UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA
DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

Universitat Politècnica de València

Departamento de Informática de Sistemas y Computadores

Despegue seguro y eficiente de enjambres de drones

Trabajo de Fin de Máster

Máster Universitario en Ingeniería de Computadores y Redes

Autor
Carles Sastre Pérez

Directores
Carlos Tavares Calafate
Jamie Wubben

Febrero de 2022

Agradecimientos

En primera instancia me gustaría agradecer a mis dos asesores, Carlos Tavares Calafate y Jamie Wubben, por su constante apoyo y extraordinaria orientación que han hecho posible cumplir con los objetivos propuestos en el presente proyecto. Asimismo, me gustaría expresar mi agradecimiento a todo el equipo del Grupo de Investigación de Redes de Computadores (GRC) por compartir su inmensa sabiduría y acogerme como uno más. Finalmente, a mi familia y amigos que nunca han dejado de apoyarme en los malos momentos y siempre me han ayudado a alcanzar cada reto que inicio.

Resumen

Durante los últimos años, el empleo de vehículos aéreos no tripulados (VANTs) se ha extendido en gran medida por diferentes sectores dados los numerosos progresos en lo que respecta a la tecnología que los envuelve, y por su considerable bajada de precio en el mercado. Aunque actualmente los multicópteros se encuentran en un continuo auge para usos recreativos, existen también varias empresas centradas en estas aeronaves para facilitar ciertas tareas que tiempo atrás eran inaccesibles para el ser humano, o que, por lo contrario, implicaban un gran coste. En este contexto, el concepto de enjambre de drones nos permite ensanchar e incorporar nuevas aplicaciones más refinadas en las que diversas aeronaves se coordinan entre sí para llevar a cabo tareas de gran envergadura.

Dicho esto, en el presente trabajo de fin de máster se va a elaborar un nuevo algoritmo que permita optimizar el tiempo de despegue de los drones pertenecientes a un enjambre. Concretamente, se busca cumplir con el requisito básico de que no existan colisiones entre ellos. Para llevar a cabo este propósito, realizaremos una gran cantidad de experimentos en el simulador ArduSim para confirmar que el algoritmo proporciona una solución robusta, y tiene un tiempo de cálculo razonable. Además de buscar optimizar el coste del algoritmo para minimizar el tiempo de despegue del enjambre, se llevará a cabo un minucioso análisis de los resultados obtenidos respecto al resto de los algoritmos de despegue ya existentes.

Palabras clave: VANTs, Despegue, Enjambre, ArduSim, Colisiones.

Abstract

In recent years, the use of unmanned aerial vehicles (UAVs) has spread to a great extent to different sectors worldwide. This, mainly due to the numerous technological advances which lowered prices considerable. Especially in terms of recreational, much advancement have been made. Also, the industry is incorporating UAVs as valuable assets. Nowadays, multiple companies are focused on facilitating certain task which were once inaccessible to humans, or otherwise involved a great cost. In this context, a swarm of drones allows broadening and incorporate new and/or more refined applications.

In this master's degree project, a new algorithm will be developed to optimize the simultaneous takeoff of a swarm of UAVs. This algorithm has the objective to reduce the takeoff time as much as possible while avoiding collisions between drones. After designing our algorithm, we will verify that it meets all the requirements using the ArduSim simulator. Once verified, we will compare our solution to other existing takeoff algorithms.

Keywords: UAVs, Takeoff, Swarm, ArduSim, Collisions.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Hoy en día, el uso de los vehículos aéreos no tripulados (VANTs), comúnmente conocidos como drones, está progresando a pasos agigantados, proporcionando una gran cantidad de beneficios a nuestra sociedad. Una de las principales razones de su popularización es su bajo coste, la cual ha proporcionado a la población una mayor facilidad de adquisición de estos dispositivos electrónicos.

Debido a este crecimiento, muchas empresas han decidido realizar un esfuerzo económico para llevar a cabo investigaciones referentes a los VANTs con el objetivo de satisfacer una necesidad no cubierta, o de mejorar el rendimiento de sus principales aplicaciones. Actualmente, aparte de ser un modo de entretenimiento para el público, el empleo de estas aeronaves de manera conjunta ha hecho posible llevar a cabo tareas más complejas.

A causa de la facilidad de desplazamiento de los drones por cualquier tipo de terreno, y de su facilidad para superar obstáculos, se ha podido aumentar la variedad de aplicaciones en áreas muy diversas. Entre las aplicaciones más populares que podemos destacar tenemos: el control de cultivos agrícolas, la monitorización del tráfico, y la supervisión de zonas para la búsqueda de personas perdidas [2].

En las tareas mencionadas anteriormente resulta fundamental que los drones trabajen conjuntamente con otros formando un enjambre con la autonomía suficiente para tomar decisiones por sí mismos, sin disponer de la necesidad de que un humano los controle. Dicha inteligencia permite realizar una misión de la forma más eficiente posible ya que, en el caso de que uno de los drones se averíe, el enjambre se reorganizaría para llevar a cabo su misión con total normalidad.

1.1 Motivación

Actualmente, pese a las múltiples investigaciones referentes a los enjambres de drones, siguen existiendo considerables problemas en su monitorización que impiden extender su uso.

Uno de los principales problemas existentes es el despegue de estos drones. Cuando se pretende gestionar un enjambre con un número elevado de aeronaves se hace muy complejo controlar que no existan colisiones entre ellos. Para evitar esto, en muchos de los casos se opta por un despegue secuencial de cada uno de ellos hasta que estos alcancen una determinada altura, o una cierta posición en el aire. Sin embargo, esta solución tiene como gran inconveniente la cantidad de tiempo invertida en el proceso de despegue.

La principal motivación que impulsó la realización del presente proyecto fue la creación de un algoritmo de código libre que logre minimizar el tiempo total asociado al proceso

de despegue, el cual permita beneficiar a cualquier usuario que desee realizar una misión con un número de aeronaves considerable.

1.2 Objetivos

Este proyecto tiene como objetivo principal desarrollar un nuevo algoritmo para los enjambres de drones capaz de optimizar el tiempo transcurrido en la etapa de despegue, y que evite cualquier colisión. Para ello, inicialmente emplearemos el *Kuhn-Munkres Algorithm (KMA)* [3] para obtener la solución óptima referente a la distancia entre las posiciones terrestres de los drones con sus respectivas posiciones en el aire y porque minimiza las colisiones existentes. A partir de estos datos de entrada podremos determinar qué multicópteros colisionan con otros y, para prevenir esto, formaremos un listado de tandas en las que se garantice que todos los drones pertenecientes a uno de estos grupos pueden despegar simultáneamente, y sin causar ningún problema.

El segundo gran objetivo de este proyecto es realizar un análisis exhaustivo de las prestaciones alcanzadas relativas al tiempo de ejecución y al número de colisiones. Posteriormente, se efectuará una comparación de los resultados obtenidos con los otros algoritmos de despegue de enjambres existentes, permitiendo determinar así cuál de ellas ofrece un mayor rendimiento. Para llevar a cabo esto, se ejecutarán diferentes experimentos empleando diferentes criterios para comprobar que el algoritmo desarrollado es, entre otras cosas, escalable con el número de drones.

Por último, con el algoritmo de detección de colisiones totalmente perfeccionado, se estudiarán los resultados obtenidos con los diferentes algoritmos de asignación existentes para ver la efectividad del método propuesto.

1.3 Estructura

El presente Trabajo de Fin de Máster consta de siete capítulos. A continuación, vamos a analizar los principales bloques de los que se compone este documento:

- **Capítulo 1. Introducción.** En este primer capítulo se plantean las motivaciones y los objetivos de la problemática a resolver en el presente proyecto.
- **Capítulo 2. Estado del arte.** En este capítulo realizaremos un análisis de las publicaciones relacionadas directamente con los enjambres de drones y su etapa de despegue.
- **Capítulo 3. Vehículos aéreos no tripulados.** En dicho capítulo se llevará a cabo una descripción general de los diferentes tipos de drones existentes, los componentes hardware que los integran, y su software.
- **Capítulo 4. ArduSim.** En esta sección se mostrará por qué se ha elegido el simulador de ArduSim, y cuáles son sus principales características.
- **Capítulo 5. Desarrollo.** En tal capítulo se mostrarán las distintas versiones del protocolo de detección de colisiones, y se detallarán sus características principales.

- **Capítulo 6. Experimentos y resultados.** En dicha sección se expondrán los diferentes experimentos efectuados, conjuntamente con los resultados que se han obtenido.
- **Capítulo 7. Conclusiones.** En este último capítulo se presentarán las conclusiones de esta tesis de máster e incluimos algunas ideas para los trabajos futuros.

Capítulo 2

Estado del arte

2.1 Enjambres de drones

Junto con la cuarta revolución industrial, los vehículos aéreos no tripulados se encuentran en un periodo de esplendor abarcando una gran cantidad de sectores en nuestra sociedad. Ya no resulta extraño observar en los medios de comunicación el uso de estas aeronaves, ya sea de modo individual o en conjunto, para ciertos encargos en los que suponen algún tipo de riesgo para los humanos, o simplemente para ofrecer una actividad de ocio en los puntos de referencia de las grandes ciudades.

Con la finalidad de dar respuesta a esta etapa tecnológica, el reciente desarrollo de pequeños vehículos aéreos no tripulados compactos y económicos posibilita su empleo en modo de enjambre para conseguir una eficiencia mayor en sus tareas. Estos grupos de drones cuentan con un tipo de misión específica en la que sus aeronaves serán las encargadas de llevarlas a cabo mediante tareas individuales.

Antes de empezar a hablar de los estudios efectuados en torno a los enjambres de drones, conviene en primer lugar comentar brevemente cuáles fueron sus inicios. Para ello, tenemos que retroceder hasta el siglo XIX, cuando las fuerzas de defensa de Israel (FDI) fueron las pioneras en usar un grupo de drones con el propósito de localizar, identificar y atacar a los militantes pertenecientes a la población de Gaza [4]. A partir de este momento, las diferentes naciones del mundo intentaron poner sus esfuerzos en la investigación de las posibles aplicaciones de las aeronaves y en su funcionamiento colaborativo para mejorar la vida de sus ciudadanos.

De vuelta a la actualidad, debido a la gran accesibilidad de los vehículos aéreos autónomos, muchos centros de investigación y empresas de ámbito privado han permitido llevar a cabo una gran cantidad de estudios para abarcar los distintos ámbitos del mercado. Sin embargo, todavía existen muchas vertientes abiertas que son el objetivo de multitud de estudios. A continuación repasaremos algunos de los avances llevados a cabo en los últimos años.

Uno de los tópicos más populares y sensibles en los enjambres de drones incluyen la comunicación entre los VANTs. Aunque estas aeronaves operan normalmente solas en la actualidad, diversos estudios apuntan a la posibilidad de que estas formen una red vehicular ad-hoc, también conocida como *Vehicular Ad-Hoc Network (VANET)*, en la que cada una de ellas efectúa la función de nodo que se conecta a la red sin la necesidad de usar ninguna infraestructura adicional. De forma periódica, todos los VANTs intercambian paquetes de datos ágilmente entre sí para efectuar una toma de decisiones inteligente y completamente autónoma. A medida que transcurre el tiempo, la cantidad

máxima de aeronaves que un enjambre puede contener se encuentra en continuo auge, llegando a alcanzar el millar de unidades. Del mismo modo que dicha cifra incrementa, también lo hace la complejidad de la VANET que se debe emplear debido a que su rendimiento puede aminorar por la pérdida de paquetes de información o del tráfico existe en un momento determinado. Para evitar estas situaciones de saturación resulta conveniente aplicar ciertas medidas de fiabilidad que nos garanticen el correcto funcionamiento de la red a utilizar. Li y Bai [5] efectuaron una propuesta para la evaluación de la resiliencia de un enjambre considerando los límites de comunicación inalámbrica y la confiabilidad de la transmisión de datos entre sus nodos. En este estudio se propone un modelo de enjambre de drones en el que se intenta mitigar el efecto de la distancia tras los continuos cambios dinámicos de los nodos y, así, reducir las apariciones de posibles amenazas correspondientes a las principales causas de fallos.

Siguiendo esta línea, Campion y Ranganathan [6] presentaron una nueva infraestructura de comunicación inalámbrica para enjambres de drones mediante la utilización de redes celulares. Con esta estructura se consiguen aliviar algunos de los factores limitantes, como es el caso del alcance por los cuales los VANTs pueden comunicarse. De esta manera, se incrementaría la eficiencia de la red y se abriría el horizonte hacia el uso comercial donde los usuarios podrán comunicarse con dichos dispositivos a través de las redes 5G. No obstante, tal trabajo tiene como limitación la dependencia de tener cobertura de datos en los teléfonos móviles, imposibilitando su utilización en ciertas áreas rurales o en países subdesarrollados.

Paralelamente a estos avances, también creció la motivación por perfeccionar la prevención de obstáculos en la navegación autónoma de un enjambre de drones. Yasin y Haghbayan [7] presentaron un protocolo de estrategia de control de múltiples prioridades, aplicable para todo tipo de aeronaves, con la intención de evitar posibles conflictos y, así, lograr un excelente control de la formación en todo momento. De este modo, las aeronaves disponen de la capacidad de tomar decisiones de forma autónoma para sortear los distintos obstáculos que se encuentren en su camino. Además, se incluyen medidas para que el resto de drones se esperen en determinados puntos durante un periodo de tiempo considerable para mantener la formación que se le había indicado inicialmente.

La búsqueda de mecanismos centrados en el aumento de la resiliencia de los enjambres de drones siempre ha sido objeto de investigación. No hay que olvidar que, como todo dispositivo electrónico, no se puede asegurar con total certeza que sus componentes hardware no vayan a fallar nunca. Ya sea por este motivo, o por el simple hecho de que un dron deje de comunicarse con el resto, se deberá garantizar que el resto de drones del enjambre puedan ejecutar la misión con éxito. Cheng y Bai [8], plantearon un marco orientado a la evaluación de la resiliencia para un enjambre de VANTs. De este modo, dicha utilidad nos facilita conocer la resistencia ante posibles amenazas, y cuáles son las adaptaciones óptimas ante eventos disruptivos. Además, este marco se puede adaptar a los diferentes tipos de misiones existentes, proporcionando una forma flexible de analizar la resiliencia del enjambre.

Otro de los problemas, el cual ha sido objetivo de múltiples investigaciones, es mejorar la eficiencia de las misiones planeadas para los enjambres de drones. En el trabajo de Zhang y Feng [9] se lleva a cabo un profundo análisis de la planificación de las tareas complejas que las aeronaves deben tomar en tiempo real ante entornos dinámicos. En su propuesta deciden emplear el algoritmo CBBA (del inglés *Consensus-Based Bundle Algorithm*), empleado en problemas de asignación de múltiples tareas. Tras estudiar sus resultados se consigue obtener un gran rendimiento, incluso teniendo en cuenta limita-

ciones como la sincronización de mensajes o el tiempo entre las tareas.

De cara a ampliar el número de aplicaciones que se pueden otorgar a los enjambres de drones, se plantearon muchas propuestas con el fin de satisfacer una nueva posible necesidad. Fabra et al. [10] presento un algoritmo de coordinación de los VANTs pertenecientes a un enjambre en la que se distinguían dos roles: mientras existía una aeronave líder que era controlada por un piloto real, el resto de drones siguen sus movimientos en tiempo real. Este algoritmo proporciona soluciones óptimas en misiones consistentes en las operaciones de búsqueda y rescate o vigilancia de zonas extensas.

Por último, me gustaría hacer una recapitulación de las actividades de ocio más recientes en las que han participado un número considerable de drones. Sin ir más lejos, durante los juegos olímpicos de Tokio de 2020 (celebrados en 2021 debido a la pandemia), pudimos ver en su ceremonia inaugural como un enjambre con un total de 1874 drones sobrevolaban el estadio olímpico formando un globo terráqueo luminoso [11]. Este espectáculo se llevó a cabo por la prestigiosa empresa Intel, y el software de la operación era controlado por una computadora de altas prestaciones. Siguiendo la misma línea de actividades de entretenimiento, en septiembre del 2020 la ciudad rusa de San Petersburgo celebró el 75 aniversario del fin de la Segunda Guerra Mundial con un espectáculo de luces aéreas protagonizado por 2.200 drones [12]. En este caso, no es posible confirmar si se trataba de un enjambre o no debido a la privacidad del proyecto, pero lo más probable es que la coordinación se haya llevado a cabo también de manera centralizada.

2.2 Algoritmos de despegue

Tras múltiples estudios referentes al despegue de enjambres de drones, a día de hoy sigue habiendo cierta incertidumbre respecto a qué procedimientos usar. De tales dudas es donde surge el actual proyecto, en el cual se va a mostrar una solución que evite cualquier posible conflicto aéreo y, asimismo, permita mejorar el tiempo total en el que todas las aeronaves alcancen su posición aérea predeterminada. Sin embargo, previo a la elaboración del estudio actual, existe una serie de trabajos precedentes cuyo análisis resultará trascendental para su correcta ejecución.

En lo que se refiere a la fase de despegue, básicamente existen dos modos: el secuencial y el simultáneo. Mientras que el primero persigue evitar colisiones a cambio de un tiempo total muy elevado, el segundo provoca justamente lo opuesto. Dicho esto, muchas investigaciones han tratado de afrontar el reto que supone obtener un protocolo que permita prevenir cualquier contratiempo aéreo con la mejor eficiencia.

Fabra et al. planteó, en su trabajo [13], una heurística que obtenía una solución para el problema de asignación de posiciones de un enjambre de drones que dispongan de la habilidad de despegar y aterrizar verticalmente, conocida también por el término *Vertical take off and landing (VTOL)*. Con este mecanismo a cada localización terrestre se le asignaba una posición aérea, la cual obtenía una distancia total de desplazamiento casi óptima. Aunque aplicando tal heurística no garantíamos la prevención de colisiones, dicha solución proporciona un mayor seguridad, y reduce el tiempo de despegue en comparación con un enfoque aleatorio (sin ningún criterio).

Con el objetivo de optimizar más aún el problema de asignación, Hernández et al. [3] propuso un nuevo esquema de despegue basado en el KMA consistente en la aplicación de teorías de grafos. Tras la implementación de su propuesta, se consiguió ofrecer una solución óptima respecto a la distancia total de desplazamiento que tiene que ejercer el

enjambre. A pesar de lograr mejores resultados que el algoritmo presentado en el párrafo anterior, seguía sin asegurar la eliminación total de zonas de conflicto aéreas.

Una vez resuelto el problema de asignación, el siguiente paso a seguir es conseguir diseñar un algoritmo con el que garantizar que no existe ningún riesgo de colisión. Wubben, en su tesis de máster [14], llevó a cabo un minucioso análisis del comportamiento de los algoritmos de asignación comentados en los dos últimos parágrafos. En sus conclusiones, a pesar de que el algoritmo de Kuhn-Munkres tiene un tiempo de cómputo superior a la heurística, el tiempo total de despegue obtenido en sus múltiples experimentos era siempre menor. Además, en el mismo trabajo, Wubben implementó un procedimiento de despegue (semi-secuencial) en el cual únicamente dos aeronaves podían despegar en un mismo momento. Empleando esta técnica se logra mantener la seguridad en el enjambre, pero disminuyendo considerablemente el tiempo de despegue.

2.3 Simulador

Generalmente, todos los estudios de enjambres de drones requieren de un simulador con el que probar sus experimentos. La función del simulador resulta esencial, ya que nos permite conocer de antemano si existen errores en nuestro software y, de este modo, ahorrarnos tiempo y dinero en caso de producirse alguna colisión entre aeronaves. Aunque no existe un simulador definitivo que nos permita reproducir las mismas condiciones reales que se pueden encontrar en un momento dado, existe una gran cantidad de propuestas cuyo análisis nos será de gran ayuda. Procedemos a comentar dos de ellas.

Page fue uno de los pioneros en crear un simulador de enjambre autoorganizado. En su trabajo [15] pretendía poner fin a la incertidumbre proporcionando un simulador de VANTs el cual denominó simulador clúster. Su finalidad era ofrecer la flexibilidad de simulación de las redes de nodos con movilidad para estudiar su rendimiento. En su estudio incluía una interfaz gráfica en el que poder ver la simulación, pero los parámetros de simulación eran bastantes limitados, ya que no se podían cargar las coordenadas terrestres ni poner una cantidad elevada de drones.

Por último, recientemente Jeroncic [16] publicó su trabajo consistente en la creación de un simulador preciso en tiempo real, combinándose con una alta calidad gráfica. Para su implementación se empleó el complemento AirSim, el cual permitía un movimiento suave y buenos gráficos gracias a al proceso de renderizado. Sin embargo, tal y como comenta el autor, su principal problema era su escalabilidad, ya que únicamente funcionaba bien para un número de seis VANTs. Esta cuestión supone una clara limitación actualmente debido a que tal cifra de aeronaves es muy baja, y la mayoría de proyectos que se proponen van destinados a enjambres mucho más ambiciosos.

Capítulo 3

Vehículos aéreos no tripulados (VANTs)

En este capítulo se realizará una descripción general del estado actual de los VANTs. Aunque con este tópico podríamos profundizar sobre una infinidad de detalles, como por ejemplo los protocolos de comunicación entre drones [17], o los distintos tipos de hélices [18], se ha decidido segmentar este capítulo en tres subcategorías donde se expondrán cuáles son los tipos de drones más frecuentes, su software, y sus componentes hardware.

3.1 Tipos de drones

Actualmente, podemos encontrar una multitud de drones disponibles en el mercado. Aunque el término dron se suele utilizar también para referirse a aquellos vehículos que circulan por el medio terrestre y marítimo, en esta sección nos centraremos en cuáles son las características que nos permiten distinguir inequívocamente los VANTs.

La principal clasificación de los drones la obtenemos según el modo en el cual se sustentan en el aire. Dentro de esta agrupación encontramos dos categorías principales: las aeronaves de ala fija y las de ala rotatoria [19]. En la Tabla 3.1 podemos ver una pequeña comparación entre las principales características de ambas categorías.

Tabla 3.1: Comparación entre los diferentes tipos de drones.

	Ala fija	Ala rotatoria
Área cubierta	Alta	Baja
Área de aterrizaje	Grande	Muy pequeña
Tiempo de vuelo y resistencia al viento	Medio/Alto	Medio/Bajo
Velocidad media	Medio/Alta	Medio/Baja
Precio	Medio/Alto	Bajo/Medio
Vuelo estático suspendido	No	Si
Dificultad en acceder a zonas inaccesibles	Alta	Baja
Aplicaciones	Vigilancia de fronteras, militares	Agricultura, inspección, topografía

Resulta importante recalcar que el algoritmo de despegue de enjambres de drones que se va a desarrollar en este proyecto es exclusivo para el uso de aeronaves de ala rotatoria debido a que son los únicos que disponen de la capacidad VTOL. El motivo de esta elección es que los drones de ala fija requieren de una pista para despegue, o una persona que los lance al aire para que puedan despegar, y no procedería determinar las posibles colisiones entre ellos usando el mismo método.

Dentro de la sección de drones con ala rotatoria encontramos otra clasificación según la cantidad de alas que posee. Si la aeronave dispone dos hélices pertenece al grupo de los helicópteros, mientras que si cuenta con más de dos pertenece a la agrupación de los multicópteros.

En la sección de multicópteros podemos encontrar distintos modelos según su número de motores. Los drones típicos suelen disponer de un motor para cada brazo. El modelo más común es el que incorpora cuatro hélices (quadcóptero), pero existen los que cuentan con seis, ocho e incluso dieciséis. Aunque el número mínimo de hélices es tres, estos suelen ser muy raros, y la mayoría de ellos son diseñados para usos recreativos.

Según la cantidad de motores que contiene un dron le permitirá realizar de manera más óptima unas tareas u otras. Por ejemplo, en el caso de una aeronave con pocos motores, esta será más ligera, lo que le permitirá desplazarse más rápidamente. Por lo contrario, dicha aeronave será más inestable, y el fallo de uno de sus motores resultaría trágico para el vehículo. En el caso opuesto tenemos el de una aeronave con una cantidad de motores considerable, lo cual le permitiría llegar a soportar cargas con un mayor peso, y también posee una mayor estabilidad.

Aunque el número de hélices usadas por los multicópteros que podemos encontrar en el mercado varía según los valores que se han comentado en los párrafos anteriores, existen diferentes configuraciones para cada número de hélices que dispone la aeronave. En la Figura 3.1.1 se pueden observar diferentes configuraciones de las hélices de multicópteros, las cuales permitirían a las aeronaves volar con total normalidad.

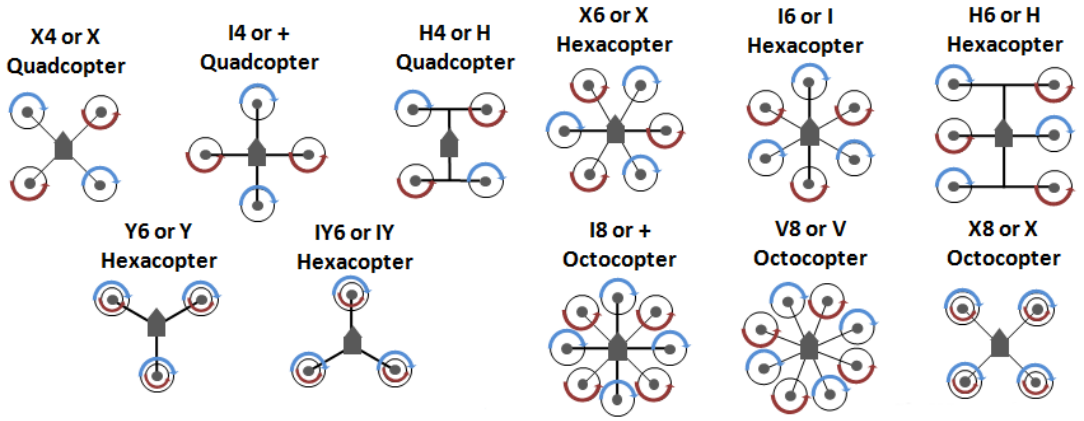


Figura 3.1.1: Posibles configuraciones en los multicópteros.

Como ya se ha mencionado en esta sección, este proyecto se centra en los multicópteros porque son los únicos drones capaces de realizar un despegue vertical. Además de esta ventaja, el precio de los multicópteros es más asequible que otras aeronaves (para tareas similares), lo cual les hace ideales para lograr nuestros objetivos.

3.2 Hardware

En la actualidad existe una gran variedad de materiales y formas asociados a los multicópteros en el mercado. Sin embargo, existe una serie de elementos hardware que son imprescindibles en todas las aeronaves (ver Figura 3.2.1). El componente más importante de cualquier dron es el controlador de vuelo. El controlador de vuelo es simplemente una placa de circuitos que se encarga de múltiples tareas, como por ejemplo el ajuste de la velocidad de los motores, centralizar el control de cualquier sensor integrado en el dispositivo, o la recepción de los comandos de usuario, entre muchas cosas más. Entre la multitud de diferentes controladores de vuelo existentes, el conocido como Pixhawk [20] es uno de los más empleados al ser *open-source*, y debido a su gran flexibilidad y confiabilidad a la hora de controlar las aeronaves.

Como ya se ha mencionado anteriormente, los multicópteros necesitan disponer de un mínimo de tres motores con sus respectivas hélices. A menudo, los motores empleados en los multicópteros utilizan la tecnología *brushless*, lo cual implica usar controladores de velocidad electrónicos (ESC, del inglés *Electronic Speed Controller*). Estos componentes tienen como principal función el poder regular la velocidad de giro del motor a partir de los pulsos enviados desde el controlador de vuelo.

Junto al controlador de vuelo encontramos el panel de distribución de energía, que será el encargado de proporcionar la energía suficiente para que los motores y ciertos sensores puedan funcionar correctamente.

Las baterías más populares para los drones son las de tipo Li-Po, y se emplean principalmente por dos motivos. El primero de ellos es que son ligeras y su tamaño es pequeño, y, en segundo lugar, permiten usar grandes cantidades de energía en períodos de tiempo muy cortos [21]. Normalmente, una batería Li-Po cuenta con unas tres o cuatro celdas, donde cada una de ellas tiene un voltaje nominal de unos 3.7 V. La capacidad de la batería de un multicóptero suele ser de unos 3000 *mAh*, y este factor es uno de los más influyentes en el tiempo máximo de vuelo de una aeronave. Sin embargo, si montáramos una batería de doble capacidad, esto no duplicaría el tiempo en el que el dispositivo puede

estar planeando por diferentes factores. El primero de ello es que estaríamos aumentando el peso del multicóptero, lo cual supondría que nos costaría más mantenerlo en el aire y, a su vez, aumentaríamos el consumo de energía. Luego, existen otros factores como el tipo de hélices y motor empleados, o las condiciones del tiempo, que pueden hacer variar nuestro tiempo de vuelo.

Otro componente típico que suelen disponer todos los multicópteros son los receptores. Estos se encargan de recibir la información, a través de su antena, de un transmisor, y envían dichos datos recibidos al controlador de vuelo, quien decidirá qué acción tomar. Aunque para controlar un dron normalmente se emplea un mando de control remoto, también se suele usar un programa de control propio (estación de tierra), el cual se encarga de recibir los datos de telemetría, y enviar a los multicópteros todas las acciones/instrucciones requeridas. En nuestro caso, usaremos el simulador ArduSim para llevar a cabo dicha operación, cuyo análisis se describirá en el capítulo siguiente.

Profundizando en la comunicación entre VANTs y la estación de control terrestre, a cada aeronave se le incorpora una placa base Rasberry Pi equipada con un adaptador WiFi (con conectividad 5 Ghz). De este modo, cualquier dron dispondrá de lo necesario para utilizar la red WiFi ad-hoc creada por la totalidad de dispositivos que componente el enjambre. Además, para la telemetría se emplea como transmisor una antena que opera en la banda de 433 MHz en Europa.

Por último, conectados al controlador de vuelo podemos encontrar una gran diversidad de sensores destinados a diferentes usos. Entre algunos de los más frecuentes existen los sensores de GPS, giroscopio o acelerómetro. Otros, como es el caso de las cámaras, suelen ser más característicos en modelos centrados en la inspección o topografía.

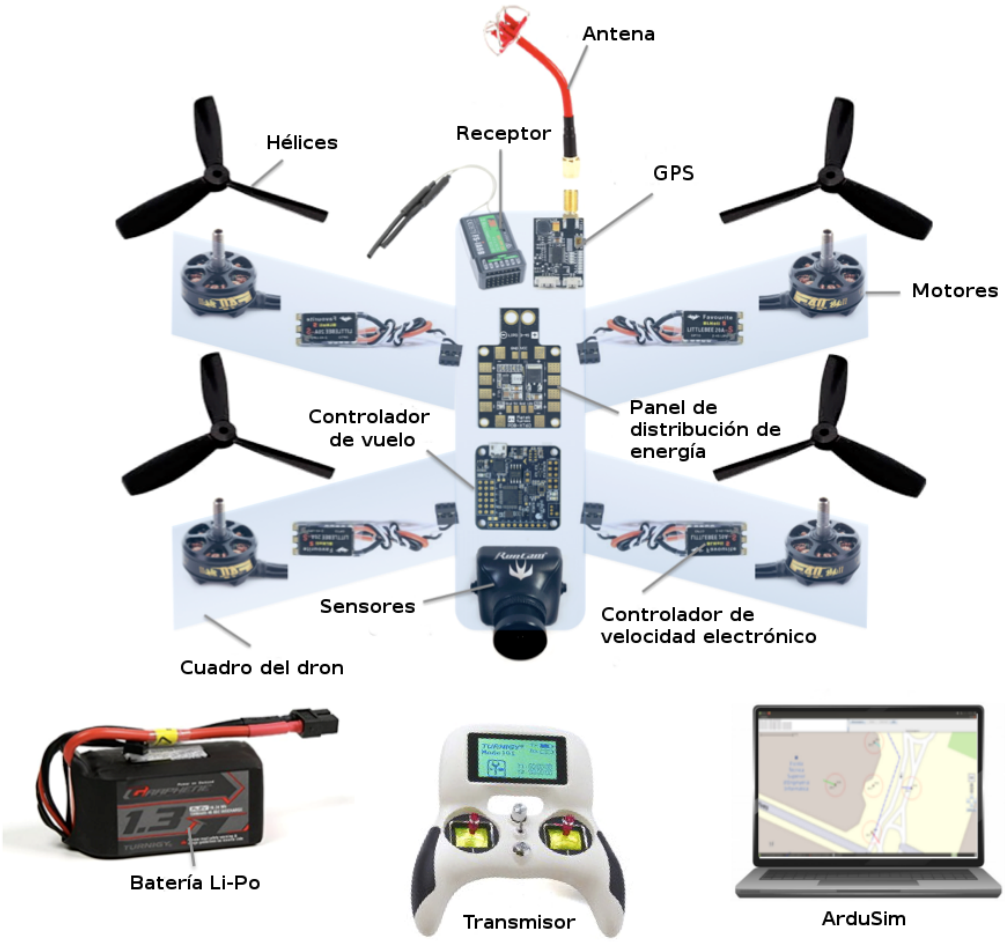


Figura 3.2.1: Componentes básicos de los multicópteros.

3.3 Software

Como se ha visto en la sección anterior, el controlador de vuelo es el cerebro de cualquier multicóptero debido a que se encarga de controlar todas sus funciones vitales para que este pueda funcionar correctamente. Dicho esto, resulta fundamental que el firmware que se encuentre dentro del controlador de vuelo deba ser el más eficiente acorde al modelo de aeronave que estemos empleando. Uno de los firmware más famosos que nos encontramos es ArduPilot porque, aparte de ser de código libre y estar constantemente actualizado, también funciona correctamente en una gran variedad de hardware, entre ellos Pixhawk, para el control de drones de todo tipo. Dentro de ArduPilot encontramos el proyecto ArduCopter [22], que es el específico para el control de los drones. Algunas de las características que podemos destacar de este firmware son las siguientes:

- **Amplio rango de modos de vuelo.** ArduCopter ofrece hasta 23 modos de vuelo en los cuales el usuario puede probar diferentes niveles de estabilidad.
- **Misiones autónomas.** ArduCopter puede realizar automáticamente misiones complejas basadas en puntos de referencia GPS. Además, tareas como el despegue o el aterrizaje también son posibles con este firmware.

- **Simulador con alta precisión.** ArduCopter nos ofrece un simulador el cual nos facilita testear los algoritmos y protocolos diseñados previamente a su uso en un entorno real. Su principal inconveniente es que solo nos permite simular el vuelo de un único dron.

- **Código libre.** En ArduCopter no existe ningún bloqueo del proveedor, es decir, es totalmente abierto a los usuarios. También cuenta con una gran comunidad de desarrolladores.

Por último, resulta ser de suma importancia mencionar que el software desarrollado se comunica con el dron mediante el empleo del protocolo *Micro Air Vehicle Link (MAVLink)* [23]. MAVLink es publicado bajo la licencia LGPL (del inglés *Lesser General Public License*), y es uno de los protocolos más usado en los vehículos no tripulados de cualquier medio. La principal característica de este protocolo es que es muy ligero, lo cual lo hace idóneo para su uso en estas pequeñas aeronaves. Entre sus múltiples características podemos destacar su fiabilidad, ya que proporciona métodos para la autenticación de paquetes, y eficiencia con sistemas que disponen de un ancho de banda de comunicación muy limitado. La versión más reciente de MAVLink es la 2.0 que fue publicada en el 2017, y tuvo como novedades más destacables el aumento a 24 bits en el identificador del mensaje, y la incorporación de más protección en los mensajes de autenticación.

Capítulo 4

ArduSim

Para cumplir el objetivo principal del presente trabajo se requiere realizar múltiples pruebas en un entorno de simulación con el que poder comprobar la inexistencia de colisiones antes de realizar experimentos en drones reales. Esta fase es una de las más importantes de cualquier proyecto destinado a los enjambres de drones, ya que una gestión precipitada podría suponer la pérdida de una gran cantidad de tiempo y dinero en caso de que dichas aeronaves sufriesen algún tipo de avería. ArduSim [1] es el simulador de vuelo en tiempo real que se ha decidido emplear en este proyecto porque nos permite simular misiones con un número considerable de drones (más de 500), y también nos ofrece una gran precisión en los datos recogidos. ArduSim está desarrollado en Java, y es totalmente de código libre [24]. Está basado en el simulador SITL (del inglés *Software in the Loop*), donde cada dron simulado en ArduSim dispone de una instancia de SITL que les permite simular sus propiedades de vuelo, como por ejemplo su posición terrestre, en la etapa de despegue.

En la Figura 4.0.1 observamos la arquitectura interna de ArduSim. En ella apreciamos que cada multicóptero dispone de tres hilos: uno para recibir mensajes, otro para enviarlos, y un tercero para efectuar las tareas lógicas del protocolo con el controlador del dron. También vemos que el simulador posee de una interfaz gráfica de usuario (GUI) en la cual podemos observar los experimentos propuestos. Por último, comentar que ArduSim contiene algunas funcionalidades que le permiten detectar las posibles colisiones entre multicópteros, y ofrece un sistema de difusión de mensajes entre todos los drones.

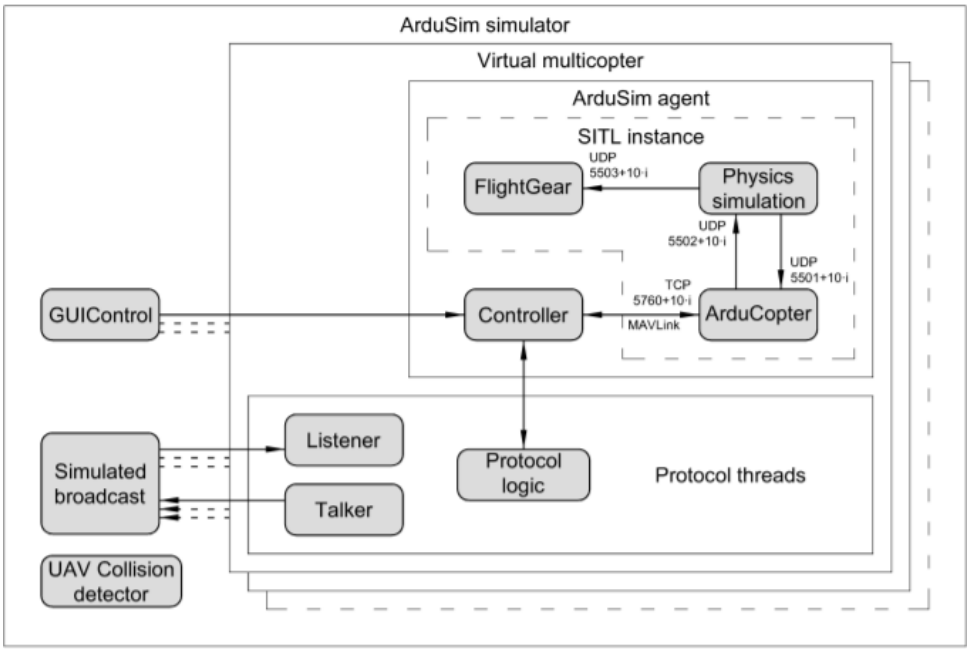


Figura 4.0.1: Arquitectura interna de ArduSim [1].

En lo que respecta a la comunicación entre los multicópteros, ArduSim se encarga de emular una red WiFi en la banda de los 5 GHz. Como ya se mencionó en el capítulo anterior, los drones se comunican entre sí mediante el uso del protocolo MAVLink. Otra propiedad característica de ArduSim es que el código desarrollado se puede ejecutar directamente en drones reales. En este caso, si se desea llevar a cabo algún experimento en un entorno real, se requiere incorporar en los drones de dos elementos hardware adicionales. El primero de ellos es el adaptador WiFi, el cual permite a las aeronaves conectarse a una red Ad-hoc formada por todos los drones que realizan una determinada misión. Este componente se emplea para comprobar las posibles pérdidas de mensajes entre drones o estaciones de control terrestres cuando estos se encuentran a una distancia considerable. En segundo lugar, se emplea una placa de computación (las más frecuentes son las Raspberry Pi) capaz de ejecutar el ejecutable Java de ArduSim. En general cualquier placa que ofrezca la potencia suficiente para ejecutar ArduSim con facilidad, y que sea lo ligera para minimizar la potencia que los drones necesitan para volar, sería una opción viable.

Otra característica destacable de ArduSim es su eficiencia a la hora de gestionar la gran cantidad de datos generados en sus simulaciones. Cada vez que se finaliza un experimento en ArduSim, los datos obtenidos se almacenan en múltiples ficheros clasificados por su función, y en los que se pueden obtener propiedades como la velocidad o las coordenadas de los drones, entre otras cosas más. Esto supone una inmensa ventaja ya que, en el momento de la depuración de código, nos podemos ahorrar mucho tiempo. Además, esto hace con que el aprendizaje de los nuevos usuarios sea muy rápido porque les permite localizar con cierta sencillez donde se producen errores.

Por último, un usuario puede llevar a cabo una simulación mediante el uso de dos interfaces: con la línea de comandos o con la interfaz gráfica. Aunque la primera nos puede resultar útil para la ejecución de determinados ficheros de prueba, asignándoles o no argumentos adicionales, la modalidad gráfica es la más comúnmente usada, ya que nos permite ver si los objetivos de la misión se han logrado o no. Con esta interfaz, nada más ejecutar la clase principal de ArduSim, se nos abre una pantalla con todos los parámetros de simulación existentes (ver Figura 4.0.2) en los que un usuario podrá modificarlos a su

gusto. Entre todos los parámetros existentes podemos destacar dos en concreto que van a ser imprescindibles en el presente proyecto. El primero de ellos es el protocolo que se va a emplear durante la simulación. El protocolo empleado será siempre el conocido como *Take Off*. Este consiste en que los drones pertenecientes a un enjambre deben despegar hasta alcanzar una determinada posición aérea para posteriormente poder realizar todo tipo de misiones. El segundo conjunto de parámetros a destacar son los que están relacionados con la detección de colisiones. En el caso de habilitar dicha opción, cuando se efectúe una simulación con una misión específica, y se detecte una colisión entre aeronaves, el progreso se detendrá informando al usuario de que ciertos drones han colisionado.

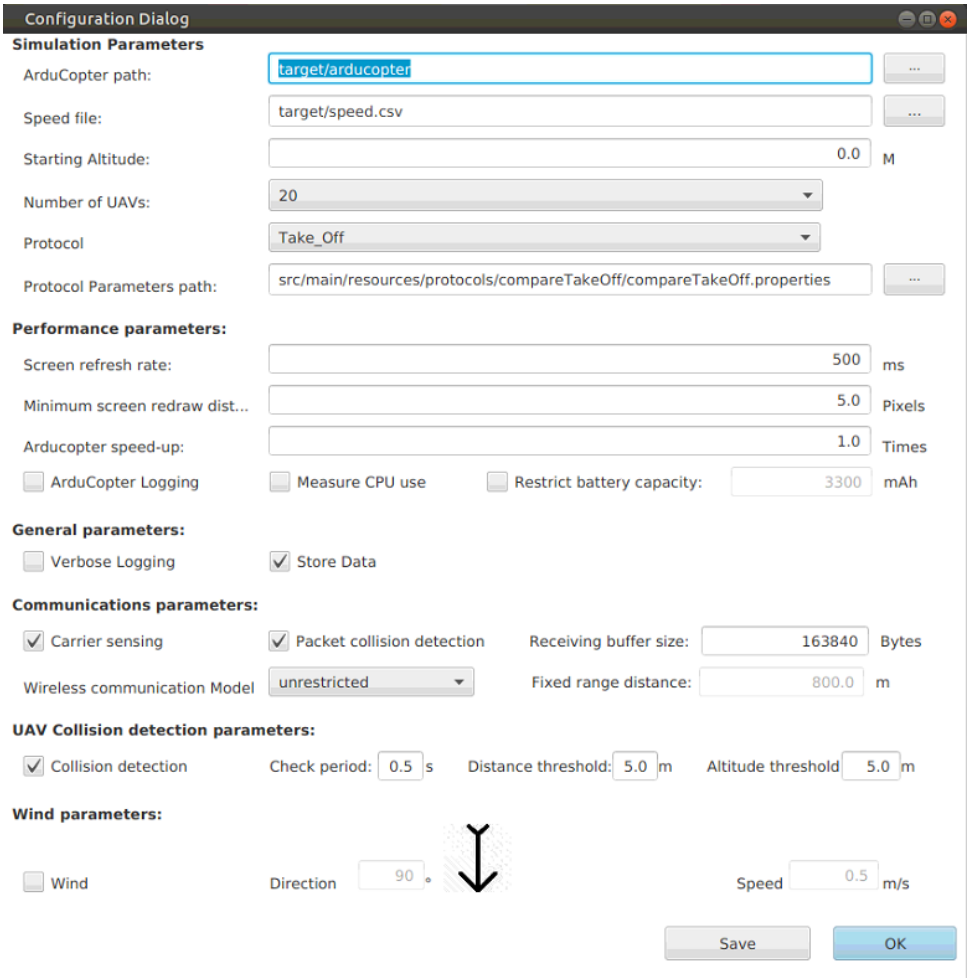


Figura 4.0.2: Pantalla de selección de parámetros generales de ArduSim.

Una vez seleccionados los parámetros deseados y confirmados sus cambios con el respectivo botón, nos aparecerá una nueva pantalla de configuración del protocolo de simulación elegido en la pantalla anterior (ver Figura 4.0.3). Aquí se nos permite seleccionar las posibles formaciones terrestres y aéreas de los drones, junto a su respectivo algoritmo de asignación.

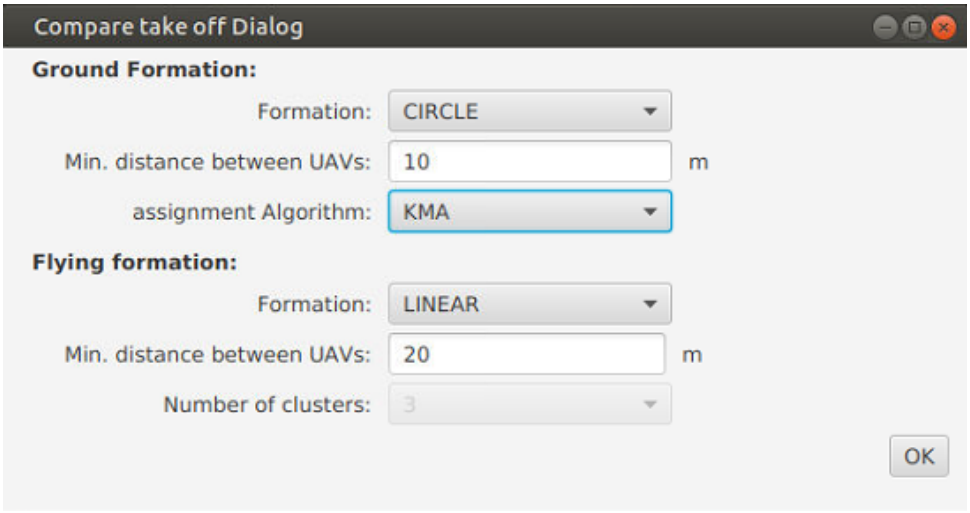


Figura 4.0.3: Pantalla de configuración del protocolo.

Tras esto, el usuario ya podrá visualizar la pantalla principal de ArduSim (Figura 4.0.4). Dicha pantalla la podemos dividir en tres partes. En la parte central encontramos el número de drones elegido dibujados encima de un mapa. Cuando cada aeronave se mueve por tal mapa, dicha trayectoria es representada en forma de línea gruesa de color, y se muestra su ruta a recorrer con una línea discontinua. En la esquina superior derecha se pueden apreciar los distintos controles que nos ofrece ArduSim para efectuar la simulación, mientras que en la esquina superior izquierda se muestra información específica sobre cada dron, como por ejemplo su modo de trabajo o su altitud.

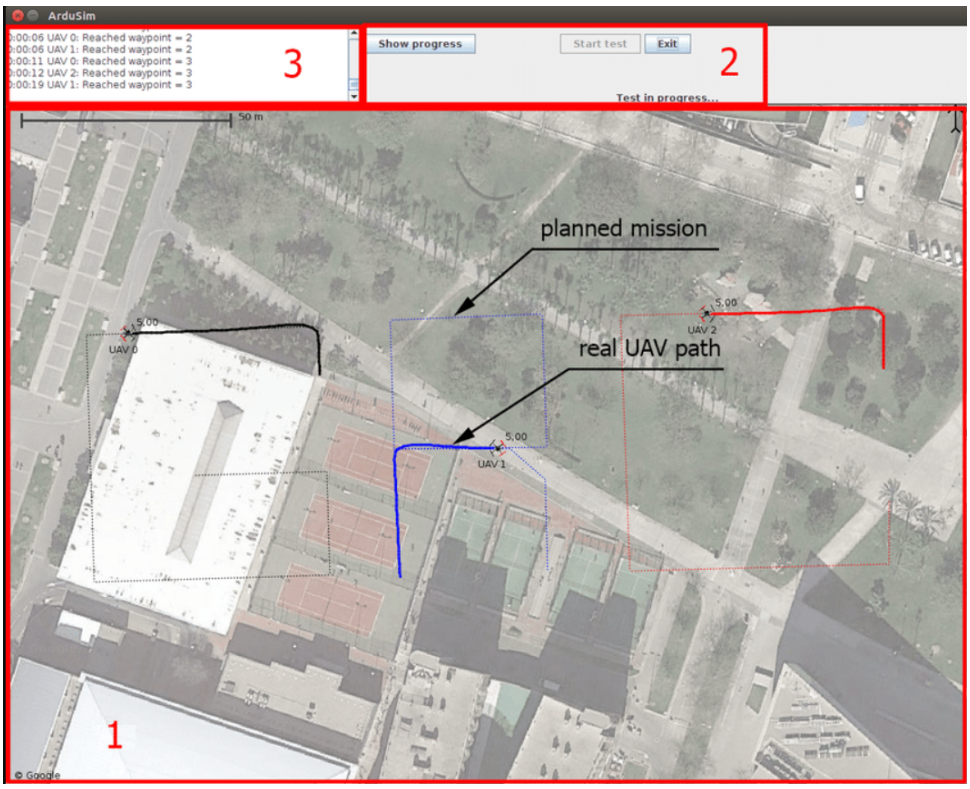


Figura 4.0.4: Pantalla principal de ArduSim.

Capítulo 5

Desarrollo

En el presente capítulo se llevará a cabo un minucioso análisis de los diferentes algoritmos propuestos referentes al tiempo de cómputo. También se explicará la estrategia empleada para agrupar las aeronaves en tandas en las que garanticemos que, entre todos sus integrantes, no se produce ningún riesgo de colisión. Por último, se narrará el procedimiento empleado para el despegue de los drones del enjambre, y cuáles son los mensajes de comunicación más esenciales en dicha fase.

5.1 Conceptos previos

Antes de entrar en detalle con el algoritmo de cálculo, resulta necesario comentar una serie de conceptos básicos de los que se parte, y que necesitamos entender para llevar a cabo una correcta implementación del algoritmo de despegue.

Gestión de maestros y esclavos. Una de las principales consideraciones de las que se debe tener constancia es que, dentro de un enjambre, existen dos tipos de drones según su funcionalidad: el maestro y el esclavo. El maestro es el que asume toda la iniciativa de comunicación para poder completar la misión del enjambre de forma correcta, mientras que los esclavos se encargan de seguir las órdenes del maestro. Aunque depende de las distintas formaciones que tomen los multicópteros en el aire, el maestro normalmente se sitúa en una localización central para disminuir la posible pérdida de mensajes en el canal inalámbrico debido a la distancia entre drones. En fases preliminares de investigaciones con enjambres de drones, el sistema maestro-esclavo resultaba ser algo ineficiente ya que, si el líder tuviese una avería y dejara de comunicar con el resto de aeronaves, éstas no podrían llevar a cabo su misión, o simplemente no aterrizarían hasta alcanzar el fin de la energía proporcionada por sus baterías. En la actualidad, se ha perfeccionado el funcionamiento del sistema de maestro-esclavo proporcionando una mayor fiabilidad y robustez en los casos donde el maestro deja de comunicarse con el resto de integrantes del grupo. Una de las propuestas realizadas en [25] consiste en que los esclavos comprueban cada cierto tiempo si el maestro está activo. Si no se da el caso, los esclavos proceden a ejecutar un proceso de votación para determinar cuál es el nuevo líder y, de este modo, proseguir la misión con total normalidad.

Detección de nodos. Previamente a iniciarse la fase de despegue del conjunto de drones que conforman un enjambre, el maestro inicia la etapa de descubrimiento de los

drones que van a efectuar la misión. En este punto, el líder enviará un mensaje de petición para obtener la cantidad exacta de los drones disponible. Cada dron que vaya a participar en el experimento deberá responder, indicando su localización en el suelo.

Asignación de posiciones. Tras un determinado tiempo de espera, el maestro pondrá en marcha el protocolo de asignación de localizaciones. En este momento, a cada multicóptero integrante en el enjambre se le asignará una posición aérea a la que tendrá que alcanzar una vez despegue. En ArduSim existen cuatro algoritmos disponibles para realizar esta asignación tierra-aire: fuerza bruta, heurístico, uso del KMA, y asignación aleatoria.

Como ya se ha mencionado en los capítulos anteriores, el algoritmo de asignación empleado para las pruebas iniciales va a ser el KMA, ya que nos ofrece la solución óptima referente a la distancia entre las localizaciones terrestres y aéreas. Una vez se hayan obtenido los resultados para dicho algoritmo de asignación, se llevará a cabo un análisis respecto al número de colisiones, tandas generadas, y coste de cálculo con alguno de los otros algoritmos restantes. También resulta ser de gran importancia aclarar que, para la elaboración de los algoritmos correspondientes al presente trabajo, se ha partido de la idea de que las trayectorias que realiza un dron para alcanzar su destino aéreo son líneas rectas, garantizando así que la distancia que debe recorrer sea la mínima posible.

Una vez terminada tal asignación de posiciones, ya estaremos listos para proceder a despegar el enjambre. Cabe destacar que la información recogida durante el protocolo de asignación es imprescindible para el desarrollo del proyecto, ya que nos permite obtener las localizaciones que cada dron hasta alcanzar su posición aérea. De este modo, solo deberemos comparar estas ubicaciones para ver si pueden llegar a colisionar.

Mecanismo de despegue. Resulta de suma importancia comentar el mecanismo de despegue de los drones. Habitualmente, cada una de estas aeronaves vuelan verticalmente hasta alcanzar una determinada altura. A partir de este momento, cada multicóptero se desplazará diagonalmente hacia su posición asignada. Dicho esto, los algoritmos que se van a explicar en las siguientes secciones buscan justamente evitar las colisiones a partir de la altura en la que se cambia el modo de desplazamiento de los drones.

Detección de colisiones. Por último, tal y como se ha explicado en el capítulo anterior, Ardusim tiene una funcionalidad encargada de la detección de colisiones. Para esta función, en cada una de las aeronaves existentes en el enjambre, se dibuja una zona de conflicto alrededor de cada uno de los drones que consiste en un círculo donde el diámetro tiene un valor por defecto equivalente a cinco metros de distancia. Normalmente se emplea este valor porque puede existir un margen de error en los sensores de GPS situados en cada uno de los multicópteros. Para nuestro caso en particular, consideramos que tal distancia es relativamente baja, y decidimos aumentarla para que el algoritmo de detección de colisiones pueda identificar previamente los drones que, debido a su trayectoria, puedan generar posibles conflictos.

5.2 Collisionless swarm takeoff heuristic (CSTH)

Esta primera versión del algoritmo nos va a ser de gran utilidad, ya que en él pondremos en práctica algunos de los conceptos básicos y, a su vez, nos servirá de referencia para la

creación de sus respectivas optimizaciones. Esta primera versión consiste en comprobar si, para una determinada localización tridimensional de un dron, situada dentro de la trayectoria que este debe tomar hasta llegar a su punto de destino, existe alguna posible colisión con el resto de integrantes que conforman el enjambre.

En un primer instante deberemos obtener todos los puntos intermedios que un determinado dron debe recorrer para lograr alcanzar su posición aérea objetivo. Para conseguir todas esas posiciones dentro de la trayectoria de una aeronave extraeremos su vector tridimensional que une su localización terrestre y la aérea. De este modo, este vector será el que nos indique la dirección que va a tomar el dron desde su punto de partida. Una vez obtenido dicho vector, deberemos normalizarlo para obtener su respectivo vector normalizado, el cual deberemos emplear para obtener cada una de las localizaciones intermedias que integran su trayectoria. Para conocer todas las posiciones intermedias deberemos sumar este vector normalizado (o una versión escalada) a la posición terrestre (inicial), y hacer esto sucesivamente hasta el momento en el que obtengamos las coordenadas de la posición aérea (final). Para optimizar este procedimiento, y no reiterarlo cuando sea innecesario, en el constructor de la clase principal se inicializa una lista de vectores normalizados de todos los drones existentes en el enjambre, con su respectivo identificador, que nos permita obtenerlo con total rapidez. Otro detalle importante a comentar es que el vector normalizado tiene una longitud unitaria. Es decir, si una determinada aeronave tiene como destino una localización que se encuentra a unos veinte metros de altura, obtendremos al menos veinte posiciones intermedias (contando la inicial y final), ya que cada vez que nos desplazamos lo hacemos en un valor de un metro de longitud repartido entre sus tres coordenadas. En el caso de que multiplicáramos por dos el vector normalizado nos desplazaríamos el doble de rápido, obteniendo la mitad de las localizaciones tridimensionales comentadas en el caso anterior. Con esto, el valor por el cual multiplicamos el vector normalizado se conoce como la granularidad, y nos será de gran utilidad para optimizar el coste del cálculo. A su vez, la granularidad es un factor sumamente delicado, ya que cuanto más elevado sea su valor, mayor la probabilidad de que situaciones con peligro de colisión no sean detectadas debido a que los saltos entre posiciones consecutivas son mayores. De este modo, deberemos estudiar qué granularidad nos permite optimizar nuestro tiempo de cómputo al mismo tiempo que garantiza la seguridad de la fase de despegue. En la Figura 5.2.1 podemos apreciar las localizaciones intermedias existentes en la trayectoria que un dron debe efectuar para alcanzar su destino aéreo. Tal y como se aprecia en la imagen, la distancia entre dos puntos será la marcada por el valor de la granularidad que, a su vez, se multiplicará con el vector normalizado de dicho dron.

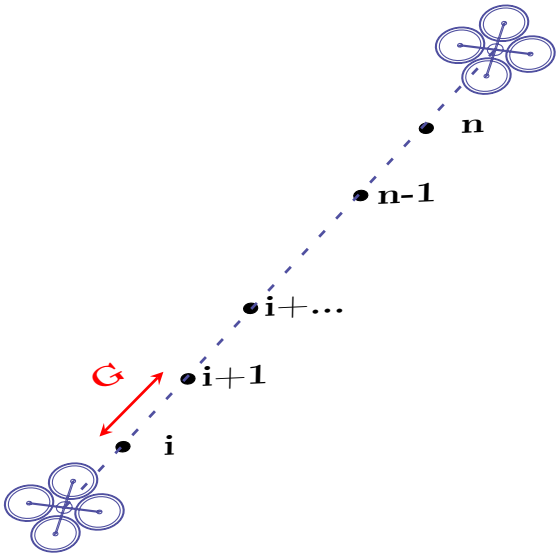


Figura 5.2.1: Obtención de las localizaciones intermedias.

Una vez terminado este proceso, ya podremos proceder a describir con más detalle la implementación y cálculo computacional. Para el desarrollo del algoritmo se requieren dos datos de entrada. El primero de ellos es la información resultante de aplicar el algoritmo de asignación. Esta variable nos permite una gran flexibilidad, ya que en todo momento podemos obtener las localizaciones que deben alcanzar los drones. La segunda de ellas es la lista con los identificadores de los drones con los que se va a pretender hallar la existencia de colisiones entre ellos. Resulta importante comentar que, en dicha lista, no es necesario que se encuentren todos los identificadores de los drones participantes en el experimento, pero esto se mostrará con más claridad en las posteriores secciones del capítulo presente.

Entrando ya en el análisis del código del algoritmo, inicialmente recorreremos cada uno de los identificadores pertenecientes a la lista comentada en el párrafo anterior. Luego, para cada dron, extraeremos su localización tridimensional terrestre y aérea junto a su respectivo vector de normalización. A partir de estos datos compararemos la distancia de cada una de las posiciones intermedias situadas en la trayectoria del dron que estamos recorriendo con todas las posiciones existentes del resto de los integrantes del enjambre. En la Figura 5.2.2 podemos apreciar como se realiza la comparación entre localizaciones de dos drones distintos, tal como se ha explicado anteriormente.

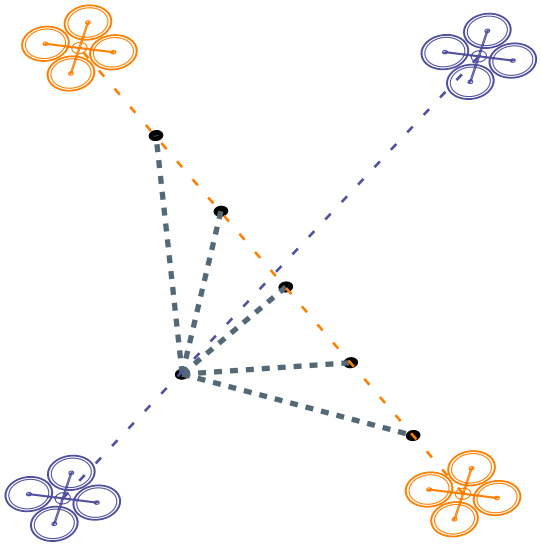


Figura 5.2.2: Comparación entre localizaciones intermedias de dos drones.

Una vez efectuada tal comparación, si la distancia entre dos localizaciones es menor que el valor del margen de error de GPS que asignamos en el constructor de la clase, afirmaremos que entre las dos aeronaves comparadas existe un riesgo de colisión, y guardaremos las posiciones de ambos drones junto a su distancia tridimensional en el momento que se ha detectado su posible colisión. Esto tiene el objetivo de evitar realizar comprobaciones innecesarias ya que, cuando se detecta una posible colisión, ya no procede comparar más posiciones con ese dron, y se procedería a realizar el mismo procedimiento con el siguiente identificador de la lista inicial. En Figura 5.2.3 podemos observar un ejemplo que muestra la trayectoria de dos aeronaves, y en la que se muestra simbolizada con dos líneas rojas la zona de peligro de colisión existente bajo un determinado parámetro de colisión.

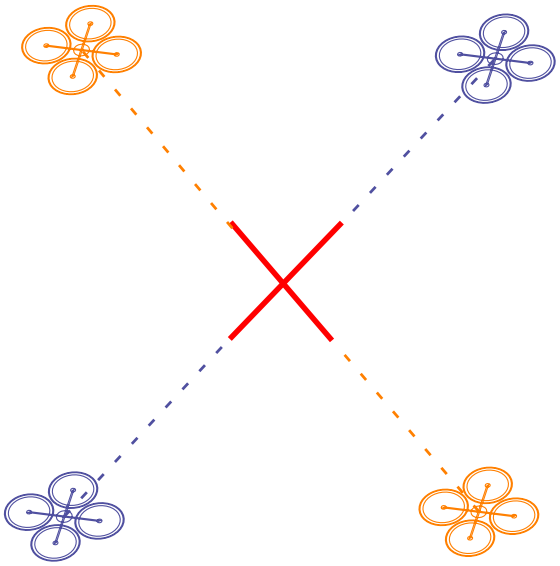


Figura 5.2.3: Zona de peligro de colisión entre dos drones.

Un detalle importante a recalcar es que, una vez que se haya recorrido una etiqueta de un determinado dron y no se haya encontrado ninguna colisión, esta etiqueta no

formará parte de las futuras comprobaciones de colisiones, ya que previamente ya se había detectado que no había problemas con ninguno de los otros miembros del enjambre. De este modo, cuando no existan más identificadores en la lista que estamos recorriendo, el algoritmo devolverá un listado con todas las colisiones que se han obtenido. En el Algoritmo 1 podemos observar la implementación del cálculo de detecciones. En él se detalla cómo se han efectuado todos los pasos que hemos comentado anteriormente.

Algorithm 1 detectCollisionCSTH(uavs)

Require: $uncheckedUAV.size = uavs.size \wedge placement \wedge safetyDist$

```
1: for uavId in uavs do
2:     uncheckedUAV.remove(uavId)
3:     position = placement.get(uavId)
4:     air = placement.getAir(uavId)
5:     while position.distance3D(air) >1 do
6:         position = displace(position)
7:         for nextUAV in uncheckedUAV do
8:             nextpos = placement.get(nextUAV)
9:             nextair = placement.getAir(uavId)
10:            while nextpos.distance3D(nextair) >1 do
11:                nextpos = displace(nextpos)
12:                if position.distance3D(nextpos) <= safetyDist then
13:                    collisionList.add(uavId)
14:                    goToLine(1)
15:                end if
16:            end while
17:        end for
18:    end while
19: end for
20: return collisionList
```

5.2.1 Optimización 1: Restricted Search Range (RSR)

En esta primera optimización se intenta mejorar el tiempo de cálculo reduciendo el número de comprobaciones de las distancias entre dos multicópteros. Para llevar a cabo esto, en lugar de comparar una posición intermedia del dron que estemos recorriendo con todas las localizaciones de los drones participantes en el enjambre, se elaborará un determinado rango de valores que tomará como punto central la altura de la posición de la primera aeronave. De este modo, por cada posición intermedia de un dron, se podrán descartar aquellas localizaciones de otras aeronaves cuyas alturas sean dispares entre sí, reduciendo notablemente el coste de cómputo. Además, para garantizar la correcta implementación de esta versión, se requiere que exista como dato de entrada un parámetro donde el usuario le asignará un valor entero para poder obtener los verdaderos límites del rango a emplear. A modo de ejemplo, supongamos que un dron se encuentra en una localización cuya altura es de diez metros, y el valor del parámetro para construir el intervalo es de dos metros. Con estos datos, a nuestro rango le corresponderían todos aquellos valores comprendidos entre ocho y doce. Una vez establecidos los límites del intervalo, descartaríamos la comparación de aquellas posiciones intermedias que se encontraran en una altura fuera de rango.

En la Figura 5.2.4 podemos visualizar el funcionamiento del algoritmo CSTH-RSR.

Tal y como apreciamos, vemos que ahora una posición intermedia del primer dron es comparada con únicamente cuatro localizaciones, reduciendo así el número de cálculos a realizar. Al igual que la anterior versión, el valor que le asociemos al rango resultará crucial para garantizar la detección de las colisiones ya que, si se emplea un valor bajo, varios posibles conflictos podrían no ser detectados. Por esta razón, deberemos efectuar un estudio exhaustivo de los distintos valores que podemos usar para definir nuestro intervalo, y de esta forma asegurar la fiabilidad de nuestro algoritmo.

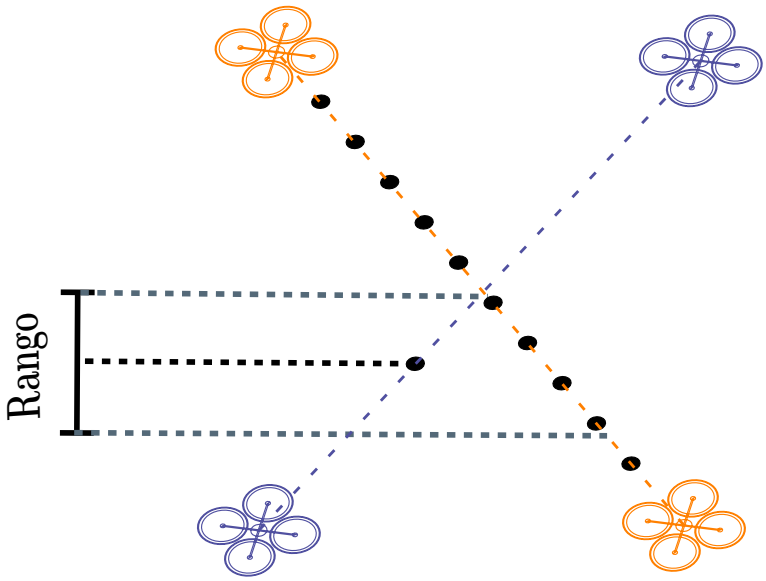


Figura 5.2.4: Funcionamiento del algoritmo CSTH-RSR.

Profundizando más en el funcionamiento de esta optimización, la lista con los identificadores de todos los drones que participan en la misión del enjambre se recorre del mismo modo que en la primera versión. La mejora del rango de valores solo afectaría aquellos multicópteros que se desea comparar con el actual. Un detalle a comentar es que, para cada uno de los puntos intermedios del identificador del dron que estamos recorriendo, obtenemos el valor de su eje Z (equivalente a su altura). A este valor le sumaremos y le restaremos el valor del parámetro de rango que el usuario ha asignado previamente para obtener los valores mínimos y máximos de nuestro intervalo de búsqueda. Aquí tenemos que tener en cuenta que los valores mínimos y máximos del intervalo deberán ser iguales o menores que la localización aérea de la aeronave, y mayores o iguales a la altura en la que empieza su desplazamiento en diagonal. Una vez que el rango de altura sea establecido, lo único que haremos es comparar las localizaciones con una altura comprendida dentro de este rango de valores. Por último, el sistema empleado para verificar si existen colisiones será el mismo que se ha comentado en su versión inicial. Del mismo modo que se hizo en la versión anterior, en el Algoritmo 2 se pueden observar los cambios realizados para implementar dicha optimización.

Algorithm 2 detectCollisionCSTH-RSR(uavs)

Require: $uncheckedUAV.size = uavs.size \wedge placement \wedge range > 0 \wedge safetyDist \wedge minHeight \wedge maxHeight$

```
1: for uavId in uavs do
2:     uncheckedUAV.remove(uavId)
3:     position = placement.get(uavId)
4:     air = placement.getAir(uavId)
5:     while position.distance3D(air) >1 do
6:         position = displace(position)
7:         for nextUAV in uncheckedUAV do
8:             nextpos = placement.get(nextUAV)
9:             minz = min(minHeight,position.z - range)
10:            maxz = max(maxHeight,position.z + range)
11:            while nextpos.z <= maxz do
12:                if nextpos.z <= minz then
13:                    nextpos = displace(nextpos)
14:                    continue
15:                end if
16:                if position.distance3D(nextpos) <= safetyDist then
17:                    collisionList.add(uavId)
18:                    goToLine(1)
19:                end if
20:                nextpos = displace(nextpos)
21:            end while
22:        end for
23:    end while
24: end for
25: return collisionList
```

5.2.2 Optimización 2: Divergent Trajectory Detection (DTD)

La presente optimización tiene como principal objetivo reducir el coste de cómputo de la versión inicial propuesta, descartando aquellos drones cuya trayectoria se vaya distanciando respecto al recorrido de la aeronave actual, ya que los recorridos definidos son en línea recta. De este modo, podemos descartar aquellas líneas que divergen de la línea correspondiente a la trayectoria del dron actual. Para implementar este algoritmo necesitaremos obtener, de las dos aeronaves que vamos a comparar, sus primeras dos posiciones intermedias consecutivas. Esta comparación se efectuará guardándonos la distancia entre las localizaciones iniciales de los dos multicópteros, y la distancia entre las localizaciones consecutivas de los mismos. Una vez hecho esto, nos podemos encontrar con dos situaciones distintas según los valores de las distancias resultantes. En la Figura 5.2.5 podemos apreciar el caso en el que la segunda distancia (d2) es inferior que la primera (d1), entendiéndose así que ambas trayectorias tienden a cruzarse en algún momento. Si esto ocurre, añadiremos el identificador del primero de los drones que estábamos comparando en una nueva lista que emplearemos para la fase de comprobación de colisiones.

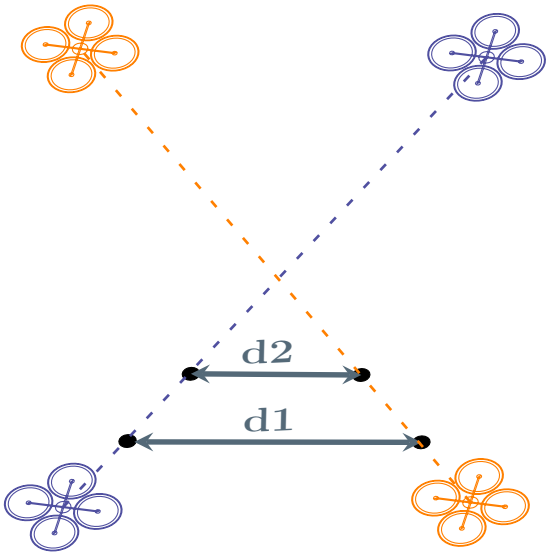


Figura 5.2.5: Ejemplo aeronaves con trayectorias convergentes.

De manera opuesta, en la Figura 5.2.6 visualizamos el caso en que las trayectorias de ambos drones se alejan a medida que estos ascienden hacia su destino. En esta situación podremos descartar con total seguridad que entre los drones comparados exista algún peligro de colisión.

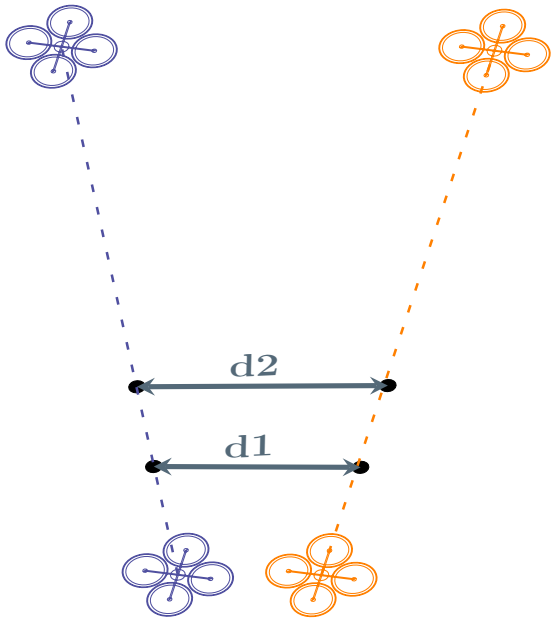


Figura 5.2.6: Ejemplo aeronaves con trayectorias divergentes.

Esta solución nos permite conocer la totalidad de riesgos de colisión existentes en la formación linear, pero no ocurre lo mismo en la circular o matricial. En determinadas ocasiones, empleando únicamente las dos primeras posiciones intermedias de ambos drones, no es suficiente para determinar que sus trayectorias son convergentes debido a que las distancias obtenidas son muy similares. Con el objetivo de obtener una precisión mucho mayor en la exclusión de drones usando la misma técnica, lo que haremos es realizar el cálculo de dos nuevas distancias equivalentes entre la primera localización de un dron

con la segunda del otro, y hacerlo lo mismo de forma inversa (ver Figura 5.2.7). Esto lo hacemos porque en determinados casos las aeronaves tienden a encontrarse en zona de conflicto en posiciones con una clara diferencia de altura debido a que los vectores normalizados de alguno de los multicópteros tienden a ser muy distintos entre sí. De este modo, si alguna de estas dos nuevas distancias (d3 y d4) es menor que la distancia que existía inicialmente entre sus primeras localizaciones (d1), insertaremos también el identificador en el nuevo listado de aeronaves. A partir de este momento, las fases siguientes serán idénticas a la versión inicial, exceptuando que emplearemos la nueva lista de etiquetas generada anteriormente. Al igual que en las anteriores secciones, en los Algoritmos 3 y 4 podemos encontrar el pseudocódigo referente a lo que se ha explicado en el presente apartado.

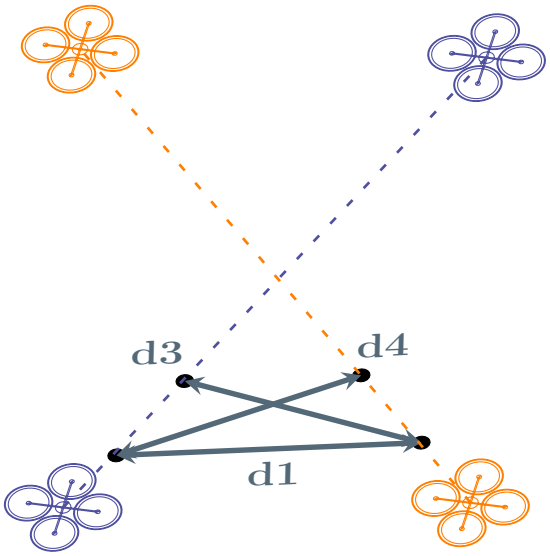


Figura 5.2.7: Comparación de las nuevas distancias obtenidas.

Algorithm 3 detectCollisionCSTH-DTD(uavs)

Require: $uncheckedUAV.size = uavs.size \wedge placement \wedge safetyDist$

```
1: for uavId in uavs do
2:     uncheckedUAV.remove(uavId)
3:     position = placement.get(uavId)
4:     nextposition = displace(position)
5:     possibleCollision = checkConvergingLines()
6:     while position.distance3D(air) >1 do
7:         position = displace(position, uavId)
8:         for nextUAV in possibleCollision do
9:             nextpos = placement.get(nextUAV)
10:            while nextpos.distance3D(nextair) >1 do
11:                nextpos = displace(nextpos)
12:                if position.distance3D(nextpos) <= safetyDist then
13:                    collisionList.add(uavId)
14:                    goToLine(1)
15:                end if
16:            end while
17:        end for
18:    end while
19: end for
20: return collisionList
```

Algorithm 4 checkConvergingLines(position, nextposition, uncheckedUAV)

```
1: for uavId in uncheckedUAV do
2:     pos = placement.get(uavId)
3:     nextpos = displace(pos)
4:     locationAux = displace(nextpos)
5:     locationAux2 = displace(nextposition)
6:     d1 = position.distance3D(pos);
7:     d2 = nextposition.distance3D(nextpos);
8:     d3 = nextposition.distance3D(locationAux);
9:     d4 = nextpos.distance3D(locationAux2);
10:    if d1 >d2 || d1 >d3 || d1 >d4 then
11:        possibleCollision.add(uavId)
12:    end if
13: end for
14: return possibleCollision
```

5.2.3 Versión combinada: CSTH-RSR+DTD

Las dos optimizaciones explicadas en las anteriores secciones son compatibles entre sí. Mientras que en la primera de ellas su objetivo es reducir la cantidad de comparaciones entre todos los puntos existentes entre las posiciones de inicio y destino de un determinado dron, la segunda de ellas tiene el fin de descartar aquellas aeronaves que tienen una trayectoria que se aleje del recorrido de un multicóptero en concreto.

Dicho esto, en el Algoritmo 5 apreciamos como quedaría el pseudocódigo de la versión combinada.

Algorithm 5 detectCollisionCSTH-RSR+DTD(uavs)

Require: $uncheckedUAV.size = uavs.size \wedge placement \wedge range > 0 \wedge safetyDist$

```
1: for uavId in uavs do
2:     uncheckedUAV.remove(uavId)
3:     position = placement.get(uavId)
4:     nextposition = displace(position)
5:     possibleCollision = checkConvergingLines()
6:     while position.distance3D(air) >1 do
7:         position = displace(position)
8:         for nextUAV in possibleCollision do
9:             nextpos = placement.get(nextUAV)
10:            minz = min(minHeight,position.z - range)
11:            maxz = max(maxHeight,position.z + range)
12:            while nextpos.z <= maxz do
13:                if nextpos.z <= minz then
14:                    nextpos = displace(nextpos)
15:                    continue
16:                end if
17:                if position.distance3D(nextpos) <= safetyDist then
18:                    collisionList.add(uavId)
19:                    goToLine(1)
20:                end if
21:                nextpos = displace(nextpos)
22:            end while
23:        end for
24:    end while
25: end for
26: return collisionList
```

5.3 Euclidean distance - Collisionless swarm takeoff heuristic (ED-CSTH)

Una vez realizado el código de los algoritmos de detección de colisiones propuestos en los anteriores apartados, decidimos empezar a comprobar los resultados obtenidos para medir cuál era el rendimiento de cada uno de ellos. En un primer momento, habíamos hecho experimentos con una granularidad igual a uno, es decir, con una longitud vectorial de una unidad, para obtener los puntos intermedios entre las posiciones terrestres y aéreas; no obstante vimos que, al usar valores de granularidad menores que uno, incrementábamos el número de colisiones potenciales detectadas. Esto suponía un grave problema, ya que cada vez que reducíamos la longitud vectorial, detectábamos nuevos peligros de colisiones que en las pruebas anteriores habíamos descartado, resultando incierto si determinado algoritmo era seguro o no.

Una posible solución que solventaba este problema era la de reducir el valor de la granularidad a un valor con dos o tres decimales para asegurarnos con total certeza que

todas las localizaciones tridimensionales se comparaban entre sí. Aunque los datos después de efectuar esto serían correctos, estaríamos generando otro problema, ya que al reducir la granularidad, estamos aumentando el número de posiciones con las que comparar, y esto implica aumentar de forma considerable el tiempo de coste de cálculo del algoritmo.

La solución que se propone a continuación se fundamenta en la distancia euclidiana entre dos trayectorias en un espacio tridimensional. Uno de los requisitos para este nuevo algoritmo es que las trayectorias de los drones hasta alcanzar su destino en el aire deben corresponder a rectas. Asumiendo este requisito podemos determinar la distancia entre ambas rectas en el espacio tridimensional.

El caso más sencillo de comprobar es si las trayectorias de dos drones son paralelas. Aquí, únicamente verificaríamos que los vectores de cada uno de los drones apuntan hacia la misma dirección, es decir, que sus tres coordenadas son exactamente las mismas. Aunque es el caso menos probable, en tal situación confirmaríamos que entre dichos drones no hay peligro de colisión siempre y cuando la distancia entre rectas supere el umbral de seguridad.

Este primer suceso lo revisamos en todas las iteraciones realizadas del algoritmo de detección de colisiones, ya que su cálculo no perjudica demasiado en el tiempo de cálculo resultante. Sin embargo, para los casos restantes efectuaremos la siguiente fórmula destinada a calcular la distancia mínima entre dos rectas tridimensionales:

$$d(r,s) = \frac{|[\vec{v_r}, \vec{v_s}, \vec{PQ}]|}{\vec{v_r} \times \vec{v_s}} \tag{5.1}$$

.

Las dos rectas tridimensionales son representadas por las letras r y s mientras que P haría referencia a un punto de la recta r y Q a otro de la recta s. En esta fórmula, los cálculos correspondientes al numerador consisten en resolver el determinante compuesto por los vectores de las dos rectas, y por el vector definido a partir de las localizaciones terrestres de ambas aeronaves. Por otro lado, en el denominador deberemos llevar a cabo el producto vectorial de los vectores de las dos rectas. Como en el cálculo ejecutado se lleva a cabo con una gran cantidad de decimales, ambas trayectorias nunca intersectarán exactamente en un mismo punto y, por lo tanto, el valor obtenido después de aplicar la fórmula siempre será mayor que cero.

Una vez obtenida la distancia mínima, deberemos obtener los dos puntos pertenecientes a cada una de las rectas para saber si esta distancia se encuentra dentro del rango de altura establecido a través de las localizaciones terrestres y aéreas. La solución propuesta para llevar a cabo esto ha sido la de crear una recta perpendicular a las rectas de cada uno de los drones que estamos comparando, de tal modo de que los puntos que se encuentran en cada una de las rectas sean los que realmente obtienen la distancia mínima entre ellas. En la Figura 5.3.1 visualizamos como quedaría la nueva recta ocasionada a partir de la distancia mínima entre las rectas r y s.

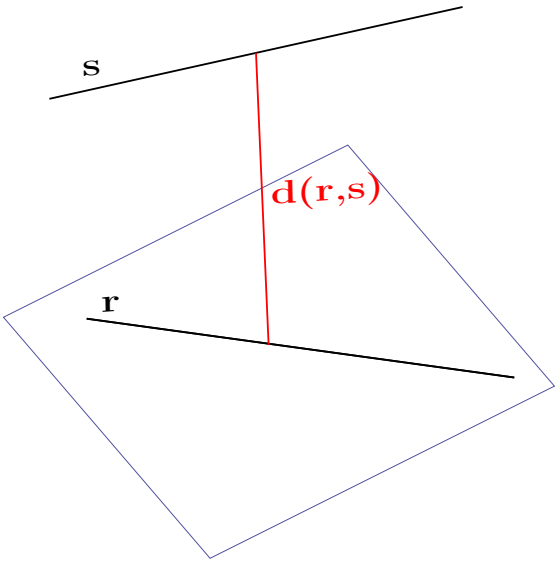


Figura 5.3.1: Recta creada a partir de la distancia mínima de dos rectas.

Para obtener los puntos requeridos necesitamos elaborar una serie de operaciones. En primer lugar, construiremos las ecuaciones de las rectas formadas por las trayectorias de los dos drones que estamos comparando. Tras esto, se determinará la distancia entre ellas de la siguiente manera:

$$\begin{cases} L1 : v_1 = p_1 + t_1 \cdot d_1 \\ L2 : v_2 = p_2 + t_2 \cdot d_2 \end{cases} \tag{5.2}$$

Las variables p consisten en los puntos iniciales pertenecientes a las rectas, mientras que las variables d pertenecen a sus vectores normalizados. En cada una de estas ecuaciones nos encontramos con una incógnita simbolizada por t. En segundo lugar, obtendremos el vector perpendicular a ambas líneas efectuando el producto vectorial de los vectores de las dos rectas.

$$n = d_1 \times d_2 \tag{5.3}$$

Una vez hecho esto, haremos el producto vectorial entre cada uno de los vectores de las rectas con respecto al vector obtenido en el paso anterior. De este modo, creamos un plano de impacto que será perpendicular a ambas líneas.

$$\begin{cases} n_1 = d_1 \times n \\ n_2 = d_2 \times n \end{cases} \tag{5.4}$$

Por último, lo único pendiente es obtener el valor de las incógnitas aplicando sus respectivas fórmulas, y remplazar su valor en la ecuación inicial.

$$\begin{cases} c_1 = p_1 + \frac{(p_2 - p_1) \cdot n_2}{d_1 \cdot n_2} \times d_1 \\ c_2 = p_2 + \frac{(p_1 - p_2) \cdot n_1}{d_1 \cdot n_1} \times d_2 \end{cases} \tag{5.5}$$

Para verificar que las coordenadas de los puntos obtenidos son las correctas, lo único que debemos hacer es calcular la distancia entre ambos puntos. Si todo ha ido bien, como resultado tendremos el mismo valor que habíamos obtenido anteriormente aplicando la ecuación de la distancia mínima entre rectas que se cruzan en el espacio tridimensional.

Después de esto, tendremos que decidir si la distancia obtenida nos sirve o no porque, al trabajar sobre rectas infinitas, puede que los puntos referentes a tal distancia pueden encontrarse fuera del rango establecido por la altura. Siendo así, en primer lugar haremos una división según cuál sea el valor del margen de error de GPS que se ha impuesto. Si este es menor que la distancia mínima entre dos rectas, aseguramos que entre dichas aeronaves no existe ningún peligro de colisión. En el caso contrario, nos podemos encontrar con tres casos distintos. Primeramente, si las coordenadas de ambos puntos están dentro del rango de altura de las aeronaves, confirmamos que existe cierto riesgo de colisión. El segundo caso consiste en que uno de los dos puntos se encuentre dentro del rango y el otro no (ya sea por debajo de la posición inicial, o por encima del destino final de la aeronave). En este caso, el punto que se encuentra fuera del rango lo moveremos al primer punto que se encuentre más cercano del rango y, empleando una granularidad muy baja, realizaremos múltiples comprobaciones con el punto fijo del segundo. Si en algún momento obtenemos un valor ligeramente inferior al margen de error del GPS, declararemos que hay una posible colisión. Por último, en caso de que los dos puntos estén fuera del rango, diremos que no existe colisión porque, a la hora de efectuar los experimentos en ArduSim, asumimos que entre todos los drones existe una distancia mínima tanto en las localizaciones terrestres como en las aéreas, las cuales nunca van a ser inferiores a la distancia de seguridad.

Del mismo modo que las anteriores versiones, en el Algoritmo 6 visualizamos el pseudocódigo referente al algoritmo ED-CSTH que acabamos de explicar en los párrafos anteriores.

Algorithm 6 detectCollisionED-CSTH(uavs)

Require: $uncheckedUAV.size = uavs.size \wedge placement$

```
1: for uavId in uavs do
2:     uncheckedUAV.remove(uavId)
3:     for nextUAV in uncheckedUAV do
4:         position = placement.get(uavId)
5:         pos = placement.get(nextUAV)
6:         if checkParallelVector(position,pos) then
7:             collisionFlag = false
8:         end if
9:         if checkPathUAV(position,pos) then
10:            collisionFlag = checkIntersectionPoint(position,pos)
11:        else
12:            collisionFlag = checkDistMin(position,pos)
13:        end if
14:        if collisionFlag then
15:            collisionList.add(uavId)
16:            goToLine(1)
17:        end if
18:    end for
19: end for
20: return collisionList
```

5.4 Algoritmo de generación de tandas de drones

Una vez conocidas cuáles son las colisiones dentro de un enjambre de drones, el siguiente paso es el de generar un mecanismo o algoritmo que nos permita separarlos en tandas o lotes, y en los que podamos garantizar que no se producen colisiones entre ninguno de sus integrantes. De este modo, estos grupos podrán despegar de forma simultánea proporcionando un tiempo mejor que si estos tuvieran que despegar de forma secuencial.

Antes de nada, resulta de suma importancia comentar que, en la lista resultante del siguiente algoritmo, las tandas de drones que se encuentran en las primeras posiciones deberían corresponder a los drones cuya distancia entre su localización de partida y su destino sea mayor. La solución que se ha empleado en el presente trabajo es modificar el algoritmo de detección de colisiones para que, en lugar de recorrer la lista de identificadores de drones que obtenemos como dato de entrada, ordene previamente esa lista según aquellas aeronaves que se caractericen por una distancia mayor hacía sus posiciones aéreas. Tras obtener el resultado de las tandas, lo único que deberemos hacer es ordenarlas por su distancia media. Efectuando este procedimiento nos aseguramos que los multicópteros que despegan en las primeras posiciones sean los que se encuentran en las posiciones más alejadas.

Entrando en los detalles del algoritmo propuesto, para su correcta implementación se requieren tres datos de entrada. El primero de ellos se trata de la lista de colisiones obtenidas en el algoritmo de detección de colisiones. El segundo de ellos es el listado de identificadores de dron. Y el tercero se trata del listado de tandas de multicópteros previamente formadas (durante su primera llamada estará vacía). Este último dato efectúa un papel importantísimo ya que, al tratarse de un método recursivo, esta variable será la encargada de almacenar el resultado de las tandas realizadas en las antiguas iteraciones del método.

En la Figura 5.4.1 visualizamos el diagrama de funcionamiento del mecanismo de generación de tandas propuesto. Para dar marcha al algoritmo recorreremos el listado de VANTs pertenecientes al enjambre y comprobaremos si cada uno de ellos aparece en la lista de colisiones que obtenemos como resultado después de haber aplicado el mecanismo de detección de colisiones. Si observamos que se encuentra en tal listado, directamente lo insertaremos en un nuevo grupo G2 donde se encontrarán aquellos drones que no cumplan con la distancia de seguridad. En caso contrario, si un VANT no aparece en dicha lista de colisiones lo introduciremos en el grupo G1. De este modo, conseguimos dividir la totalidad de identificadores existentes en el enjambre en dos subgrupos diferenciados por aquellas aeronaves que respectan o no la distancia de seguridad de vuelo.

Una vez formados tales grupos, crearemos nuestro primer lote o tanda de drones con las etiquetas de los VANTs pertenecientes a G1. Esta tanda la incorporaremos directamente a nuestro dato de salida porque al tratarse de aeronaves que no producen conflictos entre sí, podrán despegar de forma simultánea sin originar ningún tipo de riesgo. Tras haber hecho esto, el siguiente paso que deberemos llevar a cabo será el de comprobar la cantidad de etiquetas existentes en G2. Si se da el suceso de que su cifra es menor que dos, nos encontraríamos en frente de nuestro caso base del modo recursivo donde deberíamos verificar si G2 se encuentra vacía o si contiene un solo elemento. En el acontecimiento de que G2 estuviese vacía, automáticamente deberíamos devolver el dato de salida habiendo terminado así el algoritmo. Si, por lo contrario, G2 tuviera una etiqueta, la añadiríamos en un nuevo lote y también terminaríamos el algoritmo devolviendo su resultado.

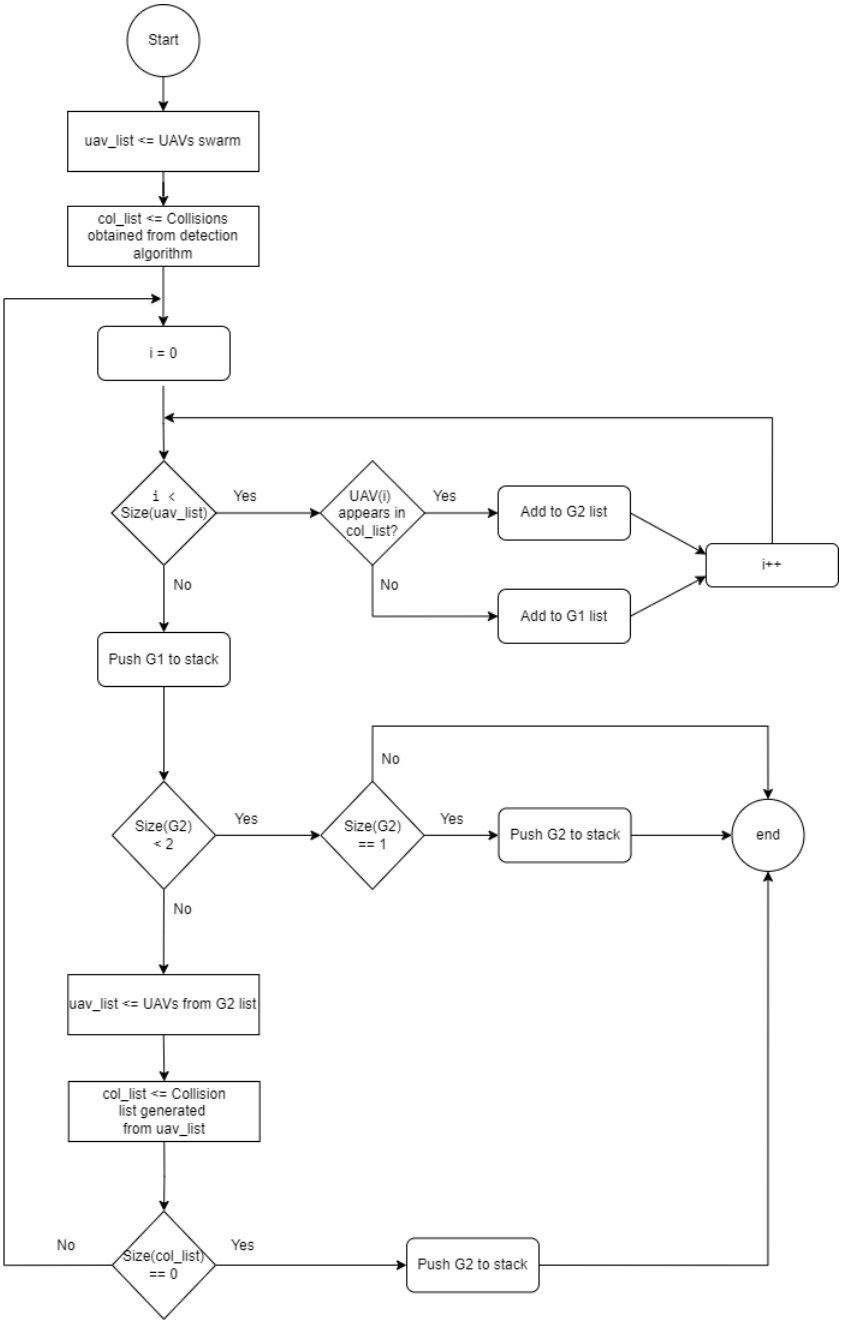


Figura 5.4.1: Diagrama del funcionamiento del mecanismo de generación de tandas.

Luego, si la cantidad de identificadores de G2 fuera mayor o igual que dos, procederíamos a elaborar una nueva lista de drones solamente que en esta ocasión estaría integrada únicamente por las etiquetas pertenecientes a G2. Una vez elaborado tal lista de VANTs, el siguiente paso que realizaremos es el de obtener las colisiones existentes entre los identificadores de G2. En este instante pueden ocurrir dos situaciones: que la nueva lista de colisiones generada esté completamente vacía o que existan nuevas áreas de conflicto. Si se da el primer caso, podremos incorporar en una nueva tanda todas las etiquetas de G2 debido a que no causan ningún peligro, terminándose de esta manera el algoritmo. En caso de que existan nuevas colisiones, regresaríamos al estado inicial del mecanismo salvo que con un listado de drones y una lista de colisiones distintas que la primera iteración. Además, vaciaremos los contenidos de los grupos G1 y G2 para que se puedan rellenar con

los nuevos identificadores. Tal y como se puede apreciar, esta vía sería el caso recursivo caracterizado por volver a regresar al punto de partida del método empleando datos de entrada más simplificados.

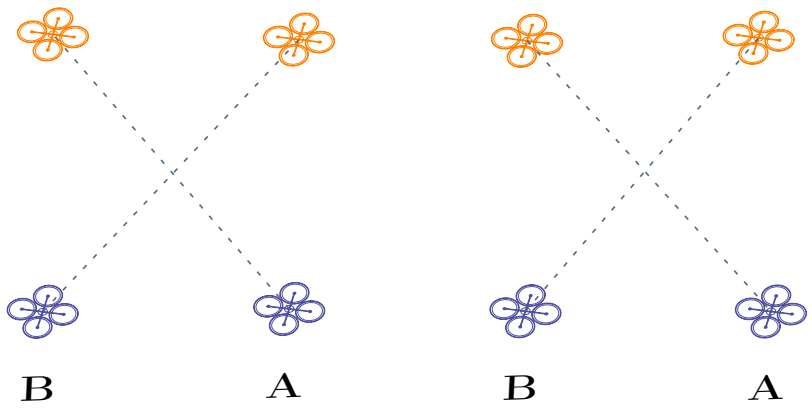


Figura 5.4.2: Ejemplo tras aplicar el mecanismo de generación de tandas.

En la Figura 5.4.2 visualizamos un ejemplo de aplicación del mecanismo de generación de tandas en un enjambre compuesto por cuatro aeronaves. En una primera iteración del algoritmo de detección de colisiones obtendríamos como resultado que existe un posible riesgo de colisión entre el primero y el segundo, así como entre el tercero y el cuarto. Entre estos conflictos, pondremos que el primero de ellos se encuentre en un grupo etiquetado como "B", y el resto lo pondremos en un grupo "A". En este momento, como la lista de colisiones obtenida contiene dos elementos, volveríamos a llamar el método de detección de colisiones para ver si los drones integrantes en el grupo "B" causan una colisión entre ellos. Al no ser así, lo que se devolvería sería un listado en el que, en primer lugar se incluyen los componentes del grupo "A", y posteriormente los del grupo "B".

En la primera sub-sección del presente capítulo se había comentado que existían dos fases de movimiento que los drones deberían efectuar para llegar a su destino en el aire: la vertical y la diagonal. Tras efectuar múltiples pruebas con este sistema llegamos a la conclusión de que evitamos las zonas de riesgo entre las aeronaves pertenecientes al mismo lote, pero no se podría garantizar que no existiera peligro de colisión entre los drones que se encuentren de camino a su posición con aquellos que se mantienen fijos, o de forma estática en su destino, debido a que ya lo han alcanzado. La solución que se propone es la de usar un margen de altura, con el mismo valor que el empleado en el despegue, de manera que el dron se desplace diagonalmente hasta conseguir las mismas coordenadas que su destino, aunque con una altura algo menor, para finalmente subir verticalmente hasta alcanzar su posición destino. Con esto, solucionaríamos el problema mencionado anteriormente porque nuestro margen de error de GPS siempre va a ser mayor en estos casos. A su vez, esta propuesta tiene un efecto negativo: al comprobar únicamente las colisiones en un rango menor, que incluye la altura compuesta por el desplazamiento diagonal, los vectores normalizados de las aeronaves tienden a ser más horizontales que en el primer momento, causando un número mayor de colisiones y, por lo tanto, de tandas. De todos modos, estamos dispuestos a que nuestro tiempo de cálculo sea algo superior con tal de mejorar la fiabilidad de la solución.

En el Algoritmo 7 observamos el pseudocódigo del mecanismo de generación de lotes de drones propuesto. Como se puede constatar, se trata de un algoritmo recursivo, y nos adentramos en la distinción de los dos casos posibles que se han comentado a través del

diagrama de funcionamiento.

Algorithm 7 batchGeneration(collisionList,uavs,batchList)

```
1: list.size = batchList.size
2: group1.size = uavs.size
3: group2 = getCollision(collisionList)
4: group1 = removeDuplicate(group1, group2)
5: if group2.size >1 then
6:     list.add(group1)
7:     collisionListAux = detectCollision(group2)
8:     if collisionListAux.size == 0 then
9:         list.add(group2)
10:    else
11:        uavsAux.size = group2.size
12:        batchGeneration(collisionListAux,uavsAux,list)
13:    end if
14: else
15:    if !group1.isEmpty() then
16:        list.add(group1)
17:    end if
18:    if !group2.isEmpty() then
19:        list.add(group2)
20:    end if
21: end if
22: return list
```

5.5 Sistemas de despegue de los drones

Una vez definidos los mecanismos de detección de colisiones y de generación de tandas de drones optimizados, el último objetivo que nos queda pendiente es el de mejorar el tiempo en el que las aeronaves deben despegar. En este punto, apreciaremos las múltiples virtudes que nos ofrece ArduSim, el cual nos ofrecerá, a través de su interfaz gráfica, una amplia visión del despegue del enjambre.

Inicialmente, se puede efectuar la etapa de despegue de dos formas distintas. Por un lado tenemos el modo secuencial por lotes de aeronaves, donde todas aquellas que pertenezcan a un mismo grupo pueden despegar simultáneamente y, hasta que no alcancen su localización aérea, no despegará el siguiente grupo en el listado de tandas. Otra opción distinta es el modo semi-simultáneo, caracterizado porque los drones integrantes del grupo posterior puedan elevarse una vez que las aeronaves de la primera tanda hayan alcanzado una determinada altura. Normalmente dicha altura se corresponde a cuando hayan terminado de ascender verticalmente en el aire y empiecen a desplazarse diagonalmente hacia su destino. Aunque empleando cualquiera de los modos de despegue comentados obtendríamos una clara mejoría respecto al algoritmo secuencial, en el presente proyecto se ha decidido emplear el modo semi-simultáneo porque se obtienen unos tiempos mucho más eficientes.

Cuando se empezó a llevar a cabo los primeros experimentos con enjambres de una cantidad considerable de drones, identificamos la existencia de un problema empleando

el modo semi-simultáneo en la formación lineal. Este contratiempo se debía a que no controlábamos si aeronaves pertenecientes a tandas consecutivas respetaban la distancia de seguridad establecida, debido a que los pertenecientes al segundo grupo poseían un vector longitudinal más inclinado.

Para el correcto funcionamiento con esta modalidad de despegue se requiere aplicar unas ciertas medidas para evitar que el problema anterior se reproduzca. Tras efectuar un minucioso análisis de lo ocurrido se propuso varias ideas para intentar solucionar dicha cuestión. La primera de ellas consistía en aumentar el margen de altura en la que los drones vuelan verticalmente, ya que al aumentarlo también lo haría el tiempo entre los grupos. Sin embargo, tras efectuar una batería de pruebas se dedujo que tal propuesta resultaría óptima si las aeronaves tuvieran una posición aérea muy elevada, pero con alturas entre los veinte y cincuenta metros continuaban existiendo algunas anomalías que causaban riesgos de colisión. La segunda propuesta fue la de invertir el orden de despegue de las tandas haciendo que las más cercanas a las posiciones centrales despegaran en un primer momento. Esta idea tampoco nos servía porque seguíamos sin garantizar la inexistencia de zonas de conflicto. Finalmente, se decidió optar por ampliar el tiempo de espera en el despegue (en 3.5 segundos adicionales) entre cada uno de los lotes del algoritmo. Aunque empleando esta técnica empeoraremos considerablemente el tiempo total del despegue (sobre todo en las formaciones con muchas tandas), se conseguiría obtener la fiabilidad requerida.

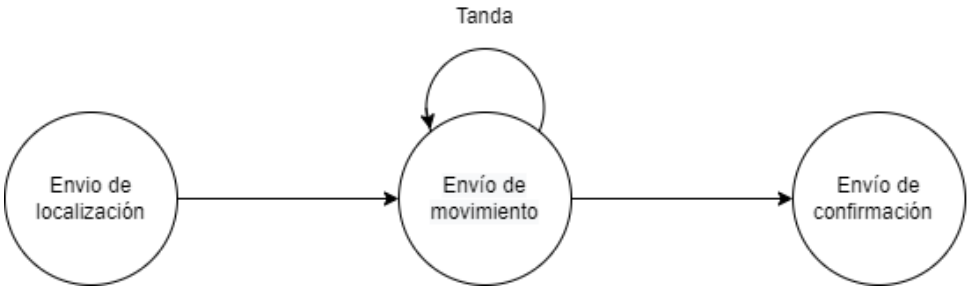


Figura 5.5.1: Sistema de comunicaciones del maestro.

Con el modo de sistema de despegue elegido, se va a proceder a realizar la explicación del sistema de comunicación entre drones existente durante el proceso de despegue. Tal y como mencionamos en la primera sección del capítulo, el maestro es la aeronave encargada de tomar las decisiones en cualquier momento. Dicho esto, en la Figura 5.5.1 visualizamos las tres fases de envío más significativas de este dron. Nada más ejecutar el algoritmo de detección de colisiones, el nodo maestro iniciará la fase de descubrimiento, y enviará a todos los nodos su localización e identificador. Los esclavos que participen en la misión le responderán con su respectiva etiqueta y posición terrestre. Tras esperar un cierto tiempo, el maestro pondrá en marcha la etapa de movimiento. Para ello, recorrerá la lista de tandas de drones previamente obtenida y, para cada una de ellas, el maestro enviará el mensaje de mover a su objetivo. En este momento, el maestro enviará constantemente mensajes a los esclavos pertenecientes al grupo de que se encuentra volando, donde éstos le responderán únicamente cuando empiece a moverse diagonalmente hacia su destino. Una vez que el maestro obtiene el mensaje de confirmación por parte de uno de los drones, dejará de enviar mensajes a tal aeronave. De este modo, el maestro se quedará durante un periodo de tiempo esperando el mensaje de confirmación de todos los esclavos. Una vez terminado, este procedimiento se realizará para cada una de las tandas existentes de

acuerdo a su respectivo listado. En nuestro caso, el último lote estará compuesto por el maestro debido a que, casi siempre, será la aeronave que tenga una menor distancia hacia su objetivo. Cuando este alcance su destino, enviará a todos los esclavos, que previamente se encuentran en espera, un mensaje de confirmación que les indica que la fase de despegue ha terminado.

Capítulo 6

Experimentos y Resultados

En el presente capítulo se realizarán una serie de experimentos con los diversos algoritmos de detección de colisiones propuestos para visualizar sus prestaciones y comprobar que la etapa de despegue se lleva a cabo con total seguridad. Para efectuar esto, se ha decidido exponer cada uno de los experimentos con sus respectivos resultados en una misma subsección para facilitar su lectura.

Antes de empezar, resulta imprescindible exponer brevemente cuáles son las formaciones de enjambre que se ha decidido emplear en el presente capítulo. Para las localizaciones terrestres se ha decidido utilizar en todos los experimentos la formación aleatoria. Aquí, los VANTs toman una posición aleatoria en el suelo conservando una pequeña distancia entre las restantes aeronaves que integran el enjambre. Se ha empleado dicha formación porque es la que más se aproxima a las pruebas con dispositivos reales en la que un usuario los sitúa uno al lado del otro sin ningún miramiento.

Por lo que respecta las formaciones aéreas, consideraremos tres: lineal, circular y matricial. En el caso de la linear, es de gran utilidad en las aplicaciones destinadas a la agricultura de precisión [26] porque permiten cubrir una área más extensa, ya que los VANTs se sitúan formando una línea aérea. La formación circular se caracteriza por contar con una aeronave en el medio, normalmente con la funcionalidad de maestro, y el resto de drones se sitúan a su alrededor formando un círculo. Por último, nos encontramos con la formación matricial que, según cual sea la cantidad de VANTs empleadas en un determinado enjambre, intentará organizarlos en una matriz de $n\ x\ n$ drones en la medida en que sea posible.

Estas formaciones se encuentran ya implementadas en ArduSim [14], y su uso nos permitirá tener una mayor flexibilidad para examinar el rendimiento de cada uno de los experimentos planteados.

La máquina que se ha empleado para la recogida de los resultados dispone de un procesador Intel modelo i5-2500K, con cuatro núcleos que proporcionan una frecuencia básica de 3,30 *GHz*. En lo que respecta a la memoria, su tamaño es de 4 *GB*, y es de tipo DDR3.

6.1 Experimento 1: Obtención de la granularidad idónea.

Uno de los principales problemas que nos topamos en el algoritmo CSTH, y sus respectivas optimizaciones, es que necesitamos asegurar que la granularidad que empleamos nos per-

mite obtener todas las áreas de conflicto dentro de un enjambre de drones. Como ya se ha explicado en los capítulos anteriores, la granularidad va a tomar un papel imprescindible a la hora de extraer todas las posiciones intermedias de un dron respecto su localización terrestre y aérea. Esto supone que la elección del valor de la granularidad va a tratarse de un proceso sutil, ya que si seleccionamos un valor elevado podríamos conseguir unas localizaciones tridimensionales cuyas distancias son mayores que el valor asignado al margen de error de GPS propuesto y, por lo tanto, no se detectan determinadas colisiones.

Teniendo en consideración este problema, el objetivo principal de este primer conjunto de experimentos va a ser el de determinar un valor para la granularidad en el que nos permita estar seguros de obtener todas las aéreas de conflicto entre todos los miembros del enjambre. Para llevar a cabo esto efectuaremos una batería de pruebas consistentes en la modificación de distintos valores sobre un número de drones fijo, y sobre cada una de las tres formaciones aéreas disponibles (circular, linear, matricial). La finalidad de dichas pruebas consiste en examinar y analizar la relación existente entre el número de colisiones que se pierden en cuanto se aumenta el valor asignado para la granularidad. Como punto de partida del experimento, se ha decidido empezar por un valor de granularidad equivalente a uno, e ir aumentando en una unidad hasta alcanzar el valor de diez. Por lo que respecta al número de drones, se ha decidido emplear la cifra de doscientos para que la cantidad de posibles zonas de peligro que no se detectan sean lo suficientemente significativas para llegar a una conclusión. La distancia de seguridad que se ha propuesto en el presente experimento es de ocho metros de longitud para que se pueda garantir la seguridad de las maniobras de desplazamiento realizadas por los drones. También resultada de gran interés mencionar que la distancia mínima existente entre las posiciones terrestres de las aeronaves es de diez metros. Además de este parámetro, también tendremos una distancia entre las ubicaciones aéreas correspondiente a veinte metros. Por último, la altitud que emplearemos en estos experimentos es también de veinte metros.

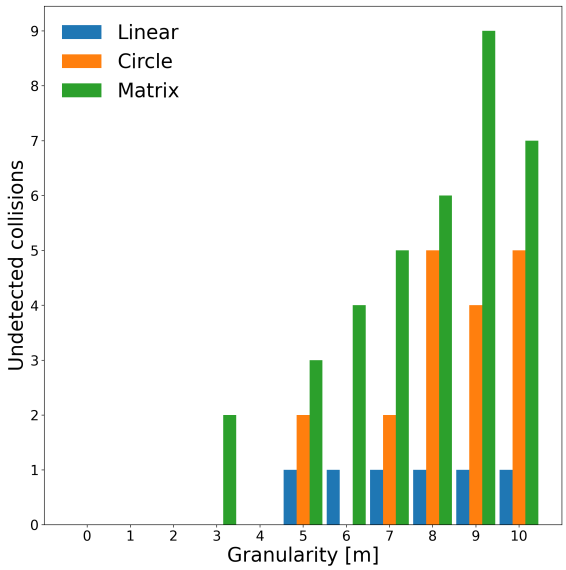


Figura 6.1.1: Zonas de riesgo de colisión según su granularidad.

En la Figura 6.1.1 podemos visualizar las zonas de riesgo de colisión que perdemos a medida que incrementamos el valor de la granularidad. Esta cifra de conflictos la extraemos efectuando la diferencia de colisiones obtenidas de cada uno de los experimentos

ejecutados con respecto a la primera prueba con granularidad igual a uno. Tal y como se aprecia en la figura, a partir de un valor equivalente a tres vemos que, para la formación aérea matricial, empezamos a fallar en la detección de algunas colisiones. Si seguimos aumentando el valor de dicha variable vemos que las pérdidas llegan a tomar valores realmente elevados, los cuales impiden que la maniobra de despegue del enjambre de drones sea fiable. Como con la granularidad igual a dos es el último valor en el cual conseguimos no perder ningún posible conflicto, lo asignaremos como nuestra solución ideal y óptima. La justificación de esto es que, tal y como vemos en la Figura 6.1.2, existe una relación entre el tiempo de cálculo del algoritmo y la granularidad, provocando que el primero se reduzca considerablemente a medida que aumentamos los valores del segundo. Resulta importante destacar que, en esta gráfica, los valores que se muestran en una escala logarítmica debido a que la diferencia del tiempo de cálculo entre las tres formaciones disponibles, y con un número de doscientos drones, es realmente cuantiosa y, usando tal escala, conseguimos reducirlos a un rango más manejable.

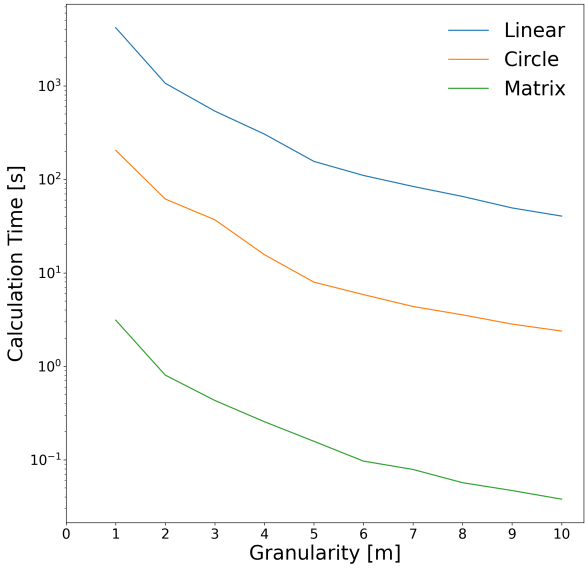


Figura 6.1.2: Tiempo de cómputo según su granularidad.

6.2 Experimento 2: Tiempo de cómputo de los algoritmos realizados.

En este nuevo experimento se tiene como propósito el de comparar el rendimiento referente al tiempo de cómputo del algoritmo CSTH respecto a sus respectivas optimizaciones. Para ello, realizaremos para cada una de las formaciones aéreas posibles una serie de pruebas con distintas cantidades de aeronaves que nos permitan identificar la mejora del rendimiento del algoritmo tras aplicarse las mejoras.

Antes de empezar, resulta primordial comentar la elección del rango de valores empleados para la versión CSTH-RSR. Como ya se ha comentado previamente en el capítulo anterior, esta primera optimización consistía en comparar la posición tridimensional de un dron, situado en alguna de sus posiciones intermedias hacia su destino aéreo, con un conjunto de localizaciones establecidas por un intervalo de valores según su altitud

pertenecientes al resto de aeronaves que conforman el enjambre. De este modo, conseguimos reducir de forma significativa el número de comparaciones efectuadas a lo largo de la ejecución del algoritmo. Resulta de suma importancia encontrar los valores límites para dicho rango que nos permita garantizar una mayor seguridad en la fase de despegue. Este intervalo de valores se obtiene al sumar/restar un parámetro, asignado de manera paramétrica en el constructor de la clase, junto con el valor de la altitud en la que se encuentra el dron que estamos comparando.

Respecto a los parámetros empleados en la actual batería de pruebas, se han empleado los mismos que se han descrito en el primer experimento, es decir, la misma distancia entre las localizaciones terrestres y aéreas, el mismo valor para el margen de error de GPS, y la misma altitud. Además, para la obtención de las posiciones intermedias se ha usado el valor de la granularidad resultante del experimento anterior.

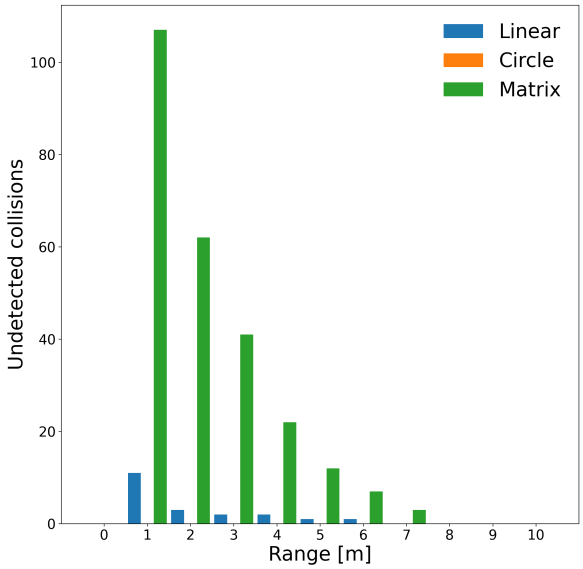


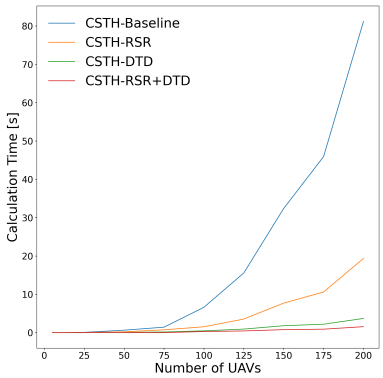
Figura 6.2.1: Colisiones no detectadas según el intervalo de metros empleado.

En la Figura 6.2.1 podemos visualizar los resultados de los experimentos realizados en búsqueda de un valor numérico que nos permita establecer un intervalo con el que asegurar que no se pierde ninguna zona de conflicto. Para ello, emplearemos una cantidad fija de drones equivalente a doscientos para que el número de conflictos sea significativo. En los datos obtenidos en la figura podemos destacar varias cosas interesantes, como es el caso que, para la formación circular, detectemos todas las colisiones independientemente del rango de metros empleado. Sin embargo, en el resto de formaciones, y para las primeras unidades de distancia utilizadas, se verifica que un gran número de potenciales colisiones quedan sin detectar, llegando a valores superiores a cien, como resulta ser el caso de la formación matricial. Tanto en la formación lineal como en la matricial vemos una clara tendencia a que, a medida que aumentamos el intervalo de valores, se consigan detectar más colisiones, hasta alcanzar un punto en el que todas se detectan correctamente. En nuestro caso, a partir de una cifra de ocho unidades asignadas por el usuario (equivalente a dieciséis metros de diferencia entre los límites inferior y superior del intervalo), vemos que se detectan todas las colisiones. Aunque para este experimento en concreto este valor resulta muy elevado debido a que la altitud máxima que pueden alcanzar los drones es de veinte metros, supone no obstante una pequeña mejora respecto al rendimiento del

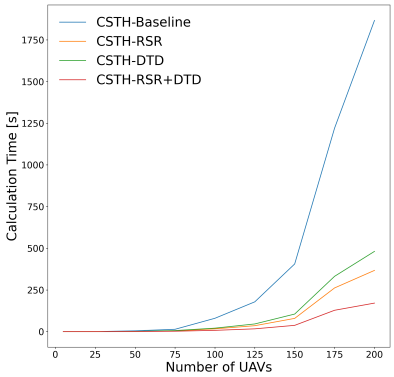
algoritmo CSTH.

Una vez explicada la elección del rango, nos vemos capacitados para proceder a realizar la respectiva comparación del rendimiento en cuanto a tiempo de cálculo. De la optimización CSTH-DTD y la versión combinada (CSTH-DTD+RSR) no se ha comentado nada previamente porque no se requiere ningún elemento adicional para su ejecución aparte de los valores de granularidad e intervalo de metros previamente obtenidos. No obstante, para este cálculo si hay diferencias entre estas tres versiones.

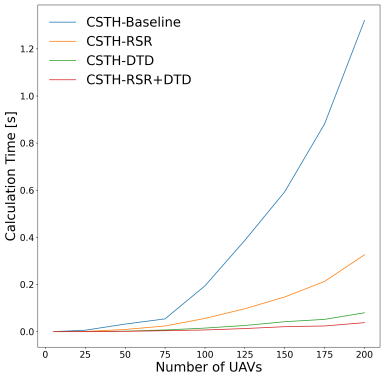
Para extraer los datos de cada una de las versiones disponibles hemos decidido efectuar varios experimentos con cada una de las formaciones aéreas disponibles, empleando un número distinto de drones hasta alcanzar las doscientas unidades. Se ha decidido mostrar los datos para dicho valor máximo porque permite apreciar más claramente la diferencia entre cada una de las versiones. También comentar que los drones están situados en el suelo a una distancia mínima de diez metros entre sí, mientras que en sus posiciones aéreas la distancia entre ellos aumenta a unos veinte metros. Por último, comentar que todas las posiciones finales de cada VANT perteneciente al enjambre disponen de una cuota de veinte metros de altitud.



(a) Circular.



(b) Lineal.



(c) Matricial.

Figura 6.2.2: Comparación del tiempo de cálculo entre los algoritmos de detección.

En la Figura 6.2.2 visualizamos la comparación del tiempo de cómputo de las cuatro versiones propuestas en cada una de las formaciones aéreas disponibles. En la 6.2.2a observamos que existe una clara diferencia entre las versiones del algoritmo a medida que vamos aumentando el número de drones. Con un número menor que setenta y cinco drones, no existen casi diferencias entre el tiempo de cálculo del algoritmo, ya que para las cuatro versiones obtenemos un valor prácticamente similar al segundo. A partir de esta cantidad de aeronaves las diferencias empiezan a ser significativas. Con un valor de doscientos drones vemos que el algoritmo CSTH Baseline le cuesta un total de ochenta y un segundos en garantizar la detección de colisiones, mientras que en la versión CSTH (RSR+DTD) solo nos cuesta un segundo y medio. Dicha esto, podemos afirmar que el

rendimiento de mejora de la versión combinada es siete veces más rápida que la inicial. En lo que respecta al algoritmo CSTH-RSR, y para la misma cantidad de drones comentada en la prueba anterior, obtenemos un tiempo de diecinueve segundos (tres veces más rápida), mientras que en la CSTH-DTD tenemos un valor de cuatro segundos, equivalentes a una eficiencia aproximadamente cuatro veces más alta.

La formación aérea linear es con diferencia la que más tiempo necesita para realizar los cálculos del algoritmo. Este fenómeno se debe a que la distancia existente entre las localizaciones terrestres y sus respectivas localizaciones aéreas es mayor que la del resto. Dicho esto, en la 6.2.2b podemos observar una considerable mejora del tiempo de cálculo entre las versiones existentes del algoritmo. Del mismo modo que sucedía en el caso anterior, a partir de una cantidad de setenta y cinco drones las diferencias derivadas de las optimizaciones propuestas empiezan a ser significativas. Empleando la máxima cantidad de drones usada en el experimento vemos que el tiempo del algoritmo CSTH aumenta hasta un total de treinta y un minutos para realizar los cálculos necesarios para detectar todas las colisiones. Este tiempo resulta algo prohibitivo ya que a este se debería sumar el tiempo de trayecto de las aeronaves desde que despegan hasta que alcanzan su posición aérea. De nuevo la versión combinada vuelve a ser la que mejores prestaciones presenta, permitiendo detectar todas zonas de peligro de colisión en unos tres minutos (ocho veces más eficiente). La optimización CSTH-RSR consigue detectar las colisiones existentes en unos seis minutos (equivalente a unas cinco veces más rápida), mientras que la mejora CSTH-DTD lo consigue en unos ocho minutos (aproximadamente 3/4 veces más eficaz).

Por último, de forma inversa que sucedía en la formación linear, en la formación matricial el tiempo necesario para el cálculo de los algoritmos es muy bajo, ya que sus localizaciones aéreas suelen encontrarse más cercanas entre sí. En la 6.2.2c visualizamos la diferencia del tiempo de cálculo de los algoritmos propuestos. También en esta formación podemos ver que el algoritmo CSTH es el que más tiempo necesita (1,32s) para obtener una solución con dos cientos aeronaves, mientras que la versión combinada de nuevo mejora las prestaciones notablemente, necesitando solo 0,038s. Por último, comentar que la optimización CSTH-RSR requiere un tiempo de 0,326s, mientras que la versión CSTH-DTD lo lleva a cabo en unos 0,08s.

De manera resumida, con los datos obtenidos podemos afirmar que el algoritmo CSTH (DTD + RSR) es la versión más eficiente, ya que consigue obtener un tiempo de cálculo considerablemente menor que las otras variantes en todas las formaciones aéreas disponibles. Además, en cada una de las pruebas realizadas con distintos números de drones, detecta el mismo número de posibles colisiones que la versión inicial, asegurando así la fiabilidad del algoritmo.

6.3 Experimento 3: Comparación entre el tiempo de cálculo de CSTH (RSR+DTD) y ED-CSTA

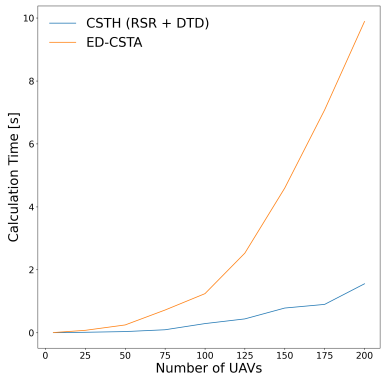
Este experimento surge de la necesidad de comprobar el número de colisiones que no detectamos en las versiones probadas en el anterior experimento. Se verifica que, a pesar de emplear una granularidad igual a dos, es posible que algún conflicto siga sin detectarse, lo cual impide garantizar plenamente la seguridad de la fase de despegue. Una solución ineficiente para verificar el número real de colisiones sería usar un número muy inferior como valor de la granularidad, pero el problema de este enfoque es que la ejecución de los

programas podría tener una duración de varios días, llegando incluso alcanzar el periodo de una semana.

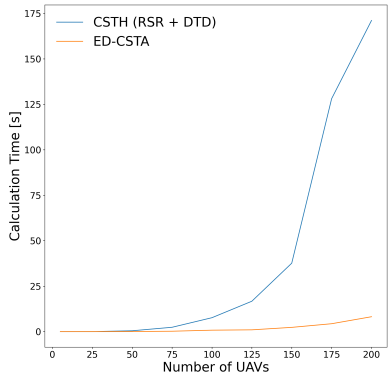
Con la solución que proponemos en el algoritmo ED-CSTA se pretende comprobar que la versión inicial, que utiliza la granularidad, es correcta, intentando conseguir de la forma más exacta posible el número zonas de conflicto existentes. Para ello, antes de mostrar la correspondiente gráfica referente al coste temporal, resultaría interesante conocer la cantidad de colisiones que detectamos según la cantidad de drones que tenemos. Tras múltiples experimentos realizados en ambas versiones, empleando los mismos parámetros de simulación, se ha verificado que en determinados casos existe alguna posible de zona de conflicto que no es detectada. A modo de ejemplo, con un experimento de doscientos drones, y con las formaciones lineal y matricial, se obtiene la misma cantidad de colisiones potenciales, mientras que en la formación circular conseguimos detectar un área de riesgo usando la versión CSTH (RSR+DTD). Ante estos resultados, se ha decidido no introducir una gráfica explicando las posibles colisiones no detectadas, ya que no podríamos deducir una relación evidente.

En vista del comportamiento de ambos algoritmos, confirmamos que el funcionamiento de la versión combinada es la correcta, puesto que la no detección de una colisión la podemos obviar. Esto porque, en nuestros resultados, no contemplamos la variable tiempo, y se podría dar el caso de que, aunque sus trayectorias tiendan ser próximas, puede ser que las aeronaves que causen dicha posible colisión se pasen por dicho punto en instantes de tiempo completamente distintos, nunca llegando a ocasionar una colisión.

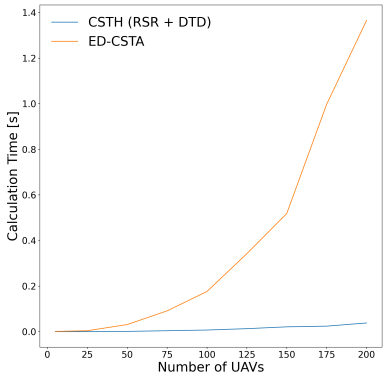
Habiendo confirmado que los cálculos de ambas versiones son correctas, podemos llevar a cabo la comparación de sus tiempos de cómputo. En la Figura 6.3.1 vemos la comparación del tiempo de cómputo de las versiones ED-CSTA y CSTH (RSR+DTD) en cada una de las tres formaciones posibles. Tal y como se puede apreciar, para la formación circular y matricial obtenemos que la versión combinada es más rápida y eficiente, siendo aproximadamente unas diez veces más eficiente. Por otro lado, en la formación linear, la cual requiere un mayor tiempo de cómputo, sucede exactamente lo contrario que en las otras formaciones. Aquí el algoritmo ED-CSTA logran resultados que son más de tres veces de rápidos que su rival, recortando en total algo más de dos minutos.



(a) Circular.



(b) Lineal.



(c) Matricial.

Figura 6.3.1: Comparación del tiempo cómputo entre ED-CSTA y CSTH (RSR+DTD).

En definitiva, la conclusión que extraemos de las gráficas es que, dependiendo de la situación en que nos encontremos, nos será de mayor utilidad emplear una versión u otra, impidiendo así elegir una de ellas por cuestión de prestaciones. Sin embargo, para los experimentos siguientes se ha decidido utilizar el algoritmo ED-CSTA porque su tiempo medio, obtenido a través de sus formaciones, es siempre inferior a la versión CSTH (RSR+DTD). De este modo, se prefiere perder algunos segundos referentes a la circular y matricial para obtener la mejoría de varios minutos en la lineal, ya que su valor es más significativo en términos globales.

6.4 Experimento 4: Comparación del tiempo total para el despegue Secuencial y el Semi-simultáneo.

Una vez elegida nuestra propuesta definitiva, el siguiente paso a realizar será el de comparar el tiempo total obtenido con el del modo de despegue secuencial. En el caso de la modalidad creada en el presente estudio (nombrada semi-simultánea), para determinar el tiempo total asociado al despegue de un enjambre de drones deberemos sumar el tiempo correspondiente al del algoritmo de asignación, al tiempo del cálculo del algoritmo de detección de colisiones, y al tiempo de despegue del enjambre de drones. Este último tiempo lo conseguimos una vez todas las aeronaves hayan alcanzado su posición aérea. Respecto al tiempo del algoritmo secuencial, no incluye el tiempo involucrados en cálculos ya que su implementación es muy simple.

Para la realización de las respectivas pruebas resulta conveniente comentar los parámetros que se han decidido emplear. En primer lugar, se ha decidido ejecutar un total de siete experimentos usando distintas cantidades de aeronaves para cada una de las formaciones aéreas disponibles. A diferencia de experimentos anteriores, la cantidad de drones máxima empleada se verá reducida de los doscientos a ciento cincuenta debido a que, para dichas cifras ya podremos determinar cuáles son las conclusiones del estudio. Otro cambio que se ha efectuado es relativo a la altitud máxima de las posiciones aéreas. En lugar de tener una altitud de veinte metros, se ha decidido aumentar dicha cifra a treinta. Los motivos de este cambio son que, tal y como se ha visto en el capítulo anterior, en la etapa de despegue disponíamos de dos momentos en los cuales las aeronaves ascienden verticalmente: tras elevarse del suelo, y cuando faltaba unos metros para alcanzar su posición final. Para estos dos márgenes de altitud se corresponderá el mismo valor equivalente a cinco metros. Dicho esto, utilizando una altitud de veinte metros, y con los márgenes comentados, cada aeronave se desplazaría diagonalmente demasiado rápido, casi en horizontal. Respecto a la distancia de seguridad, no ha habido ningún cambio con respecto a los anteriores experimentos correspondientes a los ocho metros. Como en esta ocasión disponemos de la interfaz gráfica de ArduSim para realizar los cálculos, en caso de producirse un riesgo de colisión entre dos aeronaves, se notificaría de su existencia, y se procedería a realizar un aterrizaje de emergencia. En los ficheros de registro de ArduSim podremos identificar con facilidad los identificadores de los drones que han causado la colisión con sus respectivas localizaciones tridimensionales, y la distancia existente entre ellas.

Otro valor que no ha tenido cambios en estas pruebas ha sido la distancia terrestre y aérea entre drones, dejándolas en diez y veinte metros, respectivamente. Aunque podríamos ampliar el valor perteneciente a la distancia aérea decidimos no hacerlo porque, en el caso de la formación linear, y utilizando como ejemplo cien aeronaves, ya tendríamos una distancia de unos dos kilómetros entre los extremos.

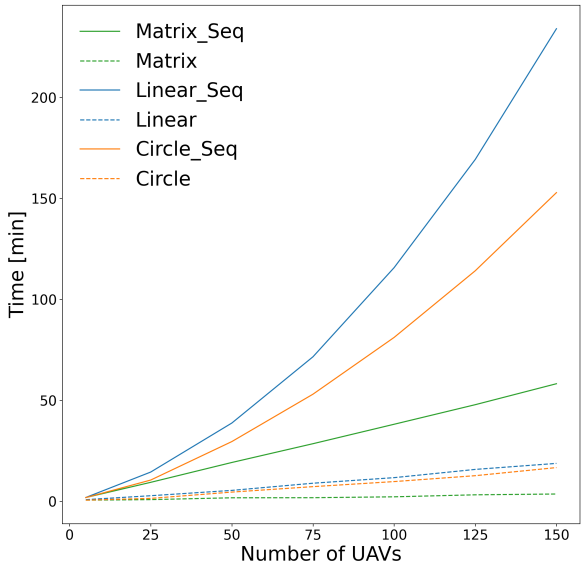


Figura 6.4.1: Comparación del tiempo total de despegue entre el modo Secuencial y el Semi-simultáneo.

En la Figura 6.4.1 apreciamos los tiempos totales, expresados en minutos, de los modos de despegue secuencial y semi-simultáneo, para cada una las formaciones aéreas. Para poder distinguir a la perfección los resultados obtenidos, las líneas continuas harán referencia a las formaciones del algoritmo secuencial, mientras que las discontinuas serán del semi-simultáneo. Profundizando en la figura vemos que, en cada uno de los experimentos realizados con distintas cantidades de aeronaves, el algoritmo semi-simultáneo mejora notablemente las prestaciones obtenidas respecto al secuencial. Entrando en detalle en cada una de las formaciones aéreas, y para el caso específico de lanzar ciento cincuenta drones, vemos que, en la formación matricial, pasamos de tener un tiempo total de 58 minutos a tan solo 3,6 minutos. Este gran cambio se debe a que, para este tipo de formación, con el algoritmo semi-simultáneo se detectan muy pocas zonas de conflicto y, por lo tanto, se requiere agrupar los drones en una cantidad menor de tandas. Otro de los aspectos que explican un tiempo de despegue tan reducido es que esa formación es la más beneficiada por el uso del algoritmo de asignación KMA, lo que significa que el desplazamiento total que tienen que efectuar los drones, situados en el suelo aleatoriamente, es considerablemente inferior al del resto de formaciones. En el caso de la circular, se contrastan las 2 horas y media necesarias frente a los casi 17 minutos con un despegue semi-secuencial. Aquí obtenemos una cifra considerable de colisiones potenciales que derivan en la creación de una gran cantidad de lotes de drones. Por último, la formación linear es la que más tiempo se demora, ya que la distancia total que ha de recorrer el enjambre es más de un mil veces superior a la de la matricial. Dicho esto, de un tiempo total de casi 4 horas se consigue reducirlo hasta los 19 minutos. En esta ocasión, la gran mayoría de VANTs pertenecientes al enjambre se encuentran en posibles zonas de conflicto debido a que sus trayectorias son muy similares; nótese que la mitad de los drones se desplazarán hacia la izquierda, mientras que la otra mitad hacia la derecha para formar una línea en el aire. Por este motivo, la cantidad de tandas de drones será elevada, ya que en la mayoría de situaciones en cada tanda habrá solamente dos aeronaves, cuyas direcciones son opuestas. Un detalle importante a comentar es que, tanto para la formación circular como para la lineal, el disponer de un número elevado de tandas de drones multiplicará el efecto de

añadir un tiempo de espera adicional (3,5s) introducido entre el despegue de cada una de ellas.

6.5 Experimento 5: Estudio de los resultados obtenidos con los distintos algoritmos de asignación

Tras analizar la mejoría obtenida por el modo de despegue Semi-Simultáneo frente al Secuencial, lo último que nos queda por estudiar es el comportamiento del mecanismo detección de colisiones propuesto cuando se combina con los alguno de los algoritmos de asignación disponibles. En este experimento se llevará a cabo una comparación exhaustiva tanto del tiempo de despegue como de la cantidad de tandas de drones existentes en cada uno de ellos.

Como ya se había comentado en los capítulos anteriores, para definir la posición que debe ocupar cada dron en el aire hemos empleado el mecanismo de asignación KMA, ya que proporciona una solución óptima en cuanto a la distancia total que los drones pertenecientes a un enjambre deben recorrer hasta alcanzar sus respectivas posiciones aéreas. De este modo, al usar dicha asignación, se descartaría que varias aeronaves tuvieran que volar hacia otro extremo del enjambre, provocando que el número de áreas en conflicto aumentara significativamente. Sin embargo, existe la incertidumbre de que, si esto sucediera, el número de tandas totales de drones se pudiera ver reducido y, por lo tanto, obtener un tiempo de despegue inferior independientemente de que la distancia fuese más extensa.

En un principio los algoritmos que íbamos a examinar eran los tres que son accesibles desde ArduSim, es decir, los resultados obtenidos por el KMA en el experimento anterior junto con las soluciones alternativas disponibles - heurística y aleatoria - ambas más rápidas. Sin embargo, se ha decidido descartar el planteamiento aleatorio debido a que el tiempo de espera adicional introducido entre el despegue de cada una de las tandas no resulta suficiente para garantizar la seguridad del algoritmo. Este motivo se debe a que, como la asignación de las posiciones es totalmente aleatoria, una determinada aeronave perteneciente a una tanda puede disponer de una posición aérea lejana y, mientras está intentando alcanzarla, otro dron de la tanda consecutiva despega, llegando a sobrepasar la distancia de seguridad establecida. En el caso de los dos primeros algoritmos nombrados esto no ocurre porque las distancias entre sus posiciones terrestres y aéreas son próximas a la óptima, y se evita recorrer metros de más.

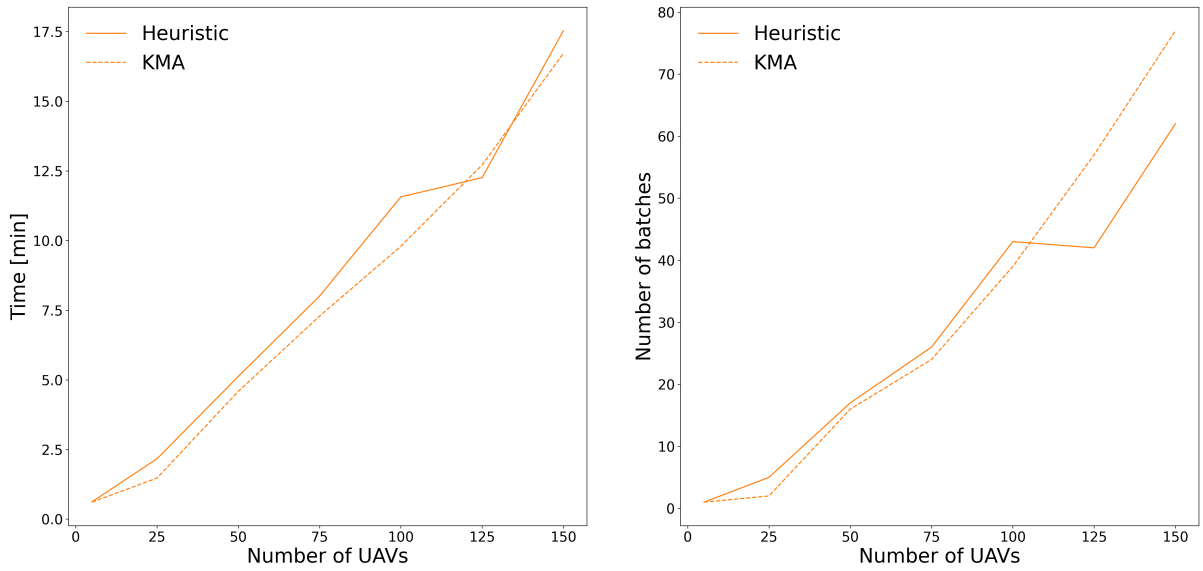
Para intentar solucionar el problema del algoritmo de asignación con el enfoque aleatorio decidimos incrementar el tiempo de espera empleado alcanzando las cifras de 4.5s y 5.5s, pero tras observar los resultados vimos que continuaban existiendo múltiples colisiones, sobre todo en la formación linear. En lo que respecta al resto de las formaciones investigadas, se logró tener un despegue seguro.

Una vez explicado lo anterior, nos centraremos en obtener los datos del algoritmo heurístico para posteriormente compararlos con los del KMA. Tal y como comentamos anteriormente, la heurística proporciona una solución ligeramente subóptima tras comprobarse en formaciones aéreas regulares, a cambio de ser más rápida en la ejecución de los cálculos de cómputo en comparación al KMA.

Para la obtención de los datos con este nuevo algoritmo de asignación vamos a usar los mismos parámetros de simulación utilizados en el experimento anterior para así poder reutilizarlos. La única diferencia es que, en esta ocasión, almacenaremos la cantidad de

colisiones y tandas en cada uno de los experimentos realizados. Esta nueva información la emplearemos como apoyo para la futura interpretación de los datos obtenidos de ambos algoritmos.

En la Figura 6.5.1 visualizamos el tiempo total de la fase de despegue y el número de tandas generadas de los dos algoritmos de asignación analizados para la formación aérea circular. En la 6.5.1a se puede ver que la heurística proporciona unos tiempos más elevados que su rival, exceptuando el experimento ejecutado con 125 drones. En esta ocasión, justo como vemos en la 6.5.1b, la heurística que siempre había tenido un número de tandas superior a KMA, requiere unas quince tandas menos. Sin embargo, con esta gran diferencias de lotes totales, solo se consigue obtener medio minuto de mejoría respecto al otro algoritmo de asignación. Por último, en la prueba ejecutada con 150 VANTs, observamos que la heurística sigue manteniendo una cantidad menor de tandas, pero está vez su tiempo total de despegue es casi un minuto más lenta que el KMA. Con este último experimento podemos confirmar la importancia de minimizar la distancia total de desplazamiento, objetivo de ambos algoritmos, debido a que, con un número menor de lotes, se obtiene un tiempo mayor porque sus aeronaves deben de recorrer una mayor distancia y, por lo tanto, implica un incremento del tiempo resultante.



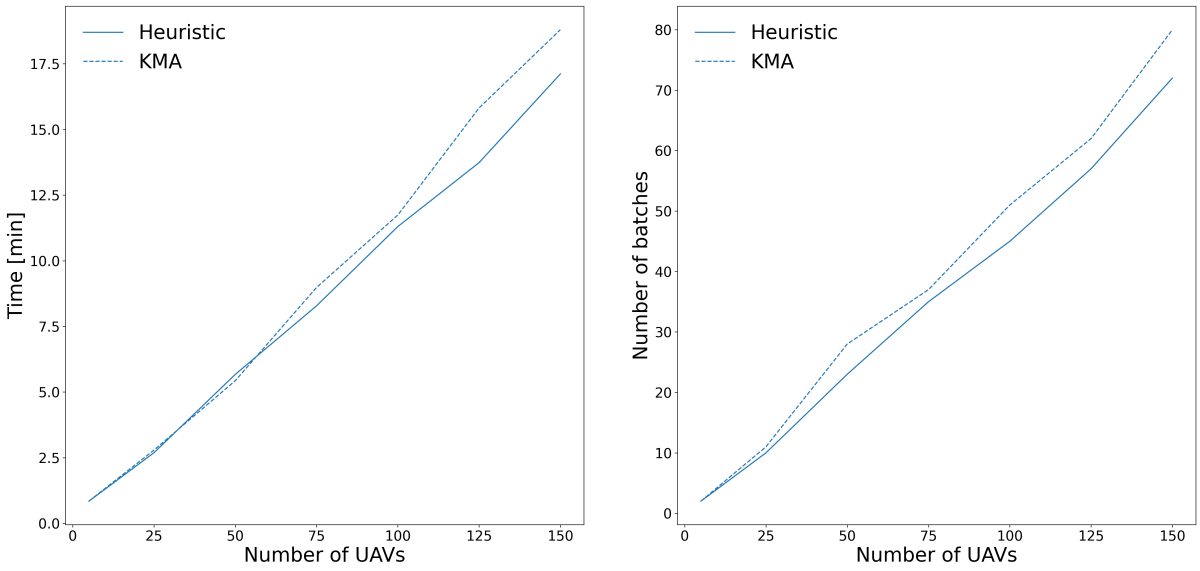
(a) Tiempo de despegue

(b) Número de tandas

Figura 6.5.1: Tiempo total de los modos de despegue Secuencial y Semi-simultáneo en la formación circular.

En la Figura 6.5.2 vemos la comparación de los datos recogidos por los dos algoritmos de asignación para la formación aérea linear. Aquí podemos observar que, salvo en los primeros experimentos realizados hasta la cifra de 50 VANTs, y para los cuales el tiempo de despegue es similar, la heurística proporciona una solución más eficiente tanto en el tiempo total como en el número de lotes. En este caso, como la distancia total que ha de recorrer el enjambre para estructurarse en una formación linear es muy elevada, no resulta tan significativa la diferencia existente entre ambos algoritmos. Por este motivo, visualizamos que no ocurre lo mismo que en la circular, donde con una cantidad menor de tandas se obtenía peor tiempo de despegue. Por lo tanto, para la prueba llevada a

cabo con 150 aeronaves, la heurística consigue tener una mejoría de un minuto y medio respecto el KMA.

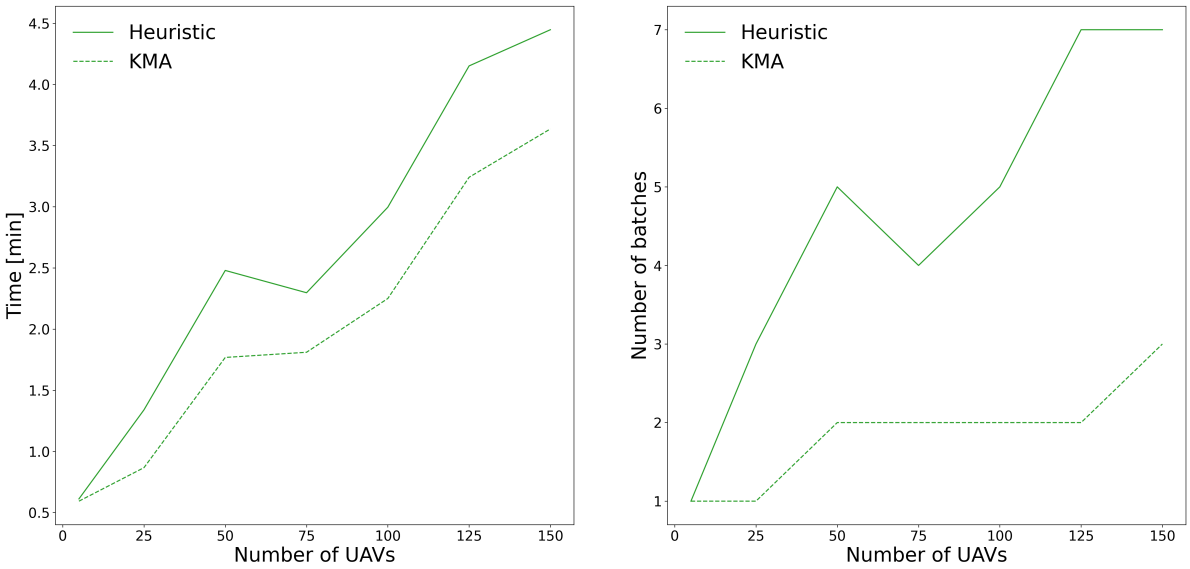


(a) Tiempo de despegue

(b) Número de tandas

Figura 6.5.2: Tiempo total de los modos de despegue Secuencial y Semi-simultáneo en la formación linear.

Finalmente, tenemos los datos resultantes de la comparativa de los dos algoritmos respecto a la formación aérea matricial (ver Figura 6.5.3). En esta ocasión, aún siendo la formación con menor distancia de recorrido total, se comprueba que KMA proporciona un mejor rendimiento en todos los experimentos realizados. La justificación de los tiempos recogidos se debe principalmente a que el número de lotes conseguidos empleando el KMA es el óptimo debido a que, a pesar de tener una cantidad considerable de VANTs, se logra realizar los despegues en tan solo dos o tres tandas.



(a) Tiempo de despegue

(b) Número de tandas

Figura 6.5.3: Tiempo total de los modos de despegue Secuencial y Semi-simultáneo en la formación matricial.

En conclusión, tras examinar el comportamiento de los dos algoritmos de asignación para las tres formaciones posibles, no nos podemos decantar claramente por una opción debido a que, mientras el algoritmo KMA obtiene mejor tiempo de despegue para las formaciones matricial y circular, no lo consigue para la linear. Sin embargo, es importante recalcar que con ambos algoritmos se obtienen tiempos similares, con la excepción de la matricial donde la heurística proporciona un rendimiento algo bajo comparado con el KMA.

Capítulo 7

Conclusiones

En los últimos años, los avances tecnológicos en el campo de los vehículos aéreos no tripulados han crecido exponencialmente permitiéndonos ampliar su utilización en muchos sectores. Estas aeronaves nos pueden ayudar a realizar tareas relativamente peligrosas de manera autónoma mejorando nuestra calidad de vida. Sin embargo, su corta autonomía y falta de inteligencia muchas veces hacen que, individualmente, los drones sean insuficientes para completar de manera eficiente diferentes tareas complejas. Para satisfacer la necesidad de hacer frente a tales tareas, el uso de enjambres de drones nos permite aumentar las capacidades potenciales, ofreciéndonos tolerancia a fallos o ampliando el área de seguimiento.

En la presente tesis de máster se presenta un planteamiento novedoso para el despegue seguro y eficiente de enjambres de drones. Valoramos que los retos planteados en este estudio se han alcanzado de forma completa y satisfactoria. A continuación, resumimos brevemente cómo se han resuelto dichos problemas.

En primer lugar, se han generado dos algoritmos de detección de colisiones (ED-CSTH y CSTH-RSR+DTD) los cuales permiten optimizar el tiempo de cómputo necesario para detectar posibles zonas de riesgo. La primera de ellas garantiza el descubrimiento de todas las aéreas de conflicto, mientras que la segunda logra el mismo resultado en general, exceptuando determinadas ocasiones (escasas) en las alguna colisión queda sin detectar. Junto a estos algoritmos, se implementa también un mecanismo de generación de tandas de drones que permite agruparlos de manera a asegurar que, en su despegue y desplazamiento, se garantiza la distancia de seguridad en cualquier momento.

Por otro lado, se lleva a cabo un minucioso análisis de los resultados obtenidos a través del simulador ArduSim empleando tres formaciones aéreas regulares. Además, se demuestra que nuestra solución mejora significativamente el tiempo de despegue en comparación con el enfoque secuencial, proporcionando la misma fiabilidad y seguridad que este último.

Por último, se ha examinado el comportamiento de la versión propuesta cuando combinada con los algoritmos de asignación KMA y Heurística. En este caso, no se ha podido elegir claramente una opción ganadora debido ya que depende de la formación aérea empleada.

7.1 Trabajos futuros

Para poner fin a este estudio, han surgido una serie de trabajos futuros que podrían ampliar o mejorar las prestaciones obtenidas por el algoritmo propuesto. En primer lugar,

se estudiará como reducir el número total de tandas de VANTs resultantes en alguna de las formaciones regulares examinadas. De este modo, podríamos ahorrarnos mucho tiempo en el despegue al permitir que más aeronaves puedan despegar en simultáneo sin provocar ningún riesgo de colisión. En segundo lugar, en este proyecto no contemplamos la variable tiempo, es decir, obtenemos las zonas que no cumplen con la distancia de seguridad a través de su trayectoria con respecto al resto de drones del enjambre. En determinados casos, podríamos detectar que un VANT causa un conflicto con otro cuando este ya ha despegado y, por lo tanto, se encuentra cerca de su localización aérea destino. Para hacer frente a esta mejora se necesitaría cambiar el código fuente completamente. En tercer punto, se debería analizar como eliminar el tiempo de espera que se introduce tras el despegue de cada una de los lotes generados. Aunque este tiempo garantiza que no existan riesgos de colisión entre drones pertenecientes a tandas consecutivas, su incorporación supone una gran demora. Por último, habría que optimizar la comunicación existente entre las aeronaves para evitar posibles pérdidas de tiempo.

Lista de Acrónimos

KMA Kuhn-Munkres Algorithm

MAVLink Micro Air Vehicle Link

VANET Vehicular Ad-Hoc Network

VTOL Vertical take off and landing

Bibliografía

[1] F. Fabra, C. T. Calafate, J. C. Cano, and P. Manzoni, "Ardusim: Accurate and real-time multicopter simulation," *Simulation Modelling Practice and Theory*, vol. 87, pp. 170–190, 2018. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1569190X18300893

[2] I. Intelligence, "Drone technology uses and applications for commercial, industrial and military drones in 2021 and the future," 2021, last visited: 2021-10-05. [Online]. Available: https://www.businessinsider.com/drone-technology-uses-applications

[3] D. Hernández, J. M. Cecília, C. T. Calafate, J.-C. Cano, and P. Manzoni, "The kuhn-munkres algorithm for efficient vertical takeoff of uav swarms," in *2021 IEEE 93rd Vehicular Technology Conference (VTC2021-Spring)*. IEEE, 2021, pp. 1–5.

[4] D. Hambling, "Israel's combat-proven drone swarm may be start of a new kind of warfare," last visited: 2022-01-01. [Online]. Available: https://www.forbes.com/sites/davidhambling/2021/07/21/israels-combat-proven-drone-swarm-is-more-than-just-a-drone-swarm/?sh=4f5e6e0e1425

[5] G. Bai, Y. Li, Y. Fang, Y.-A. Zhang, and J. Tao, "Network approach for resilience evaluation of a uav swarm by considering communication limits," *Reliability Engineering and System Safety*, vol. 193, p. 106602, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0951832019300171

[6] M. Campion, P. Ranganathan, and S. Faruque, "Uav swarm communication and control architectures: a review," *Journal of Unmanned Vehicle Systems*, vol. 7, no. 2, pp. 93–106, 2019. [Online]. Available: https://doi.org/10.1139/juvs-2018-0009

[7] J. Yasin, M.-H. Haghbayan, J. Heikkonen, H. Tenhunen, and J. Plosila, "Formation maintenance and collision avoidance in a swarm of drones," in *Formation Maintenance and Collision Avoidance in a Swarm of Drones*, 09 2019.

[8] C. Cheng, G. Bai, Y.-A. Zhang, and J. Tao, "Resilience evaluation for uav swarm performing joint reconnaissance mission," *Chaos: An Interdisciplinary Journal of Nonlinear Science*, vol. 29, no. 5, p. 053132, 2019. [Online]. Available: https://doi.org/10.1063/1.5086222

[9] Y. Zhang, W. Feng, G. Shi, F. Jiang, M. Chowdhury, and S. H. Ling, "Uav swarm mission planning in dynamic environment using consensus-based bundle algorithm," *Sensors*, vol. 20, no. 8, 2020. [Online]. Available: https://www.mdpi.com/1424-8220/20/8/2307

[10] F. Fabra, W. Zamora, J. Masanet, C. T. Calafate, J.-C. Cano, and P. Manzoni, "Automatic system supporting multicopter swarms with manual guidance," *Computers and Electrical Engineering*, vol. 74, pp. 413–428, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0045790618323577

[11] Actualidadaeroespacial, "Un total de 1.874 drones iluminaron el cielo de tokio en la inauguración de los jjoo," 2021, last visited: 2022-01-02. [Online]. Available: https://cutt.ly/FOYi8Be

[12] Sputniknews, "Miles de drones sobrevuelan san petersburgo para conmemorar el fin de la segunda guerra mundial," 2020, last visited: 2022-01-02. [Online]. Available: https://cutt.ly/WOYiRgV

[13] F. Fabra, J. Wubben, C. T. Calafate, J. C. Cano, and P. Manzoni, "Efficient and coordinated vertical takeoff of uav swarms," in *2020 IEEE 91st Vehicular Technology Conference (VTC2020-Spring)*, 2020, pp. 1–5.

[14] J. Wubben, C. Calafate, and J.-C. Cano, "A solution for the efficient takeoff and flight coordination of uav swarms," Ph.D. dissertation, UPV, 07 2021.

[15] J. Page and G. S, "A self-organized swarm simulator," in *A Self-Organized Swarm Simulator*, 10 2012.

[16] L. Jeroncic, "Drone swarm simulator," in *Drone Swarm Simulator*, 5 2021.

[17] Y. Zeng, R. Zhang, and T. J. Lim, "Wireless communications with unmanned aerial vehicles: opportunities and challenges," *IEEE Communications Magazine*, vol. 54, no. 5, pp. 36–42, 2016.

[18] J. Cai, S. Gunasekaran, A. Ahmed, and M. Ol, "Changes in propeller performance due to ground proximity," in *Changes In Propeller Performance Due to Ground Proximity*, 01 2019.

[19] A. Rodríguez, "Tipos de drones y sus características," 2020, last visited: 2021-10-05. [Online]. Available: https://iberfdrone.es/tipos-drones-y-caracteristicas/

[20] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A system for autonomous flight using onboard computer vision," in *PIXHAWK: A system for autonomous flight using onboard computer vision*, 05 2011, pp. 2992–2997.

[21] DronProfesional, "Qué debes conocer de las baterías lipo," 2020, last visited: 2021-10-05. [Online]. Available: https://dronprofesional.com/blog/que-debes-conocer-de-las-baterias-lipo/

[22] ArduPilot, "Copter home," last visited: 2021-10-23. [Online]. Available: https://ardupilot.org/copter/

[23] MaVLink, "Mavlink developer guide," last visited: 2021-10-23. [Online]. Available: https://mavlink.io/en/

[24] GRCDev, "Ardusim," last visited: 2021-10-11. [Online]. Available: https://github.com/GRCDEV/ArduSim

[25] J. Wubben, I. Catalán, M. Lurbe, F. Fabra, F. J. Martinez, C. T. Calafate, J.-C. Cano, and P. Manzoni, "Providing resilience to uav swarms following planned missions," in *2020 29th International Conference on Computer Communications and Networks (ICCCN)*, 2020.

[26] P. Radoglou-Grammatikis, P. Sarigiannidis, T. Lagkas, and I. Moscholios, "A compilation of uav applications for precision agriculture," *Computer Networks*, vol. 172, p. 107148, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S138912862030116X