UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería Informática

Planificación dinámica de vuelo coordinado para la monitorización de la calidad del aire

Trabajo Fin de Grado

Grado en Ingeniería Informática

AUTOR/A: Palomares Solanes, Joan

Tutor/a: Tavares de Araujo Cesariny Calafate, Carlos Miguel

Cotutor/a: Wubben, Jamie

CURSO ACADÉMICO: 2023/2024

Resum

ArduSim és un simulador de vol en temps real dissenyat específicament per a desenvolupar i provar protocols de coordinació de vol per a multicópters, permetent l'execució de missions planificades o la formació d'eixams. Este simulador proporciona un sistema de navegació autònoma per a multicópters, la qual cosa els permet recopilar dades de contaminació en una àrea de manera eficient, autònoma i precisa.

En el context d'aquest simulador es va desenvolupar l'algorisme PdUC-D, un sistema de moviment discretitzat que facilita l'exploració d'una àrea determinada per l'usuari. Este algorisme determina els moviments dels drons en funció de les dades recol·lectades en temps real, proporcionant informació precisa sobre les fonts de contaminació. Aquest projecte amplia l'algorisme PdUC-D per a implementar un pla de vol coordinat que utilitza múltiples drons, accelerant així l'anàlisi de la zona delimitada.

Paraules clau: ArduSim, Drons, Contaminació, Coordinat, autónom

Resumen

ArduSim es un simulador de vuelo en tiempo real diseñado específicamente para desarrollar y probar protocolos de coordinación de vuelo para multicópteros, permitiendo la ejecución de misiones planificadas o la formación de enjambres. Este simulador proporciona un sistema de navegación autónoma para multicópteros, lo que les permite recopilar datos de contaminación en un área de manera eficiente, autónoma y precisa.

En el contexto de este simulador se desarrolló el algoritmo PdUC-D, un sistema de movimiento discretizado que facilita la exploración de un área determinada por el usuario. Este algoritmo determina los movimientos de los drones en función de los datos recolectados en tiempo real, proporcionando información precisa sobre las fuentes de contaminación. Este proyecto amplía el algoritmo PdUC-D para implementar un plan de vuelo coordinado que utiliza múltiples drones, acelerando así el análisis de la zona delimitada.

Palabras clave: ArduSim, Drones, Contaminación, Coordinado, autonomo

Abstract

ArduSim is a real-time flight simulator specifically designed to develop and test flight coordination protocols for multicopters, allowing the execution of planned missions or swarm formation. This simulator provides an autonomous navigation system for multicopters, allowing them to collect contamination data in an area efficiently, autonomously and accurately.

In the context of this simulator, the PdUC-D algorithm, a discretised motion system that facilitates the exploration of a user-determined area, was developed. This algorithm determines the movements of the drones based on the data collected in real time, providing accurate information on the sources of contamination. This project extends the PdUC-D algorithm to implement a coordinated

flight plan using multiple drones, thus speeding up the analysis of the delimited area.

Translated with DeepL.com (free version)

Key words: ArduSim, Drones, Pollution, Coordinated, autonomous

Índice general

Índice de figuras

Índice de tablas

CAPÍTULO 1
Introducción

1.1 Motivación

Actualmente, la preocupación por el medio ambiente, y su consecuencia mas directa, el cambio climático, ha aumentado exponencialmente. Teniendo en cuenta la situación global actual, es imperativo que nos centremos en reducir nuestra huella de carbono. Para ello, el primer paso es conocer los puntos de mayor contaminación para poder tomar decisiones con mayor precisión.

La idea detrás de este proyecto es proporcionar un sistema de medición que permita localizar los contaminantes facilmente, con vistas a utilizar esta información para una futura toma de decisiones. El uso de múltiples drones haría más fácil la obtención de información precisa poco tiempo (dependiendo del área que se desee analizar), permitiendo así mejorar la solución al problema. Por ejemplo, en un corto periodo de tiempo se podría analizar la contaminación de una zona amplia en una ciudad y posteriormente actuar en función de esa información.

La velocidad de recogida de datos ofrece la posibilidad de obtener información mas precisa, ya que permite observar como fluctuan las variaciones de los datos en menores periodos de tiempo.

1.2 Objectivos

Este proyecto tiene como objetivo mejorar el algoritmo de vuelo autónomo para drones, PdUC-D [1], para su uso coordinado en un grupo de drones. Estos drones utilizarán sensores de contaminación para recoger datos repetidamente durante su vuelo. Para conseguir un movimiento coordinado los drones tendrán que estar en constante comunicación y elegir su próximo movimiento a partir de la información recopilada por el propio dron y los otros drones. Este algoritmo debe guiar el vuelo de los drones y será probado en el simulador ArduSim [2].

Se pueden establecer los siguientes 6 objetivos:

1. Investigar el contexto actual en las áreas de contaminación, drones y tecnologías. Esto ayudará a comprender mejor como se alinean todas estas con el proyecto.

2. Comprender la estructura y funcionamiento del simulador ArduSim, y como integrar el código en el simulador, siguiendo su estructura de paquetes.
3. Comprender el funcionamiento del algortimo PdUC-D.
4. Identificar las funcionalidades necesarias para el desarrollo del proyecto.
5. Desarrollar la ampliación del algortimo PDuC-D para un vuelo coordinado.
6. Realizar las pruebas necesarias para comparar su eficiencia y calidad de los datos obtenidos

1.3 Estructura de la memoria

Este proyecto está compuesto por 8 capítulos principales:

- **Capítulo 1**. En este capítulo se ha presentado la motivación que ha llevado al desarrollo del proyecto y los objetivos a conseguir.
- **Capítulo 2**. Durante este capítulo se dará una visión del contexto actual, centrándose en los tipos de drones, las medidas de contaminación y los simuladores de vuelo.
- **Capítulo 3**. En este capítulo se llevará a cabo un análisis detallado del problema, definiendo los requisitos a cumplir y la interacción de los distintos actores con la aplicación.
- **Capítulo 4**. Se presentará el software utilizado en el desarrollo del proyecto. En el capítulo se explicará mas en profundimiento el simulador ArduSim y el algoritmo predecesor al desarrollado en el proyecto.
- **Capítulo 5**. Durante este capítulo se explicará el desarrollo del algoritmo, analizando su código y explicando los puntos mas importantes.
- **Capítulo 6**. Durante este capítulo se analizarán los resultados obtenidos después de realizar distintos tipos de pruebas.
- **Capítulo 7**. El último capítulo. En él se expondrán las conclusiones finales del proyecto y posibles mejoras a implementar en un futuro.

Al final de la memoria se encuentra la bibliografía, con todas las fuentes de información, y el anexo de objetivos de desarrollo sostenible (ODS).

CAPÍTULO 2
Estado del arte

2.1 Vehículos aéreos no tripulados (UAV)

Los vehículos aéreos no tripulados, comúnmente conocidos como drones o UAVs (del inglés Unmanned Aerial Vehicles), son dispositivos voladores sin piloto a bordo. Estos drones pueden ser controlados de manera remota o realizar vuelos programados de forma autónoma.

2.1.1. Clasificación de drones

Las aeronaves no tripuladas deben cumplir con ciertos requisitos para ser clasificadas en una u otra categoría. Estos requisitos están relacionados principalmente con la Masa Máxima al Despegue (MTOW) del dron, su velocidad máxima, la altura máxima que puede alcanzar y, de sus características de identificación y seguridad. Según la nueva normativa europea, EASA divide a los drones en las siguientes clases[4]:

- **Drones de clase C0**: Esta clase de drones debe cumplir las siguientes características:
  - Tener una MTOW inferior a 250 g.
  - Velocidad máxima en vuelo de 19 m/s.
  - Tener limitada la altura máxima desde el punto de despegue a 120 m.
  - Estar alimentado con electricidad.
- **Drones de clase C1**: Esta clase de drones debe cumplir las siguientes características:
  - Las características anteriores, ampliando el MTOW hasta 900 g.
  - Disponer de un número de serie único.
  - Tener un sistema de identificación a distancia directa y de identificación a distancia de red.
  - Estar equipado con un sistema de geoconsciencia.

- **Drones de clase C2**: Esta clase de drones debe cumplir las siguientes características:
  - Tener una MTOW inferior a 4 kg.
  - Tener limitada la altura máxima desde el punto de despegue a 120 m.
  - Estar alimentado con electricidad.
  - Las mismas características de identificación y seguridad que C1.
  - Estar equipado con un enlace de datos protegido contra el acceso no autorizado a las funciones de mando y control (C2).
  - Salvo si es una aeronave de ala fija, estar equipado con un modo de baja velocidad seleccionable que limite la velocidad a 3 m/s como máximo.
  - Equipar luces para control de actitud y vuelo nocturno.
- **Drones de clase C3**: Esta clase de drones debe cumplir las siguientes características:
  - MTOW inferior a 25 kg y una envergadura inferior a 3 m.
  - Tener limitada la altura máxima desde el punto de despegue a 120 m.
  - Estar alimentado con electricidad.
  - Las mismas características de identificación y seguridad que C1.
  - Estar equipado con un sistema de aviso de batería baja para el dron y la estación de control (CS).
- **Drones de clase C4**: Esta clase de drones debe cumplir las siguientes características:
  - Tener una MTOW inferior a 25 kg, incluida la carga útil.
  - No disponer de modos de control automático, excepto para la asistencia a la estabilización del vuelo y para la asistencia en caso de pérdida del enlace.
  - Estar destinadas para la práctica del aeromodelismo.
- **Drones de clase C5**: Esta clase de drones debe cumplir las siguientes características:
  - MTOW inferior a 25 kg.
  - No ser una aeronave de ala fija, salvo si es una UA cautiva.
  - Disponer de un sistema que proporcione al piloto a distancia información clara y concisa sobre la altura del dron.
  - Estar equipado con un modo de baja velocidad seleccionable que limite la velocidad a 5 m/s como máximo.
  - Ante una pérdida de enlace de datos (C2), o en caso de fallo, contar con un método de recuperarlo o de terminar el vuelo de forma segura.
  - Las mismas características de identificación y seguridad que C3.

  - Si el dron dispone de función de limitación de acceso a determinadas zonas o volúmenes del espacio aéreo, esta deberá ser interoperable con el sistema de control del vuelo, y deberá informar al piloto a distancia cuando esta impida entrar a la UA a estas zonas o volúmenes del espacio aéreo.
  - Un dron de clase C5 podrá consistir en una aeronave no tripulada de clase C3 que lleve instalado un kit de accesorios que lo convierta en clase C5.

- **Drones de clase C6**: Esta clase de drones debe cumplir las siguientes características:
  - Tener una MTOW inferior a 25 kg.
  - Velocidad máxima respecto al suelo en vuelo horizontal de 50 m/s.
  - sistemas avanzados de control y seguridad para evitar exceder límites operacionales, similares a los de clase C5.

2.2 Medidas de contaminación

La contaminación se encuentra presente alrededor de todo el planeta, así como en los diversos estados de la materia. Para este proyecto nos centraremos en la contaminación atmosférica, es decir, la que se encuentra en el aire.

La contaminación atmosférica puede definirse como la presencia que existe en el aire de pequeñas partículas o productos secundarios gaseosos que pueden implicar riesgo, daño o molestia para las personas, plantas y animales que se encuentran expuestas a dicho ambiente[5].

La contaminación atmosférica es producida principalmente por los procesos industriales en donde se realiza combustión y por fuentes móviles tales como los automóviles.

2.2.1. Métodos de medición de la contaminación atmosférica

Estas son algunas de las técnicas para medir la calidad del aire [6]:

- **Muestreo Pasivo.**: Se encarga de recolectar un contaminante específico por medio de su adsorción y/o absorción en un sustrato químico.
- **Muestreo con Bioindicadores**: Es aquel que utiliza especies vivas generalmente vegetales, como árboles y plantas, donde su superficie recepta contaminantes.
- **Muestreo activo**: Este método utiliza energía eléctrica para succionar el aire a muestrear con la ayuda de un colector físico o químico.
- **Método óptico de percepción remota**: Se basan en técnicas espectroscópicas. Transmiten un haz de luz de una cierta longitud de onda a la atmósfera y miden la energía absorbida.

2.2.2. Otros proyectos que han calculado mediciones de contaminación

Algunos ejemplos de otros proyectos realizados con el objetivo de medir la contaminación ambiental son:

- **TEMPO (Tropospheric Emissions: Monitoring of Pollution)** [7]: Se trata de un instrumento que se encuentra a bordo de un satélite, que ya está en órbita, y promete ser el primer registro continuo de la contaminación atmosférica del país. El dispositivo observa la luz que se refleja en la superficie, la atmósfera y las nubes de la Tierra detectando los contaminantes atmosféricos como el ozono y el dióxido de nitrógeno al escanear América del Norte durante las horas diurnas.
- **Cycling with Clean Air** [8]: Este proyecto busca que los ciudadanos puedan obtener mediciones de la calidad del aire mientras se realizan trayectos en bicicleta. ConBici está utilizando monitores portátiles de partículas $PM_{2,5}$ para recopilar datos en zonas frecuentadas por ciclistas y ciudadanos en general.
- **Servicios de Infraestructuras de Investigación para el Refuerzo de las Capacidades de Vigilancia de la Calidad del Aire en las Zonas Urbanas e Industriales Europeas (RI-URBANS)** [9]: El proyecto RI-URBANS, financiado por la UE, buscar proporcionar mejores observaciones de la calidad del aire para la evaluación avanzada de las políticas de calidad del aire mediante el desarrollo de sinergias entre las redes de vigilancia de la calidad del aire (AQMN) y las RI en el ámbito atmosférico.

2.3 Simuladores de vuelo

Aunque los drones son cada vez mas accesibles, estos presentan varios inconvenientes:

- **Las leyes del país en el que se quiere volar** [10]: El aumento en el uso de los drones ha traído consigo una dura legislación al respecto. No resulta tan sencillo como comprarse un dron y hacerlo volar, sino que es obligatorio tener la habilitación correspondiente como piloto de drones para uso profesional, entre otros muchos requisitos.
- **Las condiciones meteorológicas**: Aunque la tecnología de los UAV esté en mejora continua y cada vez sean capaces de afrontar más situaciones adversas, una mala meteorología supone siempre un problema para el vuelo de estos vehículos. Además, puede entorpecer algunos de sus usos al afectar a la medición de sus sensores.
- **La batería** Es un hecho que la batería de cualquier dispositivo siempre va a ser limitada. Es por esto que el uso de drones físicos siempre va a tener una limitación en cuanto al tiempo de uso y tiempo de experimentación, que irá ligada a la duración de la batería del mismo.

- **Experimentación con varios drones simultáneamente** Trabajar con un dron individualmente puede ser sencillo, pero cuando se usa un enjambre de drones la tarea se vuelve más complicada. La dificultad de controlar el correcto funcionamento de todos los drones y la posibilidad de que se produzcan percances entre los drones, en caso de que haya algún error, pueden provocar retrasos en el proceso de desarrollo.

El uso de un simulador de vuelo permite evitar estos inconvenientes, siendo muy útiles para realizar estudios.

CAPÍTULO 3
Análisis y especificación de requisitos

En este capítulo se llevará a cabo un análisis detallado del problema para facilitar el desarrollo del proyecto. Aquí se definirán teóricamente los requisitos que deberá cumplir la aplicación y se propondrá la mejor manera de desarrollarla.

3.1 Identificación de actores

En un sistema, los actores son todas aquellas entidades que interactúan con él, ya sea una persona, un dispositivo o una actividad con un rol específico. Para este proyecto, se identifican los siguientes actores clave:

- **Dron**: Dispositivo volador (multicóptero) responsable de realizar el vuelo para recoger datos de contaminación.
- **Usuario**: Persona encargada de definir las variables necesarias para la ejecución del sistema, como la ubicación y el tamaño del área a medir. También supervisa el proceso de vuelo y analiza los resultados obtenidos.
- **ArduSim**: Software que emite las órdenes al dron y establece su trayectoria. Funciona como intermediario entre el protocolo implementado y el dron.

3.2 Definición de requisitos

En esta sección se establecerán los requisitos necesarios para la aplicación, dividiéndolos en dos categorías: funcionales y no funcionales.

3.2.1. Requisitos funcionales

Los requisitos funcionales especifican las funciones y capacidades que debe tener cada actor en el sistema para cumplir con sus objetivos.

- **Dron**

1. **Medir la contaminación. (D-01)**
   Utilizando su sensor, el dron mide la contaminación en un punto específico y envía los resultados a ArduSim.
2. **Recibir nueva orden. (D-02)**
   El dron recibe instrucciones para desplazarse a un nuevo punto.
3. **Desplazarse al destino. (D-03)**
   El dron se dirige al nuevo punto de destino y realiza una nueva medición de contaminación.
4. **Enviar mensaje. (D-04)**
   El UAV envia un mensaje a otro dron.
5. **Recibir mensaje. (D-05)**
   El UAV recibe un mensaje de otro dron.

- **Usuario**

1. **Configurar los parámetros de simulación. (U-01)**
   El usuario establece los parámetros, como la ubicación y el área a medir, y configura el inicio del vuelo.
2. **Iniciar la simulación. (U-02)**
   Tras la configuración de los parámetros, el usuario procede a iniciar la simulación.
3. **Recibir los resultados. (U-03)**
   Una vez concluida la simulación, el usuario recibe y analiza los datos recopilados.

- **ArduSim**

1. **Cálcular el próximo destino del dron. (A-01)**
   ArduSim analiza los datos disponibles y calcula la próxima ubicación a la que el dron debe desplazarse.
2. **Almacenar de los datos de simulación. (A-02)**
   ArduSim guarda los resultados obtenidos de manera organizada y comprensible para el usuario.

3.2.2. Requisitos no funcionales

Los requisitos no funcionales son aquellos que describen características generales de la aplicación, sin relacionarse directamente con sus funciones.

1. **Portabilidad**
   El protocolo debe poder ejecutarse en múltiples plataformas, ya sea en un dron real o en uno simulado.

2. **Mantenibilidad**
   Debe implementarse un sistema de control de versiones para garantizar un desarrollo ordenado y una recuperación efectiva ante cualquier pérdida de datos.

3. **Eficiencia**
   El sistema debe ser capaz de obtener datos detallados de contaminación sin requerir tiempos de ejecución excesivos.

4. **Usabilidad**
   La interfaz de usuario debe ser intuitiva, permitiendo que el sistema realice operaciones complejas con poca intervención del usuario.

5. **Integración**
   El protocolo debe estar completamente integrado con el simulador ArduSim, de manera que funcione como una extensión natural del mismo.

6. **Escalabilidad**
   El sistema debe ser capaz de operar en áreas de diversos tamaños, manteniendo la precisión en la recopilación de datos sin importar la extensión del área.

3.3 Diagramas de uso

3.3.1. Diagrama UML

El diagrama UML 3.1 permite visualizar todas las clases presentes en el sistema y como se relacionan entre ellas.

3.3.2. Diagrama de casos de uso

En la figura 3.2 se muestra las distintas acciones que se pueden realizar, junto con su actor, y las relaciones entre estas, en caso de que las haya.

En las siguientes tablas se puede encontrar una explicación mas detallada de cada una acciones en el sistema.

Caso de uso	Medir la contaminación. (D-01)
Actores	Dron.
Resumen	El dron mide la contaminación en el punto donde se encuentra.
Precondiciones	El dron se encuentra en un punto, que no haya sido visitado, dentro del área.
Postcondiciones	Se guarda el valor de la medición en el sistema y el punto se marca como visitado.
Incluye	-
Extiende	-

Tabla 3.1: Medir la contaminación. (D-01)

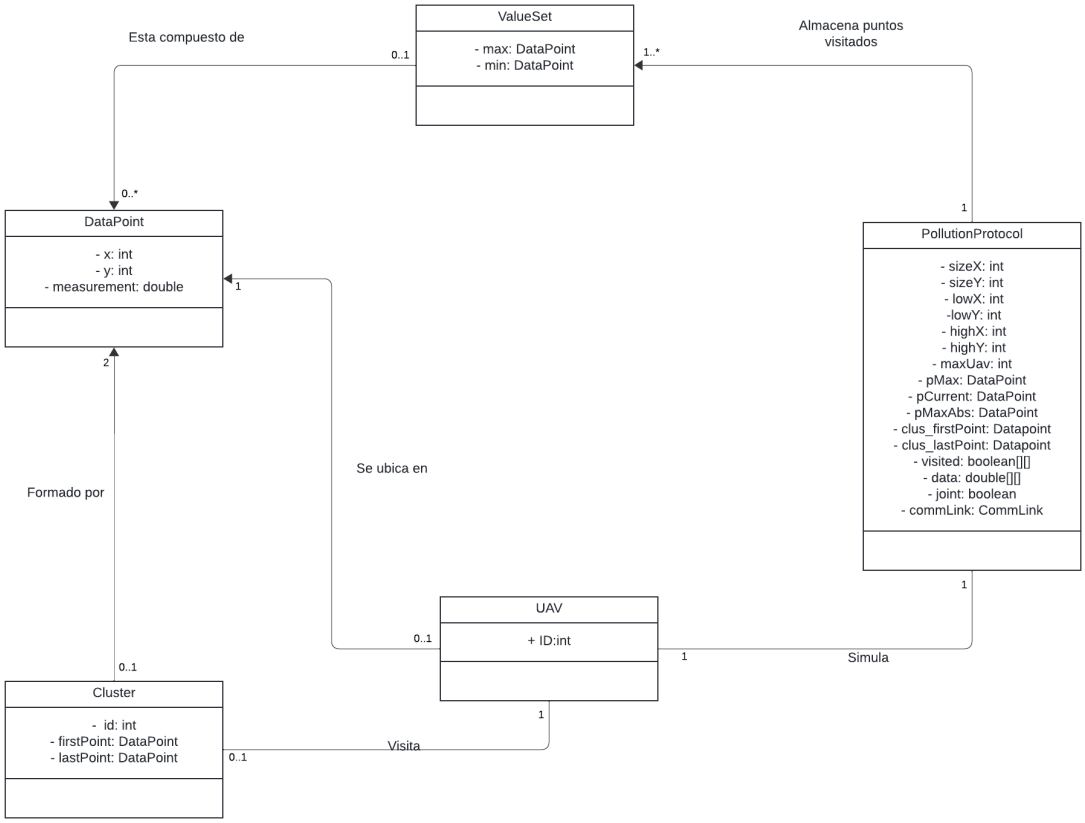

Figura 3.1: Diagrama UML del sistema

Caso de uso	Recibir nueva órden. (D-02)
Actores	Dron.
Resumen	El dron recibe una nueva órden de ArduSim.
Precondiciones	ArduSim ha calculado el próximo punto de destino y la ha enviado.
Postcondiciones	El dron obtiene el siguiente punto al que moverse.
Incluye	Cálcular el próximo destino del dron. (A-01)
Extiende	Desplazarse al destino. (D-03)

Tabla 3.2: Recibir nueva órden. (D-02)

Caso de uso	Desplazarse al destino. (D-03)
Actores	Dron.
Resumen	El dron se mueve hacia el punto indicado.
Precondiciones	El punto está dentro del área del dron y no ha sido visitado anteriormente.
Postcondiciones	El dron se posicionará en el punto indicado.
Incluye	-
Extiende	-

Tabla 3.3: Desplazarse al destino. (D-03)

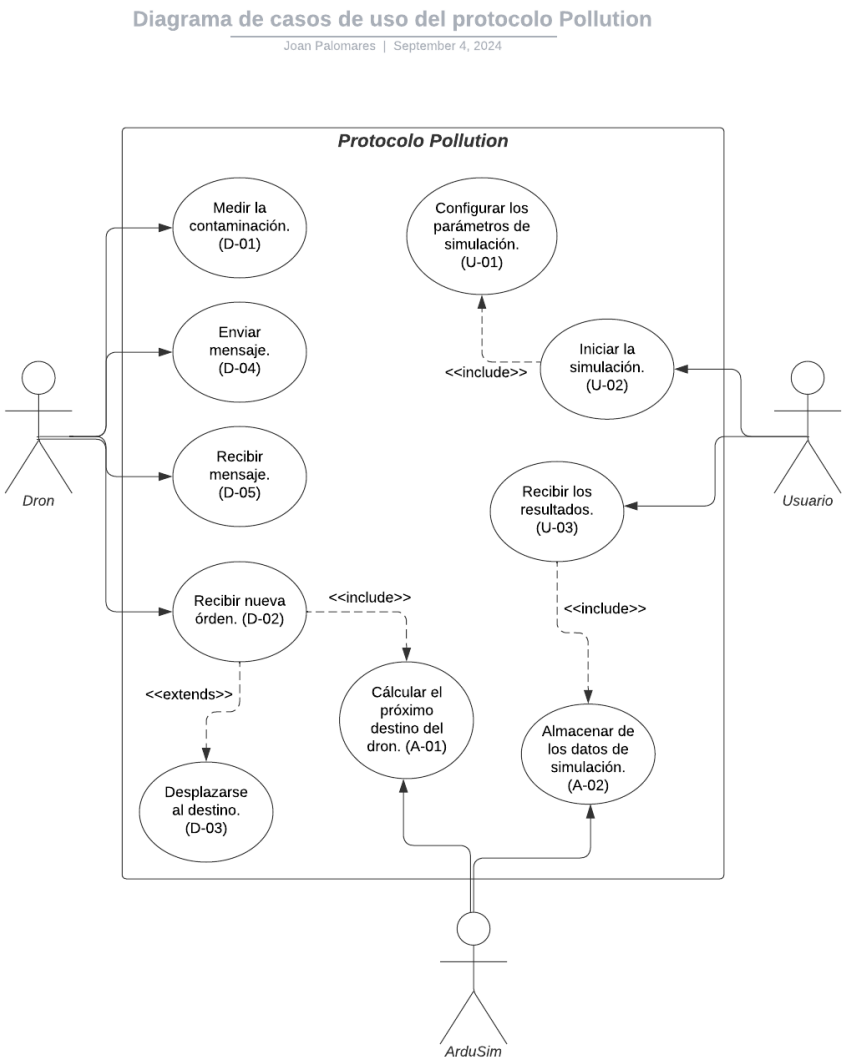

Figura 3.2: Diagrama de casos de uso del protocolo *Pollution*

Caso de uso	Enviar mensaje. (D-04)
Actores	Dron.
Resumen	El dron envía un mensaje a los otros drones.
Precondiciones	El dron puede enviar mensajes UDP mediante WIFI
Postcondiciones	Los otros drones conocerán el movimiento del dron que ha mandado el mensaje.
Incluye	-
Extiende	-

Tabla 3.4: Enviar mensaje. (D-04)

Caso de uso	Recibir mensaje. (D-05)
Actores	Dron.
Resumen	El dron recibe un mensaje de los otros drones.
Precondiciones	El dron puede recibir mensajes UDP mediante WIFI
Postcondiciones	El dron conocerá los movimientos de los otros drones.
Incluye	-
Extiende	-

Tabla 3.5: Caso de uso D-05.

Caso de uso	Cálcular el próximo destino del dron. (A-01)
Actores	ArduSim.
Resumen	ArduSim calcula el próximo punto del dron.
Precondiciones	El dron ha medido correctamente la contaminación en el punto actual.
Postcondiciones	El dron recibe su nueva orden.
Incluye	-
Extiende	-

Tabla 3.6: Cálcular el próximo destino del dron. (A-01)

Caso de uso	Almacenar de los datos de simulación. (A-02)
Actores	ArduSim.
Resumen	ArduSim almacena los datos tomados por el dron y los guarda de forma comprensible para el usuario.
Precondiciones	La simulación ha terminado correctamente y ArduSim ha recibido todos los datos necesarios.
Postcondiciones	El usuario obtiene los datos finales.
Incluye	-
Extiende	-

Tabla 3.7: Almacenar de los datos de simulación. (A-02)

Caso de uso	Configurar los parámetros de simulación. (U-01)
Actores	Usuario.
Resumen	El usuario decide qué parámetros debe tener la simulación y los introduce en el sistema.
Precondiciones	El usuario tiene una versión funcional de ArduSim
Postcondiciones	El sistema muestra una interfaz preparada para iniciar la simulación.
Incluye	-
Extiende	-

Tabla 3.8: Configurar los parámetros de simulación. (U-01)

Caso de uso	Iniciar la simulación. (U-02)
Actores	Usuario.
Resumen	El usuario inicia la simulación del protocolo.
Precondiciones	El usuario ha incluido correctamente los parámetros de la simulación.
Postcondiciones	La interfaz muesta la ejecución de la simulación.
Incluye	Configurar los parámetros de simulación. (U-01)
Extiende	-

Tabla 3.9: Iniciar la simulación. (U-02)

Caso de uso	Recibir los resultados. (U-03)
Actores	Usuario.
Resumen	El usuario recibe los resultados de la simulación.
Precondiciones	ArduSim finaliza correctamente la simulación y envía los datos.
Postcondiciones	El usuario analiza los datos.
Incluye	Almacenar de los datos de simulación. (A-02).
Extiende	-

Tabla 3.10: Recibir los resultados. (U-03)

CAPÍTULO 4
Software utilizado

4.1 ArduSim

ArduSim es un innovador simulador de vuelo en tiempo real, diseñado para el desarrollo de protocolos de coordinación de vuelo para multicópteros, ya sea para realizar misiones planificadas o para operar en formación de enjambre. Fue desarrollado por Francisco Fabra, en aquel momento 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 Universitat Politècnica de València, bajo la supervisión de Carlos T. Calafate.

ArduSim fue desarrollado para abordar la necesidad de simular múltiples drones simultáneamente. El simulador, permite a los investigadores investigar fácilmente el comportamiento de los enjambres de UAV, y proporcionar interacciones de red entre los UAV.

4.1.1. Estructura y funcionamiento

ArduSim está implementado en Java y se caracteriza por su estructura modular. Está basado en el simulador SITL (Software In The Loop) [11], que permite simular diferentes tipos de UAV con alta precisión. Sin embargo, SITL solo soporta la simulación de un único dron a la vez. Para superar esta limitación, ArduSim actúa como una capa superior que gestiona múltiples instancias de SITL, ejecutando una instancia por cada UAV en procesos independientes, lo que posibilita la simulación de enjambres de drones.

Cada UAV virtual esta compuesto por un agente ArduSim. Cada agente a su vez incluye una instancia SITL, y un hilo (Controller) encargado de enviar comandos al multicóptero, y de recibir la información que éste genera. Es decir, es el controlador el que se comunica con la lógica del protocolo que estamos ejecutando.

ArduSim incluye la simulación de emisión de paquetes entre UAVs (Simulated broadcast), y la detección de posibles colisiones (UAV Collision detector). El simulador maneja las comunicaciones entre UAVs (drone-to-drone o D2D) utilizando tecnología WiFi, conforme al estándar IEEE 802.11a (banda de 5 GHz). Esta

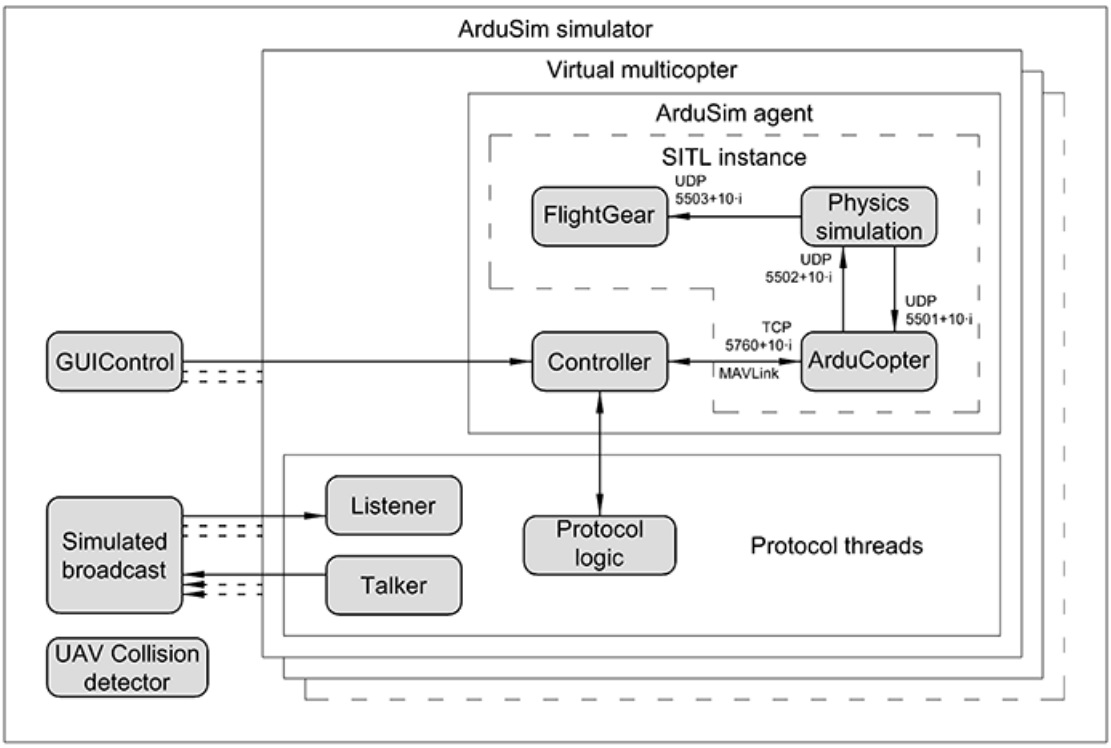

Figura 4.1: Estructura interna del simulador ArduSim.

comunicación se configura a través de los protocolos, que requieren al menos dos hilos: uno para el envío de paquetes (Talker) y otro para la recepción (Listener).

ArduSim con UAVs reales

El simulador ArduSim ha sido diseñado para facilitar el despliegue de los protocolos implementados en UAV reales usando una Raspberry Pi conectada a un dron.

Cuando se ejecuta ArduSim en una Raspberry Pi, todos los elementos de software dependientes de la simulación se desactivan simplemente cambiando un parámetro de ejecución, lo que hace que el despliegue de un protocolo recién desarrollado sea algo trivial. El controlador seguiría siendo el encargado de la comunicación con el protocolo.

En la Figura 4.1 se ilustra la estructura del simulador cuando se emplean drones reales. En este contexto, se introduce el concepto de PC Companion. Este PC externo actúa como receptor de los mensajes y registros enviados por el UAV. Sin embargo, el funcionamiento detallado del PC Companion no será abordado en este proyecto, ya que está fuera del alcance del mismo.

4.1.2. Estructura de paquetes

El proyecto está organizado en paquetes y sigue la Disposición estándar de directorios de Maven [12].

- `ArduSim`: Carpeta del proyecto.

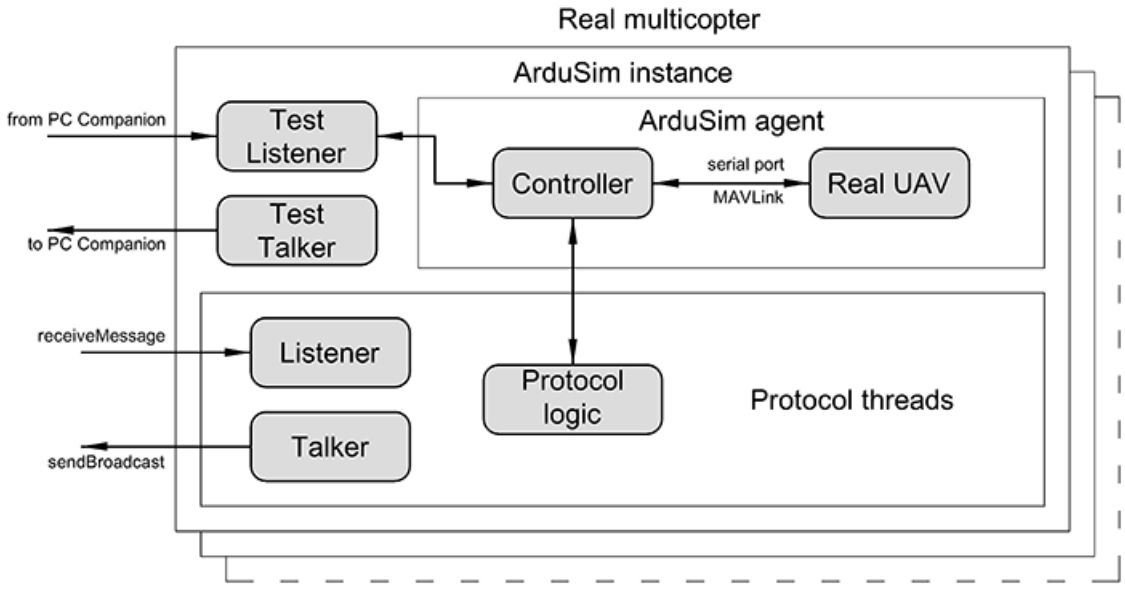

Figura 4.2: Estructura interna de ArduSim con drones reales.

- `Help`: Archivos de documentación.
- `local-maven-repo`: Incluye un archivo .jar con funcionalidad básica para ArduSim, como dibujar en la pantalla, etc.
- `src`: Todos los archivos fuente.
  - `main`
    - ◇ `Java`: todo el código Java.
    - ◇ `resources`: Archivos como .fxml y archivos de parámetros.
  - `test`: sigue la misma estructura que la carpeta Java pero incluye solo pruebas (JUnit).
- `target`: incluye archivos importantes y archivos .class de Java después de la compilación.

El código fuente puede ser dividido en:

- `ArduSim.src.main.java.com.`: Código fuente.
  - `api`: Contiene todas las cosas comunes que pueden usarse para crear un nuevo protocolo.
  - `protocols`: Contiene carpetas para cada protocolo individual. Es importante que cada procotol permanezca completamente aislado de los otros protocolos. De esta forma, se asegura que cada protocolo sea completamente independiente y que no sea necesario disponer de otros protocolos instalados.
  - `setup`: Contiene todo el código de ArduSim
  - `uavController`: Dentro de este paquete existe código para permitir que ArduSim se comunique con las diversas instancias de ArduCopter.

4.1.3. Implementación de protocolos

En ArduSim, los protocolos definen el comportamiento que deben seguir los drones mediante bloques de código específicos. Como se ha mencionado previamente, estos protocolos se diseñan de manera modular, lo que permite que sean completamente independientes del código central del simulador. Esta independencia facilita la actualización de ArduSim a nuevas versiones sin afectar a los protocolos existentes.

Dentro del proyecto Java, se distinguen dos rutas principales para almacenar los datos relacionados con cada protocolo:

- `ArduSim/src/main/resources/protocols`: En esta ruta se crea un directorio por cada protocolo con su propio nombre. Aquí se guardan los archivos de configuración de cada protocolo, como los archivos `.properties`, que contienen las variables de ejecución predeterminadas. Además, si es necesario un entorno gráfico para modificar estas variables, los archivos `.fxml` se ubicaran en esta carpeta. En el contexto de este proyecto, esta ruta también se emplea para almacenar los datos que el sensor de contaminación simulado leerá.

- `ArduSim/src/main/java/com/protocols/` En esta ruta se almacena toda la lógica de la aplicación. Al igual que en la ruta anterior, se organiza un directorio para cada protocolo. Este directorio puede contener hasta tres subdirectorios:

  - `gui`: Aquí se almacena la lógica relacionada con la interfaz gráfica de cada protocolo, si es necesario.
  - `logic`: Este subdirectorio contiene la funcionalidad esencial para el manejo de los drones. Cada protocolo debe incluir un archivo `ProtocolHelper.java`, que actúa como intermediario entre el protocolo y el código interno de ArduSim. También se almacenan aquí los archivos `ProtocolThread.java`, si el protocolo requiere hilos adicionales.
  - `pojo`: En este directorio se guardan las clases Java utilizadas para la lógica del protocolo, facilitando la organización y el manejo de datos dentro del protocolo.

Protocolos disponibles

Actualmente, en ArduSim se puede encontrar una variada selección de protocolos ya implementados y funcionales:

- **Mission.** Este protocolo permite que un grupo de multicópteros sigan una misión previamente definida.

- **MBCAP.** Este protocolo está implementado para el estudio de técnicas de prevención de colisiones entre varios multicópteros que siguen una misión previamente definida [13].

- **MUSCOP.** Con este protocolo se consigue que un enjambre de drones sigan una misión que se encuentra almacenada en uno de ellos, el dron máster. Los multicópteros se mueven siguiendo una formación, y es resistente al fallo de cualquiera de los drones que forman el enjambre [14].

- **FollowMe.** En este caso, un enjambre sigue a un dron controlado manualmente por un piloto [15].

- **Vision.** Usando este protocolo, se necesitará un UAV equipado con una cámara. El dron utilizará la información que recoja de la cámara para realizar un aterrizaje seguro [16].

- **Shakeup.** Este protocolo permite que un enjambre de drones se reconfigure para reducir las posibilidades de colisión [17].

- **CompareTakeOff.** Este protocolo está implementado para estudiar diferentes algoritmos de despegue de forma segura.

4.2 Algoritmo Pollution-driven UAV Control - Discretized (PdUC-D)

Con el objetivo de monitorizar la contaminación en diversas áreas y localizar los puntos con mayores niveles de contaminación, se desarrolló el algoritmo Pollution-driven UAV Control (PdUC) [3]. Este algoritmo inicial permitió una gestión eficiente del dron en la detección de contaminantes. Sin embargo, para optimizar aún más el proceso en entornos reales y evitar la pérdida de tiempo al tomar muestras en puntos muy cercanos entre sí, se introdujo un nuevo algoritmo. Esta mejora consistió en discretizar el algoritmo original, dando lugar a una nueva versión denominada PdUC-D (PdUC-Discreto) [1], que mejora la eficiencia en la toma de muestras.

4.2.1. Base teórica

Este algoritmo se basa en su predecesor, el algoritmo PdUC. PdUC se fundamenta en la metaheurística quimiotaxis (chemotaxis en inglés) [18]. Esta metaheurística está basada en el movimiento de las bacterias, que se define por los estímulos químicos que reciben. Las bacterias tienden a moverse hacia áreas con mayor concentración de sustancias beneficiosas, como nutrientes, mientras que se alejan de áreas con sustancias perjudiciales, como el veneno.

En cada paso del movimiento, la dirección de la próxima posición se determina por el aumento de concentración de un componente químico específico entre la posición previa y la actual.

La idea detrás del algoritmo de es que un dron, equipado con un sensor de contaminación, imite el movimiento de las bacterias para realizar un barrido de la distribución de contaminación en el área y determinar el punto de máxima contaminación.

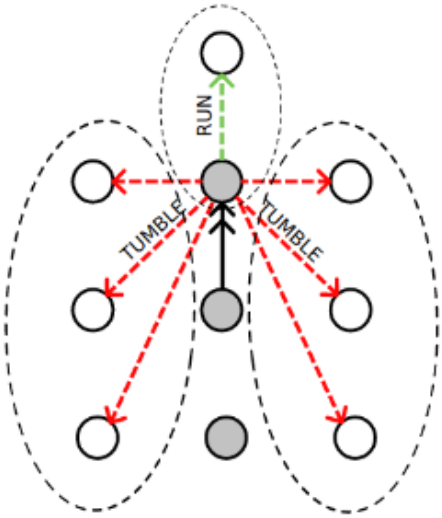

Figura 4.3: Movimiento seguido por el UAV durante la fase de busqueda.

Aunque el algoritmo PdUC obtenía buenos resultados, este requería demasiado tiempo para finalizar la ejecución. El tiempo de ejecución es una variable crítica en los multicopteros debido a su limitada batería. Si el algoritmo tarda mucho en ejecutarse la batería se agotará y el dron no podrá finalizar el recorrido. De este problema nace la necesidad de desarrollar un protocolo más eficiente.

4.2.2. Solución propuesta

Como solución a la problemática del algoritmo PdUC, se optó por discretizar el área a medir. La discretización consiste en transformar un sistema contínuo en uno discreto, realizando una división de sus componentes. Este enfoque matemático resulta muy eficiente cuando se busca optimizar un sistema [19] [1].

Para este proceso, la superficie se divide en una cuadrícula de celdas, bajo la suposición de que dentro de una celda no existe una gran variación en los niveles de contaminación ambiental. Así, el dron solo necesita desplazarse hacia los centros de las celdas, realizando una única medición por cada una de ellas. Este enfoque también ayuda a evitar la repetición de mediciones en puntos ya visitados, al definir claramente qué se considera como un área ya visitada.

El algoritmo se puede dividir en dos fases:

- **La fase de búsqueda:** Durante la fase de búsqueda el dron utilizará el movimiento quimiotaxis para buscar el punto con un mayor valor de contaminación. Si se detecta un incremento en la contaminación, se avanzará a la siguiente casilla en la misma dirección (Movimiento *Run*). Si la contaminación no aumenta, el UAV visitará las celdas alrededor de las celda con valor máximo (Movimiento *Tumble*), priorizando la celda más cercana. En la figura 4.3 se ilustra como sería este movimiento. En la figura 4.4 se puede observar el diagrama de flujo de esta fase.

- **La fase de exploración:** En esta fase se parte del punto de contaminación máxima encontrado en la fase anterior. Durante la exploración, el dron rea-

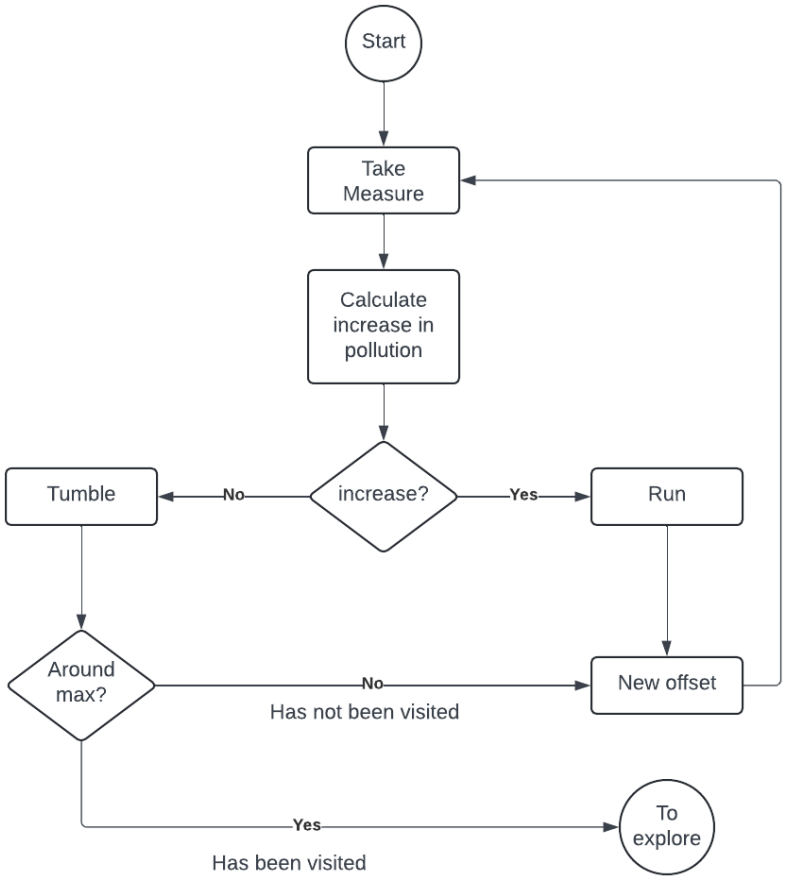

Figura 4.4: Diagrama de flujo fase de búsqueda del algortimo PdUC-D

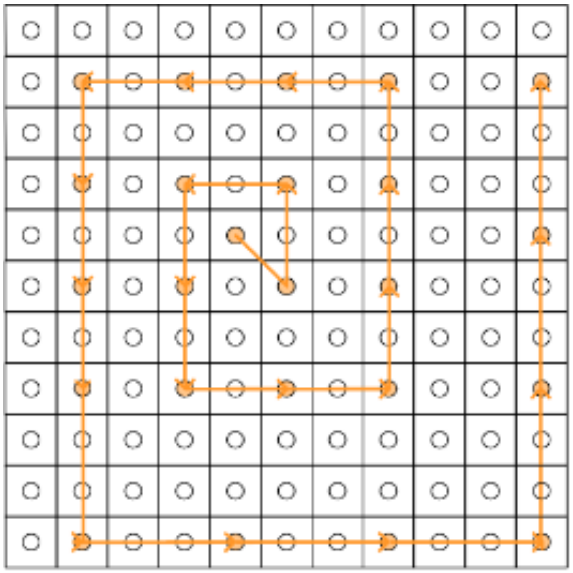

Figura 4.5: Movimiento seguido por el UAV durante la fase de exploración.

lizará un recorrido en forma de espiral alrededor del punto máximo y seguirá este movimiento hasta encontrar un punto con mayor contaminación o hasta haber cubierto el área entera que se esta analizando. En caso de encontrar un nuevo punto de máxima concentración, se volverá a realizar una fase de búsqueda sobre este punto. La figura 4.6 ilustra este movimiento de manera visual. Para este tipo de desplazamiento, se deben considerar los siguientes puntos:

1. En cada ronda de la espiral, se salta un número creciente de celdas sin medir. En la primera ronda la espiral tiene un radio de 3 celdas, y se salta 1 celda por movimiento. En la segunda ronda, tiene un radio de 5 celdas y se salta 2 celdas por movimiento, y así sucesivamente.
2. Se considerará como visitado todo el interior del cuadrado creado al terminar cada ronda de la espiral.

En la figura 4.6 se puede observar el diagrama de flujo de esta fase.

Para evitar evitar medir dos veces las mismas celdas, este algoritmo hace uso de dos matrices, $P_{m,n}$ y $B_{m,n}$, en las que se almacenarán los datos medidos y las celdas visitads, respectivamente. Cada vez que se visite un punto $t_{i,j}$, se guardará su valor en $P_{i,j}$ y se pondrá en `true` la celda $B_{i,j}$.

$$P_{m,n} = \begin{pmatrix} p_{1,1} & p_{1,2} & \cdots & p_{1,n} \\ p_{2,1} & p_{2,2} & \cdots & p_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ p_{m,1} & p_{m,2} & \cdots & p_{m,n} \end{pmatrix} \tag{4.3}$$

$$B_{m,n} = \begin{pmatrix} b_{1,1} & b_{1,2} & \cdots & b_{1,n} \\ b_{2,1} & b_{2,2} & \cdots & b_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m,1} & b_{m,2} & \cdots & b_{m,n} \end{pmatrix} \tag{4.4}$$

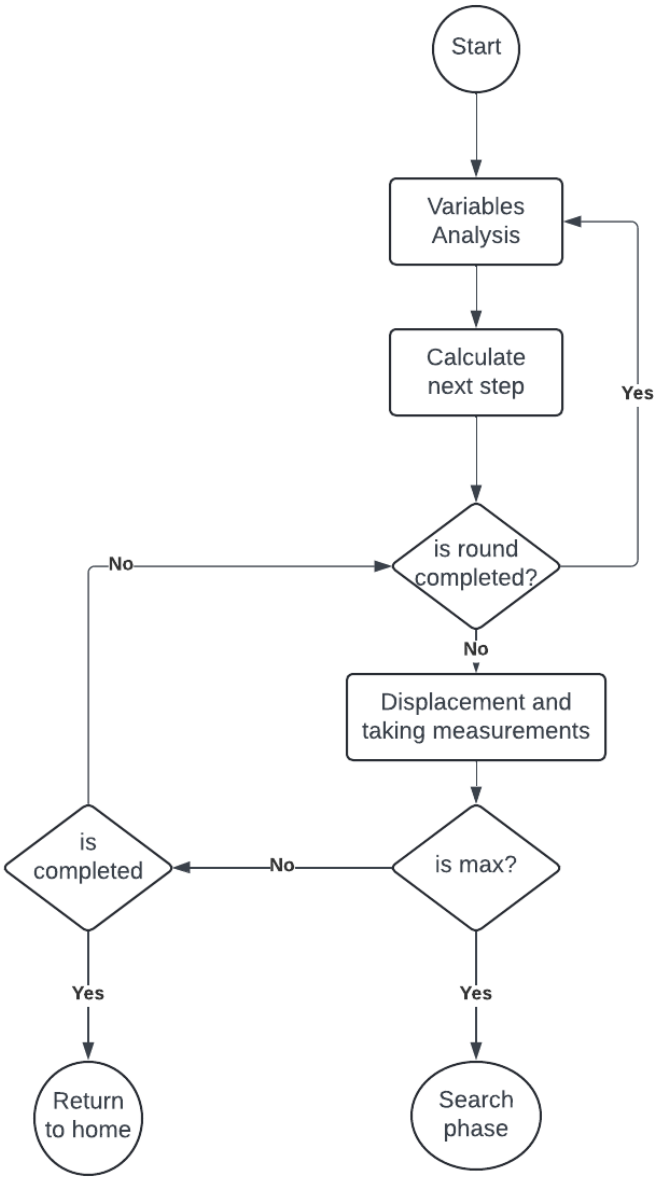

Figura 4.6: Diagrama de flujo fase de exploración del algortimo PdUC-D

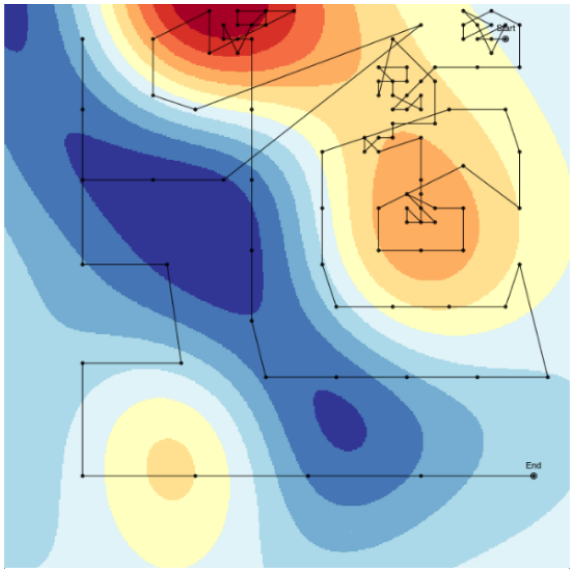

Figura 4.7: Traza de ejecución del algoritmo PdUC-D.

4.2.3. Prueba del algoritmo

La figura 4.7 muestra un ejemplo de la traza de ejecución del algoritmo PdUC-D. En la imagen se puede ver un mapa de calor con los niveles de contaminación y la trayectoria seguida por el dron.

4.3 Tecnologías usadas, mantenimiento y control de versiones

El desarrollo de este proyecto se llevó a cabo en su totalidad en un entorno Linux, utilizando la distribución Ubuntu [20]. Se adoptó una metodología de desarrollo incremental en la que el proyecto se construyó de manera progresiva, añadiendo nuevas funcionalidades y probando su funcionamiento en cada fase del proyecto. Este enfoque permitió realizar pruebas en cada fase del desarrollo, garantizando que el comportamiento de los drones correspondiera con lo esperado.

Las tecnologías utilizadas en el desarrollo de este proyecto fueron las siguientes:

- Lenguaje de programación: Java.
- Simulador de vuelo: ArduSim.
- IDE: IntelliJ IDEA[21].
- Diseño de interfaces: Scene Builder[22].

CAPÍTULO 5
Desarrollo

5.1 PdUC-D coordinado

El algoritmo PdUC-D ofrece una gran eficiencia en el número de movimientos y reducción de tiempo en finalizar la simulación, pero requiere demasiado tiempo para analizar un área de gran tamaño, lo cual implica que durante una prueba real se pueda agotar la batería del dron y no sea posible finalizar la ejecución del algoritmo. Frente a este problema, surge la idea de utilizar varios drones comunicados entre ellos para realizar el monitoreo de la contaminación en un área.

5.1.1. Solución propuesta

La idea detrás de este algoritmo es reducir el área de monitoreo para cada dron. Para ello, se ha optado por dividir el área inicial en tantas subáreas como drones tengamos y asignar cada una de estas subáreas a un dron distinto.

Dentro del área asignada, cada dron realizará el algoritmo PdUC-D de forma independiente, comunicándose con los otros drones sobre cada movimiento realizado. Cuándo el dron que haya medido el valor máximo absoluto de contaminación empiece la fase de exploración, este avisará a los otros drones para realizar una fase de exploración conjunta. Los otros drones abandonarán la tarea actual y realizarán la exploración conjunta.

Durante esta fase de exploración conjunta sigue los siguientes pasos:

1. Se generan los puntos a visitar. Los puntos generados seguirán el patrón de movimiento en espiral, visto en la figura 5.2 de la sección anterior, hasta llegar al límite del área.

2. Los puntos se agruparán en tantos clusters como drones haya, y se asignará cada cluster a un dron en base a la distancia.

3. Cada dron visitará los puntos de su cluster.

En la figura 5.1 se puede observar el diagrama de flujo de esta fase.

En la imagen 5.2 se puede apreciar un ejemplo de como quedaría la distribución de los puntos para la ejecución de una fase de exploración conjunta.

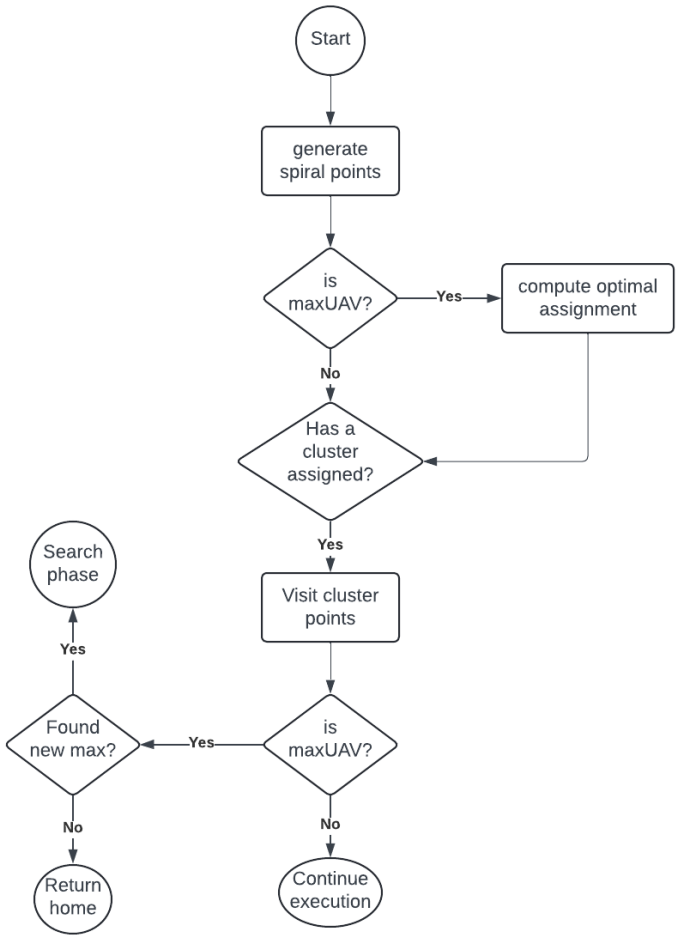

Figura 5.1: Diagrama de flujo fase de búsqueda conjunta del algortimo PdUC-D coordinado

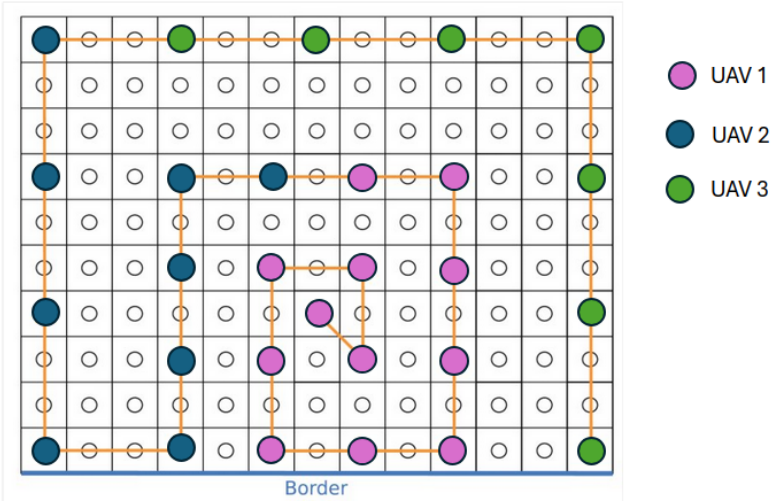

Figura 5.2: Ejemplo de la distribución de puntos duranta la fase de exploración conjunta

5.2 Código desarrollado

5.2.1. Método principal

Este método inicializa todas las variables, calcula las subáreas en las que actuaran cada dron y crea el bucle principal, encargado de lanzar las dos fases del algortimo.

```
    //PollutionThread.java
public void run() {
    // Calculate grid size
    sizeX = (int) ((double) PollutionParam.width / PollutionParam.
        density);
    sizeY = (int) ((double) PollutionParam.length / PollutionParam.
        density);
        //Initialize variables
    visited = new boolean[sizeX][sizeY];
    joint = false;
    locations = new HashMap<Long, DataPoint>();
    (...)
    test_finished = false;
    new Thread(this::listenForMessages).start(); //(1)
    /* Compute initial point */
    if (API.getArduSim().getNumUAVs() != 1) { //(2)
      pMax = computeIniPoint();
    } else {
      pMax = new DataPoint(sizeX / 2, sizeY / 2);
    }
    /* Start Algorithm */
    //Make the first move
    pCurrent = new DataPoint(pMax);
    pCurrent.addY(1);
    pMaxAbs = new DataPoint(pMax);
    maxUAV = copter.getID();
    long start = System.currentTimeMillis();
    String movement = "pducd";
    try { //(3)
      // Initial pMax and pCurrent measurement
      moveAndRead(pMax);
      moveAndRead(pCurrent);
      startPattern(movement);

    } catch (LocationNotReadyException e) {
      e.printStackTrace();
      endExperiment("Unable to calculate the target coordinates.");
      gui.exit(e.getMessage());
      return;
    }
    long executionTime = System.currentTimeMillis()-start;
    endExperiment(movement + " ended successfully in " + executionTime
        + " ms"); //(4)
  }
```

Estos son los puntos más destacados:

1. Se lanza un hilo que ejecutará el método listenForMessages(). Este hilo será el encargado de analizar los mensajes recibidos. Se hablará mas en profundidad de los mensajes más adelante.

2. Se calcula el punto de inicio de cada dron. Para ello se divide el área original en tantas subáreas como drones haya y se le asigna cada área a un dron en función de su identificador.

3. Se lanza el método que contiene el bucle principal del algoritmo:

```
1        //PollutionThread.java
2        private void startPattern(String patternName) throws
             LocationNotReadyException {
3    if(patternName.equals("pducd")){
4      //Main loop. It stops when the explore phase finishes
         without finding a new max
5      boolean newMax = true;
6      while(newMax) {
7        /* Phase 1: search */
8        runAndTumble();
9        /* Phase 2: exploration */
10       newMax = joint ? jointExplore() : explore();
11     }
12   }else if(patternName.equals("spiral")){
13     spiral();
14   }
15     }
```

4. Una vez que no se encuentra un nuevo valor máximo después de una fase de exploración, termina el bucle principal y el protocolo llega a su final con el método endExperiment(String msg). Este método es el encargado de hacer que el dron vuelva a la posición inicial y aterrice de forma segura, terminando así con el algoritmo y con la ejecución.

5.2.2. Envío y escucha de mensajes.

Para conseguir una comunicación entre los UAVs, se preveé que estos utilicen WIFI para transmitir mensajes UDP entre ellos. ArduSim es capaz simular la transmisión broadcast entre UAVs virturales cuando se ejecuta como simulador. Para que el mismo código sea válido en ambos entornos, ArduSim implementa una capa de abstracción sobre las comunicaciones. Para las comunicaciones se utilizará una instancia de CommLink, que proporciona las siguientes funciones:

- void sendBroadcastMessage(byte[]). Un multicóptero envía un mensaje de difusión a otros UAV codificado en una matriz de bytes.
- byte[] recibirMensaje(). Un multicóptero recibe un mensaje enviado desde otro UAV. El método se bloquea hasta que se recibe un mensaje, como en un socket real.
- byte[] receiveMessage(int). En este caso, el método se bloquea hasta que se recibe un mensaje, o se completa el tiempo de espera especificado como parámetro.

Centrandonos en el código, en el constructor de la clase PollutionThread.java se obtiene la instancia CommLink del UAV en cuestión con la instrucción this.commLink = API.getCommLink(numUAV);. Esta instancia se usará para enviar y recibir mensajes.

Como se ha comentado anteriormente, el método principal crea un hilo que ejecutará el método listenForMessages(), proporcionando una escucha activa de mensajes:

```
    //PollutionThread.java
private void listenForMessages(){
    byte[] b;
    String msg;
    while (!test_finished){
        b = commLink.receiveMessage(100);
        if (b != null) {
            msg = new String(b);
            if (msg.startsWith("Marking")) { // Marking point {x} {y}
                //(1)
                process_message_marking(msg);
            } else if (msg.startsWith("Going")){ //Going to: {x} {y}
                UAV: {id} //(2)
                process_message_going(msg);
            } else if (msg.startsWith("UAV")){ // UAV {ID} : Read: [{x
                }, {y}, {value}] //(3)
                process_message_UAV(msg);
            } else if (msg.startsWith("Starting")){//Starting explore
                UAV {ID} //(4)
                process_message_starting(msg);
            } else if (msg.startsWith("Found")){ //Found newMax [{x}, {
                y}, {measurement}] //(5)
                process_message_found(msg);
            } else if (msg.startsWith("Cluster")){ // Cluster {id} from
                [{x}, {y}] to [{x}, {y}] drone {id} //(6)
                process_message_cluster(msg);
            }
        }
    }

}
```

Este método compara el principio del mensaje recibido entre los distintos tipos de mensajes existentes y luego realiza las acciones correspondientes:

1. `Marking point {x} {y}`:
   - Uso: Se usa para indicar los puntos que se marcan como visatados usando el método markExploredArea(int radius, int skip).
   - Acción a realizar: Marcar el punto {x,y} como visitado.
2. `Going to: {x} {y} UAV: {id}`:
   - Uso: Se usa para indicar el próximo destino de un dron. Aunque puede parecer similar al siguiente mensaje, este es útil para evitar algún posible solapamiento causado por el tiempo de vuelo entre dos puntos.

- Acción a realizar: Marcar el punto como visitado

3. `UAV {ID} : Read: [{x}, {y}, {value}]:`
   - Uso: Se usa cuando el dron visita un punto.
   - Acción a realizar: Actualizar el valor máximo absoluto, en caso de serlo.
4. `Starting explore UAV {ID}:`
   - Uso: Se usa cuando un dron empieza la fase de exploración conjunta.
   - Acción a realizar: Se comprueba que el dron sea el que tiene el valor máximo y, en ese caso, se actualiza a verdadera la variable joint para que el dron pase a hacer la exploración conjunta
5. `Found newMax [{x}, {y}, {measurement}]:`
   - Uso: Se usa cuando se encuentra un nuevo valor máximo en la fase de exploración conjunta.
   - Acción a realizar: Comprobar si el punto esta entre los límites del área asignada al dron y, en ese caso, actualizar las variables máximas absolutas e informar a los otros drones.
6. `Cluster {id} from [{x}, {y}] to [{x}, {y}] drone {id}`
   - Uso: Se usa para informar a los drones del cluster que tienen que visitar
   - Acción a realizar: Comprobar que el mensaje este dirigido a ese dron y, en ese caso, actualizar las variables que indican el primer y último punto del cluster.

5.2.3. Movimiento del dron

El método moveAndRead(DataPoint p). Es el encargado de hacer que el dron se mueva de un punto a otro y ejecute una lectura del sensor.

```
1  //PollutionThread.java
2  private void moveAndRead(DataPoint p) throws
       LocationNotReadyException {
3
4       double m;
5  String msg = "Going to: " + p.getX() + " " + p.getY() + " UAV: " +
       copter.getID();
6  commLink.sendBroadcastMessage(msg.getBytes()); //(1)
7  locations.put(copter.getID(), p); //(2)
8  move(p); //(3)
9  (...)
10 m = PollutionParam.sensor[(int) copter.getID()].read((int) copter.
       getID()); //(4)
11 (...)
12 synchronized (visited) { visited[p.getX()][p.getY()] = true; } //
       (5)
13 p.setMeasurement(m);
```

```
14      if (m > pMaxAbs.getMeasurement()) { //(6)
15        pMaxAbs = new DataPoint(p);
16        maxUAV = copter.getID();
17      }
18
19      msg = "UAV " + copter.getID() + " : Read: " + p.toString();
20      commLink.sendBroadcastMessage(msg.getBytes()); //(7)
21    }
```

Se han omitido algunas partes del código por resultar de poco interés. Destacan los siguientes puntos:

1. Envia un mensaje a todos los otros drones informando de que va a visitar el punto.
2. Actualiza la variable Map<Long, DataPoint> locations;, donde se mantiene la última ubicación conocida de todos los drones. Esta variable se usará para calcular la asignación de clusters a cada dron en la fase de exploración conjunta.
3. Hace una llamada al método que envía la orden de moverse al dron.
4. El dron hace la lectura del valor de contaminación.
5. Se marca la celda actual como visitada.
6. Actualiza el valor máximo absoluto de contaminación, en caso de que este sea mayor que el actual.
7. Envia un mensaje a los otros drones informando del punto y el valor leído.

5.2.4. Fase de exploración conjunta

Este método se ejecutará cuando el UAV que mantiene el punto con valor máximo de contaminación inicie la fase de exploración.

```
1  //PollutionThread.java
2  private boolean jointExplore() throws LocationNotReadyException {
3      String msg = "Starting explore UAV " + copter.getID();
4      commLink.sendBroadcastMessage(msg.getBytes()); //(1)
5
6      List<DataPoint> points;
7      Cluster clus = null;
8      /* Explore phase */
9      gui.log("UAV " + copter.getID() + ": JOINT EXPLORE - Start");
10     points = generateSpiralPoints(); //(2)
11
12     if (copter.getID() == maxUAV) { //Compute the clusters //(3)
13         ArrayList<Cluster> clusters = Cluster.createClusters(points);
14         System.out.println("Starting optimal Assignment");
15         /* Prepare the matrix for the Hungarian Algorithm using drones
               as rows and clusters as columns */
16         clus = optimalAssignment(clusters);
17
```

```
18      } else {
19          if (clus_lastPoint != null) { //(4)
20              clus = new Cluster(0, clus_firstPoint, clus_lastPoint);
21          } else {
22              gui.log("UAV: " + copter.getID() + " Doesn't find any point
                    ");
23          }
24      }
25      if (clus != null) {
26          gui.log("UAV: " + copter.getID() + " visits cluster: " + clus);
27          joint = visitPoints(clus, points); //(5)
28          }
29
30      return !joint;
31  }
```

Las partes mas destacadas son:

1. Cuando un UAV entra en el método envia un mensaje a los demás, que comprobarán si este es el que tiene el valor máximo absoluto y, en cuyo caso, empezarán su ejecución del método.

2. Se generan todos los puntos de la espiral. La funcióngenerateSpiralPoints() crea todos los puntos posibles dentro del área de monitorización, siguiendo el patrón de movimiento en espiral, y los ordena en función de su distancia al punto máximo.

3. En este método se hace la agrupación de los puntos de la espiral en clusters y su posterior asignación a los UAVs en función de su proximidad. Lo ejecuta el dron que mantiene el valor máximo porque este entra primero al método y tiene tiempo de hacer los cómputos necesarios mientras los otros drones terminan su movimiento actual. Los métodos createClusters(List <DataPoint> points) y optimalAssignment(List<Cluster> clusters) se explicarán más adelante.

4. En este punto el UAV comprueba si se le ha asignado algún cluster (si así ha sido habrá recibido un mensaje con los datos del cluster) y, en ese caso, se crea una instancia de Cluster utilizando el primer y último punto del cluster, obtenidos en el método listenForMessages().

5. El método visitPoints(Cluster clus, List<DataPoint> points) buscará el primer y último punto del cluster en la lista points y visitará todos los puntos intermedios de forma ascendente o descendente, dependiendo de que punto sea mas cercano a la posición del dron. Devolverá false en caso de que encuentre un nuevo valor máximo.

5.2.5. Creación y asignación de clusters

Creación de clusters

Para crear los clusters se ha optado por dividir los puntos totales y asignar a cada cluster la misma cantidad de puntos. En el caso de que la división no

sea exacta, los primeros clusters recibirán 1 punto mas que los últimos porque la distancia entre ellos suele ser menor.

Para poder identificar los clusters se ha creado un objeto Cluster que utiliza un identificador y el primer y último punto del cluster:

```
public class Cluster {
    private final DataPoint firstPoint;
    private final DataPoint lastPoint;
    private final int id;

    public Cluster(int id, DataPoint firstPoint, DataPoint lasPoint){
        this.id = id;
        this.firstPoint = firstPoint;
        this.lastPoint = lasPoint;
    }
    public DataPoint getFirstPoint() {return this.firstPoint; }
    public DataPoint getLastPoint() {return this.lastPoint; }
    public int getId() { return this.id; }
    public String toString(){ (...) }
    public Pair<Integer, Integer> findStartingAndEndingPoints(List<
        DataPoint> points) { (...) } // Returns the position of the
        first and last points from the                    // cluster in
        the list

    // Constructor of all the cluster in the execution
    public static ArrayList<Cluster> createClusters(List<DataPoint>
        points) {
        ArrayList<Cluster> clusters = new ArrayList<Cluster>();
        int points_drone = (int) points.size() / API.getArduSim().
            getNumUAVs();
        int remainder = points.size() % API.getArduSim().getNumUAVs();
        int actual_point = 0;
        for (int i = 0; i < API.getArduSim().getNumUAVs() &&
            actual_point < points.size(); i++) { //cluster size = num
            UAVS
            clusters.add(new Cluster(i, points.get(actual_point),
                points.get(actual_point + (points_drone - 1) + (
                remainder > 0 ? 1 : 0))));
            actual_point += points_drone + (remainder-- > 0 ? 1 : 0);
        }

        return clusters;
    }
}
```

Asignación de clusters a UAV (Hungarian Algorithm)

El Algoritmo Húngaro [23] es un algoritmo de optimización el cual resuelve problemas de asignación en tiempo $O(n^3)$. En el siguiente código se utiliza una instancia de este algoritmo para asignar los clusters creados al dron mas óptimo:

```
//PollutionThread.java
private Cluster optimalAssignment(List<Cluster> clusters){
    double[][] mat = createMatrix(clusters); //(1)
```

```
4       HungarianAlgorithm hungarian = new HungarianAlgorithm(mat.length);
            //(2)
5       hungarian.fillWithCostmatrix(mat);
6       hungarian.solve();
7       ArrayList<Pair<Integer,Integer>> optAsg = hungarian.getAssignment()
            ;
8
9       /*Send their clusters to other drones */
10      for (Pair<Integer,Integer> pair : optAsg) {
11          if (pair.getValue1() < clusters.size()) {
12              String msg = clusters.get(pair.getValue1()).toString() + "
                    Drone " + pair.getValue0();
13              gui.log(msg);
14              commLink.sendBroadcastMessage(msg.getBytes()); //(3)
15          }
16      }
17
18      //Returns the cluster to visit
19      int numClus = optAsg.get((int) copter.getID()).getValue1();
20      return numClus < clusters.size() ? clusters.get(numClus) : null; //
            (4)
21  }
```

Destacan los siguientes puntos:

1. Se crea una matriz con el formato necesario para la ejecución del algoritmo. Esto es, una matriz de nxn, usando los drones como filas y los clusters como columnas, y siendo la celda el valor mínimo entre la distancia del dron al primer o al último punto del cluster.

2. Se crea una instancia del algoritmo y se ejecuta.

3. Se envia un mensaje por cada cluster a visitar, seguido del identificador del dron que debe visitarlo.

4. El método devuelve el cluster que debe visitar el UAV que ha lanzado su ejecución, o null en caso de no haberlo.

CAPÍTULO 6
Pruebas y analisis de los resultados

Durante este capítulo se analizarán los resultados obtenidos en la simulación del algoritmo PdUC-D coordinado. Para ello, se comparará el tiempo de ejecución del algortimo y los resultados obtenidos, para simulaciones con distintos números de drones. Para todas las simulaciones se utilizarán los mismos parámetros:

Parámetro	**Valor**
Área	1x1 km
Tamaño de celda	100
Número de celdas	10x10=100
Tiempo de muestreo	0.1 s
Altitud	5 m
Velocidad del dron	15 m/s

Tabla 6.1: parámetros simulación

6.1 PyKrige

PyKrige [26] es una librería de Python utilizada para realizar interpolación espacial utilizando kriging. Kriging es un método de inferencia espacial que permite estimar los valores de los datos en los lugares no muestreados.

De los distintos variogramas que soporta PyKrige, se ha optado por usar el variograma lineal, basándose en que la contaminación se concentra en su punto máximo y disminuye linealmente a medida que se aleja de este foco de contaminación.

Esta herramienta nos permitirá obtener un mapa de calor con los puntos con más contaminación a partir de los resultados obtenidos en la simulación.

6.2 Pruebas

Para las pruebas se usarán hasta un máximo de 6 drones. Todas las simulaciones usarán los parámetros de la tabla 6.1 y el mismo mapa de valores para

Núm. UAVs	Tiempo	Puntos visitados
1	11 min 46s	26
2	9 min 51 s	37
3	6 min 45 s	36
4	6 min 36 s	44
5	4 min 44 s	45
6	4 min 40 s	35

Tabla 6.2: tiempos de vuelos

asegurar que la variación en la posición de los valores no afecta al tiempo de vuelo.

Como se puede observar en la tabla 6.2, cuanto mayor es el número de drones, menos tiempo se tarda en finalizar la ejecución. Esto se debe principalmente a que cada dron recorre menos distancia, ya que su área de actuación es menor, y a que la fase de exploración conjunta permite a los drones ahorrarse mucho tiempo de desplazamiente en comparación con fases anteriores. Esta diferencia de tiempo será mas notable para áreas mas grandes.

Además, se puede apreciar que cuanto mayor sea el número de drones, más puntos se visitan, por lo que las mediciones serán más precisas (para 6 drones se disminuye el número de puntos visitados, ya que, por la distribución de áreas actual, el dron no tiene tanto rango de movimiento y se encuentran los valores máximos más rápidamente que en otra distribución).

En las siguientes figuras se puede observar la traza de cada una de las simulacios a la izquierda y el mapa de calor obtenido a la derecha (advertencia: el mapa de calor está invertido verticalmente, los puntos de contaminación deberían estar arriba a la derecha). Los mapas de calor muestran que todas las trazas identifican correctamente los focos de contaminación y, además, como se visitan mas puntos que en la ejecución con 1 dron, los datos son mas precisos.

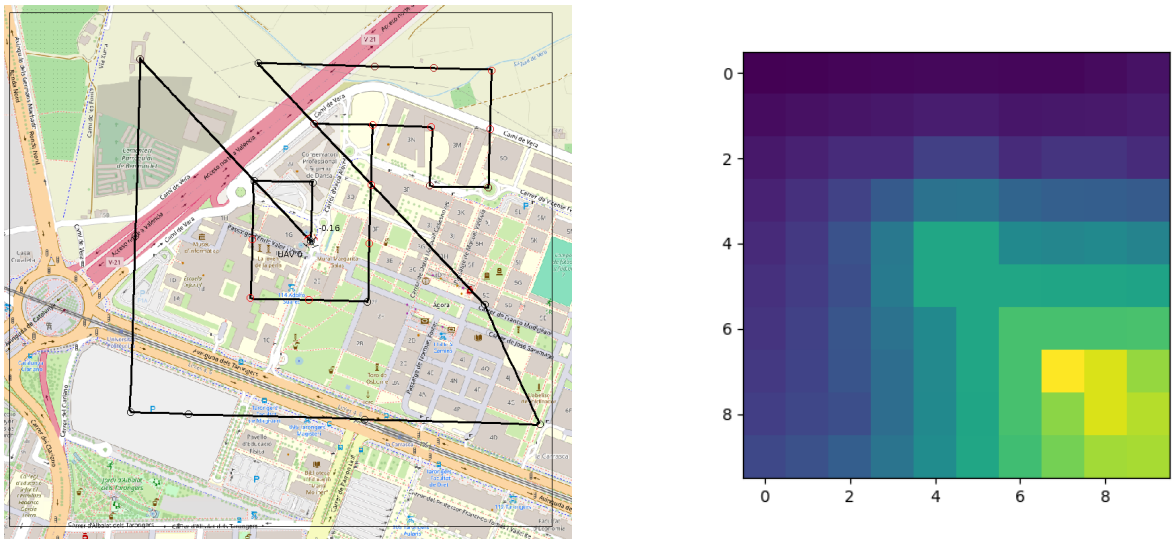

Figura 6.1: Ejecución para 1 dron

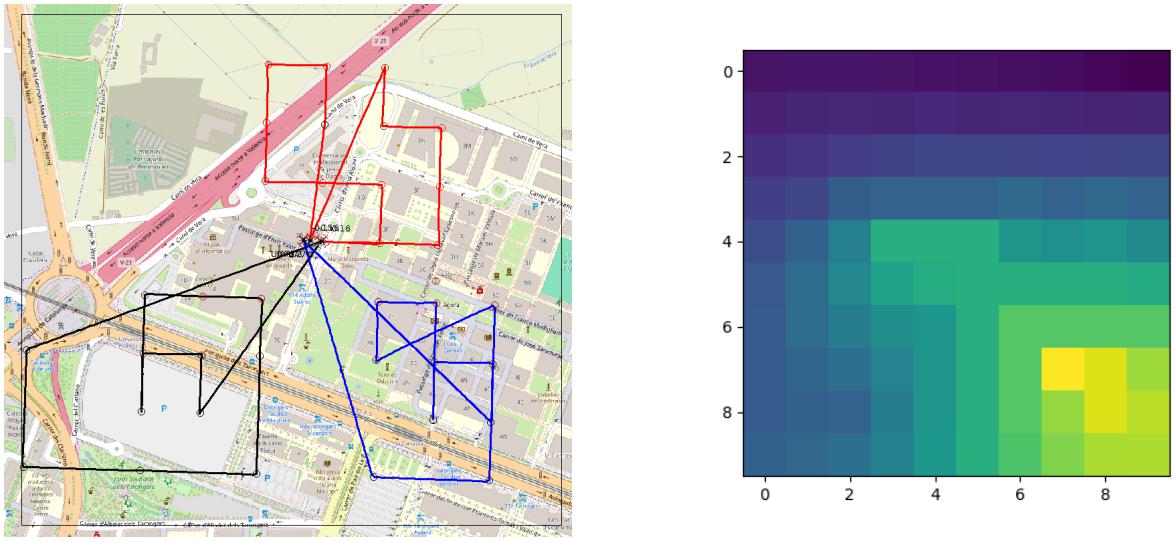

Figura 6.2: Ejecución para 2 drones

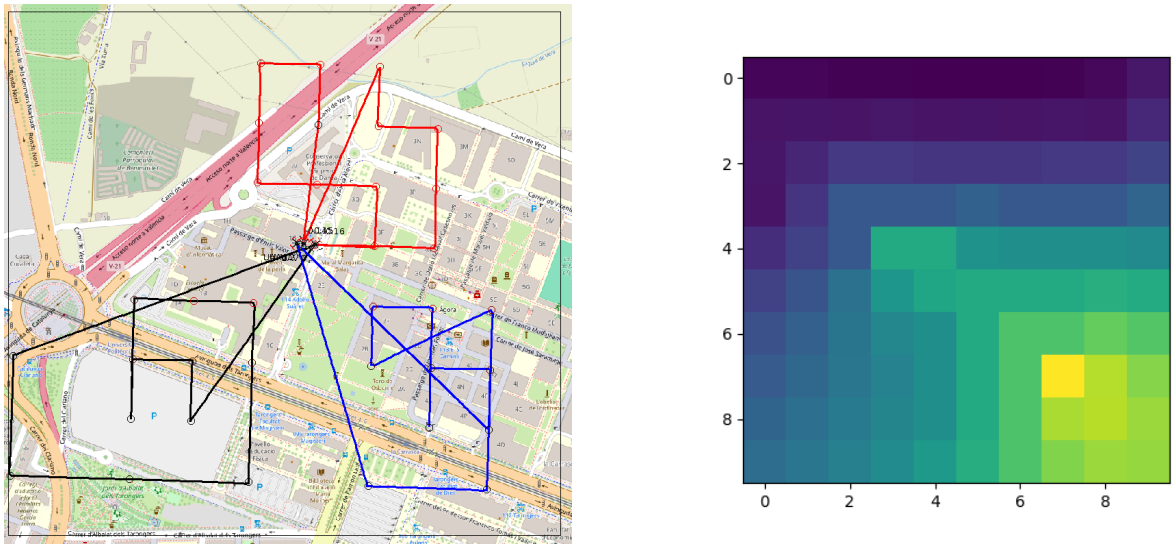

Figura 6.3: Ejecución para 3 drones

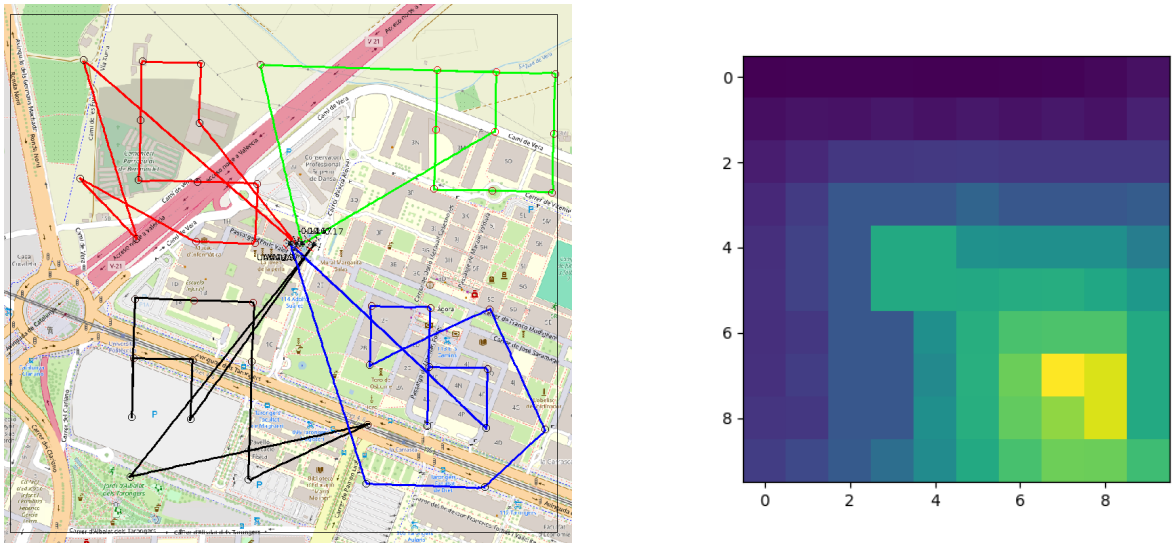

Figura 6.4: Ejecución para 4 drones

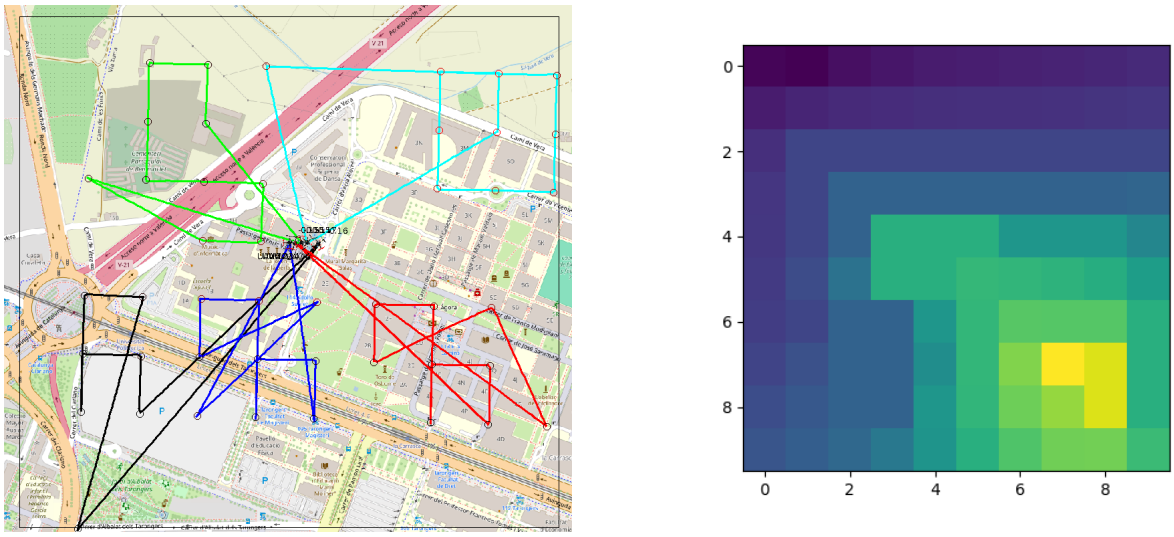

Figura 6.5: Ejecución para 5 drones

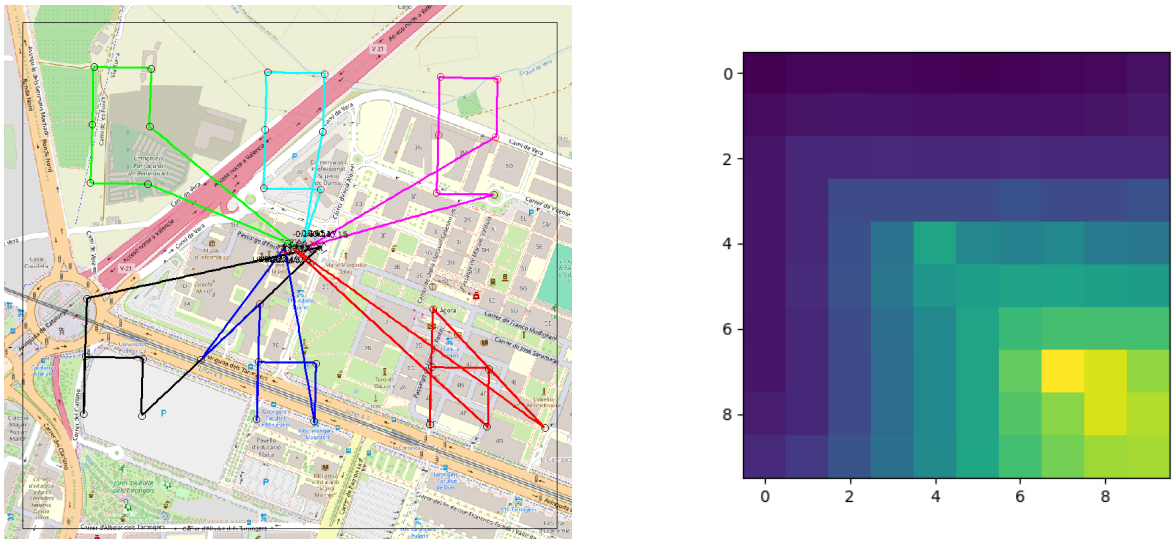

Figura 6.6: Ejecución para 6 drones

CAPÍTULO 7
Conclusiones

Este proyecto ha hecho posible la ejecución del algoritmo PdUC-D de forma coordinada entre varios UAVs. Gracias a este algoritmo, los drones son capaces de desplazarse autonomamente por un área, calculando su contaminación y facilitando una futura toma de decisiones con el objetivo de reducir los niveles de contaminación atmosférica.

Atendiendo a los resultados presentados en el capítulo anterior, se confirma que el algoritmo coordinado presenta una mejora sustancial en tiempo respecto a una simulación realizada con un solo UAV.

Además, el proyecto cumplió todos los objetivos definidos en la sección 1.2, siguiendo tres fases:

1. Investigación y búsqueda de información.
2. Desarrollo.
3. Pruebas y análisis de resultados.

7.1 Mejoras futuras

Algunas mejoras que se podrían aplicar para optimizar el tiempo son:

- **Optimización de la división de áreas**. El código actual divide la altura en dos y luego reparte la mitad de los drones en la parte superior y la otra en la inferior. Posiblemente se pueda mejorar este planteamiento para intentar que las subáreas generen una forma mas cuadrada, lo cual mejoraría
- **Optimización de la fase de exploración conjunta**. Si se realizará la clusterización de los puntos de la espiral basandose en el tiempo en recorrer cada segmento, en vez de la cantidad de puntos, se podría reducir el tiempo de total destinado a esta fase.
- **Aproximaciones con Inteligencia Artificial**. El uso de técnicas de Inteligencia Artificial como *Particle swarm optimization* [24] o *Deep Reinforcement Learning* [25] podrían ayudar a mejorar el movimiento de los drones, reduciendo así el tiempo necesario para el monitoreo del área.

Bibliografía

[1] Alvear-Alvear, Ó.; Tavares De Araujo Cesariny Calafate, CM.; Zema, N.; Natalizio, E.; Hernández-Orallo, E.; Cano, J.; Manzoni, P. (2018). A Discretized Approach to Air Pollution Monitoring Using UAV-based Sensing. Mobile Networks and Applications. 23(6):1693-1702. https://doi.org/10.1007/s11036-018-1065-4

[2] Fabra Collado, FJ.; Tavares De Araujo Cesariny Calafate, CM.; Cano, J.; Manzoni, P. (2018). ArduSim: Accurate and real-time multicopter simulation. Simulation Modelling Practice and Theory. 87:170-190. https://doi.org/10.1016/j.simpat.2018.06.009

[3] O. Alvear, N. R. Zema, E. Natalizio, C. T. Calafate. Using UAV-Based Systems to Monitor Air Pollution in Areas with Poor Accessibility. Journal of Advanced Transportation, 10.1155/2017/8204353, agosto, 2017. https://doi.org/10.1155/2017/8204353

[4] Tipos de drones según la nueva normativa europea de EASA 2024 Consultado en https://aerocamaras.es/clases-de-drones-segun-la-nueva-normativa-europea/#cuantas-clases-drones-hay

[5] Contaminación atmosférica Consultada en http://www.ideam.gov.co/web/contaminacion-y-calidad-ambiental/contaminacion-atmosferica.

[6] Métodos de medición de la calidad del aire: 5 muestreos Consultada en https://ingenieriaambiental.net/metodos-de-medicion-de-la-calidad-del-aire/.

[7] Una nueva tecnología promete medir con precisión la contaminación del aire en Estados Unidos: ¿cómo funciona? Consultado en https://www.telemundo.com/noticias/noticias-telemundo/medio-ambiente/una-nueva-tecnologia-promete-medir-con-precision-la-contaminacion-del-rcna1252

[8] Cycling with clean air Consultado en https://cyclingwithcleanair.conbici.org/

[9] Research Infrastructures Services Reinforcing Air Quality Monitoring Capacities in European Urban & Industrial AreaS (RI-URBANS) Consultado en https://cordis.europa.eu/project/id/101036245

[10] Real Decreto 1036/2017, de 15 de diciembre, por el que se regula la utilización civil de las aeronaves pilotadas por control remoto. Consultado en https://www.boe.es/boe/dias/2017/12/29/pdfs/BOE-A-2017-15721.pdf.

[11] SITL Simulator (Software in the Loop) Consultado en https://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html

[12] Introduction to the Standard Directory Layout Consultado en https://maven.apache.org/guides/introduction/introduction-to-the-standard-directory-layout.html

[13] Fabra F, Zamora W, Sangüesa J, Calafate CT, Cano J-C, Manzoni P. A Distributed Approach for Collision Avoidance between Multirotor UAVs Following Planned Missions. *Sensors (2019).* https://www.mdpi.com/1424-8220/19/10/2404.

[14] Fabra Collado, FJ.; Zamora, W.; Reyes, P.; Sangüesa, JA.; Tavares De Araujo Cesariny Calafate, CM.; Cano, J.; Manzoni, P. MUSCOP: Mission-Based UAV Swarm Coordination Protocol. *IEEE Access (2020).* http://hdl.handle.net/10251/164054.

[15] Fabra Collado, FJ.; Zamora, W.; Masanet, J.; Tavares De Araujo Cesariny Calafate, CM.; Cano, J.; Manzoni, P. Automatic system supporting multicopter swarms with manual guidance. *Computers & Electrical Engineering. (2019)* http://hdl.handle.net/10251/156662.

[16] Wubben, J.; Fabra Collado, FJ.; Tavares De Araujo Cesariny Calafate, CM.; Krzeszowski, T.; Márquez Barja, JM.; Cano, J.; Manzoni, P. Accurate Landing of Unmanned Aerial Vehicles Using Ground Pattern Recognition. *Electronics (2019).* http://hdl.handle.net/10251/144317.

[17] Wubben, J.; Aznar, P.; Fabra Collado, FJ.; Tavares De Araujo Cesariny Calafate, CM.; Cano, J.; Manzoni, P. Toward secure, efficient, and seamless reconfiguration of UAV swarm formations. *IEEE. 1-7 (2020).* http://hdl.handle.net/10251/179834.

[18] I. Boussa ïD, J. Lepagnot, and P. Siarr. A survey on optimization metaheuristics. *Information Sciences. An International Journal*, pp. 82–117, 2013.

[19] Paul Mínguez, J. (2022). Desarrollo de sistema de guiado para multicópteros en base a datos de contaminación ambiental. *Universitat Politècnica de València*. http://hdl.handle.net/10251/187325

[20] Canonical. *Distribución Ubuntu*. https://ubuntu.com/

[21] JETBRAINS IDEs. *IntelliJ IDEA IDE* https://www.jetbrains.com/idea/

[22] Gluon. *Scene Builder*. https://gluonhq.com/products/scene-builder/

[23] Algoritmo húngaro. Consultado en https://es.wikipedia.org/wiki/Algoritmo_h%C3%BAngaro.

[24] Particle swarm optimization Consultada en https://en.wikipedia.org/wiki/Particle_swarm_optimization.

[25] Deep reinforcement learning Consultada en https://en.wikipedia.org/wiki/Deep_reinforcement_learning.

[26] PyKrige Consultada en https://geostat-framework.readthedocs.io/projects/pykrige/en/stable/.

APÉNDICE A
Objetivos de Desarrollo Sostenible (ODS)

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

Reflexión sobre la relación del TFG/TFM con los ODS y con el/los ODS más relacionados.

La contaminación atmosférica representa un creciente riesgo para la salud pública. Frente a la necesidad de tomar medidas para reducir la contaminación ambiental, este proyecto se ha alineado con el objetivo 3 (Salud y bienestar) buscando facilitar la localización de la contaminación atmosférica para ayudar a tomar las medidas necesarias.

Para abordar eficazmente este problema y mejorar la salud de las personas el primer paso es trabajar sobre los focos de contaminación en las ciudades y zonas con gran densidad de población. La contaminación en los nucleos urbanos representa una amenaza para todas aquellas personas que respiren ese aire. Por ello, este proyecto también se ha alineado con el objetivo 11 (Ciudades y comunidades sostenibles) para encontrar y controlar los altos niveles de contaminación en nucleos urbanos.

Además, este proyecto se relaciona con el Objetivo 9 (Industria, innovación e infraestructuras) y el Objetivo 12 (Producción y consumo responsables). El uso de drones para medir los niveles de contaminantes generados por procesos industriales permite monitorear la sostenibilidad de las actividades industriales y garantizar que se mantengan dentro de los límites establecidos. Esto facilita la detección temprana de aumentos en la contaminación, permitiendo intervenciones como la imposición de sanciones o la reducción de la producción hasta estabilizar los niveles de contaminación.

Los altos niveles de contaminación producidos por todos nosotros esta teniendo un impacto mas que notable sobre la salud del planeta. En los últimos años se ha experimentado un aumento notable de las temperaturas y la aparición de fenómenos atmosféricos cada vez con mas frecuencia. Trabajando con el objetivo 13 (Acción por el clima), se buscaría, si bien no revertir puesto que ya es imposible, reducir al mínimo posible el cambio climático.

Los humanos no somos los únicos afectados por las consecuencias de la contaminación atmosférica. La vida de los ecosistemas terrestres se ha vido ampliamente afectada por los efectos del cambio climático llevando a la destrucción de hábitats naturales y la perdida de biodiversidad. Por esta razón, el objetivo 15 (Vida de ecosistemas terrestres) también cobra gran importancia en este proyecto.

De manera similar, aunque en menor medida, el Objetivo 14 (Vida submarina) también es pertinente. La contaminación atmosférica contribuye a la contaminación de mares y ríos, afectando la vida acuática. Además, el calentamiento global está elevando las temperaturas marinas y deteriorando los ecosistemas acuáticos. El deshielo, que afecta a los ecosistemas gélidos y podría tener consecuencias a largo plazo para los ecosistemas marinos y terrestres, también es una preocupación relevante.

Por último, el Objetivo 6 (Agua limpia y saneamiento) está estrechamente vinculado con este trabajo, ya que la reducción de la contaminación atmosférica también puede contribuir a la disminución de la contaminación en fuentes de agua.