UNIVERSITAT POLITÈCNICA DE VALÈNCIA

etsinf
Escola Tècnica Superior d'Enginyeria Informàtica

Escola Tècnica Superior d'Enginyeria Informàtica
Universitat Politècnica de València

Soluciones para la escalabilidad de enjambres de vehículos aéreos no tripulados

Trabajo Fin de Grado

Grado en Ingeniería Informática

Autor: Clérigues Ferrer, David

Tutor: Tavares de Araújo Cesariny Calafate, Carlos Miguel

Co-tutor: Wubben, Jamie

Curso 2020-2021

Resumen

Hoy en día el sector de los UAV está demostrando tener un sin fin de posibilidades, y entre las muchas de sus aplicaciones destacan las misiones de rescate, la búsqueda de objetivos, los controles de zonas, etc... Estos aparatos no solo operan individualmente, sino que son capaces de operar como un enjambre de cara a realizar misiones más complejas. Dentro de este campo existen un gran conjunto de proyectos centrados en tecnologías para UAVs, aunque todos se centran en operar dichos enjambres con soporte de infraestructura, y restringidos a un rango de comunicaciones concreto. En este trabajo se busca extender un protocolo para la gestión de enjambres de UAVs llamado MUSCOP, buscando lograr que pueda llevar a cabo misiones con nodos ubicados más allá del rango de alcance del nodo central. Las pruebas de vuelo usadas para validar la solución propuesta han sido realizadas con el simulador Ardusim, donde se ha buscado en todo momento medir el tráfico de la red que se genera con la nueva solución, así como los tiempos de vuelo asociados a la ejecución de la misión. Los resultados alcanzados son prometedores, y validan la aplicabilidad de la solución desarrollada.

Palabras clave: UAV, Enjambre, Comunicaciones, MUSCOP, Ardusim.

Resum

Actualment el sector dels UAV està demostrant tindre una infinitat de possibilitats. Algunes de les aplicacions que més destaquen son l'ús de la tecnologia per a les missions de rescat, la cerca d'objectius, els controls de zones, etc... Aquests aparells no sols operen individualment, sinó que són capaços d'operar com un eixam de cara a realitzar missions més complexes. Dins d'aquest camp existeixen un gran conjunt de projectes centrats en tecnologies per a UAVs, encara que tots es centren en operar aquests eixams amb suport de infraestructura, i restringits a un rang de comunicacions concret. En aquest treball es busca estendre un protocol per a la gestió d'eixams de UAVs anomenat MUSCOP, buscant aconseguir que puga dur a terme missions amb nòduls situats més enllà del rang d'abast del nòdul central. Les proves de vol usades per a validar la solució proposada han sigut realitzades amb el simulador Ardusim, on s'ha buscat en tot moment mesurar el trànsit de la xarxa que es genera amb la nova solució,

així com els temps de vol associats a l'execució de la missió. Els resultats aconseguits són prometedors, i validen l'aplicabilitat de la solució desenvolupada.

Paraules clau: UAV, Eixam, Comunicacions, MUSCOP, Ardusim

Abstract

Nowadays the UAV sector is proving to have endless possibilities, and among its many applications we can find rescue missions, the search for objectives, area monitoring, etc. These aircrafts not only operate individually, but are also capable of operating as a swarm in order to carry out more complex missions. Within this field there is a large set of projects focused on technologies for UAVs, although all focus on operating such swarms with infrastructure support, and restricted to a specific range of communications. This work seeks to extend a protocol for the management of UAVs swarms called MUSCOP, seeking to achieve that it can carry out missions with nodes located beyond the range of the central node. The flight tests used to validate the proposed solution have been carried out with the Ardusim simulator, where at all times it has been sought to measure the network traffic generated with the new solution, as well as the flight times associated with the execution of the mission. The results achieved are promising and validate the applicability of the developed solution.

Keywords : UAV, Swarm, Communications, MUSCOP, Ardusim.

Índice de contenidos

Índice de Ilustraciones

Índice de tablas

1. Introducción

Hace treinta años era descabellado pensar que el ciudadano común fuese capaz de poseer un teléfono móvil. Hoy en día no solo es posible, sino que, además, el tamaño total que ocupa dicho artilugio no suele superar el de una mano estándar. Todo este avance ha sido gracias al hambre voraz e incesable que tiene el ser humano ante el progreso y la evolución.

Hoy en día hay muchos avances que aún están por descubrir y, por otro lado, hay otros que ya están en unas fases más estandarizadas como bien son los drones y sus aplicaciones, objeto de esta memoria.

Los drones han sido aplicados a distintos contextos, desde el militar por parte de la *United States Army* para misiones de reconocimiento, hasta contextos donde se ven involucrados los desastres naturales y la vigilancia u observación de entornos.

Aunque, ¿Cómo es posible evaluar y ejecutar dichas comandas? Para estos propósitos se realizan estudios y distintas pruebas de evaluación sobre simuladores dando lugar a la observación y corrección de posibles comportamientos defectuosos.

En del Departamento de Informática de Sistemas y Computadores (DISCA), situado en la Universidad Politécnica de Valencia, se desarrolló una aplicación capaz de abarcar todo este ámbito gracias al arduo trabajo del estudiante Francisco Fabra en su tesis doctoral, y a la supervisión del tutor Carlos T. Calafate. Dicho simulador, llamado ArduSim [1], es capaz de ejecutar distintas pruebas de vuelo virtuales y físicas mediante el empleo de distintos protocolos, donde cada uno se encarga de abarcar un ámbito operacional distinto. El enfoque principal de esta memoria se basa en la extensión de un protocolo ya implementado, el protocolo MUSCOP [2], capaz de gestionar un enjambre de drones que tenga que seguir una misión planificada.

1.1. Motivación

El campo de trabajo de los drones abarca cada día nuevas aplicaciones innovadoras, y es por ello que cada día es más común observar un gran abanico heterogéneo de funcionalidades. Los enjambres son un foco de interés ya que, mediante su uso, se puede satisfacer un sinfín de misiones desde el rastreo de individuos para su rescate hasta monitorizaciones de fuegos o fronteras.

Los enjambres suelen estar coordinados para que todos sus elementos se ubiquen dentro del rango de comunicaciones radio, ya que actualmente no existen soluciones en el mercado que ofrezcan una solución para la coordinación de enjambres dispersos (con nodos más allá del rango de comunicaciones del coordinador). Una de las motivaciones principales para el desarrollo de este proyecto es el suplir esta dependencia.

Tener enjambres cuyos nodos estén más allá del rango de cobertura del nodo central permitiría tener más flexibilidad de actuación y, por ende, se podría cubrir nuevos horizontes.

A parte de la motivación técnica para el desarrollo de este trabajo, existe también una motivación a nivel personal. Dicha motivación reside en mi deseo por conocer aún más a fondo todo este ámbito para poder potenciar y elevar el nivel de mi carrera profesional. Además, también me ayuda a demostrarme a mí mismo que todo el esfuerzo invertido en estos años ha dado su fruto.

1.2. Objetivos

El enfoque principal de este proyecto es proponer una extensión al protocolo MUSCOP para poder dotarlo de capacidades de comunicación multi-salto. Al ampliar las funcionalidades de comunicación actuales el protocolo será capaz de permitir la gestión de vuelo en enjambre con formaciones donde existan drones ubicados más allá del rango de transmisión del dron maestro. Los objetivos planteados para lograr satisfactoriamente el cumplimiento de tales operaciones son los siguientes:

- Análisis, Entendimiento, Comprensión y Visión del funcionamiento actual de MUSCOP sobre ArduSim.
- Modificación de la implementación actual de MUSCOP para que cada dron pueda ser capaz de conocer a sus vecinos en entornos multi-salto.
- Actualización y rediseño de la estructura de comunicación actual para poder establecer un orden jerárquico de comunicación entre los drones
- Pulir el funcionamiento del protocolo, permitiendo al dron central conocer aquellos drones que están fuera del rango de comunicación como aquellos que sirven de puente entre estos.
- Observación y realización de distintos experimentos para corroborar el correcto funcionamiento del proyecto junto con un análisis e interpretación de los resultados.

1.3. Estructura de la memoria

La estructura empleada para el desarrollo de la memoria sigue un modelo detallado donde se cubren los aspectos más relevantes con un total de seis capítulos.

- El primer capítulo tiene como enfoque mostrar los aspectos generales de la memoria, siendo estos la introducción del contexto, las motivaciones que me

llevan a trabajar en esta área, los objetivos a desarrollar y la estructura de la propia memoria.

- Por otra parte, el capítulo 2 abarca trabajos relacionados más relevantes en esta área, incluyendo un análisis crítico de los mismos.

- El capítulo 3 da una visión general del software que se usa como punto de partida (simulador ArduSim), así como del protocolo de gestión de enjambres que se pretende mejorar (MUSCOP).

- A continuación, el capítulo 4 tiene como objetivo primordial formalizar y detallar las mejoras que se ha introducido al protocolo MUSCOP.

- En el capítulo 5 se incluyen validación y pruebas mediante una serie de experimentos de vuelo en enjambre.

- Finalmente, en el capítulo 6 se presentan las conclusiones principales de este trabajo, y se hace referencia a trabajos futuros.

2. Estado del arte

El uso e investigación de los vehículos aéreos no tripulados (VANT), o como comúnmente se conocen, Unmanned aerial vehicle (UAV), abarca un gran abanico de posibilidades, y es por ello por lo que, a medida que pasa el tiempo, se van consensuando nuevas vertientes que permiten a estos artilugios ser dotados con funciones únicas para satisfacer un sinfín de actividades; desde aquellas más simples, como podría ser transportar información entre distintos puntos sin ningún tipo de coordinación hasta aquellas más complejas como coordinar distintos UAV para dar forma a un enjambre que opere conjuntamente en misiones de rescate. A continuación, se exponen una serie de trabajos heterogéneos realizados por ilustres investigadores en esta área de investigación.

Todo enjambre ha de estar coordinado para llevar a cabo satisfactoriamente su misión, y es por ello por lo que una de las ideas clave para cumplir el objetivo se basa en la aplicación de consenso sobre enjambres a gran escala de forma robusta y eficiente presentado en el trabajo [3]. Al intentar abarcar un consenso común sobre un enjambre de dimensiones desmesuradas provoca situaciones críticas que han de ser solventadas para el uso correcto de éste, como bien son los tiempos de vida (TTL) y la propia robustez frente a las pérdidas de los nodos. Según se expone en el propio documento un consenso tradicional no es capaz de solventar dichos problemas, y es por ello por lo que se ha tenido que reinventar un nuevo tipo de consenso. El consenso propuesto consigue suprimir el fallo de los tiempos de vida empleando un protocolo basado en árboles de expansión dinámicos que consiguen adecuarse a cualquier tipo de topología dinámica. Por otra parte, la robustez ha sido completamente rediseñada empleando una combinación de un consenso basado en la media, junto con la aplicación de '*Grey Prediction*' y el *'Molly-Reed Criterion'*.

Los enjambres son composiciones complejas que requieren un alto grado de concurrencia para poder funcionar en niveles óptimos. Para poder suplir la coordinación de una forma efectiva dentro de este proyecto presentado en [4] los autores han desarrollado un método para poder coordinar cada UAV mediante el uso de una arquitectura dinámica y flexible. Este otro proyecto concluye con una investigación que contiene una característica bastante distintiva, ya que para la propia planificación de las misiones se hace uso de un '*Global Mission Planner*' (GMP), el cual se encarga de la asignación y monitorización de las distintas misiones avanzadas mediante el empleo de un *'Agent Mission Planner'* (AMP). El agente en cuestión tiene como finalidad monitorizar y proporcionar cada tarea distinta a cada UAV determinado. Mediante el proceso de coordinación se pretende, como objetivo principal, poder lograr satisfactoriamente el cumplimiento de todas aquellas misiones dinámicas en tiempo real, siendo éstas, por ejemplo, misiones de rescate. Es indudable que el sector informático crece cada día de forma abrumadora y con ello aparecen tanto avances como problemas. Un problema crucial del sector son los ataques informáticos, donde estos van más allá de las tecnologías convencionales como los ordenadores o los propios teléfonos móviles.

Otro trabajo presente abarca un planteamiento interesante donde pretende mejorar la seguridad de los enjambres empleando redes software basadas en las arquitecturas actuales. En el trabajo [5], los autores han presentado formalmente dicha idea. Según mencionan los autores, hay enjambres que deben ser expuestos a espacios públicos, y es por ello por lo que la seguridad de estos debe ser reforzar. En una primera instancia se plantea hacer uso de redes basadas en el software (SDN) para posteriormente contrastar su funcionalidad frente a una arquitectura más clásica la cual emplea el protocolo AODV y tablas de enrutamiento. Se llega a la conclusión que, al implementar SDN, se consigue monitorizar la actividad de la red, y de esta forma se puede llegar a detectar distintos tipos de ataques y actuar en consecuencia adaptando las reglas de seguridad.

Los enjambres microscópicos son una vertiente que crece exponencialmente con el paso del tiempo, y dentro del nuevo proyecto [6] los autores afirman haber creado una variante barata empleando distintos tipos de tecnologías . Las características principales que abarcan el articulo están relacionadas con el uso de redes Wi-Fi mesh entrelazando las operaciones con un middleware ROS2 y estableciendo los rangos de comunicaciones con tecnologías UWB.La escalabilidad de las comunicaciones y de la localización es lograda al empotrar datos que son publicados por un ROS2 predeterminado dentro del rango de mensajes proporcionado por UWB.

Con el paso del tiempo nacen nuevas tecnologías para satisfacer el hambre voraz del ser humano por la evolución. Las redes 5G son la nueva vertiente de las telecomunicaciones, y no es descabellado pensar en éstas como una posible mejora en el ámbito de los enjambres; es por ello que, en esta investigación, los autores en [7], han expuesto como enfocar esta tecnología 5G hacia los enjambres, además de analizar cómo se puede emplear blockchain para reforzar la seguridad de la red. La propuesta para emplear el 5G se basa en la búsqueda de una tecnología capaz de asegurar una tasa alta de transmisión. Al ser una tecnología en desarrollo y que opera en nuevas frecuencias es posible que alcance velocidades de transmisión de datos superiores, siendo estas de múltiples gigabits. Por otra parte, en el artículo se menciona el uso de las tecnologías Mobile Edge Computing (MEC) para poder dar fuerza a la topología, ya que esta red distribuida hace uso de aquellos dispositivos que están al borde como nodos de altas capacidades de cómputo. Además, se hace énfasis en el uso del blockchain debido a que esta tecnología es capaz de incrementar y endurecer la seguridad aportando confidencialidad, integridad y disponibilidad. Finalmente, se concluye que estas tecnologías son aptas para ser incorporadas, aunque cabe destacar que su aplicación puede ser compleja debido a la falta de recursos para simular distintos escenarios óptimos. No obstante, se debe tener en cuenta el peso al que es sometido cada UAV, junto con el consumo de las propias baterías.

Para finalizar, la propuesta que abarca esta memoria consiste en el desarrollo de un protocolo completamente operativo y funcional que rompa los estándares de comunicación actual del sector, es decir, que sea capaz de llevar a cabo distintas misiones planificadas donde las distancias entre algunos de los elementos de la formación superen el rango de transmisión máximo de los interfaces inalámbricos. Esto lleva a la obligación y necesidad de diseñar un protocolo de comunicaciones multi-salto.

3. Visión general de Ardusim y MUSCOP

Ardusim [8] es un simulador en tiempo real innovador el cual hace uso del protocolo MAVLink (Protocolo de mensajería ligero empleado para la comunicación entre drones) y de un canal inalámbrico virtual compartido para establecer la conexión entre los distintos drones. El propio simulador es capaz de satisfacer la necesidad de ejecutar pruebas de vuelo con enjambres (o con un solo dron) guiadas por una misión preestablecida donde, además, al ser de código abierto, disponible en [9], permite a nuevos usuarios desarrollar sus propios protocolos de vuelo. La ilustración 1 muestra la interfaz del simulador en ejecución.

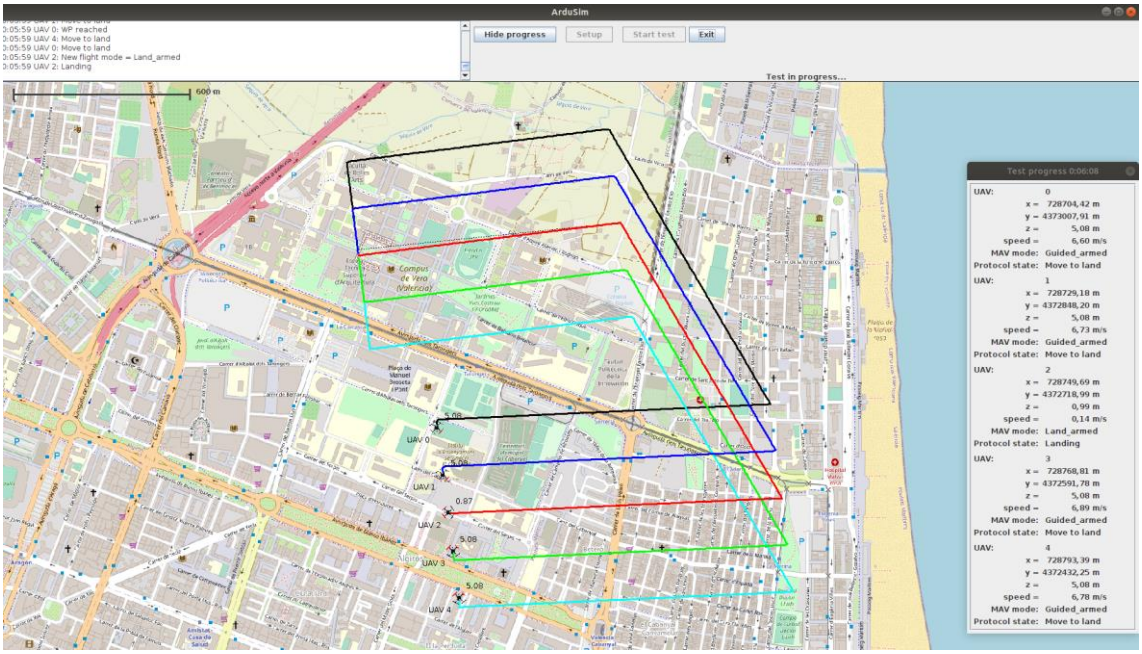

Ilustración 1 - Interfaz de Ardusim.

El simulador ofrece un gran abanico de funcionalidades, pero en esta memoria no se pretende detallar aquellos conceptos complejos que estén fuera del enfoque establecido, como los tipos de conexión disponibles o los paquetes empleados para el desarrollo y extensión de MUSCOP.

La arquitectura general está compuesta por un mecanismo capaz de detectar las colisiones entre los UAV (*UAV Collision detector*), por un gestor gráfico de parámetros que permite dotar a la simulación de características únicas (*GUIControl,* aunque también existe una interfaz *commandline* para hacer pruebas de forma automática) y, por último, por otro mecanismo capaz de simular el envío por difusión (*Simulated broadcast*). Los hilos empleados por los UAVs han de ser siempre un mínimo de dos,

siendo estos uno para enviar datos (*talker*) y otro para recibirlos (*listener*) con un posible uso de un tercero que dote a éstos con el comportamiento lógico del protocolo empleado (*logic*). Las capas más internas contienen conceptos más abstractos y complejos fuera del alcance de esta redacción.

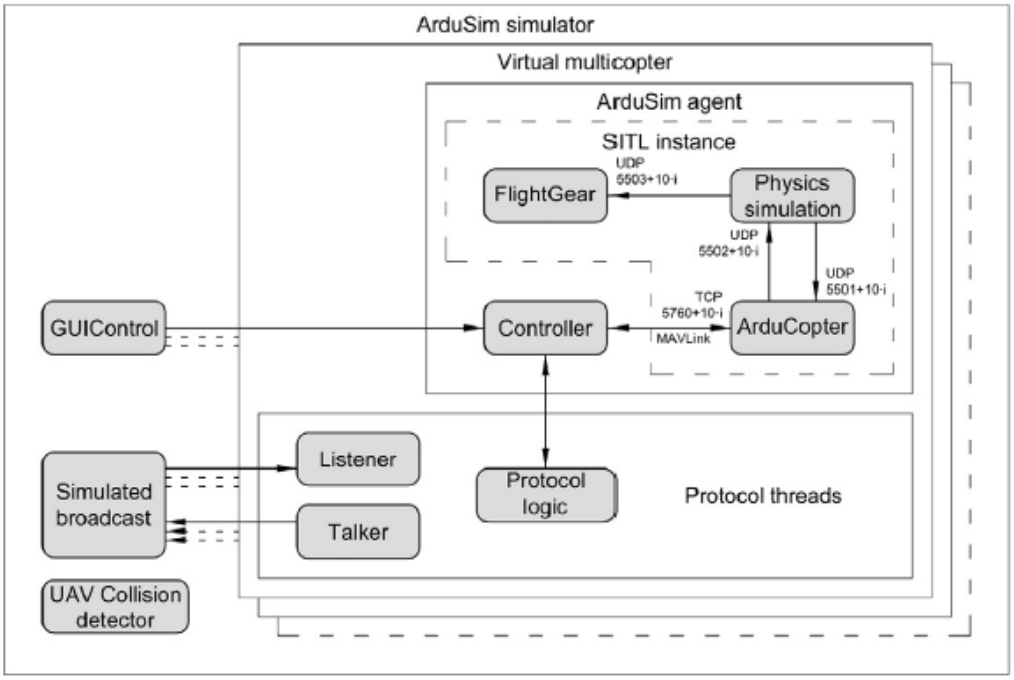

Ilustración 2 - Arquitectura de Ardusim.

Por otra parte, para dar un poco de contexto al usuario, se pretende explicar brevemente cuales son los protocolos disponibles actualmente junto con las formaciones disponibles. De esta forma, la lista mostrada a continuación explica con brevedad los protocolos actualmente disponibles:

- Un protocolo diseñado para evitar las colisiones entre distintos drones es **MBCAP** [10], el cual se encarga de establecer una prioridad entre los distintos UAV que están siguiendo una misión para que, en caso de interferir entre sus misiones, éstos sepan tratar la situación y así evitar las colisiones.

- **Mission** es uno de los protocolos primerizos de Ardusim donde, los drones involucrados se encargan de seguir una misión preestablecida.

- El protocolo **Follow Me** [11] diverge frente a los demás en su comportamiento ya que necesita de un piloto real que controle al maestro. Los vehículos no tripulados asignados como esclavos se encargan de seguir al maestro en tiempo real.

- **Shake Up** es un protocolo dinámico el cual permite a los drones reconfigurarse en plena misión, siendo capaces de reducir los riesgos de colisión.

- **Vision** [12] requiere de un elemento hardware extra sobre el UAV para que funcione. Dicho UAV se equipa con una cámara que utiliza los datos captados para poder aterrizar de una forma más precisa sobre un marcador Aruco. Dentro del articulo referenciado se encuentra explicado en detalle su funcionamiento.

- Por último, el protocolo **MUSCOP** [13] va a ser explicado con más detalle ya que esta memoria hace uso de él como base de todo el proyecto. Este protocolo permite a los distintos UAV ser coordinados entre sí en el vuelo mediante una jerarquía maestro-esclavo para poder satisfacer la ejecución de una misión preestablecida correctamente. En la ilustración 3 se muestra la máquina de estados vigente del protocolo.

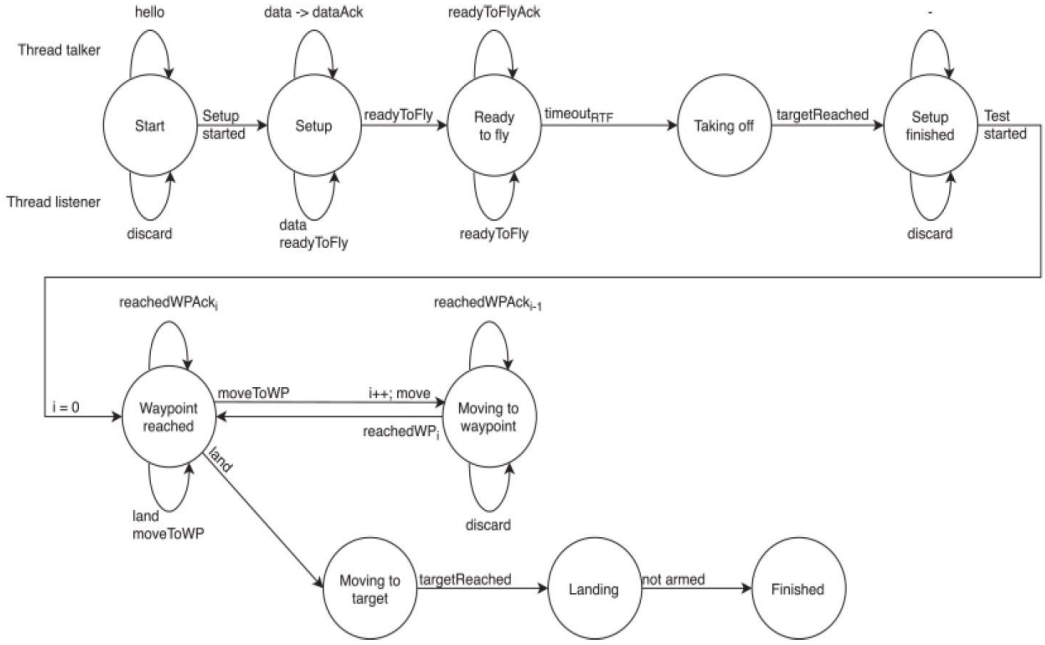

Ilustración 3 - Máquina de estados adaptada de MUSCOP.

Los tres primeros estados son de configuración, y tienen lugar cuando la formación aún no ha sido desplegada en el aire. Estas tres primeras fases son las encargadas de comunicar los drones entre sí, y permiten al maestro reconocer a sus esclavos y que estos reciban todos los datos necesarios para poder conocer al detalle la misión que se ejecutará posteriormente.

Los dos estados siguientes son el paso intermedio entre las operaciones terrestres y las aéreas. *Taking off* se encarga de posicionar cada dron en su altitud y latitud correspondiente; una vez el punto asignado a sido alcanzado, los drones notificarán su llegada. Cuando el maestro haya recibido todos los

reconocimientos necesarios dará esta fase por finalizada, dando paso a la finalización de los preparativos.

Los estados *Waypoint Reached* y *Moving to waypoint* son los encargados de gestionar todas y cada una de las operaciones aéreas que permiten el transcurso de la misión. La primera fase sirve para que los esclavos notifiquen su presencia en el enjambre indicando su identificador y el punto de ruta. El maestro se encarga de registrar tanto el identificador como el tiempo de recepción del mensaje hasta que consigue almacenar a todos los esclavos.

Llegados a este punto el maestro define como objetivo el siguiente punto en la ruta y se dedica a notificar este nuevo valor a los esclavos mientras él cambia al estado *Moving to waypoint*. Por una parte, mientras que el maestro ha conseguido actualizar su estado, los esclavos siguen trabajando en *WP Reached*, es por esto por lo que, al recibir el nuevo valor del maestro éstos proceden a incrementar su valor de ruta y a cambiar al nuevo estado. Cabe destacar que los esclavos siempre están registrando los identificadores y tiempos de mensaje recibidos por parte de los otros drones independientemente del valor de punto de ruta recibido.

Una vez que todo el enjambre se encuentra en el nuevo estado *Moving to WP* se procede a realizar un intercambio de mensajes entre todos para almacenar de nuevo los identificadores y los tiempos de recepción hasta llegar al nuevo destino donde si no es el último *Waypoint* se procederá de nuevo a ejecutar las mismas operaciones.

MUSCOP se diferencia de los protocolos actuales debido a su capacidad de adaptación, es decir, con este protocolo el enjambre es capaz actualizarse en caso de sufrir pérdidas es por esto por lo que en estos dos últimos estados se almacenan los tiempos de recepción junto con el identificador. En cada ejecución del estado *Waypoint Reached* se comprueba que ningún tiempo asociado a un dron del enjambre sea superior al tiempo de vida establecido, ya que si se llega a cumplir dicha condición se procederá a considerarlo muerto y no se tendrá más constancia de él. La ilustración 4 muestra un escenario donde el enjambre está compuesto por un maestro y cuatro esclavos en formación linear sin pérdidas, las líneas discontinuas marcan el rango de comunicaciones del maestro englobando a los drones alcanzables.

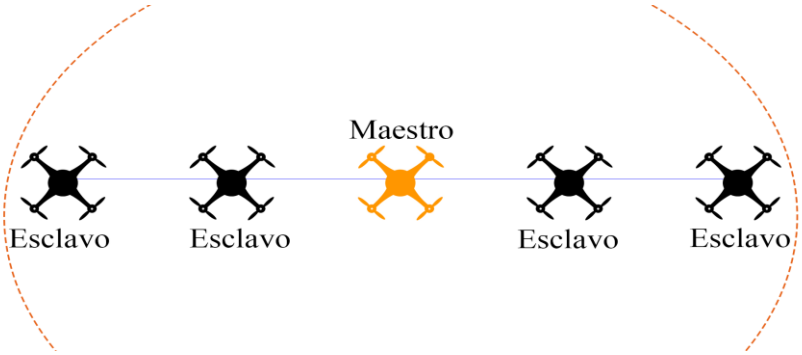

Ilustración 4 – Ejemplo de formación básica en MUSCOP.

Por otra parte, supongamos que el maestro de la ilustración 4 muere. ¿Qué pasaría ahora con todo lo explicado? El comportamiento que adaptaría MUSCOP frente a esta situación sería ajustar de nuevo el enjambre de manera que, dependiendo el rango de comunicaciones establecido, se podría seguir manteniendo un único enjambre con un nuevo maestro y 3 esclavos o, por otro lado, dos enjambres pequeños con dos nuevos maestros capaces de cumplir satisfactoriamente la misión. En la ilustración 5 se muestra dicho comportamiento, remarcando sobre cada circulo discontinuo los nuevos enjambres generados.

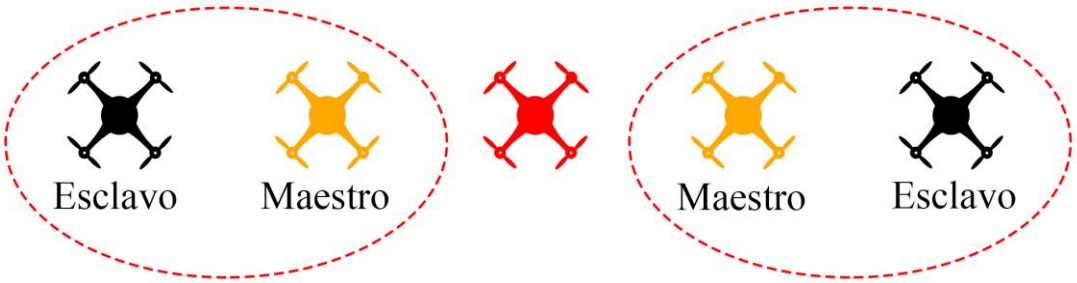

Ilustración 5 - Funcionamiento de MUSCOP.

Por último, tras llegar al último punto, se da paso a los tres últimos estados. *Moving to target* es utilizado por los esclavos para moverse hacía una posición relativa cercana al maestro, siendo esta posición enviada por él. La fase de *Landing* sirve para aterrizar cada dron, y, para finalizar, la fase *Finished* se encarga de cerrar todas las comunicaciones y dar por finalizada la operación.

Cada protocolo tiene una misión propia establecida, la cual puede ser modificada mediante el código fuente, aunque las posibles formaciones sí están disponibles en cada uno de ellos, siendo estas las que se, muestran en la ilustración 6 (los colores de los drones representan el recorrido realizado al despegar desde el suelo):

- Linear: Dicha formación tiene como objetivo situar todos y cada uno de los drones en una línea recta separados entre sí de forma homogénea.

- Circular: Los distintos vehículos no tripulados son situados de forma que lleguen a formar una circunferencia entre sí.

- Random: La formación, como su nombre indica, se encarga de generar posiciones aleatorias para situar cada componente del enjambre en un lugar completamente distinto en cada ejecución.

- Split-Up: Esta es posiblemente la configuración más polivalente ya que nos permite seleccionar con cuantos clústeres queremos trabajar. Principalmente se pretende trabajar siempre con dos o tres clústeres ya que empleando un único clúster nos proporciona un comportamiento similar a la formación linear y no suele ser lo óptimo. Si el valor del clúster es superior a uno generará unas formaciones donde cada clúster quedará aislado de los otros, permitiendo de esta forma una comunicación única con el maestro. Al emplear MUSCOP podremos observar que si maestro muere se generaran tantos enjambres como clústeres tengamos, es decir, en caso de emplear 3 clústeres y sufrir una pérdida del maestro, generará 3 nuevos enjambres independientes, donde cada uno tendrá un maestro distinto.

- Compact Mesh: La formación de malla compacta tiene similitudes encontradas respecto a las formaciones en matriz, pero la posición de los UAV suele divergir un poco con el objetivo de minimizar la ocupación espacial del enjambre.

- Regular & Compact Matrix: Ambos tipos de matrices emplean el mismo patrón, siendo la diferencia principal entre ambas que una comprime el enjambre para minimizar la superficie necesaria, mientras que la otra no.

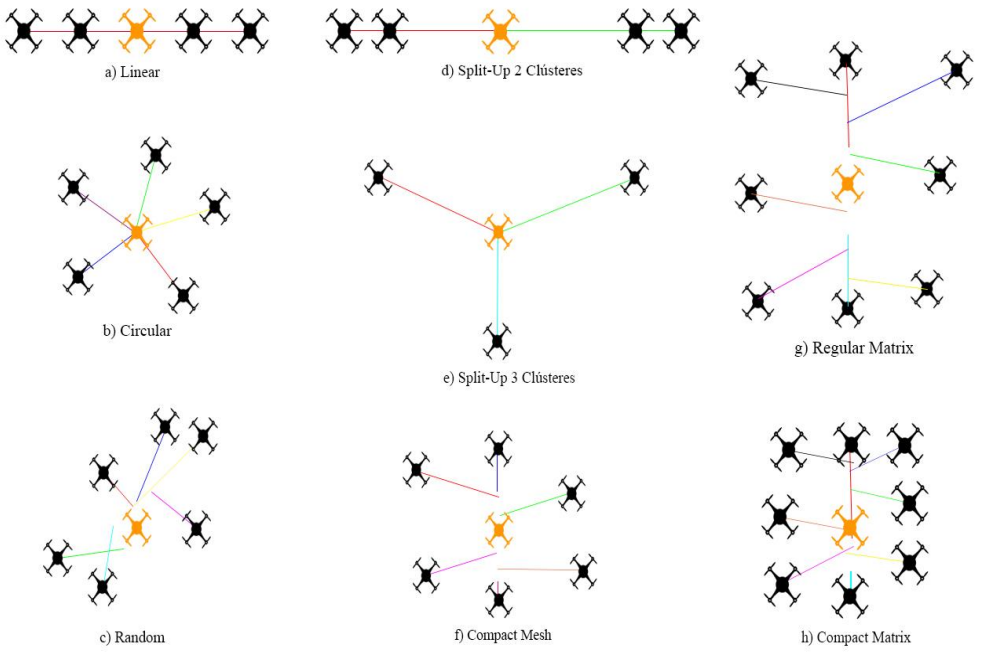

Ilustración 6 - Formaciones actualmente disponibles

4. Planteamiento del Problema

Este cuarto capítulo de la memoria está dedicado a realizar un análisis de los requisitos necesarios para abordar los problemas vigentes. No obstante, también se pretende exponer las propuestas planteadas en una primera instancia junto con el nuevo diagrama de estados adaptado que nos permitirá detallar la implementación una vez se adecue a las necesidades.

4.1. Requisitos

El contexto actual de MUSCOP permite experimentar con la escalabilidad siempre y cuando todos los UAVs del enjambre estén en el rango de comunicaciones del nodo maestro. Sin embargo, al intentar trabajar con rangos superiores a los permitidos, nos encontramos con el problema de que los nodos se ubican fuera del alcance radio del maestro. La ilustración 7 muestra el ejemplo del problema expuesto, donde se comprueba que el dron maestro, en este caso C, solo es capaz de comunicarse con sus nodos vecinos (B y D), pero no con los nodos A y E, ya que éstos se encuentran fuera de su alcance radio.

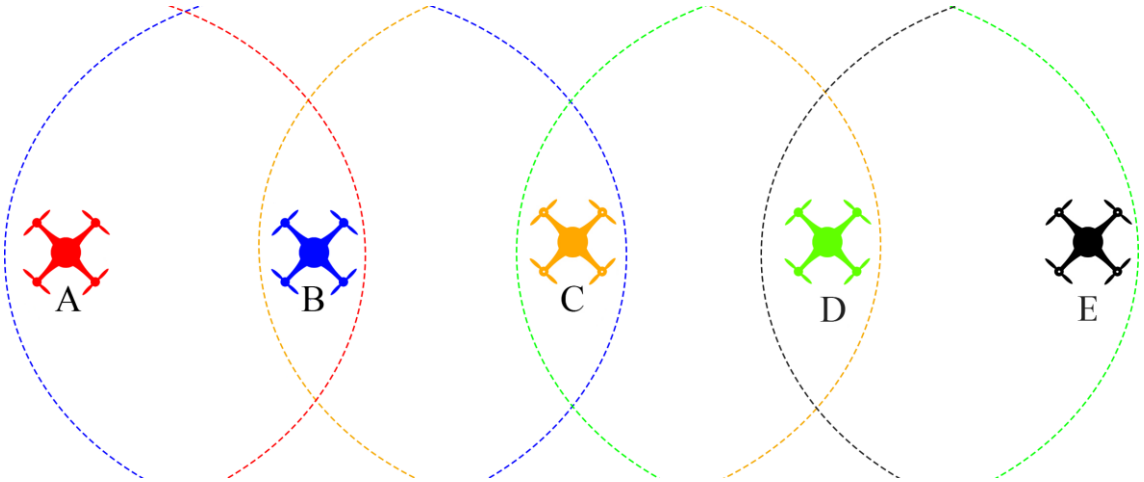

Ilustración 7 - Problema de la alcanzabilidad radio en entornos multi-salto.

Una solución para el problema representado en la figura es habilitar las comunicaciones multi-salto, permitiendo que los nodos actúen como routers, reenviando los paquetes que reciben. En el ejemplo de la figura, el nodo B es capaz de actuar como relay entre en maestro (nodo C) y el nodo A, habilitando así que el maestro consiga comunicarse con ese nodo esclavo también.

En el ámbito del protocolo MUSCOP, los nodos fuera de alcance no permiten al maestro reconocer a todos los esclavos que tiene registrados en fases previas al vuelo, y

debido a esto surge otro nuevo problema en la ejecución del protocolo al llegar al final de la fase *taking off*. Este estado hace uso de un nuevo hilo distinto al *talker* y al *listener,* en concreto a la clase *take off helper,* la cual es la encargada de asegurarse que todos los drones están correctamente desplegados en el aire y funcionando mediante los reconocimientos para así tener un vuelvo seguro, por lo que se puede considerar esta funcionalidad como un "mecanismo de seguridad". La presencia de los nodos fuera del alcance del maestro no permite avanzar en la simulación ya que bloquea toda interacción posible al estar esperando indefinidamente la confirmación de llegada de todos los drones registrados; esto se aplica tanto al maestro como a los esclavos.

El contexto analizado, concluye que para abordar estos problemas los UAVs tienen que ser capaces de mantener la consistencia del enjambre, es decir, se tienen que implementar unas funcionalidades capaces de registrar todos y cada uno de los drones pertenecientes al enjambre independientemente del rango de comunicaciones. De esta forma el dron maestro, por ejemplo, será capaz de reconocer a un dron esclavo que esté fuera de su rango de comunicaciones. Dentro de este punto, se da a entender la necesidad de la implementación de una red ad-hoc multi-salto capaz de suplir las carencias previamente mencionadas. Además, otro requisito necesario para abordar los problemas restantes debe ser la capacidad de anular el bloqueo indefinido asociado a la implementación actualmente disponible de MUSCOP.

4.2. Propuesta a nivel conceptual

Al tener un radio de transmisión limitado tenemos un gran abanico de propuestas, desde emplear esclavos específicos para hacer *relaying* (concepto empleado para indicar retransmisión, donde se pretende emplear nodos del enjambre únicamente como retransmisores del maestro), hasta el uso de una retransmisión secuencial donde todos los esclavos tienen como objetivo hacer *relaying*. No todas las ideas son tan eficientes ni óptimas, razón por la cual se necesita analizar el contexto detenidamente para dar con la propuesta correcta, y poder formalizar una respuesta acorde que permita abarcar correctamente una red ad-hoc multi-salto.

En rasgos generales, la propuesta principal busca determinar el número de saltos hacia el nodo maestro, y elegir los nodos pasarela, que son aquellos encargados de retransmitir. Este número de saltos tiene que ser definido de forma rigurosa siguiendo una numeración en función de la mejor información recibida por los nodos vecinos de un determinado nodo. La propagación de los mensajes ha de ser bidireccional para llegar a todos los nodos, y en formato *multicast* ya que Ardusim a día de hoy, no es capaz de emplear el *unicast*. La ilustración 8 muestra el concepto en su formato más abstracto, las líneas azules representan la distancia actúa, en saltos, de los drones frente al Maestro.

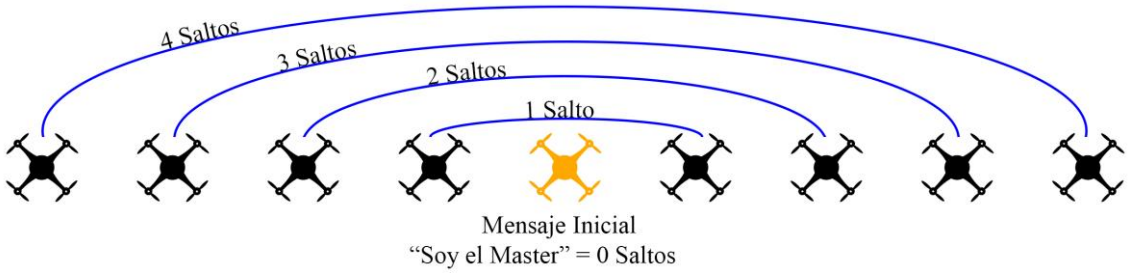

Ilustración 8 - Propuesta genérica inicial.

Los conceptos generales son clave para mantener una base sobre la cual trabajar, aunque para llegar a dicha idea primero se han de definir una serie de pasos a seguir. En una primera instancia, se propone un nuevo estado para MUSCOP donde cada nodo del enjambre aprenderá cuáles son sus vecinos. Esta nueva fase de reconocimiento ha de llevarse a cabo en el mismo instante para cada dron, manteniendo de esta forma la concurrencia. La duración propuesta para abarcar esta nueva fase debe buscar un equilibrio entre eficacia y resiliencia ante pérdidas, y es por ello por lo que se plantea fijar un tiempo estimado de 8 segundos (Se ha fijado dicho tiempo para observar el comportamiento de la ejecución frente a una carga de trabajo media). Durante este periodo de tiempo (ajustable dependiendo del nivel de escalabilidad buscado) se pretende que cada nodo difunda un mensaje de reconocimiento de vecinos con una periodicidad de 0,5 segundos (también adaptable).

Para comprobar la validez de estos mensajes se busca obtener una lista de vecinos estables, y para ello, se deben recibir al menos 3 mensajes consecutivos para considerarse fiable. Este paso adicional es requerido ya que el simulador trabaja mediante UDP, y este protocolo no tiene ningún tipo de mecanismo fiable para gestionar la pérdida de paquetes; el tener mensajes con numeración consecutiva busca emular un tipo de fiabilidad similar a la proporcionada por TCP. La ilustración 9 muestra el comportamiento ideal de la propuesta.

Ilustración 9 - Paso 1, Descubrir vecinos.

El segundo paso está dedicado a la diseminación de mensajes del maestro hacia los esclavos. Para empezar, el maestro envía un mensaje que incluye el listado de todos sus vecinos. De esta forma, cada nodo al recibir un mensaje, independientemente de si proviene del maestro o de otro esclavo, lo propaga solamente si se cumplen tres condiciones a la vez: (1) su número de saltos es superior al del mensaje recibido; (2) su identificador está en el listado recibido, y (3) consigue alcanzar nodos no incluidos en el

listado recibido (a los que añade los suyos propios, e incrementa el número de saltos antes de retransmitir). La ilustración 10 contiene los valores esperados de una ejecución exitosa.

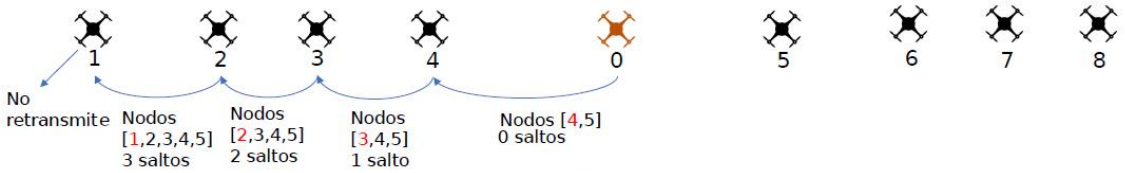

Ilustración 10 - Paso 2, propagación del listado.

Por último, el tercer paso tiene un enfoque *unicast*, pero debido a la falta de este mecanismo dentro del simulador se pretende trabajar con *multicast,* aunque simulando el comportamiento *unicast*. En primer lugar, como etapa previa, cada nodo almacena el nodo origen del primer mensaje generado por el maestro que ha recibido de un vecino estable, tal y como se muestra en la ilustración 11.

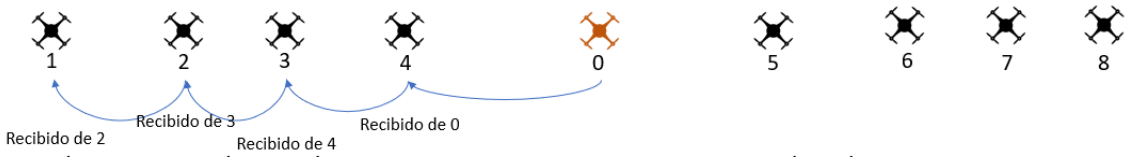

Ilustración 11 - Paso 3, comunicación maestro-esclavo.

Seguidamente, en la respuesta hacia el maestro, envía sus reconocimientos (ACK) únicamente hacia ese primer nodo, que también los reenvía siguiendo la misma regla, estando este comportamiento representado en la ilustración 12.

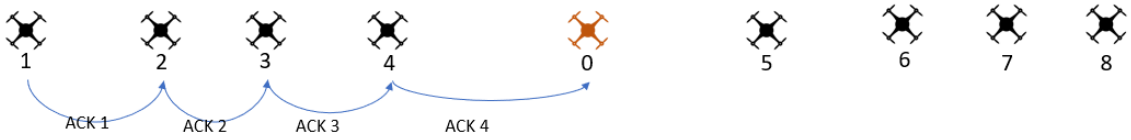

Ilustración 12 - Paso 3, comunicación esclavo-maestro.

4.3. Máquina de estados adaptada: modelo provisional

Las ideas planteadas como requisitos han requerido ser desarrolladas como conceptos previos los cuales han llevado a la creación de nuevos estados, en concreto, a

un único estado llamado *Neighbor Discovery*. La solución provisional requiere modificar el diagrama de estados presentado previamente.

El nuevo estado se ejecutará tras finalizar *Setup Finished*, cuando la ejecución ha sido puesta en marcha. Se define que los valores enviados por el *talker* del nuevo estado serán reconocimientos (los cuales contendrán el identificador del nodo en cuestión) para que estos sean tratados por el *listener* de sus nodos adyacentes, siendo estos capaces de retransmitir al *talker* la información que deben propagar.

Cuando todos los pasos hayan sido completados, desde el descubrimiento de vecinos hasta la comunicación esclavo extremo – maestro, se procederá a finalizar *Neighbor Discovery* para dar paso al siguiente estado: *Waypoint Reached*. En la ilustración 13 se muestra el resultado esperado.

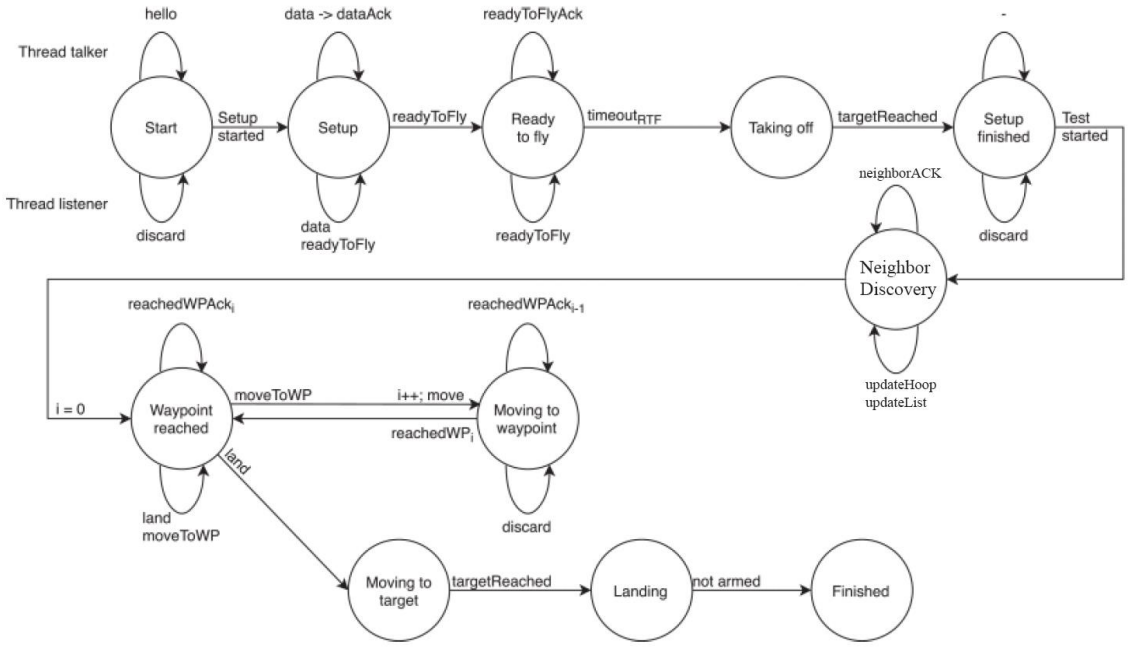

Ilustración 13 - Máquina de estados adaptada: modelo provisional.

Llegados a este punto de la memoria, aún no se ha abarcado el desarrollo, por lo que todo lo mostrado en los 3 primeros subapartados de este capítulo puede distar del resultado final planteado.

4.4. Desarrollo de la solución

La propuesta a nivel conceptual ha dejado las bases bien definidas para abarcar una solución válida, aunque durante el desarrollo han ido surgiendo algunos cambios al respecto. Las modificaciones han afectado a la máquina de estados previamente propuesta, ya que se buscaba abarcar la solución en un único estado. El desarrollo

actual a conseguido establecer dos nuevos estados; *Neighbor Discovery* y *hoopUpdate* (aunque este es un concepto general ya que en verdad equivale a cuatro estados que se explicarán posteriormente). Ambos estados emplean la misma estructura para dar fiabilidad a los mensajes, aunque cada uno establecerá sus propias operaciones para tratar los datos recibidos en el *listener*. Por otra parte, se ha conseguido adecuar el comportamiento de los estados vigentes posteriores a *Setup Finished*.

Tabla 1 - Estructura general para dar fiabilidad a los mensajes

```
Leer el Identificador
LastMSHUAV.put(ID, consecutiveTime)
IF NeighMSG.containsKey(ID)
  Msg_consecutivos = NeigMSG.get(ID)
  IF consecutiveTime – LastMSHUAV.get(ID) > 500
    NeigMSG.replace(ID, 1)
  ELSE IF Msg_consecutivos == 3 && !Neighbors.contains(ID)
    OPERAR
  ELSE
    NeigMSG.replace(ID, ++Msg_consecutivo)
ELSE
  NeigMSG.put(ID, 1)
```

La tabla 1 muestra el pseudocódigo general de la estructura que da soporte a la fiabilidad. Esta sección de código viene dada tras asegurarse de que el tipo de mensaje leído es el correcto. Las variables *NeigMSG* y *LastMSHUAV* son mapas donde el primero se encarga de gestionar cuentos mensajes consecutivos se han recibido de un nodo, mientras que el segundo se encarga de almacenar el tiempo de llegada de cada mensaje (este tiempo se recoge al inicializar cada iteración del bucle). Los fragmentos del pseudocódigo resaltados en rojo indican que cada método hará un uso distinto de dicha operación.

El estado *Neighbor Discovery* desarrollado permite a cada UAV conocer a sus vecinos adyacentes, manteniendo así una lista de ellos en el listener. La explicación al detalle se desglosa en los siguientes puntos dedicados a los hilos *listener* y al *talker* del protocolo:

- Para realizar el envío de datos se hace uso de un buffer. Primero es necesario limpiar el buffer ya que, de esta forma, evitamos enviar datos innecesarios (datos de fases previas que aún siguen almacenados) y, así, evitamos la excepción del *buffer underflow* (este problema se da cuando se reciben menos datos de los esperados o un valor no esperado). Una vez se tiene el buffer vacío se escribe, en un primer lugar, el tipo de mensaje, siendo éste de tipo *Neighbor Discovery*. Es necesario indicar siempre el tipo de mensaje primero para que pueda tratarse debidamente y no cause errores en la lectura. Seguidamente el nodo añade al buffer su identificador y, tras esto, se realiza un *flush* del buffer para traspasar los datos de una región temporal a otra permanente. Con los datos bien estructurados se procede a construir el mensaje, siendo éste una copia de la posición indicada del buffer. Antes de realizar el envío de datos es necesario instanciar los tiempos que van a ser utilizados (en milisegundos). La

primera variable, *totalTime*, se encarga de tener asignado el tiempo necesario para poder ejecutar *Neighbor Discovery*; en este caso propuesto se asigna una constante con valor equivalente a 8 segundos. Por otra parte, la segunda variable, *cicleTime*, guarda el instante actual de ejecución. El bucle empleado se encarga de enviar mediante broadcast el mensaje construido previamente, siempre y cuando el tiempo total (*totalTime*) sea mayor a 0. Los ajustes de tiempo son empleados para controlar el bucle y para establecer un valor que permita a los hilos dormir unos pocos milisegundos. En cada pasada del hilo, al tiempo total se le hace la diferencia con el intervalo propuesto del envío de mensajes, unos 0,5 segundos. Para dormir el hilo se hace una diferencia entre *cicleTime* y el instante actual de ejecución. Al finalizar el bucle se da paso al siguiente estado, el cual varía dependiendo del nodo. La tabla 2 muestra el seudocódigo descrito.

Tabla 2 - Seudocódigo NeighborDiscovery Talker

```
Reinicio del buffer
Escritura de valores
Flush del buffer
Construcción del mensaje
Instanciación de cicleTime y totalTime
WHILE totalTime > 0
  sendBroadcast
  Ajustes de cicleTime y totalTime
  sleep()
END
IF soy el maestro
  currentState.set(Sending_ACK_Center)
ELSE
  currentState.set(Waiting_ACK_Center)
```

La ilustración 14 muestra el envío de mensajes realizado por el *talker* durante la ejecución de este estado.

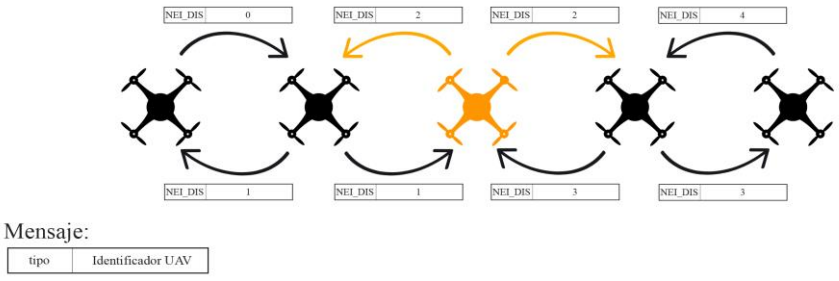

Ilustración 14 – Envío de identificadores.

- Mientras el estado actual sea *Neighbor Discovery* se operará para obtener la lista de vecinos dentro del *listener*. Para poder realizar las operaciones pertinentes, primero se debe recibir un mensaje, en caso de recibir uno se

almacenan los datos en un buffer. Primero se hace una lectura del tipo de dato si este es del tipo *Neighbor Discovery*, se procede a seguir la estructura general de fiabilidad de los mensajes. Previamente se ha explicado el funcionamiento de la fiabilidad, es por eso por lo que, en esta sección, se explica las operaciones especificas realizadas durante el transcurso de este estado. Tras comprobar que el emisor ha enviado tres mensajes consecutivos y que, además, no contiene al emisor en su lista de vecinos, se añade dicho identificador recibo a su lista de vecinos. Una vez trascurridos los 8 segundos, el estado es cambiado por el *talker* provocando la salida del bucle. Al salir de éste los esclavos que no tengan registrado al maestro como vecino marcarán que se encuentran fuera de rango en ambos hilos. Por otra parte, el maestro enviará a su *talker* la lista de vecinos previamente creada. La tabla 3 muestra un ejemplo de una lista de vecinos.

Tabla 3 - Ejemplo de una lista de Vecinos.

UAV	Lista de Vecinos
0	[1]
1	[0,2]
2	[1,3]
3	[2,4]
4	[3]

Previamente se mencionó el uso de un nuevo estado llamado *hoopUpdate;* este estado es un concepto general, ya que, en realidad, se va a trabajar con 2 estados, uno dedicado al envío y el otro a la recepción de reconocimientos. Para no ser redundante al mencionar ambos estados, se ha decidido nombrar al conjunto como la fase *hoopUpdate*. Esta fase busca calcular el número de saltos hacia el maestro y, además, obtener la lista de enrutamiento pertinente de cada nodo, tal y como se presentó en las ideas conceptuales. El desglose de esta sección es más extenso, pero siguen estando implicados tanto el *talker* como el *listener*:

- Es necesario remarcar que los métodos *hoopUpdate* de ambos hilos son los encargados de llevar a cabo todas las operaciones mediante el uso de los estados *Waiting ACK Center y Sending ACK Center,* aunque el estado posterior, WayPoint Reached también interviene. Estos estados han sido implementados para gestionar el envío y recepción de mensajes entre los esclavos extremos y el maestro. Tras salir de *Neighbor Discovery* los nodos esclavos se encuentran en el estado *Waiting ACK Center*; una vez estos reciben un mensaje de un nodo que se encuentra en *Sending ACK Center* cambian su estado a éste. Cabe destacar que el maestro será siempre quien empiece en el *Sending ACK Center*. Tras realizar las operaciones necesarias para actualizar la lista de enrutamiento y los saltos, el estado cambiará a *Following Mission*.

- El *talker* al igual que en el estado anterior, hace uso de un temporizador donde posteriormente ayudará a dormir el hilo. Por otra parte, el bucle de método estará enviando mensajes siempre y cuando el valor del estado sea inferior al de *Following Mission*; aquí se rompe dicha condición a través del *listener*. En un

primer lugar, el buffer se limpia para evitar problemas de funcionamiento y seguidamente, se envía un valor determinado para indicar el tipo de mensaje, donde este es del tipo *Sending ACK Center*. A continuación, se envía la cantidad de saltos que hay hacía el maestro, donde todos parten con la idea de que ellos son el maestro y tienen un valor 0 de salto, hasta que desde el *listener* se actualiza dicho valor en cada pasada de su bucle. Además, se envía el identificador del nodo junto con la lista de enrutamiento donde, en un inicio, los esclavos no tienen nada almacenado pero el maestro por otra parte contiene a sus vecinos, y a medida que avanza la ejecución esta lista se va actualizando. Una vez el buffer ha sido cargado con todos los datos, se aplica un *flush* sobre él para, seguidamente, enviar el mensaje por broadcast y ajustar los temporizadores que duermen el hilo. La tabla 4 contiene el seudocódigo empleado en el desarrollo del *talker*.

Tabla 4 - Seudocódigo Talker hoopUpdate

```
Instanciar cicleTime
WHILE estado < Following Mission
  Reinicio del Buffer
  Escritura Tipo de Mensaje, Salto e Identificador
  FOR uav_ID en Lista Enrutamiento -> Escritura ID
  Flush al Buffer
  Instanciar mensaje y enviar por Broadcast
  Ajustar tiempos
  IF tiempo de espera > 0
    Dormir hilo
FIN
```

La ilustración 15 expone el comportamiento del *talker* al enviar los mensajes de tipo *SEND_ACK_CENTER*, en una prueba real.

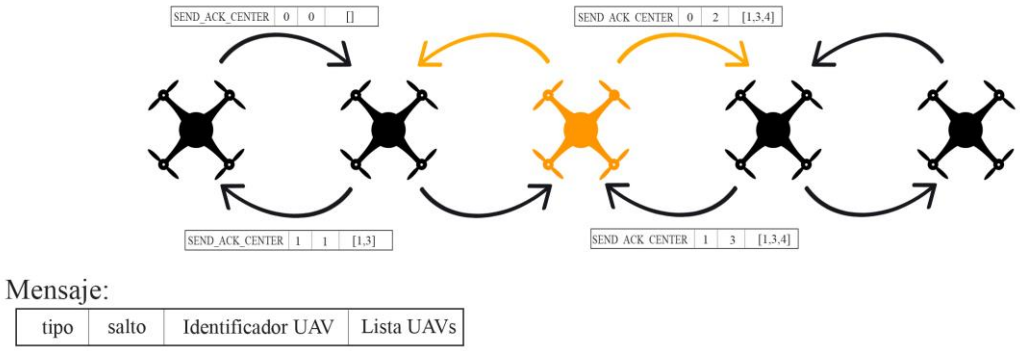

Ilustración 15 - Envío de saltos y lista de enrutamiento.

- Dentro del *listener* se busca mantener la estabilidad de los mensajes y, es por eso por lo que se va a emplear otra vez la estructura general de fiabilidad. Además, se va a hacer uso de una nueva lista, *checkNeighbors*, que se va a encargar de comprobar que todos los vecinos registrados han vuelto a comunicarse de forma estable. Para acceder a la estructura de fiabilidad

primero se debe acceder al bucle principal, el cual se mantendrá en ejecución siempre y cuando el estado actual del nodo sea inferior a *Following Mission*. Durante cada iteración se lee el mensaje recibido, si este es del tipo *Sending ACK Center* o Waypoint Reached ACK se procede a tratar los datos de forma fiable. Al igual que en *Neighbor Discovery*, para poder tratar los datos se debe haber recibido 3 mensajes consecutivos y en este caso, a diferencia de este, el emisor del mensaje debe ser un vecino. Dependiendo el estado del nodo se opera de una forma distinta, en estos cuatros puntos se repasa todo el trabajo abordado y, además, en las tablas 5, 6, 7, se muestran los seudocódigos de cada caso:

1. En el estado *Waiting ACK Center,* en una primera instancia, se recoge e incrementa el salto recibido para poder asignarlo al nodo y comunicar este nuevo valor al *talker;* seguidamente se añade el vecino a *checkNeighbors* en caso de que este no pertenezca aún a la lista. A continuación, en caso de que el buffer aún contenga datos, se tratan estos como los nodos almacenados en la lista de enrutamiento, *Listing Path*, del emisor. En este caso, se va leyendo y almacenando valores en la lista de enrutamiento del nodo hasta que el buffer esté vacío. Tras esto, se añaden los vecinos propios a la lista de enrutamiento para posteriormente enviarla al *talker*. Una vez llegados a este punto se limpian los mapas *LastMSHUAV* y *NeigMSG* para poder mantener la fiabilidad en el siguiente estado *Sending ACK Center*, el cual es accedido tras completar esta operación. Es importante remarcar que este es el estado inicial de todos los esclavos al iniciar hoopUpdate, y que, a medida que se reciba un mensaje del maestro (o de los retransmisores), se irá cambiando el estado.

Tabla 5 - Caso Waiting ACK Center

```
Actualizar el salto
IF checkNeighbors NO contiene al emisor
  Se añade a la lista
WHILE hay datos en buffer
  IF ListPath NO contiene el valor -> Añade en ListPath
FOR dron en Neighbors -> Añade en ListPath
Envío al talker la nueva lista de enrutamiento
Limpia las listas LastMSHUAV y NeigMSG
Cambio de estado a Sending ACK Center
```

2. Durante el estado *Sending ACK Center* se gestiona la lista *checkNeighbors* para poder cambiar de estado; si el identificador del vecino recibido aún no se encuentra dentro de esta lista, se procede con su registro. Tras esto, se comprueba que tanto la lista *checkNeighbors* como la lista de los vecinos son iguales. Si la condición se cumple significa que los vecinos del nodo están funcionando correctamente, actualizando sus saltos junto con la creación de sus listas, es por eso por lo que se procede a limpiar los mapas *LastMSHUAV* y *NeigMSG* y a cambiar el estado. Si el nodo (independientemente de si es un esclavo o el maestro) no es el más alejado, cambia su estado a *Waiting ACK Last*;

en caso contrario, cambia su estado a *Sending ACK Last*. Para finalizar, se limpia la lista *checkNeighbors* para poder tratar correctamente los siguientes estados.

Tabla 6 - Caso Sending ACK Center

```
IF checkNeighbors NO contiene al emisor
  Se añade a la lista
IF checkNeighbors == Neighbors
  Limpia las listas LastMSHUAV y NeigMSG
  Cambio de estado a Following Mission
  Limpia la lista checkNeighbors
```

3. Los dos estados presentados son complementados con la intervención de la fase *WayPoint Reached* del estado *Following Mission*, concretamente, con los mensajes generados por dicha fase del tipo *Waypoint Reached ACK*. Esto se debe a que tras finalizar el estado *Sending ACK Center* en un nodo no implica que sus adyacentes hayan finalizado también. De esta forma, si el mensaje recibido por un nodo es del tipo *Waypoint Reached* almacenará dicho identificador en *checkNeighbors* (se descartarán todos los otros valores del mensaje ya que no deben ser tratados en esta sección) donde en caso de que esta lista sea igual a la de sus vecinos provocará un cambio de estado; Siendo este cambio de *Waiting ACK Center* hacía *Sending ACK Center* o desde este último hacía *Following Mission*. Este comportamiento se da por válido ya que si un nodo vecino ha conseguido alcanzar el estado *Following Mission* presenta un escenario donde los valores recibidos previamente por parte de éste han sido manipulados correctamente, por tanto, podemos mantener nuestros valores actuales. Siendo más concisos, este fragmento de código sirve para verificar la integridad de los vecinos y que dichos están operando como es debido validando así las operaciones del nodo en cuestión. La tabla 7 muestra el fragmento de seudocódigo empleado.

Tabla 7 - Caso Waypoint Reached ACK

```
IF checkNeighbors NO contiene al emisor
  Se añade a la lista
IF checkNeighbors == Neighbors
  Limpia las listas LastMSHUAV y NeigMSG
  IF Estado es igual a Sending ACK Center
    Cambio de estado a Following Mission
  IF Estado es igual a Waiting ACK Center
   Cambio de estado a Sending ACK Center
```

En conclusión, el método *hoopUpdate* es un concepto general que hace uso de 2 nuevos estados. Los dos estados sirven tanto para el envío y como para la recepción de reconocimientos en base al centro para actualizar los saltos de cada nodo junto con la lista de enrutamiento. Además, durante la ejecución de estos el siguiente estado

Following Mission se encarga de aportar integridad y válidez a las operaciones realizadas por los vecinos, ayudando así al cambio de estados del propio nodo. En la tabla 8 se representa el resultado esperado en una ejecución con 7 drones implicados.

Tabla 8 - Enrutamiento y Saltos.

Nodo	Saltos	Enrutamiento
0	3	[2,4,1,0]
1	2	[2,4,1,0]
2	1	[2,4,1]
3	0	[2,4]
4	1	[2,4,5]
5	2	[2,4,5,6]
6	3	[2,4,5,6]

Following Mission es el encargado de ejecutar las operaciones que permiten completar la misión establecida, es decir, se hace cargo de ejecutar correctamente las fases de *Waypoint Reached* y *Move to Waypoint* (las cuales van a permitir la comunicación entre estos nodos). Mediante estas fases, como se menciona previamente, es posible completar la misión establecida, pero para ello el maestro debe comunicarse con los esclavos, y éste debe recibir los reconocimientos de ellos. Cada nodo almacena en el mapa *lastTimeUAV* el identificador del emisor junto el tiempo de lectura del mensaje. Este mapa se encarga de gestionar el enjambre dentro de la función *updateSwarm*, la cual se ejecuta durante el estado *WayPoint Reached* en cada iteración. En dicha función se comprueba, por cada nodo registrado, que la diferencia entre el instante actual y el instante almacenado no supera el tiempo de vida establecido; en caso de superar dicho valor, el nodo considera como caído a todo que rompa dicha condición. Ambas fases no tienen ningún tipo de mecanismo para suplir el problema de rango, y es por ello por lo que los cambios introducidos en ambas fases son:

- El *talker* ha sido modificado para que, en la fase *Waypoint Reached,* sea capaz de retransmitir la lista de los reconocimientos acumulados, *accumulativeACKList*, junto con la identificación que justifica si está fuera de rango o no del nodo emisor. Los cambios sobre *Waypoint Reached* del *listener* han afectado al comportamiento inicial de éste. Al empezar la ejecución se notifica al *talker* que no se busca retransmitir y, además, se hace un borrado de los reconocimientos acumulados. Dentro del bucle existen 3 casos distintos:

  1. Los mensajes de tipo LAND anteriormente eran enviados únicamente por el maestro, y es por eso por lo que, al necesitar retransmisión, se ha modificado el comportamiento. Ahora, además de establecer el punto de aterrizaje en el nodo, se da permisos a éste para retransmitir dichas coordenadas hacía los vecinos. De esta forma el método *land* del *talker,* con

las modificaciones vigentes, permite a los nodos retransmisores enviar las coordenadas de aterrizaje extraídas del maestro para así ser tratadas en el *listener*.

2. Por otra parte, los mensajes del tipo MOVE_TO_WAYPOINT tienen varias operaciones a realizar. Primero se añade tanto el identificador propio como el recibido por parte del emisor a la lista *accumulativeACKList* donde seguidamente, en caso de tener más valores en el buffer, se procede a registrar los nodos recibidos en la lista mencionada. Tras actualizar la lista de reconocimientos se notifica al *talker*. Cabe destacar que en todo momento se ha actualizado la lista *lastTimeUAV* con sus identificadores y tiempos pertinentes. A continuación, se lee el punto de ruta recibido donde, si el esclavo tiene un valor inferior al recibido, procede a incrementar el suyo, y notificar al *talker* que debe retransmitir.

3. Por último, los mensajes del tipo WAYPOINT_REACHED_ACK son tratados de distinta manera, ya que los esclavos actualizan la lista de reconocimientos añadiéndose a sí mismos junto con los identificadores recibidos; tras esto notifican al *talker* de los cambios. Es importante remarcar, que, durante el almacenamiento de los reconocimientos, se van actualizando los identificadores y tiempos del mapa *lastTimeUAV*. Por otra parte, el maestro se encarga de leer el identificador del emisor junto con la lista de reconocimientos acumulados recibida, actualizando de esta forma el mapa *lastTimeUAV*.

- *Move to waypoint* ha sido modificado para adaptarse a los cambios producidos en la fase anterior. Dentro del *talker* aquellos nodos que retransmiten o que son el maestro se encargan de enviar el punto de ruta actual, junto con la lista de reconocimientos. Por otro lado, los esclavos siguen enviando su punto anterior salvo que esta vez también envían la lista de reconocimientos, y notifican si están fuera de rango. Dentro del *listener* se han realizado las modificaciones pertinentes para poder leer los nuevos valores y actualizar de esta forma el mapa de tiempos *lastTimeUAV*. La ilustración 16 muestra las estructuras empleadas en cada mensaje retransmitido en estas fases.

MOVE TO WAYPOINT: Maestro y escalvos retransmisores

tipo	Identificador	Punto de ruta	Reconocimientos

WAYPOINT REACHED: Esclavos

tipo	Identificador	Punto de ruta	Fuera de rango	Reconocimientos

LAND: Maestro y Esclavo retransmisor.

tipo	Punto X	Punto Y

Ilustración 16 - Estructura de los mensajes en WP Reached y Move to WP.

Si no se hubiera empleado la lista de reconocimientos acumulados (*accumulativeACKList*), todos aquellos drones que están fuera del rango de comunicaciones del maestro serían eliminados. Si un dron falla al desplazarse entre puntos de ruta, éste es eliminado del enjambre tras llegar al punto destino. Con todos los pasos completados se ha observado que la máquina de estados planteada previamente queda desactualizada, por lo que en la ilustración 17 se presenta la máquina de estados actualizada.

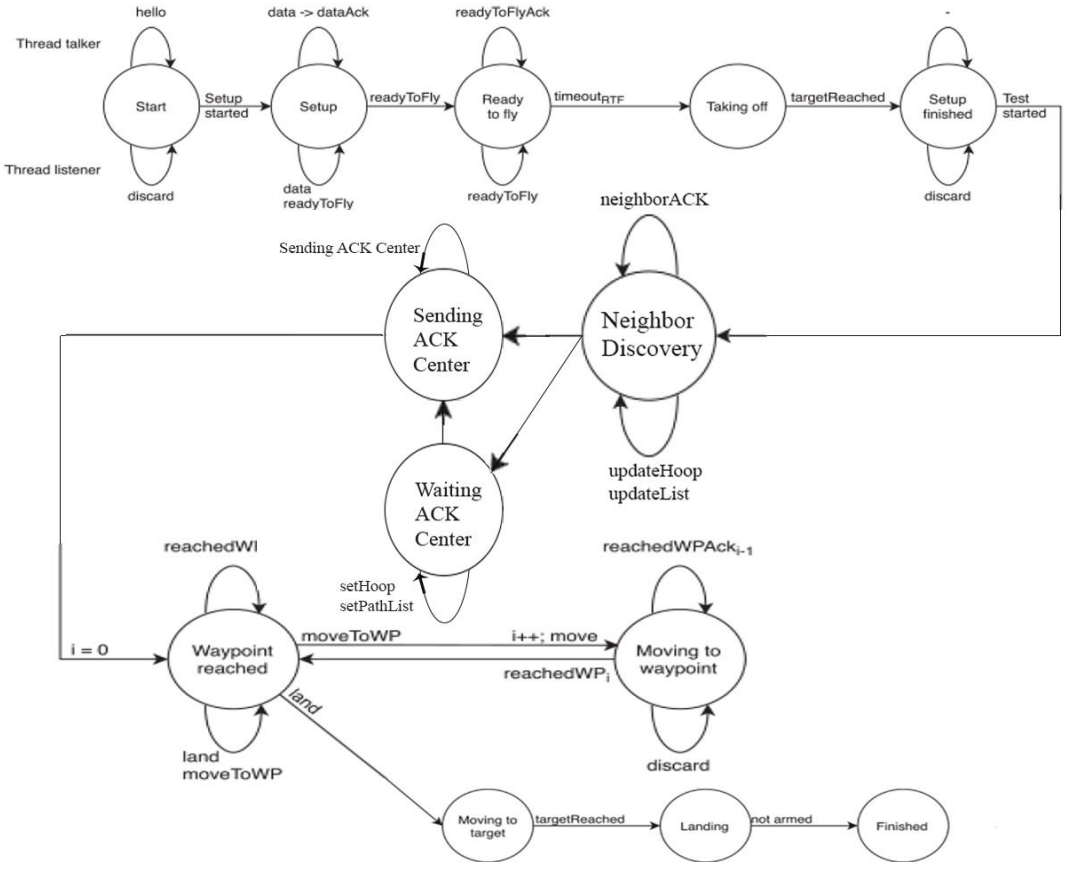

Ilustración 17 - Máquina de estados adaptada: versión final.

5. Experimentación

El capítulo 5 contiene los datos de todos los ensayos realizados de manera práctica sobre el simulador haciendo uso de la nueva versión multi-salto de MUSCOP mostrada en el capítulo anterior. Los datos han sido tratados de forma distinta según cada experimento. A rasgos generales, la gran mayoría comparten las mismas características, salvo que algunas varían respecto a la cantidad de drones utilizados, la distancia de separación en pleno vuelo, y la formación utilizada. La ilustración 18 contiene los datos de configuración generales que comparten todas las pruebas.

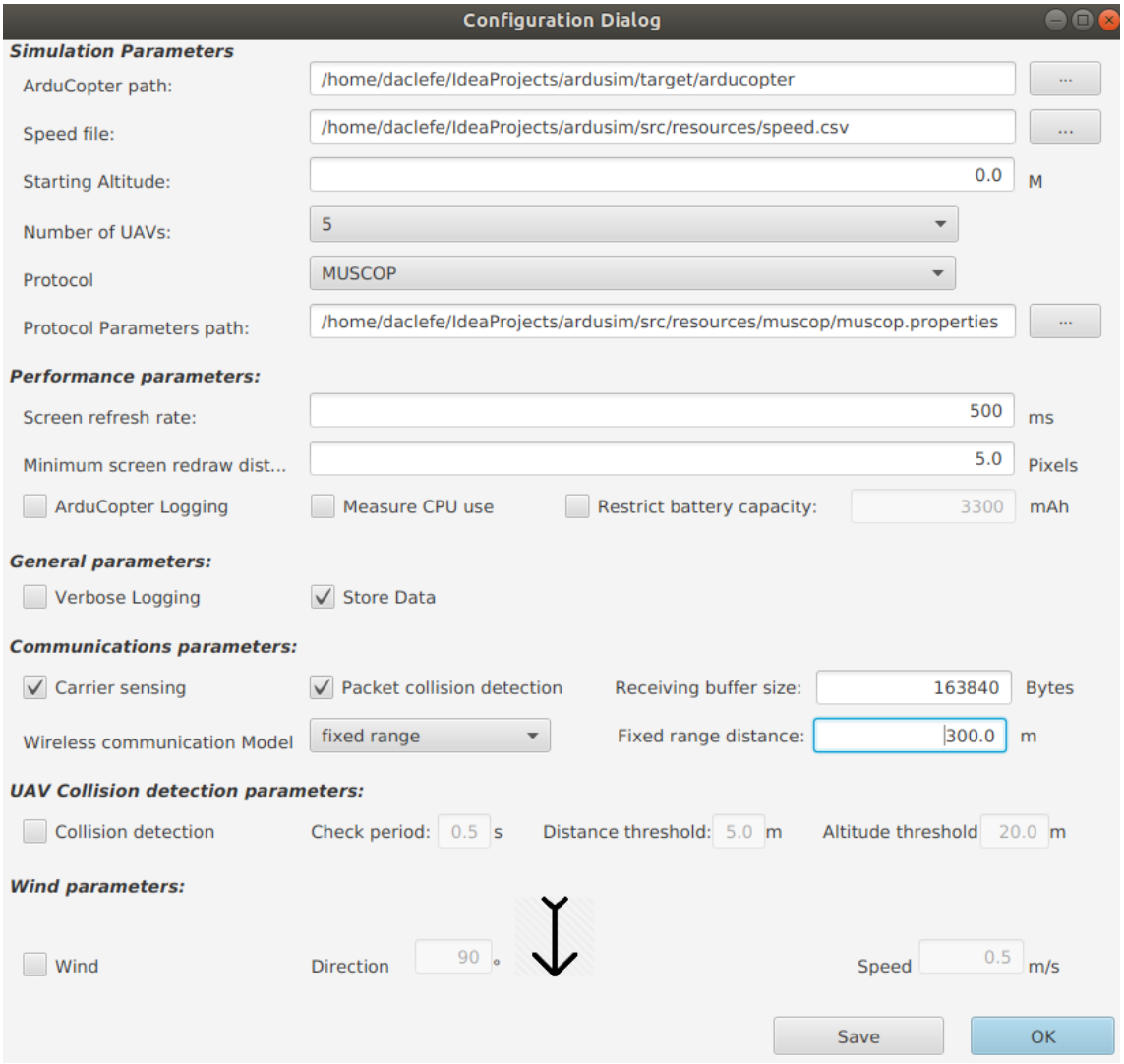

Ilustración 18 - Configuración general de las pruebas.

Para validar una gran cantidad de datos, primero se ha decidido abordar pruebas menos complejas para corroborar que el funcionamiento del programa es el esperado y, además, obtener datos que ayuden a observar el tiempo de vuelo en cada dron junto

con los mensajes recibidos. Estos datos han sido capturados desde el inicio del estado *Neighbor Discovery* hasta la finalización del estado *Landing*. Una vez llegada la ejecución al estado *Finished* se deja de contar cualquier valor, ya que este último estado implica que los drones han aterrizado satisfactoriamente.

La primera prueba consta de diversas ejecuciones en formación linear con un total de 5 drones separados 160 metros en el aire tal y como se muestra en la Ilustración 19.

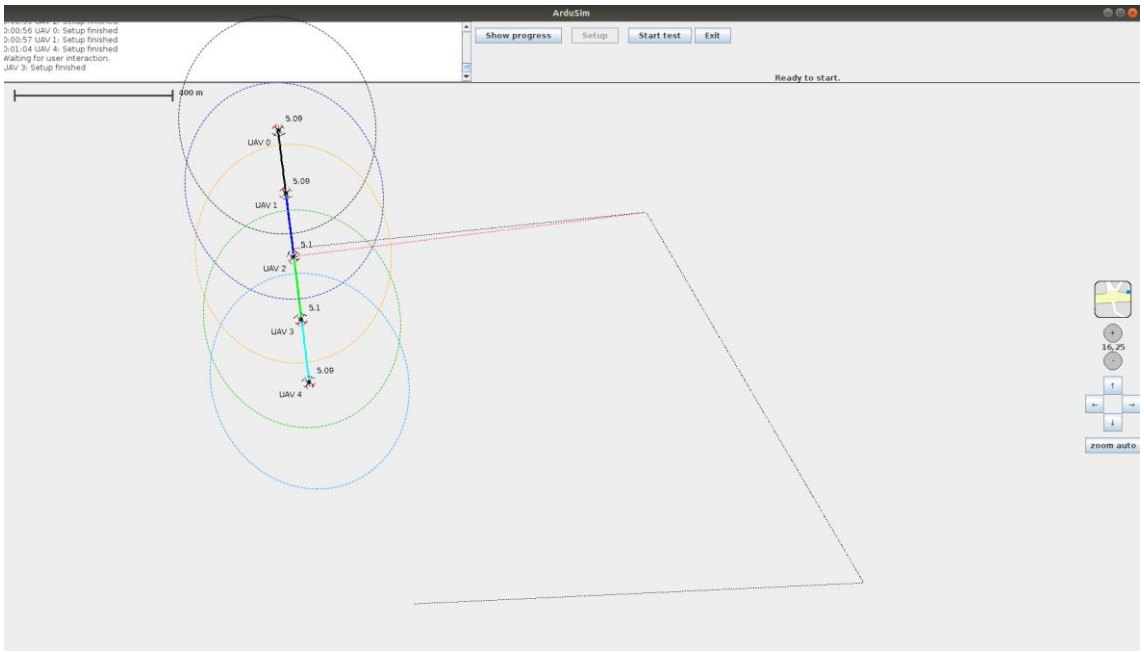

Ilustración 19 - Escenario Práctico de la prueba 1.

Las ilustraciones 20 y 21 representan todos los datos obtenidos en la primera prueba, donde se puede afirmar que, a mayor distancia del centro, menos vecinos y, a menor número de vecinos, menor cantidad de mensajes recibida. De forma inversa, a mayor distancia del maestro, mayor será el tiempo de vuelo, ya que aquellos más alejados deben acercarse a una posición relativa del centro para el proceso de aterrizaje.

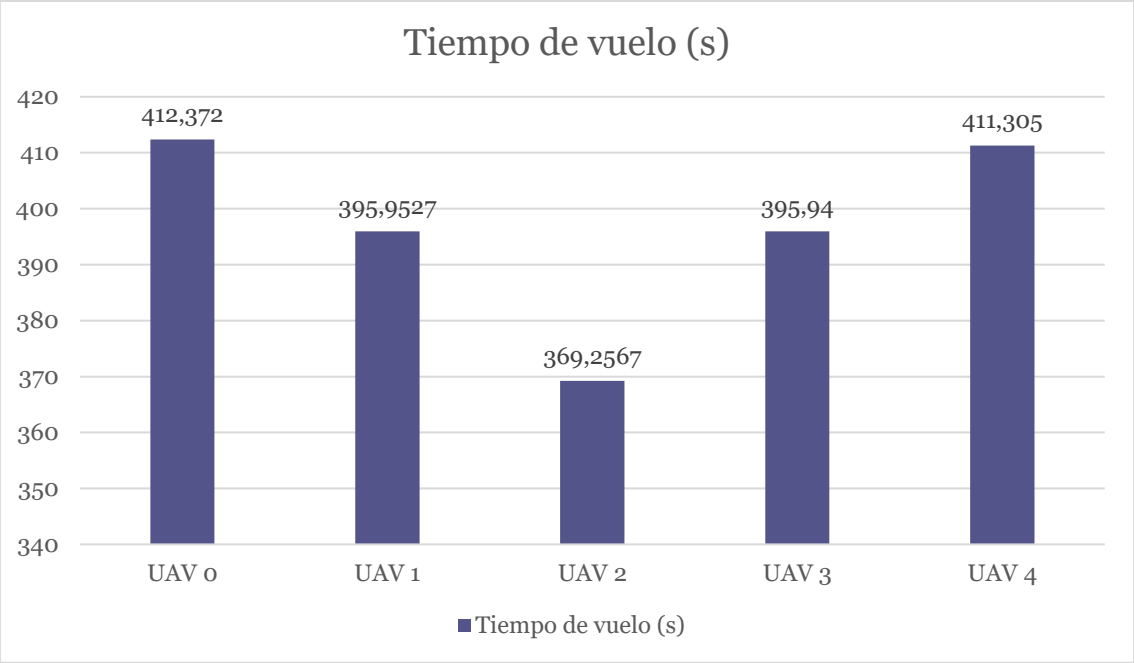

Ilustración 20 - Tiempo de vuelo por cada dron en la prueba 1.

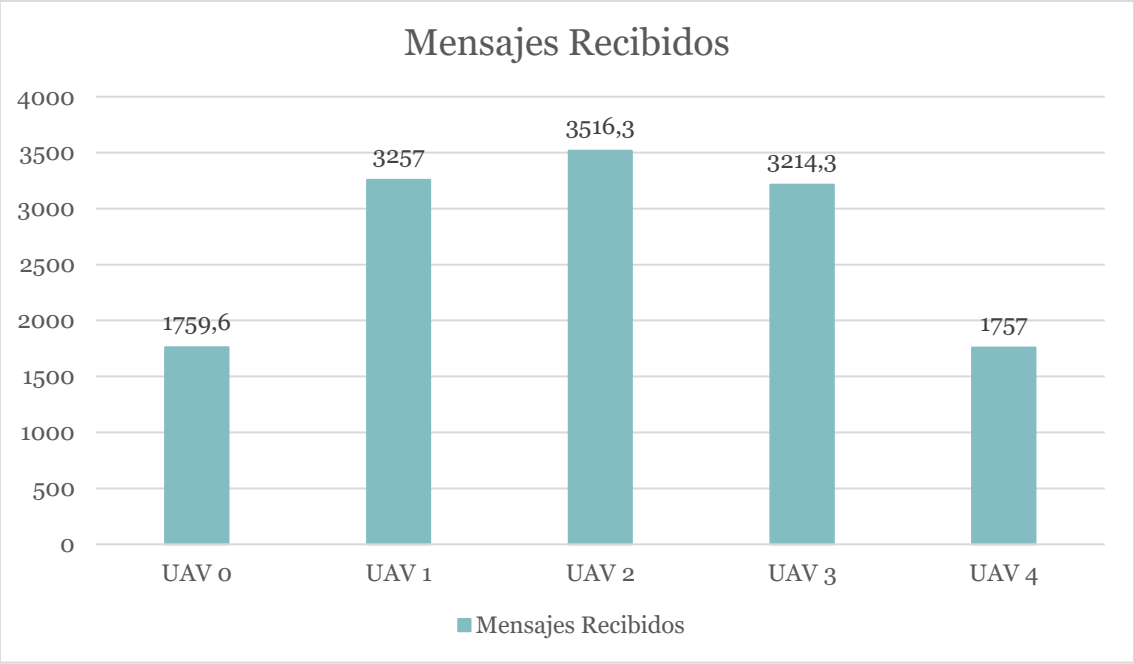

Ilustración 21 - Mensajes recibidos por cada dron en la prueba 1.

La tabla 9 representa así un número medio de mensajes cercano a las 2701 unidades, junto con un tiempo medio de unos 7 minutos aproximados. Esto significa que cada dron está recibiendo entre 6 y 7 mensajes por segundo.

Tabla 9 - Medias, mínimos y Máximos de la prueba 1.

Tiempo de vuelo Menor (s)	395,95
Tiempo de vuelo Mayor (s)	412,372
Media de Tiempo en vuelo (s)	396,96
Media de Mensajes Recibidos	2700,80

La segunda prueba de ejecución comparte casi las mismas características que la primera prueba, salvo que, en ésta, se ha decidió aumentar la complejidad del enjambre, siendo un total de 15 drones empleados, tal y como se representa en la ilustración 22.

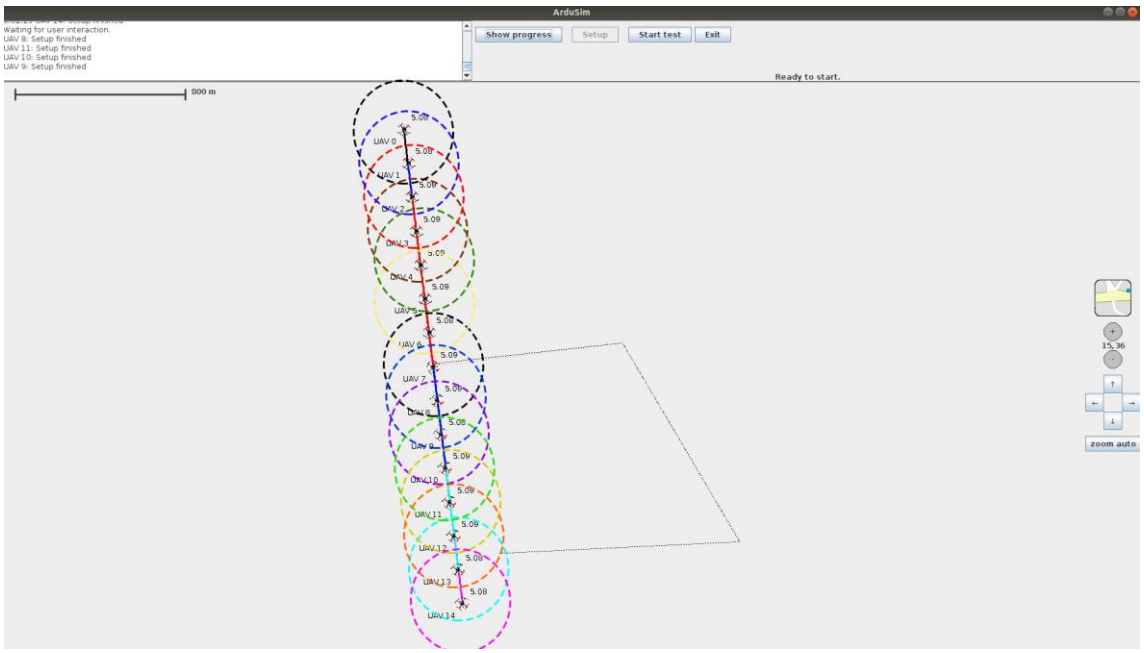

Ilustración 22 - Escenario práctico de la prueba 2.

Las ilustraciones 23 y 24 nos permiten observar los datos obtenidos en cada dron; de esta forma en la tabla 10 los datos medios pueden ser representados junto a sus máximos y mínimos. Es importante remarcar que, en este nuevo escenario, los drones externos siguen siendo aquellos con menor carga de mensajes; sin embargo, todo dron distinto a cualquiera de estos tiene una carga similar, a diferencia de la anterior prueba. Por otra parte, se puede observar que los tiempos siguen teniendo el mismo peso en ambas pruebas, es decir, aquellos más alejados deberán ejecutar una breve fracción de tiempo para poder suplir la distancia que han de recorrer.

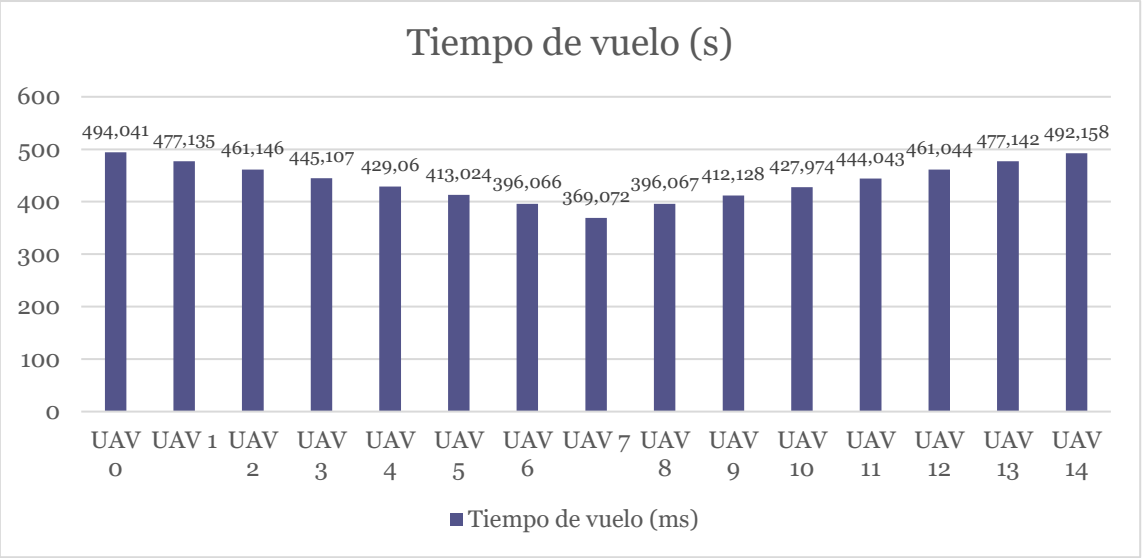

Ilustración 23 - Tiempo de vuelo por cada dron en la prueba 2.

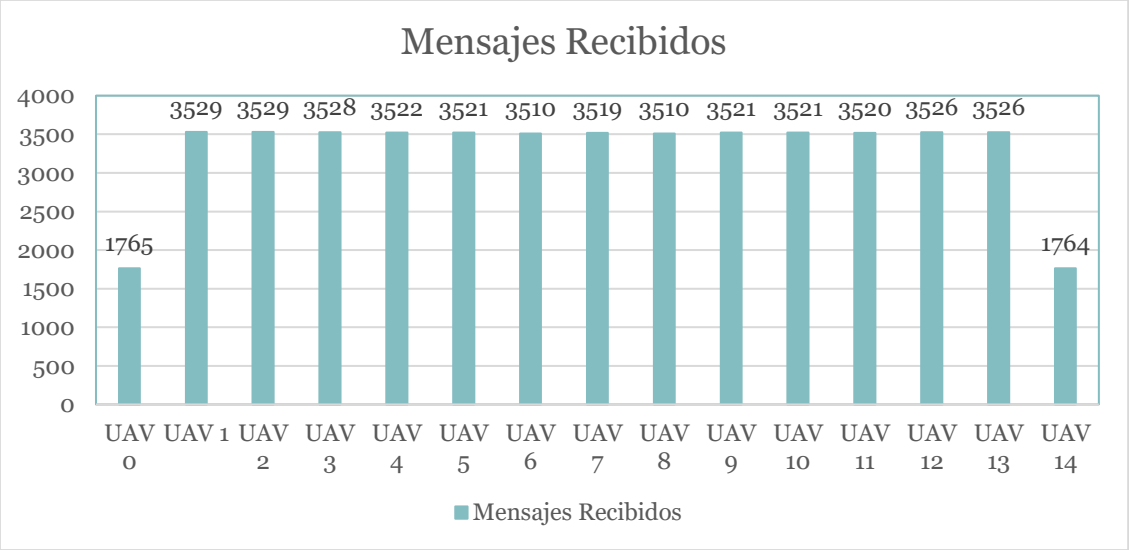

Ilustración 24 - Mensajes recibidos por cada dron en la prueba 2.

A diferencia de la primera prueba, en esta segunda se observa (ver tabla 10) que tanto el tiempo medio como los mensajes recibidos tiene un valor superior, lo cual es lógico al emplear un mayor número de drones, pero es remarcable que estas diferencias entre ambas no son muy elevadas.

Tabla 10 - Medias, mínimos y máximos de la prueba 2.

Menor tiempo de vuelo (s)	369,07
Mayor tiempo de vuelo (s)	494,04
Tiempo medio de vuelo (s)	439,68
Media de mensajes recibidos	3287,40

La tercera prueba dista un poco de las anteriores, aunque sigue actuando sobre la formación linear. En pruebas anteriores se usaba una separación entre drones elevada para que se observara el comportamiento de los retransmisores cuando oscilaban entre 1 o 2 vecinos. En este caso se ha decidido reducir la separación de estos a unos 140m para que así todos los nodos tengan el doble de carga. El enjambre resultante se compone de un total de 9 drones, mostrándose dicha ejecución en la Ilustración 25.

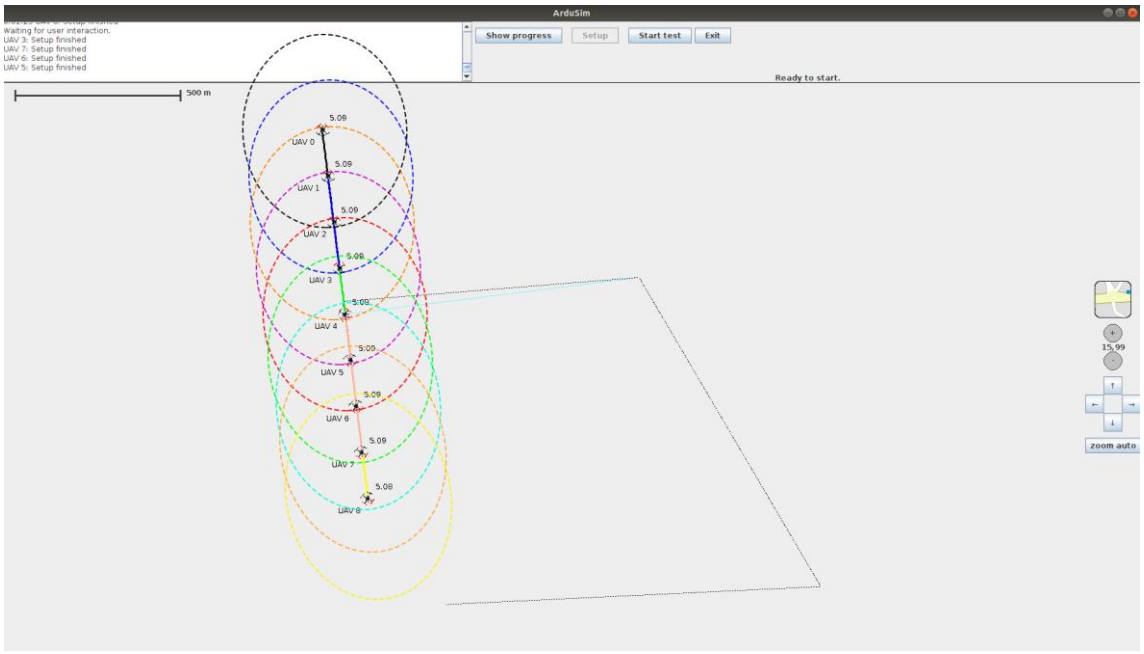

Ilustración 25 - Escenario práctico de la prueba 3.

Observando los gráficos de las ilustraciones 26 y 27 se puede ver una amplia variedad entre los tiempos totales y los mensajes recibidos. Al igual que en pruebas anteriores, el tiempo de cada dron sigue respetando el patrón, donde a mayor distancia más tiempo de ejecución. Por otra parte, a pesar de tener un menor número de drones, en comparación con la prueba anterior, es observable que hay un gran incremento en el envío de mensajes. En algunos nodos se llega a cumplir la recepción del doble de mensajes, aunque en otros varía, y todo esto se debe a la gestión de fiabilidad y al tráfico pertinente. A pesar de tener una mayor carga, los drones extremos siguen siendo los menos saturados. Los datos técnicos medios más interesantes se encuentran en la tabla 11.

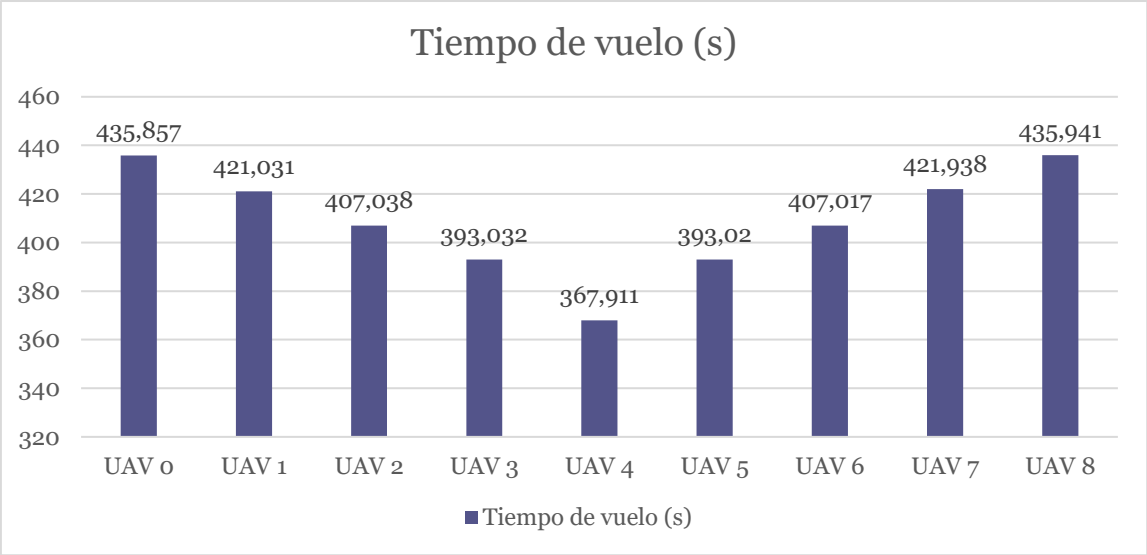

Ilustración 26 - Tiempo de vuelo por cada dron en la prueba 3.

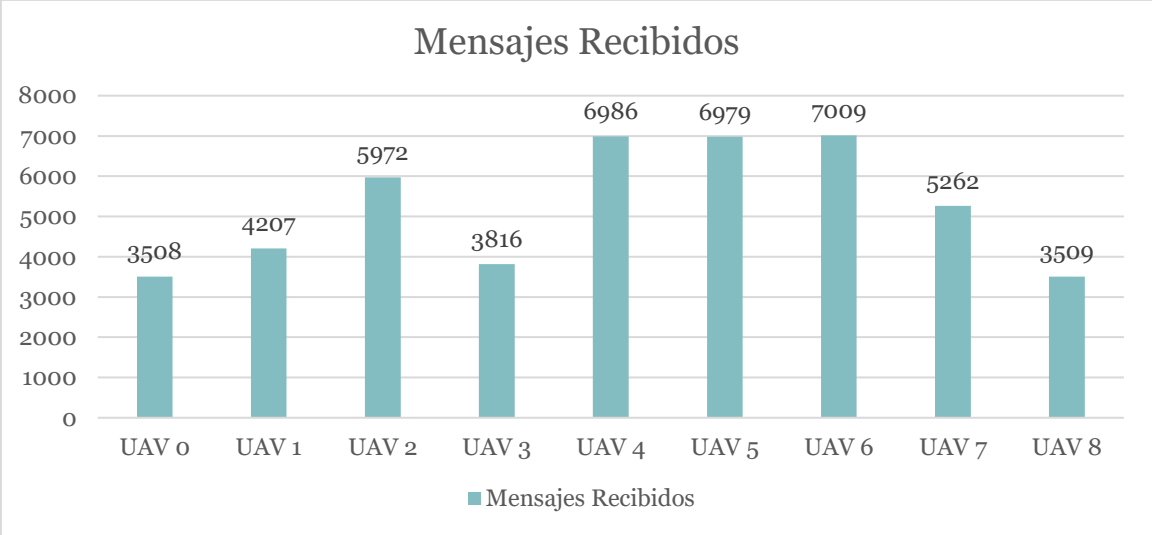

Ilustración 27 - Mensajes recibidos por cada dron en la prueba 3.

Al comparar la tabla 11 con los anteriores resultados se puede observar que hay un incremento considerable en los mensajes recibidos al duplicar los vecinos de cada nodo, aunque, no obstante, se puede observar que los tiempos no varían mucho y son bastante similares, tal y como se ha mencionado previamente.

Tabla 11 - Medias, mínimos y máximos de la prueba 3.

Menor tiempo de vuelo (s)	367,91
Mayor tiempo de vuelo (s)	435,94
Tiempo medio de vuelo (s)	409,19
Media de mensajes recibidos	5249,80

Todas las pruebas presentadas hasta ahora han estado relacionadas con la formación linear; no obstante, también se ha realizado pruebas sobre la formación matriz, en este caso la regular. En una primera instancia, la extensión realizada de MUSCOP es completamente funcional sobre esta formación. A pesar de esto, se ha decido ver el comportamiento original del protocolo aplicando los nuevos patrones, es por ello por lo que en esta prueba se ha decidido omitir el funcionamiento del *relaying* y se ha buscado observar el comportamiento que tiene una red cuando todos son capaces de descubrirse entre sí. Sobre los 300 metros de rango se ha utilizado un enjambre de 7 drones separados unos 100 metros entre sí, calcando de esta forma el escenario plasmado en la ilustración 28.

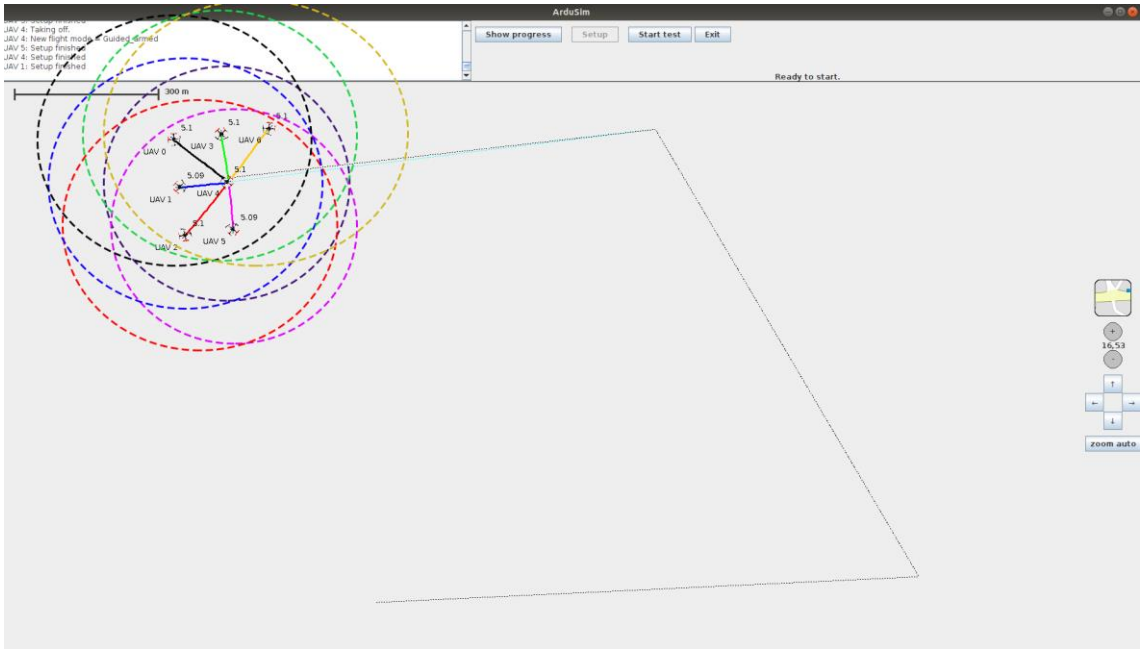

Ilustración 28 - Escenario práctico de la prueba 4.

En las Ilustraciones 29 y 30 se pueden observar los valores obtenidos en esta nueva prueba donde, a diferencia de las otras, los nodos son capaces de comunicarse entre sí dando lugar a una alta carga en la red ya que, con unos pocos drones, se presenta un intercambio de mensajes bastante alto. Sin embargo, los tiempos de vuelo no varían mucho, a excepción del nodo maestro. Aun así, se puede corroborar el buen funcionamiento de la extensión a pesar de no necesitar el *relaying*.

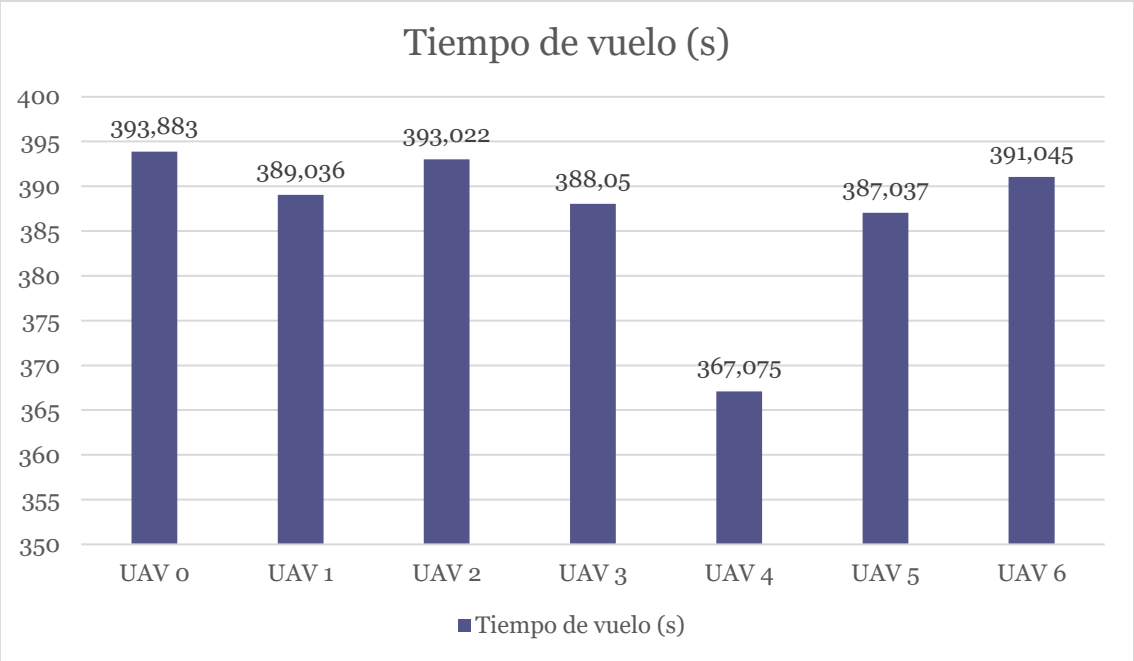

Ilustración 29 - Tiempo de vuelo por cada dron en la prueba 4.

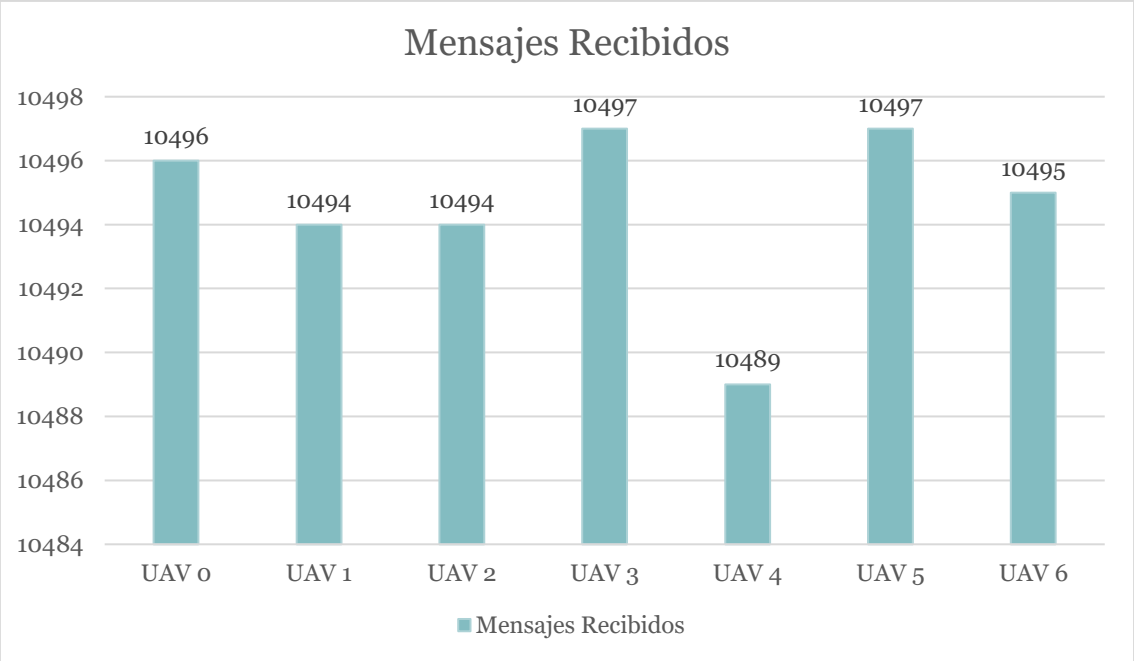

Ilustración 30 - Mensajes recibidos por cada dron en la prueba 4.

La tabla 12, a diferencia de los otros experimentos, presenta unos datos más homogéneos ya que son todos muy similares y tan solo presentan minúsculas variaciones entre sí.

Tabla 12 - Medias, mínimos y Máximos de la prueba 4.

Menor tiempo de vuelo (s)	367,07
Mayor Tiempo de vuelo (s)	393,83
Tiempo medio de vuelo (s)	387,02
Media de mensajes recibidos	10494,60

Por último, como prueba final se ha utilizado una separación de 160m y una cantidad de 11 drones en el enjambre para observar el funcionamiento del *relaying* sobre esta nueva formación presentada en la ilustración 31.

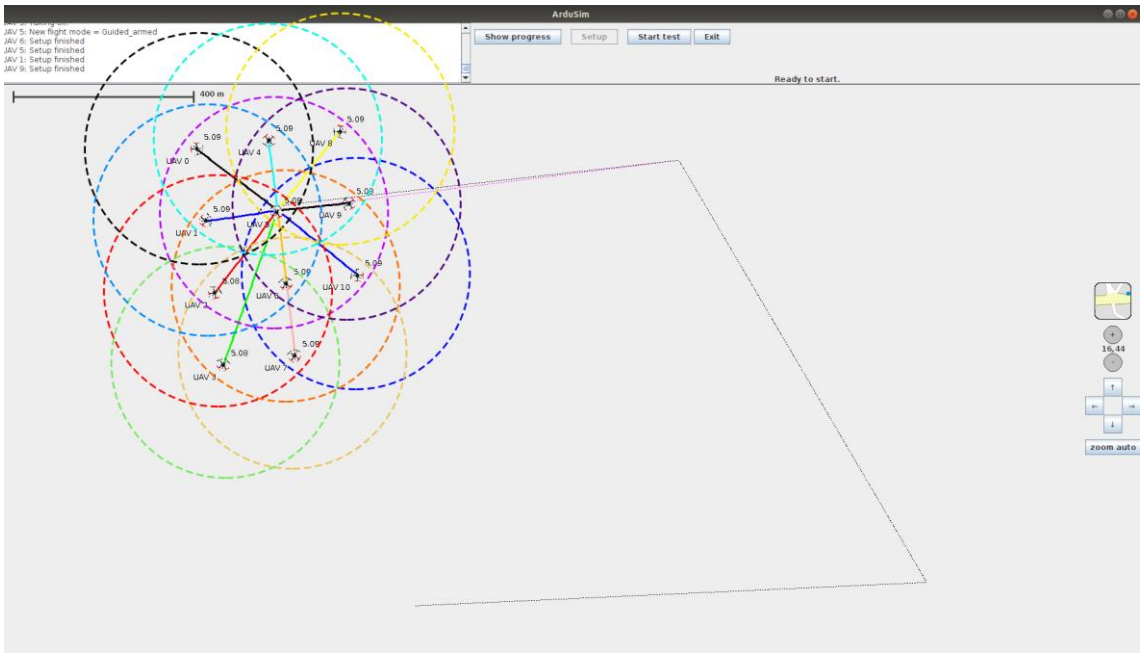

Ilustración 31 - Escenario práctico de la prueba 5.

Al observar las gráficas representadas en las ilustraciones 32 y 33 se observa una breve varianza en los mensajes recibidos ya que algunos nodos no han recibido una carga tan elevada. Sin embargo, los tiempos de ejecución no varían significativamente, pues hay pequeñas fluctuaciones en los tiempos, pero vienen a tener un patrón muy similar a las demás pruebas donde el maestro siempre es el que tiene un menor tiempo.

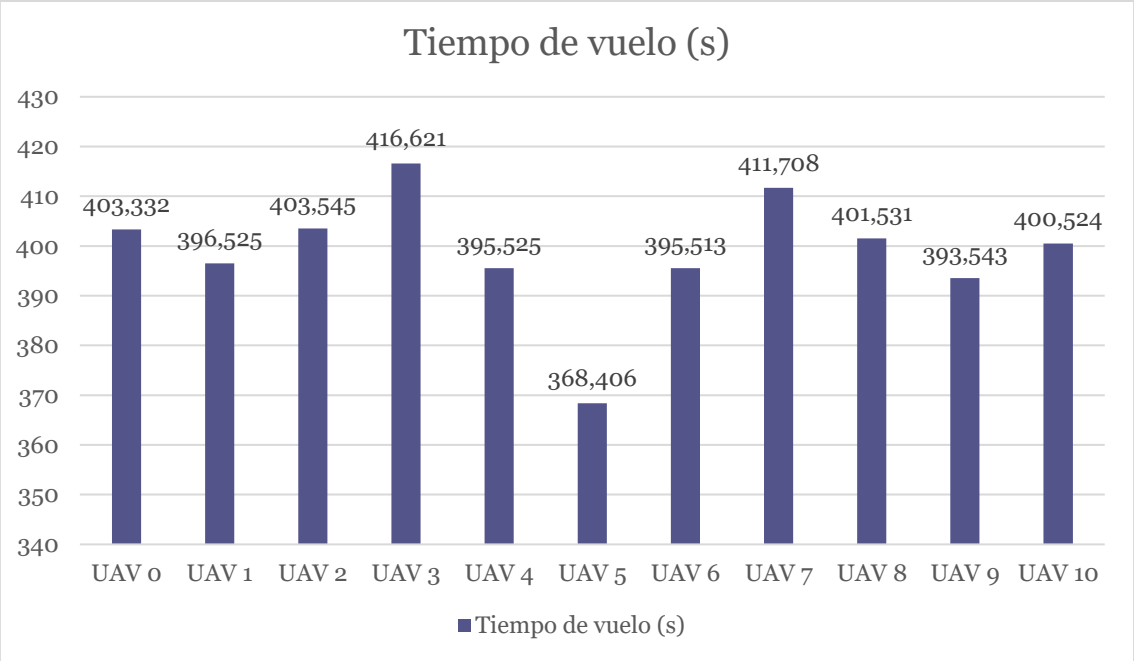

Ilustración 32 - Tiempo de vuelo por cada dron en la prueba 5.

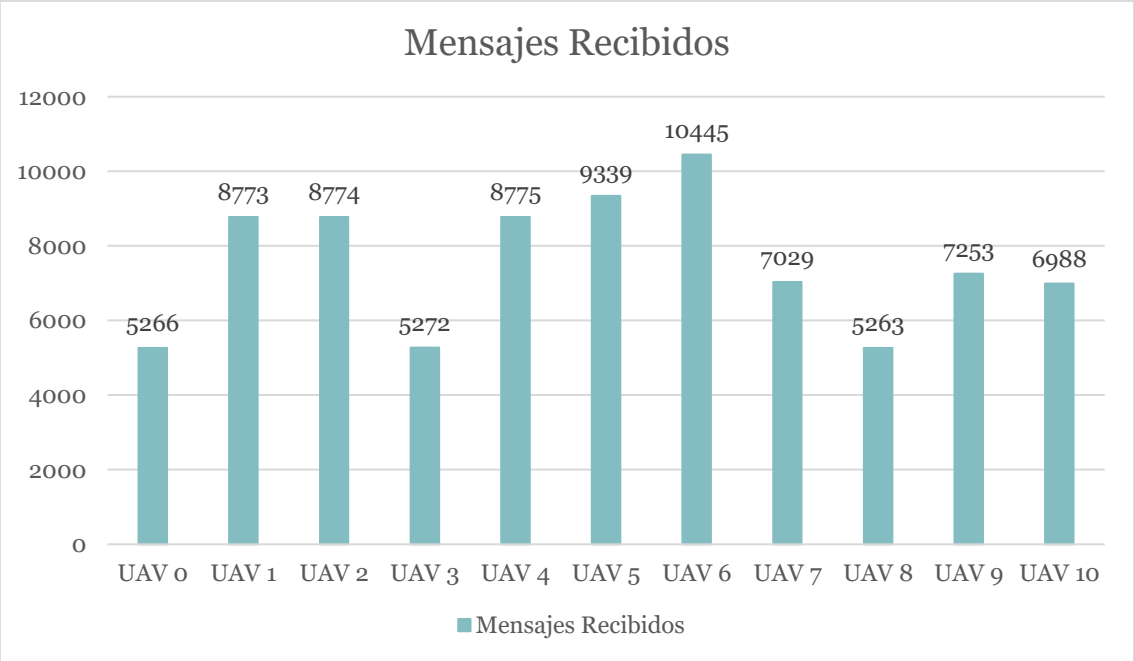

Ilustración 33 - Mensajes recibidos por cada dron en la prueba 5.

En la tabla 13 se observa un gran incremento en el número de mensajes recibidos ya que, en esta formación, el número de vecinos por cada nodo se incrementa drásticamente, aunque, por otra parte, los tiempos de vuelo siguen siendo muy similares. Toda la información extraída tiene los rasgos característicos de la tabla 11, aunque aquí sí se pueden apreciar variaciones más notables.

Tabla 13 - Medias, mínimos y máximos de la prueba 5.

Menor tiempo de vuelo (s)	368,40
Mayor tiempo de vuelo (s)	416,62
Tiempo medio de vuelo (s)	398,79
Media de mensajes recibidos (s)	7561,50

Para concluir con el muestreo de tiempos de vuelo y mensajes recibido, en las ilustraciones 34 y 35 se muestran unas gráficas comparativas de los tiempos medios y los mensajes recibidos en cada una de las pruebas realizadas, teniendo así una fuente visual que resalte el contraste de éstas.

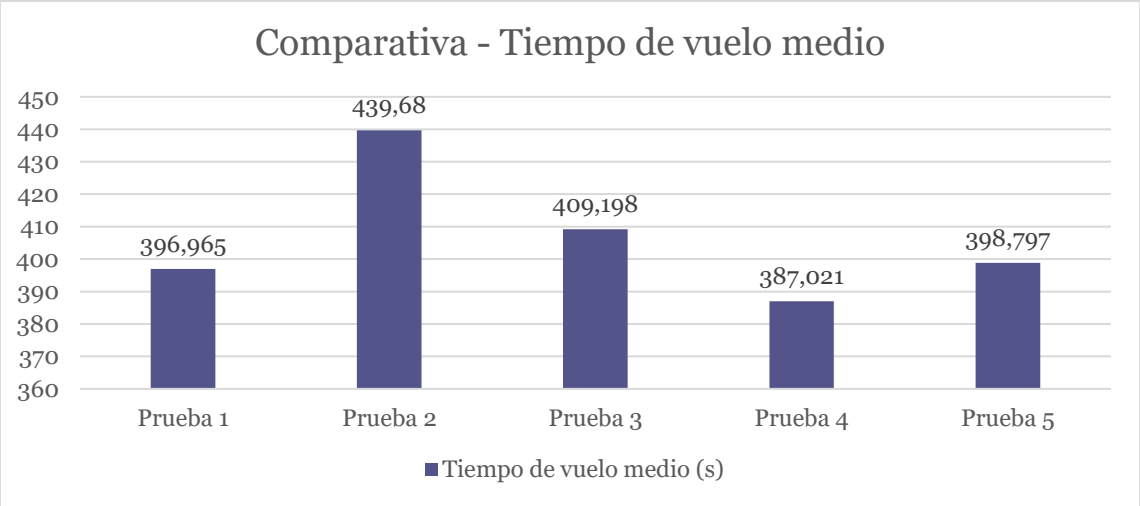

Ilustración 34 - Comparación resultados de las varias pruebas respecto los tiempos medios de vuelo

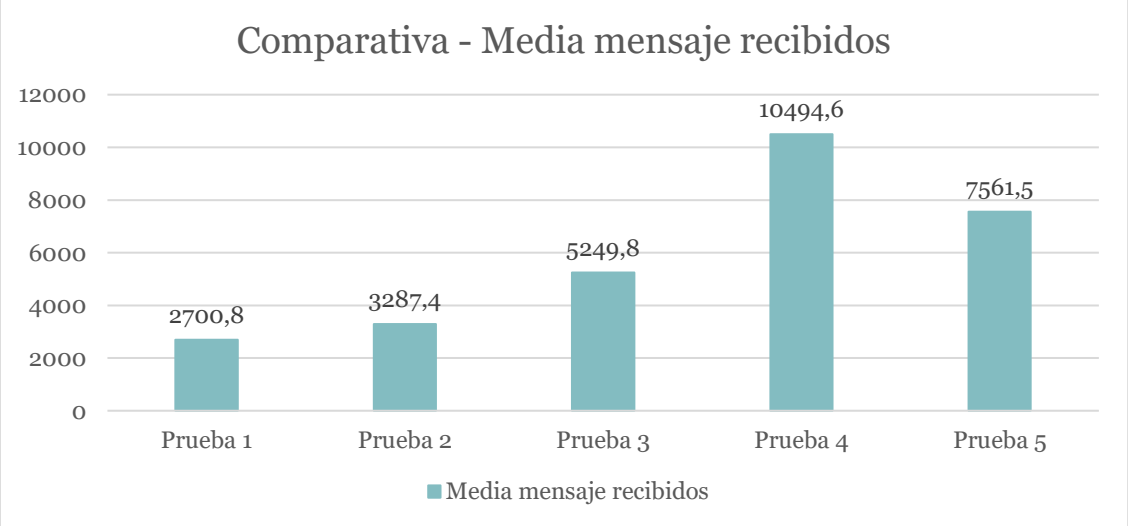

Ilustración 35 - Comparación resultados de las varias pruebas respecto los mensajes recibidos.

Como último punto a destacar, se ha querido comprobar más o menos cuanto se tarda en cambiar de punto de ruta, es decir, se busca mediar cuanto tiempo se tarda en cambiar al siguiente punto de ruta una vez alcanzado el destino y haber enviado los reconocimientos de movimiento. Para ello, este pequeño análisis se ha implementado sobre la prueba 2, ya que es la que más carga computacional acarrea. Previamente a la extensión de MUSCOP el tiempo medio de espera en un punto para poder proceder l siguiente era de 0,5 segundos (en media). En la ilustración 36 se adjuntan los datos extraídos en cada nodo tras alcanzar el segundo punto de ruta y actualizar el valor previo, del primer punto al segundo hasta moverse de nuevo.

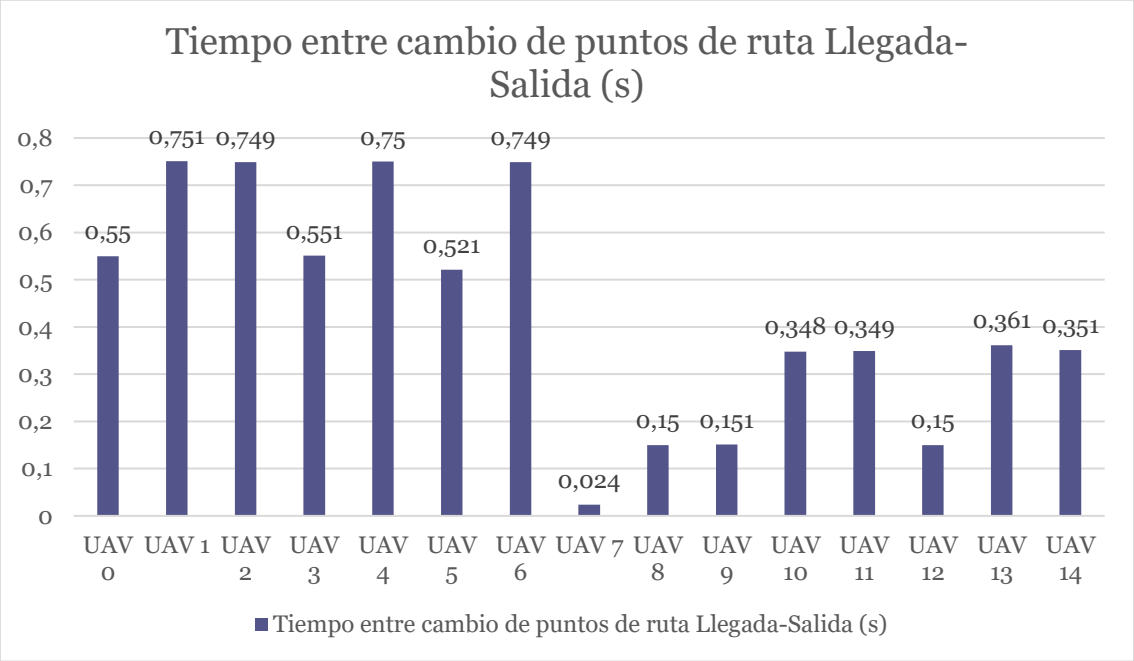

Ilustración 36 - Periodicidad entre cambio de puntos de ruta en la prueba 2.

En total, el tiempo medio obtenido entre el cambio realizado de los puntos de ruta 1 y 2 es de 0,43 segundos. Este tiempo se asemeja al buscado originalmente. Cabe destacar que hay fluctuaciones entre las ramas derecha e izquierda, ya que una presenta un coste mayor. Si suponemos el peor caso, donde todos rondan los 0,8 segundos, se observaría un incremento de 0,3 segundos del total. Aunque, por otro lado, si suponemos el peor caso donde todos rondan sobre los 0,3 segundos, se observaría un decremento de 0,2 segundos respecto al original.

6. Conclusión y Trabajos Futuros

El sector de los vehículos aéreos no tripulados tiene una amplia variedad de frentes abiertos. Este trabajo ha tenido como objetivo en todo momento aportar nuevas características innovadoras para el uso de enjambres de drones mediante el uso del simulador Ardusim.

Hoy en día los protocolos vigentes del mercado se centran en la coordinación del enjambre siempre y cuando cada nodo sea capaz de comunicarse dentro del rango de comunicaciones establecido; es por eso por lo que, mediante la extensión propuesta al protocolo MUSCOP, se ha buscado abarcar la retransmisión y comunicación de aquellos nodos que se encuentran fuera del alcance del nodo maestro del enjambre.

Los requisitos por suplir han podido ser plasmados de forma conceptual para así poder abordar una solución óptima. Durante el proceso del desarrollo se ha ido adaptando cada concepto a algunas necesidades descubiertas a posteriori, dando lugar a nuevos conceptos.

Mediante un arduo desarrollo se ha conseguido extender MUSCOP para que éste sea capaz de comunicarse con aquellos drones que estén fuera de rango. Para lograr dicha meta se ha empezado descubriendo los vecinos de cada UAV. Tras esto, cada nodo del enjambre ha conseguido actualizar su número de saltos hacía el maestro y, además, crear una lista de enrutamiento. Seguidamente, durante el transcurso de la misión, se ha conseguido abordar cada punto de la ruta con éxito gracias a la retransmisión de los reconocimientos.

No obstante, no solo se han expuesto los conceptos de forma teórica, sino que, además, se han realizado una serie de experimentos prácticos para dar validez al producto obtenido mediante distintas pruebas con configuraciones totalmente distintas donde, en algunos casos, se respetan algunas características en común. Las pruebas multi-salto realizadas han conseguido plasmar unos resultados aceptables donde se puede apreciar el correcto funcionamiento de la extensión, y donde se demuestra que la cantidad de drones empleados en un enjambre tiene una gran influencia en términos de métricas como costes temporales y sobrecarga de la red.

Por último, el trabajo mostrado en esta memoria da paso a una gran infinidad de ideas y proyectos a desarrollar, aunque también existe la posibilidad de mejorar y optimizar las ideas planteadas previamente. A continuación, se expone una lista de posibles trabajar futuros:

- El primer trabajo planteado está relacionado con Ardusim. Una de las características de este simulador es el envío de mensajes mediante difusión. La idea principal es que el lector se interese por la implementación de un envío de mensajes por unicast ya que, de esta forma, se da paso a nuevas vertientes dentro del simulador, aunque la implementación de dichos mecanismos debería necesitar un cierto grado de adaptación hacía los protocolos actuales vigentes.

- Siguiendo la misma vertiente es posible plantear un nuevo patrón, por ejemplo, del tipo maestro-esclavo un poco más avanzado, el cual incluya de por sí las funcionalidades de retransmisión. De esta forma, no solo MUSCOP sería capaz de trabajar con nodos más allá del alcance radio, sino que otros protocolos como MBCAP o Vision también tendrían esta posibilidad. Sin embargo, MUSCOP debería ser modificado de nuevo para que se adapte al patrón general, y no genere errores de inconsistencia.

- Por otra parte, el protocolo MUSCOP puede ser optimizado para que cubra de forma más eficiente la retransmisión. Una posible idea clave sería la mejora del reenvío de los reconocimientos; en este caso, los nodos que no reenvíen mensajes (aquellos más alejados, situados en la periferia) son los primeros en enviar sus reconocimientos. Los demás nodos intermedios hacen *piggybacking* cuando reciben los mensajes (incluyendo todos los posibles nodos de los que ha recibido un reconocimiento). La ilustración 37 muestra un posible escenario a desarrollar.

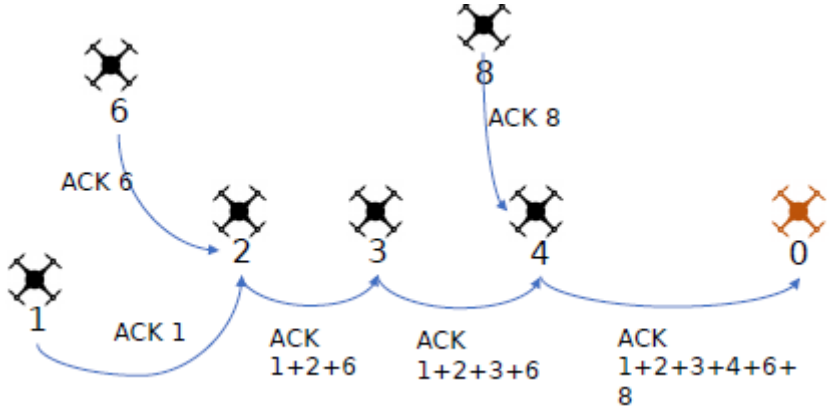

Ilustración 37 - Trabajo futuro: Optimización del reenvío de reconocimientos.

- El punto anterior buscaba reforzar y optimizar el último paso desarrollado, el envío de reconocimientos. En este apartado se busca informar y plantear al lector otro tipo de refuerzo para el protocolo MUSCOP, concretamente para la construcción de la lista de enrutamiento y el cálculo de saltos. En este caso los mensajes del maestro se encargan de enviar una lista de todos sus vecinos, junto con una lista con todos aquellos nodos cuyo reconocimiento aún está pendiente de recibir. De esta forma, solo los nodos que están en el listado de pendientes envían su reconocimiento (para reducir el tráfico). El escenario correspondiente se muestra en la Ilustración 38.

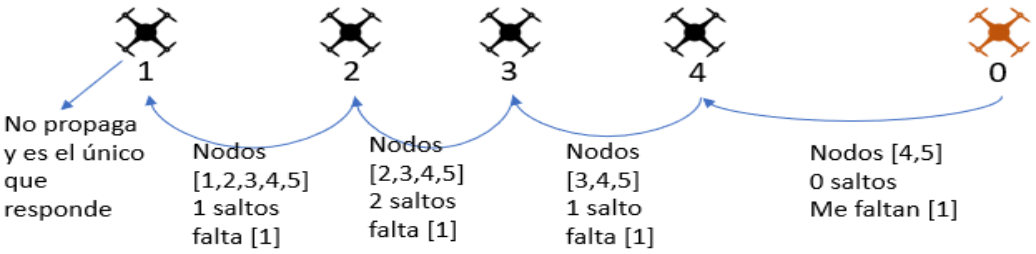

Ilustración 38 - Trabajo Futuro: Optimización construcción lista de enrutamiento y cálculo de saltos.

Referencias Bibliográficas

[1] Francisco Fabra, Carlos T. Calafate, Juan Carlos Cano, Pietro Manzoni, "ArduSim: Accurate and real-time multicopter simulation", Simulation Modelling Practice and Theory, Volume 87, September 2018, Pages 170-190.

[2] Francisco Fabra, Willian Zamora, Pablo Reyes, Julio A. Sanguesa, Carlos T. Calatafe, Juan-Carlos Cano y Pietro Manzoni (2020). *MUSCOP: Mission-Based UAV Swarm Coordination Protocol* en IEEE Xplore, vol. 8, pp. 72498-72511.

[3] W. Chen, J. Liu and H. Guo, "Achieving Robust and Efficient Consensus for Large-Scale Drone Swarm," in IEEE Transactions on Vehicular Technology, vol. 69, no. 12, pp. 15867-15879, Dec. 2020, doi: 10.1109/TVT.2020.3036833.

[4] C. Sampedro et al., "A flexible and dynamic mission planning architecture for UAV swarm coordination," 2016 International Conference on Unmanned Aircraft Systems (ICUAS), 2016, pp. 355-363, doi: 10.1109/ICUAS.2016.7502669.

[5] C. GUERBER, N. LARRIEU and M. ROYER, "Software defined network based architecture to improve security in a swarm of drones," 2019 International Conference on Unmanned Aircraft Systems (ICUAS), 2019, pp. 51-60, doi: 10.1109/ICUAS.2019.8797834.

[6] Yu, X., Li, Q., Queralta, J. P., Heikkonen, J., & Westerlund, T. (2021). Applications of uwb networks and positioning to autonomous robots and industrial systems. arXiv preprint arXiv:2103.13488.

[7] T. Han et al., "Emerging Drone Trends for Blockchain-Based 5G Networks: Open Issues and Future Perspectives," in IEEE Network, vol. 35, no. 1, pp. 38-43, January/February 2021, doi: 10.1109/MNET.011.2000151.

[8] Francisco Fabra, Carlos T. Calafate, Juan Carlos Cano, Pietro Manzoni, ArduSim: Accurate and real-time multicopter simulation, Simulation Modelling Practice and Theory, Volume 87, 2018, Pages 170-190, ISSN 1569-190X, https://doi.org/10.1016/j.simpat.2018.06.009.

[9] Ardusim Project, https://github.com/GRCDEV/ArduSim, Accesed:July 7,2021

[10] Fabra, Francisco; Zamora, Willian; Sangüesa, Julio; Calafate, Carlos T.; Cano, Juan-Carlos; Manzoni, Pietro. 2019. "A Distributed Approach for Collision Avoidance between Multirotor UAVs Following Planned Missions" Sensors 19, no. 10: 2404. https://doi.org/10.3390/s19102404

[11] Francisco Fabra, Willian Zamora, Joan Masanet, Carlos T. Calafate, Juan-Carlos Cano, Pietro Manzoni, Automatic system supporting multicopter swarms with manual guidance, Computers & Electrical Engineering, Volume 74, 2019, Pages 413-428, ISSN 0045-7906, https://doi.org/10.1016/j.compeleceng.2019.01.026.

[12] Wubben, Jamie; Fabra, Francisco; Calafate, Carlos T.; Krzeszowski, Tomasz; Marquez-Barja, Johann M.; Cano, Juan-Carlos; Manzoni, Pietro. 2019. "Accurate

Landing of Unmanned Aerial Vehicles Using Ground Pattern Recognition" Electronics 8, no. 12: 1532. https://doi.org/10.3390/electronics8121532

[13] F. Fabra et al., "MUSCOP: Mission-Based UAV Swarm Coordination Protocol," in IEEE Access, vol. 8, pp. 72498-72511, 2020, doi: 10.1109/ACCESS.2020.2987983.