárselas al piloto de forma visual y sonora. Por su lado, la interfaz envía la telemetría recibida del dron para que la plataforma pueda realizar todos los cálculos pertinentes con respecto a la separación de aeronaves.

En la Figura 4.1 se describe de forma gráfica todo lo que se ha comentado anteriormente y se presenta la situación de la interfaz con respecto a los actores que interactúan con la misma. Como se desprende, la interfaz ocupa un puesto central en la arquitectura del proyecto, pues es la encargada de gestionar todas las interacciones entre la plataforma *U-TraC*, el operador y el *UAS*. A continuación se describe un breve resumen de cada una de las partes:

- Enlace con el dron: este bloque es el encargado de gestionar todas las operaciones de registro, acción y control que se ejecuten mediante el uso del *SDK*.
- Conexión con *U-TraC*: es la parte encargada de gestionar la telemetría que se utiliza para comunicar las entidades de la interfaz y que establece conexión con la plataforma *U-TraC*.
- Vuelo autónomo: es el entorno de la interfaz donde se realizan los procedimientos relacionados con el cometido de conseguir que el dron se maneje de forma automática sin intervención del piloto.
- Interfaz gráfica: se trata de la presentación de todos los bloques anteriormente descritos de forma visual para que el piloto pueda interaccionar con el dron.

4.1. Plataforma *U-TraC*

De acuerdo con lo expuesto durante la introducción de la plataforma *U-TraC*, esta se compone principalmente de tres bloques: *tracker*, detector de conflictos y servicio de información de tráfico. Además, hace uso de herramientas externas que permiten la detección de degradación en las comunicaciones, el cálculo de la separación mínima acorde a la situación del entorno de operaciones y el almacenamiento de la telemetría, *tracks* y conflictos para su posterior procesado y estudio.

El objeto principal de la plataforma es comunicar, mediante el protocolo de publicación y suscripción *MQTT*, los posibles conflictos que se produzcan al usuario que maneje el dron, por lo cual

el bloque más crítico del sistema es el detector de conflictos. Este nutre su información a partir del cómputo realizado mediante inteligencia artificial que se encarga de obtener las separaciones mínimas, a partir de los detalles provistos en las *tracks*.

El componente del *tracker* es el encargado de recibir y procesar los datos de la telemetría que recibe de la interfaz. A partir de estos, mediante técnicas que incluyen filtros de *Kalman*, este elemento se encarga de elaborar las *tracks* de cada uno de los drones. Una vez se han creado, son transmitidas a la herramienta externa para el cálculo de la separación mínima de acuerdo con las condiciones de vuelo.

Evaluar la situación para poder obtener cuál es la separación mínima que se debe respetar en cada caso requiere establecer una clasificación de las posibles operaciones que realicen los drones. De acuerdo con el Reglamento de Ejecución (UE) 2019/947 [14], se definen tres tipos de categorías de operaciones: abiertas, específicas y certificadas.

En el caso de las operaciones abiertas, los requisitos expuestos en el artículo 4 del RE 2019/947, que sientan la base de la normativa para realizar vuelos con los drones, son:

- El *UAS* debe pertenecer a una de las clases establecidas en el Reglamento Delegado (UE) 2019/945 o ser de construcción privada.
- El peso máximo al despegue (*MTOW*) ha de ser menor a 25 kg.
- Se tiene que mantener una distancia horizontal de seguridad con las personas y no superar las concentraciones de personas.
- Es obligatorio que el *UAS* permanezca dentro del alcance visual (*VLOS*) del piloto a distancia en todo momento, con excepción del vuelo en modo de seguimiento o cuando haya una aeronave no tripulada observada.
- El límite máximo de altura de vuelo desde el punto de despegue de 120 metros. Durante el vuelo, el *UAS* no dejará caer objetos ni transportará mercancías peligrosas.

A su vez, la categoría abierta se divide en 3 subcategorías: A1, A2 y A3. Las operaciones de la subcategoría A1 se realizan evitando el sobrevuelo de personas no participantes y las concentraciones de personas. Por su parte, las de subcategoría A2 se realizan con una distancia horizontal de seguridad de al menos 30 metros respecto a las personas que no participan en la operación. Finalmente, las operaciones de la subcategoría A3 son las más restrictivas, pues se llevan a cabo en zonas en las que no se pone en peligro a ningún no participante y a una distancia horizontal de seguridad mínima de 150 metros de zonas residenciales, comerciales o industriales [15].

En otras instancia, la categoría específica incluye operaciones *UAS* con un riesgo medio que no pueden realizarse en una categoría abierta. En esta categoría, los criterios de clasificación se basan en el valor *SAIL* de la operación *SORA*, que a su vez depende de los niveles de riesgo terrestre y aéreo. Utilizando este criterio, pueden definirse tres clases: riesgo menor (*SAIL* I o II), riesgo medio (*SAIL* III o IV) y riesgo mayor (*SAIL* V o VI) [14].

La tercera y última categoría es la certificada. En RE 2019/947 [14], se establece que las operaciones de *UAS* realizadas en esta categoría requerirán la correspondiente certificación del dron

por la Agencia Europea de Seguridad Aérea (*EASA*), la certificación del operador y, en su caso, la obtención de una licencia por el piloto remoto. Asimismo, se estipula que las operaciones se clasificarán en esta categoría únicamente cuando [15]:

- El *UAS* tenga una dimensión característica igual o superior a 3 metros, su diseño esté certificado por la *EASA* para ser utilizado sobre concentraciones de personas y la operación implique sobrevolar concentraciones de personas.
- El diseño del dron está certificado por la *EASA* para el transporte de personas y la operación implica el transporte de personas.
- El diseño de la aeronave está certificado por la *EASA* para el transporte de mercancías peligrosas exigiendo una gran robustez para mitigar los riesgos para terceros, en caso de accidente, y la operación implica el transporte de mercancías peligrosas que pueden suponer un alto riesgo para terceros, en caso de accidente.

En vista de lo anterior, *BUBBLES* propone la clasificación del tráfico aéreo en el *U-space* en función del riesgo de acuerdo con la Tabla 4.1.

Categoría	Pasajeros	Nomenclatura	Clase
Abierta no tripulada	No	A1	I
		A2	II
		A3	III
Específica no tripulada		SAIL I-II	IV
		SAIL III-IV	V
		SAIL V-VI	VI
Certificada no tripulada		Sin pasajeros	VII
	Sí	Con pasajeros	VIII
Certificada tripulada		VFR	IX
		IFR	X

Tabla 4.1: Clasificación del tráfico en BUBBLES de acuerdo con el riesgo de operación [9].

Considerando dicha taxonomía, se realizan los cálculos de las distancias mínimas. Una vez obtenidos los resultados, el detector de conflictos se encarga de clasificar la severidad de las posibles conflictos, de acuerdo con las reglas establecidas en el concepto de operaciones de *BUBBLES* [9]. Se definen así cuatro volúmenes de separación en el modelo de colisión propuesto que se corresponden con las severidades que se mostrarán en la interfaz gráfica.

- *Near mid-air collision* (*NMAC*): se define de forma cuantitativa como un incidente asociado a la operación de una aeronave en el que se produce una posibilidad de colisión como resultado de la proximidad a menos de 500 pies de otra aeronave [16]. La aparición de un *NMAC* se desencadena entonces por la superación del umbral de volumen de referencia *NMAC*, denominado volumen de colisión. Asimismo, Weinert et al. [16] proponen una expresión cuantitativa alternativa para el volumen *NMAC*, aplicable a los encuentros cercanos entre

UAS más pequeños (es decir, que miden menos de 12 pies de altura efectiva y 25 pies de anchura efectiva), que se denominó *sNMAC* (donde s significa más pequeño) y se definió mediante un umbral horizontal de 50 pies y un umbral vertical de 15 pies.

- Colisión inminente (*IC*): la colisión inminente se activa cuando se supera el umbral de evitación de colisión. La ocurrencia de este evento de separación significa que la barrera de provisión de separación ha fallado completamente, y la barrera anticolisión actuará entonces para evitar que las aeronaves entren en el volumen de colisión, es decir, activando un *NMAC*.

- Pérdida de separación (*SL*): se produce una pérdida de separación siempre que una aeronave se encuentra más cerca de un peligro que los mínimos de separación aplicables. Se pueden definir varios eventos de pérdida de separación, dependiendo del peligro específico, pero *BUBBLES* se centra en el peligro que supone una colisión en pleno vuelo entre un *UAS* y otro *UAS* o una aeronave tripulada. Una pérdida de separación significa que la barrera de provisión de separación ha fallado a la hora de evitar que se incumplan los mínimos de separación, pero no implica necesariamente su fallo total. En este estado, existe un margen para que la barrera de provisión de separación actúe y recupere los mínimos de separación.

- Conflicto táctico (TC): en *U-space* puede definirse cualitativamente como la convergencia prevista de aeronaves en el espacio y en el tiempo que constituye una violación de un conjunto dado de mínimos de separación mientras ambas aeronaves están en vuelo. Según la definición de los conflictos tácticos como predicciones, el proceso seguido para detectar conflictos, que es una búsqueda de conflictos, se basa en el cálculo y la comparación de las trayectorias de vuelo previstas de dos o más aeronaves con el fin de determinar los conflictos. La detección de un conflicto táctico marca el inicio de la aplicación de la barrera de provisión de separación, que actuará para evitar una pérdida de separación. En el proceso de detección de conflictos, la distancia de pérdida esperada en el punto de aproximación más cercano (*CPA*) y el tiempo restante hasta dicho punto son los criterios de referencia clave. Para que se desencadene un conflicto táctico, la distancia en el *CPA* debe ser inferior a los mínimos de separación correspondientes, mientras que el tiempo debe ser inferior a un umbral preestablecido.

En la Figura 4.2 se realiza una representación gráfica de los cuatro volúmenes de separación que se han descrito con anterioridad.

Tras haber aplicado todas las reglas de separación, el detector de conflictos clasificará las posibles colisiones en cuatro niveles de severidad, que se corresponden con los volúmenes anteriormente descritos. Esta información será trasladada al servicio de información de tráfico, que se encarga de almacenar las severidades de todos los conflictos conflictos actuales y transmitir dicha información a la interfaz.

La última de las funcionalidades de la plataforma es almacenar toda la información relativa a las *tracks* que han seguido los drones, la detección los posibles conflictos, las maniobras de solución aplicadas, la posible degradación de las comunicaciones, etc. En definitiva, se guardan todas las entradas, operaciones y salidas de *U-TraC* con objeto de analizar y obtener conclusiones acerca del funcionamiento de las interacciones entre las operaciones de vuelo y la interfaz. Así se ofrece la posibilidad de añadir, quitar o ampliar características del sistema creado que ayuden a mejorar la experiencia de usuario y a construir una herramienta potente, fluida y ágil.

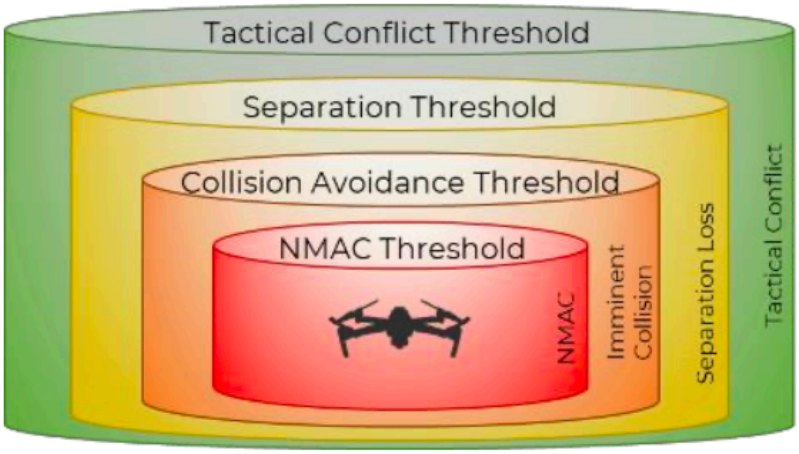


Figura 4.2: Volúmenes de separación para el modelo de colisión de *BUBBLES* [9].

4.2. Drones

Este proyecto se ha desarrollado en torno al planteamiento emitido por la Unión Europea de compatibilizar el espacio aéreo que ocupan los drones con el resto de las aeronaves. Consecuentemente, la razón de ser de este TFM parte de cómo se gestionen los *UAS*, por lo que resulta indispensable analizar sus propiedades. Debido a la decisión de realizar la interfaz en base a los equipos de *DJI*, a continuación se mostrarán sus principales características.

Esta empresa fabrica pequeñas aeronaves teledirigidas multirrotor de gran capacidad, así como cámaras portátiles y estabilizadores perfectos para aplicaciones comerciales y de consumo. Los productos son muy accesibles y fáciles de usar, existen gran cantidad de drones para elegir que ofrecen una gama de características, rendimiento, tamaño y precio determinados. Sin embargo, no entra dentro de los objetivos de este TFM analizar en detalle todas estas prestaciones, sino más bien presentar aquellas que han sido utilizadas como parte de la arquitectura del proyecto.

Figura 4.3: Vistas del modelo *DJI Phantom 4 Advanced* [17].

En términos generales, *DJI* dispone de una amplia gama que incluye, entre otros: *Mavic 2 Enterprise Advanced*, *Mavic Air 2*, *Mavic Mini*, *Mavic 2 Series*, *Mavic 2 Enterprise*, *Mavic Air*, *Mavic Pro*, *Phantom series*, *Inspire series*, *Matrice 200 V2 series*, *Matrice 100* y *Matrice 600* [17].

El propósito de presentar aquí esta información es realizar una descripción de los componentes básicos de los equipos con los que se ha trabajado a lo largo del proyecto, con el fin de aportar una mayor comprensión a la arquitectura que se deriva de la interfaz realizada. En la Figura 4.3 se pueden observar las vistas frontal, lateral y nadir del modelo *Phantom 4 Advanced*, que se usará como referencia para describir dichas especificaciones:

- Propulsores: montados en los motores proporcionan el empuje vertical, el cual puede ajustarse para permitir que la aeronave flote, gire, ascienda, descienda o vuele horizontalmente. Este sistema es clave para dirigir la aeronave y evitar las posibles colisiones entre *UAS*.
- Sensores: fundamentales para el funcionamiento del dron, *DJI* provee a sus equipos de múltiples sensores, tales como acelerómetros, giroscopios, brújulas, barómetros, sensores ultrasónicos, cámaras y sistemas de posicionamiento por satélite. Se utilizan para determinar el estado actual y predecir el estado futuro de la aeronave y el entorno que la rodea. Son los encargados de capturar la telemetría que se transmitirá a la plataforma *U-TraC*, a través de la interfaz. Los valores que contienen estos datos son la clave para determinar si la separación es adecuada o se debe ampliar.
- Controlador de vuelo: se trata del ordenador de a bordo que combina la información de control del piloto con la datos de los sensores para ajustar el empuje de cada hélice y volar la aeronave como se desee. Este elemento es fundamental para el caso en que el piloto desee volar el dron de forma automática, ya que será el elemento que procese y ejecute las acciones contenidas en el plan de vuelo programado.
- Cámara: puede grabar datos de imagen y vídeo localmente o transferirlos de forma inalámbrica a un dispositivo móvil. Además de la transferencia inalámbrica, incluye la opción de transmisión de vídeo en directo, lo cual ofrece al piloto un alto control sobre la situación de vuelo que esté ejecutando.
- *Gimbal*: es el encargado de sujetar la cámara y puede girarla alrededor de tres ejes. La rotación puede utilizarse tanto para controlar la dirección a la que apunta la cámara como para proporcionar estabilización rotacional cuando la aeronave no está horizontal. El *gimbal* está montado sobre una placa amortiguada, lo que significa que la cámara está estabilizada tanto contra las vibraciones laterales como contra el movimiento de rotación.
- Sensores de visión: ayudan a la aeronave a percibir el mundo que la rodea. Se utilizan cámaras estereoscópicas para detectar obstáculos cerca del producto. Las cámaras orientadas hacia abajo y los sensores ultrasónicos se utilizan para determinar la posición relativa en el suelo, mientras que las situadas en la parte frontal se encargan de evitar la colisión con obstáculos próximos al dron.
- Batería: proporcionan la energía necesaria para el funcionamiento del sistema. Junto con el controlador de vuelo, la batería permite estimar el tiempo de vuelo restante y proporcionar avisos cuando se cruzan los umbrales de batería baja.

Además de todos estos componentes de la aeronave, también es importante recalcar la función de los mandos de control a distancia del *UAS*. Su propósito reside en permitir el acceso al control de la interfaz a través del sistema operativo *Android*, por lo que se trata del nexo de unión entre dron e interfaz. El mando a distancia proporciona palancas de control, botones y ruedas que permiten controlar el vuelo del dron, la cámara y el *gimbal*. Esto es posible gracias al establecimiento de un enlace inalámbrico que, bajo condiciones de entorno específicos, puede alcanzar los 5 km de distancia.

4.3. Operadores

Diseñar la interfaz desde el punto de vista de Factores Humanos implica proveer a los operadores de vuelos de una serie de servicios y funcionalidades que les permitan desarrollar una experiencia de vuelo eficiente, cómoda y amigable. Con ello, desde el inicio del proyecto se detallaron una serie de características mínimas que la interfaz debería cumplir:

- Permitir al piloto elegir entre un abanico de drones *DJI*, independientemente de las diferencias entre los modelos y características que puedan existir.
- Autorizar al operador a activar o desactivar las conexiones con el servidor de comunicaciones de la plataforma *U-TraC*, así como la recepción de alertas.
- Facilitar una gestión de vuelo segura, por lo que se necesitará adoptar medidas que protejan la integridad de las comunicaciones, mediante la monitorización de los sistemas.
- Alertar sobre posibles conflictos con otros *UAS* próximos, así como sugerir una posible maniobra que ayude a evitar la colisión.
- Proveer un control total sobre el dron, informando sobre la posición mediante un mapa, mostrando la visión de la cámara del dron y ofreciendo la posibilidad de aterrizar o devolver al inicio la aeronave.

Una vez cubiertos estos requisitos mínimos, mediante pruebas de validación, los operadores realizaron una serie de evaluaciones y sugerencias en torno a la interfaz. Esto resulto ser muy útil ya que permitió, a partir de su experiencia real, mejorar la calidad de los servicios de la aplicación e implementar las mejoras que indicaron durante el desarrollo de dichas pruebas.

En base a esto, se desarrollaron utilidades de forma adicional. Por ejemplo, la interfaz también provee a los usuarios la posibilidad de cargar un plan de vuelo previamente diseñado en formato *KML*. Esto dota al sistema de una versatilidad inmensa, por ejemplo podría ser muy útil para explorar entornos muy complicados donde la maniobrabilidad del piloto pueda ser determinante y requiera de una previa investigación y comprensión del lugar. En consecuencia, este modo de operación permite al usuario pausar, reanudar o detener la ejecución del plan de vuelo en el momento que considere oportuno.

En la Figura 4.4 se muestra el diagrama de caso de uso donde se expone la funcionalidad que la interfaz ofrece al piloto de dron y que ha sido detallada con anterioridad.

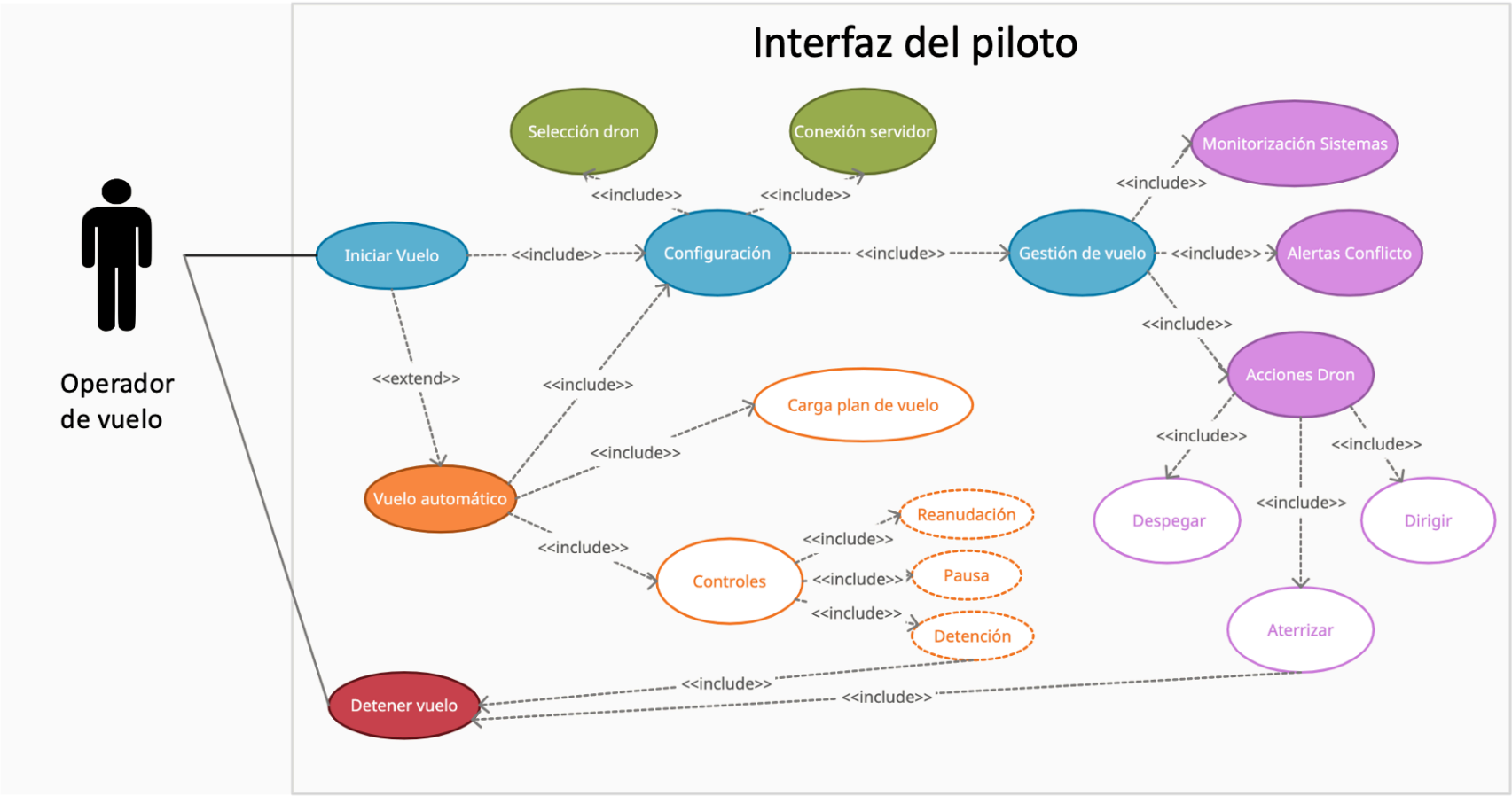


Figura 4.4: Diagrama de casos de uso de la interfaz desde el punto de vista del piloto.

4.4. Interfaz

La última parte de la arquitectura se centrará en la propia interfaz. Con el fin de realizar una explicación lo más detallada posible del funcionamiento a alto nivel de la misma se ha decidido segmentarla en los cuatro bloques que interaccionan entre sí: la conexión con el dron, la cual se realiza mediante el *SDK* de *DJI*; la conexión con la plataforma *U-TraC*, haciendo uso del protocolo *MQTT*; el plan de vuelo automático y la interfaz gráfica que se comunica con el operador.

4.4.1. Conexión con el dron

La parte fundamental sobre la que se basa la funcionalidad de la interfaz reside en el *DJI SDK Mobile V4*, el *software* encargado de ofrecer acceso a las capacidades de las aeronaves y cámaras portátiles de *DJI*. Con este kit se simplifica el proceso de desarrollo de aplicaciones al encargarse de funcionalidades de bajo nivel como la estabilización del vuelo, la gestión de la batería, la transmisión de señales y la comunicación.

A continuación, se describen algunas de las principales funcionalidades que han sido empleadas a lo largo del proyecto:

- Control de vuelo: se permiten tres formas de controlar el vuelo de la aeronaves, las cuales se han usado en el proyecto para satisfacer distintas finalidades.
    1. Manual: el usuario pilota la aeronave con el mando a distancia, mientras el *SDK* permite monitorizar vídeo en directo y datos de los sensores.
    2. Comandos de *joystick* virtual: posibilita generar movimientos con los *joysticks* del controlador remoto de forma virtual, simulando la ejecución de un piloto. Este modo se usó para realizar las validaciones de la interfaz en el laboratorio, previo a la segunda tanda de pruebas de vuelo.
    3. Misiones: control de alto nivel de la aeronave, cómodo y fácil de implementar. Por ejemplo, como ya se ha comentado a lo largo del proyecto, en este TFM se ha hecho uso de las misiones tipo *Waypoint* para conseguir programar rutas de vuelo de dron que se ejecutan de forma autónoma.
- Cámara: la funcionalidad conjunta del *gimbal* y la cámara dotan a las aeronaves de muchas prestaciones, las cuales pueden ser ajustadas por los operadores. Entre las más destacadas existen: el modo de cámara, con la captura de vídeo o imágenes fijas; la exposición, a través de parámetros como el obturador, la apertura y la compensación de exposición; los parámetros de imagen (relación de aspecto, contraste, tono, nitidez, saturación y filtros) y vídeo (resolución y frecuencia de imagen) o la dirección (*gimbal*).
- Vídeo en directo: es una de las funcionalidades más importantes de los drones de esta empresa. Esta característica está disponible incluso cuando la cámara está capturando imágenes o vídeo en su medio de almacenamiento, permitiendo al piloto observar en todo momento lo que sucede alrededor de su aeronave.

- Datos del sensor: estos son utilizados durante el proyecto para formar los mensajes de telemetría que se enviarán a la plataforma *U-TraC*. Incluyen la posición *GPS*, detalles sobre la brújula, lecturas del barómetro y velocidad de vuelo, entre otros.

En la posterior sección de la memoria se entrará más al detalle de cómo se ha hecho uso de los elementos que componen el *SDK* para programar ciertas funcionalidades del proyecto. En esta última parte de la descripción del *software* empleado para la programación de la interfaz, se mostrará a grandes rasgos la arquitectura del mismo.

La arquitectura está diseñada para ser altamente extensible. Se utilizan clases abstractas de *Product* y *Component* para que las aplicaciones puedan controlar distintos productos con el mismo código. La gran modularidad y flexibilidad de crear una arquitectura como esta permite que no sólo la interfaz sea usada en los actuales múltiples productos de la marca, sino que además no requiera de grandes modificaciones cuando se lancen nuevos drones al mercado.

Debido a la gran demanda que se prevé para este mercado, la facilidad de contar con un *software* tan potente para la interfaz es un valor añadido enorme, ya que cualquier nueva característica del nuevo producto tendrá que ser incorporada, mientras que las existentes no necesitarán apenas ninguna modificación.

En la Figura 4.5 se muestran la jerarquía a alto nivel de la arquitectura en la que se basa el *software* de desarrollo. Seguidamente, se realiza una breve descripción de cada uno de los principales bloques que la componen, así como de las especificaciones que ofrecen:

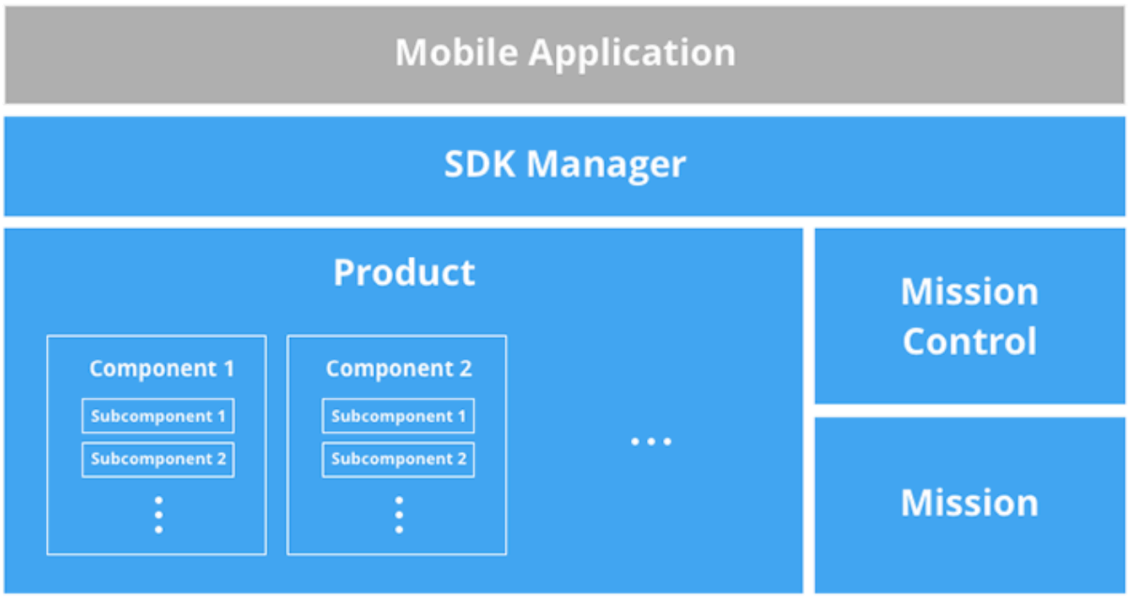


Figura 4.5: Arquitectura a alto nivel del *DJI SDK Mobile V4* [18].

- *SDK Manager*: gestiona el registro del *SDK*, la conexión del producto y proporciona acceso a las funcionalidades del mismo.

- *Product*: esta clase mantiene las propiedades básicas del producto y contiene sus componentes principales.
- *Component*: describen el *gimbal*, la cámara, el controlador de vuelo, el controlador remoto y el enlace inalámbrico. Proporcionan información de control de componentes, información de estado y pueden contener *Subcomponents*.
- *Mission*: clases que describen diferentes misiones como las de tipo *Waypoint* y *ActiveTrack*. Además, controlan las características de su estructura y configuraciones de múltiples estados.
- *Mission Control*: gestiona la ejecución de las misiones. Pueden ejecutarse misiones individuales a través de operadores de misión dedicados, o una serie de misiones, las cuales pueden ejecutarse en serie utilizando la función de *Timeline*.

4.4.2. Conexión con la plataforma *U-TraC*

La comunicación que realiza la interfaz con la plataforma *U-TraC* requiere ciertas especificaciones en cuanto a tamaño de los mensajes, rapidez y escalabilidad, pues a medida que la solución desarrollada amplíe sus horizontes en el mercado de los drones, se realizarán vuelos con más y más equipos, por lo que la comunicación debe ser lo suficiente fiable. Todas estas características las cumple el protocolo seleccionado, el cual se explicará a continuación.

MQTT es un protocolo de transporte de mensajería de publicación/suscripción cliente-servidor. Es ligero, abierto, sencillo y está diseñado para ser fácil de implementar. Estas características lo hacen ideal para su uso en muchas situaciones, incluidos los entornos limitados, como la comunicación en contextos Máquina a Máquina (*M2M*) e Internet de las Cosas (*IoT*), donde se requiere una pequeña huella de código y/o el ancho de banda de la red es un bien escaso [19].

El modelo en que se basa este protocolo es del tipo *publish/subscribe*, también conocido como *pub/sub*. A diferencia del modelo tradicional cliente/servidor, este desacopla el cliente que envía un mensaje (*pub*) del cliente o clientes que reciben los mensajes (*sub*). Estos dos agentes nunca se comunican directamente, ni siquiera saben de la existencia del otro, ya que la conexión entre ellos la gestiona un tercer componente (*broker*).

En la Figura 4.1, la conexión con la plataforma *U-TraC* hace referencia a la figura del *broker MQTT*, de forma que los suscriptores serían tanto los drones, cuya comunicación la realizan a través del *SDK*; los usuarios, que participan mediante la interfaz gráfica y el propio sistema *U-TraC*. El *broker* es el encargado de recibir todos los mensajes, filtrarlos, determinar quién está suscrito a cada mensaje y enviarlo a sus pertenecientes clientes.

Por lo tanto, y como ya se ha comentado con anterioridad, es fundamental que el *broker* sea altamente escalable, fácil de monitorizar y resistente a fallos, proporcionando un sistema interfaz-dron-piloto eficiente y seguro de comunicación. Existen infinidad de *brokeres MQTT*, pero el escogido para el desarrollo del proyecto ha sido *HiveMQ*, pues bajo la percepción del estudiante su diseño cumplía con todos los requisitos expuestos. Se trata de un sistema de fácil implementación que se apoya en un procesamiento de red basado en eventos de última generación, proveedores de monitorización estándar y la seguridad como principales baluartes.

El correcto funcionamiento de la telemetría aboga por la monitorización de la calidad de servicio (*QoS*) de la transmisión de los mensajes, siendo esta una característica clave del protocolo *MQTT*. Se trata de un acuerdo entre el emisor de un mensaje y el receptor del mismo que define la garantía de entrega de un mensaje específico. Existen tres niveles de calidad de servicio, los cuales se distinguen mediante los valores 0, 1 y 2:

- Como máximo una vez (0): es el nivel mínimo de *QoS*. Este nivel de servicio proporciona una entrega bajo la práctica *best effort*, por lo que no hay garantía de entrega.
- Al menos una vez (1): garantiza que un mensaje se entregue al menos una vez al receptor. El emisor almacena el mensaje hasta que recibe un paquete del receptor que sirve de acuse de recibo del envío.
- Exactamente una vez (2): es el nivel más alto de servicio. Su principal valor reside en asegurar que cada mensaje es recibido una sola vez por los destinatarios previstos, siendo el servicio más seguro y lento.

4.4.3. Vuelo automático

Una de las prestaciones más importantes de la interfaz desarrollada es la posibilidad de realizar la carga de un archivo que contenga la ruta de vuelo para el dron y que esta se ejecute de forma automática cuando el piloto así lo desee. Esta característica se gestiona a través el uso de los bloques *Mission* y *Mission Control* del *SDK*, en concreto usando las funcionalidades de *Waypoint Mission*.

Para poder emplear esta funcionalidad en la interfaz, primero será necesario generar un plan de vuelo mediante alguna herramienta externa que permita la posibilidad de trabajar con archivos de formato tipo *KML*, por ejemplo *Google Earth*. En el proceso de la generación de este archivo será suficiente con indicar las coordenadas 3D por los que se requiera que circule la aeronave. Posteriormente, se guardará el archivo en el dispositivo donde se ejecute la aplicación, con el fin que el usuario pueda seleccionar y realizar la carga del fichero al dron.

Mediante el uso de la clase *Waypoint Mission*, la aeronave viajará entre *waypoints*, ejecutará las acciones que se hayan podido solicitar y ajustará el rumbo y la altitud en cada uno de estos. Los *waypoints* son localizaciones físicas, cuya referencia básica son coordenadas en 3D de latitud, longitud y altura a las que la aeronave volará. Adicionalmente, se pueden añadir acciones a los *waypoints*, que se llevarán a cabo cuando la aeronave alcance el destino. Por defecto, el dron viaja entre *waypoints* automáticamente a una velocidad base, si bien es cierto que el piloto puede cambiar la velocidad utilizando el *joystick* de cabeceo.

Asimismo, mediante la clase *Mission Control* se le ofrece al piloto el control total de la situación del vuelo automático. Entre las funciones más destacadas, este puede decidir pausar e incluso detener la ruta de vuelo con tan sólo pulsar un botón.

Una de las desventajas de esta parte del *SDK* son sus limitaciones, pues no es compatible con los *Mavic Pro* cuando se utiliza la conexión *WiFi*. Tampoco es posible su ejecución en los modelos: *Spark*, *Mavic Mini*, *DJI Mini 2*, *DJI Mini SE*, *Mavic Air 2*, *DJI Air 2S* y *Matrice 300 RTK* [20].

4.4.4. Interfaz gráfica

La parte visual de la interfaz se ha desarrollado en base a los Factores Humanos comentados en los objetivos del proyecto. Tanto el diseño de la interfaz de usuario como la experiencia se ha basado en esta técnica para poder ofrecer un producto sencillo y útil para el piloto.

Con este propósito en mente y tras una serie de versiones detalladas en la parte de desarrollo, el resultado final de la aplicación se compone de un total de seis pantallas que permiten la interacción del usuario con el sistema creado. En la Tabla 4.2, se muestran su relación con el código creado.

Función	Clase Java	Diseño
Pantalla de carga	*SplashActivity*	*activity_splash.xml*
Configuración del dron	*ConfigDrone*	*activity_config_drone.xml*
Configuración de telemetría	*MqttConnection*	*activity_mqtt_connection.xml*
Selección de modo	*ChooseFlightMode*	*activity_choose_mode.xml*
Gestión de vuelo automático	*AFPSelection*	*activity_afp_selection.xml*
Interfaz del piloto	*CompleteWidgetActivity*	*activity_default_widgets.xml*

Tabla 4.2: Pantallas de la versión final de la aplicación.

Estas seis son las clases sobre las que puede interaccionar el operador, las cuales se cimientan sobre el código de otras clases con el fin de realizar las operaciones necesarias para llevar a cabo el funcionamiento correcto de la aplicación. Todo esto se profundizará en detalle en la siguiente sección de la memoria.

Cabe destacar que, para la creación de cada una de estas pantallas, se realizaron varios borradores hasta dar con un diseño que satisficiese tanto al equipo como a los usuarios. Tras los primeros pasos y, con las sugerencias de los pilotos durante las pruebas de vuelo, se consiguió pulir la imagen de la interfaz hasta llegar al diseño final que se presenta en la parte de los resultados.

Capítulo 5

DESARROLLO Y RESULTADOS

En este capítulo de la memoria se pretende exponer el desarrollo del código creado para la interfaz, resaltando las partes más importantes y críticas de la misma, así como las dificultades que han ido surgiendo durante el proceso. Posteriormente, se abordarán las dos pruebas de vuelo que se realizaron durante el proyecto junto con la implementación de las sugerencias de mejora por parte de los probadores. Finalmente, se mostrará el producto final diseñado donde se podrá apreciar todo lo expuesto con anterioridad a través de las imágenes de las vistas.

5.1. Primera versión

Tras el estudio de las herramientas necesarias para llevar a cabo el proyecto, se empezó a diseñar la primera versión de la interfaz. En primer lugar, se crearon los dos archivos principales, el *AndroidManifest.xml* y *MApplication*.

En el manifiesto de la aplicación se describen el nombre del paquete de la aplicación (*MApplication*), los componentes de la aplicación (actividades) y los permisos que necesita la aplicación para acceder a las partes protegidas del sistema o a otras aplicaciones, siendo los más relevantes los relacionados con la conexión al *SDK* de *DJI*.

Por su parte, *MApplication* es la clase principal en la jerarquía del código y, por tanto, la raíz a partir de la cual se crean todas las demás clases.

A partir de estos dos documentos iniciales y con el fin de empezar a crear las clases que compondrían el código de la interfaz, se establecieron una serie de prioridades mínimas que debía cumplir:

1. Registro de los servicios de *DJI*.
2. Selección del plan de vuelo y dron escogido.
3. Conexión con la plataforma *U-TraC*.
4. Visión del vuelo del dron.

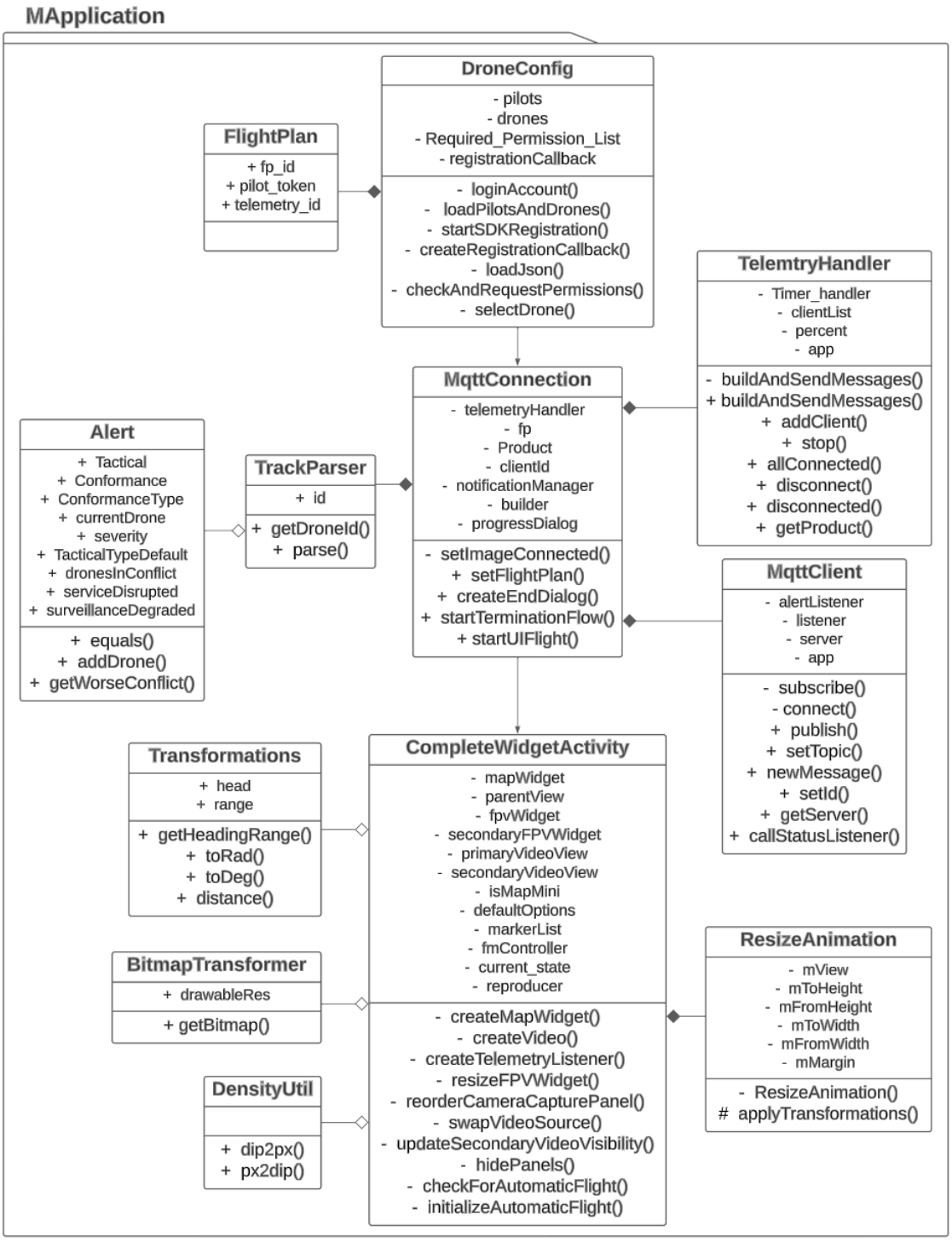


Figura 5.1: Diagrama de clases de la primera versión de la interfaz.

A partir de estas especificaciones, se diseñaron tres grandes bloques: el enlace con el dron, la conexión con la plataforma *U-TraC* y la interfaz gráfica de vuelo. Adicionalmente, también se desarrolló un bloque auxiliar que se encargó de las operaciones y transformaciones de imágenes e iconos. En la Figura 5.1 se puede apreciar la jerarquía de clases del código desarrollado, junto con todas las interacciones que las relacionan. A continuación, se describirán en profundidad cada uno de los bloques descritos.

5.1.1. Enlace con el dron

El primero de los cuatro grandes bloques de la versión inicial está relacionado con el arranque de la interfaz. Este bloque está compuesto por una clase Java principal *DroneConfig* y una clase Java auxiliar *FlightPlan*.

5.1.1.1. *FlightPlan*

La clase auxiliar es bastante simple ya que consta de un constructor que se encarga de configurar el plan de vuelo a partir de tres parámetros: *fp_id*, identificador del plan de vuelo; *pilot_token*, cadena alfanumérica que se obtiene al registrarse como usuario de *DJI* para identificar inequívocamente al piloto y *telemetry_id*, código para identificar y separar las comunicaciones dentro de la telemetría.

5.1.1.2. *DroneConfig*

En la clase principal *DroneConfig*, se realizan distintos procesos. El primero de ellos, y el más importante, es la solicitud y comprobación de aprobación de los permisos necesarios para el correcto funcionamiento de la aplicación. En concreto, se requiere acceso a internet, gestión de las conexiones *WiFi*, acceso a la localización del dispositivo, conexión mediante *Bluetooth*, lectura y escritura de archivos y grabación de sonidos.

El segundo procedimiento a destacar sería la conexión con el *SDK*. Para ello se ejecuta una función asíncrona que se encarga de registrar la aplicación en el *SDK*, permitiendo al usuario iniciar sesión y conectarse a los servicios de *DJI*. Sin este registro, la interfaz no podría conectarse con el dron, así que se trata de un proceso crítico. El requisito es una clave de acceso (*API Key*) única asociada al nombre del paquete de la aplicación, en este caso *com.dji.ux.sample*, que permite registrarla en la web de *DJI*, de donde se obtiene dicha clave. Si el resultado es exitoso, entonces se aparecerá en pantalla el mensaje de la Figura 5.2.

Posteriormente, mediante el uso de un *callback*, se comprueba que si la conexión ha sido exitosa y se ejecutan funciones como la consulta de si existe algún producto *DJI* conectado, el chequeo de un cambio de estado del producto o un cambio de producto.

Otra de las funcionalidades de esta clase es la gestión y presentación de los ficheros *drone_id.json* y *pilots.json*, los cuales contienen los parámetros para la configuración de la clase *FlightPlan* de los drones y los pilotos, respectivamente. Con esta información es con la que se realiza la configuración del plan de vuelo, que es el objetivo de este bloque de código. Como dicha

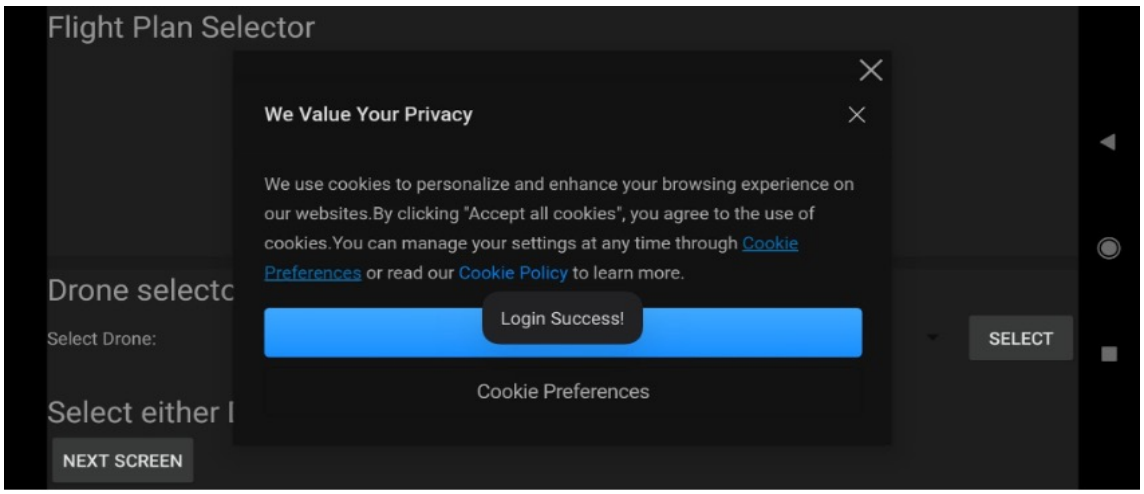


Figura 5.2: Conexión exitosa con los servicios del *SDK* de *DJI*.

información se encuentra en un formato *JSON*, se ha creado una función que se encargue de leer los archivos y almacenarlos en variables Java para su uso dentro de la aplicación.

Las últimas funcionalidades están asociadas con el funcionamiento de la pantalla correspondiente a este bloque: *activity_drone_config_id.xml*. Desde el punto de vista del usuario, la primera decisión que deberá tomar el piloto al acceder a la interfaz es la configuración del plan de vuelo. En concreto, la interfaz permite al piloto la selección del dron que se va a volar.

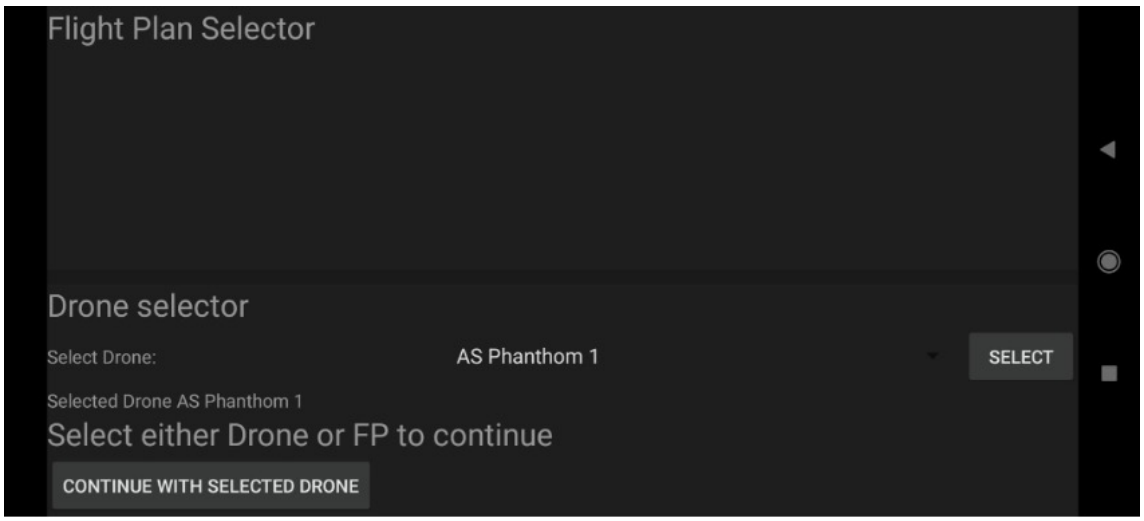


Figura 5.3: Vista de la configuración del plan de vuelo.

En la Figura 5.3, se puede observar la pantalla que se le presenta al piloto. En ella se puede apreciar el título de la pantalla en la parte superior izquierda y la posibilidad de seleccionar el dron que se va a volar en la zona *Drone Selector*, a través de un desplegable donde están almacenados los distintos equipos disponibles. A la derecha de este desplegable, aparece un botón que se emplea

para confirmar la selección realizada con en dicho elemento de la vista. En el margen inferior izquierdo de la aplicación aparece un botón con el texto *Next Screen* que sirve para pasar a la siguiente vista de la aplicación, donde se gestionan las conexiones con el servidor encargado de la telemetría.

5.1.2. Conexión con la plataforma *U-TraC*

La gestión de la telemetría en la aplicación se lleva a cabo mediante la clase Java *MQTT Connection* y las interacciones con las clases Java *MQTT Client*, *Telemetry Handler* y *Track Parser*. A continuación se detallarán las funcionalidades de sus operaciones.

5.1.2.1. *Track Parser*

En primer lugar, *Track Parser* se compone de una clase pública llamada *Alert* y dos funciones *getDroneId* y *parse*. La más sencilla de estas dos es la primera, cuyo única finalidad es diferenciar el tipo de identificador del dron en cuestión y devolver el resultado en forma de cadena. Se trata pues de una función auxiliar que sirve a *parse* para distinguir los drones según su *drone_id*.

La función *parse* es la función principal de la clase cuyo propósito reside en analizar los pares nombre-valor de un mensaje en formato *JSON*, para así gestionar la proximidad de los conflictos existentes o las posibles degradaciones en el ámbito de las comunicaciones. Para ello, crea un *array JSON* con el mensaje recibido y mediante la construcción de un objeto de tipo *JSON* lo recorre obteniendo la información deseada.

Así, construye y devuelve un objeto de la clase *Alert* que contiene como principales parámetros: el número de drones en conflicto, la severidad de los conflictos con otros drones, la calidad de las comunicaciones y la calidad de la supervisión. Este proceso se visualiza en el diagrama de flujo de la Figura 5.4

En definitiva, *Alert* es una clase auxiliar que nos permite empaquetar toda la información de una alerta, esto es la asignación de los parámetros y variables descritos en el párrafo anterior, y procesarla para mostrarla al piloto de la forma más indicativa posible. Para ello, hace uso de dos funciones: *addDrones* y *getWorseConflict*.

La primera de ellas añade a lista de conflictos aquellos drones cuya severidad sea inferior a 99 (valor por defecto), mientras que la segunda tiene el propósito de determinar entre todos los conflictos detectados cuál es el más severo. En total existen cinco niveles de severidad: 0, 1, 2, 3 y 4, donde 0 indica colisión y 4 conflicto táctico.

5.1.2.2. *TelemetryHandler*

Por su parte, *TelemetryHandler* consta de un constructor, ocho funciones y dos *listeners*. El constructor crea el cliente *MQTT* y lo añade a la lista de clientes. Además, ejecuta un *Handler* temporal en el que se comprueba si existe algún producto conectado al que se deba enviar un mensaje, tras esperar un período breve de tiempo. A su vez, los dos *listeners* sirven para gestionar cambios en las alertas y en el estado de los clientes suscritos a la lista.

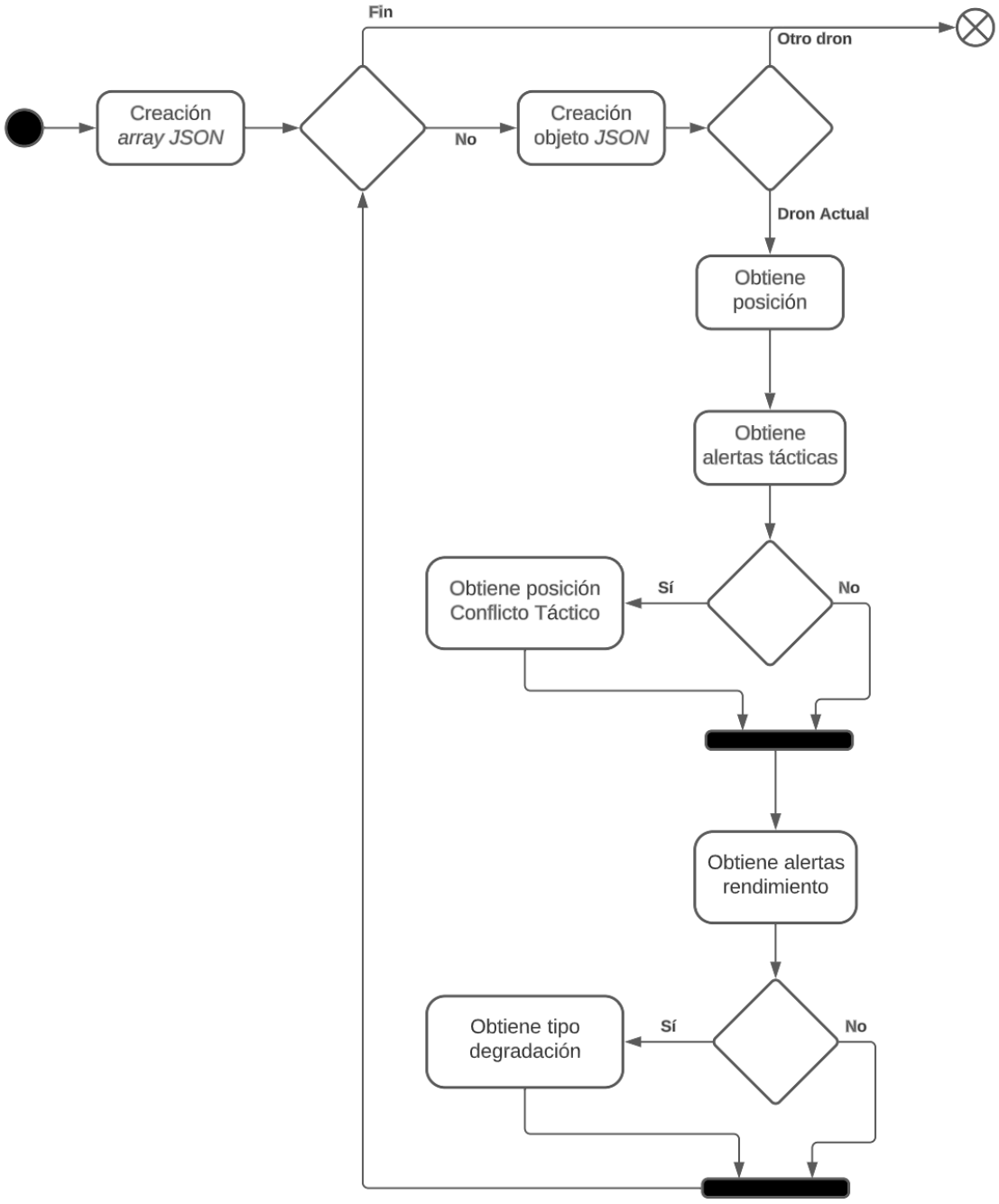


Figura 5.4: Diagrama de flujo de la función *parse*.

La función más relevante de esta clase es *buildAndSendMessages*. Como parámetro de entrada recibe un producto *DJI*, si existe alguno conectado. Las operaciones que realiza sobre el estado del dron se utilizan para calcular la velocidad y el ángulo de trazado. Las características restantes para componer el mensaje se obtienen mediante un objeto de tipo *FlightControllerState*, donde se almacena el estado actual del dron en cuestión.

Una vez obtenidos los parámetros se confecciona el mensaje y se envía a todos los clientes suscritos, es decir, aquellos que se encuentren en la lista de clientes. En concreto, el mensaje contiene los siguientes campos: balanceo, inclinación, viraje, batería, latitud, longitud, altitud, tiempo, velocidad, ángulo de trazado y velocidad de altitud.

De las siete funciones restantes, las más relevantes serían *getProduct* y *addClient*. Esta última es la más simple, ya que sólo se encarga de dado un cierto cliente, incluirlo dentro de la lista de clientes. La función *getProduct* tiene como misión comprobar la existencia de conexión de algún producto *DJI* y mediante un *callback*, recuperar la información del porcentaje restante de batería, el voltaje y la corriente del dron. El resto de funciones sirven para comprobar la conexión o desconexión de los clientes *MQTT*, por ejemplo, ante un evento como la finalización de los *Handlers* temporales.

5.1.2.3. *MqttClient*

La clase *MqttClient* consta de dos interfaces, un constructor y ocho funciones que tienen el propósito de proveer a la aplicación con las capacidades necesarias para crear un cliente de protocolo *MQTT*. Los interfaces creados se dedican a detectar los posibles cambios de estado de la situación del dron y los relacionadas con las alertas de conflictos, para lo cual hace uso de la clase *Alert* de *TrackParse*.

El constructor de esta clase es el encargado de generar nuevos clientes *MQTT* en la conexión. Para ello requiere de parámetros como el usuario y la contraseña contraseña del cliente, el servidor al que se debe conectar, el identificador del cliente, el contexto de la aplicación y el gestor de notificaciones de *Android*.

Al cliente creado se le asigna un *callback* que se encarga de mostrar un mensaje en pantalla sobre si la conexión ha sido exitosa o si por el contrario el establecimiento de la comunicación ha fracasado. También se emplea para calcular las distancias vertical y horizontal del peor conflicto con el dron actual y mostrarlo por pantalla en forma de mensaje de alerta.

Para implementar un cliente en el protocolo *MQTT* se necesitan ciertas funciones básicas, las cuales serían: *subscribe*, donde se suscribe al cliente al tema, en este caso las alertas de conflicto; *connect*, cuyo objetivo es establecer la conexión con el servidor *MQTT* e informar en caso que no haya sido posible dicho proceso; *publish*, publica mensajes de un cierto tema y se lo transmite a los clientes suscritos a este; *setId*, establece los distintos temas para el intercambio de mensajes y *newMessage*, que crea un nuevo mensaje a transmitir mediante la ayuda del formato *JSON*.

El resto de funciones están destinadas a la comprobación del funcionamiento y la supervisión de las comunicaciones mediante el protocolo *MQTT*. En este sentido, comprueban si el cliente está desconectado o conectado, el estado de las interfaces, la conexión con el servidor y los temas creados.

5.1.2.4. *MqttConnection*

La clase *MqttConnection* es la que engloba todos los procesos relacionados con la gestión de lea telemetría. Se compone de seis funciones cuyo funcionamiento está adscrito a la segunda vista que interacciona con el usuario: *activity_mqtt_connection.xml*. En la Figura 5.5, se presenta la pantalla en la que el piloto podrá gestionar todo lo referente a la telemetría previo al vuelo.

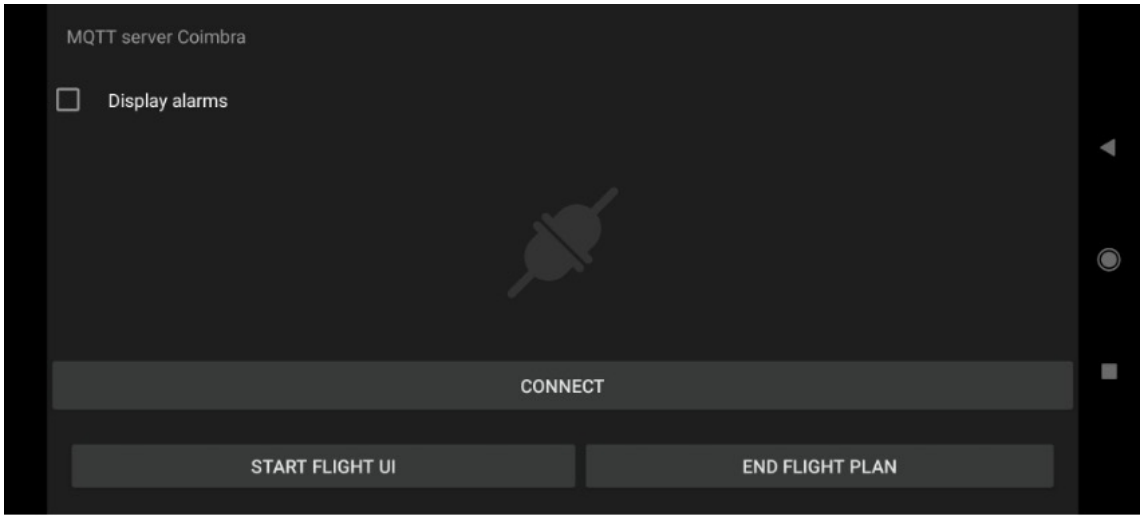


Figura 5.5: Vista de la gestión de la telemetría.

En esta pantalla el operador de vuelo dispone de diversas opciones. En primer lugar, debe decidir si activar o desactivar las alertas de vuelos, mediante la casilla seleccionable que se encuentra en la parte superior izquierda. Como se trata de un función muy útil en el vuelo, por defecto aparecerá siempre activa, aunque se deja a elección la capacidad de desactivarla.

La segunda decisión que deberá tomar es si desea realizar la conexión con el servidor *MQTT* (botón *Connect*). Si el usuario pulsa este botón, se activa la acción nativa *OnClick* y se intenta establecer la conexión con el servidor *MQTT* creando un nuevo cliente. Si la conexión es fructífera, entonces se crea un nuevo objeto de tipo *telemetryHandler* asignado a dicho cliente, se cambia el texto del botón a *"Disconnect"*y el logo de la conexión cambia de color a verde (Figura 5.6). Por el contrario, si la conexión fracasa, entonces se desconecta el *telemtryHandler*, se cambia el texto del botón de nuevo a *"Connect"* y el logo aparece ahora en color rojo.

Otra opción que tiene el usuario es la de empezar el plan de vuelo con o sin configurar nada de lo anterior. Para ello, debe pulsar el botón *"StartFlightUI"*, el cual desencadena la función de la clase que lleva ese mismo nombre e intenta lanzar la siguiente pantalla. Si no existe ningún dron conectado, entonces este paso no será posible y se deberá conectar un dron para avanzar.

También se ha añadido un botón con texto *"End Flight Plan"* que se utiliza para terminar un plan de vuelo ya existente. Cuando este se pulsa, se lanza la función *startTerminationFlow*, la cual muestra un diálogo en la pantalla par confirmar la finalización del plan de vuelo por el usuario y la posterior salida de la aplicación. Si por el contrario no hubiese ningún plan de vuelo previamente establecido, la aplicación lo notificaría con un mensaje.

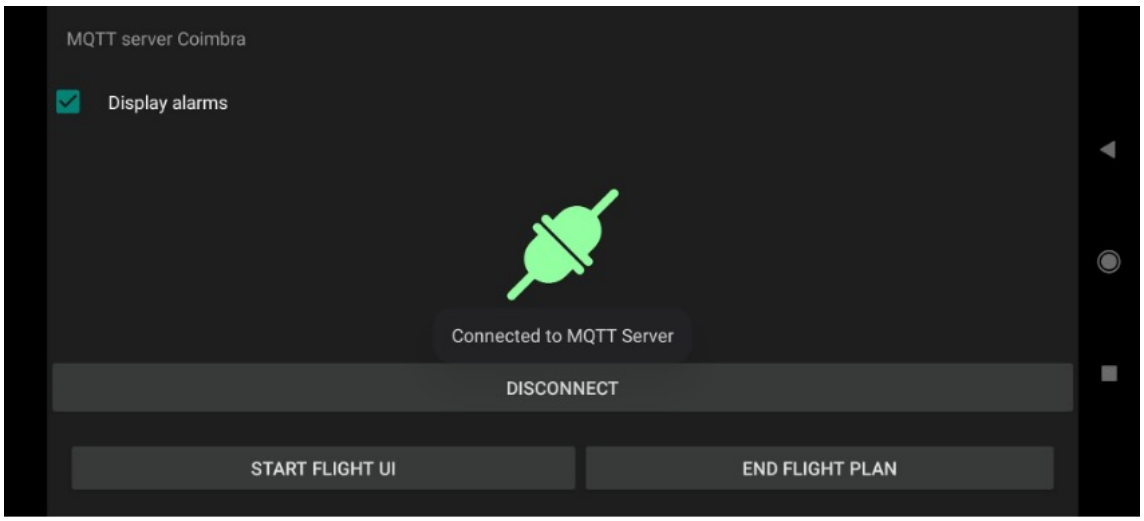


Figura 5.6: Conexión completada con el servidor *MQTT* de la telemetría.

5.1.3. Interfaz gráfica de vuelo del dron

Una vez realizada toda la configuración de vuelo y telemetría, se dirige al piloto hasta la interfaz gráfica de vuelo. En ella, el piloto dispone de toda la información que facilita *DJI* para llevar a cabo el control del vuelo de la forma más segura posible. Esta parte de la interfaz se basa en la clase Java *CompleteWidgetActivity*. Como se trata de la clase más importante del proyecto, a continuación se describirá en detalle cada una de sus funciones.

CompleteWidgetActivity es la clase donde se describen todos los comportamientos relacionados con la interfaz gráfica de vuelo de la que dispone el piloto mientras está realizando sus operaciones. En concreto, la clase cuenta con siete funciones y una clase privada llamada *ResizeAnimation*, extensión de la clase *Android Animation*, que cambia el tamaño de la vista, mediante la aplicación de operaciones de transformación.

El propósito principal de esta clase es realizar la inicialización del mapa, lo cual se consigue en la función *createMapWidget*. En esta función se crea el mapa que verá el piloto mientras vuele el dron, para lo cual se utiliza la clase *Android MapboxMap*. Una vez se ha creado con éxito el mapa, se inicializan los *listeners* para gestionar las interacciones del usuario con el mapa y se establecen los parámetros iniciales de configuración. Por defecto, el mapa aparecerá como tipo satélite y serán visibles los botones de retorno a inicio y despegue de la aeronave.

El resto de funciones sirven como apoyo a la anteriormente descrita y se encargan de gestionar bien aspectos visuales o bien características del dron. La función *createVideo* se encarga de realizar grabaciones al tiempo que permite al piloto seguir vigilando el recorrido del dron. Para ello, utiliza las propiedades del *VideoFeeder* de la cámara del *SDK*.

Existen otras dos funciones relacionadas con la captura de vídeo: *swapVideoSource* y *updateSecondaryVideoVisibility*, cuyos cometidos son distinguir entre modo de captura primario o secundario y activar o desactivar la visibilidad de estos, respectivamente.

Por su parte *resizeFPView* se utiliza cuando el usuario decide minimizar el mapa para centrarse en la imagen que provee la cámara del dron y se encarga de reconfigurar los tamaños de la vista. Con este fin, hace una llamada a la función *reorderCameraCapturePanel*, cuyo cometido es cambiar los tamaños de la parte que se encarga de mostrar la imagen de la cámara. Adicionalmente, la función *hidePanels* se encarga de la visualización de los paneles relacionados con la configuración de la cámara.

La clase *CompleteWidgetActivity* está asociada a la última vista de la aplicación, cuyo diseño está descrito en el archivo *activity_default_widgets.xml* y se puede apreciar en la Figura 5.7. Tomando como referencia dicha imagen, se va a proceder a explicar cada una de las partes que componen la vista.

Figura 5.7: Vista de la interfaz gráfica del piloto.

Empezando por la parte alta de la vista, se aprecian una serie de iconos en la barra superior. De izquierda a derecha, se muestra el estado antes de volar del dron, el tipo de modo de vuelo, la señal *GPS*, la calidad de la visión, la señal del mando remoto, la señal del vídeo, la señal de *WiFi*, la batería restante y el estado de la conexión.

Inmediatamente debajo de esta barra y situado a la derecha, se encuentran todas las características de la configuración de la cámara, tales como el foco, el brillo o el obturador, así como un menú en el lateral derecho que sirve para controlar las operaciones respecto a la grabación de vídeo o capturas de imágenes.

En la parte inferior derecha se halla la imagen del punto de vista del dron o bien del mapa, el cual se intercambia de acuerdo con el modo de vídeo seleccionado (primario o secundario), como se ha explicado anteriormente. A la misma altura y situado a la izquierda de la vista se aprecia el compás y una serie de métricas derivadas del vuelo del dron.

Finalmente, en la parte superior del compás, se sitúan dos importantes botones cuyas funciones son devolver el dron al punto de partida y despegar, respectivamente.

5.1.4. Operaciones y transformaciones

La última parte de la primera versión de la interfaz versa sobre los cálculos externos que realizan algunos de los módulos descritos con anterioridad. Se trata de un bloque auxiliar que se compone de tres clases Java, empleadas como soporte a los cálculos que se realizan en la pantalla de la interfaz gráfica:

- *BitmapTransformer*: su objetivo es generar una imagen tipo *Bitmap* a partir de un *drawable* previamente creado, para lo cual emplea las propiedades de la función *createBitmap* de *Android*.
- *DensityUtil*: realiza un escalado de unidades de píxeles (*px*) a píxeles de densidad independientes (*dip*), con el fin de recalibrar tamaños según las vistas que así lo requieran.
- *Transformations*: esta clase se emplea para obtener la dirección y el rango de recorrido que ocupará el dron. Para ello se crea la función llamada *getHeadingRange* que se sirve de las funciones auxiliares *toRad* y *toDeg*, cuyo cometido es la transformación angular de radianes a decimales y viceversa. Además, emplea la función *distance*, que recibe las coordenadas de los puntos de partida y destino para calcular el rango.

5.2. Primera validación

Las validaciones que se describen en este documento forman parte del plan de validación desarrollada para el proyecto *BUBBLES* [21]. En concreto, se presentan aquí los detalles relevantes a los planes 2 (primera validación) y 5 (segunda validación) de dicho documento, puesto que son los directamente relacionados con el trabajo realizado.

5.2.1. Descripción y escenarios

En el caso de la primera validación, los vuelos planificados se desarrollaron en distintas zonas de Valencia [21]. Así pues, se definieron una serie de casos de uso representativos, según las operaciones que se espera que se realicen en cada uno de los dos escenarios descritos a continuación:

1. Escenario 1: se corresponde a zonas rurales en espacio aéreo no controlado, donde la mayoría de las operaciones tienen bajo riesgo, como operaciones agrícolas, inspecciones lineales o transporte *BVLOS* entre ciudades.
2. Escenario 2: es un entorno de riesgo medio en espacio aéreo controlado dentro del *CTR* del aeropuerto pero fuera del entorno aeroportuario.

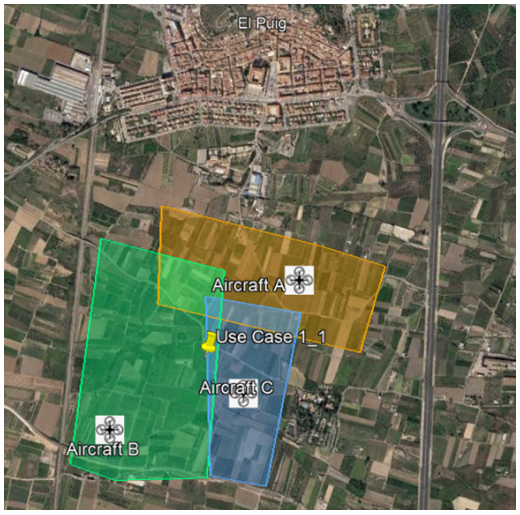
Para recopilar los datos necesarios, los conflictos no se generan espontáneamente, sino que se inducen. Para cada escenario, se han definido varios casos de uso que implican de dos a tres *UAS* y el desencadenamiento de un conflicto, lo que da como resultado un total de nueve casos de uso, que se describen detalladamente en las Tabla 5.1 y Tabla 5.2, respectivamente. Por su parte, los casos de uso para los distintos escenarios se presentan en las ilustraciones de las Figura 5.8 y Figura 5.9.

Escenario 1					
Caso		Dron	Misión	Conflicto	Localización
1.1	A	*Matrice 300 (A3)*	Fumigación de cultivos	Un dron *Mavic* se desvía de su ruta, por ejemplo, por un fallo de navegación debido a interferencias electromagnéticas, y se acerca a los otros drones, provocando un conflicto.	El Puig (39º34’29’’ N, 0º18’33’’ W)
	B	*Mavic 2 Ent. (STS-ES-02)*	Inspección de línea de alta potencia		
	C	*Mavic 2 Ent (A3)*	Tareas de agricultura		
1.2	A	*Matrice 300 (STS-ES-02)*	Suministro médico entre un pueblo y una casa aislada	El dron *Mavic* se sale del volumen autorizado. Ejemplo: el fotógrafo quiere hacer tomas desde fuera de su volumen aprobado.	Chóvar (39°50’57.76”N, 0°19’17.41”W)
	B	*Mavic 2 Ent. (A2)*	Fotografía		
	C	*Mavic 2 Ent. (A3)*	Tareas de agricultura		
1.3	A	*Mavic 2 Ent. (STS-ES-02)*	Vigilancia de prevención de incendios	El dron *Mavic* detecta humo cerca y abandona su zona de operación antes de que se actualice su plan de vuelo.	Gilet (39º39’32’’ N, 0º19’19’’ W)
	B	*Mavic 2 Ent. (A3)*	Tareas de agricultura		
	C	*Matrice 300 (A3)*	Misión de fotogrametría en parcelas agrícolas		
1.4	A	*Mavic 2 Ent. (A1)*	Ocio	El dron *Matrice 300 RTK* detecta una actividad sospechosa y procede a seguir al vehículo.	Costa de Puçol (39°36’40.29”N, 0°15’59.13”W)
	B	*Matrice 300 (STS-ES-02)*	Vigilancia de los robos de cultivos		
	C	*Mavic 2 Ent. (A3)*	Tareas de agricultura		

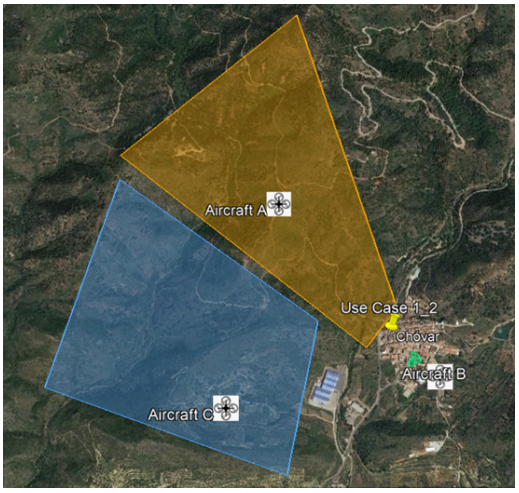
Tabla 5.1: Descripción de las características de los casos de uso en el escenario 1.

Escenario 2					
Caso		Dron	Misión	Conflicto	Localización
2.1	A	*Matrice 300 (STS-ES-01)*	Transporte de equipo médico al centro de salud de Alboraya	El dron de la policía es un usuario prioritario y el servicio de autorización de vuelo no actualiza el *Matrice* a tiempo, provocando un conflicto.	Valencia-Alboraya (39º 29’ 27’’ N, 0º 21’ 10’’ W)
	B	*Mavic 2 Ent. (A2)*	Actuación policial tras accidente en Ronda Norte de Valencia		
2.2	A	*Matrice 300 (STS-ES-01)*	Inspección de líneas ferroviarias	El dron de rodaje pierde temporalmente el control y abandona su área de operación.	Massanassa (39°24’23.06”N, 0°23’52.59”W)
	B	*Mavic 2 Ent. (STS-ES-01)*	Rodaje		
2.3	A	*Matrice 300 (STS-ES-01)*	Transporte en un parque industrial	El inicio del vuelo del *Mavic* se retrasa y crea un conflicto con el otro dron.	Parque industrial Alfafar (39°24’54.54”N, 0°23’4.10”W)
	B	*Mavic 2 Ent. (STS-ES-01)*	Construcción en un parque industrial		
2.4	A	*Mavic 2 Ent. (STS-ES-01)*	Vigilancia de la playa	El dron de transporte es un usuario prioritario y el servicio de autorización de vuelo no actualiza el dron *Matrice* causando un conflicto.	Playa de la Malvarrosa (39°28’36.79”N, 0°19’26.84”W.)
	B	*Matrice 300 (STS-ES-01)*	Transporte de larga distancia desde el Hospital de la Malvarrosa al Norte de Valencia		
2.5	A	*Matrice 300 (STS-ES-01)*	Transporto logístico en el puerto de Valencia	Aparecen niños entrando en el mar y el dron sale de su volumen operativo para llamar su atención desde el altavoz.	Puerto de Valencia (39°26’32.94”N,0°18’55.23”W)
	B	*Mavic 2 Ent. (STS-ES-01)*	Vigilancia del puerto de Valencia		

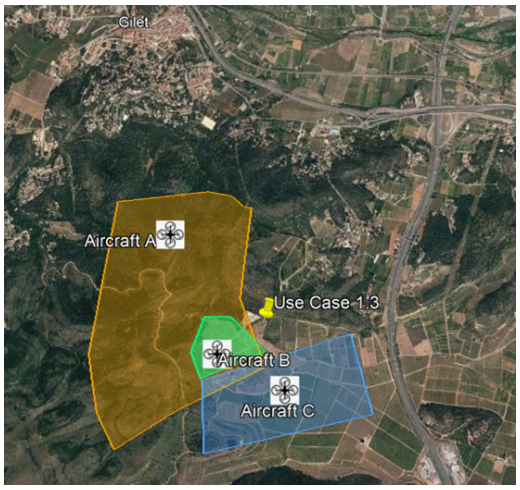
Tabla 5.2: Descripción de las características de los casos de uso en el escenario 2.



(a) Caso de uso 1.1



(b) Caso de uso 1.2



(c) Caso de uso 1.3

(d) Caso de uso 1.4

Figura 5.8: Localización de los casos de uso del escenario 1 [21].

(a) Caso de uso 2.1

(b) Caso de uso 2.2

(c) Caso de uso 2.3

(d) Caso de uso 2.5

(e) Caso de uso 2.4

Figura 5.9: Localización de los casos de uso del escenario 2 [21].

5.2.2. Evaluación

Una vez descrita las características de la primera validación, a continuación se exponen los resultados. Estas pruebas incluyen la evaluación elemental del rendimiento humano mediante cuestionarios basados en los desarrollados por *EUROCONTROL* y convenientemente adaptados a los servicios *U-space*.

El objetivo del cuestionario es recibir comentarios de los pilotos sobre los distintos factores que afectan a la gestión táctica de conflictos, especialmente los que intervienen en el cálculo de las separaciones mínimas. Los pilotos rellenaron los formularios al finalizar cada misión.

En la primera parte del cuestionario, se preguntaba a los pilotos cómo valorarían el impacto de los servicios *U-space* proporcionados por las plataformas en el éxito de su misión, su carga de trabajo y su conocimiento de la situación, cuyos resultados se aprecian en la Figura 5.10.

Más del 75 % considera que el impacto de los servicios *U-space* prestados es neutro en el éxito de su misión. Esto se debe a que actualmente la plataforma sólo avisa de un conflicto táctico o de conformidad, pero no proporciona información de tráfico sobre las otras aeronaves en conflicto, por lo que los pilotos mencionan que la plataforma es útil, pero que necesitan más información.

Por la misma razón, alrededor del 50 % no valora positivamente el impacto de los servicios *U-space* prestados sobre su carga de trabajo, ya que no disponen de información suficiente para resolver los conflictos y tener que buscar drones con la vista resulta a veces complicado.

En la segunda parte del cuestionario, se preguntó a los pilotos cómo calificarían la utilidad de la plataforma *BUBBLES* en una evaluación a alto nivel de sus principales características. Los resultados se muestran en la Figura 5.11.

Casi el 70 % considera positiva la facilidad de uso de la plataforma, lo cual contrasta con el hecho de que solo el 8 % evalúa positivamente su utilidad. Los pilotos se quejan de que el mensaje *"¡Tactical Conflict!"* no es suficiente información y que sería bueno conocer la gravedad del conflicto, es decir, el volumen infringido.

En cuanto a la claridad de la información facilitada, la fiabilidad de los datos, lo cual comprende disponibilidad, pertinencia, exactitud, entre otros, y la coherencia, alrededor del 50 % la calificaron de neutra. Por su parte, la rapidez de la interfaz obtuvo resultados mixtos, si bien es cierto que hubo más porcentaje de pilotos valorándola positiva que negativamente.

Por tanto, la conclusión de la segunda parte del cuestionario es que para los pilotos la información que llega es un único mensaje de *"¡Tactical Conflict!"*, pero ni el número de conflictos ni la distancia o dirección a la que se encuentra el *UAS* en conflicto. Los resultados del cuestionario son muy mixtos, ya que tienden hacia la neutralidad.

En la tercera parte del cuestionario, se preguntó a los pilotos si la herramienta ha sido útil para resolver el conflicto, a lo que el 69 % respondió que no. Al igual que antes, la valoración fue negativa porque no se proporcionó suficiente información a los pilotos. Estos afirmaron que les gustaría recibir información adicional sobre la posición y la distancia de los otros drones en conflicto, la orientación (rumbo con respecto a la aeronave en conflicto), el número de conflictos, la diferencia de altitud. También señalaron que sería interesante disponer de una interfaz gráfica o de un mapa 2D para consultar esta información.

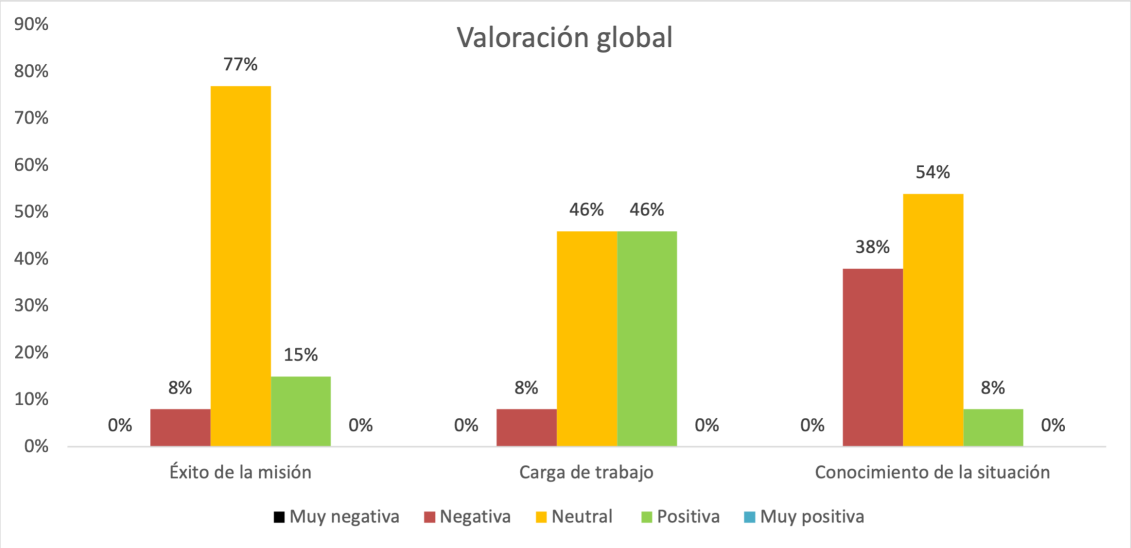


Figura 5.10: Evaluación global de la plataforma sobre las misiones en *U-space* en la primera validación [22].

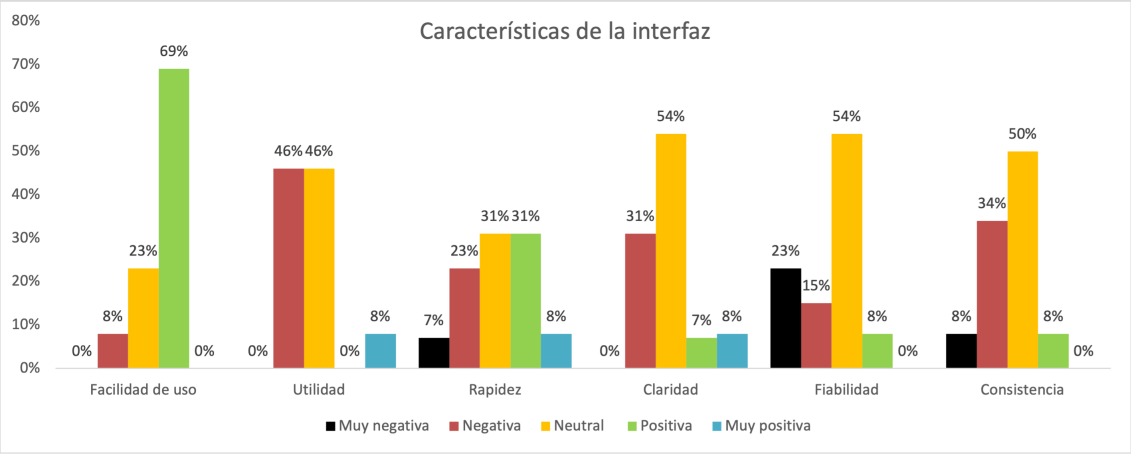


Figura 5.11: Evaluación de las características de la interfaz en la primera validación [22].

5.3. Segunda versión

En esta sección de la memoria se expondrán todos los cambios realizados desde la primera versión, siguiendo la misma estructura que se ha realizado con anterioridad. En vista de las recomendaciones propuestas en la primera validación se decidió ampliar las funcionalidades de la interfaz, proviniendo así a los pilotos del modo de vuelo autónomo mencionado a lo largo de la arquitectura. Por tanto, las prioridades para esta segunda versión son:

1. Control sobre la capacidad de vuelo autónomo de la aeronave.
2. Posibilidad de cargar una ruta de vuelo predeterminada.
3. Presentación de los conflictos tácticos con mayor información.
4. Muestra del estado de las comunicaciones en la interfaz gráfica.

De acuerdo con estas nuevas especificaciones se producen varios cambios en la jerarquía de clases del código de la aplicación y se introducen modificaciones en los archivos descritos en la sección que abarca la primera versión. En la Figura 5.12 se puede apreciar el nuevo contenido del código, que se detallará en los siguientes apartados.

5.3.1. Vuelo autónomo

La funcionalidad de vuelo autónomo por parte de la aeronave es una de las características más destacadas de la interfaz. Para cumplir con las prioridades de la nueva versión, se necesitaba transformar archivos con formato *KML* para poder cargar y ejecutar las rutas de vuelo en el dron y emplear las características de la documentación del *SDK* para poder controlar dicha ruta a lo largo de su recorrido. Así pues, y como se desprende de la Figura 5.12, la incorporación de esta especificación a la interfaz requería la creación de tres nuevas clases Java: *KMLmanager*, *FlightMission* y *AFPSelection*.

5.3.1.1. *KMLmanager*

La clase Java *KMLmanager* es la encargada de recabar toda la información relativa a la descripción de la ruta del archivo *KML* para que el *SDK* pueda realizar correctamente todas las operaciones que desee el usuario. Se compone de tres funciones llamadas *waypointConstructor*, *openFile* y *waypointParser*, donde la primera de estas hace llamadas a las siguientes dos para transformar una entrada de tipo *uri* (directorio donde se ubica el fichero) en una lista de *Waypoints* que se utilizarán para construir la misión.

La primera llamada se realiza a la función *openFile* que, mediante un bloque *try-catch*, intenta acceder al directorio y al archivo seleccionado por el operador de vuelo y cuya función es almacenar toda la información en una variable de tipo *String*. Para ello se sirve de elementos de gestión de archivos propios del lenguaje Java como *StringBuilder*, *InputStream* y *BufferedReader* y de funciones como *openInputStream*, *readLine*, *append* y *toString*.

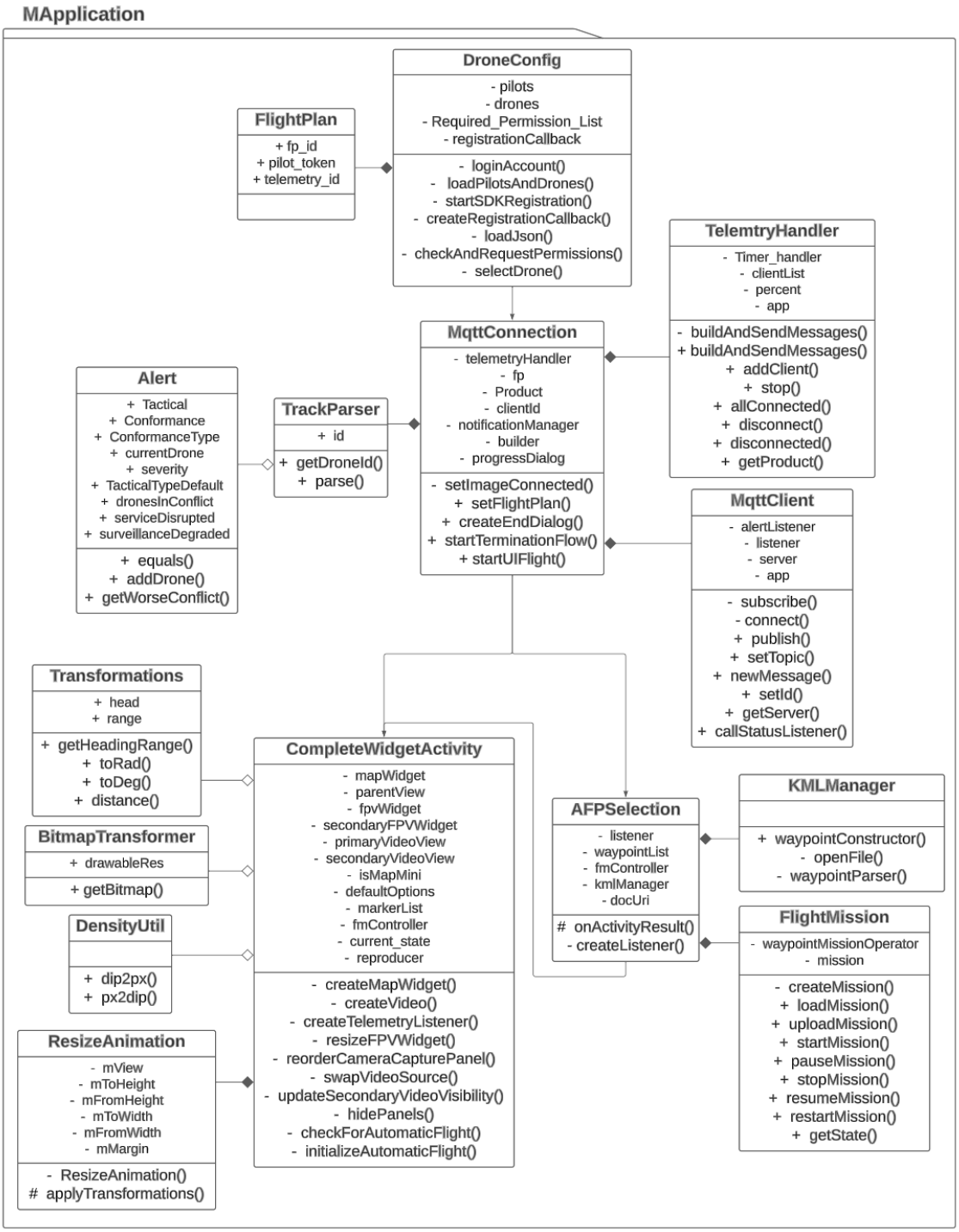


Figura 5.12: Diagrama de clases de la segunda versión de la interfaz.

Tras concluir con éxito la lectura del archivo, se llama a la función *waypointParser*, que recibe como parámetro de entrada la variable *String* obtenida con anterioridad. Básicamente, el propósito de esta función es recorrer dicha variable en busca de los distintos puntos georreferenciados por donde se pretende que circule el dron.

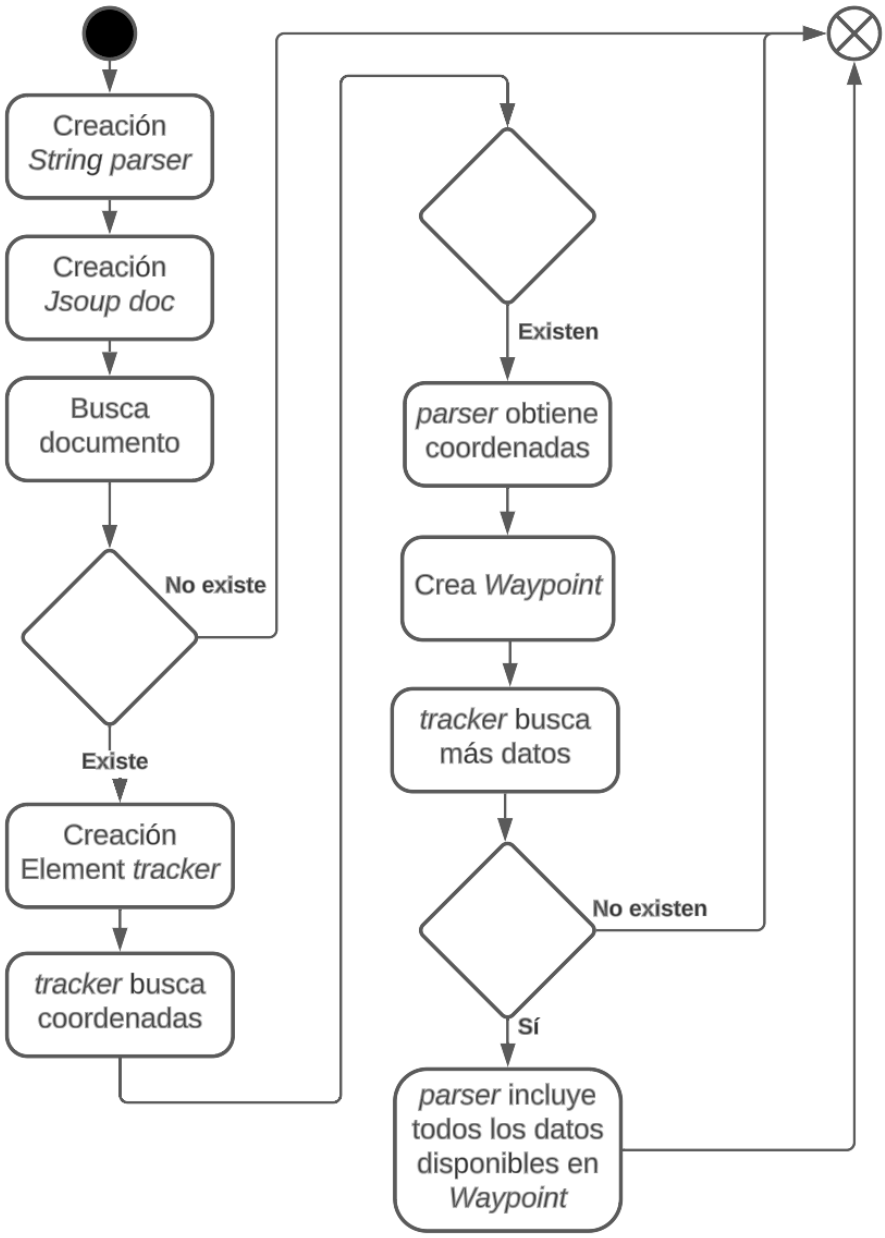


Figura 5.13: Diagrama de flujo del lector de *KML*.

Para ello, se emplea una biblioteca de Java llamada *jsoup*, cuyo fin es analizar y manipular datos almacenados en documentos. Con esta herramienta se inspecciona el *String* buscando las

coordenadas de los distintos puntos y, para cada uno de estos, se extraen las propiedades que permiten formar el recorrido. Las más importantes son la dirección, el cabeceo, la velocidad y el modo de giro. Este proceso se describe en el diagrama de flujo de la Figura 5.13.

Una vez se ha obtenido cada una de las propiedades, entonces se añaden al *waypoint* que llevará como coordenadas 3D las encontradas por el *parser*. Cuando se termina de obtener todas las características, se añade el *waypoint* a la lista que los almacena en el orden especificado. Dicha lista es el resultado que devuelve esta clase y que será empleado por *AFPSelection*.

5.3.1.2. *FlightMission*

Se trata de la clase encargada de gestionar toda la operativa relacionada con la ejecución del vuelo autónomo, así como de ofrecer la posibilidad de ejercer cierto control sobre la ruta en curso. Esto se consigue mediante el bloque de la arquitectura del *SDK Mission*. Debido a que el tipo de misión escogido para este proyecto es *Waypoint*, al crear esta clase se realiza una llamada *DJISDKManager* para obtener el *WaypointMissionOperator*, elemento que permite manipular las funciones del vuelo autónomo.

En total, la clase se compone de nueve funciones y cada una se encarga de realizar una acción distinta sobre el estado de la misión: *getState*, *createMission*, *loadMission*, *uploadMission*, *restartMission*, *startMission*, *pauseMission*, *stopMission* y *resumeMission*. Los nombres de las funciones son muy indicativos, pues las últimas cuatro se encargan de empezar, pausar, parar o resumir la misión en curso. Por su parte, la función *restartMission* hace uso de una combinación lineal de otras tres: *loadMission*, *uploadMission* y *startMission*.

La primera de las funciones nombradas, *getState*, se emplea para devolver el estado de la misión actual. A continuación, se describen los posibles estados y se presenta un diagrama de estados en la Figura 5.14 para facilitar su comprensión.

- *NOT_SUPPORTED*: el producto conectado no admite la misión de *Waypoint*, ya que, tal y como se ha expuesto en la memoria, no todo el *hardware* de *DJI* soporta dicha funcionalidad.
- *READY_TO_UPLOAD*: la misión está disponible para ser subida al producto.
- *UPLOADING*: la carga se ha iniciado correctamente y la información detallada de cada *waypoint* se carga individual y secuencialmente en el dron.
- *READY_TO_EXECUTE*: la misión *Waypoint* se ha cargado completamente y el dron está listo para iniciar su ejecución.
- *EXECUTING*: se ha iniciado de forma exitosa la ejecución de la misión y se está llevando a cabo el recorrido previsto.
- *EXECUTION_PAUSED*: la misión se ha pausado con éxito tras demandarlo el operador. Para continuar con la ejecución el usuario deberá volver a interactuar con la interfaz.
- *DISCONNECTED*: la conexión entre el dispositivo móvil, el mando a distancia y la aeronave está interrumpida.

- *RECOVERING*: se intenta establecer la conexión entre el dispositivo móvil, el mando a distancia y la aeronave con el fin de sincronizar el estado.
- *UNKNOWN*: el estado del operador de la misión es desconocido. De hecho, este es el estado inicial cuando se acaba de crear dicho elemento.

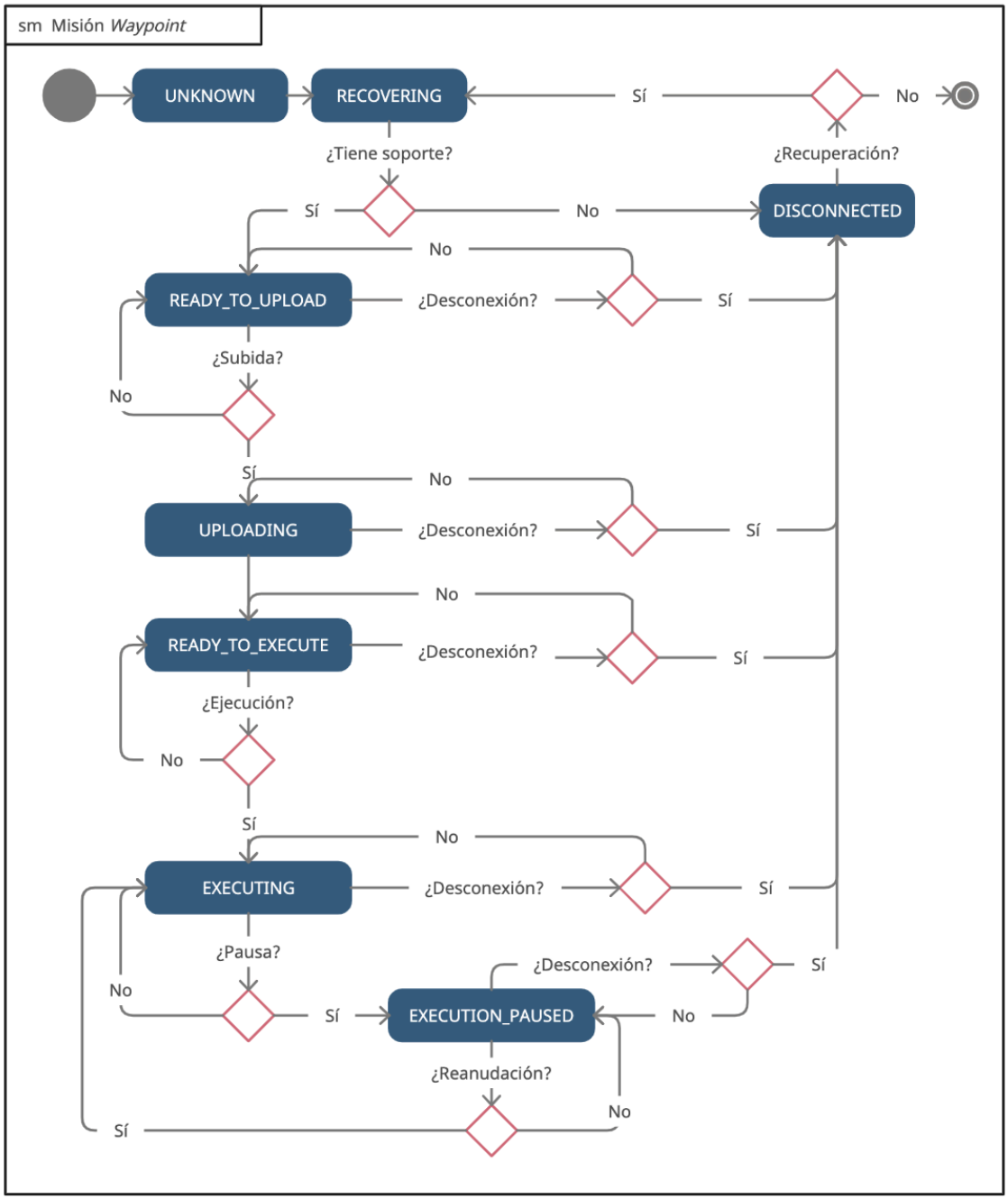


Figura 5.14: Diagrama de estados de la misión *Waypoint*.

Para finalizar la descripción de las funciones que se encargan de gestionar la configuración y ejecución del plan de vuelo autónomo, se expone la diferencia entre *loadMission* y *uploadMission*. La primera de ellas es la encargada de mediante el empleo de las funciones descritas en *KMLmanager* generar una misión *Waypoint* a partir de un fichero almacenado en el sistema. Para ello, se sirve de la función *createMission*, que emplea el constructor del *SDK WaypointMission.Builder*. En contraposición, *uploadMission* es la encargada de realizar la subir dicha misión al ordenador de abordo del dron.

5.3.1.3. *AFPSelection*

Esta es la clase cuyo objetivo es gestionar las interacciones con la vista de la aplicación referente a la configuración de vuelo autónomo, por lo que hace uso del archivo *afp_selection.xml* cuyo resultado se muestra la Figura 5.15.

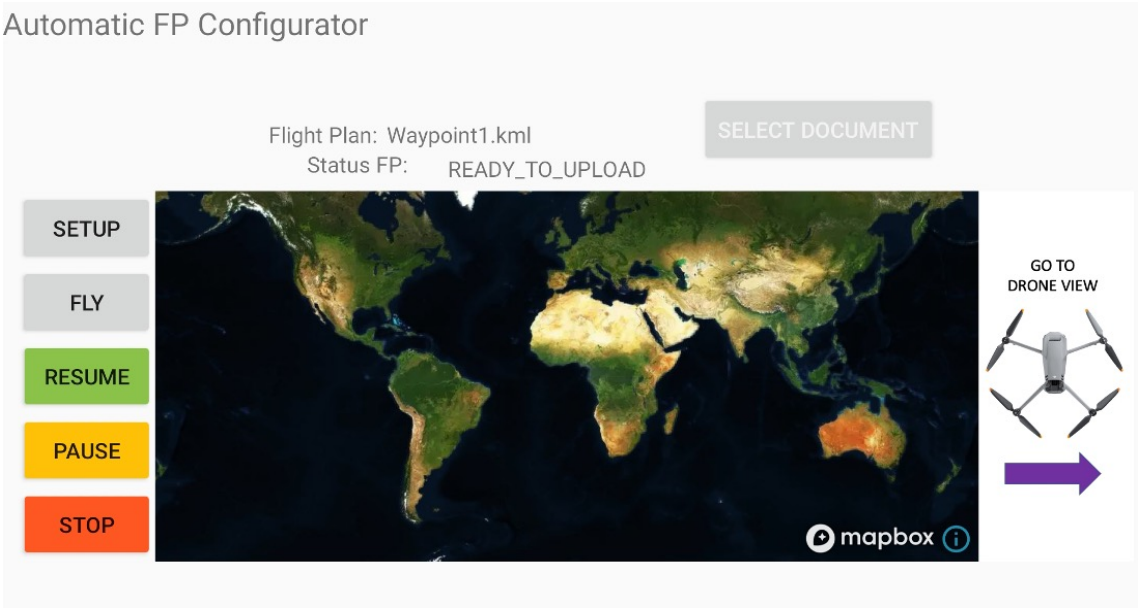


Figura 5.15: Vista del configurador y gestor del plan de vuelo autónomo.

En dicha imagen se aprecian los elementos que componen la vista de *AFPSelection*. En primer lugar, en la parte superior derecha se observa un botón con el texto *"Select Document"*, que una vez pulsado por el usuario, redirige al piloto al sistema de ficheros del sistema *Android* para que seleccione el archivo *KML* donde se encuentra la descripción del vuelo. En caso de seleccionar cualquier otro documento, la aplicación devolverá un error al piloto haciéndole constar que debe seleccionar un documento de este tipo. Con este fin, se sobrescribe la función nativa *onActivityResult* y se comprueba que los parámetros sean los adecuados.

A la misma altura que este botón y más centrado en la pantalla, aparecen dos textos que indican el nombre del plan de vuelo seleccionado y el estado en que se encuentra la configuración. Este coincide con los descritos con anterioridad y se actualiza de acuerdo con las acciones del usuario. Cuando aún no se ha seleccionado ningún documento, ambos textos están en blanco.

En la parte inferior se detectan tres columnas diferenciadas. A la izquierda se hallan cinco botones distintos, cuyas funciones están relacionadas con la carga del plan de vuelo al dron y su posterior control. El botón *"Setup"* realiza una llamada a la función *loadMission* para que se genere la misión *Waypoint* que el usuario haya seleccionado mientras que *"Fly"* hace uso de *uploadMission* para empezar con la misión. Por su parte, *"Resume" "Pause"* y *"Stop"* realizan llamadas a las respectivas funciones *resumeMission*, *pauseMission* y *stopMission*.

Estos botones aparecen en esta pantalla, donde el usuario aún no está viendo las cámaras del dron, con el fin de corregir cualquier error por parte del piloto lo antes posible o para poder actuar ante la situación de un posible reinicio de la aplicación. Para ello, es necesario definir en la clase una objeto de tipo *WaypointMissionOperatorListener*, que permitirá realizar las actualizaciones del estado actual, así como guiar al piloto en la ejecución del vuelo autónomo.

Por su parte, en el centro se muestra una imagen del mapa y a la derecha se tiene un botón que traslada el usuario a la vista de la interfaz gráfica de vuelo del dron. Se ha decidido incluir el mapa en forma de imagen para embellecer esta pantalla, de forma que no se puede manipular ni acceder a las funciones propias de *MapBoxMap*.

5.3.2. Modificaciones

En esta parte de la segunda versión se van a detallar las modificaciones introducidas en la segunda versión respecto al código ya descrito durante la versión inicial.

La primera de ellas, como se observa en la Figura 5.16, es la posibilidad de, desde la pantalla de la configuración de la conexión *MQTT* (*MQTTConnection*), seleccionar el modo de vuelo deseado: manual (dirige a *CompleteWidgetActivity*) o automático (accede a *AFPSelection*).

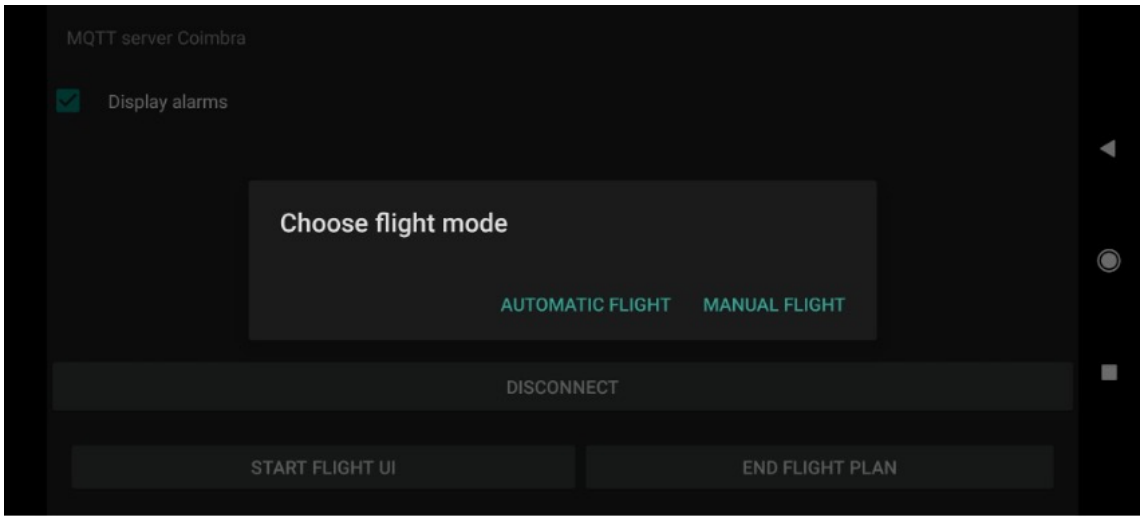


Figura 5.16: Segunda versión de la vista *MQTTConnection* para la selección de vuelo.

El cambio se ha introducido en la función *startUIFlight*, función que se activa tras pulsar el botón *"Start Flight UI"*. En lugar de lanzar directamente la vista de interfaz gráfica de vuelo, lo

que aparece ahora es un diálogo de alerta donde se le ofrecen las opciones de operativa al usuario y se le dirige hacia las mismas cuando este pulse sobre cualquiera de ellas.

La segunda gran modificación de esta versión se encuentra precisamente en la vista de la interfaz gráfica, ya que está relacionada con las recomendaciones que los pilotos realizaron durante la primera validación. En los cuestionarios se detallaba el hecho que la información sobre las alertas no era lo suficientemente descriptiva, por lo que se decidió mejorar esta parte. En la Figura 5.17 se muestran dichos cambios.

Figura 5.17: Modificaciones en la segunda versión de la vista *CompleteWidgetActivity*.

1. Botón *Reload*: permite repetir la última misión de vuelo automático. Para ello realiza una llamada a la función *restartMission* de la clase *FlightMission*. Esto se incorporó como consecuencia de las peticiones de los pilotos que querían volver a empezar con el mismo plan sin repetir todo el proceso.

2. Icono *MQTT*: indica el estado de la conexión del dron al servidor *MQTT*. Como se ha descrito anteriormente, tiene tres estados posibles:

   - Gris: no se ha realizado la conexión con el servidor.
   - Verde: está conectado y se envía/recibe la telemetría.
   - Rojo: se ha perdido la conexión.

3. Indicador de altitud (*Height*): su función es mostrar la altitud del dron en metros. Los pilotos requerían que este parámetro estuviese más fácilmente visible dentro de la pantalla, ya que debían tomar decisiones cuando se les presentaba un conflicto.

4. Botón *Stop*: este sólo será visible en el modo de vuelo automático y permite parar la misión, mediante el uso de la función *stopMission*. Antes de ejecutarse, preguntará al usuario si efectivamente quiere detener la misión, lanzando un diálogo de alerta. Si se detiene la misión, el dron vuelve al punto de inicio.

5. Botón *Play/Pause*: de igual forma que con el anterior, este sólo será visible en el modo de vuelo autónomo. Sus funciones son las de pausar (*pauseMission*) o reanudar (*resumeMission*) la misión.

6. Icono de conflicto táctico: se trata de un visual gráfico en forma de dron con distintos colores para indicar la severidad del conflicto más importante con drones adyacentes. Los posibles estados coinciden con los volúmenes de separación para el modelo de colisión de *BUBBLES* descritos en la Figura 4.2:

   - Morado: *NMAC*.
   - Rojo: colisión inminente .
   - Naranja: pérdida de separación.
   - Amarillo: conflicto táctico.
   - No se muestra el icono cuando no existe conflicto alguno.

7. Icono *QoS*: muestra la calidad de servicio en cuanto a si se están recibiendo con éxito los paquetes del *tracker* o si por el contrario se producen errores en la comunicación. Existen tres posibles estados:

   - Verde: no hay pérdidas, el funcionamiento es correcto.
   - Naranja: existe bien degradación de control o bien de comunicación.
   - Rojo: ambos problemas están presentes.

Como ya se ha podido apreciar, la creación de un plan de vuelo autónomo impactó en el diseño de la vista de interfaz de usuario. A continuación, se detallan los cambios realizados en el código de la clase *CompleteWidgetActivity* que hacen posible interactuar con todos estos elementos.

Para gestionar las interacciones se crearon dos funciones en la clase Java, *initializeAutomaticFlight* y *checkForAutomaticFlight*, la cual servía a la primera para comprobar tanto si el usuario había iniciado el modo autónomo como si el producto disponía de dicha posibilidad. Para ello, realizaba una comprobación de la variable global *isAutomatic*, creada para diferenciar entre los dos modos junto con la verificación que los estados del operador de misiones fueran distintos de *NOT_SUPPORTED* y *UNKNOWN*.

Una vez este chequeo es positivo, es decir, el usuario ha activado el modo autónomo y el dron soporta dicho modo, entonces la función *initializeAutomaticFlight* se encarga de crear un *listener* sobre el operador de misiones *Waypoint* con el fin de sobrescribir las funciones *onExecutionStart* y *onExecutionFinish*. La primera sirve para hacer visibles los botón de detención (*stop*) y resumen/pausa (*play/pause*), así como imprimir el texto *Starting Mission* para advertir al usuario. Por su parte, la segunda lanzará el texto *Mission execution finished* por pantalla, mientras desactiva la visibilidad de los otros botones y activa el botón para reanudar la misión (*restart*).

Posteriormente, también iniciará un objeto controlador de tipo *FlightMission*, ofreciendo dicho *listener* para obtener los estados de la misión y ofrecer las funciones de manejo de la misión al usuario, a través de los botones. En concreto, según el estado de la misión aparecerá la opción de iniciar o pausar la misión, para lo cual se emplea este controlador.

Este objeto también se emplea cuando se pulsa el botón de detener de la misión, ya que si la misión se está ejecutando aparecerá un diálogo para interrumpir la ejecución de la misión, mientras que si no se está ejecutando el diálogo que aparecerá será para reiniciar la misión.

5.3.3. Simulación

Con motivo de la introducción de estas nuevas características en el diseño de la aplicación, se decidió realizar unas pruebas de simulación en el laboratorio de trabajo, previo a las segundas pruebas de validación. Su finalidad era comprobar que el funcionamiento de los códigos implementados cumplían con su cometido.

Para ello se empleó como entorno de trabajo el portátil del alumno, un programa *software* propietario de *DJI* llamado *DJI Asssitant 2 (Consumer Drone Series)* (Figura 5.18), el modelo de *UAS DJI Air 2S* y el mando *DJI RC Pro* para controlar el dron (Figura 5.19).

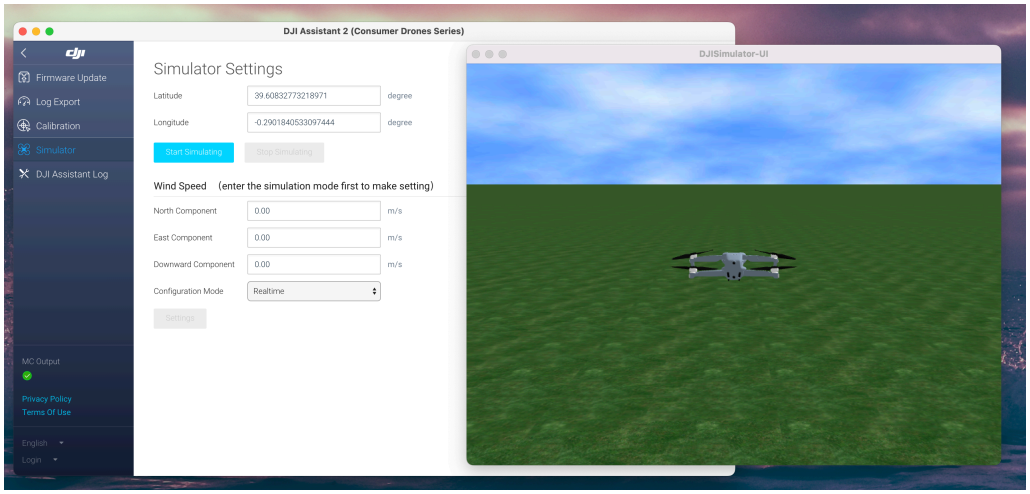


Figura 5.18: Programa de *DJI* empleado para la simulación.

Figura 5.19: Equipos físicos empleados en la simulación.

Mediante estas herramientas, se creó un archivo de prueba en formato *KML* con hasta 5 *Waypoints*, diseñando la simulación de una posible ruta que debería seguir el dron. Tras esto, se cargó el archivo en el sistema de ficheros del mando de *DJI*, lo que permitiría realizar la posterior carga en el dron.

El flujo de desarrollo que se llevó a cabo durante el período de simulación fue el siguiente:

1. Descarga de la aplicación en el mando de *DJI*.
2. Selección del documento *KML* mediante la ventana *AFPSelection*.
3. Carga del plan de vuelo en el dron.
4. Ejecución del plan de vuelo a través de *DJI Asssitant 2 (Consumer Drone Series)*.

Si en alguno de los anteriores eventos, la aplicación se detenía o surgía cualquier error inesperado, entonces se inspeccionaba la parte del código que estaba fallando y se intentaba solucionar el problema. De forma iterativa, se volvía al inicio del proceso hasta completar todos los pasos y constatar que la nueva parte de código creada era funcional.

Durante estas simulaciones fue donde se usó el modo de vuelo de comandos de *joystick* virtual. Su mayor cometido fue realizar aterrizajes forzosos durante las pruebas, así como comprobar que el dron podía realizar el despegue correctamente.

Fueron necesarias muchas modificaciones para conseguir obtener una versión funcional de la nueva característica. La mayoría de los problemas surgieron en torno a la subida del archivo al dron y a fallos en la ejecución del plan de vuelo que se tuvieron que corregir. Además, también se encontraron errores peculiares, como el hecho de que los *KML* generados tenían las coordenadas de latitud y longitud invertidas, por lo cual en el programa de simulación el dron no empezaba con la ruta, debido a una descolocación de su posición.

Una vez embebido esta nueva parte dentro de la interfaz, se consiguió el objetivo marcado de construir una pantalla que gestionase la característica de vuelo autónomo propia de los equipos *DJI*. El resultado se ha expuesto a lo largo de las anteriores secciones de esta segunda versión de la herramienta. Tras finalizar, tan solo quedaba realizar las pruebas a tiempo real, objeto de discusión de la siguiente sección.

5.4. Segunda validación

5.4.1. Descripción y escenarios

En este ejercicio de pruebas se planificaron vuelos en una zona rural de 15 km^2 en Puçol y El Puig, pueblos del norte de Valencia (España), que se encuentran en espacio aéreo no controlado. Dicha campaña de vuelos se realizó en los días 11 y 12 de abril de 2022 [22], con la participación de siete operadores y un total de 14 drones, de los cuales sólo uno no pertenecía a la empresa *DJI*:

- Operador 1: *WUAS-UPV*, con dos *DJI Mavic 2 Enterprise Zoom*; un multirrotor pequeño con un peso inferior a 1,1 kg y un *DJI Matrice 300 RTK*; también multirrotor de tamaño medio con 9 kg de *MTOW*.
- Operador 2: Policía Local de Benidorm, con un *DJI Mavic 2 Enterprise Zoom* y un *DJI Mavic 2 Enterprise Dual* (pequeño multirrotor de peso inferior a 1,1 kg).
- Operador 3: AsDrón España, con un *DJI Phantom 4 PRO*, un dron multirrotor de pequeño tamaño y peso inferior a 1,4 kg.
- Operador 4: *UAV Works*, con un *Valaq Patrol*, un *UAS* de ala fija con 4,5 km de *MTOW*. Un dron con hasta 60 minutos de autonomía y 70 km de alcance construido por ellos mismos.
- Operador 5: Bomberos de Valencia, con dos *DJI Mavic 2 Enterprise Advanced* (pequeño multirrotor de menos de 1 kg de peso).
- Operador 6: Policía Local de Valencia (PLV), con dos *DJI Mavic 2 Enterprise Advanced* y dos *DJI Mavic Enterprise Dual*, un pequeño multirrotor de peso inferior a 1 kg.
- Operador 7: drones ASD, con un *DJI Mavic 2 Enterprise Advanced*.

Con todos estos operadores y sus respectivos drones se realizaron una serie de misiones que se pueden apreciar en la Figura 5.20, donde se muestra la zona en la que se realizaron los vuelos y las trayectorias de referencia previstas.

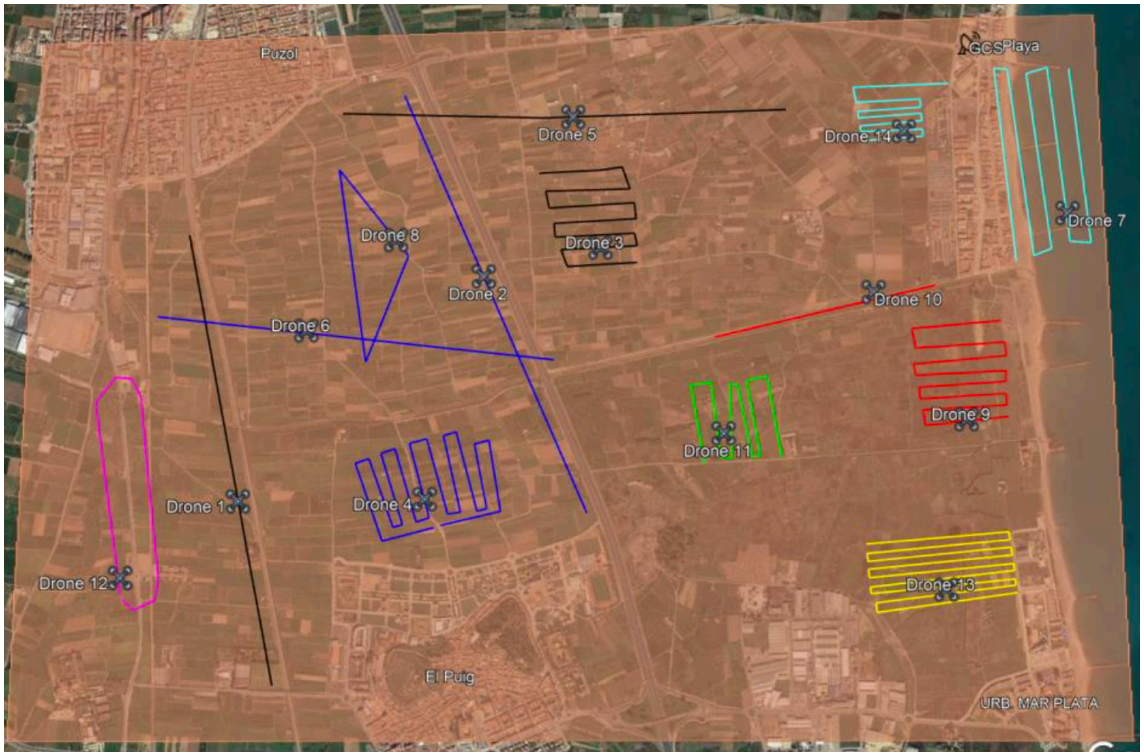


Figura 5.20: Mapa de las rutas de las misiones realizadas en la segunda validación [22].

5.4.2. Evaluación

Para esta segunda validación, tanto el objetivo, obtener información de los pilotos sobre su experiencia con la plataforma y las mejora existente, junto con el procedimiento, realizar un cuestionario al final de las pruebas, coinciden con el procedimiento llevado a cabo durante la primera evaluación de la interfaz.

En la primera parte del cuestionario, se preguntaba a los pilotos cómo valorarían el impacto de los servicios *U-space* prestados por las plataformas en el éxito de su misión, su carga de trabajo y su conocimiento de la situación. Cabe mencionar que, en este caso, a diferencia de la primera validación, las respuestas *Muy negativas* y *Negativas* y *Muy positivas* y *Positivas* se han agrupado en *Negativas* y *Positivas*, respectivamente, para mayor claridad en la interpretación de los resultados.

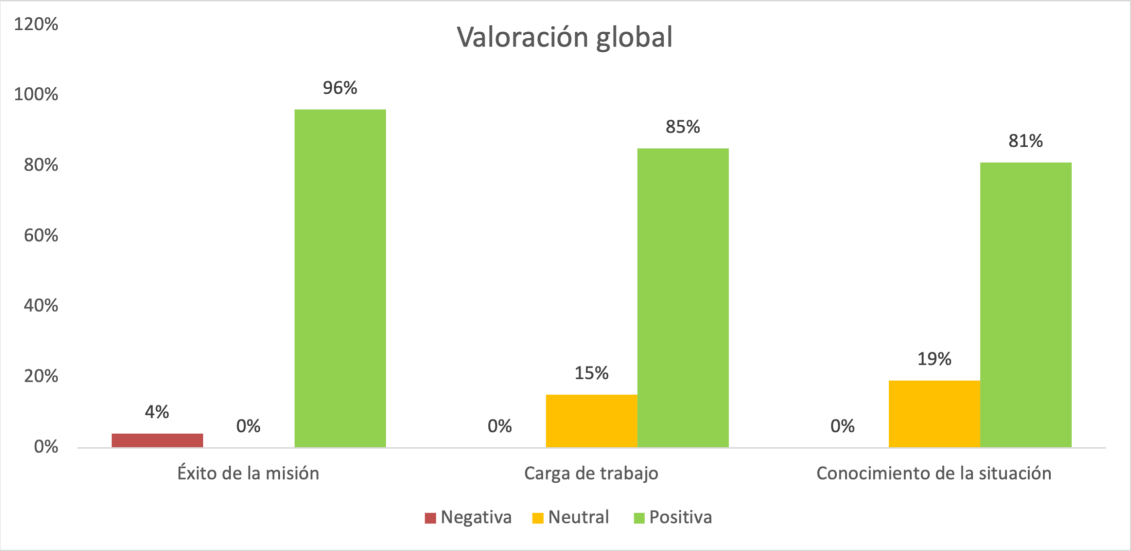


Figura 5.21: Evaluación global de la plataforma sobre las misiones en *U-space* en la segunda validación [22].

Más del 96 % considera positivo el impacto de los servicios *U-space* prestados en el éxito de su misión. Además, alrededor del 85 % valora positivamente el impacto de los servicios *U-space* prestados en su carga de trabajo.

En cuanto al impacto de los servicios *U-space* proporcionados en su conocimiento de la situación, el 80 % lo valora positivamente, mientras que el 20 % se mantiene neutral, porque al principio de las pruebas los pilotos no estaban familiarizados con la interfaz de la aplicación y no sabían interpretar bien la información.

Como se puede observar, en esta segunda fase de vuelos, los resultados globales son mucho mejores que durante las primeras pruebas (Figura 5.10), ya que se han mejorado las herramientas según las sugerencias recibidas en los primeros cuestionarios.

Similarmente, en la segunda parte del cuestionario, se preguntó a los pilotos cómo calificarían la utilidad de la plataforma en una evaluación de alto nivel. A diferencia con la primera validación,

esta vez se decidió eliminar la pregunta acerca de la consistencia. Los resultados se presentan en la Figura 5.22.

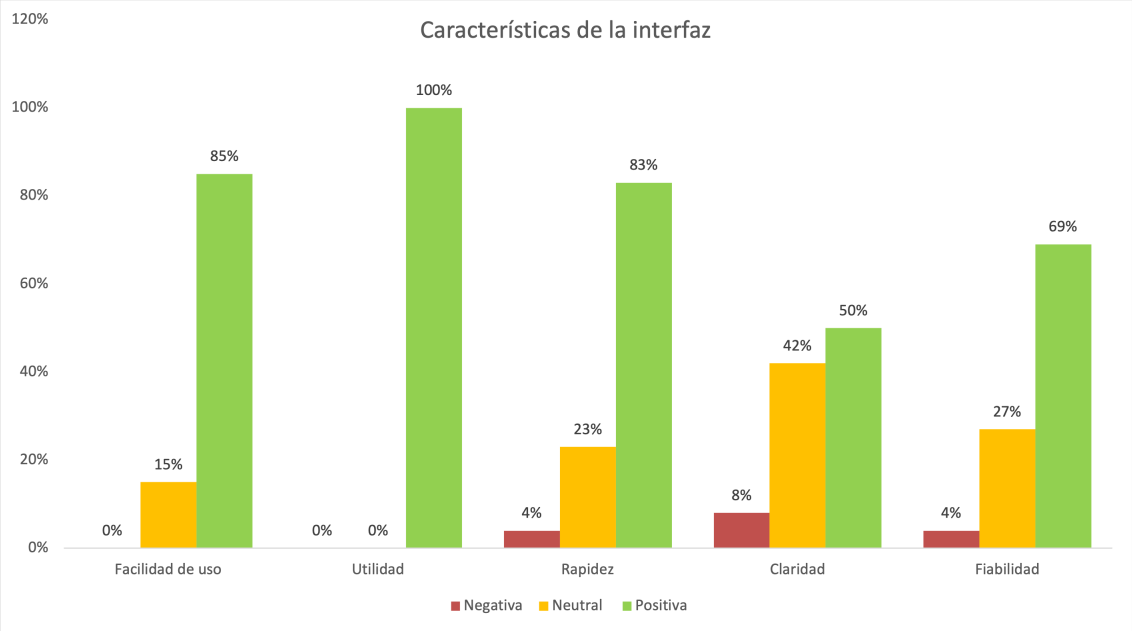


Figura 5.22: Evaluación de las características de la interfaz en la segunda validación [22].

Alrededor del 85 % considera positiva la facilidad de uso de la plataforma mientras que nadie valora negativamente su utilidad. A continuación, el 83 % considera la puntualidad de la información positiva y sólo un 4 % la considera negativa, debido a pérdidas puntuales de conexión 4G.

Acerca de la claridad de la información proporcionada, el 50 % la valora positivamente pero el 8 % la considera negativa. Esto se debe a que la maniobra de resolución de conflictos verticales propuesta se realizaba con un signo negativo en la distancia vertical y creaba confusión en el piloto. Por último, alrededor del 70 % responde positivamente sobre la fiabilidad de los datos.

Como puede verse, esta parte de las respuestas al cuestionario también es mucho más positiva que en la campaña de vuelo de la primera fase de validación (Figura 5.11). Sin embargo, una de las quejas más destacadas en esta parte es que las alertas duraban muy poco tiempo a los mandos, no dando tiempo al piloto a interpretar bien el mensaje en algunos casos.

Adicionalmente, los pilotos también sugirieron incluir avisos sonoros o hacer más visual el icono que marca la gravedad del conflicto. Respecto a las maniobras, los pilotos dijeron que sería mejor cambiar el tipo de indicación, por ejemplo, cambiar el signo positivo/negativo por flechas. Todo esto junto con la visión de la posición de los drones en conflicto en el mapa del mando serían algunos aspectos para mejorar en la aplicación.

En la tercera parte del cuestionario, se preguntó a los pilotos si la herramienta ha sido útil para resolver el conflicto, a lo que el 100 % respondió afirmativamente. A continuación, también el 100 % dijo que es útil conocer la gravedad del conflicto, es decir, la separación mínima que se infringe.

Sobre la pregunta acerca de la necesidad de información adicional, los pilotos sugirieron que sería interesante recibir la posición del otro dron en conflicto en el mapa de mando, así como su rumbo y velocidad. Al igual que en el apartado anterior, también afirmaron que sería necesaria una recomendación de maniobra más comprensible y directa, así como avisos sonoros según la gravedad para que si se está concentrado en vigilar al dron en *VLOS* también se pueda estar atento a los conflictos. En caso de conflicto con otro dron en prioridad, también sería interesante mostrar esta información.

5.5. Tercera versión

Tras la segunda prueba de validación, tomando como base las sugerencias de los pilotos, se decidió mejorar la interfaz de usuario para conseguir una experiencia satisfactoria en la presentación de la información relativa a la resolución de conflictos. En consecuencia, las prioridades para esta tercera versión fueron:

1. Mejorar el sistema de corrección de conflictos.
2. Incluir efectos sonoros para las alertas.
3. Detallar la posición del dron en conflicto.
4. Mostrar la ruta de vuelo autónomo al cargar un plan de vuelo.

A partir de estas especificaciones, se realizaron los cambios pertinentes en la aplicación. En concreto, estos afectaron al código de la interfaz gráfica del piloto (*CompleteWidgetActivity*) y de la gestión de la configuración del modo de vuelo autónomo (*AFPSelection*) y a sus vistas, *activity_default_widgets.xml* y *activity_afp_selection.xml*, respectivamente. Las modificaciones se muestran en el diagrama de clases de la Figura 5.23.

5.5.1. Interfaz gráfica del piloto

La primera de las dos modificaciones de esta versión es la inclusión de un panel de alertas de conflicto en la interfaz gráfica de vuelo del usuario, el cual se muestra con un ejemplo de conflicto en la Figura 5.24.

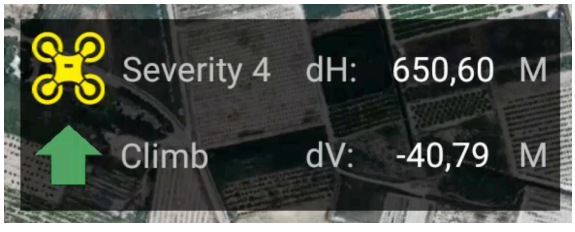


Figura 5.24: Detalle de la información que aparece en el panel de alertas de conflictos.

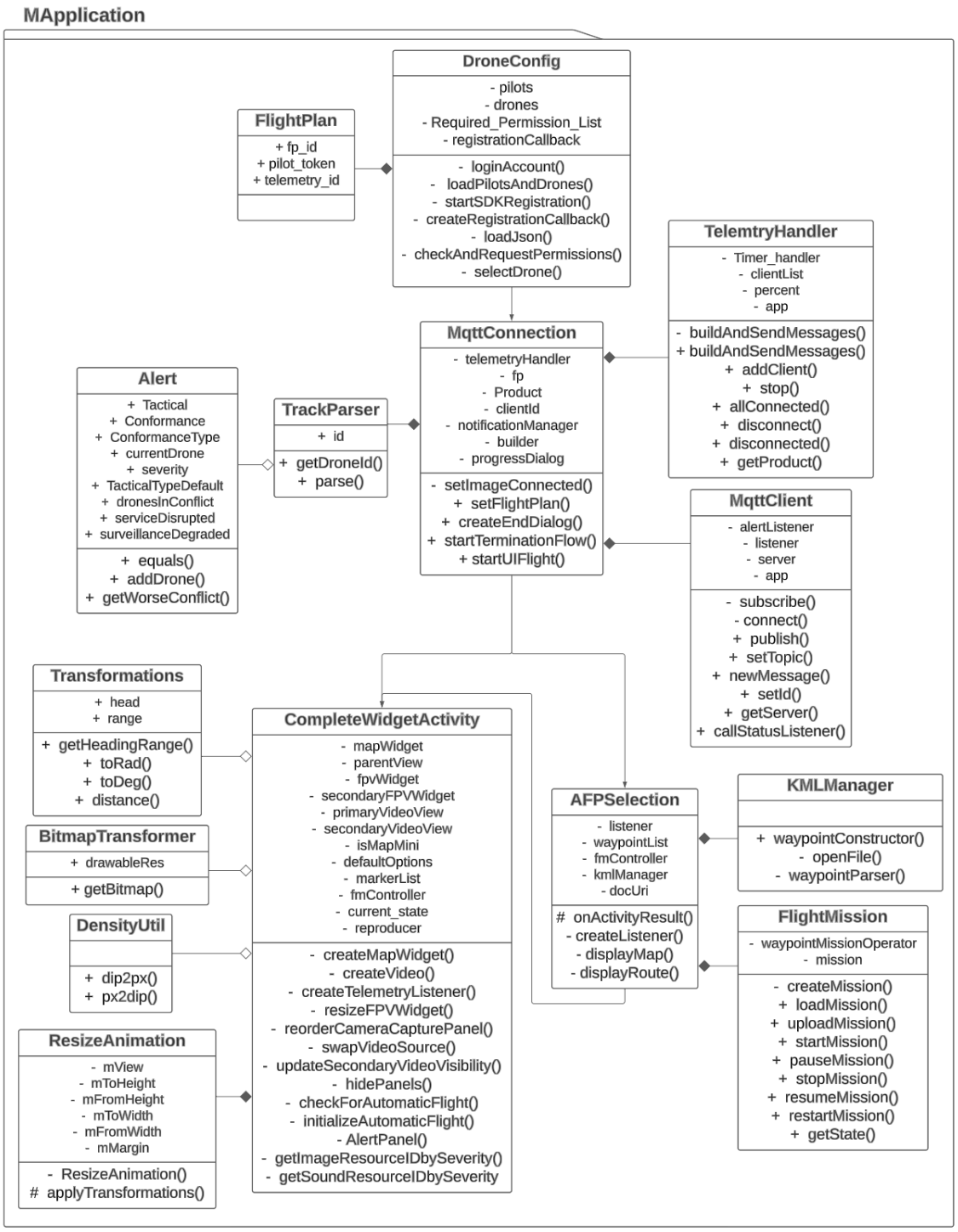


Figura 5.23: Diagrama de clases de la tercera versión.

El panel de conflictos se divide en cuatro elementos:

1. El tipo de alerta, la cual se muestra mediante el icono de un dron que puede ser de hasta cuatro colores distintos, tal y como se ha explicado con anterioridad, junto con un texto que detalla la severidad del conflicto.
2. La distancia horizontal (*dH*) en metros del dron en conflicto con respecto al propio.
3. La sugerencia de una maniobra de elusión de conflicto, que se compone de una flecha (cuya dirección puede ser ascendente o descendente) y un texto que la acompaña para facilitar aún más la comprensión.
4. La distancia vertical (*dV*) del dron en conflicto con respecto al propio, expresada en metros.

Así pues, en la Figura 5.25 se muestra el nuevo aspecto de la vista de interfaz gráfica de usuario. La única diferencia con respecto a la segunda versión (Figura 5.17) es la inclusión de este panel de alertas en la parte izquierda por encima de los botones.

Figura 5.25: Tercera versión de la vista *CompleteWidgetActivity*.

En cuanto al código de la clase *CompleteWidgetActivity*, se han añadido tres nuevas funciones privadas: *AlertPanel*, *getImageResourceIDbySeverity* y *getSoundResourceIDbySeverity*. Las dos últimas son funciones auxiliares que se utilizan para, dada la gravedad del conflicto, seleccionar tanto la imagen del conflicto que se mostrará como el sonido que se reproducirá.

La función *AlertPanel* es más compleja, ya que es la encargada de gestionar el panel de alertas creado en la vista. Para ello, necesita como entrada un objeto de tipo *Alert*, sobre el cual realiza una primera comprobación cuyo fin es determinar si existe algún conflicto. En caso negativo, en el panel de alertas no aparecerá ninguna información, esto es, los campos estarán en blanco.

Si por el contrario existiese algún conflicto con el dron, entonces activaría la visibilidad de los elementos del panel de alertas y mediante un bloque *try-catch* iniciaría el reproductor de sonido nativo con la melodía correspondiente a la situación de conflicto, la cual se recupera de la función *getSoundResourceIDbySeverity*.

Posteriormente, mediante un bucle recorrería los elementos del objeto *Alert*, añadiendo marcadores en el mapa con sus posiciones y sus iconos correspondientes, haciendo uso de la función *getImageResourceIDbySeverity*. Tras realizar esta adición, se comprueba si la severidad del conflicto con dicho dron es menor o igual que la severidad de la alerta, en cuyo caso mediante las funciones de transformación obtiene las distancias vertical y horizontal que lo separan del dron. Esta información es la que se refleja en el panel de alertas como *dH* y *dV*.

Finalmente, tan sólo queda comprobar si la distancia vertical es inferior a cero, para discernir si la sugerencia de maniobra será hacia arriba, por lo que aparecerá una flecha verde apuntando en dicha dirección junto con el texto *Climb*, o hacia abajo y, por tanto, la flecha será de color rojo apuntando hacia el suelo y el mensaje pasará a ser *Descend*.

5.5.2. Configuración de vuelo autónomo

La segunda modificación de esta versión está relacionada con el vuelo autónomo, en concreto, con la presentación del recorrido que realizará el dron de acuerdo al plan de vuelo autónomo seleccionado.

En la Figura 5.26 se muestra un ejemplo de una ruta de vuelo ya cargada para reflejar el nuevo cambio introducido en la aplicación.

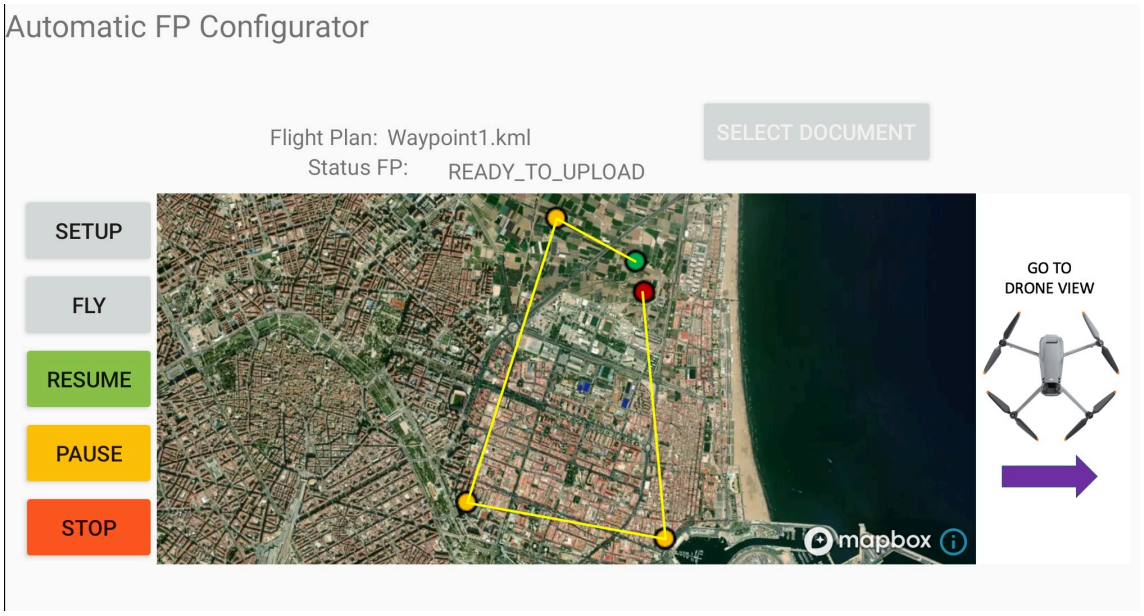


Figura 5.26: Nueva versión de la vista *AFPSelection*.

Para cumplir con este objetivo se realizaron cambios en el código de *AFPSelection*, en concreto, se incluyeron dos funciones adicionales: *displayMap* y *displayRoute*. Como sus propios nombres indican, la primera función sirve para cargar el mapa dentro de la vista, mientras que la segunda se encarga de elaborar e insertar la ruta dentro de este mapa.

Así pues, la función *displayMap* consiste en la adaptación de la parte del código de la clase *CompleteWidgetActivity* que se encarga de lanzar el mapa, para lo cual emplea la función auxiliar *initMapboxMap*.

Por su parte, *displayRoute* se encarga de leer la lista de *waypoints* que contiene el archivo *KML* seleccionado para extraer sus coordenadas. Una vez almacenadas, se añaden al mapa en forma de icono circular, donde el inicio se marca en verde y el final en rojo. También se encarga de unir los distintos puntos mediante líneas negras. Finalmente, emplea las funciones internas del *SDK* para centrar la ruta en el mapa.

5.6. Resultados

Los resultados pretenden definir el estado final de la aplicación tras haberse presentado su evolución a lo largo de toda la sección de desarrollo previamente descrita.

Tras pulir el código e incluir las últimas modificaciones en la tercera versión, se decidió dotar a la aplicación de un aspecto completamente distinto al ya mostrado. Esta decisión surge de la necesidad de embellecer la parte visible de la interfaz, ya que, hasta el momento, la principal preocupación había sido el funcionamiento exitoso del código programado.

En consecuencia, el diseño de la aplicación se había quedado al margen. Para cumplir con este propósito, se han incluido dos nuevas pantallas, que se corresponden a las clases Java: *SplashActivity* y *ChooseFlightMode*.

La primera de ellas es la que se inicia al lanzar la aplicación y cuyo objetivo es mostrar una pantalla de carga al usuario, tal y como se muestra en la Figura 5.28.

Por su parte, la clase *ChooseFlightMode* se ha diseñado con el fin de distinguir la elección del modo de vuelo del usuario. Su diseño se presenta en la Figura 5.31 y consiste en dos botones para seleccionar el modo, lo cual se controla con la variable *isAutomatic*, como ya se ha comentado en el desarrollo.

En otra instancia, los cambios realizados en el código de la aplicación han ido siempre dirigidos a adaptar las características ya existentes al nuevo diseño creado. Donde más problemas se han detectado al intentar realizar el nuevo diseño ha sido en la clase *DroneConfig*, puesto que contenía el mayor número de elementos a modificar.

La clase que no ha recibido ningún cambio con respecto al diseño ha sido *CompleteWidgetActivity*, puesto que no se consideró necesario mejorar la interfaz del piloto.

Debido a los cambios acontecidos respecto a la tercera versión, se expone en la Figura 5.27 el diagrama de clases de la última versión desarrollada junto con las distintas pantallas que componen la aplicación final, desde la Figura 5.28 hasta la Figura 5.33. A continuación, se resumen las funciones de cada una de las pantallas:

1. Pantalla de carga: al iniciarse la aplicación, esta será la primera pantalla que verá el usuario, donde tras una breve animación, será redirigido a la siguiente pantalla.

2. Pantalla de configuración del dron: es la siguiente pantalla a la de carga. Aquí el usuario puede seleccionar la configuración de vuelo.

3. Pantalla de conexión con *U-TraC*: una vez seleccionado el dron, el usuario puede avanzar a esta pantalla, en la cual se configuran las conexiones con el servidor de telemetría y las alertas por pantalla.

4. Pantalla de selección de modo: se le ofrece al usuario una dicotomía para seleccionar el tipo de vuelo con el que desea proceder: automático o manual.

5. Pantalla de vuelo autónomo: es el configurador del plan de vuelo automático. Se permite seleccionar y cargar el documento deseado para ejecución, la visualización de la ruta contenida en el archivo y el control del vuelo.

6. Pantalla de interfaz gráfica del piloto: esta es la más importante de la aplicación, pues es donde el piloto tiene el control sobre el vuelo del dron. Se puede acceder tanto en modo manual como en automático.

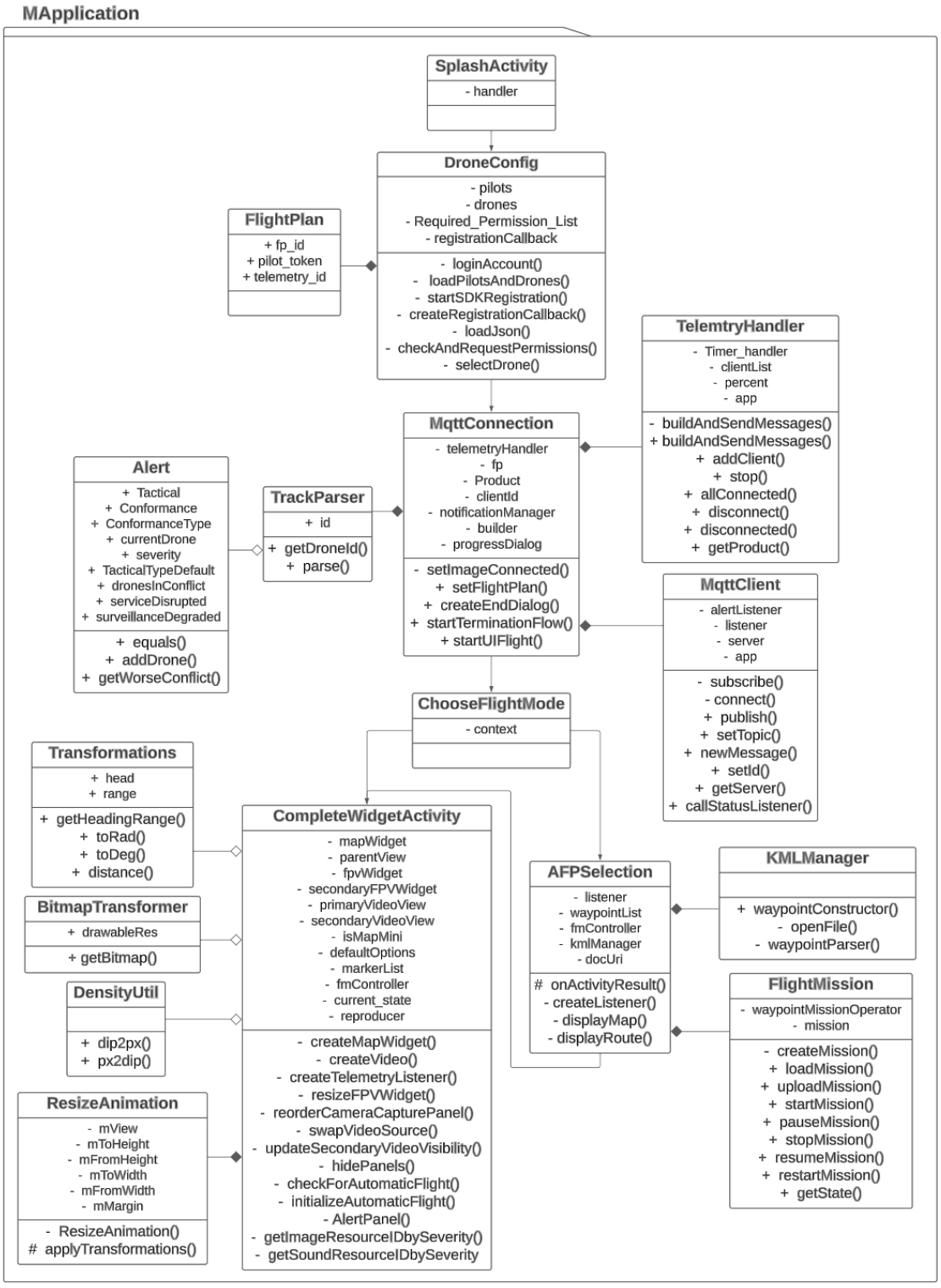


Figura 5.27: Diagrama de clases de la versión final de la aplicación.

Figura 5.28: Pantalla de carga de la aplicación.

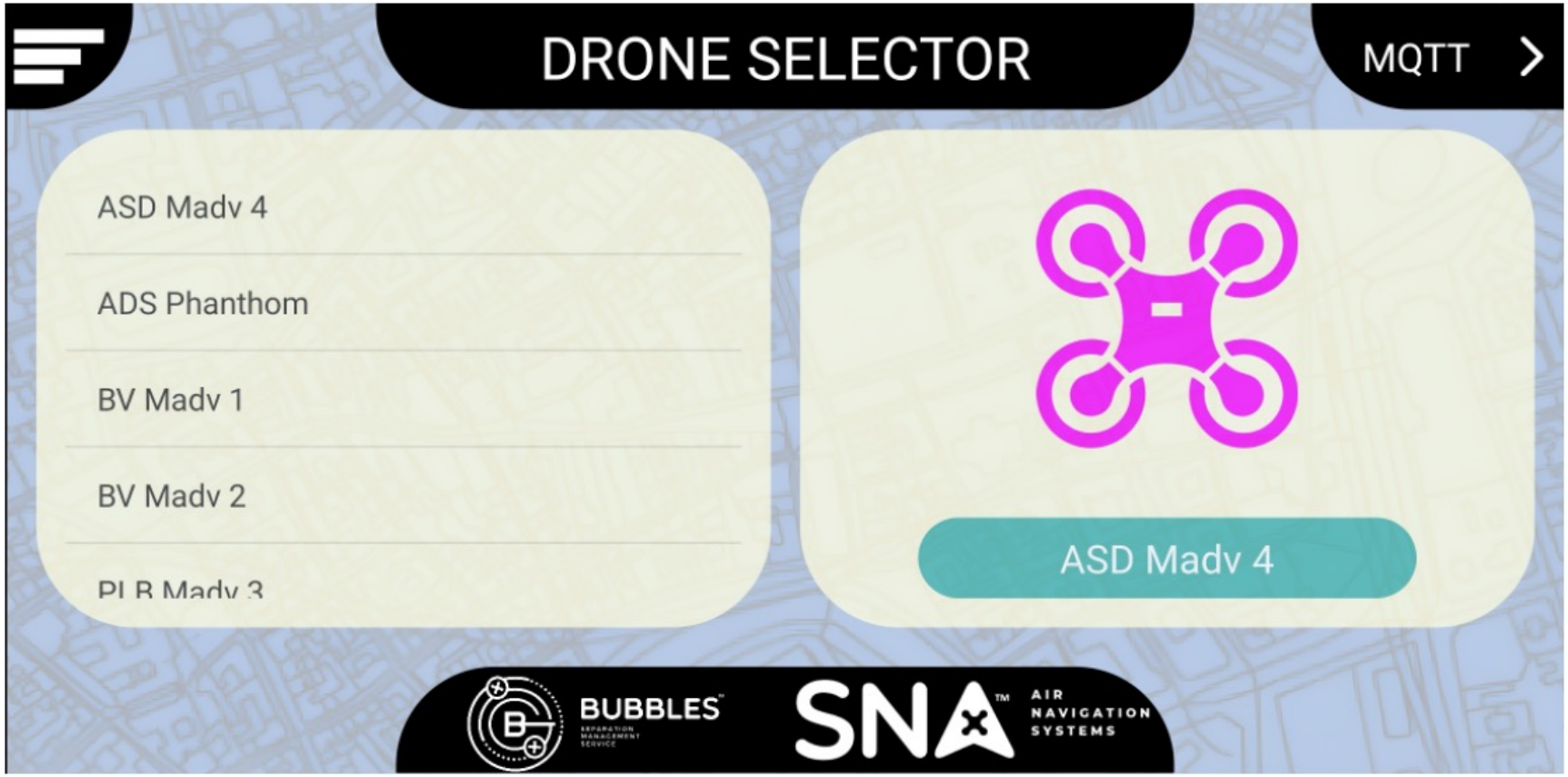


Figura 5.29: Pantalla de selección de dron.

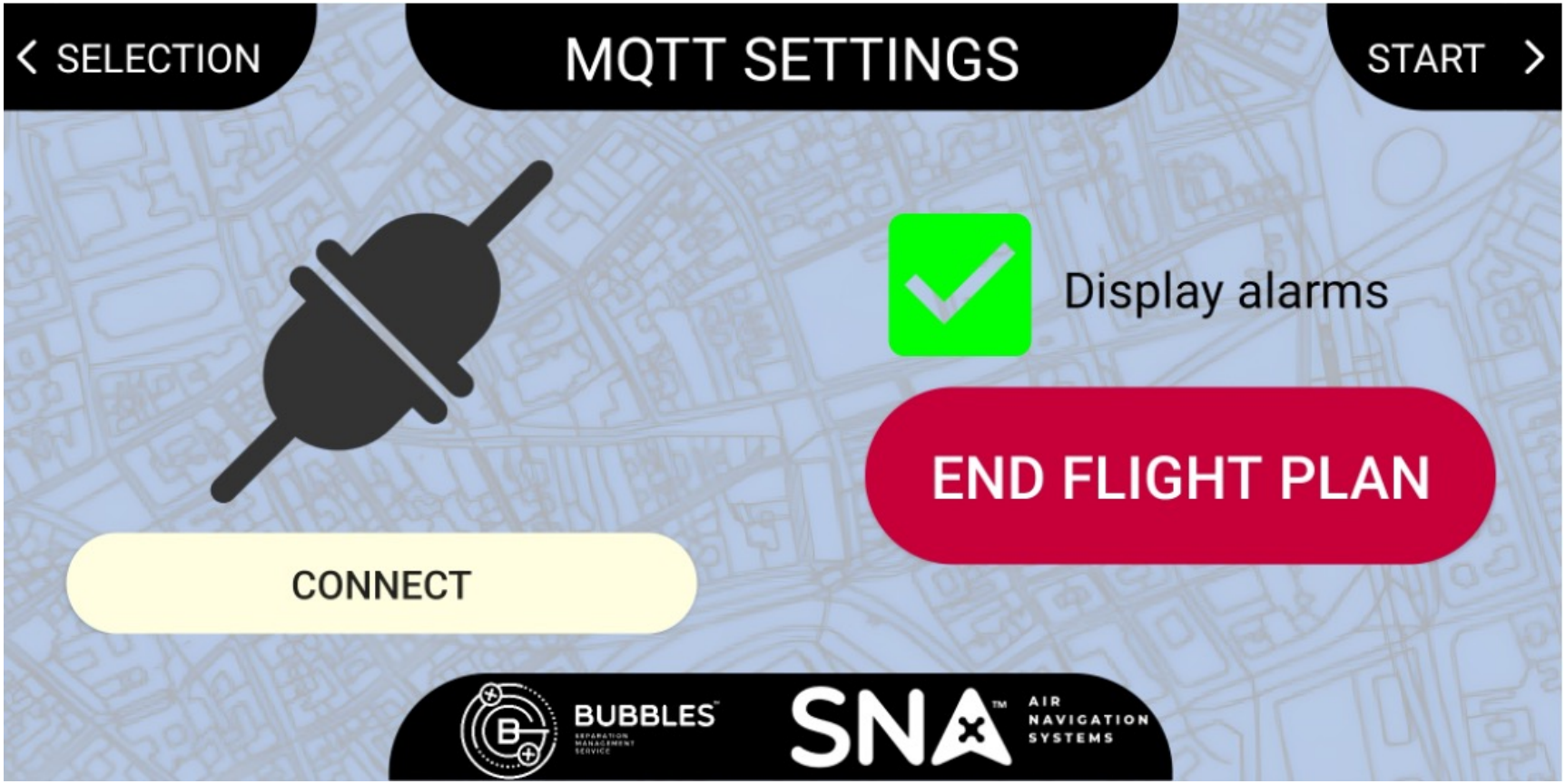


Figura 5.30: Pantalla de configuración de la conexión con la plataforma *U-TraC*.

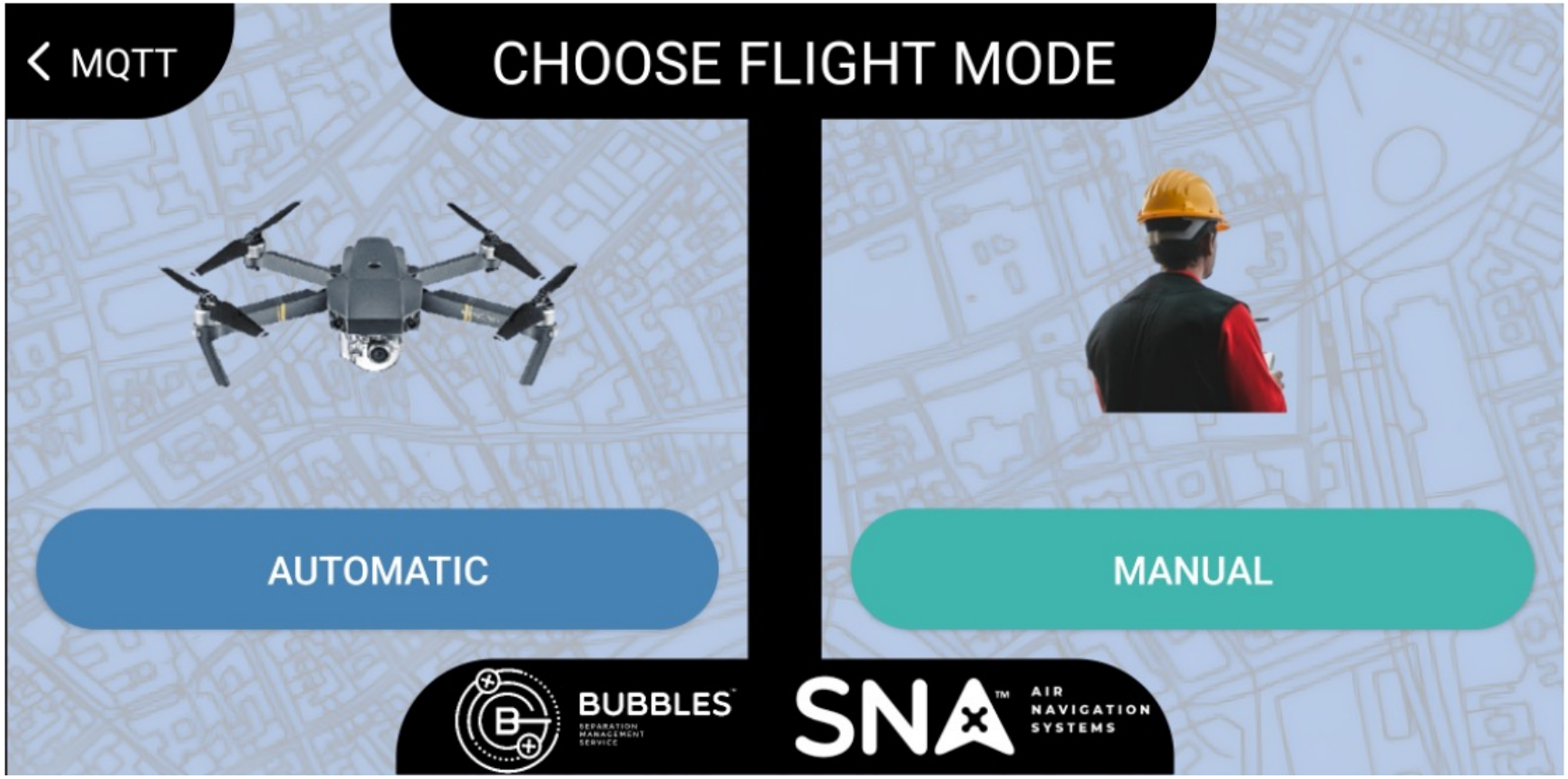


Figura 5.31: Pantalla de selección de modo de vuelo del dron.

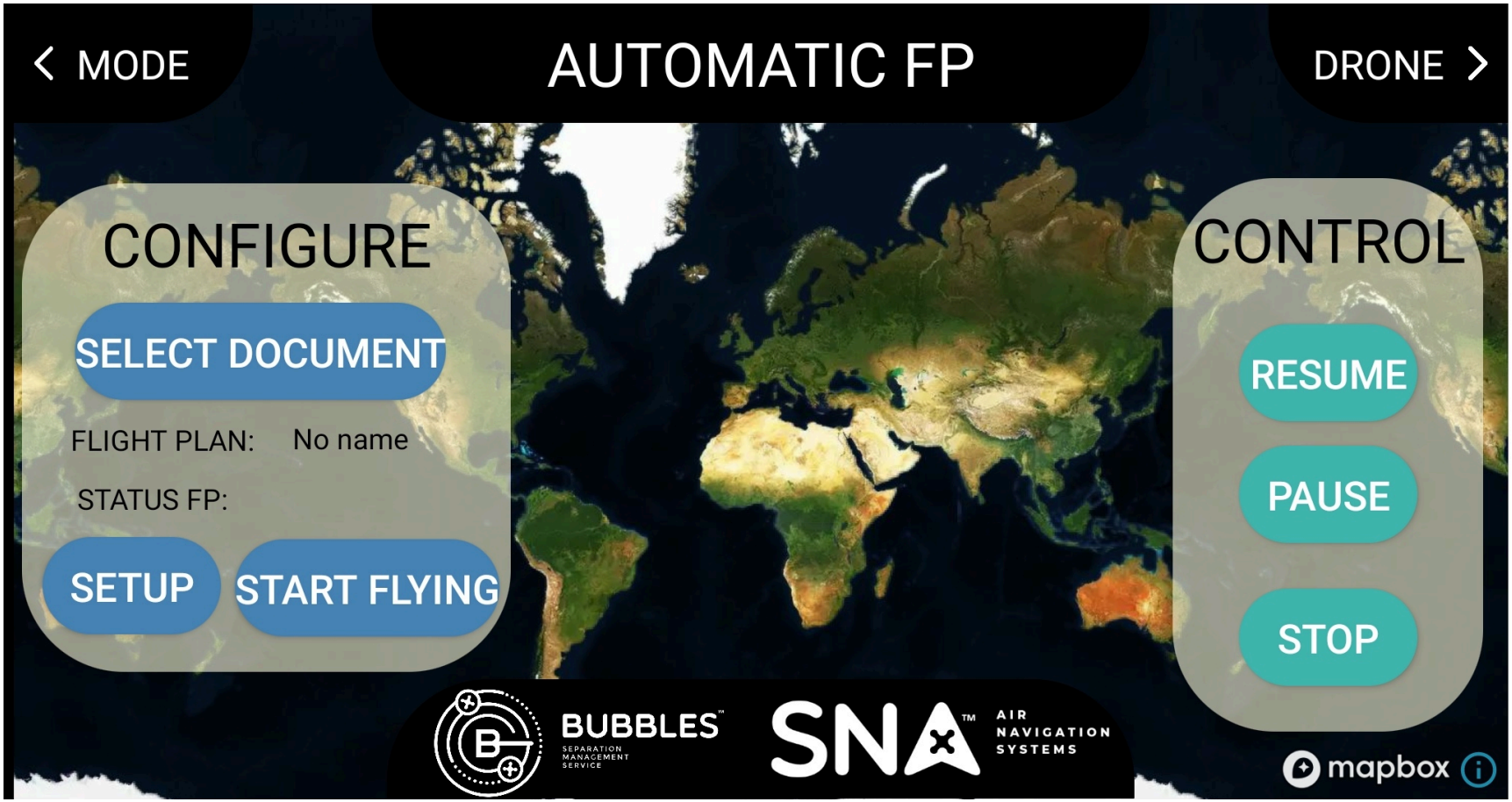


Figura 5.32: Pantalla de configuración de vuelo autónomo.

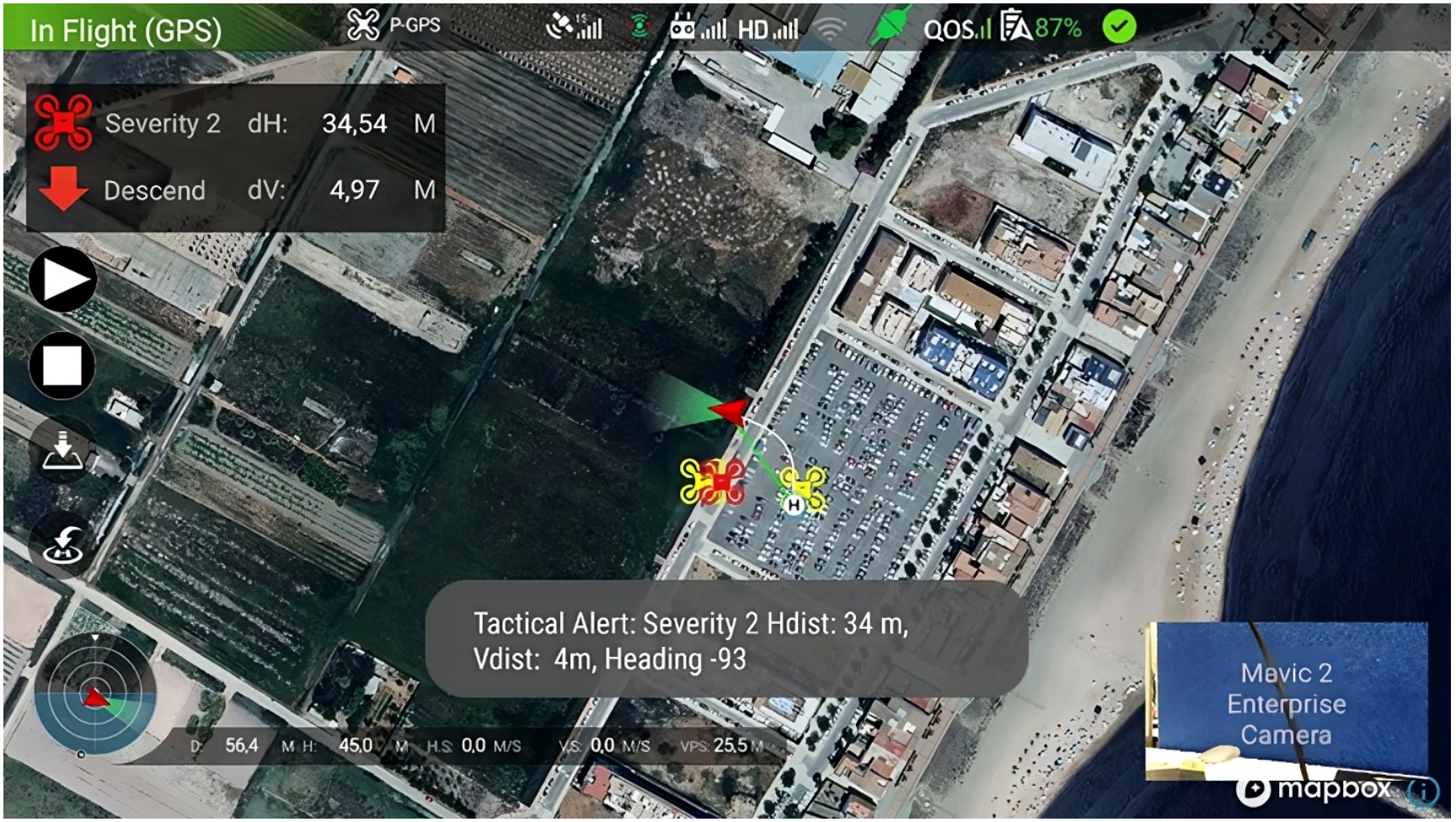


Figura 5.33: Pantalla de la interfaz gráfica del piloto.

Capítulo 6

CONCLUSIONES Y LÍNEAS FUTURAS

6.1. Conclusiones

Una vez expuesta la arquitectura del proyecto, donde se han definido las partes que componen la aplicación y, tras realizar una descripción detallada del proceso de evolución del desarrollo de la interfaz, en esta sección final se establecerán las conclusiones del TFM.

En primer lugar, cabe mencionar que se ha cumplido con éxito el objetivo principal que se planteó al iniciar este proyecto: proporcionar a los usuarios de vuelo de los *UAS* una interfaz de vuelo práctica para operar en el entorno de *U-space*, controlando el tráfico cercano y las alertas de conflicto en el marco del proyecto *BUBBLES* y mediante el soporte de la plataforma de gestión de telemetría *U-TraC*.

Para ello, se partió de unas especificaciones técnicas dispuestas por el equipo de investigación *SNA* para crear una primera versión de la aplicación, estableciendo el punto de partida del diseño. A partir de esta, las siguientes versiones de la interfaz, aportaron mejoras considerables tanto al funcionamiento como a la estética, la cual siempre se ha desarrollado teniendo en cuenta los Factores Humanos.

Respecto al sistema creado, resalta la interfaz gráfica del piloto, donde este dispone de todas las herramientas provistas por el *SDK* de la empresa *DJI* para manejar el dron y asegurar la zona de vuelo. Adicionalmente, se ha decidido incluir todavía más funcionalidades como el panel de alertas, que indica el conflicto más crítico junto a una sugerencia de maniobra para poder impedir el choque, y los botones relativos al control del vuelo autónomo.

Precisamente, una de las características más destacadas de la aplicación sería la dualidad del modo de vuelo que permite al usuario elegir la operativa que desee. Específicamente, el modo autónomo ha sido la creación más importante del proyecto, la cual permite realizar rutas de vuelo previamente diseñadas y controlar su ejecución en vivo.

Además, se trata de la parte más compleja de todo el diseño desarrollado, ya que se encontraron muchos obstáculos en el camino. Pese a ello, el resultado ha sido muy satisfactorio.

Todo este sistema se sostiene y es útil gracias tanto al *software* de la empresa *DJI* como a la plataforma *U-TraC*. El primero se encarga de incorporar y permitir la manipulación de las funcionalidades del dron en tiempo real, mientras que el segundo tiene como función realizar, mantener y supervisar el flujo de la telemetría a lo largo de las operaciones. En consecuencia, requiere de un protocolo de comunicaciones ligero y ágil, por lo que se ha diseñado en base al protocolo *MQTT*. Así pues, la plataforma *U-TraC* es el gran soporte que sustenta el control y registro de las operaciones de vuelo.

Por supuesto, todo esto no hubiese sido posible sin la colaboración de los operadores de vuelo que realizaron pruebas exhaustivas. Fueron los responsables de comprobar el funcionamiento de las herramientas creadas y proveer sugerencias de posibles mejoras para la interfaz. Igualmente, las simulaciones realizadas en el laboratorio fueron claves para detectar y corregir errores con anterioridad a dichas pruebas, dotándolas de más rigor y sentido.

Finalmente, el resultado de este trabajo es el desarrollo de una aplicación *Android* funcional, ágil, intuitiva y práctica para el manejo de los drones en entornos *U-space*. Dicha interfaz consta de un total de seis pantallas por las que el usuario puede navegar con mucha fluidez y facilidad para primeramente, configurar las características de la sesión de vuelo y, posteriormente, vigilar y manejar el producto en tiempo real.

6.2. Líneas futuras

Cualquier trabajo desarrollado en un ámbito de investigación contribuye al estado del arte del campo en el que se enmarque y genera nuevas líneas de vías de trabajo. A continuación, se exponen algunas de los posibles caminos que se podrían seguir tomando este proyecto como raíz.

La primera senda para recorrer sería adaptar la aplicación creada en dispositivos de otras marcas de *UAS* que ofrezcan, al igual que *DJI*, la posibilidad de realizar operaciones con sus productos a través de sus respectivos *SDK*. Un ejemplo claro sería la compañía *Parrot*, la cual permite acceder a su kit de desarrollo de *software* tanto para el sistema operativo *Android* como para el sistema operativo *iOS*.

Enlazando con lo anterior, otra propuesta de trabajo sería conseguir implementar la aplicación en un sistema operativo diferente, como *iOS*, cubriendo así prácticamente todo el mercado de dispositivos finales. En concreto, se podría usar el *SDK* de *Parrot* como soporte y las ideas de este TFM como punto de partida para crear una interfaz similar en este sistema.

Si bien las dos primeras opciones consisten en usar el trabajo como punto de partida, en esta tercera opción el planteamiento sería establecer mejoras en el código ya creado. La más significativa que se podría implementar sería el desarrollo de una visualización 3D del mapa en la pantalla de la interfaz gráfica, donde el piloto dispone de la visión del dron. Esto permitiría a los operadores de vuelo conocer, además de latitud y longitud, la elevación de todos los drones de su alrededor, ofreciendo un salto cualitativo en la resolución de conflictos.

Bibliografía

[1] Grand View Research. *Consumer Drone Market Size & Share Report 2022-2030*. URL: https://www.grandviewresearch.com/industry-analysis/consumer-drone-market (visitado 17-02-2023).

[2] The European Commission. *A Drone Strategy* 2,0 *for a Smart and Sustainable Unmanned Aircraft Eco-System in Europe*. The European Commission, 2022. URL: https://transport.ec.europa.eu/system/files/2022-11/COM%5C_2022%5C_652%5C_drone%5C_strategy%5C_2.0.pdf.

[3] *About Us - DJI*. URL: https://www.dji.com/es/company?site=brandsite&from=footer (visitado 18-02-2023).

[4] *Delivering the Digital European Sky*. URL: https://www.sesarju.eu/ (visitado 17-02-2023).

[5] *Single european sky: New traffic light system assesses environmental performance of Air Navigation Services*. Sep. de 2022. URL: https://transport.ec.europa.eu/news/single-european-sky-new-traffic-light-system-assesses-environmental-performance-air-navigation-2022-09-19_en (visitado 17-02-2023).

[6] Empresa Común para la Investigación sobre ATM en el Cielo Único Europeo 3. *Plan maestro ATM europeo : la hoja de ruta para lograr una aviación de altas prestaciones en Europa : resumen ejecutivo : edición de 2015*. Oficina de Publicaciones, 2016. DOI: doi/10.2829/280650.

[7] *U-space*. URL: https://www.sesarju.eu/U-space (visitado 17-02-2023).

[8] *Sesar Joint Undertaking: Bubbles - defining the building basic blocks for a U-space separation management service*. URL: https://www.sesarju.eu/projects/bubbles (visitado 17-02-2023).

[9] Cecilia Claramunt-Puchol, Norberto Vera-Vélez y Juan Vicente Balbastre-Tejedor. *Concept of Separation Management for UAS in the U-space*. Ver. 04.00.00. Oct. de 2022. DOI: 10.5281/zenodo.7220800. URL: https://doi.org/10.5281/zenodo.7220800.

[10] *Global Air Traffic Management Operational Concept (DOC 9854)*. URL: https://store.icao.int/en/global-air-traffic-management-operational-concept-doc-9854 (visitado 24-02-2023).

[11] Rosa Fernández. *Sistemas operativos (so) de smartphones: Cuota de Mercado Mundial Por Trimestre*. Nov. de 2022. URL: https://es.statista.com/estadisticas/635543/cuota-de-mercado-mundial-de-sistemas-operativos-de-smartphones-desde-2009-por-trimestre/#:~:text=Android%5C%20mantuvo%5C%20su%5C%20posici%5C%C3%5C%B3n%5C%20como,una%5C%20cuarta%5C%20parte%5C%20del%5C%20mercado. (visitado 18-02-2023).

[12] *Tutorial KML | Keyhole markup language | google developers*. URL: https://developers.google.com/kml/documentation/kml_tut?hl=es-419 (visitado 04-03-2023).

[13] *The standard for IOT messaging*. URL: https://mqtt.org/ (visitado 19-02-2023).

[14] *Commission Implementing Regulation (EU) 2019/947 of 24 May 2019 on the rules and procedures for the operation of unmanned aircraft, 2019 OJ L 152*.

[15] *Operations with UAS/Drons*. URL: https://www.seguridadaerea.gob.es/en/ambitos/drones/operaciones-con-uas-drones (visitado 17-03-2023).

[16] Andrew Weinert et al. "Near Midair Collision Analog for Drones Based on Unmitigated Collision Risk". En: *Journal of Air Transportation* 30.2 (2022), págs. 37-48. DOI: 10.2514/1.D0260. eprint: https://doi.org/10.2514/1.D0260. URL: https://doi.org/10.2514/1.D0260.

[17] *Documentation Introduction: Hardware Products*. URL: https://developer.dji.com/document/52b7f3b9-a773-47b3-b9e5-71c7a3190736 (visitado 04-03-2023).

[18] *Documentation Introduction: SDK Architectural Overview*. URL: https://developer.dji.com/document/45a1decb-e86c-4cfc-bafd-95420bf2acb8 (visitado 04-03-2023).

[19] Andrew Banks y Rahul Gupta. *MQTT 3.1.1 specification*. URL: http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/mqtt-v3.1.1.html (visitado 17-03-2023).

[20] *Mobile SDK: WaypointMission*. URL: https://developer.dji.com/api-reference/android-api/Components/Missions/DJIWaypointMission.html (visitado 17-03-2023).

[21] Juan Vicente Balbastre-Tejedor et al. *Concept Validation Test Plan (CVALP)*. Ver. 01.00.00. Mar. de 2022. DOI: 10.5281/zenodo.6922553. URL: https://doi.org/10.5281/zenodo.6922553.

[22] Cecilia Claramunt Puchol y Diego Herce de Esteban. *BUBBLES Validation Report*. Ver. 02.00.00. Oct. de 2022. DOI: 10.5281/zenodo.7687699. URL: https://doi.org/10.5281/zenodo.7687699.

[23] United Nations. *Home - United Nations Sustainable Development*. URL: https://www.un.org/sustainabledevelopment/ (visitado 20-03-2023).

[24] UN DESA. *The Sustainable Development Goals Report 2022*. 2022. URL: https://unstats.un.org/sdgs/report/2022/ (visitado 20-03-2023).

Parte II

Anexos

Apéndice A

Relación del trabajo con los Objetivos de Desarrollo Sostenible de la Agenda 2030

Los Objetivos de Desarrollo Sostenible (ODS) [23] son un llamamiento a la acción de todos los países para promover la prosperidad al tiempo que se protege el planeta. Reconocen que la erradicación de la pobreza debe ir de la mano de estrategias que fomenten el crecimiento económico y aborden una serie de necesidades sociales como la educación, la sanidad, la protección social y las oportunidades de empleo, a la vez que se hace frente al cambio climático y la protección del medio ambiente.

Número	Objetivo	Grado de relación
1	Fin de la pobreza	No procede
2	Hambre cero	No procede
3	Salud y bienestar	No procede
4	Educación de calidad	No procede
5	Igualdad de género	No procede
6	Agua limpia y saneamiento	No procede
7	Energía asequible y no contaminante	No procede
8	Trabajo decente y crecimiento económico	No procede
9	Industria, Innovación e Infraestructura	Alto
10	Reducción de las desigualdades	No procede
11	Ciudades y comunidades sostenibles	Alto
12	Producción y consumo responsables	No procede
13	Acción por el clima	No procede
14	Vida submarina	No procede
15	Vida de ecosistemas terrestres	No procede
16	Paz, justicia e instituciones sólidas	No procede
17	Alianzas para lograr los objetivos	No procede

Tabla A.1: Grado de relación de los objetivos de desarrollo sostenible con el TFM.

La Agenda 2030 es deliberadamente ambiciosa y transformadora, con un conjunto de 17 Objetivos de Desarrollo Sostenible integrados e indivisibles y metas para guiar el camino. En la Tabla A.1 se presentan los objetivos junto con su relación con este trabajo. De entre todos los objetivos descritos, este trabajo contribuye a la implementación adecuada de los objetivos 9 y 11. A continuación, se realizará una breve descripción de cada uno de ellos, así como el impacto que este TFM tiene sobre los mismos.

A.1. Objetivo 9: Industria, Innovación e Infraestructura

El objetivo 9 de este plan es: *construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación* [23].

Invertir en infraestructuras, promover la industrialización y apoyar el desarrollo tecnológico, la investigación y la innovación son tres fuerzas motrices del desarrollo económico sostenible. Estos motores pueden ayudar a los países a reducir la pobreza creando oportunidades de empleo estimulando el crecimiento y fomentando la construcción y mejora de instalaciones físicas esenciales para el funcionamiento de las empresas y la sociedad.

De hecho, la pandemia COVID-19 ha demostrado la importancia de la industrialización, la innovación tecnológica y las infraestructuras resilientes para alcanzar los ODS. Las economías con un sector industrial diversificado e infraestructuras sólidas (por ejemplo, transporte, conectividad a Internet y servicios públicos) sufrieron menos daños y están experimentando una recuperación más rápida [24].

En concreto, este proyecto se enmarca en la parte de la innovación tecnológica y la investigación. Pretende crear un entorno digital, dirigido a los operadores de vuelo de drones, con el propósito de realizar misiones en *U-space* sin generar colisiones entre los equipos que estén volando en la zona de operación.

A.2. Objetivo 11: Ciudades y Comunidades Sostenibles

Por su parte, el objetivo 11 consiste en: *lograr que las ciudades y los asentamientos humanos sean inclusivos, seguros, resilientes y sostenibles* [23].

En la actualidad, más de la mitad de la población mundial vive en ciudades [24]. En 2050 se calcula que 7 de cada 10 personas vivirán en zonas urbanas. Las ciudades son motores del crecimiento económico y aportan más del 80 % del producto interior bruto mundial. Sin embargo, también son responsables de más del 70 % de las emisiones mundiales de gases de efecto invernadero. Con ello, si se planifica y gestiona de manera adecuada, el desarrollo urbano puede ser sostenible y generar prosperidad inclusiva.

Por el contrario, una urbanización rápida y mal planificada conlleva numerosos retos, como la escasez de viviendas asequibles, infraestructuras insuficientes (como transporte público y servicios básicos), espacios abiertos limitados, altos niveles de contaminación atmosférica y un mayor riesgo climático y de catástrofes.

Las profundas desigualdades expuestas por la pandemia COVID-19 y otras crisis que se han sucedido en cascada ponen aún más de relieve la importancia del desarrollo urbano sostenible.

Precisamente, ese es el objetivo de iniciativas como *U-space*, el marco donde se engloba la creación de esta interfaz. La aplicación desarrollada pretende crear un entorno más seguro en los núcleos urbanos donde existe una mayor densidad de población. Habilitar el uso del espacio aéreo de baja cuota, así como ofrecer una herramienta de gestión de separación que sea eficiente ayudará a conseguir este ODS.