UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf

Escola Tècnica Superior d'Enginyeria Informàtica

Escuela Técnica Superior de Ingeniería Informática
Universidad Politécnica de Valencia

Vuelo seguro de enjambres de drones mediante integración de datos topográficos

TRABAJO FIN DE GRADO

Grado en Ingeniería Informática

Autor: Christian Morales Guerrero

Tutor: Carlos Tavares Calafate

Cotutor: Jamie Wubben

Curso 2020-2021

Resumen

Las actuales herramientas de planificación de vuelos para drones no tienen en cuenta los datos de relieve del terreno, lo cual puede derivar en colisiones o en sobrepasar la altura máxima legal del vuelo.

En este contexto, este proyecto tiene como objetivo la integración de datos topográficos en una herramienta de simulación de vuelos de drones. Se trata en concreto del simulador/controlador ArduSim, el cual permite gestionar las comunicaciones entre drones y la creación de enjambres, aunque carece de datos topográficos reales, ofreciendo solo información de mapa que importa de la plataforma Bing.

Se realiza la implementación de un protocolo de vuelo integrado en el simulador, el cual, utilizando datos topográficos de un Modelo Digital de Elevación (MDE), obtiene, gracias a la geolocalización del dron, su altura respecto al suelo en tiempo real, teniendo en cuenta tanto elevaciones del terreno como edificios y vegetación. Además, la implementación es suficientemente flexible como para adaptarse a MDEs con diferentes resoluciones.

En base al trabajo realizado, la herramienta ArduSim es ahora capaz de funcionar tanto en vuelos simulados como en vuelos reales, incluyendo vuelos en enjambre, siendo capaz de detectar obstáculos con anticipación y actuar en consecuencia, dotando al dron de conciencia situacional de su entorno.

Palabras clave: Dron, Topografía, Simulador, MDE, Vuelo, Autónomo, Enjambre

Abstract

Current drone flight planning tools do not take terrain relief data into account, which can lead to collisions or exceeding the legal maximum flight height.

In this context, this project aims to integrate topographic data into a drone flight simulation tool. It is specifically the ArduSim simulator / controller, which allows managing communications between drones and the creation of swarms, although it lacks real topographic data, offering only map information that is imported from the Bing platform.

The implementation of a flight protocol integrated into the simulator is carried out, which, using topographic data from a Digital Elevation Model (DEM), obtains, thanks to the geolocation of the drone, its height with respect to the ground in real time, taking into account counts both terrain elevations and buildings and vegetation. Furthermore, the implementation is flexible enough to accommodate DEMs with different resolutions.

Based on the work done, the ArduSim tool is now capable of working both in simulated flights and in real flights, including swarm flights, being able to detect obstacles in advance and act accordingly, providing the drone with situational awareness of its surroundings.

Key words: Drone, Topography, Simulator, DEM, Flight, Autonomous, Swarm

Índice general

Índice de figuras

Índice de algoritmos

CAPÍTULO 1
Introducción

1.1 Motivación

Hoy en día vivimos en una sociedad cada vez más dependiente de la movilidad invertida, es decir, no deseamos desplazarnos hasta algo que queremos, sino que venga hasta nosotros. Estamos cada vez más acostumbrados al repartidor de Amazon que nos deja el paquete en la puerta, o a los *raiders* que cada vez más vemos por nuestras calles llevando la comida hasta cada uno de nuestros hogares.Por otro lado, el ser humano tiene limitaciones de movilidad y acceso contra las que tiene que enfrentarse cada día, véase un bombero intentando acceder a un edificio en llamas o quizá a punto de derruirse para buscar posibles supervivientes. En estos casos realmente se ponen en juego vidas por necesidad, por falta de una alternativa más segura.

Pues bien, esto es solo una pequeña muestra de las problemáticas hoy en día ya resueltas por el uso de drones [1]. Con un crecimiento de aplicaciones exponencial, un dron es capaz de realizar una gran variedad de tareas tales como [2]:

- Entregar paquetes y comida a domicilio.
- Prevenir y mitigar incendios.
- Buscar supervivientes en zonas catastróficas.
- Vigilar fronteras.
- Monitorizar y actuar sobre grandes cultivos.
- Realizar estudios arqueológicos.
- Acceder al interior de volcanes.
- Manipular materiales radioactivos.

Con esta versatilidad, el dron se convierte en la herramienta perfecta para reducir costes, riesgos, material y tiempos en las tareas que desempeñan. La posibilidad de realizar todas las tareas de forma autónoma y/o grupal provoca que la necesidad de disponer de simuladores y controladores de vuelo de drones resulte primordial. Con el conjunto hardware/software adecuado, las capacidades de un dron quedan sólo limitadas por nuestra imaginación.

Sin embargo, los avances tecnológicos requieren de una legislación en la que sostenerse, y aunque existente, es todavía bastante estricta, añadiendo burocracia y requisitos

para cada vuelo que se desee efectuar, imposibilitando casi las operaciones en ciudad y exigiendo un nivel de precisión en el vuelo que refuerza la necesidad de un controlador fiable. Entre otras cosas, la normativa actual[3] exige no superar la altura de 120 metros en vuelo (salvo autorización explícita), lo cual puede resultar complejo en terrenos irregulares.

Esta situación supone desafíos adicionales cuando se trata de enjambres de drones, ya que no sólo hay que multiplicar las complicaciones por cada dron que se desee volar sino que también hay que controlar las distancias entre los propios drones y sus trayectorias.

1.2 Objetivos

El principal objetivo de este trabajo es integrar datos topográficos en un simulador de vuelo de drones ya existente, pero carente de una topografía real, y desarrollar un plan de vuelo coherente y seguro en base a dichos datos. Esto consiste en dotar los drones de una conciencia situacional, utilizando los datos de un modelo digital de elevación (MDE) de precisión variable, y siendo capaces de tomar decisiones en base a su entorno para asegurar el cumplimiento de la altura de vuelo definida para la misión, así como de la legislación nacional y europea al respecto, la cual exige no sobrepasar los 400 pies de altura respecto al suelo (120 metros) en ningún momento.

El proyecto puede dividirse en los siguientes hitos:

1. Investigar y elegir el origen de los datos de elevación.
2. Analizar el simulador existente para determinar la estrategia de integración más óptima.
3. Elaborar un plan de desarrollo coherente.
4. Estudiar las estrategias disponibles para controlar el movimiento tridimensional de los drones.
5. Integrar la base de datos topográficos en la herramienta de simulación.
6. Obtener datos de relieve y altura de vuelo en tiempo real.
7. Detectar obstáculos y actuar sobre los parámetros de vuelo para lograr un vuelo seguro.

1.3 Estructura

Este proyecto consta de 7 capítulos contando el actual, a través de los cuales se desarrolla el proceso de incorporación de los datos topográficos a un simulador de vuelo existente.

En el Capítulo 2 (Estado del arte) se parte del conjunto de estudios anteriores que hayan tratado los mismos temas o materias afines al área de investigación, y se presenta una síntesis crítica de las ideas principales.

En el Capítulo 3 (Tecnologías utilizadas) se definen en detalle las dos tecnologías principales del proyecto, siendo estas los modelos digitales de elevación (MDE) y el simulador/controlador de vuelo ArduSim [4]. Se cubre además en este apartado el proceso de investigación seguido para alcanzar los objetivos 1 y 2.

En el Capítulo 4 (Análisis del problema) se procede a indagar en el objetivo principal y cómo conseguirlo, y se ofrece un análisis exhaustivo de las distintas aproximaciones posibles, con una elección final.

En el Capítulo 5 (Propuesta de solución) se especifica el proceso de desarrollo de la solución elegida, desde su diseño hasta la implantación. Se ofrecen distintos diagramas que facilitan su comprensión y funcionamiento.

En el Capítulo 6 (Validación y resultados) se analizan los resultados obtenidos de la propuesta y su grado de consecución del objetivo principal. Se ofrecen datos estadísticos a fin de probar su validación.

Finalmente se incluye el Capítulo 7 (Conclusiones) donde como dice su nombre se concluye el proyecto con una perspectiva hacia trabajos futuros.

Una vez finalizados los capítulos se encuentra la bibliografía y posteriormente los apéndices A (Configuración del sistema) y B (Código fuente).

CAPÍTULO 2
Estado del arte

En este ámbito de investigación nos encontramos con dos puntos principales, los drones y la topografía. Estas áreas de conocimiento han establecido una estrecha relación, ya que hay numerosas aplicaciones en las que el uso de drones ha permitido perfeccionar la topografía y sus precisos modelos de elevación, tal y como puede apreciarse en el artículo «High resolution DEM generation using small drone for interferometry SAR» [5].

En cuanto a las investigaciones dedicadas a invertir esta relación, es decir, utilizar la topografía para ayudar al vuelo del dron, estas son todavía escasas. La principal utilidad de la topografía en un vuelo autónomo consiste en trazar rutas y evitar colisiones, y por ahora su principal ámbito de aplicación consiste en vuelos preprogramados con alturas fijas y constantes, sin obstáculos, siendo uno de sus usos más conocidos la agricultura de precisión, tal y como puede apreciarse en el artículo «Review on Application of Drone Systems in Precision Agriculture» [6].

Sin embargo, los vuelos autónomos con toma de decisiones y elección de rutas no están lejos de ser una realidad, y ya podemos encontrar artículos como «Last mile delivery by drones: an estimation of viable market potential and access to citizens across European cities» [7] que hablan de su potencial y viabilidad como medio de reparto, así como otros incluyendo «Drone transportation of blood products» [8], que defienden su utilidad como transporte de sangre en situaciones de emergencia dada su velocidad y eficacia.

Los drones como medio de transporte están cerca de ser una realidad, y la topografía es el elemento clave para conseguirlo. En la actualidad existen algunas investigaciones que combinan los drones y los modelos digitales de elevación, los cuales se proceden a detallar a continuación.

Una de las primeras investigaciones que incorporan la topografía a la evasión de colisiones de forma autónoma fue «Automatic Ground Collision Avoidance System design, integration, flight test» [9]. Esta investigación progresó de forma constante entre 2011 y 2013, partiendo su marco teórico del IEEE [9], y siendo validada por las fuerzas aéreas en un F-16 [10][11]. Sin embargo, esta tecnología se limitaba a intervenir en la dirección del vuelo ante una colisión inminente. Un estudio reciente parece haber retomado esta tecnología [12].

La investigación que justifica las bases teóricas del proyecto es «Unmanned aerial vehicle collision avoidance using digital elevation model» [13], publicado en 2013. Este artículo estudia y establece mediante formulación matemática la viabilidad del uso de modelos de elevación como fuente de datos para el vuelo autónomo de drones.

Finalmente en 2015 la NASA publicó «*Small UAV automatic ground collision avoidance system design considerations and flight test results*» [14], un memorando técnico en el que se

utilizó el algoritmo aplicado al F-16 [9][10][11] en drones. En el memorando, se obtienen las siguientes conclusiones extraídas y traducidas del mismo:

- «*Aunque la implementación de SUAV Auto GCAS no fue pensada para producción, los resultados de las pruebas son lo suficientemente positivas como para proporcionar una base sólida para escalar a muchas plataformas de UAV de producción.*» [14]
- «*Algunos de los conceptos han demostrado el potencial de proporcionar capacidades significativas para las implementaciones [...] Estos conceptos incluyen el uso de MDE altamente comprimidos, múltiples trayectorias de evasión, y técnicas de exploración del terreno.*» [14]

Tal y como puede observarse, existe un gran potencial en el uso de modelos digitales de elevación para el vuelo autónomo y su cálculo de trayectorias. Sin embargo, todos sus usos han sido hasta ahora militares o científicos, siendo necesaria su implementación en un simulador/controlador de uso cotidiano, para permitir el cálculo de vuelos complejos que de otra forma no podrían realizarse de forma autónoma.

CAPÍTULO 3
Tecnologías utilizadas

En este capítulo se va a dar una visión general de las tecnologías utilizadas en este TFG. Para ello se empezará por analizar el modelo digital de elevación utilizado. A continuación se dará una visión general de ArduSim, el simulador usado para el desarrollo de la propuesta, y la realización de pruebas de validación.

3.1 Modelo digital de elevación

Un modelo digital de elevación o MDE («*Digital Elevation Model - DEM*» en inglés) consiste en una base de datos con un sistema de referencia con el que se obtiene la altura sobre el nivel del mar de una determinada ubicación. Un MDE se caracteriza por su versatilidad en términos de precisión/compresión, siendo posible elegir si deseamos una base de datos más ligera/precisa/amplia, siendo estos tres factores inversamente proporcionales entre ellos.

Un MDE puede ser de dos tipos: modelo digital del terreno (MDT), en el cual solamente se incluyen los datos relativos al relieve geográfico excluyendo edificación y vegetación (Figura 3.1, izquierda); y el modelo digital de superficies (MDS), en el cual sí se han incluido los datos de edificación y/o vegetación (Figura 3.1, derecha).

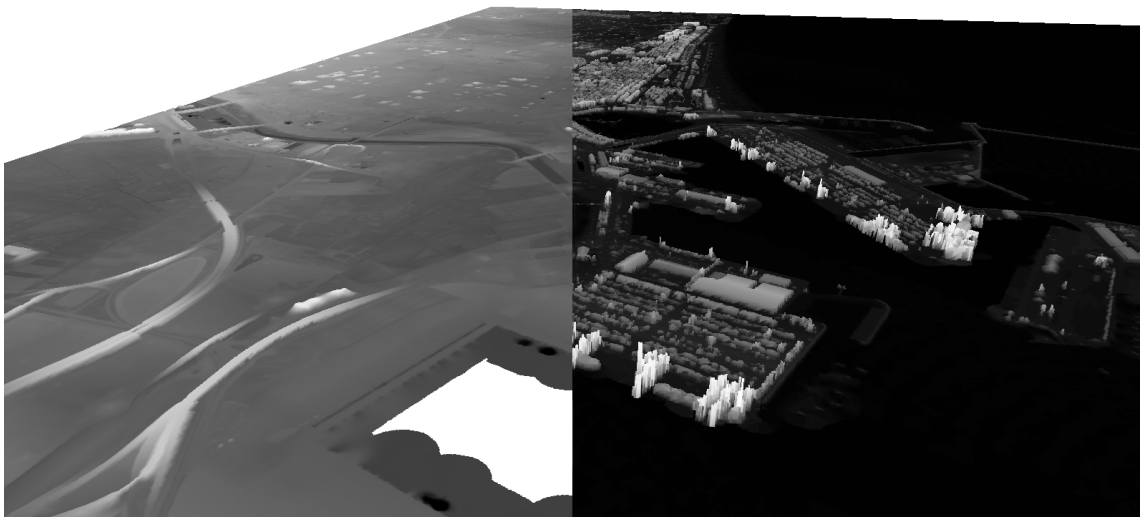

Figura 3.1: Comparación del renderizado 3D de un MDT y un MDS

El elemento principal del MDE es una matriz de gran tamaño en la que cada valor es una medición de altura, y la distancia recorrida entre celdas consecutivas es la llamada precisión, tamaño de celda o paso de malla, y es lo que define la calidad de un MDE, ya que cuanto menos distancia haya entre dos celdas, mayor será su granularidad. Actualmente pueden encontrarse MDEs de todo tipo de precisiones, dependiendo de la superficie que se quiera abarcar, sin alcanzar un tamaño excesivo (ver Figuras 3.2 y 3.3).

Figura 3.2: Renderizado 3D de un MDS con precisión de 5m (100MB)

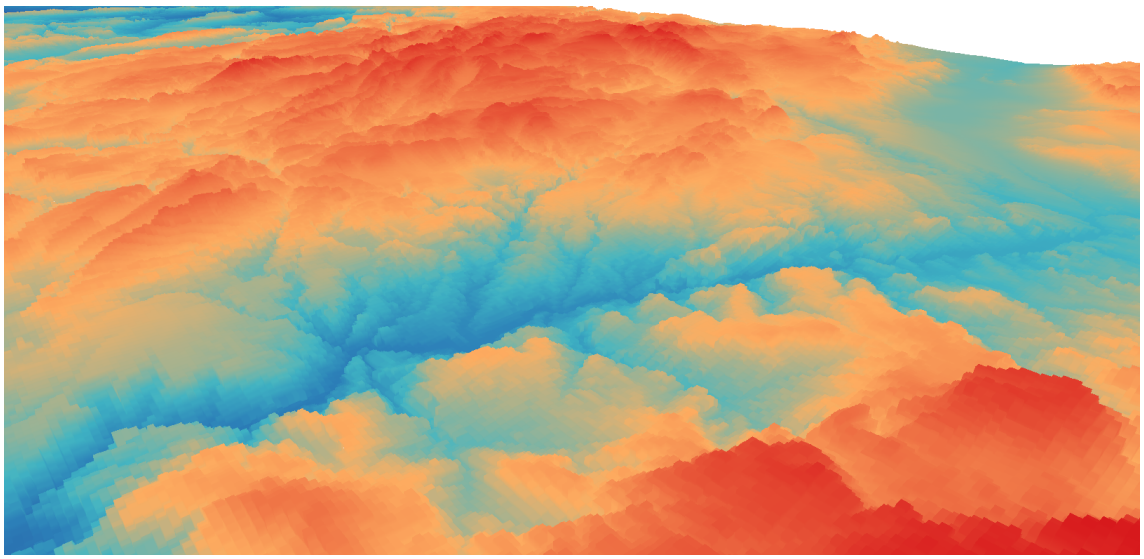

Figura 3.3: Renderizado 3D de un MDT con precisión de 200m (8MB)

Actualmente existen MDEs de todas partes del mundo, y entre ellas cabe destacar el proyecto «The Shuttle Radar Topography Mission» [15] de la NASA que ha conseguido mapear mediante satélite la mayoría de la superficie terrestre con una precisión de hasta 30 metros. Sin embargo, en el proyecto que nos concierne una precisión de 30 metros puede no ser suficiente para un vuelo seguro y eficiente. Pongamos el ejemplo de la ciudad, donde en una superficie de 900 m^2 puede haber distintas alturas que se suman

en una sola, impidiendo que el dron vuele bajo cerca de zonas altas. Sin embargo, esta granularidad de datos podría ser aceptable y aplicable en el caso de paisajes sin un relieve excesivamente pronunciado.

Para este proyecto, se propone utilizar los recursos del «*Plan Nacional de Ortofotografía Aérea*» [16], en el cual disponemos de MDTs (2, 5, 25, 200 metros) y MDSs (5 metros), todas ellas cubriendo el territorio nacional al completo. Esto nos permite tener una amplia variedad de elección de MDEs, según se requieran archivos ligeros o una alta precisión.

3.1.1. WGS / UTM

Cuando se trata de geolocalización, existen dos estándares principales, el *WGS* y el *UTM*. El sistema de geolocalización *WGS* [17] es el más extendido y conocido, ya que se basa en ubicar una posición según sus coordenadas geodésicas (Grados, minutos y segundos) tomando como referencia los ejes del ecuador y el meridiano. La principal ventaja de este método es que permite ubicar cualquier posición a nivel mundial sin depender de otro punto de referencia.

En cuanto al sistema *UTM* [18], se basa en cuadricular el planeta en función de sus usos horarios y una serie de líneas horizontales, inciando el sistema de referencia en la esquina inferior izquierda de cada cuadrícula, lo cual implica que unas mismas coordenadas en este sistema pueden ubicarse en tantos sitios como cuadrículas haya. Su principal ventaja (Y por lo cual es el elegido para los MDE) es que su unidad de representación es el metro, de esta forma permite localizar una ubicación esférica en un plano bidimensional.

Sin embargo, cabe mencionar que los MDEs utilizados utilizan un sistema europeo basado en UTM, el ETRS89 [19], el cual divide los usos horarios únicamente por el ecuador.

3.2 ArduSim

Este proyecto, tal y como se menciona en Objetivos, consiste en integrar la información topográfica de los MDEs en un simulador existente, siendo este simulador ArduSim [4] [20].

ArduSim es un novedoso simulador de vuelo en tiempo real, orientado al desarrollo de protocolos de coordinación de vuelo para drones, realizando misiones planificadas o formando un enjambre. Es capaz de simular múltiples drones simultáneamente, y la cantidad de drones que se pueden simular está restringida principalmente por el ordenador que se utiliza. ArduSim simula una red inalámbrica Ad-hoc para comunicaciones dron a dron. Entre otras cosas, ArduSim genera la ruta seguida por cada dron en formato OMNeT++ o NS2 para proporcionar trazas de movilidad, cuando se realiza una simulación, o incluso al ejecutar ArduSim en un dron real.

Para simular los drones, se utiliza la aplicación SITL como módulo de desarrollo básico. SITL contiene un código de control que se asemeja a un dron real, simulando sus propiedades físicas y de vuelo con gran precisión. Se ejecuta una instancia de SITL para cada dron virtual junto con su motor físico en un solo proceso (Figura 3.4).

La comunicación con los drones utiliza el protocolo MAVLink, un estándar *de facto* para los controladores de vuelo abiertos actuales, convirtiendo el despliegue de un nuevo protocolo en drones reales en una tarea trivial.

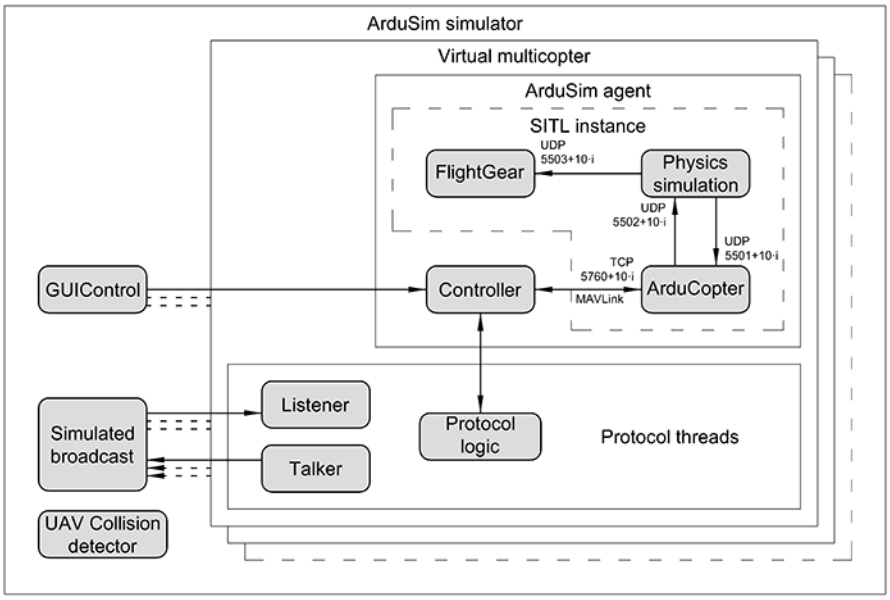

Figura 3.4: Arquitectura de simulación de ArduSim

Fuente: *Repositorio ArduSim*

La comunicación entre drones emula un enlace de red WiFi Ad-hoc en la banda de frecuencia de 5 GHz, donde se difunden todos los paquetes de datos. ArduSim está preparado para incluir nuevos modelos inalámbricos en futuras versiones.

Ardusim fue desarrollado por Francisco Fabra, entonces estudiante de doctorado en el grupo de investigación Grupo de Redes de Computadores (GRC) del Departamento de Informática de Sistemas y Computadores (DISCA) de la Universidad Politécnica de Valencia. Actualmente es mantenido por Jamie Wubben bajo la misma afiliación, y con el mismo supervisor.

Como ejemplo de las capacidades de ArduSim, la Figura 3.5 muestra 72 drones siguiendo misiones elípticas como electrones de un átomo.

Este simulador funciona mediante la implementación y ejecución de protocolos, siendo el más sencillo el protocolo `Mission` en el cual se indican una formación y una serie de banderas o *waypoints*, y el conjunto de drones navega a través de ellas (ver Figura 3.6).

Existen también otros protocolos como `MBCAP` [21], `MUSCOP` [22], `Follow Me` [23], `Vision` [24] y `ShakeUp` [25]. La ejecución del simulador consiste en seleccionar un protocolo, establecer los parámetros del mismo, realizar la simulación, y obtener los resultados (ver Figura 3.7).

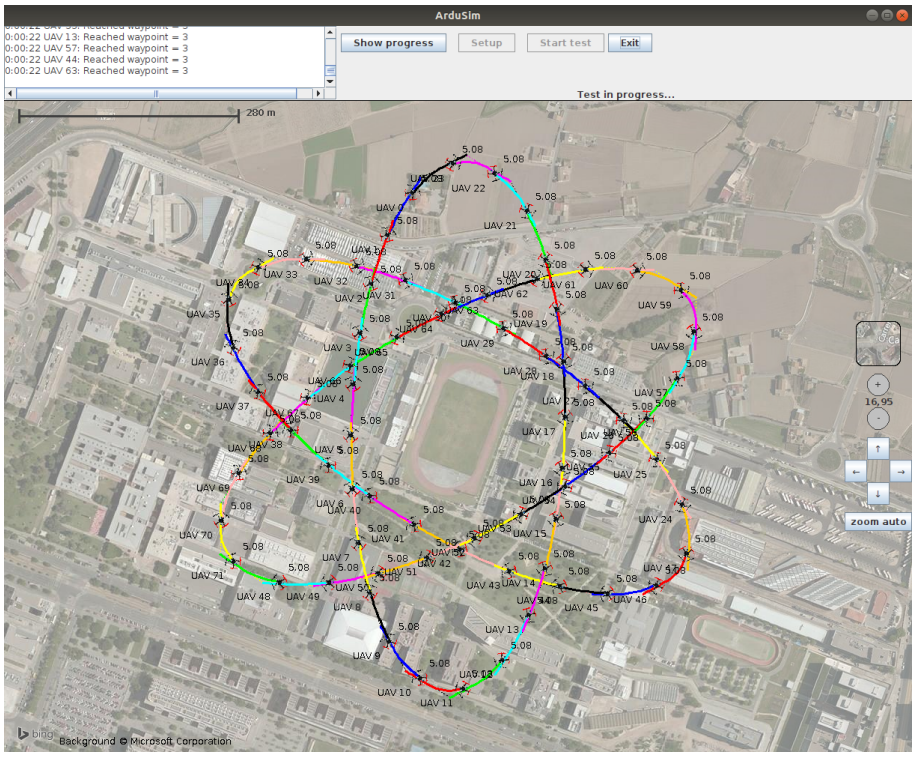

Figura 3.5: Vuelo elíptico de 72 drones simulado en ArduSim

Fuente: *Repositorio ArduSim*

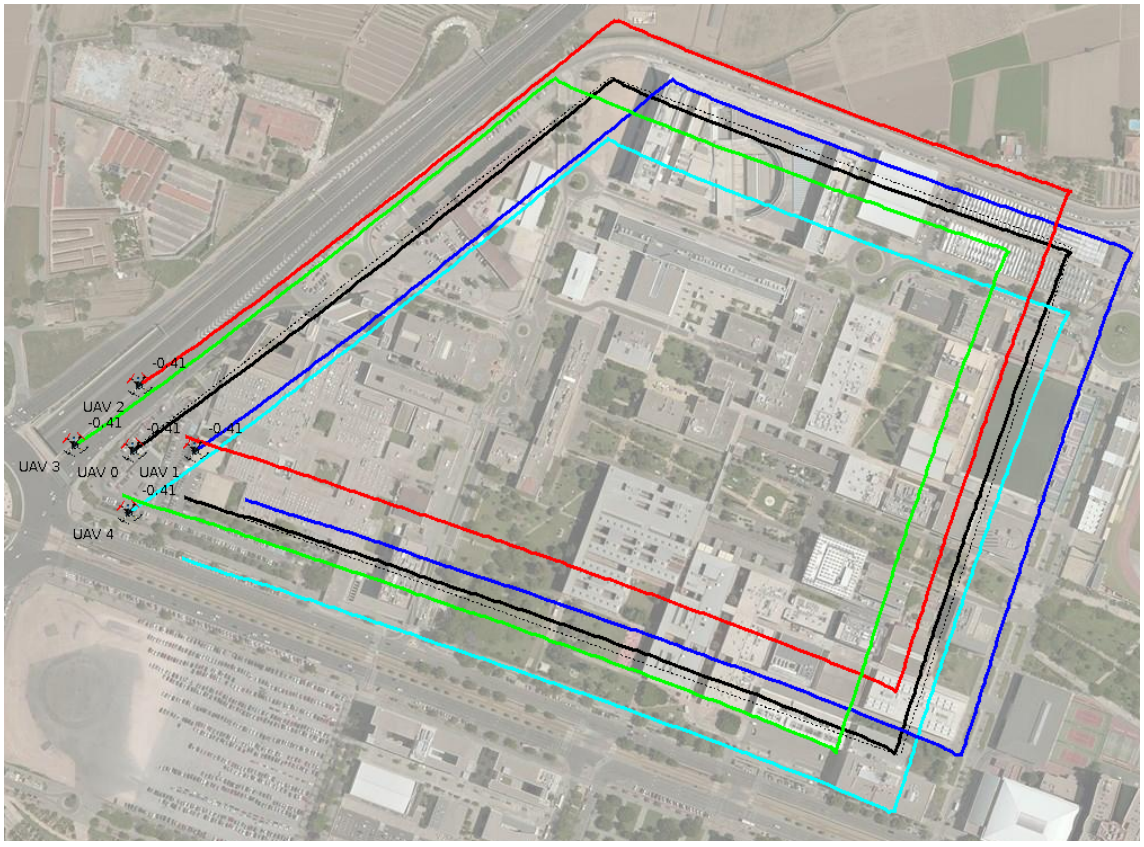

Figura 3.6: Vuelo simulado en ArduSim con 4 *waypoints*

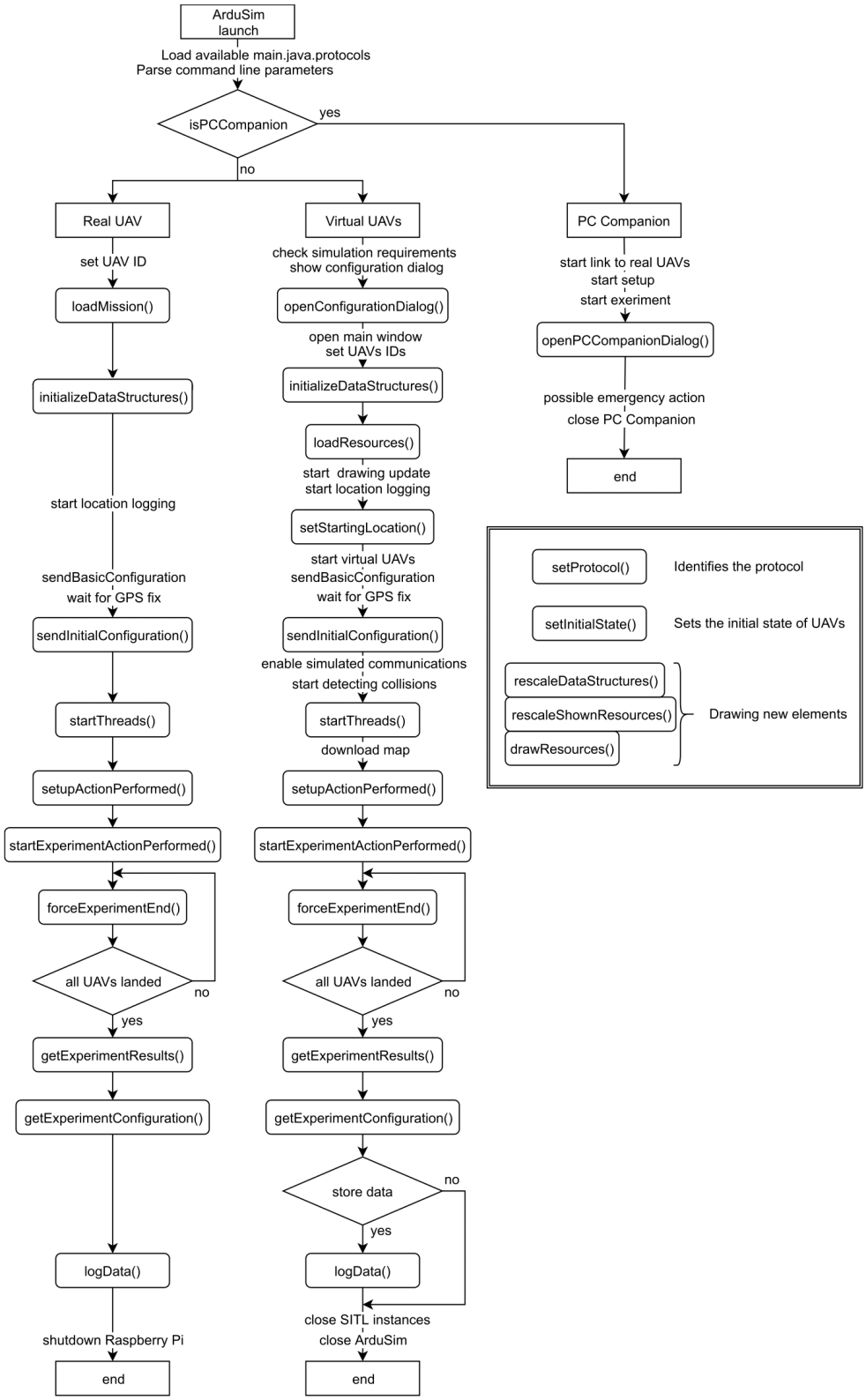

Figura 3.7: Flujo de ejecución de ArduSim

Fuente: *Repositorio ArduSim*

CAPÍTULO 4
Análisis del problema

Actualmente el simulador ArduSim dispone de todas las herramientas necesarias para desarrollar un vuelo seguro y controlado. Utilizando el protocolo `Mission` es posible volar a la altitud indicada de forma constante, de manera que el dron avanza de forma progresiva a través de los distintos *waypoints* y, finalmente, aterriza (Figura 3.6). Sin embargo, esta simulación de vuelo no es en absoluto realista, ya que los datos de altitud de los que dispone el simulador proceden de un altímetro, y por lo tanto son relativos al nivel del mar, sin tener en cuenta la orografía del terreno. Esto significa que mantiene una altura constante sobre el nivel del mar, pero si el terreno que sobrevuela tiene desniveles, el dron no es capaz de corregir su trayectoria. Esto conlleva varias situaciones problemáticas que impiden su uso en la realidad

4.1 Colisión con la superficie terrestre

El primero y más evidente problema al no tener en cuenta la topografía durante el vuelo es la colisión con el terreno cuando en este se produce un desnivel positivo hasta tal punto que alcanza la altura de vuelo del dron (Figura 4.1).

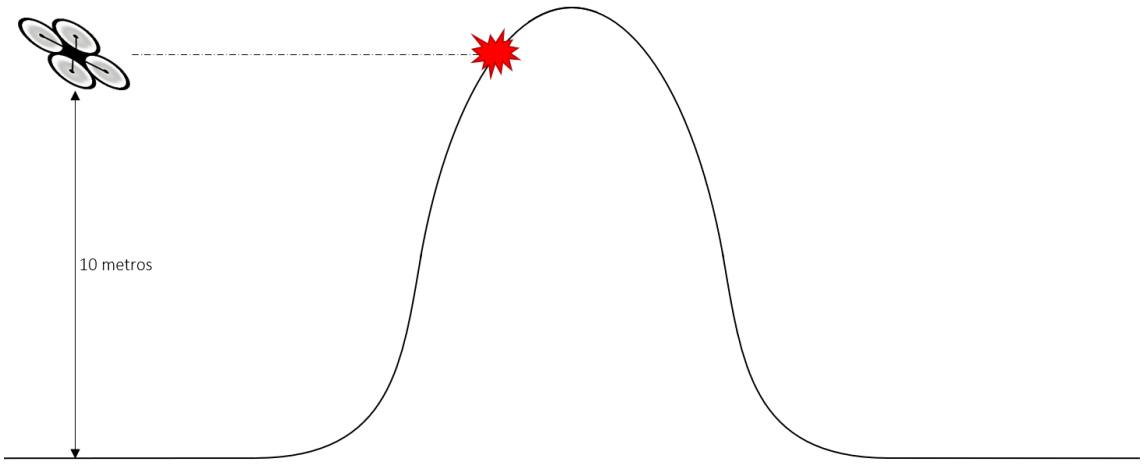

Figura 4.1: Gráfico de colisión por trayectoria.

Dada la orografía nacional, esta problemática se presenta en la mayoría del territorio español, teniendo en cuenta que *España es uno de los países con mayor geodiversidad de Europa y la mitad de su territorio está formado por sistemas montañosos* [26].

4.1.1. Colisión con obstáculos

Suponiendo que se realiza el vuelo en una llanura, con el fin de evitar la situación de la Sección 4.1 tampoco podríamos hablar de un vuelo seguro, ya que además de la orografía, existen otro tipo de obstáculos con los que un dron puede colisionar, como un árbol o un edificio, los cuales además suelen ser poco previsibles y con grandes desniveles en poca distancia; esto, sumado a la problemática de la superficie terrestre, hace que encontrar un área para el vuelo seguro de drones sea casi imposible, especialmente cuando estos tienen que operar de manera autónoma.

4.2 Normativa actual

La actual normativa europea [3] impone la obligación de no superar los 120 metros de altura con respecto al suelo durante el vuelo (salvo autorización explícita), lo cual puede ser tan complejo como los problemas ya mencionados ya que, ante un desnivel negativo, el dron seguiría a una altura ahora superior sin variar su trayectoria (Figura 4.2)

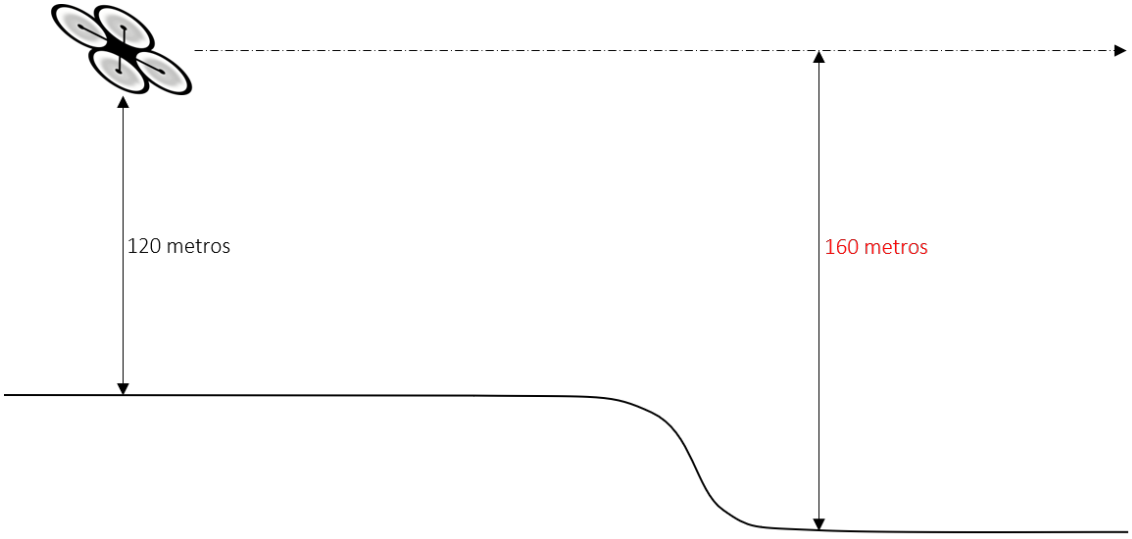

Figura 4.2: Gráfico de desnivel fuera de límites.

Por todo ello, es especialmente relevante el poner integrar datos topográficos en el sistema de vuelo del dron, y el la planificación de misiones, de cara a evitar colisiones con obstáculos, y para cumplir con la actual normativa, de una manera sencilla e precisa.

CAPÍTULO 5
Propuesta de solución

Para resolver los problemas indicados en el punto anterior (4) se propone elaborar un protocolo nuevo basado en `Mission` con el que, dadas una serie de banderas o *waypoints*, ser capaces de volar a través de ellas manteniendo la altura indicada durante todo el vuelo, y teniendo en cuenta los desniveles y obstáculos presentes en el terreno. Para ello se utilizará un MDS facilitado por el Instituto Geográfico Nacional [16].

5.1 Diseño

Para comenzar la etapa de diseño, y tras haber leído la documentación disponible[20], se conoce que disponemos de una API proporcionada por el código en la que se nos ofrecen las distintas herramientas para desarrollar nuestro código, por lo que se elaboró un esquema UML de la misma (Figura 5.1). Además, se ofrece una clase abstracta `ProtocolHelper` para facilitar la implementación de un protocolo nuevo. Esta clase será el punto de partida del nuevo protocolo, que además habrá de seguir la estructura estándar utilizada en ArduSim (Figura 5.2).

Para controlar el movimiento del dron existen dos métodos actualmente disponibles en la clase `Copter`:

- `moveTo(Geo3DLocation)`: El cual navega de forma autónoma el dron hasta una ubicación 3D facilitada.
- `moveTo(vx, vy, vz)`: El cual aplica el vector de velocidad facilitado al dron.

Teniendo en cuenta que la altura tendrá que ser modificada constantemente en función del terreno que se esté sobrevolando, la primera opción no es funcional, ya que no está preparada para la actualización del destino, sino que calcula un sólo vector de movimiento entre el objetivo y el origen, y lo mantiene hasta llegar. Por lo tanto, se desarrollará un algoritmo de cálculo del vector de velocidad en función de la altura deseada en cada momento, y se irá actualizando para cada dron.

Figura 5.1: Esquema UML de la API disponible.

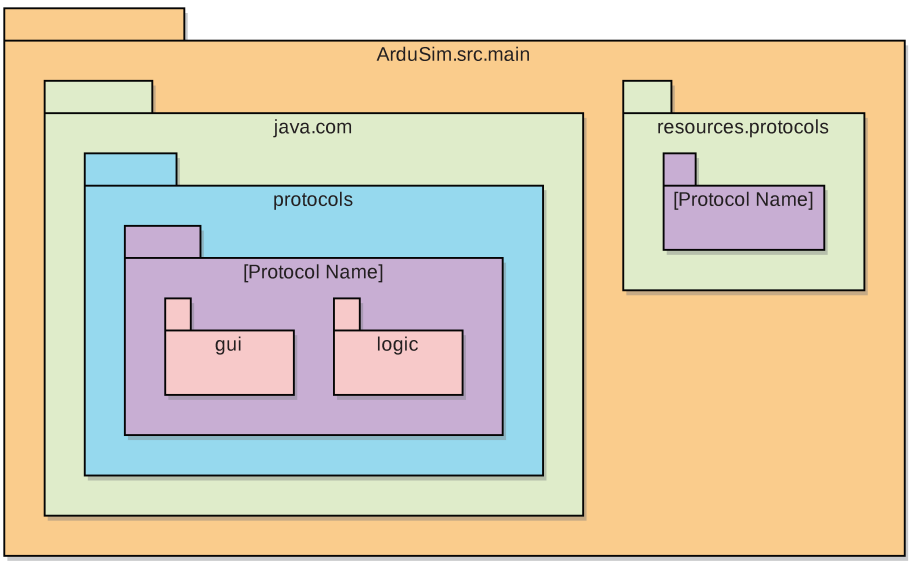

Figura 5.2: Estructura estándar de ArduSim.

Se establecen por lo tanto los siguientes objetivos dentro del desarrollo:

1. Comprobar el funcionamiento de `moveTo` usando una ubicación gps en 3D y realizando el recorrido.
2. Omitiendo la altura, realizar el recorrido usando los ejes «x, y» mediante vectores.
3. Incluir la altura en el cálculo de vectores suponiendo un terreno plano.
4. Incorporar el MDS al código.
5. Obtener la altura del dron en tiempo real desde el MDS.
6. Mantener una altura constante incluyendo los datos del MDS.
7. Predecir los cambios de altura en el recorrido y actuar con antelación.

En cuanto al MDS, se trata de un archivo «Esri ASCII (.asc)», lo cual significa que su información está en formato legible (existen también MDE cuya información está compactada en binario). Dado que un solo archivo para toda la superficie (Nacional o terrestre) tendría un tamaño imposible de tratar, desde la fuente de datos [16] se ha parcelado el territorio nacional, teniendo que elegir el MDS adecuado en función de la zona de vuelo.

A la hora de integrar el MDS en el código, tendrá que hacerse de tal forma que el cambio de MDS sea transparente para el usuario, habiendo de cambiar únicamente el archivo en si. Para lograrlo se propone abstraer el código relativo al tratamiento de MDEs a una clase específica. De esta forma las herramientas de tratamiento de MDEs serán también accesibles desde otros protocolos.

Se propone por lo tanto un desarrollo en tres capas (Figura 5.3):

- Una capa de presentación para los parámetros del protocolo importada directamente del protocolo existente: `Mission`
  - `topography.gui.TopographyDialogApp`
  - `topography.gui.TopographyDialogController`
  - `topography.gui.TopographySimProperties`
- Una capa lógica desarrollada en la clase que implementa: `ProtocolHelper`
  - `topography.logic.TopographyHelper`
- Una capa de datos que trata la información presente en el MDE:
  - `topography.logic.DEM`

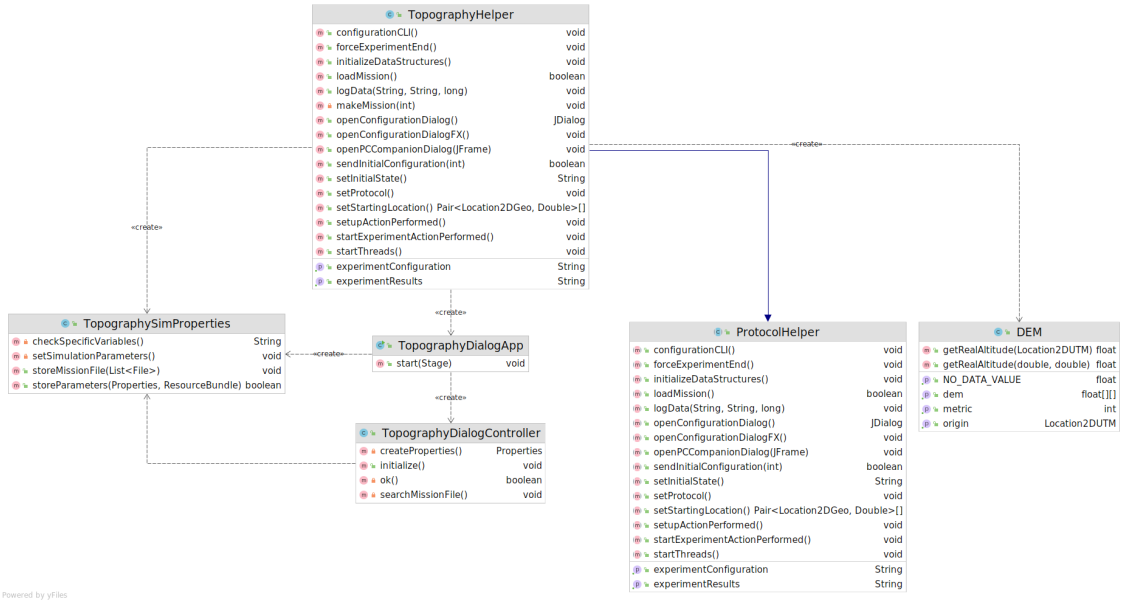

Figura 5.3: Esquema UML del protocolo creado.

5.2 Desarrollo

Siguiendo con las pautas marcadas en la etapa de Diseño, se comienza el protocolo `Topography` partiendo del ya existente protocolo `Mission`. Para ello se intenta realizar un vuelo simple a un determinado *waypoint*. Para lograrlo, se modifica el método principal, creando en él una serie de hilos (uno por dron) desde los cuales se llama al método `makeMission(dronId)`; este método será donde se realice el control de los drones durante el vuelo, y habrá tantos hilos ejecutándolo como drones volando. En el Algoritmo 1 puede apreciarse cómo se obtienen los `waypoints`, y cómo se ordena el movimiento de los drones.

Algoritmo 1 Realizar misión del hito 1.

Require: $numUAV$
  $copter = API.getCopter(numUAV)$
  $waypoints[] = copter.getMissionHelper().getMissionsLoaded()[0]$
  for all $waypoint$ in $waypoints[]$ **do**
    $waypoint3D = Location3D(waypoint.getUTM(), waypoint.getAltitude())$
    $copter.moveTo(waypoint3D)$
  end for

Una vez conseguido el hito 1, el siguiente objetivo es utilizar el movimiento por vectores, lo cual se ha conseguido mediante el Algoritmo 2; en él, además del cálculo del vector unitario de movimiento, puede apreciarse el uso de un factor de ralentización utilizado cuando el dron se aproxima al `waypoint` para reducir su velocidad progresivamente.

Algoritmo 2 Realizar misión del hito 2.

Require: $numUAV$
  $copter = API.getCopter(numUAV)$
  $waypoints[] = copter.getMissionHelper().getMissionsLoaded()[0]$
  for all $waypoint$ in $waypoints[]$ **do**
    $distancia = |waypoint.distancia(copter.ubicacion)|$
    while $distancia > 5$ **do**
      $distancia = |waypoint.distancia(copter.ubicacion)|$
      $vectorX = waypoint.X - copter.ubicacion.X$
      $vectorY = waypoint.Y - copter.ubicacion.Y$
      $aux = \text{máx}\,(|vectorX|, |vectorY|)$
      $slowFactor = \text{mín}\,(distancia/(copter.velocidadBase * 7{,}5), 1)$
      $vectorX = slowFactor * copter.velocidadBase * vectorX/aux$
      $vectorY = slowFactor * copter.velocidadBase * vectorY/aux$
      $copter.moveTo(vectorX, vectorY, 0)$
    end while
    $copter.moveTo(0, 0, 0)$
  end for
  $copter.moveTo(0, 0, 0)$

Conseguir el hito 3 es relativamente sencillo una vez se comprende el movimiento tridimensional; para ello basta con ajustar el cálculo de la distancia al `waypoint`, e incorporar la componente al vector (Algoritmo 3). Se ha implementado también un factor de ralentización cuando el dron se aproxima al `waypoint` para procurar que frene a tiempo. En el caso del `vectorZ` los valores se invierten ya que el simulador orienta un valor positivo hacia abajo.

Algoritmo 3 Realizar misión del hito 3.

Require: $numUAV$
  $copter = API.getCopter(numUAV)$
  $waypoints[] = copter.getMissionHelper().getMissionsLoaded()[0]$
  for all $waypoint$ in $waypoints[]$ **do**
    $distancia3D = |waypoint.ubicacion3D.distancia3D(copter.ubicacion3D)|$
    while $distancia3D > 5$ **do**
      $distancia3D = |waypoint.ubicacion3D.distancia3D(copter.ubicacion3D)|$
      $vectorX = waypoint.X - copter.ubicacion.X$
      $vectorY = waypoint.Y - copter.ubicacion.Y$
      $vectorZ = waypoint.altura - copter.altura$
      $aux = \text{máx}\,(|vectorX|, |vectorY|, |vectorZ|)$
      $slowFactor = \text{mín}\,(distancia/(copter.velocidadBase * 7{,}5), 1)$
      $vectorX = slowFactor * copter.velocidadBase * vectorX/aux$
      $vectorY = slowFactor * copter.velocidadBase * vectorY/aux$
      $vectorZ = -1 * slowFactor * copter.velocidadBase * vectorZ/aux$
      $copter.moveTo(vectorX, vectorY, vectorZ)$
    end while
    $copter.moveTo(0, 0, 0)$
  end for
  $copter.moveTo(0, 0, 0)$

A continuación se muestra cómo actúa el constructor de la clase `DEM`, el cual procesa los datos del MDE y ofrece acceso a sus distintas propiedades. Nótese que la matriz se rellena desde el final hasta el principio; esto es porque, en una matriz usual, el inicio se concibe en la esquina superior izquierda, mientras que en el MDE se sitúa en la esquina inferior izquierda, por lo que el algoritmo lo corrige de esta forma (ver Algoritmo 4).

Una vez disponemos del objeto `DEM`, falta localizar una ubicación real en la matriz. Para ello basta con calcular el desplazamiento desde el punto origen de la matriz hasta el dron, y aplicarlo sobre la misma (Algoritmo 5) teniendo en cuenta el paso de malla o precisión. El algoritmo utilizado se proporciona mediante un método público desde la clase `DEM` para facilitar su uso. Nótese que se accede a la matriz de la forma $[y][x]$; esto se debe a que la matriz utiliza un sistema de coordenadas, por lo tanto avanzar en el eje vertical (filas) corresponde a la componente y, mientras que avanzar en una misma fila corresponde a la componente x.

El hito 6 resulta uno de los más complejos, ya que se basa en combinar los algoritmos 3 y 5, sustituyendo los datos de altura hasta ahora obtenidos respecto al nivel del mar por los datos obtenidos por el MDE. Tal y como puede verse en el Algoritmo 6, se ha implementado una nueva forma de calcular el vector `Z`.

Además, se aplica un factor de ralentización en el descenso, siendo este la mitad de la velocidad base del dron, esto se debe a que en pruebas beta del protocolo se observaron colisiones contra el suelo por inercia de bajada, siendo el dron incapaz de frenar a tiempo. Se modifica también el cálculo del factor de ralentización del eje `Z` cuando el dron se aproxima al `waypoint` volviéndolo independiente de los ejes `X` e `Y`, de esta forma tiene prioridad sobre ellos. Se ha implementado también un sistema de frenado de emergencia en caso de detectar una altura anómalamente baja, en cuyo caso se deja de avanzar en los ejes `(X, Y)` y se sube hasta recuperar la altura mínima.

Algoritmo 4 Realizar misión del hito 4

Require: $MDE.asc$
  $nCols = MDE.linea1$
  $nRows = MDE.linea2$
  $origenX = MDE.linea3$
  $origenY = MDE.linea4$
  $precision = MDE.linea5$
  $noDataValue = MDE.linea6$
  $tab = nRows - 1$
  $mde = float[nRows][nCols]$
  while not $MDE.estaVacio$ **do**
    $filaS[] = MDE.siguienteLinea.dividir(' ')$
    $filaF[] = float[filaS.length]$
    for all $valorS$ in $filaS$ **do**
      $valorF = parseFloat(valorS)$
      if $valorF = noDataValue$ **then**
        $valorF = 0$
      end if
      $filaF.add(valorF)$
    end for
    $mde[tab] = filaF$
    $tab = tab - 1$
  end while

Algoritmo 5 Realizar misión del hito 5.

Require: $numUAV$
  $dem = DEM(MDE.asc)$
  $mde[][] = dem.getDem()$
  $copter = API.getCopter(numUAV)$
  $desplazamientoY = (copter.posicion.Y - dem.origenY)/dem.precision$
  $desplazamientoX = (copter.posicion.X - dem.origenX)/dem.precision$
  return $mde[desplazamientoY][desplazamientoX]$

Algoritmo 6 Realizar misión del hito 6.

Require: $numUAV, dem$
  $copter = API.getCopter(numUAV)$
  $waypoints[] = copter.getMissionHelper().getMissionsLoaded()[0]$
  for all $waypoint$ in $waypoints[]$ **do**
    $distancia2D = |waypoint.distancia(copter.ubicacion)|$
    $distancia3D = |waypoint.ubicacion3D.distancia3D(copter.ubicacion3D)|$
    while $distancia3D > 5$ **do**
      $distancia2D = |waypoint.distancia(copter.ubicacion)|$
      $distancia3D = |waypoint.ubicacion3D.distancia3D(copter.ubicacion3D)|$
      $vectorX = waypoint.X - copter.ubicacion.X$
      $vectorY = waypoint.Y - copter.ubicacion.Y$
      $aux = \text{máx}\,(|vectorX|, |vectorY|)$
      $slowFactor = \text{mín}\,(distancia/(copter.velocidadBase * 7{,}5), 1)$
      $groundAlt = dem.getRealAltitude(copter.ubicacion)$
      $dronAltReal = copter.altura - groundAlt$
      $objetivoAlt = waypoint.altura + groundAlt$
      $bool = objetivoAlt \leq dronAltReal$
      $vectorZ = 0$
      if $bool$ **then**
        $vectorZ = 0{,}5$
      else
        $vectorZ = -1$
      end if
      $vectorX = slowFactor * copter.velocidadBase * vectorX/aux$
      $vectorY = slowFactor * copter.velocidadBase * vectorY/aux$
      $vectorZ = \text{mín}\,(copter.velocidadBase, |objetivoAlt - copter.altura|) * vectorZ$
      if $dronAltReal < 9$ **then**
        $vectorX = 0$
        $vectorY = 0$
        $vectorZ = -1$
      end if
      $copter.moveTo(vectorX, vectorY, vectorZ)$
    end while
    $copter.moveTo(0, 0, 0)$
  end for
  $copter.moveTo(0, 0, 0)$

El último hito (7) completa el algoritmo final del proyecto (Algoritmo 7), en el cual no sólo se tiene en cuenta la altura real presente del dron, sino también sus futuras alturas hasta un máximo de metros que se indican como parámetro, esta será la capacidad de predicción del dron, la cual le permite obtener cuál va a ser su peor dato de altura en los próximos X metros y actuar en consecuencia. Llegados a este punto, el dron es capaz de predecir sus futuros objetivos y actuar con antelación, teniendo también en cuenta si no va a ser capaz de superarlos con un movimiento gradual, en cuyo caso se activa también el sistema de frenado de emergencia.

Algoritmo 7 Realizar misión del hito 7.

Require: $numUAV, dem, prediccion$
  $copter = API.getCopter(numUAV)$
  $waypoints[] = copter.getMissionHelper().getMissionsLoaded()[0]$
  $iteraciones = prediccion/dem.precision$
  for all $waypoint$ in $waypoints[]$ **do**
    $distancia2D = |waypoint.distancia(copter.ubicacion)|$
    $distancia3D = |waypoint.ubicacion3D.distancia3D(copter.ubicacion3D)|$
    while $distancia3D > 5$ **do**
      $distancia2D = |waypoint.distancia(copter.ubicacion)|$
      $distancia3D = |waypoint.ubicacion3D.distancia3D(copter.ubicacion3D)|$
      $vectorX = waypoint.X - copter.ubicacion.X$
      $vectorY = waypoint.Y - copter.ubicacion.Y$
      $aux = \text{máx}\,(|vectorX|, |vectorY|)$
      $slowFactor = \text{mín}\,(distancia/(copter.velocidadBase * 7{,}5), 1)$
      $groundAlt = dem.getRealAltitude(copter.ubicacion)$
      $dronAltReal = copter.altura - groundAlt$
      $maxFutureAlt = 0$
      for $i = 1$ **to** $iteraciones$ **do**
        if $i * dem.precision \leq distancia2D$ **then**
          $proyecX = ((copter.ubicacion.X - dem.origenX)/dem.precision + vectorX * i$
          $proyecY = ((copter.ubicacion.Y - dem.origenY)/dem.precision + vectorY * i$
          $futureAlt = dem.getDem()[proyecY][proyecX]$
          if $futureAlt > maxFutureAlt$ **then**
            $maxFutureAlt = futureAlt$
          end if
        end if
      end for
      $objetivoAlt = waypoint.altura + maxFutureAlt$
      $bool = objetivoAlt \leq dronAltReal$
      $vectorZ = 0$
      if $bool$ **then**
        $vectorZ = 0{,}5$
      else
        $vectorZ = -1$
      end if
      $vectorX = slowFactor * copter.velocidadBase * vectorX/aux$
      $vectorY = slowFactor * copter.velocidadBase * vectorY/aux$
      $vectorZ = \text{mín}\,(copter.velocidadBase, |objetivoAlt - copter.altura|) * vectorZ$
      if $dronAltReal < 9$ **or** $maxFutureAlt - copter.altura > prediccion$ **then**
        $vectorX = 0$
        $vectorY = 0$
        $vectorZ = -1$
      end if
      $copter.moveTo(vectorX, vectorY, vectorZ)$
    end while
    $copter.moveTo(0, 0, 0)$
  end for
  $copter.moveTo(0, 0, 0)$

5.3 Implantación

Una vez finalizada la etapa de desarrollo, sólo queda implantar y ejecutar los módulos creados. Para ello, y dado que no se han tratado en la etapa anterior, lo primero es explicar cuáles son los distintos recursos presentes en el protocolo.

- `mission.fxml`: Archivos de configuración para la capa de presentación.
- `mission.kml`: Archivo descargable de Google Earth con los datos de los `waypoints`.
- `missionParam.properties`: Parámetros modificables de la misión.
- `dem.asc`: MDE descargado.

En cuanto al archivo `kml`, la forma más sencilla de obtenerlo consiste en crear un nuevo `path` en *Google Earth Pro* y exportarlo como archivo `kml`, sin embargo, es posible modificar el archivo existente indicando en el apartado de «coordinates» las coordenadas de los distintos waypoints por orden de consecución, comenzando por la posición incial.

Es importante utilizar el MDE adecuado, ya que sino se producirá una excepción al no poder ubicar el dron dentro de la matriz. En caso de que el trayecto del vuelo transcurra entre dos MDEs, se aconseja utilizar una MDE de menor precisión que englobe el área completa.

Una vez los archivos están en la ubicación correcta (*ArduSim/src/main/resources/protocols/topography*) ya se puede ejecutar ArduSim, y tras las pantallas de configuración (ver Figura 5.4) llegaremos a la simulación, donde primero hay que iniciar el `setup` y posteriormente el `test`. Podemos obtener información en tiempo real durante el vuelo, y estos valores se utilizarán en la siguiente sección de Validación y resultados para comprobar que el protocolo funciona correctamente.

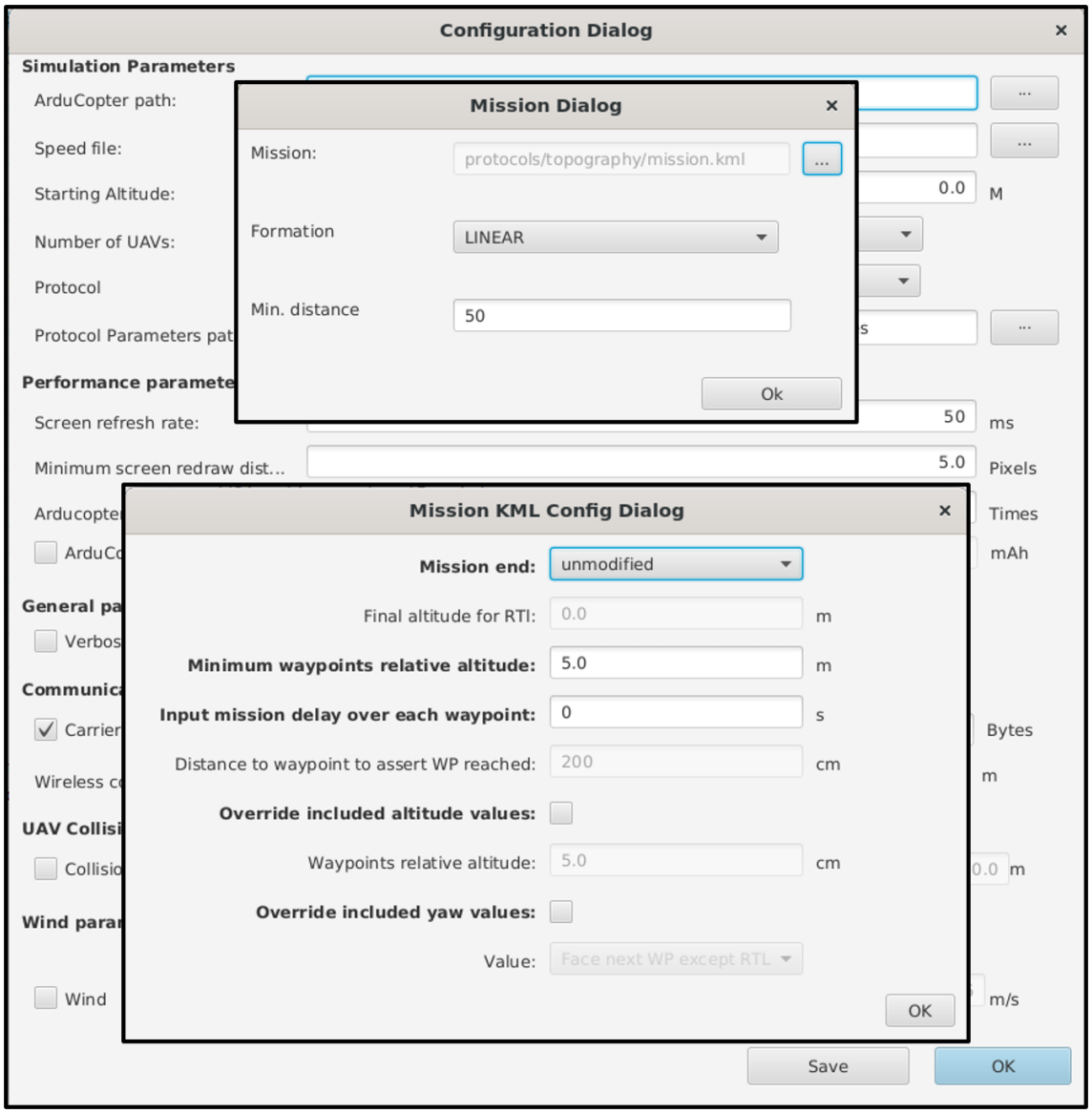

Figura 5.4: Ventanas de configuración de ArduSim.

CAPÍTULO 6
Validación y resultados

Para comprobar que el protocolo cumple con los objetivos propuestos, el simulador genera unos archivos de log con el formato `logn.csv` donde `n` es el identificador del dron. Estos logs consisten de un fichero en formato *csv* donde cada 500 ms aproximadamente (depende de la tasa de refresco calculada) se introduce una nueva línea con los datos del vuelo en ese momento, en la misma se incluyen parámetros tanto del dron como del terreno y los distintos `waypoints`.

Una vez se han generado los distintos ficheros, se ensamblan en un único *csv* para su posterior procesamiento. Se ha utilizado la herramienta PowerBI para obtener las distintas gráficas.

Desde la herramienta PowerBI, una vez importado el log, se pueden realizar tantas combinaciones de las distintas variables como se desee.

Para realizar la validación del protocolo se han propuesto tres escenarios muy distintos entre sí, en todos ellos volarán tres drones en enjambre (formación linear), comprobando que cumple con los requisitos establecidos en todos ellos:

- Vuelo en perpendicular a la falda inclinada de una montaña, con grandes desniveles a lo largo de largas distancias.
- Vuelo sobre la UPV teniendo en cuenta los edificios con muchos desniveles en cortas distancias.
- Vuelo a través de los acantilados de Barbate (Cádiz), con desniveles de +100 metros en muy cortas distancias.

Para realizar los experimentos se siguieron los pasos mencionados en la Sección 5.3 (Implantación). Para ello se descargaron las distintas DEMs y se obtuvieron los archivos `kml` para las misiones. Una vez ubicados los archivos se ejecutó el experimento y se generaron los logs. A continuación se muestran los resultados obtenidos.

6.1 Falda de montaña

Los experimentos realizados en este apartado se ubican en la Sierra de Chiva (WGS: 39.529933, -0.796618), una zona con un desnivel muy grande y una pendiente de 45º. En este entorno se espera que el dron se desenvuelva sin problemas ya que, a pesar de los desniveles, tiene distancia suficiente para poder esquivarlos. Sin embargo, para poner a prueba el simulador, el vuelo se ha realizado con los `waypoints` a 5 metros de la superficie, y con un margen de previsión de 50 metros. De esta forma el dron tiene poco tiempo de

respuesta y unas alturas bastante ajustadas. Los resultados, sin embargo, fueron del todo exitosos (Figura 6.1).

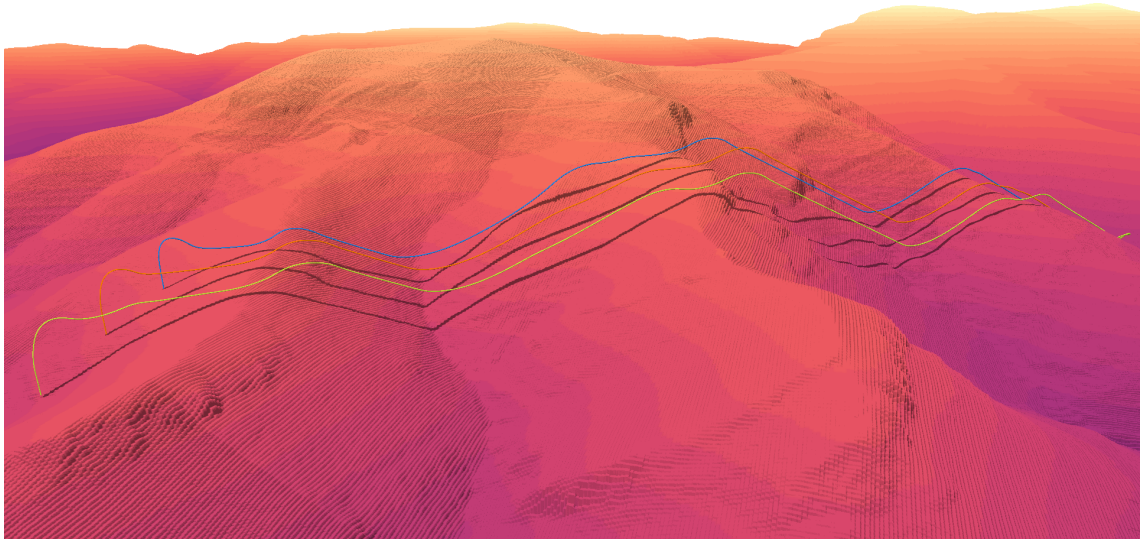

Figura 6.1: Renderizado 3D del vuelo sobre la Sierra de Chiva.

Tal y como puede apreciarse en la Figura 6.1, los drones siguen la trayectoria del terreno sin llegar a colisionar, manteniendo la altura más baja posible hasta 5 metros. En este punto destacar que mantener la altura a 5 metros no siempre es posible debido a 2 factores principales: (i) existe un factor de ralentización en el descenso para evitar colisiones por inercia; y (ii) el dron comienza a ascender con antelación ante un desnivel positivo inminente.

Posteriormente se procede al cálculo de los gráficos para comprobar que el vuelo cumple los parámetros establecidos (ver Figura 6.2).

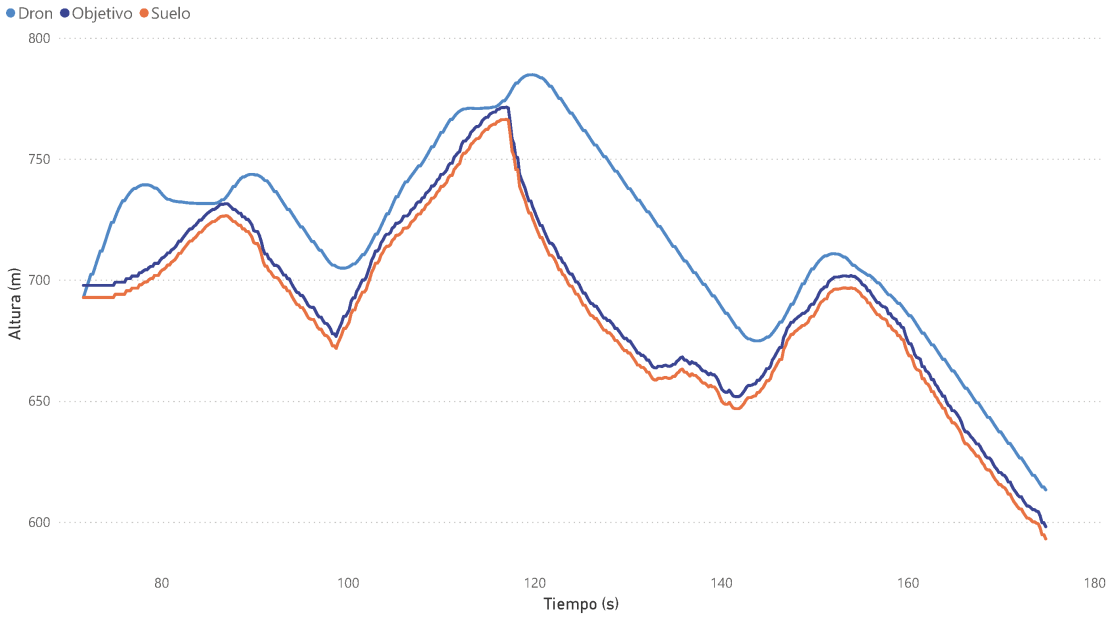

Figura 6.2: Vuelo Montaña: Altitudes (Dron 1).

Observando la Figura 6.2 podemos observar, a lo largo del tiempo, las diferencias entre la altura del dron y la altitud marcada como objetivo, se incluye también la altitud del terreno como referencia. Mediante esta gráfica comprobamos cómo el dron nunca decae por debajo de la altitud definida como objetivo (terreno + altitud del `waypoint`). Pueden apreciarse también los factores de ralentización y prevención descritos anteriormente.

La siguiente gráfica (Figura 6.3) muestra las alturas relativas de los distintos drones, pudiendo comprobar que, en general, los drones no han bajado de los 5 metros, o en el caso de hacerlo se ve que rápidamente han podido recuperar la altitud.

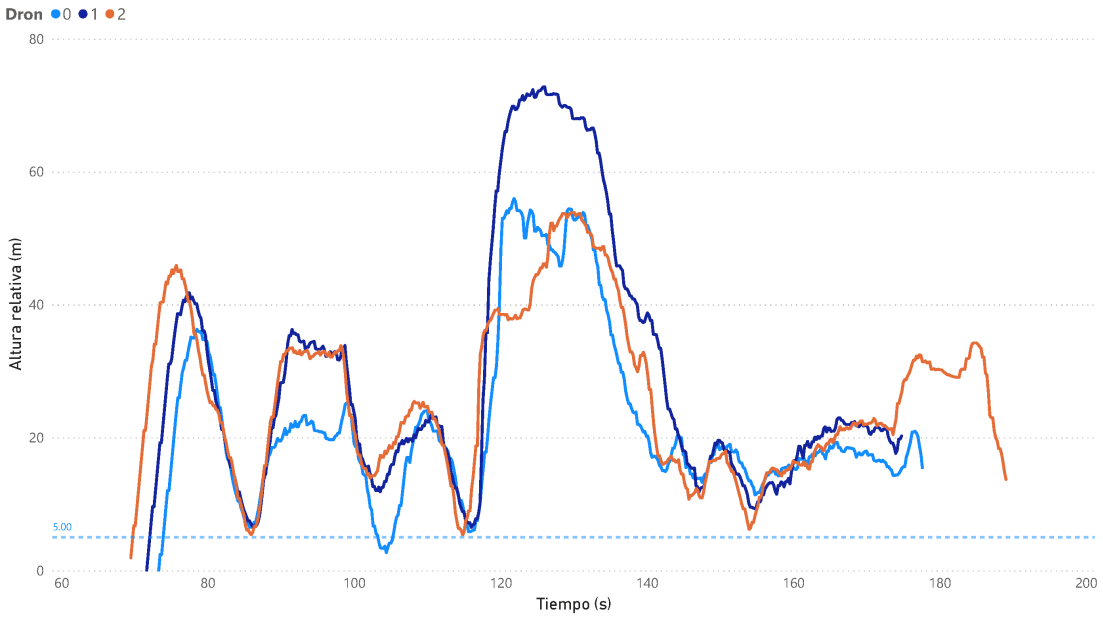

Figura 6.3: Vuelo en Montaña: Alturas relativas de los drones.

Otro factor a evaluar es la sincronización entre los drones, esto es, si todos han llegado al `waypoint` a la vez, o si alguno ha sufrido un retraso durante el vuelo. Podemos comprobarlo mediante la gráfica de la Figura 6.4, en ella podemos observar un ligero retraso en el dron 2 al llegar al `waypoint`, sin embargo, esto fue debido a un mayor desnivel en su ruta en concreto.

Dado que este vuelo no tiene una complejidad elevada, estas gráficas son suficientes para concluir que el vuelo se realizó con éxito.

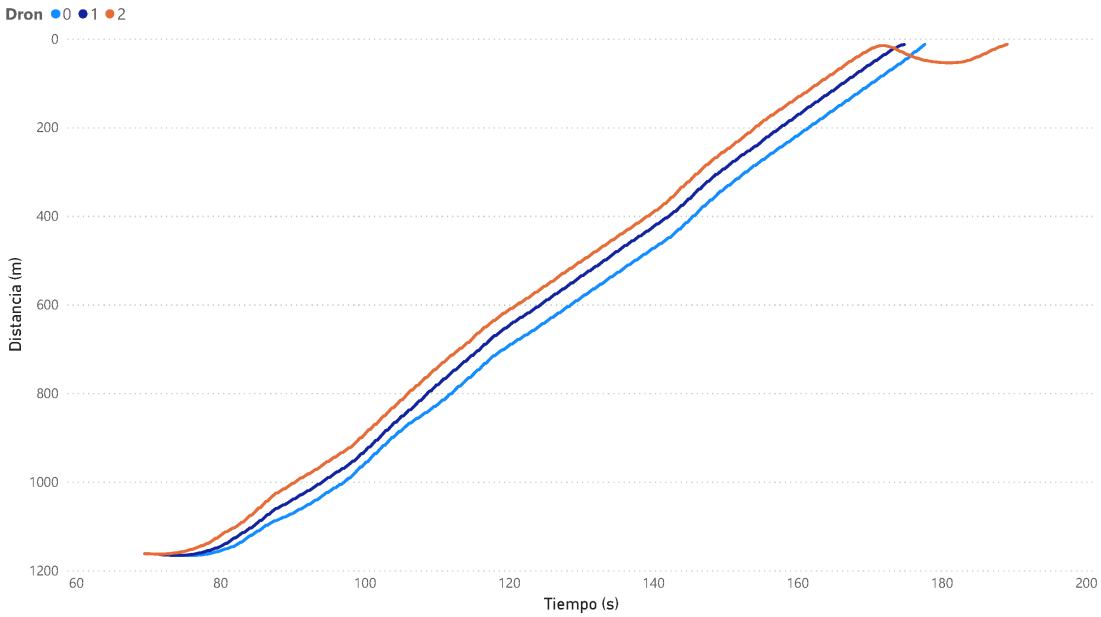

Figura 6.4: Vuelo Montaña: Sincronización entre drones

6.2 UPV

Este segundo vuelo se simuló sobre la UPV, siendo su misión rodearla por completo. Se trata de un entorno bastante complejo para el protocolo ya que, al usar un MDS, se tienen en cuenta todos los edificios, lo cual supone una constante variación de alturas en pendiente de 90º; además, se han mantenido los 50 metros de previsión. Sin embargo, la altura de los `waypoints` se encuentra a 10 metros de altura, ofreciendo un poco más de margen (ver Figura 6.5).

En cuanto a los resultados obtenidos, no hubo ninguna colisión y los drones fueron capaces de superar con éxito el desafío. Sin embargo, se detectó una pequeña falta de precisión: al volar dos de los drones en momentos dispares muy cerca de la fachada de un edificio, durante unos milisegundos, el protocolo ubicó al dron sobre el edificio en lugar de al lado. Esto se debe a la precisión del MDS, que en este caso es de 5 metros (25 m^2).

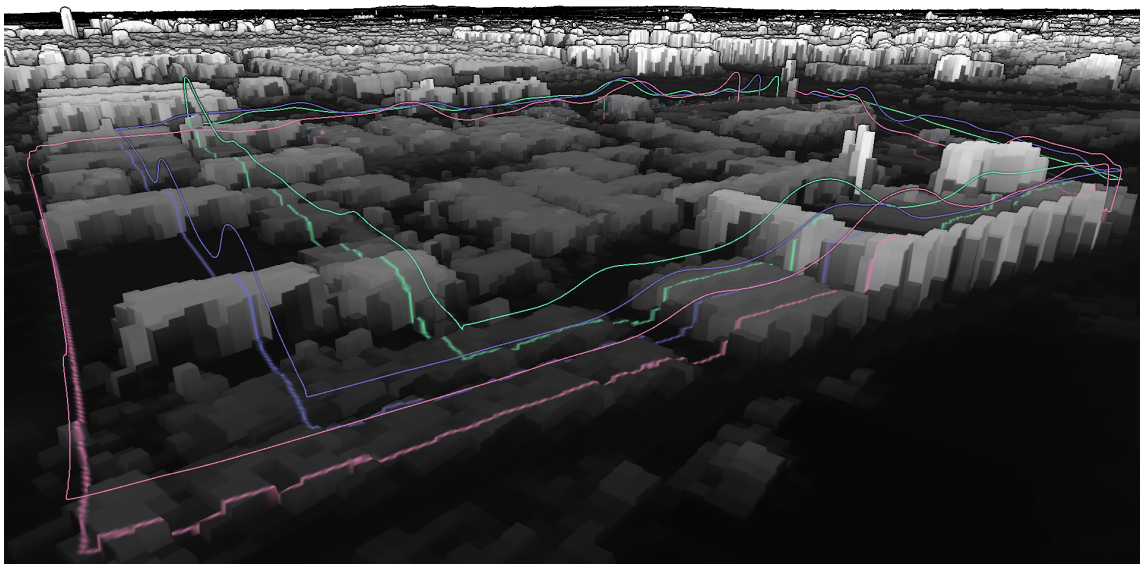

Figura 6.5: Renderizado 3D del vuelo sobre la UPV.

Para realizar un análisis más detallado contamos con las mismas gráficas que en el apartado anterior, incluyendo alguna más dada la complejidad del vuelo.

En la Figura 6.6 puede observarse cómo el dron actúa en previsión de las distintas alturas del terreno, manteniéndose siempre por encima del umbral impuesto. En este caso se ha omitido la línea del terreno ya que, dada la complejidad del mismo dificultaba la comprensión del gráfico. A lo largo del recorrido puede observarse también cómo el dron decide no bajar hasta la altura objetivo para evitar la colisión ante un futuro desnivel.

Gracias a la gráfica de alturas relativas (ver Figura 6.7) podemos visualizar rápidamente en qué momento y en qué dron sucede la anomalía previamente comentada. Cabe destacar que, incluso tratándose de un dato erróneo, el dron no llegaría a colisionar, pasando a ras de suelo.

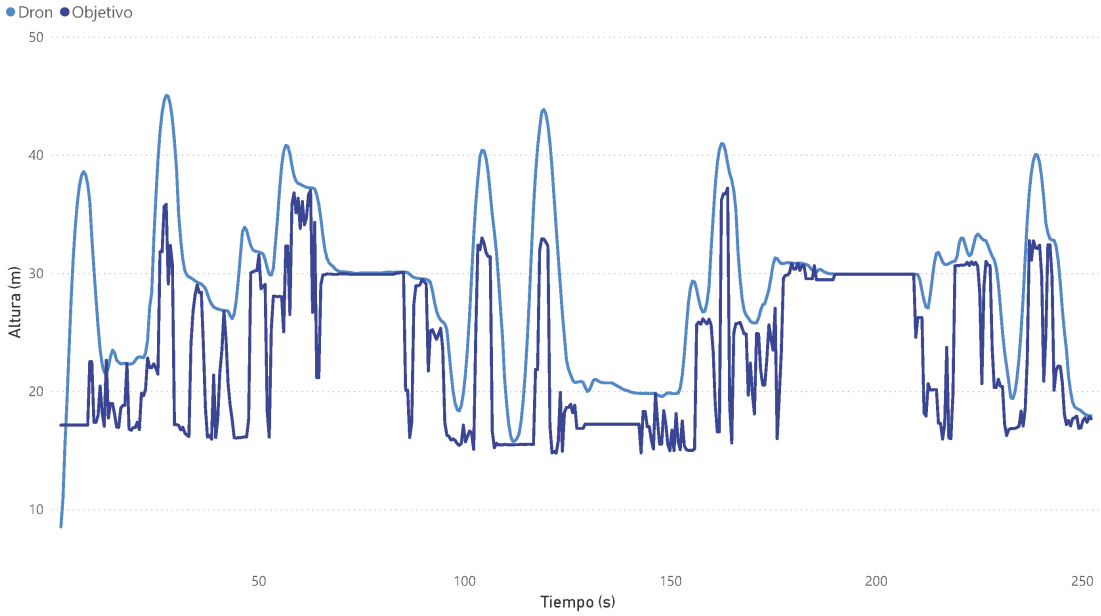

Figura 6.6: Vuelo UPV: Altitudes (Dron 1).

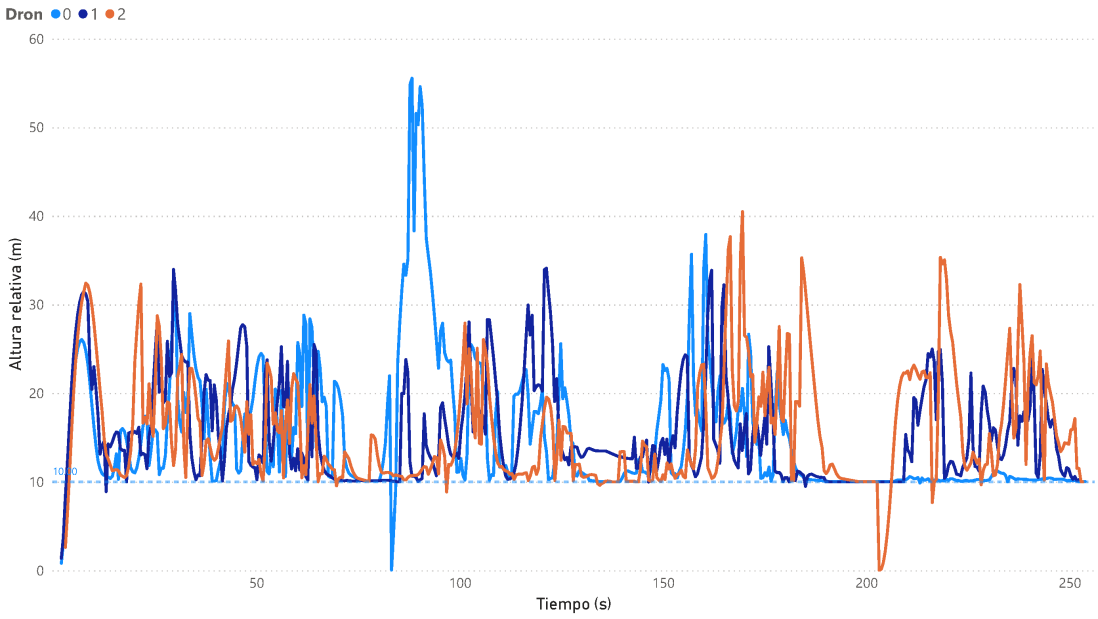

Figura 6.7: Vuelo UPV: Alturas relativas de los drones

En la gráfica de sincronización de drones (ver Figura 6.8) se puede observar la calidad de sincronización a pesar de los desniveles en el vuelo. En cuanto a los saltos, estos se deben al cambio de `waypoint`, momento en que la valoración de la distancia comienza de nuevo. Puede apreciarse también cómo los drones esperan a que todos lleguen al `waypoint` antes de continuar, ya que empiezan 100 % sincronizados en cada `waypoint`, como suele ser habitual en los vuelos de enjambres basados en misiones. También en esta gráfica puede apreciarse la anomalía ya que, debido al fallo de lectura, el dron 2 tarda unos segundos en detectar que ha llegado al tercer `waypoint`.

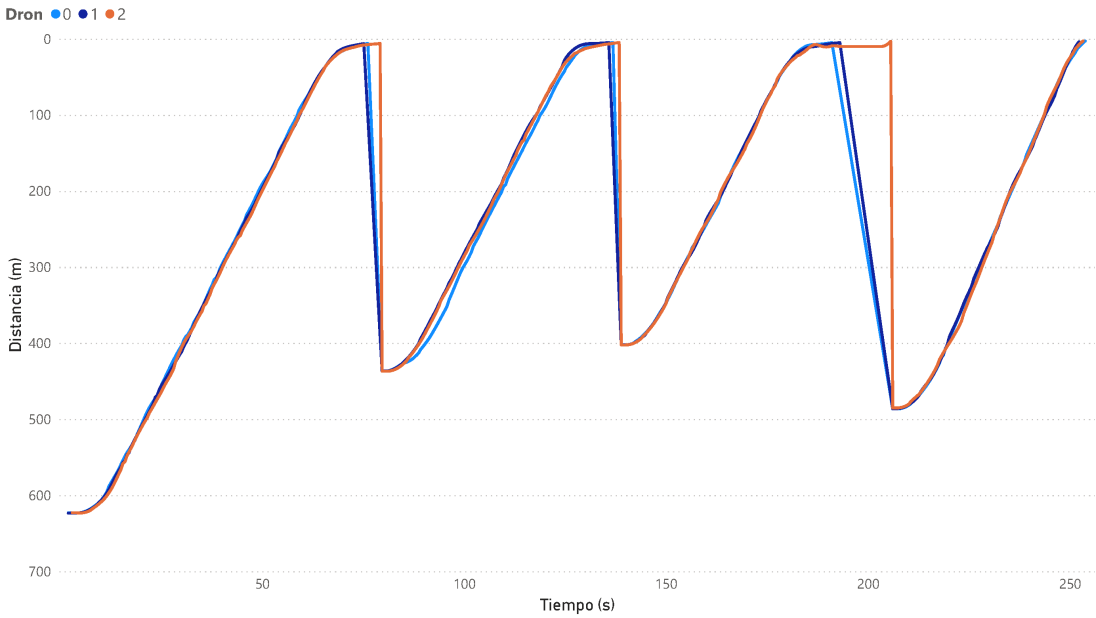

Figura 6.8: Vuelo UPV: Sincronización entre drones.

Gracias a la gráfica de velocidades (ver Figura 6.9) podemos comprobar tanto la velocidad media, como detectar posibles anomalías en el cálculo de los vectores de velocidad (aunque no se ha dado el caso). En este caso la velocidad media fue de 8 m/s, lo cual es un valor más que válido teniendo en cuenta las constantes subidas y bajadas en el trayecto (la velocidad base del dron son 10 m/s). No se han detectado anomalías en la velocidad más allá del tiempo de espera ya mencionado previamente.

Con la gráfica de la Figura 6.10 podemos comprobar el correcto funcionamiento del factor de ralentización aplicado cuando el dron se aproxima al `waypoint`, en concreto se aplica cuando la distancia al `waypoint` es menor a 7,5 veces la velocidad del dron (75 metros). Se trata de un gráfico con dos ejes Y, uno con la velocidad de forma lineal y otro con la distancia de forma logarítmica. De esta forma puede apreciarse cómo se aplica el factor de reducción una vez pasados los 75 metros. Se muestran los datos de los 3 drones para comprobar que todos lo cumplen.

Con todos los datos expuestos se considera válido el uso del protocolo en ciudades con gran volumen de edificios; Sin embargo, hasta que se obtengan MDS más precisos o se mejore el protocolo, no se recomienda su uso sobre concentraciones de personas en este contexto.

Se estima además un retraso del 6 % respecto al tiempo de vuelo usando el protocolo `mission` sin datos topográficos, lo cual supone una pérdida muy baja teniendo en cuenta la diferencia de datos y vuelo.

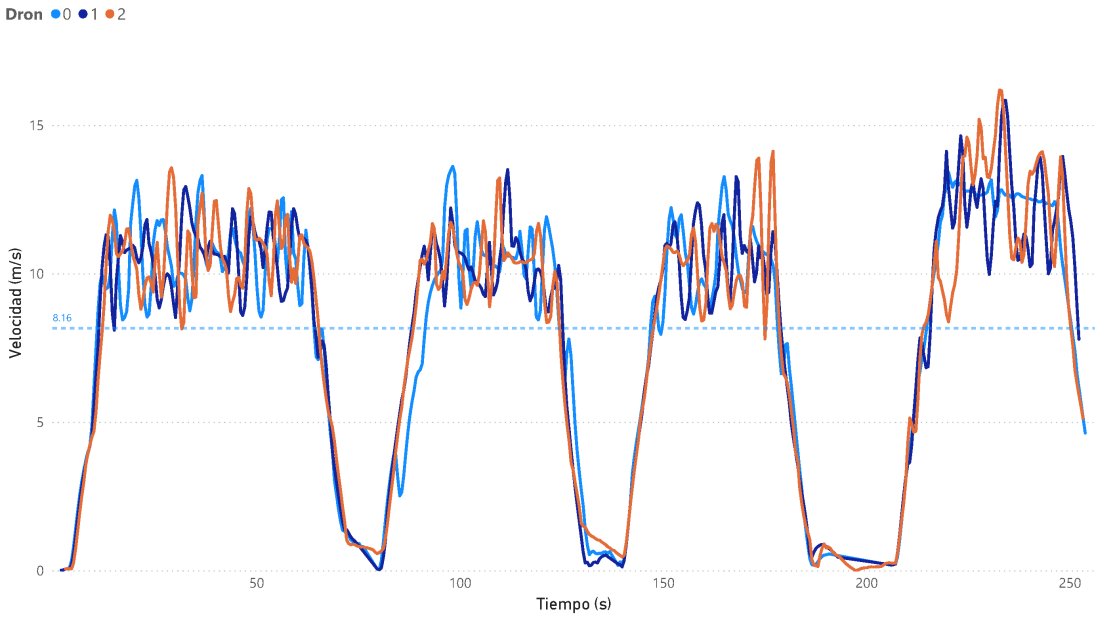

Figura 6.9: Vuelo UPV: Velocidad de los drones.

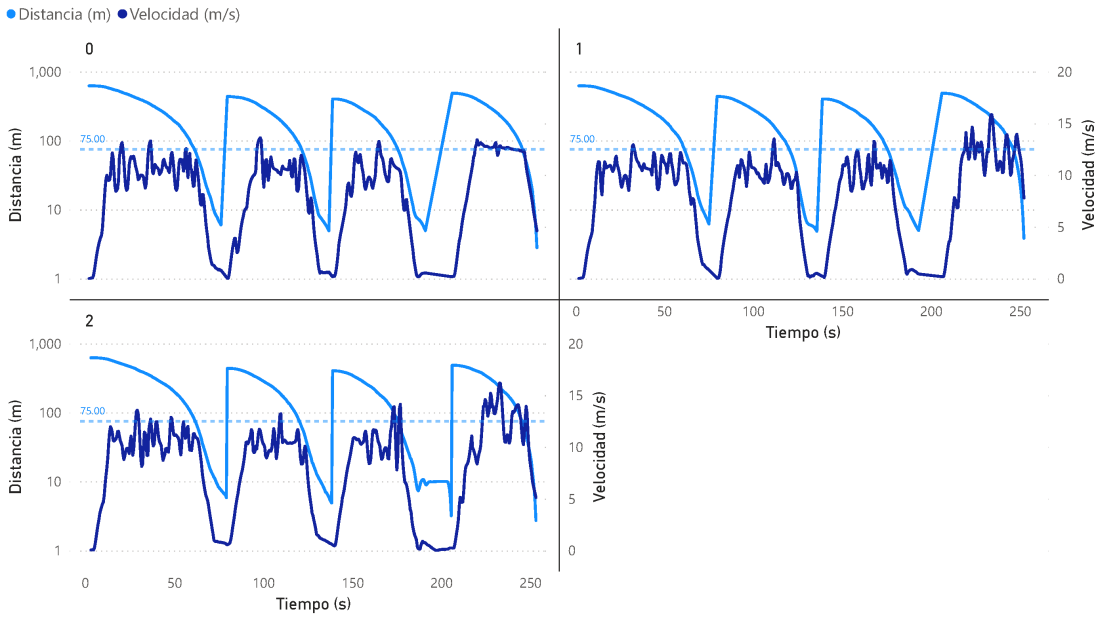

Figura 6.10: Vuelo UPV: Factor de ralentización para los 3 drones involucrados.

6.3 Acantilados de Barbate

Este vuelo ha supuesto uno de los mayores retos durante el desarrollo del protocolo, ya que pone a prueba los cálculos realizados al tener un desnivel de más 100 metros en una pendiente a 90º, teniendo que superarlo tanto en ascenso como descenso varias veces. La altura de los `waypoints` se ha situado a 10 metros sobre el suelo, y se cuenta con una dsitancia de previsión mayor, en este caso de 100 metros (Figura 6.11).

El objetivo de este vuelo es verificar el funcionamiento del sistema de prevención de colisión inminente, el cual detiene por completo el avance del dron cuando no es capaz de ascender y avanzar a la vez sin colisionar, teniendo que ascender primero y avanzar una vez se considere seguro. Por otro lado, fue este el entorno en el que se decidió aplicar la ralentización vertical, ya que la inercia de bajada tras un desnivel pronunciado y alto provocaba que el dron se "sumergiera".

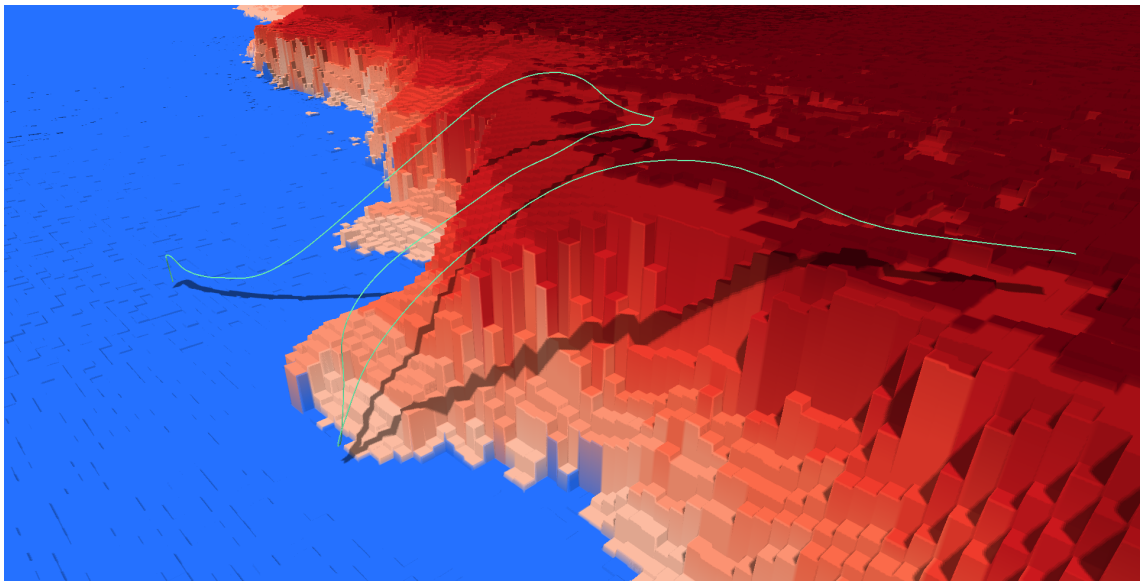

Figura 6.11: Renderizado 3D del vuelo sobre los Acantilados de Barbate (Dron 0).

Aunque resulta similar a otras gráficas presentadas, en este caso la gráfica de altitudes (ver Figura 6.12) se ha medido en base a la distancia recorrida, de esta forma puede apreciarse la "falta"de avance previamente a un desnivel excesivo, comprobando que se ejecuta de forma correcta el sistema de prevención mencionado.

En la gráfica de alturas relativas (ver Figura 6.13) puede apreciarse la necesidad de prevenir los desniveles con antelación, ya que sería imposible mantenerse sobre los 10 metros de altura sin hacerlo. Además, es destacable la precisión de cálculo del protocolo, que logra ajustar una diferencia de 100 metros quedando exactamente a 10 metros sobre la zona superior del acantilado una vez superado el desnivel.

En cuanto a la sincronización entre drones (Figura 6.14), ha resultado prácticamente perfecta, lo cual tampoco es de extrañar dado que los tres drones tenían un trayecto casi idéntico al tratarse de cortes limpios del terreno, sin diferencia de curvas como ocurría en la falda de la montaña (Sección 6.1) o distintos edificios como en la UPV (Sección 6.2).

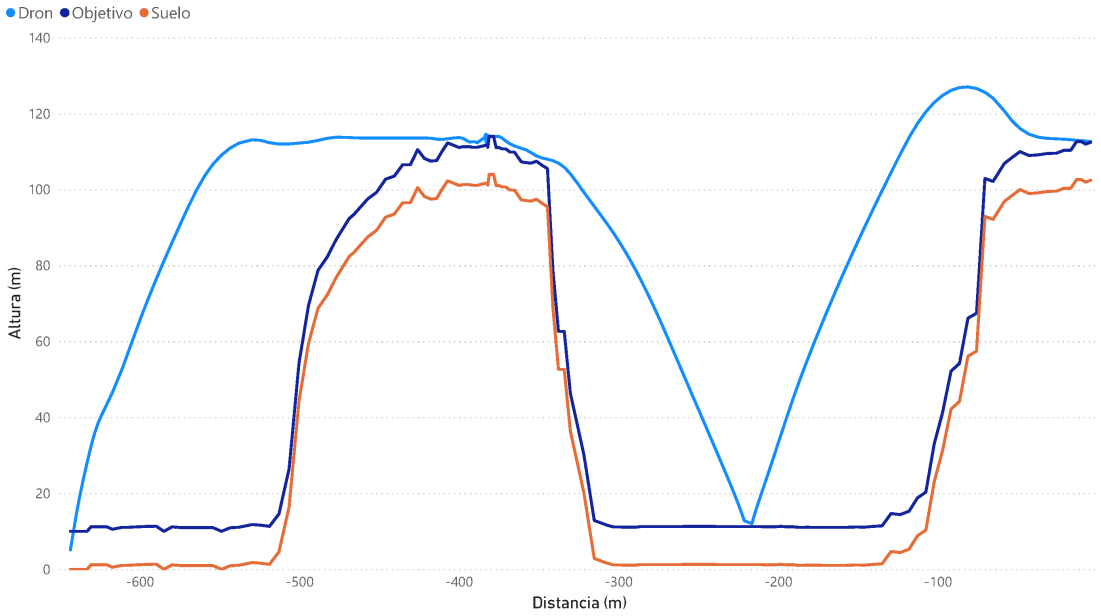

Figura 6.12: Vuelo en Acantilados: Altitudes (Dron 1)

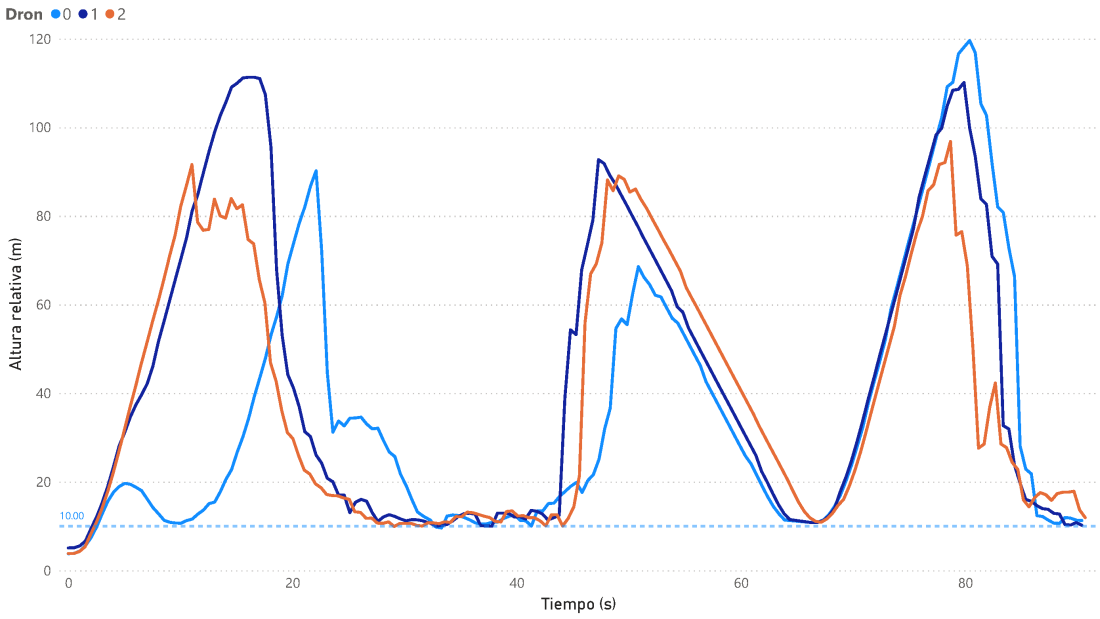

Figura 6.13: Vuelo en Acantilados: Alturas relativas de los 3 drones.

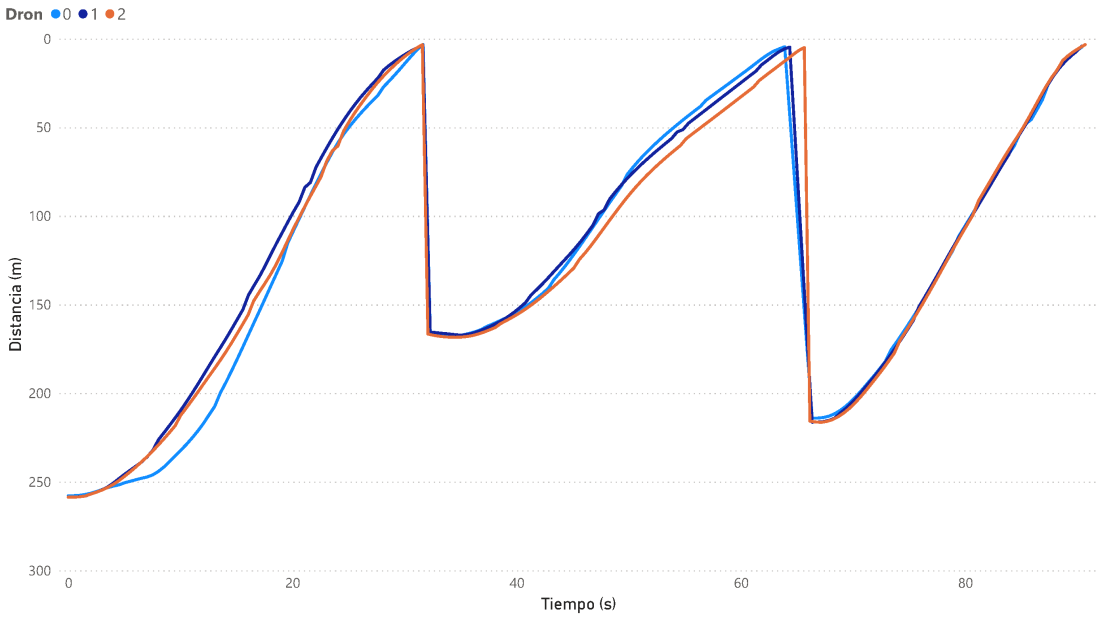

Figura 6.14: Vuelo en Acantilados: Sincronización entre drones.

Visualizando la gráfica de velocidades (ver Figura 6.15) resulta llamativa la velocidad media resultante (6,34 m/s), la cual es algo inferior a los casos anteriores. Sin embargo, este valor es fácilmente justificable dada la intervención del sistema de emergencia y el factor de ralentización en el descenso (Mencionado ya en la Sección 6.1) aplicado durante todo el descenso del acantilado, además, este último se ve acentuado cuando se suma el factor de ralentización por aproximación al `waypoint`.

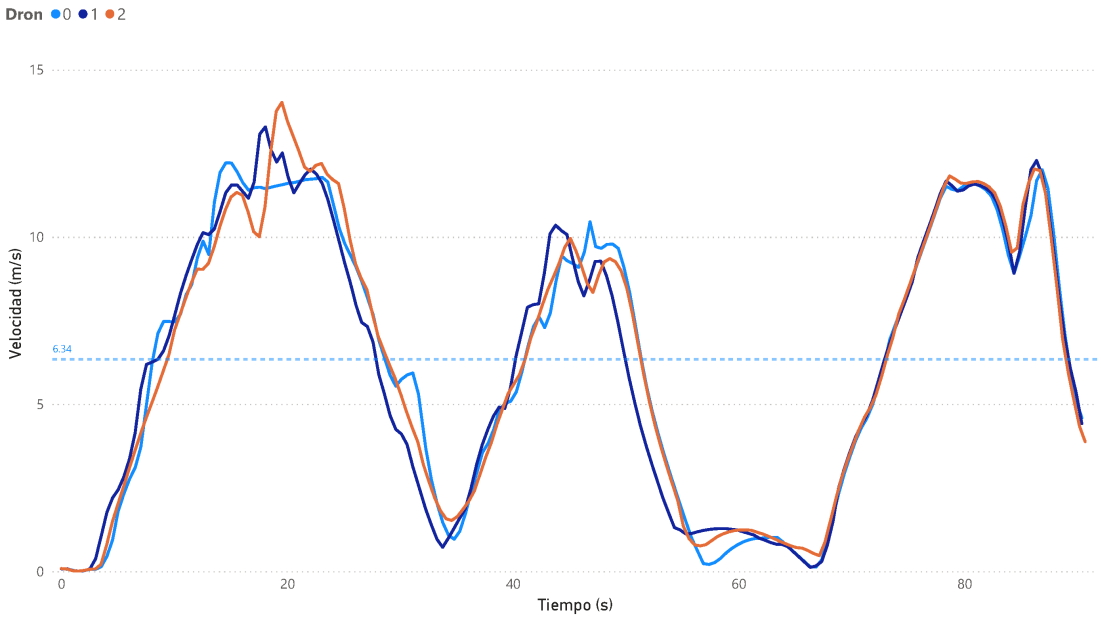

Figura 6.15: Vuelo en Acantilados: Velocidad de los drones.

Resulta destacable de la gráfica de ralentización (ver Figura 6.16) el cúmulo de factores de ralentización aplicados en la aproximación al `waypoint` 2, ya que se trata de una aproximación en un gran descenso, lo cual provoca un retraso general en la aproximación a este `waypoint`. Sin embargo, no se trata de un error sino de un factor de prevención para evitar la colisión contra el terreno, el mar en este caso, por el nivel de inercia, tal y como se comentaba al principio de la sección.

Dada la elevada complejidad de este vuelo, se considera todo un éxito cómo el protocolo actúa con conciencia situacional, previniendo cualquier colisión y superando los desniveles con precisión.

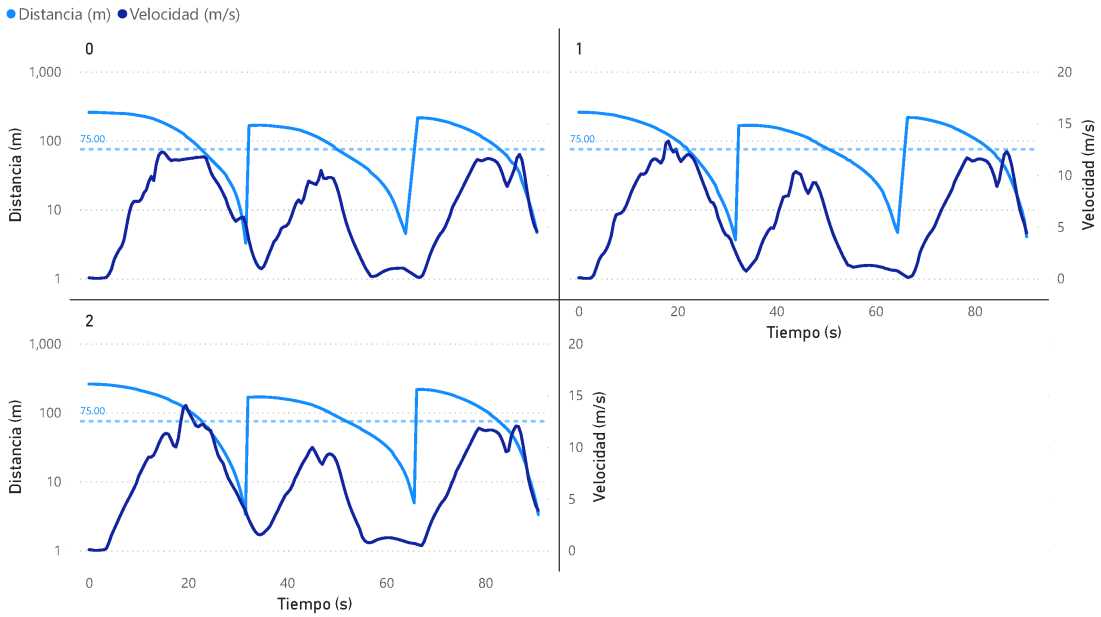

Figura 6.16: Vuelo en Acantilados: Factor de ralentización.

CAPÍTULO 7
Conclusiones

Tras haber finalizado el proyecto, ha quedado más que comprobado que se han completado los objetivos impuestos, se ha implementado con éxito la navegación tridimensional de drones de forma autónoma e individual y/o en enjambre, el simulador es ahora capaz de reproducir un vuelo y ser consciente de su entorno como si de un piloto se tratara, siendo capaz incluso de realizar vuelos en la vida real sin poner en riesgo a la población.

Se han detectado también mejoras del protocolo que podrán ser implementadas en el futuro como son reducir la precaución sin aumentar por ello el riesgo, obteniendo trazas de vuelo más precisas e igual de seguras.

A modo de reflexión, cuando se comenzó este proyecto, estaban claros los puntos en los que se podía mejorar, dónde y cómo actuar y qué resultados esperar, era casi una tarea pendiente del simulador. Sin embargo, después de todo el trabajo realizado y todo el conocimiento que se ha extraído, realmente es constatable lo lejos que está el desarrollo humano de alcanzar los límites del uso de drones en la vida cotidiana, ya que tras haber acabado el proyecto aún es posible seguir innovando con nuevas ideas y nuevos proyectos a futuro.

7.1 Trabajos futuros

A modo de cierre, se mencionan posibles mejoras o nuevos desarrollos que ofrecerían una mejor experiencia con el simulador.

1. Modificar los protocolos actuales utilizando ahora las herramientas de topografía proporcionadas y las alturas reales del dron sobre el terreno.
2. Implementar mapas tridimensionales con el renderizado de los MDEs y mostrar los trazos del vuelo en tiempo real.
3. Optimizar el protocolo actual aumentando su precisión de cálculo vectorial.

Se sugiere la implementación de los citados proyectos a futuro en el orden que se indica, por importancia y nivel de mejora.

Bibliografía

[1] K. Nonami. Drone Technology, Cutting-Edge Drone Business, and Future Prospects. *Journal of Robotics and Mechatronics*, 28(3):262-272, 2016.

[2] D. Vitali. *Los 14 usos de drones que seguro no conocías*. `https://agencia.donweb.com/los-14-usos-de-drones-que-seguro-no-conocias/` Última visita: 15-06-2021. Oct. de 2014.

[3] EASA. *Easy Access Rules for Unmanned Aircraft Systems*. Ene. de 2021.

[4] F. Fabra, C. T. Calafate, J. C. Cano y P. Manzoni. ArduSim: Accurate and real-time multicopter simulation. *Simulation Modelling Practice and Theory*, 87:170-190, 2018. ISSN: 1569-190X.

[5] L. T. Sze, W. G. Cheaw, Z. A. Ahmad, C. A. Ling, K. V. Chet, H. Lateh y L. Bayuaji. High resolution DEM generation using small drone for interferometry SAR. En: *2015 International Conference on Space Science and Communication (IconSpace)*. 2015, 366-369.

[6] U. R. Mogili y B. B. V. L. Deepak. Review on Application of Drone Systems in Precision Agriculture. *Procedia Computer Science*, 133:502-509, 2018. International Conference on Robotics and Smart Manufacturing (RoSMa2018). ISSN: 1877-0509.

[7] J.-P. Aurambout, K. Gkoumas y B. Ciuffo. Last mile delivery by drones: an estimation of viable market potential and access to citizens across European cities. *European Transport Research Review*, 11(1):1-21, 2019.

[8] T. Amukele, P. M. Ness, A. A. Tobian, J. Boyd y J. Street. Drone transportation of blood products. *Transfusion*, 57(3):582-588, 2017.

[9] D. E. Swihart, A. F. Barfield, E. M. Griffin, R. C. Lehmann, S. C. Whitcomb, B. Flynn, M. A. Skoog y K. E. Processor. Automatic Ground Collision Avoidance System design, integration, flight test. *IEEE Aerospace and Electronic Systems Magazine*, 26(5):4-11, 2011.

[10] E. M. Griffin, R. M. Turner, S. C. Whitcomb, D. E. Swihart, J. M. Bier, K. L. Hobbs y A. C. Burns. *Automatic ground collision avoidance system design for pre-Block 40 F-16 configurations*. Inf. téc. AIR FORCE RESEARCH LAB WRIGHT-PATTERSON AFB OH MATERIALS y MANUFACTURING ..., 2012.

[11] B. Eller, P. Stanfill, R. Turner, S. Whitcomb, D. Swihart, A. Burns y K. Hobbs. Test and Evaluation of a Modified F-16 Analog Flight Control Computer. En: *AIAA Infotech@Aerospace (I@A) Conference*. American Institute of Aeronautics y Astronautics, 2013, 4726.

[12] J. D. Carpenter. Simulation and piloted simulator study of an automatic ground collision avoidance system for performance limited aircraft. *Theses and Dissertations*, 2214, 2019.

[13] V. Kharchenko y N. Kuzmenko. Unmanned aerial vehicle collision avoidance using digital elevation model. *Proceedings of the National Aviation University*, 1:21-25, 2013.

[14] P. Sorokowski, M. Skoog, S. Burrows y S. Thomas. *Small UAV automatic ground collision avoidance system design considerations and flight test results*. Technical Memorandum. National Aeronautics y Space Administration (NASA), 2015.

[15] T. G. Farr, P. A. Rosen, E. Caro, R. Crippen, R. Duren, S. Hensley, M. Kobrick, M. Paller, E. Rodriguez, L. Roth, D. Seal, S. Shaffer, J. Shimada, J. Umland, M. Werner, M. Oskin, D. Burbank y D. Alsdorf. The Shuttle Radar Topography Mission. *Reviews of Geophysics*, 45(2), 2007.

[16] I. G. Nacional. *Plan Nacional de Ortofotografía Aérea*. Proyecto PNOA-LiDAR. Centro Nacional de Información Geográfica, 2009–2021.

[17] *World Geodetic System — 1984 (WGS-84) Manual*. 2.ª ed. International Civil Aviation Organization. Doc 9674 AN/946, 2002.

[18] I. A. Fernández-Coppel. La Proyección UTM. *Área de Ingeniería Cartográfica, Geodesia y Fotogrametría, Departamento de Ingeniería Agrícola y Forestal, Escuela Técnica Superior de Ingenierías Agrarias, Palencia, UNIVERSIDAD DE VALLADOLID*, 2001.

[19] C. Bruyninx, Z. Altamimi, E. Brockmann, A. Caporali, R. Dach, J. Dousa, R. Fernandes, M. Gianniou, H. Habrich, J. Ihde y col. Implementation of the ETRS89 in Europe: Current status and challenges. En: *REFAG 2014*. Ed. por T. van Dam. Cham: Springer International Publishing, 2017, 135-145. ISBN: 978-3-319-45629-4.

[20] F. Fabra, J. Wubben y C. T. Calafate. *Repositorio ArduSim*. `https://github.com/GRCDEV/ArduSim.git`.

[21] F. Fabra, W. Zamora, J. Sangüesa, C. T. Calafate, J.-C. Cano y P. Manzoni. A Distributed Approach for Collision Avoidance between Multirotor UAVs Following Planned Missions. *Sensors*, 19(10), 2019. ISSN: 1424-8220.

[22] F. Fabra, W. Zamora, P. Reyes, C. T. Calafate, J.-C. Cano, P. Manzoni y E. Hernandez-Orallo. An UAV Swarm Coordination Protocol Supporting Planned Missions. En: *2019 28th International Conference on Computer Communication and Networks (ICCCN)*. 2019, 1-9.

[23] F. Fabra, W. Zamora, J. Masanet, C. T. Calafate, J.-C. Cano y P. Manzoni. Automatic system supporting multicopter swarms with manual guidance. *Computers and Electrical Engineering*, 74:413-428, 2019. ISSN: 0045-7906.

[24] J. Wubben, F. Fabra, C. T. Calafate, T. Krzeszowski, J. M. Marquez-Barja, J.-C. Cano y P. Manzoni. Accurate Landing of Unmanned Aerial Vehicles Using Ground Pattern Recognition. *Electronics*, 8(12), 2019. ISSN: 2079-9292.

[25] J. Wubben, P. Aznar, F. Fabra, C. Calafate, J.-C. Cano y P. Manzoni. Toward secure, efficient, and seamless reconfiguration of UAV swarm formations. En: sep. de 2020.

[26] J. Taboada. *España es uno de los países con mayor geodiversidad de Europa y la mitad de su territorio está formado por sistemas montañosos*. `https://www.tysmagazine.com/espana-uno-los-paises-mayor-geodiversidad-europa-la-mitad-territorio-esta-formado-sistemas-montanosos/` Última visita: 18-06-2021. Feb. de 2017.

APÉNDICE A
Configuración del sistema

Se indica a continuación el resto de funcionamiento del protocolo.

A.1 Fase de inicialización

El método `setStartingLocation` indica cuál es la posición inicial de los drones y básicamente calcula en función de la formación elegida y el número de drones cual es su posición antes del despegue. Pudiendo ser la formación inicial distinta a la del vuelo si se desease.

Previamente al inicio del experimento, es necesario realizar la acción llamada `setup`, esto es indicar la posición de inicio del experimento, en este caso el protocolo enciende los drones y los eleva en formación a una posición vertical de 5 metros.

A.2 Identificación de dispositivos

Todas las pruebas se han realizado con tres drones (0, 1, 2) para comprobar que el simulador era capaz de trabajar con ellos a la vez y en enjambre, sin romper la formación y actuando de forma individual sobre cada uno.

APÉNDICE B
Código fuente

Se incluye a continuación el código del método `makeMission` el cual se ejecuta sobre un hilo por cada dron y se encarga del control topográfico explicado en este proyecto.

```
private void makeMission(int numUAV){

    //Initialize parameters
    Formation f = UAVParam.groundFormation.get();
    Copter copter = API.getCopter(numUAV);
    List<Waypoint> wps = copter.getMissionHelper().getMissionsLoaded()[0];
    copter.setFlightMode(FlightMode.GUIDED);
    int metric = dem.getMetric();
    double originX = dem.getOrigin().getX();
    double originY = dem.getOrigin().getY();
    int prevision = 50;
    int iterator = prevision/metric;
    File logFile = new File(API.getFileTools().getResourceFolder() + "/
        protocols/" + this.protocolString.toLowerCase() + "/log" + numUAV + ".
        csv");
    FileWriter xmlLog = null;
    try {
      logFile.createNewFile();
      xmlLog = new FileWriter(logFile.getAbsolutePath());
      xmlLog.write("UAV, ID, LAT, LONG, ALT, groundALT, desiredALT, TIME,
          distance, speed, wpID, wpX, wpY, wpZ\n");
    } catch (IOException e) {
      e.printStackTrace();
    }
    int lines=1;
    long startTime = System.currentTimeMillis();

    //Browse the waypoint list
    for(int index = 2; index < wps.size(); index++) {

      //index = 0 is somewhere in Africa and index = 1 is the starting location
      Waypoint wp = wps.get(index);
      API.getGUI((int)copter.getID()).logUAV("Moving to WP: " + (index-1));

      //A safe flight requires a minimum of 5 meters of altitude
      if(wp.getAltitude() < 5) { wp.setAltitude(5); }

      //Waypoint UTM coordinates
      Location2DUTM waypointUTM = f.get2DUTMLocation(wp.getUTM(),numUAV);
      double waypointX = waypointUTM.getX();
      double waypointY = waypointUTM.getY();
      double waypointZ = wp.getAltitude() + dem.getRealAltitude(waypointUTM);

```

```
41        //Parameters to decide when the UAV is close enough to the waypoint (at
              least 99% close or 5m)
42        double distance2D = Math.abs(waypointUTM.distance(copter.getLocationUTM()
              ));
43        double distance3D = Math.abs(new Location3DUTM(waypointUTM, waypointZ).
              distance3D(new Location3DUTM(copter.getLocationUTM(), copter.
              getAltitude())));
44        double precision = Math.max(distance3D * 0.01, 5);
45
46        //Looping until arriving to waypoint
47        while (distance3D > precision) {
48
49          distance2D = Math.abs(waypointUTM.distance(copter.getLocationUTM()));
50          distance3D = Math.abs(new Location3DUTM(waypointUTM, waypointZ).
                distance3D(new Location3DUTM(copter.getLocationUTM(), copter.
                getAltitude())));
51
52          //Copter UTM coordinates
53          Location2DUTM copterUTM = copter.getLocationUTM();
54          double copterX = copter.getLocationUTM().getX();
55          double copterY = copter.getLocationUTM().getY();
56
57          //Direction vector (X and Y axis)
58          double routeX = waypointX - copterX;
59          double routeY = waypointY - copterY;
60          double aux = Math.max(Math.abs(routeX), Math.abs(routeY));
61          routeX = routeX/aux;
62          routeY = routeY/aux;
63
64          //Slowing factor applied when getting close to destiny
65          double slowFactor = 1;
66          if(precision <= copter.getPlannedSpeed()*7.5 && distance2D <= copter.
                getPlannedSpeed()*7.5) { slowFactor = distance2D/(copter.
                getPlannedSpeed()*7.5); }
67
68          //Real ground altitude
69          double groundAlt = dem.getRealAltitude(copterUTM);
70
71          //Real altitude from the copter over the ground
72          double copterRealAlt = copter.getAltitude() - groundAlt;
73
74          //Prediction algorithm to detect future altitude movements
75          float maxFutureAlt = 0;
76          for(int i = 0; i <= iterator; i++) {
77            if(i*metric <= distance2D) {
78              double projectionX = ((copterX - originX) / metric) + routeX * i;
79              double projectionY = ((copterY - originY) / metric) + routeY * i;
80              float height = dem.getDem()[(int) projectionY][(int) projectionX];
81              if (height > maxFutureAlt) {
82                maxFutureAlt = height;
83              }
84            }
85          }
86
87          //Direction vector (Z axis)
88          boolean a = (wp.getAltitude() + maxFutureAlt) <= copter.getAltitude();
89          double routeZ = 0;
90          if(a) { routeZ = 0.5; } else { routeZ = -1; }
91
92
93          //Applying slowing factors (Approaching to desired position) and
                calculating speed vector
94          routeX = slowFactor * copter.getPlannedSpeed() * routeX;
95          routeY = slowFactor * copter.getPlannedSpeed() * routeY;
```

```
96          routeZ = Math.min(copter.getPlannedSpeed(), Math.abs(wp.getAltitude() +
                maxFutureAlt - copter.getAltitude())) * routeZ;
97
98          //Emergency altitude recovering due to imminent increase and possible
                collision
99          if(copterRealAlt < 9 || maxFutureAlt - copter.getAltitude() > prevision
                ) {
100           routeY = 0;
101           routeX = 0;
102           routeZ = -1 * copter.getPlannedSpeed();
103         }
104
105         //Movement order
106         copter.moveTo(routeY, routeX, routeZ);
107
108         //Write into log files
109         try {
110           xmlLog.write(numUAV + "," + lines + "," + copterX + "," + copterY + "
                ," + copter.getAltitude()
111               + "," + groundAlt + "," + (groundAlt + wp.getAltitude()) + "," +
                      (System.currentTimeMillis() - startTime)
112               + "," + distance3D + "," + copter.getSpeed() + "," + index + ","
                      + waypointX + "," + waypointY + "," + waypointZ + "\n");
113         } catch (IOException e) {
114           e.printStackTrace();
115         }
116         lines++;
117
118         //Refreshing rate
119         API.getArduSim().sleep((long) Math.min(1000 * metric/copter.
                getPlannedSpeed(), 500));
120       }
121
122       //Waypoint reached
123       copter.moveTo(0, 0, 0);
124
125       try {
126         barrier.await();
127       } catch (InterruptedException | BrokenBarrierException e) {
128         e.printStackTrace();
129       }
130     }
131
132     //Last waypoint reached
133     copter.moveTo(0, 0, 0);
134     try {
135       xmlLog.close();
136     } catch (IOException e) {
137       e.printStackTrace();
138     }
139   }
```