uc3m | Universidad Carlos III de Madrid

Doble Grado en Ingeniería Informática y Administración de Empresas.

2021-2022

*Trabajo Fin de Grado*

# Desarrollo de arquitectura de enjambres de drones descentralizados en entorno simulado

---


Lázaro Fornis Herranz

Tutor

Daniel Amigo Herrero


Colmenarejo, junio de 2022


*[Incluir en el caso del interés de su publicación en el archivo abierto]*
Esta obra se encuentra sujeta a la licencia Creative Commons **Reconocimiento – No Comercial – Sin Obra Derivada**

# RESUMEN

Los enjambres de drones actualmente son tendencia, no solo por multiplicar el rendimiento con respecto a sus integrantes actuando en solitario, sino por poder trabajar de forma distribuida generando una inteligencia colectiva completamente autónoma, sin depender de un nodo central inteligente que les hace vulnerables ante ataques y fallos. En este trabajo se diseña e implementa una arquitectura para la simulación de enjambres de drones distribuidos haciendo uso de AirSim, PX4, MAVSDK y para el lenguaje de programación Python, la forma de simulación más cercana a los vuelos con drones reales.

La arquitectura ha sido probada y verificada para enjambres de hasta 6 drones y diversos casos de uso simulando funcionamientos en enjambre. Específicamente, los drones son capaces de comunicarse, colocarse en formaciones de forma distribuida y sin chocarse, desplazarse mientras mantienen la formación e incluso evadir obstáculos de forma individual o ayudándose entre ellos.

Este trabajo sienta las bases para futuros trabajos de enjambres de drones descentralizados diseñados y probados íntegramente en simulación que puedan funcionar con el menor esfuerzo posible en la realidad.

**Palabras clave:** Enjambre distribuido, descentralización, drones, AirSim, PX4, MAVSDK, MAVSDK-Python

# ABSTRACT

Drone swarms are currently a trend, not only for increasing the performance compared to those acting alone, but also for being able to work in a distributed way, generating a completely autonomous collective intelligence, without depending on an intelligent central node that makes them vulnerable to attacks and failures.

This work designs and implements an architecture for distributed drone swarm on simulation, using AirSim, PX4, MAVSDK and the Python programming language, the nearest way to simulate real drone flights. The architecture has been tested and verified for swarms of up to 6 drones and several use cases simulating swarm operations have been carried out. Specifically, the drones are able to actively communicate, place themselves in formations in a distributed and collision-free manner, to fly while maintaining the formation, and even to evade obstacles either independently or by helping each other.

This work sets the basis for future work on decentralised drone swarms designed and tested entirely in simulation that could be operated in real-world conditions with as little effort as possible.

**Key words**: Distributed Swarm, decentralisation, drones, AirSim, PX4, MAVSDK, MAVSDK-Python

# DEDICATORIA

A mi padre por todo, a mi tío Jesús y a mi tía Araceli que impulsaron mi formación, a mi hermano y a Vanesa por apoyarme en todo momento. A Daniel, mi tutor, por corregirme y aconsejarme.

# ÍNDICE DE CONTENIDOS

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# 1 INTRODUCCIÓN

## 1.1 Visión general

¿Es posible mejorar el rendimiento y eficacia la mayoría de los trabajos sobre drones publicados hasta la fecha? Si las acciones o tecnologías aportadas en los mismos pudieran ser ejecutadas simultáneamente por varios de estos UAV, y si, además, pudieran comunicarse entre sí para realizar tareas de manera óptima reaccionando ante cualquier imprevisto, es razonable pensar que la respuesta a esta pregunta es afirmativa. Pero para conseguirlo, primero es necesario disponer de una plataforma de simulación que permita investigar y probar esta propuesta hasta ser llevada a cabo en la realidad con éxito.

La mayoría de las plataformas de simulación existentes no sirven para el desarrollo de enjambres de drones descentralizados por varias razones [1]. En primer lugar, están diseñadas para un propósito específico, y sus funcionalidades son difíciles de ampliar. En segundo lugar, las plataformas existentes carecen de compatibilidad para ser aplicadas a diferentes tipos de escenarios. En tercer lugar, el modelado de estas plataformas está demasiado simplificado para simular con precisión la dinámica del movimiento del vuelo y la comunicación ruidosa, lo que puede provocar una diferencia de rendimiento entre la simulación y la aplicación en el mundo real. Para abordar las cuestiones mencionadas, en este trabajo se diseña e implementa una arquitectura para la simulación de enjambres de drones distribuidos de forma que sea lo más cercana posible a los vuelos con drones reales.

Para poder desarrollar esta plataforma con éxito primero debemos conocer el estado de arte de los elementos que formarán parte del sistema, elaborar una metodología adecuada de trabajo y un presupuesto que contabilice el valor del proyecto. También, debemos conocer tanto la legislación vigente sobre el ámbito de aplicación de la plataforma como su impacto económico y social. Entonces, podremos comenzar su desarrollo, el cual, una vez finalizado, debe ser verificado mediante las técnicas expuestas en la metodología elaborada.

## 1.2 Motivación del trabajo

Para desarrollar un enjambre de drones deberemos solventar el problema de la descentralización, es decir, tratar no depender de un único nodo (dron o estación de tierra) desde donde se gestiona al resto de los drones y se marca el flujo que debe seguir una determinada misión. Ya que, si este nodo principal sufre algún daño, sufriría el mismo daño el resto de los drones, pues este les controlaba.

Por otro lado, si consiguiéramos dicha descentralización, ¿cómo coordinamos y comunicamos a todos esos drones sin depender de un nodo central? Veamos los beneficios que supondría conseguirlo.

Las posibilidades de un enjambre de drones son innumerables. Desde misiones de rescate o búsqueda, donde los drones se comuniquen repartiendo automáticamente la zona a investigar de manera óptima, hasta misiones de ataque, donde se coordinen automáticamente realizando ataques tácticos basados en estrategias militares, en las cuales puedan capturar drones enemigos e incluso algunos puedan llegar a sacrificarse por un bien común mayor. Hablamos de un ataque sin precedentes, en el que pueden tomarse decisiones estratégicas en milisegundos, algo similar a lo que ocurre con el HFT (high frequency trading) en el mercado bursátil [2]. Un ser humano no supone un gran rival ante un ejército volador que define estrategias, se coordina y toma decisiones en un abrir y cerra de ojos. La única forma razonable de enfrentarse a algo así es con otro enjambre de drones como argumenta, Jesús Martín Sánchez, líder del departamento de UAS/Swarming de la empresa Escribano en su entrevista sobre el programa LISS [3].

Claro que, hasta la fecha, un ataque con varios drones es relativamente sencillo de mitigar. La clave está en abatir al UAV “con más antenas”. Si cae el rey, cae su ejército, pues en estos casos no existe un enjambre como tal. Aquí es donde se encuentra la complejidad del trabajo, en definir una arquitectura que permita descentralizar el sistema, pudiendo tomar cada dron sus propias decisiones, y votar entre todos las acciones que afecten al enjambre. De esta manera, si cualquier dron es abatido, sus funciones dentro del enjambre son sustituidas o, si es posible, distribuidas entre los demás, continuando con la misión.

Una de sus mayores ventajas sería diseñar un sistema escalable y adaptable, en el que se pudieran incorporar a la red cualquier elemento, persona o vehículo dirigido o no, como expone la empresa Escribano en su programa LISS [4]. Un ejemplo sería que los drones

colaborasen con los vehículos que los despliegan como los cazas o los carros de combate terrestres e incluso con los mismos soldados para cumplir un objetivo en el que ellos también deban de participar. Incluso en el ámbito marítimo, podrían colaborar con la marina, detectando submarinos enemigos, atacarlos, etc.

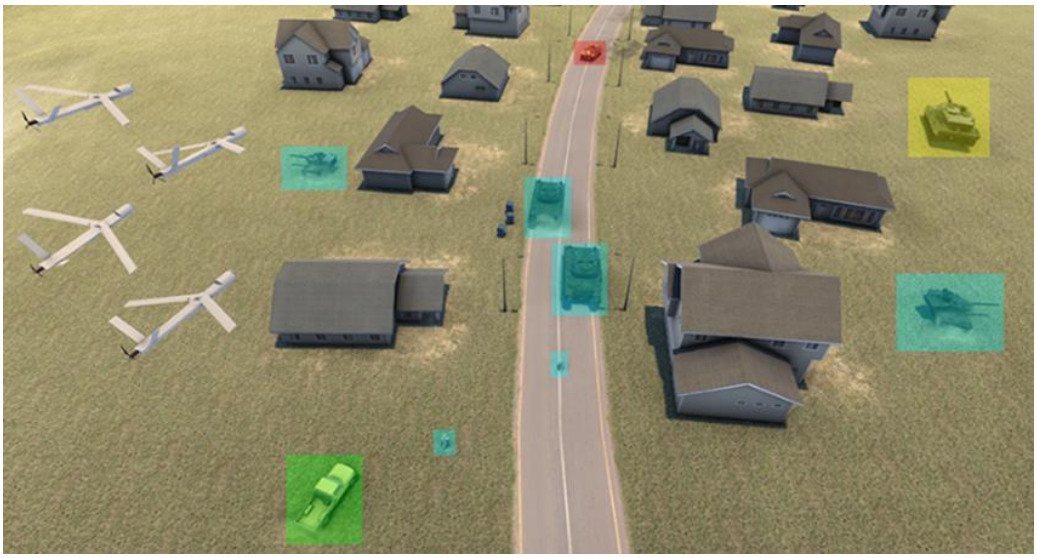


*Figura 1. Enjambre de drones del programa LISS [4]*

Pero, no todas las posibilidades de esta tecnología se encuentran en el ámbito militar. También, podrían colaborar en materias de protección civil, emergencias o seguridad ante cualquier calamidad, atentado o simple vigilancia.

Por último, para que sea aplicable en el mundo real, hay que tener en cuenta las limitaciones físicas del enjambre [1], como la pérdida de paquetes en la comunicación y la percepción de los sensores influida por el ruido. Sumado a la dificultad que conlleva este experimento en el mundo real debido tanto a los fondos necesarios para construir un enjambre de drones, como el riesgo de daños causados por la caída [5] de un UAV del enjambre o a la normativa que limita el vuelo de los UAV en las zonas urbanas. Por lo tanto, la validación y las pruebas basadas en la simulación pueden facilitar considerablemente el estudio de la planificación y el control de enjambres de drones.

## 1.3 Objetivos

A continuación, se detallan una serie de objetivos a cumplir por este trabajo para considerar que se ha encontrado una solución al problema propuesto. Estos objetivos son:

- Generar una arquitectura de simulación de enjambre de drones distribuidos ampliable, mantenible, sin licencia comercial y lo más cercana posible a los vuelos con drones reales. Donde se puedan controlar varios drones al mismo tiempo. Los drones actuarán de forma coordinada, comunicándose y cooperando entre sí de forma descentralizada, es decir, sin depender de un nodo central que marque el rumbo de las actuaciones del enjambre.
- Elaboración de formaciones en enjambre para viajar de manera uniforme y sincronizada hacia un objetivo.
- Permitir una navegación segura sin que estos drones colisionen entre ellos ni con otros objetos inmóviles del entorno. Para ello, deberán de colaborar de forma que compartan información relevante para evitarse entre sí y a los obstáculos del medio.
- El sistema será de código abierto permitiendo a cualquier usuario mejorarlo de forma que se puedan generar sus propias misiones y llegar a cumplir tareas como las vistas en la motivación. Es decir, poder simular enjambres de drones en misiones de exploración, reconocimiento, búsqueda, etc. Al ser descentralizado el sistema garantizará que la misión pueda continuar mientras queden drones que puedan realizarla o lo que es lo mismo, si algún dron es abatido o inhabilitado el enjambre pueda percibirlo y continuar con la misión de la manera más conveniente al caso.

## 1.4 Estructura del documento

A continuación, se expone la estructura del resto del documento.

- En el Apartado 2 se exponen las tecnologías más novedosas necesarias para la creación del entorno de ejecución, lo cual incluye a los drones y por su puesto a los enjambres de drones.
- El Apartado 3, enumera los costes y coordinación, de los recursos fundamentales para la elaboración de este trabajo.

- Toda la legislación relevante referente a los vehículos aéreos no tripulados se encontrará en el Apartado 4, junto con las diferentes razones por las que este trabajo tiene influencia en el ámbito social y económico.
- El desarrollo de la solución al problema planteado se encuentra en el Apartado 5.
- Seguidamente, en el Apartado 6, se detalla una evaluación de dicha solución.
- Por último, en el Apartado 7 introduce las formas de continuar con este trabajo, así como las conclusiones obtenidas del mismo.

# 2 ESTADO DEL ARTE

En este apartado se muestra la situación actual de los diferentes componentes del entorno de simulación del enjambre. Es de vital importancia obtener y procesar toda la información y conocimiento posible sobre cada aspecto técnico del trabajo para poder seleccionar la mejor opción dentro de las que nos ofrece cada componente del entorno de simulación.

## 2.1 Drones

El primer paso para conocer un enjambre de drones es conocer las características de sus miembros los UAV o vehículos aéreos no tripulados, también conocidos coloquialmente como drones.

Los UAV se pueden clasifican en base a diferentes parámetros:

- Según su peso encontramos:

| Tipo | Peso (Kg) |
|---|---|
| Nano | < 0,25 |
| Micro | entre 0,25 y 2 |
| Small | entre 2 y 25 |
| Medium | entre 25 y 150 |
| Large | > 150 |

*Tabla 1. Tipos de UAV según su peso [6]*

- En base al número de alas y motores que posee:

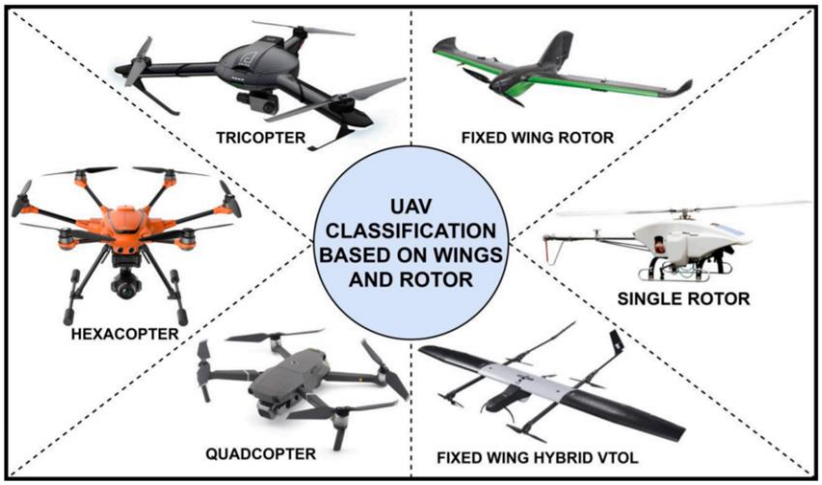



*Figura 2. Tipos de UAV según su número de alas y motores [7]*

- En función de su altitud y alcance:

| Tipo | Altitud (m) | Alcance (Km) |
|---|---|---|
| Hand Held | <600 | <2 |
| Close | <1500 | <10 |
| NATO | <3000 | <50 |
| Tactical | <5500 | <160 |
| Medium Altitude Long Range | <9100 | <200 |
| High Altitude Long Range | >9100 | NA |
| Hypersonic | <15200 | >200 |

*Tabla 2. Tipos de UAV según su altitud y amplitud [7]*

- Según su uso:

| Tipo | Uso |
|---|---|
| Personal | Fotografía, video, acrobacias… |
| Comercial | Vigilancia, reparto, cine… |
| Gubernamental | Colaboración con Administraciones Públicas |
| Militar | Ataques, defensa, abastecimiento… |

*Tabla 3. Tipos de UAV según su uso [7]*

Todos ellos están equipados con componentes que necesitan mucha energía, como la carga de pago (sensorización para captar el entorno), motores, placas de cómputo, etc. Por ello, sus baterías son de litio (Li-ion), que tienen el mayor índice de descarga y son las más ligeras [8].

Existen problemas con las baterías de litio actuales, su capacidad debe encontrarse siempre entre el 20% y el 80%, de igual forma existen límites de voltaje y temperatura, todo ello representado en la Figura 3. En el momento que se superen alguno de estos límites, se origina un riesgo irremediable sobre la batería, la cual puede llegar a incendiarse. Al intentar incrementar la capacidad de la batería de un dron, aumenta el peso que debe cargar al volar, lo que a su vez aumenta el consumo de batería y por ende se disminuye el tiempo de vuelo [9]. La sección verde del gráfico de la Figura 4 indica las condiciones necesarias para el funcionamiento seguro de las baterías de iones de litio.

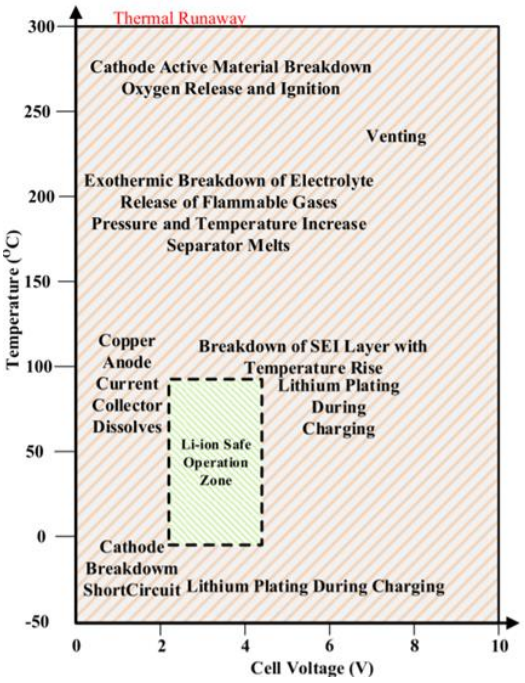



*Figura 3. Zona de funcionamiento seguro de las baterías de litio [10]*

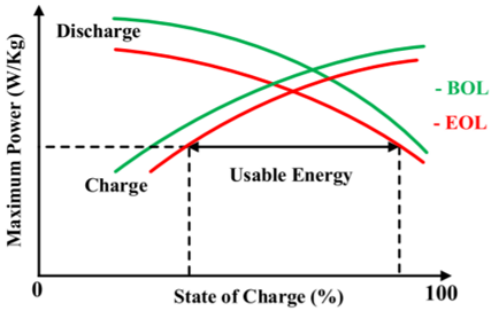



*Figura 4. Características de carga y descarga de las baterías de litio [8]*

Por ello se está trabajando en diferentes formas de cargar la batería de los drones, entre ellas encontramos:

- Baterías desechables: [11] son baterías que van equipadas en el dron. Conforme una se gaste, esta es desechada automáticamente como muestra la Figura 5 (a), de forma que el dron continúa en vuelo con menor peso, por lo que las baterías restantes tienen mayor duración.
- Paneles solares: [12] la superficie del dron es equipada con paneles solares, por el día se consigue energía suficiente para recargar la batería del dron mientras este está volando, y por la noche se usa la energía de la batería Figura 5 (b). El problema lo encontramos cuando el día no es suficientemente luminoso, ya que la actividad del dron se ve limitada.
- Rayo láser: [13], [14] consiste en un rayo láser que se proyecta sobre un panel en el centro del dron que le permite cargar su batería en vuelo, Figura 5 (c). Mediante esta tecnología se consiguió mantener 12 horas de vuelo con un cuadricóptero.
- Aprovechando las ráfagas de viento: [15], [16] se basa en la técnica de algunas aves para mantener el vuelo sin mover sus alas aprovechando las corrientes de aire, representado en la Figura 6, lo que les permite ahorrar energía. Por lo que

solo puede ser usado por drones alados y al igual que los paneles solares, existe una gran dependencia meteorológica.

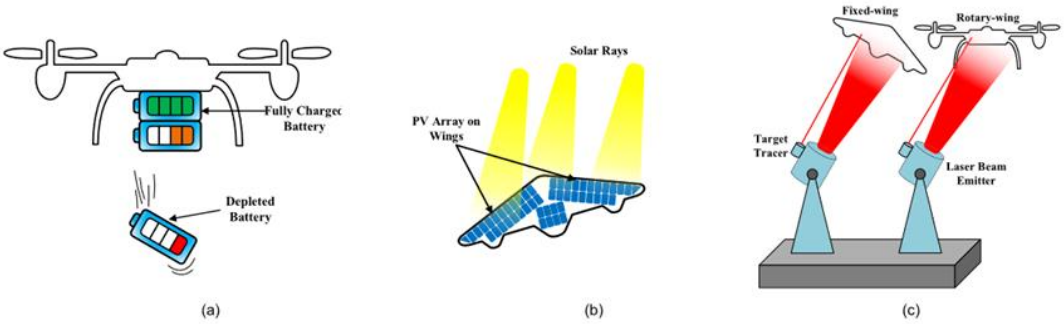



*Figura 5. Formas de cargar las baterías de los drones [8]*

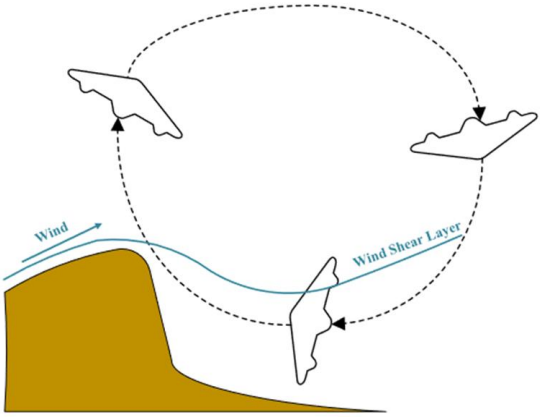



*Figura 6. Ahorro de energía aprovechando las ráfagas de viento [8]*

Por lo que la solución definitiva la encontramos con la carga basada en campos electromagnéticos, concretamente en la resonancia magnética donde dos bobinas (una que origina la emisión y otra que la recibe) repercuten a la misma frecuencia, también se puede incorporar una intermedia para mejorar la trasmisión. Permiten al UAV alcanzar características útiles tales como: resistencia al polvo o al agua, inconcebibles para un dron recubierto con placas solares, por ejemplo. Además, este tipo de carga ocupa menos espacio y permite transmitir más energía que cualquiera de las otras técnicas. Lo más interesante para nuestro estudio es que permite realizar múltiples cargas a través de múltiples frecuencias por lo que sería ideal para un enjambre [8].

En cuanto al dron que se utilizará en la simulación. Será un cuadricóptero ligero, de corto alcance y de uso personal, pues son características que contemplan los drones de menor coste, lo que facilitará el paso de la simulación al mundo real. A continuación, se muestra una imagen del dron Figura 7.

*Figura 7. Dron usado en las simulaciones [17]*

## 2.2 Enjambre de drones

Existen ciertos grupos de animales capaces de cooperar, generando ciertos patrones colectivos, interesantes científicamente y visualmente espectaculares [18]. Establecen estructuras ordenadas sin colisiones en un tiempo limitado [19], [20]. También, pueden reaccionar con extrema rapidez a los cambios del entorno, como la aparición repentina de un depredador o un obstáculo [21], [22]. Aunque estos sistemas son enormemente complejos, también están perfectamente optimizados y, por lo tanto, sus patrones de movimiento expresados siguen siendo graciosamente naturales [23].

Han surgido cientos de modelos para describir el movimiento colectivo sincronizado de animales, seres humanos o incluso células en migración [24]–[26]. Llamamos a estos sistemas autoorganizados porque las interacciones en ellos son locales y las decisiones las toman los propios agentes.

Conducir el comportamiento de tales sistemas hacia algún patrón deseable no es trivial [27]. En primer lugar, los drones son autónomos e imperfectos. Es decir, cada dron tiene su propio ordenador de a bordo para realizar los cálculos necesarios para controlar sus propias acciones, su propio sistema de sensores para medir las posiciones y velocidades relativas, y su propio dispositivo de comunicación para el intercambio de datos con los agentes vecinos. Estas características reflejan la definición actual de autonomía sensorial y reactiva descrita en [28].

En segundo lugar, estos sistemas deben funcionar sin control central. Es decir, aunque los agentes pueden observarse mutuamente e intercambiar información, no envían ni reciben órdenes de control directas porque no hay un líder dentro del grupo ni un supervisor externo, como una estación base o un supervisor humano.

Los mayores enjambres de drones hasta ahora fueron desarrollados para el mundo del espectáculo por Intel [29] y por Ehang [30] con más de 1000 drones cada uno; sin embargo, estos drones estaban programados individualmente para trayectorias predefinidas o eran controlados de forma centralizada y no satisfacían los criterios de autonomía mencionados Figura 8. También, el grupo musical Metallica incluyó recientemente docenas de drones en sus conciertos que parecían exhibir algún tipo de comportamiento de enjambre parcialmente autónomo mediante el uso de un sistema de posicionamiento en interiores y mecanismos de control centralizados [31].


*Figura 8. Espectáculo de luces con enjambre centralizado [29]*

Ahora que conocemos a sus integrantes y sus características podemos definir que es un enjambre de drones. Es un conjunto de drones descentralizado donde sus integrantes son capaces comunicarse y coordinarse con el resto para cumplir un objetivo común. Estos pueden ser centralizados o descentralizados como vemos en la Figura 9.

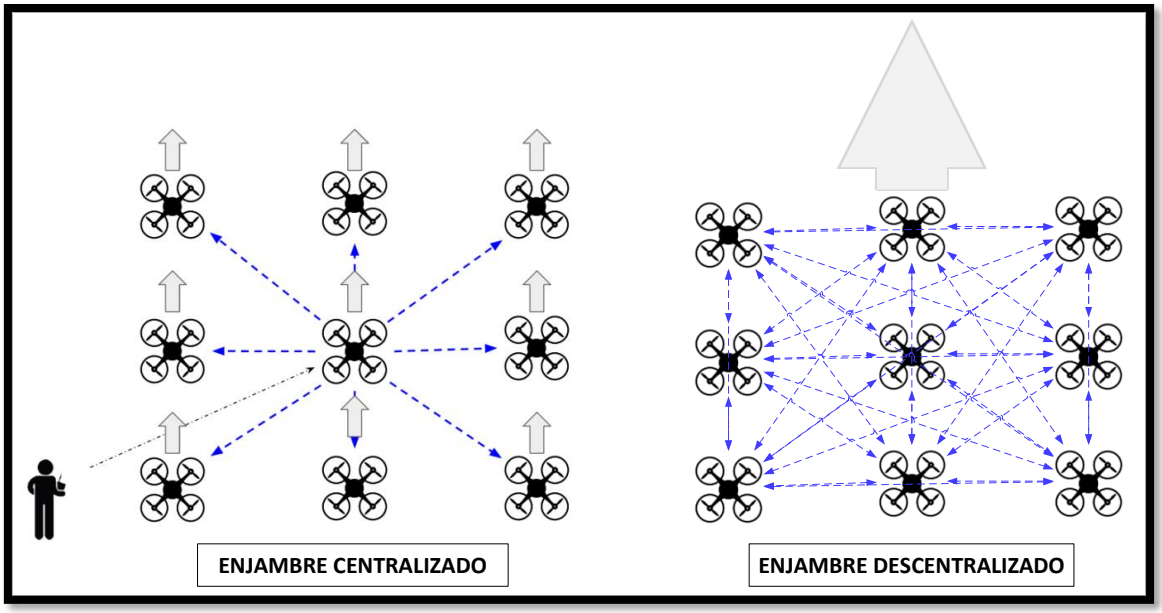



*Figura 9. Enjambre centralizado y descentralizado*

En este sentido, encontramos un gran abismo entre las simulaciones de enjambres y las pruebas reales, que limita fuertemente el número de miembros que pueden componerlo [32]. Aun así, la compañía Escribano ha conseguido volar un enjambre descentralizado compuesto por 12 drones [3].

El principal problema a la hora de programar un enjambre es controlar el comportamiento del grupo realizando la implementación sobre los individuos. Actualmente no existe una programación óptima de este comportamiento, únicamente se programan bloques de código para situaciones determinadas que cuando el problema o el número de individuos que componen el enjambre crecen, se observa una caída en el rendimiento general del sistema. Por lo que no se pueden ofrecer garantías suficientes para cumplir los estándares de verificación [33]. Aunque si se ha conseguido implementar un algoritmo óptimo para el reparto de tareas basado en el nivel de batería de cada UAV [34].

También para complementar su implementación, se han usado técnicas de algoritmos evolutivos como en el caso de la exploración óptima de zonas desconocidas [35], inspirada en la naturaleza (como lo haría una colonia de hormigas). También técnicas de aprendizaje profundo, las cuales han demostrado ser las mejores en la clasificación de imágenes, lo cual es una cualidad muy útil para un enjambre ya que sus miembros pueden observar desde diferentes puntos de vista a un mismo objetivo. El trabajo [36] explota esta cualidad, ya que presenta un método para la reconstrucción 3D del exterior de edificios mediante un conjunto centralizado de múltiples drones, que realizan un proceso

de captura de datos de forma autónoma y únicamente equipados con una cámara RGB Figura 10. La reconstrucción visual en 3D tiene como objetivo restaurar la estructura 3D del entorno o la escena a partir de la entrada de imágenes multivistas basadas en la teoría de la visión estereoscópica.

*Figura 10. Resultado de reconstrucción 3D del exterior de un edificio con un enjambre de drones centralizado [36]*

Una vez implementado, el enjambre siempre debería estar bajo la supervisión humana. Por lo que en la interacción enjambre-humano existe un punto crítico en el nivel de estrés que puede generar una persona observando una interfaz sobre la que se le bombardea con información recopilada por cada uno de los agentes. sobre la que se evidencia su falta de control por no comprender ciertas decisiones o actuaciones llevadas a cabo por el enjambre. Por lo que es necesario realizar predicciones de comportamiento y un filtrado de información para que el usuario pueda sentirse cómodo, seguro e informado de la situación [31]-[33].

Además, hay que prestar especial atención a la seguridad del enjambre, ya que pueden adherirse al mismo, miembros con intenciones maliciosas de control y sabotaje. Esto incluye el cifrado de las comunicaciones, donde se expone información valiosa en las conexiones entre drones [33].

Se espera que los enjambres autónomos no tripulados superen a un solo UAV complejo en términos de flexibilidad y robustez, ofreciendo así una mayor adaptabilidad, capacidad de supervivencia y tolerancia a los fallos [1]. Por ejemplo, en las misiones de

coordinación de ataques y de reconocimiento, un enjambre de vehículos aéreos no tripulados puede cubrir una zona determinada en menos tiempo que uno solo, y el fallo de cualquier miembro del enjambre no provocará el fracaso de toda la misión.

Los enjambres de drones pueden clasificarse de diferentes formas según su complexión y jerarquía [40]. La Figura 11 ilustra como los enjambres pueden clasificarse en total o parcialmente autónomos. Desde otro punto de vista, la clasificación puede concebirse en enjambres de una sola capa, en los que cada dron es su propio líder, y enjambres de varias capas con drones líderes en cada capa, que informan a sus líderes de una capa superior, donde una estación de servidor en tierra es la capa más alta de esta jerarquía.

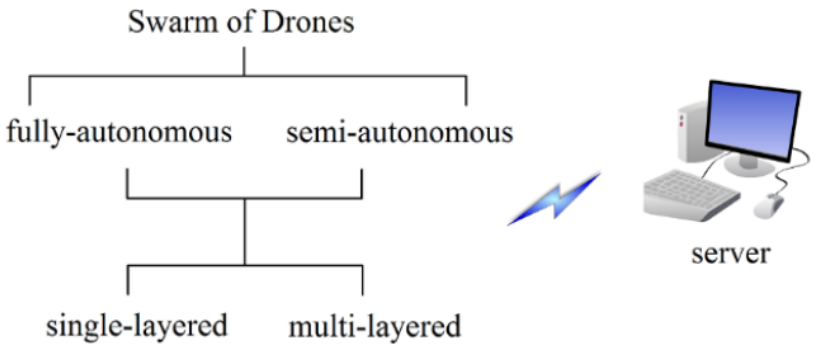



*Figura 11. Clasificación de enjambres [40]*

Para resolver el problema planteado necesitamos un enjambre completamente autónomo de una sola capa que garantice una completa descentralización.

Algunos ejemplos de arquitecturas de enjambre de drones existentes son los siguientes:

- El MultiUAV2 [41] publicado por el Laboratorio de Investigación de la Fuerza Aérea de Estados Unidos, que forma parte del software MATLAB y Simulink, representa una plataforma de simulación capaz de simular múltiples UAVs que cooperan para cumplir misiones tácticas. Sin embargo, esta plataforma puede suponer un problema de derechos de autor para el grupo de investigación sin una licencia comercial.
- CoUAV [42] permite una rápida implementación de la simulación de múltiples UAV, y su código fuente y su demostración están disponibles para el acceso público. Pero en esta plataforma, un miembro de un enjambre de UAV es controlado por un centro de control en tierra, lo que constituye un esquema centralizado en lugar de distribuido.

- La plataforma de simulación "Infoplanner", de código abierto, de Schlotfeldt y sus compañeros [43] es capaz de simular la planificación cooperativa descentralizada utilizando un sensor imperfecto.

## 2.3 Protocolo de comunicación

Para comunicar y controlar a cada UAV existe el protocolo MAVLink [7]. Este es Open Source y liviano, permitiendo manejar varios drones simultáneamente, lo que le hace adecuado para esta tarea. Los datos que se manejan son temas, sobre los que se puede ejecutar un modelo en el que un encargado publica varios temas y los demás los reciben, estos pueden emitir información sobre el tema, la cual también es recibida por todos (publicación-suscripción). Por otro lado, cuenta con otro modelo en el que la información solo queda entre el emisor y el receptor (punto a punto), utilizado en subprotocolos como el de misión.

La información es transmitida en paquetes de datos de 14 bytes compuestos por la siguiente información [44]:

| ACRONYM | STX | LEN | INC FLAG | CMP FLAG | SEQ | SYS ID | COMP ID | MSG ID | PAYLOAD | Checksum | SIGNATURE |
|---|---|---|---|---|---|---|---|---|---|---|---|
| SIZE IN BYTES | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 3 | 0-255 | 2 | 13 |
| SHORT DESCRIPTION | Packet start marker | Payload Length | INCOMPATIBILITY FLAGS | COMPATIBILITY FLAGS | Packet Sequence No. | System ID | Component ID | Message ID | Payload Data | Checksum | (OPTIONAL) SIGNATURE |

*Figura 12. Estructura de los paquetes de MAVLink 2.0 [45]*

- STX o start of text: Sirve para marcar el comienzo de un nuevo paquete de datos.
- SEQ: Identifica el lugar que ocupa un paquete en una serie de paquetes, lo que permite identificar si algún paquete se ha perdido.
- SYS ID: Identificador del sistema emisor.
- COMP ID: Identificador del componente emisor.
- PAYLOAD: Datos de la carga de pago, opcionalmente se puede incluir el identificador del sistema y/o componente de destino.
- Checksum: Almacena el resultado de un cálculo sobre un bloque de datos que al ser calculado y comprobado en el receptor le permite saber si algún dato se ha alterado. Utiliza X.25 CRC.
- SIGNATURE (13B): Campo opcional que previene la falsificación de paquetes mediante la firma de estos. Está compuesto por:
    - LinkID (1B): Identificador de conexión suele coincidir con el canal

- Timestamp (6B): Medido en microsegundos marca la emisión del paquete y aumenta de forma monótona para una misma conexión.
- Sign (6B): Se trata de un hash (resumen cifrado) del paquete completo (sin contar este campo) realizado mediante SHA-256 en cuya clave está incluido el timestamp del paquete. La clave (32B) se almacena en los extremos del canal.

Como podemos observar se trata de un sistema integro y fiable que al ser Open Source permite ser analizado y también mejorado mediante investigación.

De hecho, en un artículo reciente se ha implementado un control de fallos para este protocolo permitiendo su ajuste y demostrando su eficacia ante fallos de medición del dron [46].

## 2.4 Controlador de vuelo

El controlador del dron es el encargado de recibir las órdenes (comandos) del usuario y ejecutarlas mediante su correspondiente traducción en impulsos eléctricos para la actuación del hardware del dron. Este controlador se apoya en los sensores del dron, para poder realizar cálculos en base a los datos obtenidos de los mismos, determinar la situación del dron, y poder responder adecuadamente a los comandos que se le soliciten. Los controladores más destacados de código abierto son Ardupilot y PX4.

### 2.4.1 Ardupilot APM

El firmware de Ardupilot, como el resto del software que desarrolla, está licenciado bajo la GPL (GNU General Public License) [47]. Esta licencia es más restrictiva que la licencia PX4 y garantiza que el software derivado original continúe manteniendo el catálogo de software libre. Las aplicaciones obtenidas del firmware pueden ser utilizadas con fines comerciales, pero cualquier cambio debe ser compartido y publicado en el código fuente, indicando los cambios realizados y las instrucciones de uso para cualquier usuario que quiera usarlo. Tampoco permite la creación de diferentes sublicencias, es decir, si un programa se desarrolla utilizando Ardupilot, debe seguir teniendo la licencia GPL. Forzar a publicar de esta manera el trabajo realizado puede hacer que el sector privado pierda

interés en usar la plataforma, pero por otro lado es beneficioso para el desarrollo y mejora del firmware.

### 2.4.2 PX4 Pixhawk

Pixhawk es un sistema avanzado de conducción autónoma diseñado como parte del software de código abierto PX4. Se trata de una ampliación de ArduPilot APM, debido a que el procesador APM es limitado y no puede ejecutar algoritmos más complejos. Pixhawk integra un procesador ARM (Advanced RISC Machine) de 32 bits, con mayor potencia que su antecesor [48]. Asimismo, se basa en el sistema operativo NuttX, que demuestra un rendimiento, flexibilidad y fiabilidad excepcionales en el campo de la conducción autónoma. Por esta razón ha sido seleccionado como el controlador que incorporarán los drones del enjambre simulado.

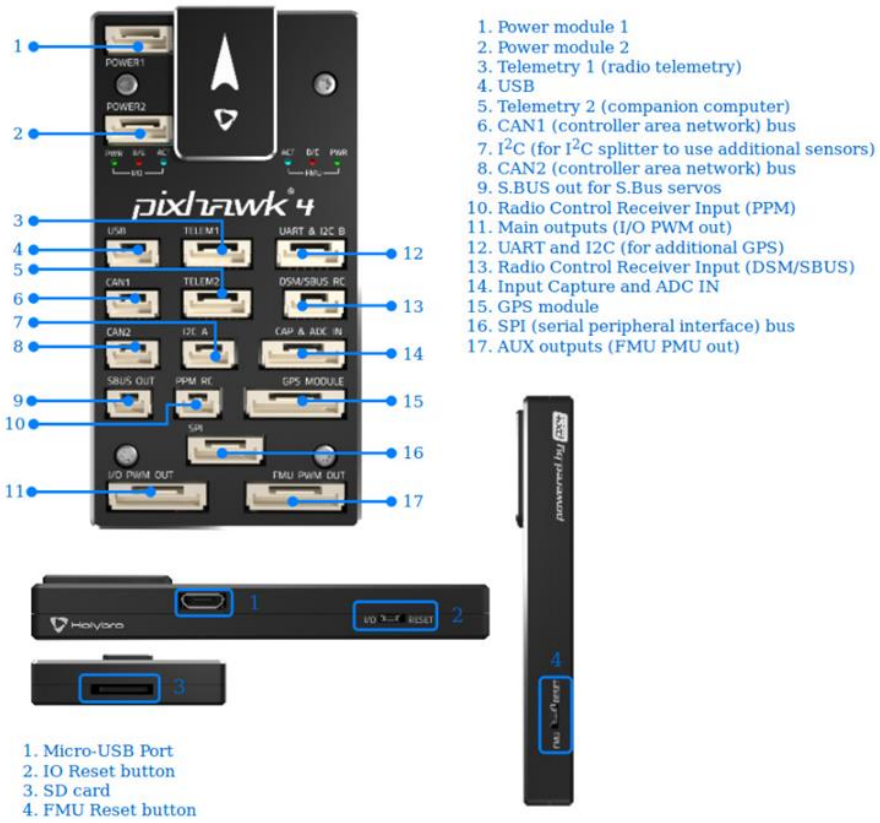



*Figura 13. Partes del controlador de vuelo PX4 [49]*

Todos los componentes de hardware de Pixhawk se pueden consultar en el manual oficial de PX4 [7].

## 2.5 Simulador

Para realizar la simulación del sistema se presentan dos conocidos simuladores de código abierto, Gazebo y AirSim. Expondremos en que consiste cada uno de ellos y cuál ha sido el elegido para este trabajo.

### 2.5.1 Gazebo

Gazebo es un simulador 3D de código abierto, que simula robots para una amplia gama de ámbitos, como muestra la Figura 14, incluido la robótica marítima o aérea. Entre sus características principales encontramos [50]:

- Gazebo admite el uso de múltiples servidores para mejorar el rendimiento.
- Utilizando la información espacial, puede cargar y descargar automáticamente los activos de simulación para mejorar drásticamente el rendimiento. Esta característica se combina bien con la simulación distribuida.
- Soporte multiplataforma en Linux y MacOS.
- Integración en la nube.
- Integración con ROS.
- Cámaras monoculares, cámaras de profundidad, LIDAR, IMU, sensores de contacto, altímetro y magnetómetro están disponibles con más sensores en camino. Cada sensor puede utilizar opcionalmente un modelo de ruido para inyectar propiedades de ruido gaussiano o personalizado.
- Ogre 2.1 está disponible a través de Gazebo Rendering, que proporciona acceso a las últimas técnicas de renderizado, incluyendo mapas de sombra mejorados, materiales PBR y un pipeline de renderizado más rápido.
- DART es el motor de física por defecto en Gazebo Physics, proporcionando un nivel de precisión que supera a los motores de juegos.
- La mayoría de las librerías de Gazebo ofrecen una interfaz de plugins que admiten el uso de código personalizado en tiempo de ejecución. En particular, Gazebo Rendering y Gazebo Physics proporcionan los ganchos necesarios para integrar motores de renderizado y físicas adicionales.

- Gazebo proporciona un mecanismo para cargar sistemas personalizados que pueden interactuar directamente con la simulación. Estos sistemas pueden utilizarse para inspeccionar y modificar la simulación sobre la marcha.
- Gazebo Transport utiliza ZeroMQ y Protobuf para una rápida y eficiente comunicación asíncrona inter/intra-proceso.
- La interfaz de herramientas de línea de comandos gz es utilizada por múltiples bibliotecas de Gazebo para proporcionar cómodas herramientas de línea de comandos, incluyendo introspección de temas, introspección de mensajes, lanzamiento y registro.
- Una interfaz gráfica basada en QtQuick se utiliza para visualizar la simulación, y proporciona un conjunto de plugins útiles para la visualización de temas, la entrega de mensajes, y el control del mundo de la simulación y las estadísticas.

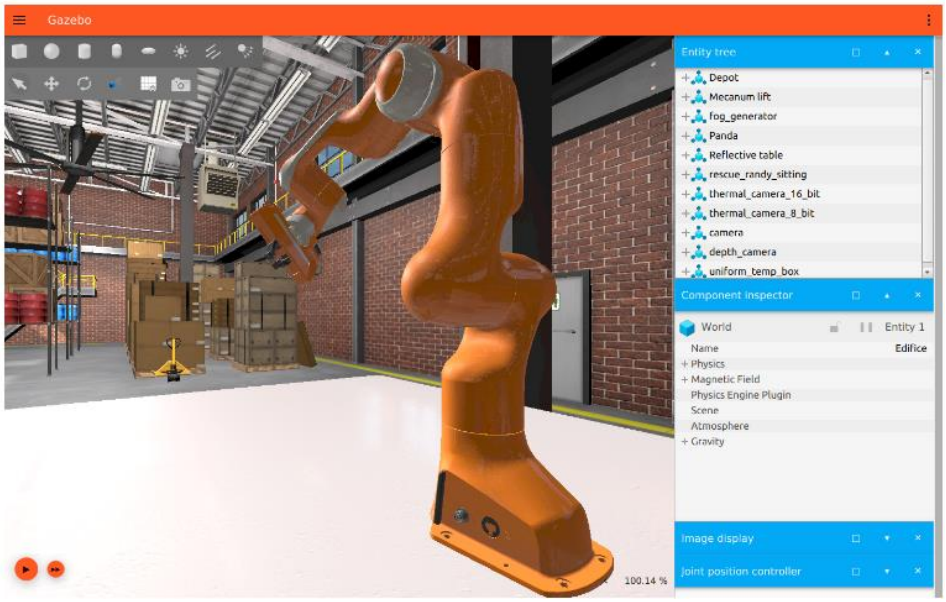



*Figura 14. Caso de uso del simulador Gazebo en un brazo mecánico industrial [50]*

### 2.5.2 AirSim

AirSim es un simulador elaborado por Microsoft para emular drones y automóviles. AirSim se basa en Unreal Engine, lo que aporta mayor realismo, capacidades y customización al generar entornos y escenas simuladas, como se ilustra en la Figura 15 [51]. Se trata de un software multiplataforma y de código abierto. Además, es compatible

con PX4, lo que permite trasladar el desarrollo probado en simulación, al mundo real [52]. Por estas razones, se ha elegido como el simulador idóneo para este trabajo.

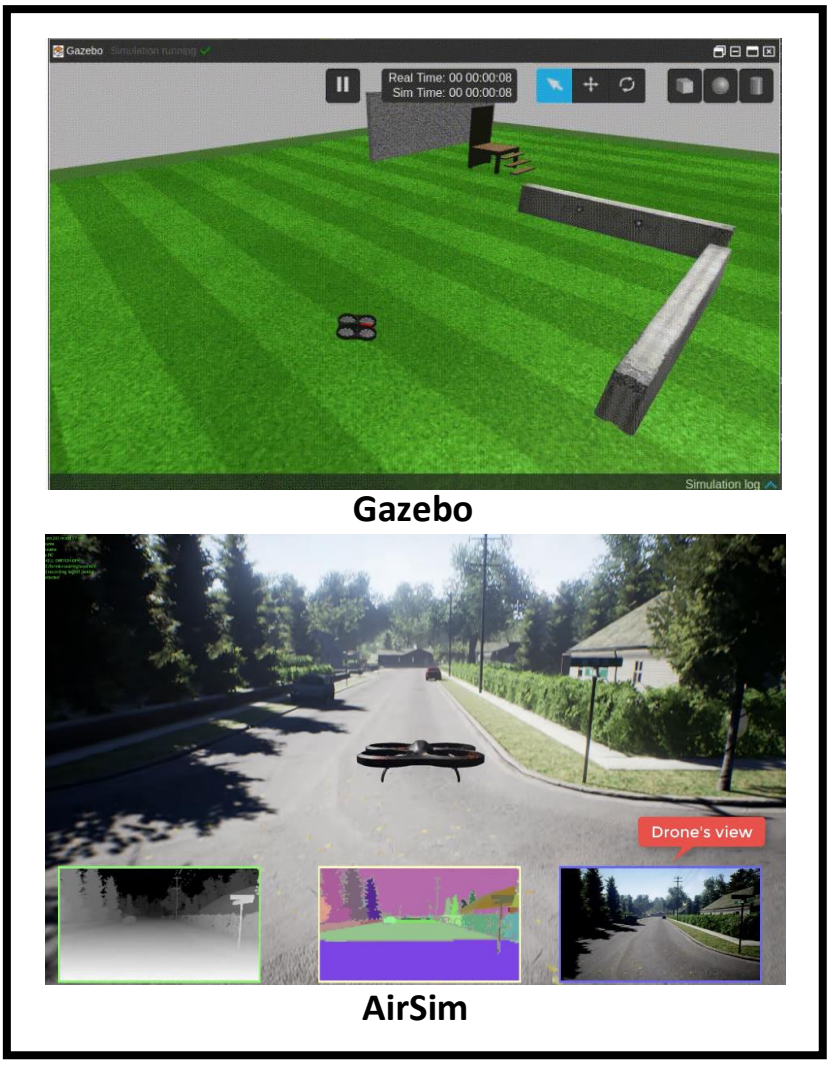



*Figura 15. Escena de un UAV en Gazebo y AirSim*

AirSim se centra específicamente en la simulación de automóviles y drones e incluye diferentes APIs que le otorgan las siguientes funcionalidades:

- Control de la climatología.
- Control del paso del tiempo.
- Variación del tipo de terreno, como bosques o prados.
- Tres tipos diferentes de cámaras.
- Integración con Unity.
- Sensor Lidar.
- Puede manejar varios vehículos en la misma simulación.

Estas APIs posibilitan la interacción con el entorno y los agentes intervinientes en el mismo. Son accesibles mediante diferentes lenguajes de programación: Python, Java, C++ y C#.

AirSim admite, simular uno o más vehículos, pero únicamente del mismo tipo, es decir, varios drones o varios coches, pero no la mezcla de ambos al mismo tiempo.

Además, admite la ejecución distribuida por lo que es ideal para crear aplicaciones que utilizan aprendizaje automático o "Deep Learning".

Además, encontramos ejemplos de arquitecturas en enjambre que han sido aplicadas en este simulador, como Scrimmage [53], que puede utilizarse para generar imágenes de cámara fotorrealistas y datos LIDAR/IMU realistas durante una simulación de múltiples drones usando dicha arquitectura, Figura 16.

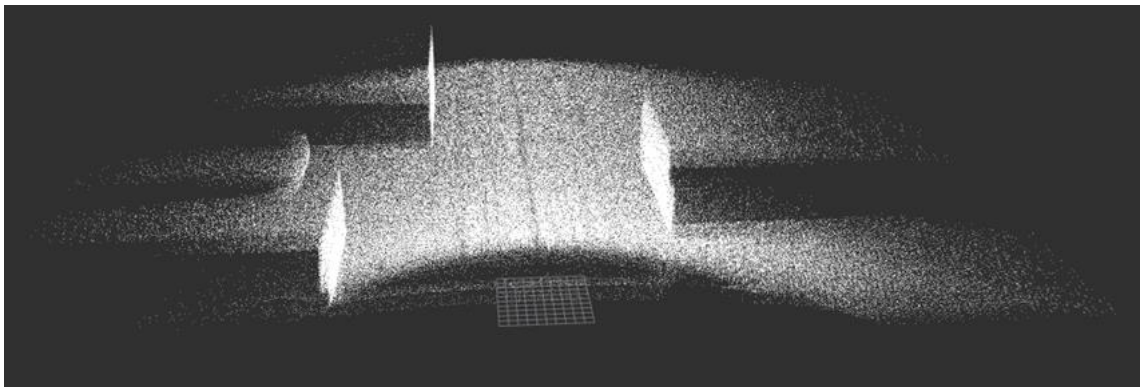


*Figura 16. Nube de puntos LiDAR realista obtenida a partir de múltiples drones en entorno simulado [53]*

#### 2.5.2.1 Sensores de AirSim

De forma predeterminada, estos son los sensores de drones en AirSim [54]:

- IMU: Sensores de velocidad, orientación y gravedad.
- Magnetómetro: Dispositivo que mide las señales magnéticas provenientes de los polos magnéticos de la Tierra.
- GPS: Es un sensor que permite determinar la posición de cualquier punto terrestre.
- Barómetro: Sensor de presión barométrica para determinar la altitud de la aeronave.
- Sensor de distancia: Permite saber la distancia a un objeto cuando se encuentra orientado hacia el mismo.

Así mismo podemos incorporar el sensor Lidar (Light Detection and Ranging): Es un sensor que determina distancias mediante la emisión de un láser hacia superficies u objetos sobre los que rebota volviendo a ser captado por el origen. La distancia se determina midiendo el tiempo que tarda dicho láser en volver al origen, Figura 17.

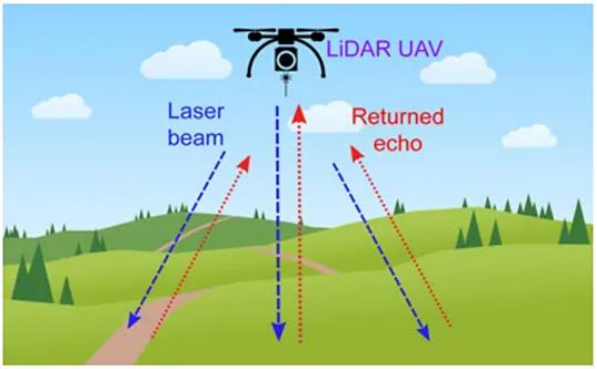



*Figura 17. Funcionamiento del sensor LiDAR [55]*

También, cuenta con diferentes tipos de cámaras además de las fotográficas. Como las que permiten generar imágenes térmicas infrarrojas (IR) [56], o la cámara de eventos, que mide los cambios en el brillo logarítmico y sólo informa de los "eventos", es decir, cuando se produce un cambio de brillo en la imagen [57].

## 2.6 Software In the Loop (SITL)

Para construir un sistema de simulación, se necesita conectar varios elementos a la vez. La ejecución de la simulación se basa principalmente en la simulación SITL (Software In The Loop) de PX4, quien implementa todos los algoritmos de evaluación, control y operación.

En el protocolo de comunicación MAVLink (Apartado 2.3) existe un conjunto específico de mensajes emulados, estos conforman la API de MAVLink [58]. Es similar a MAVLink, con la diferencia de que esta interactúa con la simulación, enviando los datos captados por los sensores durante la simulación a PX4 y devuelve valores para aplicarlos a la simulación de motores y actuadores del vehículo, como se muestrea en la Figura 18.

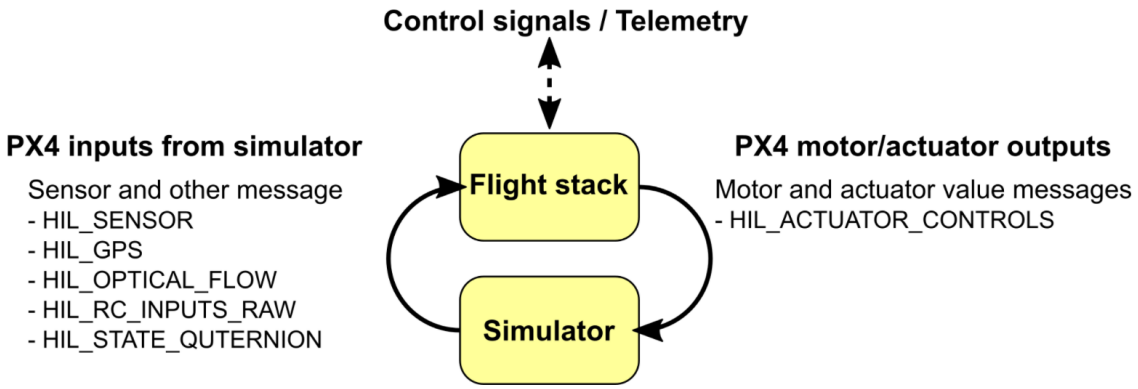



*Figura 18. PX4 en SITL [58]*

Además, al conectar todos los componentes al controlador PX4 durante la simulación, debe de existir un puerto UDP para poder comunicarse con cada uno de ellos, como se muestra en la Figura 19.

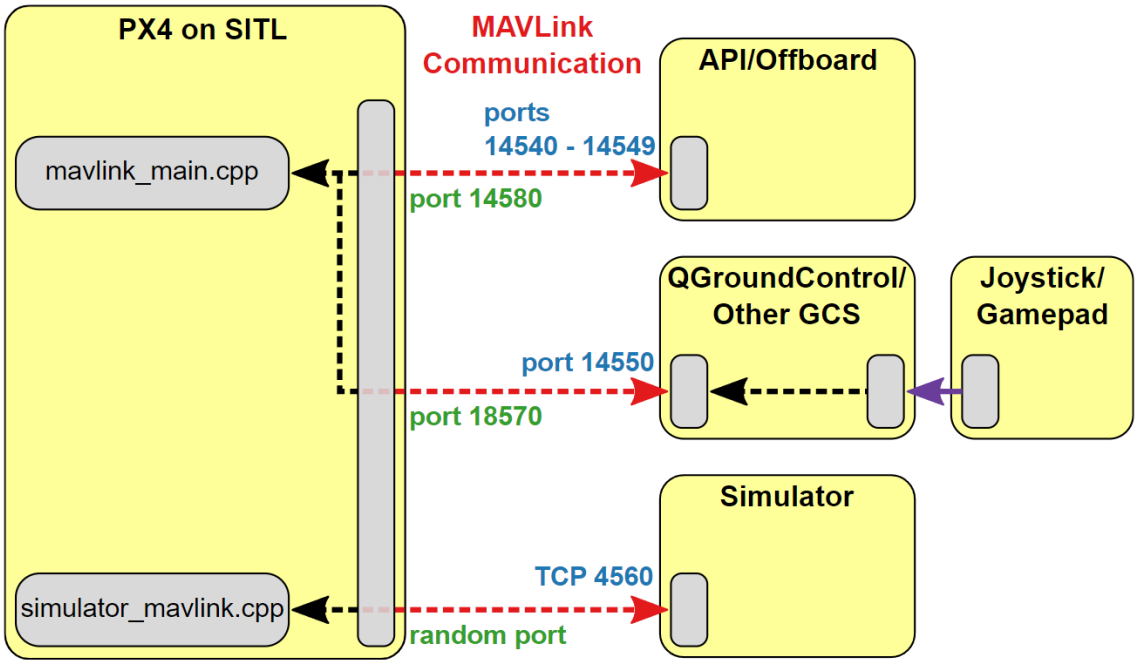



*Figura 19. Comunicación de PX4 con el resto de elementos de la simulación [58]*

## 2.7 Programación concurrente

Para poder simular un enjambre de drones desde un único ordenador es necesario usar programación concurrente, de manera que las acciones de cada dron se ejecuten simultáneamente con las de los demás. Las únicas opciones para dar soporte a esto son el uso de varios hilos o el uso de corrutinas.

### 2.7.1 Programación multi-hilo

Una de las particularidades básicas de los programas es que se ejecutan secuencialmente, es decir, línea por línea. Si un programa requiere un recurso, no hará nada mientras solicita el recurso y esperará una respuesta. En ciertos casos, este comportamiento es inapropiado. La forma estandarizada de manejar estas situaciones es mediante “threads" o hilos. Un hilo, es un fragmento de código que se puede ejecutar simultáneamente con otro/s fragmento/s de código. De esta forma, un programa puede ejecutar varios hilos, cada uno de los cuales realiza una operación de manera simultánea. Juntos, permiten que un programa realice múltiples tareas al mismo tiempo.

Pero, el uso de “threads” conlleva ciertas desventajas, como una mayor complejidad del código o problemas de concurrencia como condiciones de carrera, inanición o bloqueo mutuo, ver Figura 20.

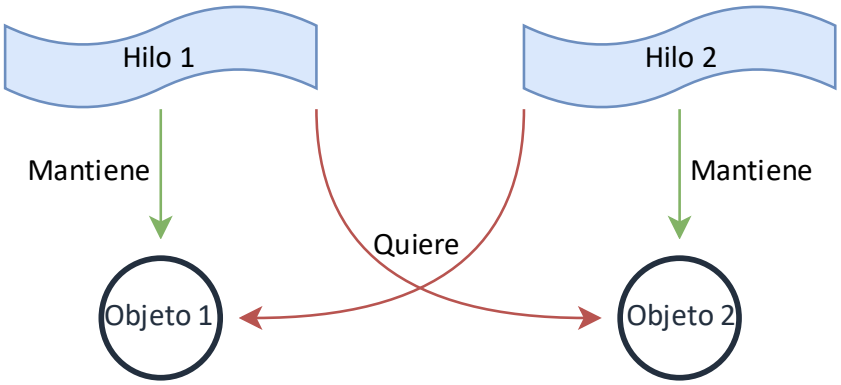



*Figura 20. Ejemplo de bloqueo mutuo*

A esto hay que añadir que cada núcleo de la CPU ejecuta un número determinado de hilos al mismo tiempo. Entonces, si superamos dicho limite, el núcleo de la CPU se encontrará cambiando de contexto continuamente, es decir, cambia el “thread” en ejecución constantemente simulando un comportamiento concurrente cuando en realidad no es así. Aunque este cambio de contexto es muy breve en las CPU actuales, supone un coste en tiempo a tener en cuenta, ilustrado en la Figura 21.

### 2.7.2 Programación asíncrona o corrutinas

La programación asíncrona es una emulación de hardware mediante software para la ejecución de hilos. Estos hilos y los cambios de contexto son dirigidos por el programa ejecutor.

En comparación con la programación "multi-thread", las corrutinas otorgan las siguientes ventajas [59]:

- Reduce la posibilidad de problemas de concurrencia: En el código asíncrono se puede saber exactamente cuándo el programa cambiará de tarea, lo que hace que este tipo de errores sean poco probables.
- Reduce los cambios de contexto: Como el cambio de tareas se realiza a nivel de software, no se cambia continuamente de contexto. Este hecho, combinado con que los cambios son controlados y menos frecuentes, conlleva una reducción del coste temporal que se pierde durante dichos cambios.

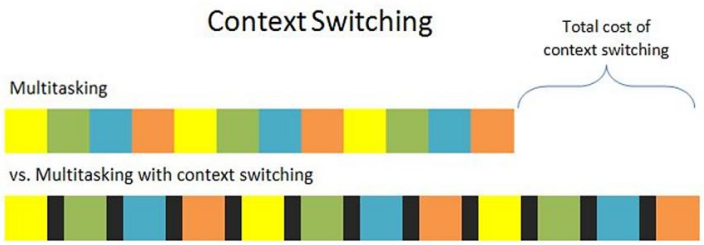



*Figura 21. Comparación de cambios de contexto mediante corrutinas [60]*

- Reduce la cantidad de memoria usada: Un hilo tiene su propia pila de variables y registros, es decir se le asigna una cantidad de memoria a cada hilo, ver Figura 22. Mientras que en la programación asíncrona como los hilos son simulados mediante software, únicamente se usa una sola pila de variables y registros, lo que reduce la cantidad de memoria requerida.

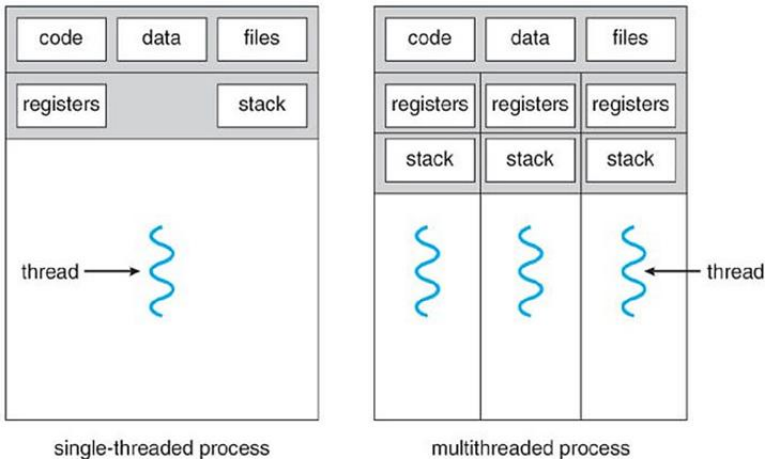



*Figura 22. Comparación de uso de memoria en programación multi-thread [61]*

Sin embargo, la programación asíncrona no aprovecha los hilos del procesador, lo que deriva en tareas exigentes computacionalmente, en una caída en el rendimiento general del programa.Pero, en programas poco exigentes computacionalmente y con una elevada tasa de operaciones de entrada y salida o E/S (como acceder a datos del teclado o la memoria), esta programación puede mejorar significativamente el rendimiento general. Para este trabajo, dado que los enjambres hacen un gran uso de la comunicación y por ende de los mensajes, hay muchas operaciones E/S lo que supone un incentivo para la adopción de este tipo de programación.

### 2.7.3 Corrutinas en Python

El uso de corrutinas en Python es cada vez más frecuente. Encontramos varias bibliotecas para la programación asíncrona, como Tornado [62] o Gevent8 [63], las más populares en versiones anteriores a la 3.4 de Python. A partir de entonces, aparece Asyncio, una nueva biblioteca de Python cuyo objetivo es hacer más sencillo y casi tan legible el código asíncrono como el secuencial.

La forma en la que trabaja Asyncio es mediante un bucle de eventos. En él podemos ir añadiendo diferentes tareas. Dichas tareas o fragmentos de código se ejecutan de manera secuencial hasta que se realiza una llamada bloqueante, como E/S o “dormir”. Cuando esto ocurre, el bucle ejecuta la siguiente tarea hasta que ocurra lo mismo que en la primera o finalice. Cuando el bucle de eventos vuelve a ejecutar la primera tarea, esta continua su

ejecución desde la llamada bloqueante que desencadeno el paso a la siguiente tarea, ver Figura 23.

A estas tareas se las conoce como corrutinas, es decir, subrutinas capaces de ser paradas y reanudadas. Para definirlas en Python utilizamos la siguiente sintaxis:

- Usamos "*async*" en la definición de la función asíncrona (únicamente para versiones iguales o superiores a Python 3.6): *async def function():*

La sintaxis para cambiar el contexto mediante corrutinas es la siguiente:

- Mediante "*await*" (únicamente para versiones iguales o superiores a Python 3.6): *resultado = await function()*

Esto significa que cuando se llama a una corrutina desde un "await", la función invocadora se pausa, cambiando el contexto para la ejecución de la corrutina invocada ( *function()* en el ejemplo de sintaxis anterior). Tras el fin de la corrutina, se envía una excepción "StopIteration" con el valor de retorno, el cual es asignado a la variable especificada, denominada "resultado" en el ejemplo de sintaxis anterior.

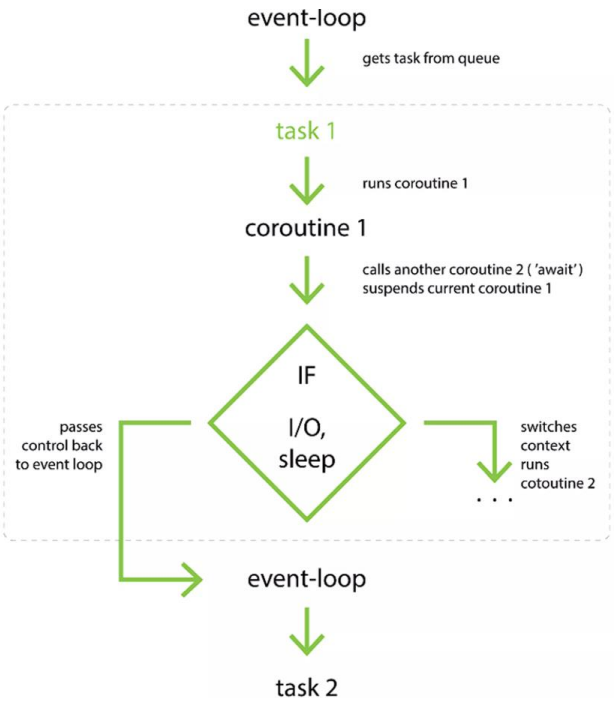



*Figura 23. Ciclo de eventos de Asyncio [64]*

De esta manera, podemos ejecutar fragmentos de código mientras se llevan a cabo llamadas bloqueantes en otro fragmento de código. Es decir, obtenemos una

programación concurrente a partir de una secuencial, habiendo modificado escasas líneas de código.

## 2.8 SDK (Software Development Kit)

En el software de código abierto encontramos un conjunto de herramientas llamado SDK (Software Development Kit) que permite crear aplicaciones para un sistema en particular. Entre las herramientas que las contienen encontramos la API (Application Programming Interface), es decir, una interfaz de programación de aplicaciones que trabaja con un determinado lenguaje informático. Los dos kits de desarrollo más importantes para sistemas no tripulados basados en el protocolo MAVLink son Dronekit [65] y MAVSDK [66].

Tanto Dronekit como MAVSDK están incluidos en el proyecto Dronecode y su finalidad es la misma, pero existen algunas diferencias entre ellos que pueden afectar a la necesidad de elegir entre ambos en función de nuestros requerimientos. La principal es su compatibilidad con el firmware que se está considerando en este proyecto, pues Dronekit es para Ardupilot y MAVSDK es para PX4. Aunque Ardupilot y PX4 funcionan con MAVLink, tienen diferentes formas de enviar e interpretar paquetes de datos, lo que genera problemas de compatibilidad. MAVSDK, originalmente conocido como Dronecode SDK, es mucho más nuevo y ofrece soporte para más lenguajes de programación. Aunque es software libre, está desarrollado por Auterion, una empresa privada relacionada con el proyecto Dronecode. MAVSDK se centra exclusivamente en PX4, software en el que funciona de manera óptima, ya que actualmente no es oficialmente compatible con Ardupilot.

Sus APIs le permiten crear aplicaciones para los entornos C++, iOS, Python y Android que forman parte del mismo ecosistema que ejecuta C++, un lenguaje muy potente.

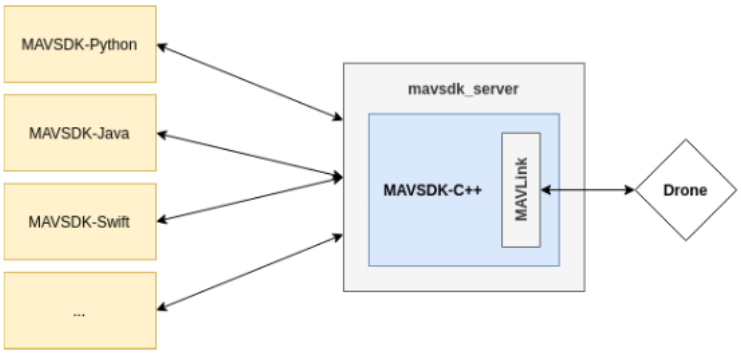



*Figura 24. Comunicación APIs y MAVSDK*

Además, tiene otras APIs betas para Java, JavaScript, Rust y CSharp. La principal ventaja sobre Dronekit es que es mucho más fácil integrar lenguajes de programación [67]. En Dronekit cada lenguaje tiene su propia implementación [68], mientras que en MAVSDK todos se integran automáticamente sobre el mismo core basado en C++, ver Figura 24. Esto permite la creación de entornos colaborativos multiplataforma en los que cada participante puede unirse fácilmente, independientemente del idioma utilizado. MAVSDK tiene licencia bajo los mismos términos BSD-3 que PX4 y es casi tan permisible como Dronekit, pero presenta ciertas limitaciones para realizar patentes.

# 3 PLANIFICACIÓN Y PRESUPUESTO

Antes de comenzar el desarrollo del sistema debemos plantear una metodología adecuada de trabajo, planificar el mismo y elaborar un presupuesto teniendo en cuenta los recursos invertidos en el proyecto.

## 3.1 Metodología de trabajo

Para validar el sistema se usarán requisitos, su identificación se ha basado en los objetivos que debe cumplir el sistema, vistos en el Apartado 1.3.

Además, se han elaborado una serie de casos de uso, que mediante una matriz de trazabilidad permitirán visualizar y determinar qué requisitos se han cumplido.

Para la validación del documento se ha empleado un sistema basado en versiones. Donde el nombre del documento es acompañado por un valor que indica la versión a la que corresponde (Ej. enjambre_drones_v0.0.docx). Cada modificación sobre el documento incrementa en una décima el valor de cada versión (Ej. enjambre_drones_v0.1.docx). Las modificaciones tras una corrección del tutor suponen el incremento en una unidad en el valor de la versión (Ej. enjambre_drones_v1.0.docx). Este sistema ha sido igualmente empleado para las versiones del código que eran actualizadas y guardadas en la nube gracias a la herramienta GitHub [69].

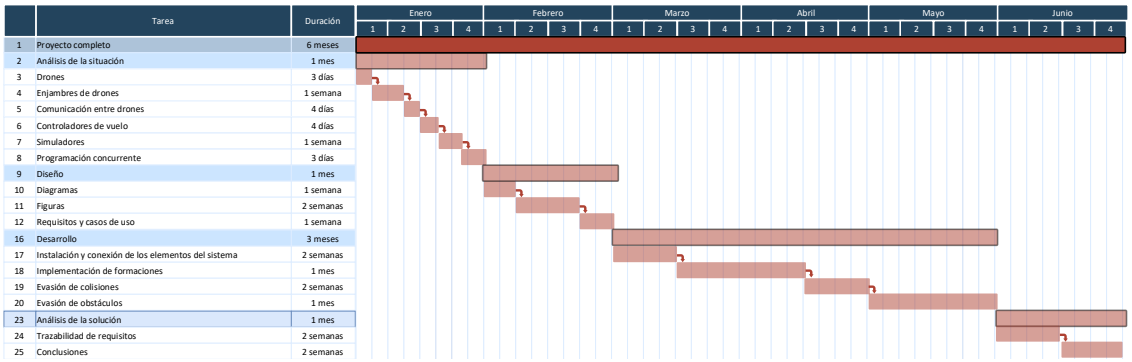
En cuanto a la recopilación de información, se ha usado la herramienta Zotero [70]. Esta permite guardar referencias y documentos durante la navegación, lo que facilita enormemente la búsqueda y la organización de la información recopilada. Para tener acceso a información más rica, se ha utilizado la VPN de la universidad [71] la cual da acceso gratuito a multitud de bancos de publicaciones científicas.

## 3.2 Planificación

Para alcanzar los objetivos planteados, es necesario planificar el trabajo a realizar, lo que permite organizar toda la carga de trabajo asociada a este proyecto. En este sentido, las etapas en las que se divide este trabajo son:

- Análisis de la situación: Esta es la primera fase del proyecto, donde el objetivo es investigar tanto las técnicas necesarias para lograr los objetivos establecidos como los conceptos que juegan un papel relevante. Es la base fundamental de todo el trabajo. Además, en esta etapa también se analizan las soluciones existentes al problema, con el objetivo de conocer el punto de partida de esta investigación. Su duración estimada es de 1 mes.
- Diseño: Después de comprender las técnicas requeridas para realizar la investigación, el siguiente paso es diseñar la solución para el problema investigado. Esta etapa incluye la visualización de los elementos software que se utilizarán y la creación de un método de trabajo a seguir para lograr los objetivos establecidos. Su duración estimada es de 1 mes.
- Desarrollo: Una vez que se ha conceptualizado la solución, el siguiente paso es desarrollar el código necesario para resolver el problema en cuestión. Se debe tener en cuenta que, para poder comenzar con el desarrollo, deben estar correctamente instalados todos los elementos del entorno propuesto. Su duración estimada es de 3 meses.
- Análisis de la solución: Esta etapa es de vital importancia, pues en ella se verificará la eficacia, validez y adecuación del sistema desarrollado en la etapa anterior, a través de pruebas empíricas y métricas diversas. Su duración estimada es de 1 mes.

El proyecto se inició al comienzo de 2022, abarcando hasta la semana de entrega para la segunda sesión de tribunales de TFG. La planificación, junto a sus fases y componentes principales, se ilustra en la Figura 25 mediante un diagrama Gantt.




| | Tarea | Duración |
|---|---|---|
| 1 | Proyecto completo | 6 meses |
| 2 | Análisis de la situación | 1 mes |
| 3 | Drones | 3 días |
| 4 | Enjambres de drones | 1 semana |
| 5 | Comunicación entre drones | 4 días |
| 6 | Controladores de vuelo | 4 días |
| 7 | Simuladores | 1 semana |
| 8 | Programación concurrente | 3 días |
| 9 | Diseño | 1 mes |
| 10 | Diagramas | 1 semana |
| 11 | Figuras | 2 semanas |
| 12 | Requisitos y casos de uso | 1 semana |
| 16 | Desarrollo | 3 meses |
| 17 | Instalación y conexión de los elementos del sistema | 2 semanas |
| 18 | Implementación de formaciones | 1 mes |
| 19 | Evasión de colisiones | 2 semanas |
| 20 | Evasión de obstáculos | 1 mes |
| 23 | Análisis de la solución | 1 mes |
| 24 | Trazabilidad de requisitos | 2 semanas |
| 25 | Conclusiones | 2 semanas |

*Figura 25. Diagrama Gantt del proyecto*

## 3.3 Presupuesto

A la hora de elaborar un presupuesto debemos tener en cuenta 3 tipos de costes: directos, indirectos y recursos humanos.

- Costes directos: Son aquellos derivados de bienes que han intervenido directamente en la realización del trabajo. Para implementar el sistema, se ha adquirido un equipo de gama media-alta, permitiendo que las simulaciones se realicen de manera eficaz, cómoda y sin problemas relacionados con la falta de rendimiento del equipo. El desglose de precios de sus componentes se muestra en la Tabla 4.

| Componentes | Precios |
|---|---|
| Corsair Carbide Spec-Delta RGB Cristal Templado USB 3.0 | 79,76 € |
| Asus TUF GAMING B660-PLUS WIFI D4 | 185,52 € |
| Intel Core i7-12700F 4.9 GHz | 370,48 € |
| Cooler Master Hyper 212 RGB Black Edition | 42,92 € |
| Team Group T-FORCE Dark Za 3200MHz PC4-25600 64GB 2x32GB CL16 | 319,00 € |
| MSI GeForce RTX 3080 VENTUS 3X PLUS OC LHR 10GB GDDR6 | 979,89 € |
| 2 Unidades: Kioxia Exceria G2 Unidad SSD 1TB NVMe M.2 2280 | 171,22 € |
| Seasonic G12 GC Series 750W 80 Plus Gold | 79,64 € |
| AOC 27G2AE/BK 27" LED IPS FullHD 144Hz FreeSync Premium | 179,99 € |
| **TOTAL** | **2.408,42 €** |

*Tabla 4. Costes directos*

- Costes indirectos: Son aquellos bienes que no han intervenido directamente en la realización de este trabajo. Como el agua, luz e internet consumidos durante el periodo de realización del mismo. Son difíciles de contabilizar, por lo que su coste se estima mediante un porcentaje atribuible al proyecto del coste total del recurso, como se muestra en la Tabla 5.

| Coste | Mensual | % atribuible al proyecto | Meses | TOTAL |
|---|---|---|---|---|
| Luz y agua | 60 € | 15 % | 6 | 54 € |
| Internet | 50 € | 15 % | 6 | 45 € |
| **TOTAL** | 110 € | 15 % | 6 | **99 €** |

*Tabla 5. Costes indirectos*

- Costes de personal: Es el coste que supone cada hora trabajada por la persona encargada de la realización del proyecto. Según Jobted, un Ingeniero Informático recién graduado, con menos de 3 años de experiencia laboral, percibe un salario medio de unos 20.450 € brutos al año [72]. Lo que se traduce en 10,49 € por hora trabajada, habiendo trabajado una media de 3 horas al día durante 6 meses, equivale a un total de 5.769,5 €.

De esta forma, el presupuesto total estimado para este proyecto se resume en la Tabla 6.

| Costes | TOTAL | + IVA |
|---|---|---|
| Directos | 2.408,42 € | 2.914,19 € |
| Indirectos | 99 € | 119,79 € |
| Personal | 5.769,5 € | 6.981,10 € |
| **TOTAL** | 8.276,92 € | **10.015,08 €** |

*Tabla 6. Presupuesto*

Por lo tanto, el presupuesto estimado es de **DIEZ MIL QUINCE COMA CERO OCHO EUROS.**

# 4 MARCO REGULADOR, ECONÓMICO Y SOCIAL

En cuanto a la regulación vigente sobre drones, varía dependiendo de cada país, aunque la mayoría comparten ciertos requisitos en común. A continuación, veremos algunas de las regulaciones más importantes [7]:

## 4.1 Legislación en la Unión Europea

En un ámbito civil, la regulación de la UE exige que los UAV que se deseen volar deben estar registrados en la EASA (European Union Aviation Safety Agency) a excepción de los drones nano (<250g) sin cámara. Para completar el registro, se recibirá un entrenamiento que certificará su permiso para volar drones en la Unión Europea, con una pegatina que deberá ser visible en los mismos. No se podrá sobrepasar los 120 metros de altitud, volar cerca de objetos, multitudes o zonas aeroportuarias. Tampoco se podrá tomar fotos o grabar sin el consentimiento de los que aparezcan en las mismas. Los UAVs se categorizan en Open, Specific o Certified en base a su amenaza, la cual es determinada teniendo en cuenta su peso y zona de vuelo, algunas de estas categorías pueden requerir autorizaciones adicionales.

Todo ello sin perjuicio de los requisitos esenciales previstos en el anexo IX del Reglamento (UE)2018/1139 [73]. En particular en lo que respecta a las características y funcionalidades específicas necesarias, para mitigar los riesgos relativos a la seguridad del vuelo, la privacidad y la protección de los datos personales, la seguridad o el medio ambiente, derivados del funcionamiento de estos UAVs [74].

### 4.1.1 Legislación en España

Aunque le es de aplicación la normativa europea, España también contempla cierta legislación específica para este ámbito. En el Real Decreto 1036/2017 [75] Artículo 29.3 aparece "*El piloto y los observadores no podrán realizar sus funciones respecto de más de una aeronave pilotada por control remoto (RPA) al mismo tiempo*" lo cual limita fuertemente la experimentación real de enjambres de drones. Entonces, para cumplirla, o contamos con tantas personas como drones requiera el experimento, o bien estos drones son completamente autónomos y descentralizados. Aun así, el gobierno continúa

invirtiendo en esta tecnología, como podemos ver en el contrato contraído entre el mismo y la empresa Escribano para el programa SENDISTAR (Sistema autónomo de enjambre distribuido y multiplataforma para misiones ISTAR) [76].

## 4.2 Legislación en Reino Unido

En Reino Unido no hace falta registrar ni obtener un certificado para volar un dron, aunque si estos tienen cámara, estarán sujetos a regulación adicional. Independientemente de su peso, no deben sobrepasar los 122 metros de altitud, ni volar a menos de 50 metros de objetos o personas. Además, tienen que volar manteniendo un mínimo de 150 metros de cualquier zona multitudinaria y de 5 kilómetros de zonas aeroportuarias o militares. No podrán llevar incorporados ninguna carga de pago que pueda causar ningún tipo de daño.

## 4.3 Legislación en Estados Unidos

Para volar un dron en Estados Unidos, este debe estar registrado en la FAA (Federal Aviation Authority) a excepción de los drones nano (<250g). Si se trata de un dron comercial, deberá además obtener un certificado y estarán sujetos a regulación adicional de la que hablaremos más adelante. Existen requisitos específicos dependiendo de cada estado, pero la mayoría coinciden en que no deben sobrepasar los 365 metros de altitud en espacios no regulados (espacio aéreo clase G) y en espacios regulados se debe obtener una autorización especial.

Si se trata de un dron personal cuya finalidad de vuelo es el entretenimiento, no se deberá perder el contacto visual con el UAV ni realizar otras actividades durante su manejo. No podrá superar los 25 Kg de masa, si se desea volar drones de mayor peso, se deberá poseer una autorización especial. Siempre se volarán en un espacio de clase G y lejos de cualquier personal de emergencias.

Si el ámbito es comercial, se necesita un permiso que es concedido por la FAA. Una vez obtenido, para poder volar un UAV, siempre se deberá: estar en contacto visual con el mismo, este no superará los 25Kg (incluida la carga de pago), hacerlo en un espacio de clase G, entre la puesta y la salida del sol, donde su velocidad y altitud máximas serán de

160Km/h y 120m respectivamente. Nunca volará cerca de personas, ni se contralará desde un automóvil en desplazamiento. Mediante un permiso especial, se podrán omitir estas medidas a excepción del peso máximo del dron y la clase G del espacio de vuelo.

## 4.4 Entorno económico

El primer ámbito en el que se instauran este tipo de tecnologías es el militar. Una vez se depuran y perfeccionan, de manera que se reducen considerablemente sus costes, aparecen en el ámbito civil [77]. Algunos ejemplos de este hecho son: Internet, GPS, las cámaras digitales, los drones, etc.

El impacto económico es visible en el ámbito militar, ya que el coste de un dron es sustancialmente más bajo que la mayoría de las armas y vehículos de combate utilizados [78]. Un ejemplo para ilustrar este hecho lo encontramos en una noticia reciente [79], donde se nos informa de que el ejército ruso ha diseñado un cañón láser, similar al mostrado en la Figura 26, que es capaz de derribar drones a 5 kilómetros de distancia con un tiempo de recarga de unos 5 segundos. Si quisiéramos combatir esta arma con drones, tras unos cálculos y suponiendo un entorno ideal (en el que no interviniesen más agentes, ni fuerzas de rozamiento e independiente de la climatología), obtenemos que, durante el recorrido hasta alcanzar el cañón, este, derribaría 20 drones viajando a una velocidad constante de 180 km/h. Si comparamos el precio de un dron con estas capacidades y la carga de pago suficiente para derribar el cañón una vez lo alcanzara, multiplicado por el número de drones que serían derribados, la cantidad que supone es mínima comparada con el coste de fabricación del cañón láser. Eso, suponiendo que los drones fuesen directos hacia él sin hacer nada por evitar ser alcanzados. Si ahora suponemos un enjambre que se coordina para elaborar una estrategia contra este tipo de cañones, cabe pensar que las pérdidas serán mucho menores.

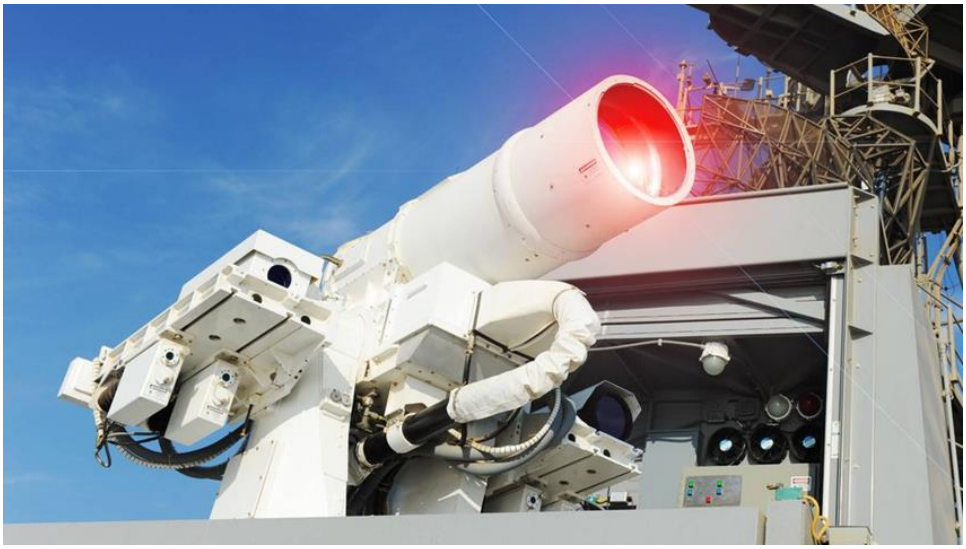


*Figura 26. Ejemplo de cañón láser norteamericano [79]*

El impacto económico en el ámbito civil es notable. Según estimaciones recientes, el mercado total de servicios basados en drones ascenderá a 63.000 millones de dólares en 2025 [80]. Un ejemplo claro lo encontramos incorporando esta tecnología en empresas de reparto, lo que supondría una reducción considerable de sus costes, acompañada de un crecimiento en su rendimiento.

Por otro lado, en una encuesta experimental realizada a un grupo de población académica [40], se concluye que los enjambres de drones son una agenda de futuro fundamental y que serán adoptados con el paso del tiempo.

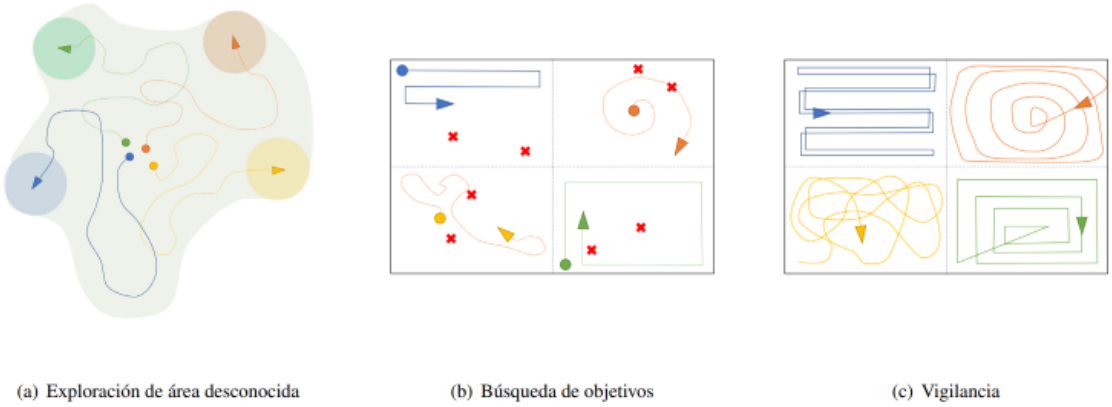
## 4.5 Entorno social

En cuanto al impacto social en el ámbito militar, se podría reducir considerablemente el número de bajas sufridas por el ejército que integrase esta tecnología: "*este arma desechable de bajo coste va a cambiar el campo de batalla para siempre 'incrementando la velocidad, alcance, visión y letalidad' de sus unidades*" [78]. De esta forma, sería posible desempeñar las estrategias militares exclusivamente mediante el enjambre, de manera que no sea necesario que los soldados arriesguen sus propias vidas.

En el ámbito civil, esta tecnología supone una mejora en la calidad de vida [81]. Tanto el precio como los tiempos de espera en envíos y recepción de comida u objetos, se verían reducidos, así como, se aumenta la facilidad y comodidad de realizarlos. Sin embargo,

conlleva un incremento de desempleo en este sector, que se transformará en nuevos puestos de mantenimiento, control y actualización de los enjambres de reparto. Otro problema adicional, es la existencia de una preocupación generalizada sobre el impacto en la privacidad si estos sistemas se extendiesen sobre zonas pobladas.

Aun así, los drones de bajo coste y sus enjambres proporcionan una plataforma prometedora para proyectos de investigación innovadores, y futuras aplicaciones comerciales que ayudarán a las personas en su trabajo y en su vida cotidiana [40]. Además, contar con esta tecnología en materias de protección civil y seguridad, mejora en gran medida las labores de exploración, visible en la Figura 27 (a), rescate, Figura 27 (b), y vigilancia, Figura 27 (c), entre otras.



(a) Exploración de área desconocida (b) Búsqueda de objetivos (c) Vigilancia

*Figura 27. Enjambres de drones en materias de protección civil [81]*

# 5 DESARROLLO DEL PROYECTO

En este capítulo se detallan todas las implementaciones y configuraciones realizadas para lograr la solución propuesta en el trabajo.

## 5.1 Arquitectura del sistema de simulación

En este apartado se explicará la configuración del simulador, el entorno, la creación de las conexiones necesarias, la adaptación entre diferentes programas y las diferentes clases, objetos y demás elementos esenciales implementadas en el lenguaje Python.

En la Figura 28 se muestra un diagrama que resume la comunicación entre los diferentes elementos que componen el sistema, explicados en el Apartado 2. Como se puede observar, contamos con dos sistemas operativos, el principal es Windows necesario para el funcionamiento de AirSim. Sin embargo, PX4 firmware es el "lenguaje" del sistema de vuelo autónomo con el que trabajamos en simulación y que es fácilmente trasladable a la realidad. Este firmware funciona en Linux, por lo que se debe usar WSL2 (Windows Subsystem for Linux) [82] para utilizarlo en Windows. WSL2 permite que la distribución de Linux se instale junto con la instalación de Windows para que ambos sistemas puedan funcionar juntos. Para que AirSim funcione en SITL (es decir, en lazo cerrado con todos los componentes emulados), lógicamente es necesario instalar y conectar todos los procesos.

Para que la ejecución funcione es necesario abrir los puertos que utilizan AirSim y PX4 para comunicarse entre sí, que por defecto están cerrados al ser distintas direcciones IP.

Hay que crear 4 reglas: 2 de entrada y 2 de salida (inbound y outbound). En nuestro caso los puertos de las reglas de entrada y salida serán 4650-4660 (TCP) y 14540-14590 (UDP).

Para poder utilizar varios PX4 (uno por cada dron), se ha utilizado un script en Python que invoca tantos PX4 como se le solicite en puertos adyacentes de los indicados anteriormente, que coincidirán con los de los drones simulados, determinados en la configuración de AirSim (settings.json).

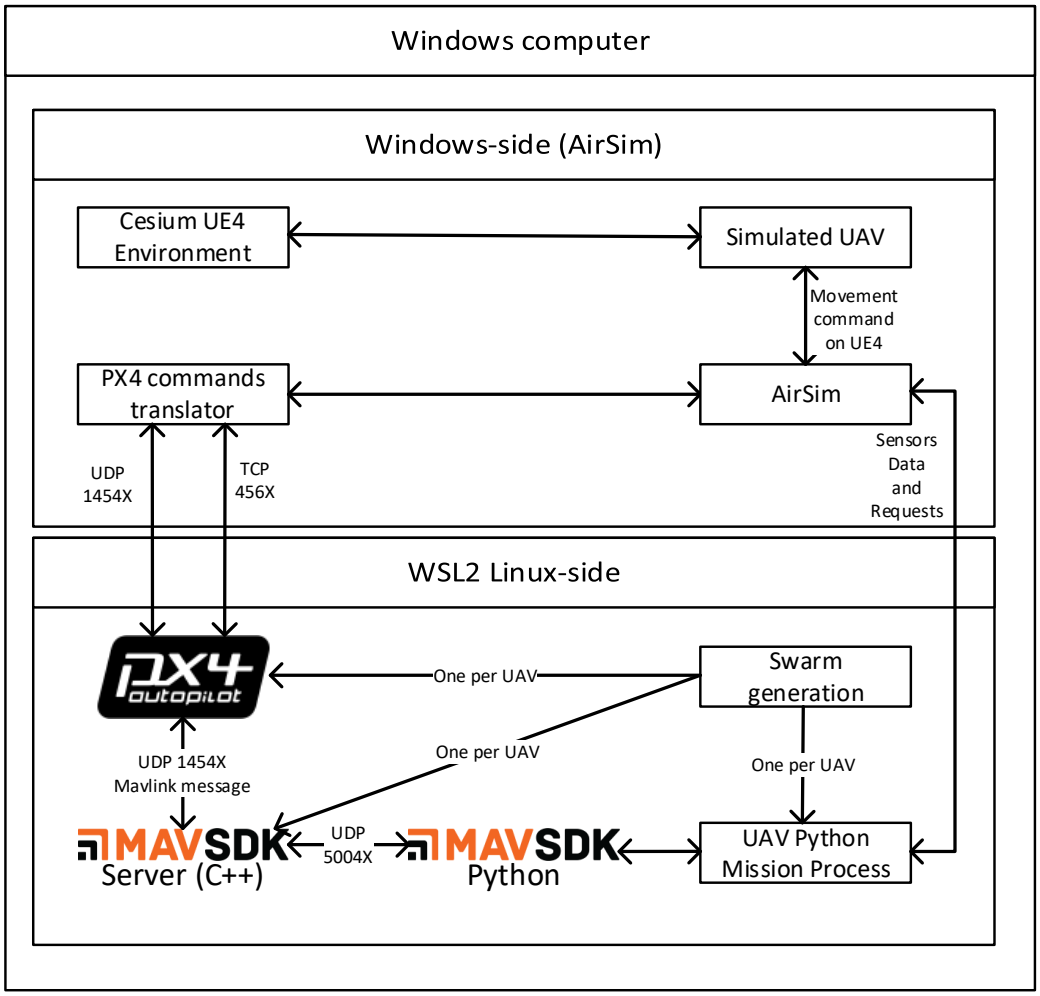



*Figura 28. Arquitectura del sistema de simulación*

Por otro lado, para vincular MAVSDK a PX4 es necesario que las IPs se encuentren. La forma de hacerlo es modificando dicho parámetro en el archivo "*PX4Autopilot/src/modules/MAVLink/MAVLink_main.h"*, de esta manera conseguiremos que envíe la IP en modo broadcast, facilitando que la encuentre el receptor.

Para arrancar la simulación, el entorno de Unreal debe estar funcionando, Apartado 5.2.1 del documento, y el dron preparado para conectarse a las API de Python.

Con esta sencilla línea en código Python *"cliente = airsim.VehicleClient(ip)"*, conseguimos conectar al simulador con el entorno y los sensores de los drones. Cada dron que se conecte poseerá su propia variable "cliente" que será sobre las que se solicitarán los datos de los sensores.

De forma similar nos concretaremos con MAVSDK-server: *"sistema = System(mavsdk_server_address, port)"*. Cada dron que se conecte poseerá su propia variable "sistema" que será sobre las que se solicitará las diferentes acciones de control que requiera la misión o proceso descentralizado, explicado en el Apartado **¡Error! No se encuentra el origen de la referencia.**, para que así se muestren en el entorno.

## 5.2 Entorno gráfico de simulación

### 5.2.1 Unreal Engine

AirSim Simulator utiliza Unreal Engine para generar el entorno de la simulación. Unreal, permite construir, escalar y controlar cada aspecto de la simulación. Es de código abierto y funciona mediante una sólida API en C++ y scripts basados en nodos para los no programadores. Además, ofrece imágenes de alta fidelidad y es gratuito [83].

Al instalar y ejecutar Unreal Engine por primera vez, encontramos un entorno con varios bloques apilados creando estructuras que recuerdan a edificios separados por calles, como se muestra en la Figura 29. Se trata de un mapa sencillo, que no consume demasiados recursos para su renderizado.

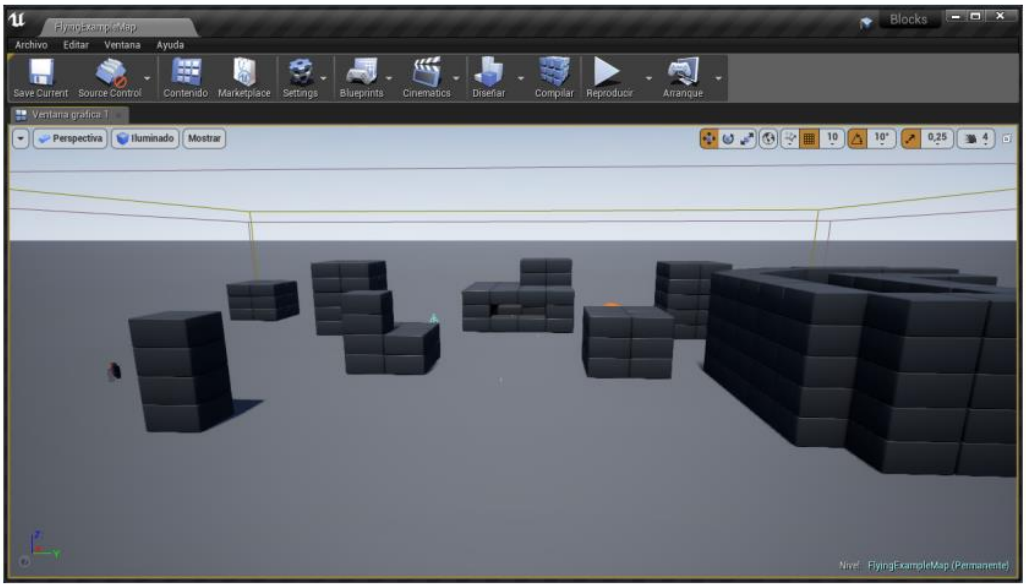



*Figura 29. Entorno por defecto de Unreal Engine*

### 5.2.2 Cesium

Para tratar de acercar la simulación a la realidad usaremos la herramienta Cesium para Unreal Engine. La cual nos ofrecerá una visualización más realista del entorno, como se puede apreciar en la Figura 30, y poder trabajar en latitudes y longitudes reales.


*Figura 30. Entornos geoespaciales en 3D de Cesium [84]*

Cesium es una plataforma creada sobre estándares y APIs de código abierto, útil crear entornos geoespaciales en 3D [85]. Cesium se combina con la potencia de renderizado de alta fidelidad de Unreal Engine, desbloqueando un ecosistema geoespacial 3D para los motores gráficos, es decir, un globo terráqueo de gran precisión WGS84 a escala real para Unreal Engine (UE). Integrado con este, los actores y componentes, blueprints y otras características de UE permiten un alto grado de interactividad, realismo físico y fotorrealismo.

Además, posibilita la visualización de fotogrametría masiva de alta resolución del mundo real y contenido 3D en tiempo de ejecución utilizando 3D Tiles, representadas en su esquema de funcionamiento en la Figura 31.

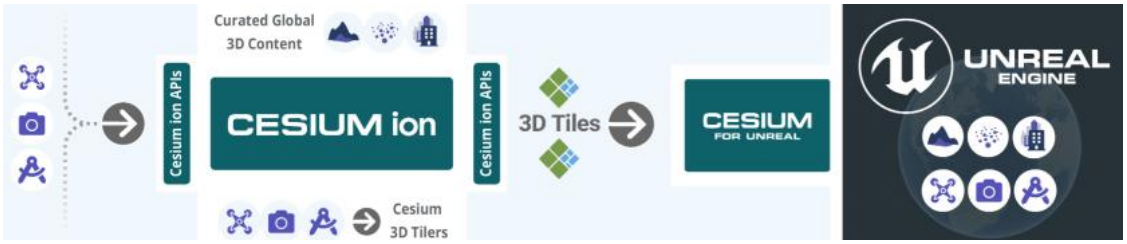



*Figura 31. Funcionamiento de la herramienta Cesium para Unreal Engine [84]*

## 5.3 Identificación de requisitos

Para identificar los requisitos necesarios para validar el sistema, se ha prestado especial atención en los objetivos determinados al principio del documento, Apartado 1.3. La plantilla utilizada para la especificación de cada requisito consta de diferentes atributos, todos ellos han sido descritos en la Tabla 7.

| | |
|---|---|
| **Identificador:** Código identificativo del requisito. | |
| **Nombre:** | Nombre del requisito. |
| **Prioridad:** Alta, Media o Baja. | |
| **Necesidad:** Indica si el requisito es obligatorio o es opcional. | |
| **Claridad:** Indica el nivel de interpretación unívoca. | **Verificabilidad:** Indica el nivel de satisfacción con respecto al sistema final. |
| **Estabilidad:** | Indica la durabilidad del requisito. |
| **Descripción:** | Breve descripción sobre en qué consiste el requisito. |

*Tabla 7. Plantilla de identificación de requisitos*

Para los requisitos funcionales, es decir, los que describen funciones que deben ser realizadas por el sistema, se propone una clasificación en áreas temáticas, en función de la operación que describen y que el software debe ser capaz de realizar. De esta manera, los podemos clasificar en: disponibilidad de la información, modificación de la información, descentralización, comunicación, adopción de formaciones y evasión de colisiones.

Por su parte, los requisitos no funcionales son requisitos que delimitan tanto el diseño como los estándares de calidad y nos aportan información sobre cómo debe operar el sistema para llevar a cabo las distintas funcionalidades. Se les ha asignado una clasificación en función de las cualidades que delimitan. En esta clasificación encontramos: disponibilidad, seguridad, restricciones, fiabilidad, rendimiento, mantenibilidad, portabilidad, interfaz y legislativos.

Para identificar cada requisito de manera rápida y cómoda se ha empleado la siguiente nomenclatura que permite identificar si es de tipo funcional (RF) o no funcional (RNF) seguido del número de requisito al que corresponde en el documento, para trazar un orden lógico.

Los diferentes requisitos encargados de validar el sistema se encuentran dentro del Anexo A REQUISITOS, debido a que se trata de información muy extensa.

## 5.4 Casos de uso

Para verificar que se cumplen los requisitos propuestos, se han elaborado una serie de casos de uso del sistema, descritos en la Tabla 8.

| Identificador | Descripción |
|---|---|
| **CU1** | El enjambre se desplaza en formación desde un punto a otro. Este último es un objetivo común, cuya ubicación ha sido comunicada por el emisor al resto de miembros del enjambre. |
| **CU2** | El enjambre se desplaza en formación desde un punto a otro. Durante el trayecto uno de sus miembros es inhabilitado, por lo que el resto del enjambre se da cuenta, reestructuran su formación y continúan con la misión. |
| **CU3** | El enjambre se desplaza en formación desde un punto a otro. Uno de los miembros del enjambre percibe un obstáculo en su trayectoria y lo esquiva desplazándose a uno de los puntos por los que discurre la trayectoria de su compañero. Quien previamente le había informado de que su camino estaba libre de obstáculos. |
| **CU4** | El enjambre se desplaza en formación desde un punto a otro. Todos los miembros del enjambre perciben un obstáculo en su camino, pero lo esquivan y continúan con la misión. |

*Tabla 8. Casos de uso*

La manera de relacionar los requisitos del sistema con los casos de uso propuestos, logrando la verificación de los mismos, será mediante la matriz de trazabilidad de requisitos.

### 5.4.1 Matriz de trazabilidad

Sus filas representan los diferentes casos de usos mediante su identificador único, así como, las columnas representan los diferentes requisitos a través de su identificador correspondiente. Diremos que se ha cumplido un requisito cuando sea necesario para cumplir un caso de uso, y este último, se haya realizado correctamente. Con esta idea en mente se ha elaborado la plantilla representada en la

Tabla 9. Si un requisito ha sido satisfecho durante un caso de uso aparecerá un tick verde, si no una cruz roja. Por último, si el requisito y el caso de uso no están relacionados aparecerá un espacio en blanco.

| **Requisitos** / **Casos de uso** | **Identificador** | **Identificador** |
|---|---|---|
| **Identificador** | ✓ (Cumplido) | X (No cumplido) |
| **Identificador** | (No relacionado) | |

*Tabla 9. Plantilla de la matriz de trazabilidad*

## 5.5 Proceso software descentralizado

A continuación, se explica en detalle la implementación final del proceso descentralizado que ejecuta cada dron. Para ayudar a la compresión y al orden de la explicación, nos apoyaremos en una serie de diagramas que representan funcionalidades lógicas del código Python utilizado por todos los drones durante su ejecución. Téngase en cuenta que todas las figuras del documento pueden ser ampliadas sin perder calidad para apreciar cualquier detalle que se desee.

Nótese que ciertos elementos de la misión que ejecuta el dron han sido resueltos de manera simple, puesto que la finalidad del mismo es demostrar las posibilidades que ofrece esta arquitectura, no realizar la comunicación o comportamientos del enjambre. La manera óptima de resolver estos problemas está presente en la literatura de dicha rama de investigación y se escapa del ámbito de este trabajo.

Dicho proceso es mostrado en la Figura 32. Esta ilustra la misión desarrollada, donde todos los drones reciben como objetivo desplazarse en enjambre hasta un punto P

determinado por una tupla de latitud y longitud. En caso de que se modifique el número de drones de la formación, ésta, en tiempo real, es capaz de recolocarse dando soporte a todos en una nueva formación.

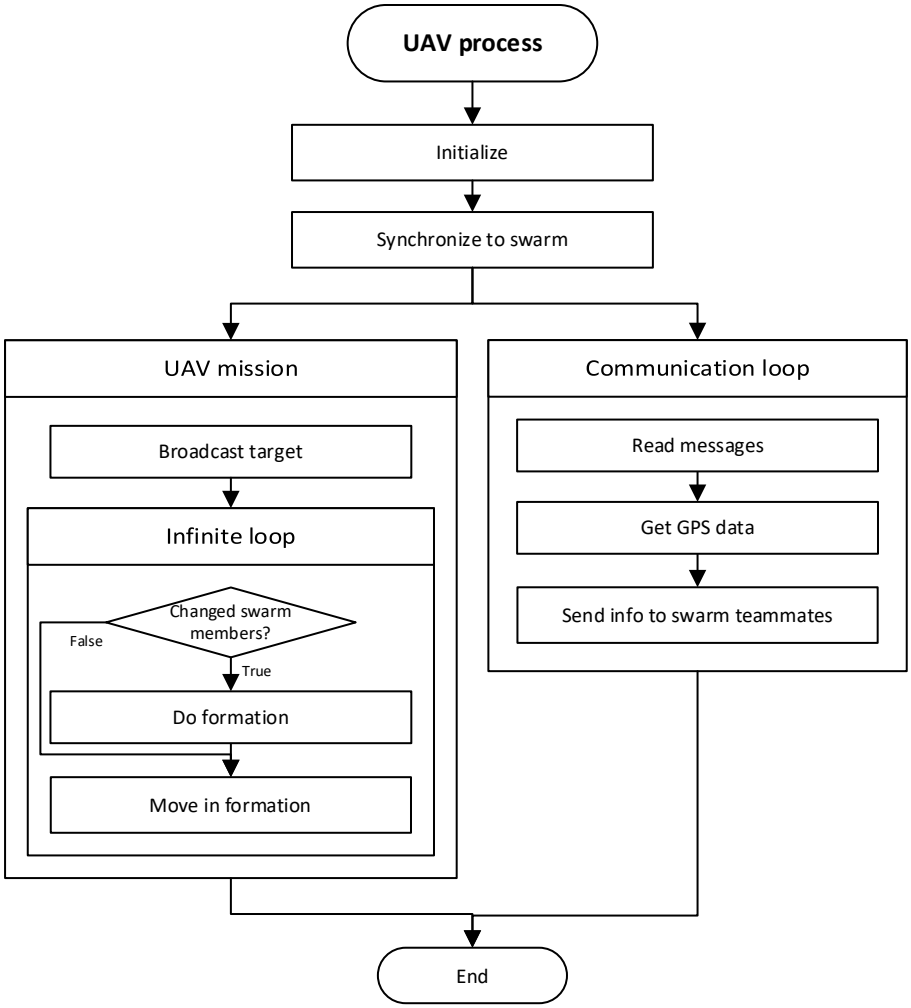



*Figura 32. Diagrama UAV process*

### 5.5.1 Initialize

El primer método se denomina “Initialize”. En él, cada dron se conecta vía Python con AirSim para recibir la información de los sensores y con su MAVSDK-server único, también vía Python, que le permite comunicarse con su controlador de vuelo PX4.

Una vez todo está conectado, el dron despega. Tras alcanzar la altura de despegue por defecto, se llama al método **¡Error! La autoreferencia al marcador no es válida.** para que todos los drones se sincronicen, ya que han podido experimentar retrasos durante la inicialización.

### 5.5.2 Synchronize to swarm

Al tratar con drones descentralizados, es importante asegurar que el enjambre puede coordinarse por lo que deberán de ejecutar e interpretar sus instrucciones en el tiempo adecuado. Para ello, el principal escollo es la sincronización temporal entre sus relojes internos.

En un primer momento se pensó en tratar de aproximar el retraso que experimentaban los drones con respecto al orden en el que ejecutaban una misma instrucción de código, determinando así el tiempo que debía de esperar cada dron en ese punto para sincronizarse con los demás. Intentar estimar dicho tiempo no solventó el problema.

Por lo que este problema se abordó desde la perspectiva de uno de los drones, es decir, pensemos que queremos llegar a un sitio al mismo tiempo que el resto de nuestros compañeros. ¿Y si en lugar de intentar predecir cuanto tiempo debo esperar a mis compañeros, no pactamos entre todos un tiempo que asegure que estaremos todos en ese sitio? Es similar a lo que hacemos las personas para reunirnos, concretamos la hora a la que aseguramos que estaremos en un determinado lugar. De forma análoga a las personas, los drones se envían un mensaje con el tiempo en el que creen que estarán en ese lugar del código, y el máximo de dichos tiempos será el elegido por todos. De esta forma, cada uno sabe que tiene que esperar exactamente la diferencia entre el tiempo en el que está listo y en el que lo estará el que llegue más tarde.

### 5.5.3 Communication loop

Una vez que se encuentran sincronizados, de forma paralela a la misión se crea esta tarea. Este método es llamado recursivamente hasta que termine la misión del dron. Su objetivo es que cada dron este informado en todo momento del estado del enjambre, pues en él se leen constantemente los mensajes que envían los demás drones con su posición y demás datos relevantes que más adelante se detallan. También es donde los drones obtienen su posición GPS y la envían a los demás, de forma que el resto también pueda saber su situación en todo momento, completando así este ciclo de información continua.

Para comunicar a los drones que conforman el enjambre, se ha optado por simular dicha comunicación mediante la escritura (enviar mensaje) y lectura (recibir mensaje) de un fichero JSON. Esta forma de comunicación es debida a trabajar en simulación, por lo que

no es recomendable sobrecargar el sistema con una comunicación más compleja de la imprescindible. En caso de implementar el sistema en drones reales, deberían implementarse métodos específicos de mensajería sobre la librería MAVSDK-Python que usa el protocolo MAVLink, siendo posible comunicar a los drones mediante el envío de este tipo de paquetes de datos. Este fichero JSON simula la comunicación entre distintos drones, con placas, procesos y memorias completamente independientes. Cada vez que un dron escribe en el fichero se registra mediante un timestamp. En cambio, cuando lee, se añade a una lista de destinatarios que han recibido el mensaje junto con el timestamp de cuando lo recibieron, como si de un ACK se tratase. En la Figura 33 se muestra un ejemplo de mensaje, en el que el dron 5 recibe un mensaje del 1 donde le comunica las coordenadas del objetivo.

### 5.5.4 Broadcast target

Al comenzar la misión todos los drones hacen notar mediante el método de comunicación su objetivo, de forma que el resto de los drones con el mismo objetivo puedan continuar comunicándose. En este caso, para diseñar la formación hacia el punto marcado.

```
|16:39:47,916|Drone 5|READ a message from Drone 1:
{
    "1": {
        "to": ["6","4","2",5,"3"],
        "msg": {
            "target": [
                40.546990393783204,
                -4.015595080273331,
                132.09912457010387]
        },
        "read by": ["3","6","4",2,"5"],
        "time_ns": 1655908787137913194
    }
}
```

*Figura 33. Broadcast de las coordenadas del objetivo*

### 5.5.5 Do formation

Una vez que todos informan su objetivo, comienzan a generar la formación. Nótese que se realiza de forma dinámica, puesto que ningún dron tiene prioridad respecto a los demás.

Se han elaborado diferentes formaciones clasificadas mediante un número que, a partir del valor 2, hace referencia a lados del polígono que formarán:

0. Sin formación.
1. Drones en fila, es decir, uno a continuación del otro.
2. Drones en paralelo, es decir, uno al lado del otro.
3. Triángulo.
4. Cuadrado.
5. Pentágono.
6. Hexágono.

El valor de la formación a adoptar por defecto será el número de drones que participen en la misión. Para saber la posición que debe adoptar cada dron en la formación, se inscriben en una lista que posteriormente envían a los demás, el orden que ocupan en la lista es el orden en el que han llegado a esa instrucción en el código. El primero de la lista, permanecerá en su posición y los demás realizarán la formación en base a este.

Para lograr este objetivo varios bloques son necesarios, tal y como se ilustra en la Figura 34

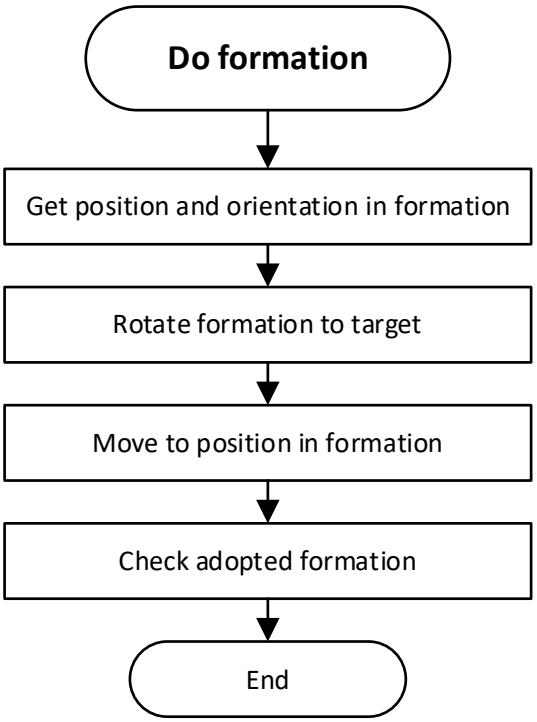



*Figura 34. Diagrama Do formation*

#### *5.5.5.1* Get position and orientation in formation

Este método recibe como argumento la ubicación del primer dron de la lista, y devuelve la posición a la que debe desplazarse el dron invocador con respecto a este para adoptar la formación pertinente. La forma de calcularla sencilla, en la Figura 35 se muestra cómo, dado uno de los vértices, se han calculado el resto para un triángulo ABC en un plano paralelo al XY del espacio cartesiano de 3 dimensiones. Siendo *distancia* un parámetro que podemos modificar según queramos que se alejen entre si los drones en las formaciones. Este parámetro se expresa en la Tabla 10, en nuestro caso su valor será de 2,5 metros.

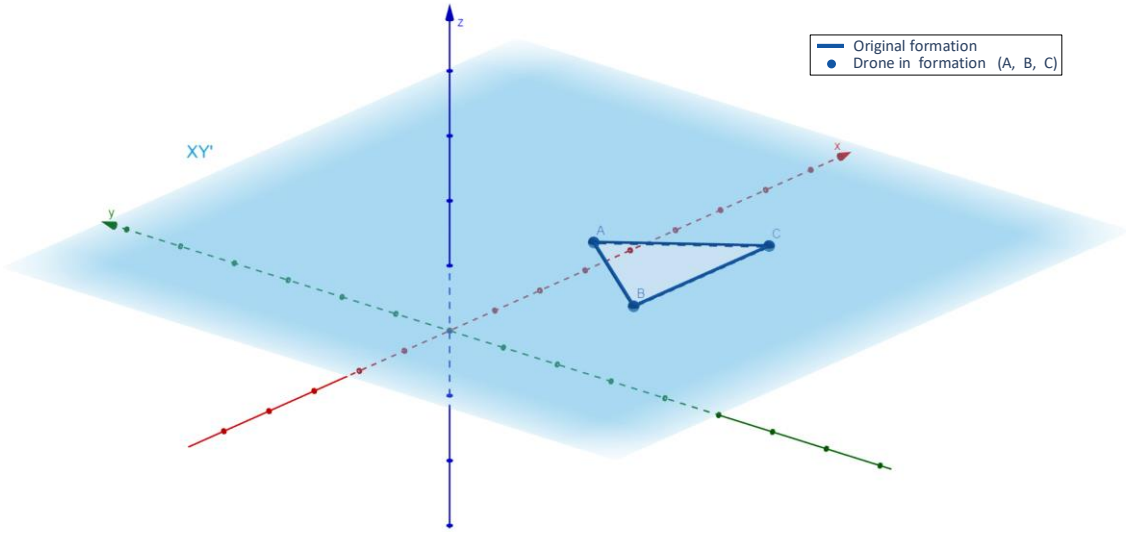



*Figura 35. Formación en un plano paralelo al XY*

$$A = (x, y, z)$$
$$B = (x - distancia, y - distancia, z)$$
$$C = (x + distancia, y - distancia, z)$$

*Tabla 10. Ecuaciones Figura 35*

Si quisiéramos realizar formaciones en un plano paralelo al XZ, Figura 36, bastaría con aplicar el algoritmo explicado anteriormente, pero sustituyendo los cálculos de coordenadas que hacen referencia al eje Y por el eje Z, representados en la Tabla 11.

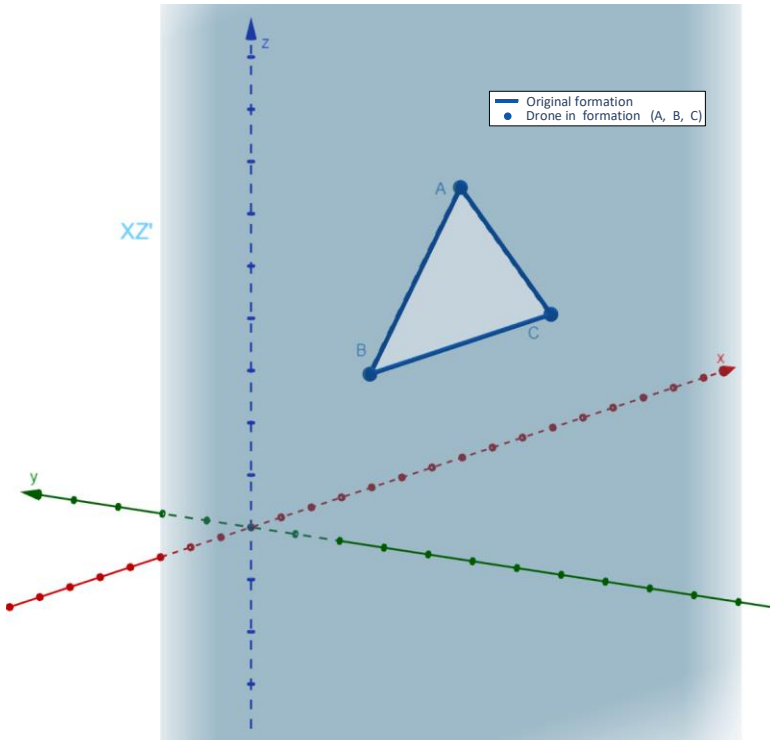



*Figura 36. Formación en un plano paralelo al XZ*

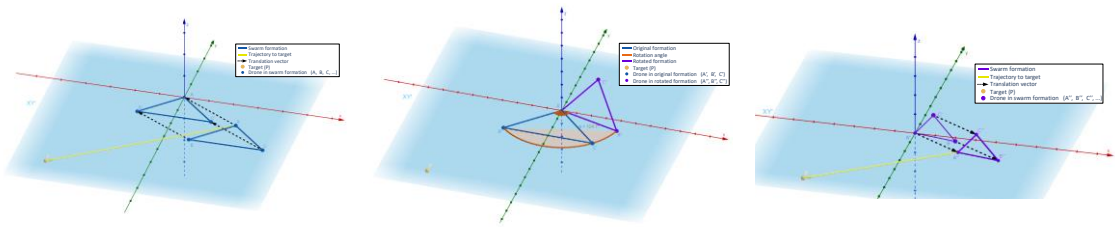
$$A = (x, y, z)$$
$$B = (x - distancia, y, z - distancia)$$
$$C = (x + distancia, y, z - distancia)$$

*Tabla 11. Ecuaciones Figura 36*

#### *5.5.5.2* Rotate formation to target

Ahora que cada dron sabe la posición que debe ocupar en la formación, aún existe otro inconveniente. Como la formación viaja hacia el objetivo, esta debería de "apuntar" hacia el mismo. Para conseguirlo, rotaremos la posición de cada dron con respecto al primer dron de la lista exactamente el ángulo que guarde este último respecto al objetivo, los cálculos necesarios se muestran en la

Tabla 12. Nos serviremos de la Figura 37 para visualizar este proceso.



(a) (b) (c)

*Figura 37. Rotación axial de la formación*

$$\overrightarrow{AP} = (APx, APy)$$
$$\alpha = -\left(\frac{180}{\pi} \cdot arctg\left(\frac{APx}{APy}\right)\right)$$

| | | |
|---|---|---|
| $A = (ax, ay, z)$ | $B = (bx, by, z)$ | $C = (cx, cy, z)$ |
| $A' = (0, 0, z)$ | $B' = (bx - ax, by - ay, z)$ | $C' = (cx - ax, cy - ay, z)$ |
| $A'' = (0, 0, z)$ | $B'' = \begin{pmatrix} bx' \cdot \cos(\alpha) - by' \cdot sen(\alpha), \\ bx' \cdot \cos(\alpha) + by' \cdot sen(\alpha), \\ z \end{pmatrix}$ | $C'' = \begin{pmatrix} cx' \cdot \cos(\alpha) - cy' \cdot sen(\alpha), \\ cx' \cdot \cos(\alpha) + cy' \cdot sen(\alpha), \\ z \end{pmatrix}$ |
| $A''' = (ax, ay, z)$ | $B''' = (bx'' + ax, by'' + ay, z)$ | $C''' = (cx'' + ax, cy'' + ay, z)$ |

*Tabla 12. Ecuaciones de la Figura 37*

Los cálculos necesarios para realiza dicha rotación fueron obtenidos de manera intuitiva y sencilla gracias a las nociones y habilidades obtenidas en la asignatura Informática Gráfica [86]. El principal problema a la hora de realizar una rotación es que si el punto

sobre el que se rota no es el origen de coordenadas el resultado se ve alterado. Por lo que primero debemos realizar una traslación del punto de rotación, junto con el resto de los puntos involucrados, hasta el origen de coordenadas Figura 37 (a). Una vez se encuentre allí, realizar la rotación Figura 37 (b). Tras realizarla, debemos de trasladar de nuevo los puntos a la posición donde queríamos obtener el resultado de la rotación, o lo que es lo mismo, deshacer la traslación inicial Figura 37 (c).

#### *5.5.5.3* Move to position in formation

Cuando cada dron obtiene la posición que ocupará en la formación y su orientación, debe dirigirse hacia allí sin colisionar con ninguno de sus compañeros. La forma de conseguirlo es que cada dron comparta con los demás la trayectoria que va a seguir durante su viaje. Para lograr este objetivo varios bloques son necesarios, tal y como se ilustra en la Figura 38.

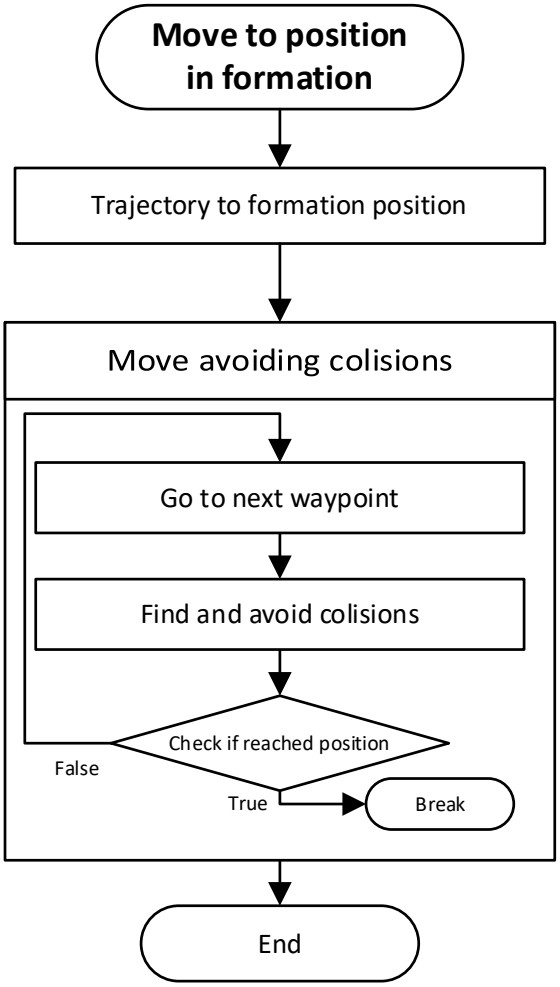



*Figura 38. Diagrama Move to position in formation*

#### *5.5.5.4* Trajectory to formation position

Este bloque se encarga de devolver una serie de puntos (waypoints) que conforman la trayectoria inicial a desarrollar por el dron para dirigirse desde su posición actual hasta la posición que debe ocupar en la formación. Estos puntos llevan asociado un tiempo, que es en el que se estima que se encontrará en ese punto.

La forma de realizar esta estimación es mediante el uso de la ecuación vectorial de la recta en el espacio cartesiano de 3 dimensiones junto con la aplicación de la ecuación de movimiento rectilíneo uniforme (MRU). Mediante esta última podemos obtener el tiempo en el que el dron viaja de un punto a otro del espacio con una velocidad constante. Si formamos la recta que une el punto de origen y el de destino, podemos obtener tantos waypoints como queramos, la forma de hacerlo es dividiendo el tiempo, obtenido mediante la ecuación de MRU ($time_{Total}$), entre el tiempo en el que queremos que el dron pase de un waypoint a otro (tick), lo que nos dará el número de waypoints durante la

trayectoria. En nuestra ecuación de la recta, con el valor de lambda igual a 1 se obtiene el punto final en el que se encontrará el dron y con el valor de lambda igual a 0, como es obvio, se obtendrá el punto de origen del dron, por lo que podemos hacer una división proporcional de lambdas entre 0 y 1 que representan los waypoints en los que se encontrará el dron en cada tick. En la Tabla 14 se representan estos cálculos, junto con la Figura 44 que nos servirá para ilustrar esta idea.

#### *5.5.5.5* Move avoiding colisions

Como cada dron envía a los demás su trayectoria, todos saben los puntos por los que va a viajar cada uno y en qué tiempo aproximado se van a encontrar en dichos puntos. Por tanto, en este bloque, cada uno revisa su trayectoria, y, si observa cierta coincidencia en tiempo y espacio con otro compañero, modificará su trayectoria para evitar dicha coincidencia. Además, enviará la trayectoria modificada para que el resto la tenga en cuenta.

Se debe asegurar que no se realizan modificaciones simultáneas, ya que esto podría generar que, si dos drones observan que su trayectoria coincide, la modifiquen los dos simultáneamente, volviendo a coincidir y así sucesivamente. Esta comprobación de coincidencias en las trayectorias se realizará continuamente hasta que cada dron llegue a su destino, finalizando así el bloque de la Figura 38. El resultado de esta implementación puede visualizarse en la Figura 39, donde se muestra un ejemplo para la adopción de una formación hexagonal, que son los círculos rojos. Las líneas y puntos de colores son la trayectoria que sigue cada dron hasta su posición. Podemos ver que el dron amarillo evita al rojo recalculando su ruta unos metros por encima. También encontramos casos de más de una coincidencia, como el morado y azul, ambos detectan una colisión con el rojo, pero entonces coinciden entre ellos por lo que nuevamente recalculan su trayectoria. Aunque se ven trayectorias abruptas, en la simulación aparecen suavizadas.

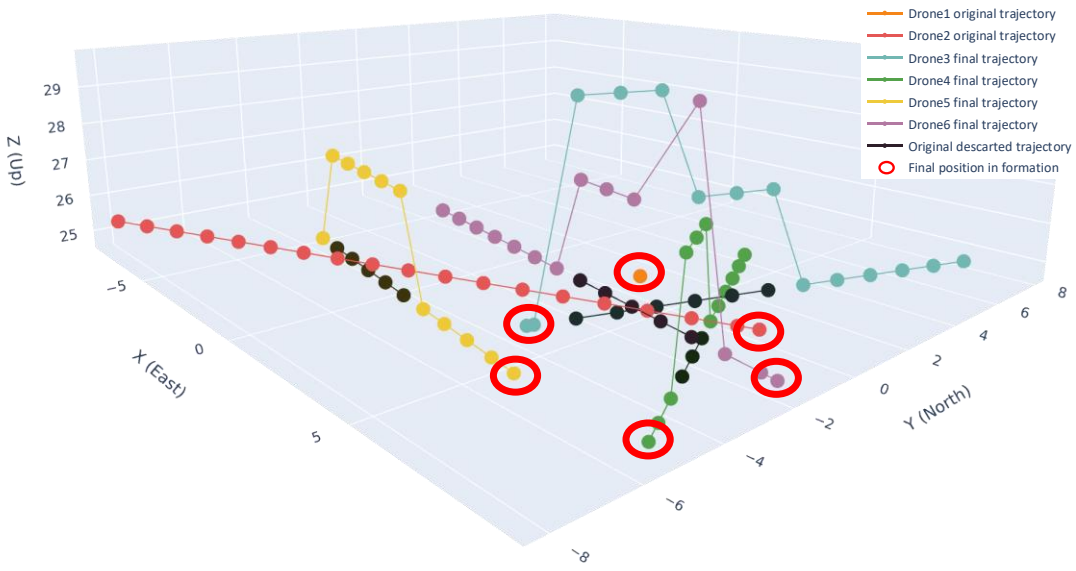



*Figura 39. Evasión de colisiones entre miembros del enjambre*

#### *5.5.5.6* Check adopted formation

Finalmente se comprueba si el dron ha llegado a su posición destino dentro de la formación. Simplemente se comprueba que la distancia del dron a dicho punto precalculado sea menor a 10 centímetros. Por otro lado, hasta que todos los drones de la formación comuniquen que cumplen con su posición en la formación mediante un mensaje, la formación no comienza la misión. Finalizando así el bloque de la Figura 34.

### 5.5.6 Move in formation

Una vez que los drones se encuentran en formación pueden comenzar la misión, en este caso de desplazamiento conjunto. Esta tarea se realiza en un bucle infinito hasta llegar al lugar deseado, tal y como se ilustró en Figura 32.

Para lograr este objetivo varios bloques son necesarios, tal y como se ilustra en la Figura 40.

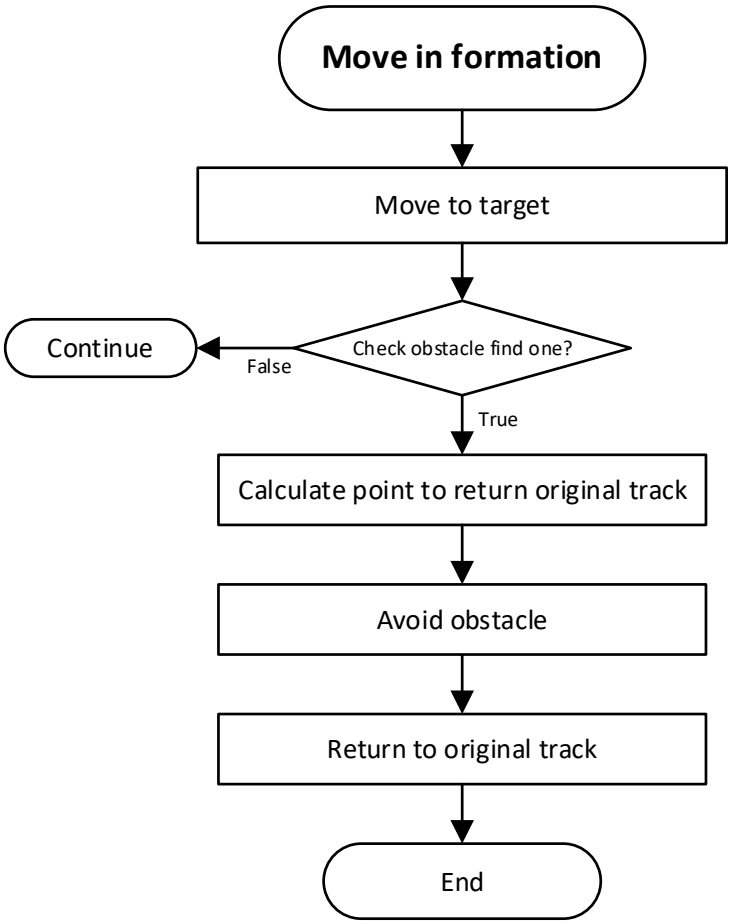



*Figura 40. Diagrama Move in formation*

Además de conocer la ubicación del objetivo, cada dron debe saber la posición en la que finalizará su viaje respecto a este, y así mantener la formación en su destino. Para obtener dicha posición, simplemente se modifica el método Get position and orientation in formation para que calcule la posición respecto al objetivo. También, se añade un parámetro adicional, *sobrevuelo*, que indica la altura la que se desea que el enjambre sobrevuele al objetivo una vez lo alcance, La Figura 41 ilustra esta idea.

Al igual que antes, también se debe aplicar una rotación axial Figura 37. De esta forma cada dron conoce su posición y orientación respecto al objetivo, consiguiendo así que la formación llegue orientada en la dirección en la que viajaba hacia este.

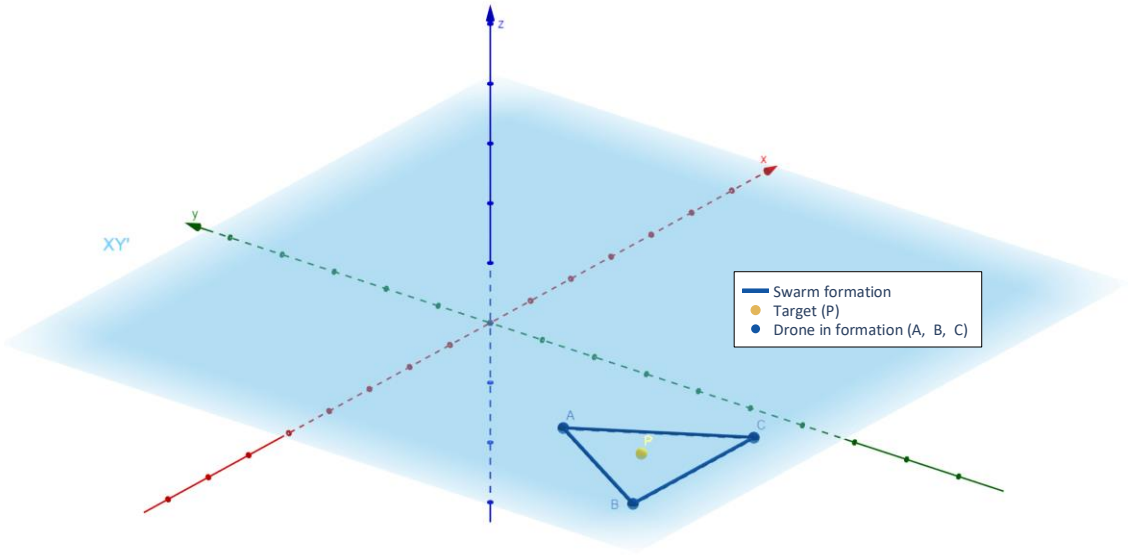



*Figura 41. Formación en el objetivo*

$$P = (x, y, z)$$

$$A = \left(x, y + \frac{distancia}{2}, z + sobrevuelo\right)$$

$$B = \left(x + distancia, y - \frac{distancia}{2}, z + sobrevuelo\right)$$

$$C = \left(x - distancia, y - \frac{distancia}{2}, z + sobrevuelo\right)$$

*Tabla 13. Ecuaciones Figura 41*

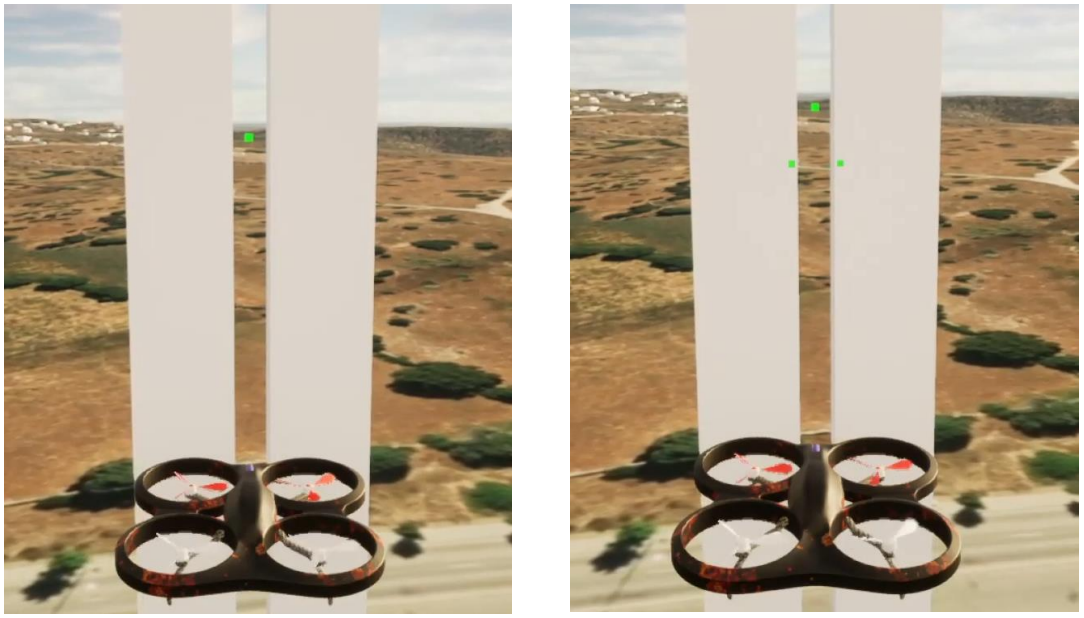
#### *5.5.6.1* Move to target

Ahora que cada dron sabe su punto exacto de destino, deben sincronizarse de nuevo, de forma que comiencen su viaje al mismo tiempo, haciendo que la formación se deforme lo menos posible durante el trayecto.

Este bloque se enfrenta a un reto principal, ¿qué ocurre si en su camino encuentran un obstáculo? Deberían de poder esquivarlo, retomar su posición en la formación y continuar su viaje.

Para llegar al objetivo evitando los obstáculos que los drones encuentren durante el camino y evitando en todo lo posible la deformación de la formación adoptada, hacemos uso de sensores de distancia. Tres de estos sensores son colocados en la parte frontal de cada dron, ya que estos siempre circulan de frente tal y como se ha diseñado esta misión. Estos están colocados en puntos de forma que abarquen la máxima superficie frontal posible del dron, es decir, el centro y los extremos de la misma. Esto nos permiten

garantizar que el dron no se encuentre situaciones donde el sensor pueda no detectar obstáculo y chocar contra ellos. La Figura 42 ilustra esta idea, donde se representa en verde claro el puntero del sensor de distancia.



(a) (b)

*Figura 42. Ejemplo detección de obstáculos con uno (a) y tres (b) sensores de distancia frontal*

#### *5.5.6.2* Check obstacle

Solo los drones que no tengan un compañero delante en la formación comprobarán en todo momento la distancia captada por sus sensores de distancia. Ya que, de otra forma, los anteriores detectarían como obstáculo a sus mismos compañeros de formación.

Si la distancia que captan es en algún caso no máxima, significa que han detectado un objeto en frente de ellos.

#### *5.5.6.3* Calculate point to return original track

Si un dron detecta un obstáculo, calcula el punto en el que considera que habrá esquivado dicho obstáculo y donde retomará su trayectoria original. La forma de calcularlo es sencilla, como se sigue una trayectoria recta, podemos calcular un waypoint futuro en dicha recta. Este waypoint se encontrará a una determinada distancia de seguridad, un valor suficiente con el que consideremos que hemos evitado el obstáculo. La Figura 44 ilustra esta idea mediante el punto C'', el cual se encuentra a dicha distancia de seguridad respecto del obstáculo. En concreto se ha establecido a 10 metros este valor, funcionando

correctamente para obstáculos de una profundidad menor al mismo. Esta implementación es más que suficiente para cualquier obstáculo común.

#### *5.5.6.4* Avoid obstacle

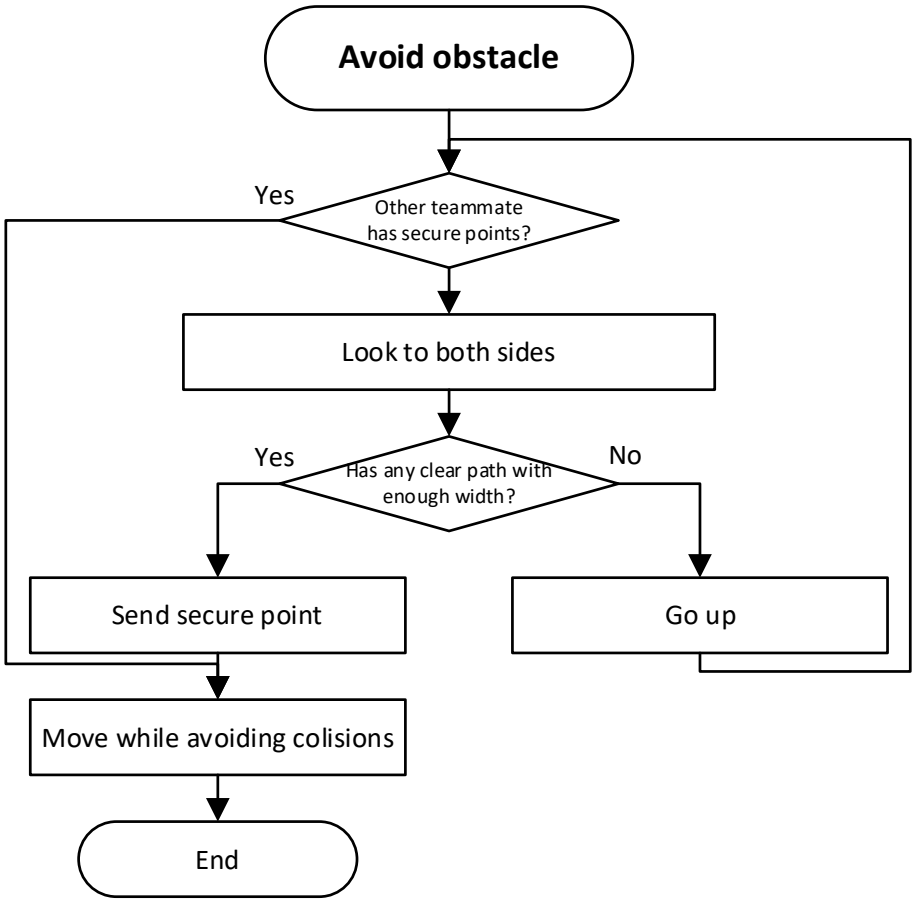



*Figura 43. Diagrama Avoid obstacle*

Una vez que obtenemos el punto seguro en el que retomaremos la trayectoria, comprobaremos si alguno de nuestros compañeros tiene su camino libre (pues mandan un mensaje cuando detectan un obstáculo en frente de ellos). De ser así, el dron se acercará al compañero más cercano, garantizando que no colisionará con ningún otro compañero que se pueda dirigir también a ese punto, utilizando de nuevo el bloque Move avoiding colisions. En la Figura 44 se ilustra esta idea a través del dron A, quien comunica que tiene su camino libre, entonces C se desplaza hasta allí para evitarlo. Al ser un sistema dinámico, A viaja por delante de C, por lo que no se estorban entre ellos.

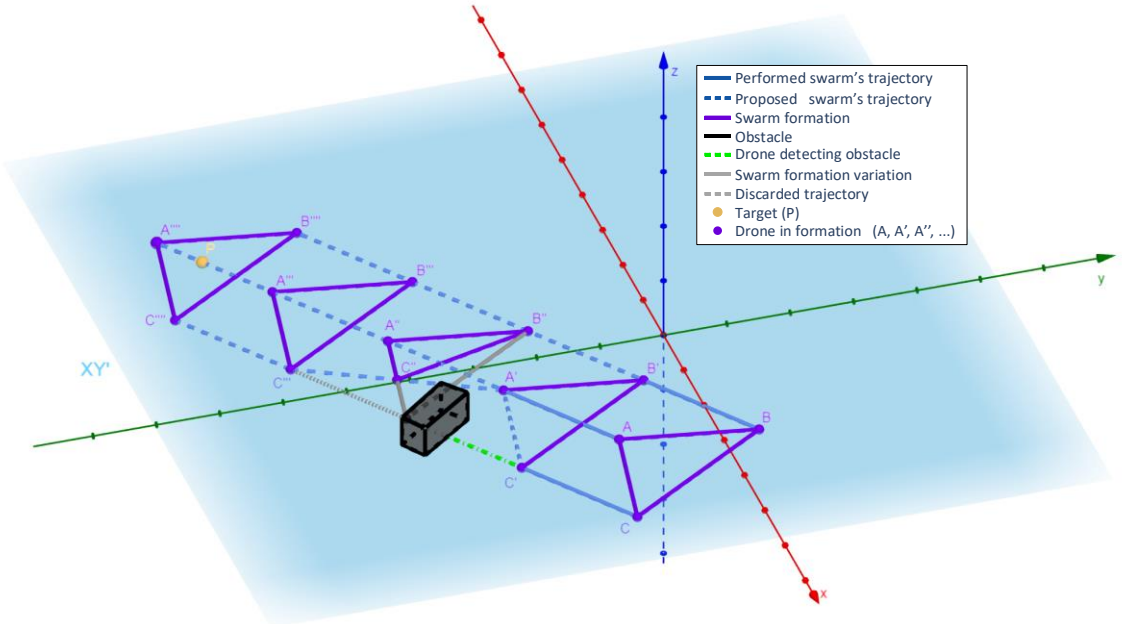


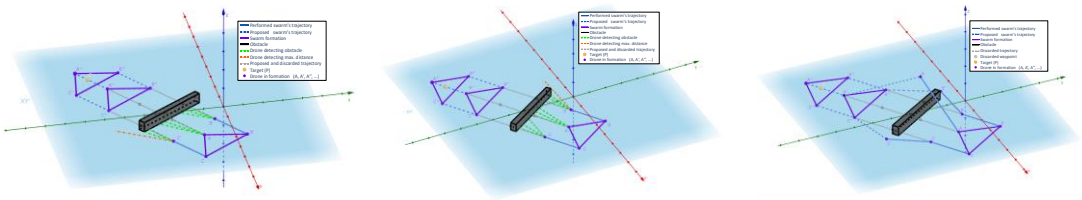

*Figura 44. Evasión de obstáculo por información de un compañero*

$$A_i = A_0 + \overrightarrow{A_o A_{final}} \cdot \lambda_{\mathrm{i}}$$

$$\lambda_{\mathrm{i}} = \frac{tick}{time_{Total}} \cdot i$$

*Tabla 14. Ecuaciones Figura 44*

Esta es la versión simple, pero no siempre es válida, puesto que es posible que todos presentan obstáculos en sus trayectorias. En dicho caso, otra lógica es aplicada, donde los drones en primera fila se giran primero hacia un lado, Figura 45 (a), y luego hacia el otro, Figura 45 (b), en un ángulo determinado para realizar un barrido horizontal y buscar en el obstáculo posibles puntos por donde esquivarlo.



(a) (b) (c)

*Figura 45. Evasión de obstáculo por todo el enjambre*

Si no se encuentra ningún punto seguro en el eje horizontal, los drones se desplazan hacia arriba Figura 45 (c), hasta que en algún momento encuentren un hueco suficiente.

Si lo encuentran, lo compartirán con los demás y se dirigirán hacia ese punto seguro para evitarlo. Una vez se encuentren en un punto seguro, se dirigirán al waypoint en el que habíamos considerado esquivado el objeto y que pertenecía a su antigua trayectoria recta hacia el objetivo. Cabe destacar que, al ser un sistema dinámico y descentralizado, en el ejemplo ilustrado en la Figura 45, los compañeros del dron A le podrían comunicar a este los puntos seguros que han encontrado, pero en este caso el dron A viaja por delante y al no ver ni recibir puntos seguros, decide sortear el obstáculo por encima.

# 6 Resultados

En este capítulo se presentan los resultados obtenidos por el sistema implementado. Donde se expondrán cada una de las ejecuciones de la simulación para los diferentes casos de uso. Por último, se evaluará la solución propuesta mediante la matriz de trazabilidad de requisitos, donde visualizaremos que requisitos han sido satisfechos en cada caso de uso.

## 6.1 Introducción a los casos de uso

El escenario que se ha elegido para la realización de los diferentes casos de uso ha sido el campus de Colmenarejo de la Universidad Carlos III de Madrid, Figura 46 (a). Su visualización es posible gracias a Cesium, Figura 46 (b), por lo que trabajamos en las coordenadas reales de este campus.

(a) (b)

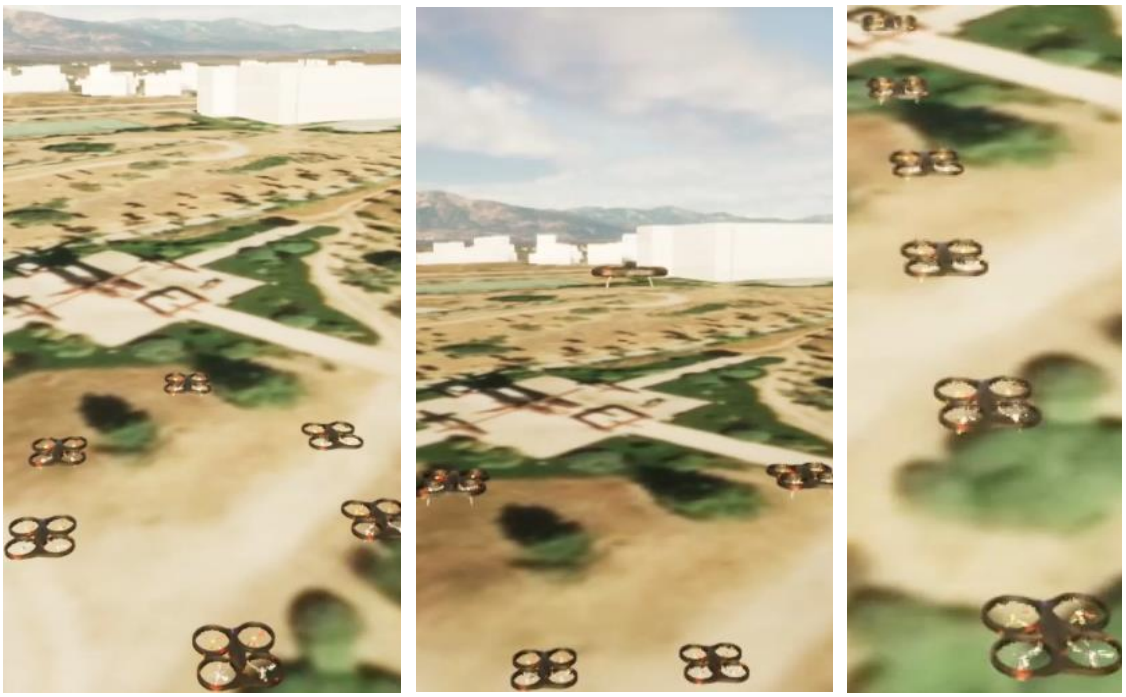
*Figura 46. Campus de Colmenarejo de la Universidad Carlos III de Madrid, real (a) y simulado (b)*

A continuación, se detalla la ejecución del sistema para los diferentes casos de uso expuestos anteriormente. Donde el caso de uso 1 consiste en ir de un punto a otro en formación. En el caso de uso 2, uno de los drones es inhabilitado durante el trayecto y el resto del enjambre reacciona para continuar con la misión. El caso de uso 3 consiste en que un dron evite un obstáculo mediante la información proporcionada por otro compañero del enjambre. Por último, en el caso 4, todo el enjambre se enfrenta a un obstáculo en su camino por lo que deberán de evitarlo.

## 6.2 Caso de uso 1: ir de A a B en formación

Repetiremos este caso de uso para 3 formaciones diferentes: formación tipo 6 en horizontal (hexágono en un plano paralelo al XY), representada en la Figura 47 (a);

formación tipo 5 en vertical (pentágono en un plano paralelo al XZ), representada en la Figura 47 (b), y formación tipo 1 de 6 drones (6 drones en fila), representada en la Figura 47 (c).



(a) (b) (c)

*Figura 47. Formaciones del enjambre adoptadas en simulación*

Por lo que el tipo de formación será un parámetro de la misión, junto con el punto objetivo al que se desea ir en formación, representado en la Figura 48, y el número de drones (6 para la primera y la última ejecución, y 5 drones para la segunda).

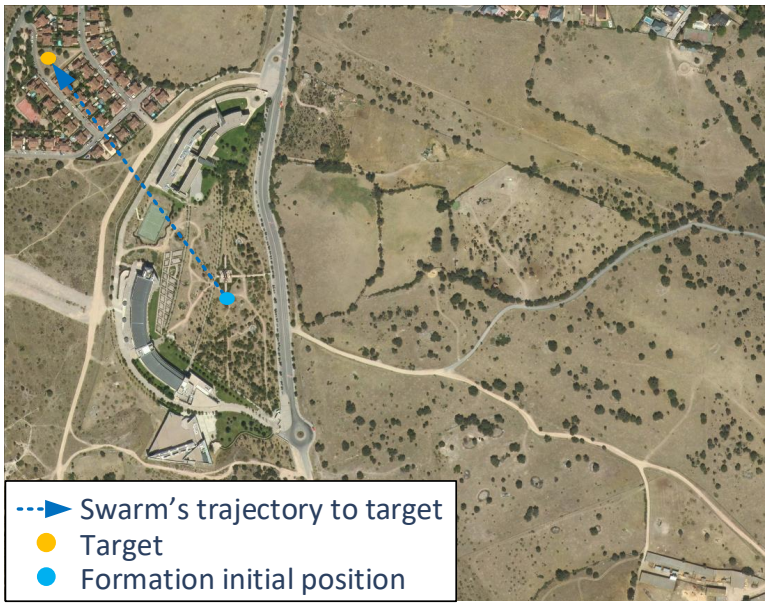



*Figura 48. Trayectoria del enjambre hacia el objetivo del caso de uso 1*

Los drones comunicarán a los demás las coordenadas del objetivo, que será el destino del desplazamiento en formación. Un ejemplo recogido del log de ejecución donde el dron 5 se une a la lista de los que han recibido las coordenadas del objetivo, enviadas en broadcast por el dron 1, se encuentra en la anterior Figura 33.

A continuación, realizarán la formación que corresponda según los parámetros de ejecución. En la Figura 49 se muestra la lista "in" en la que se inscriben los drones y envían al resto. Según el lugar que ocupen en dicha lista, adquieren una posición u otra en la formación.

```
|16:39:59,252|Drone 3|has adopted formation:
{
    "type": "6",
    "in": ["1","3","6","4","2","5"],
    "adopted_by": ["1","6","4","5","2","3"]
}
```

*Figura 49. Mensaje broadcast de dron comunicando que ha adoptado la formación*

En ese momento, tendrán en cuenta la trayectoria del resto de miembros del enjambre para colocarse cada uno en su posición sin colisionar con ningún compañero. En la Figura 50 (a), se muestra como inicialmente cada dron presentaba una trayectoria recta hacia la posición que debía ocupar cada uno en la formación, pero es descartada cuando se comunican entre sí y descubren coincidencias entre sus waypoints. Por lo que algunos deciden modificar su trayectoria incluso hasta dos veces, como es el caso del dron que sigue la trayectoria verde. Pues este último, percibe una segunda coincidencia con el dron

que sigue la trayectoria marcada en naranja, tras haber modificado ambos sus trayectorias por culpa del dron que sigue la trayectoria marcada en azul. Finalmente, todos encuentran un waypoint libre de colisiones y se dirigen hacia el para después alcanzar su posición en la formación pertinente, hecho representado en la Figura 50 (b). En la lista "adopted_by", de la Figura 49, se inscriben los miembros que han considerado que ya han adoptado la formación.

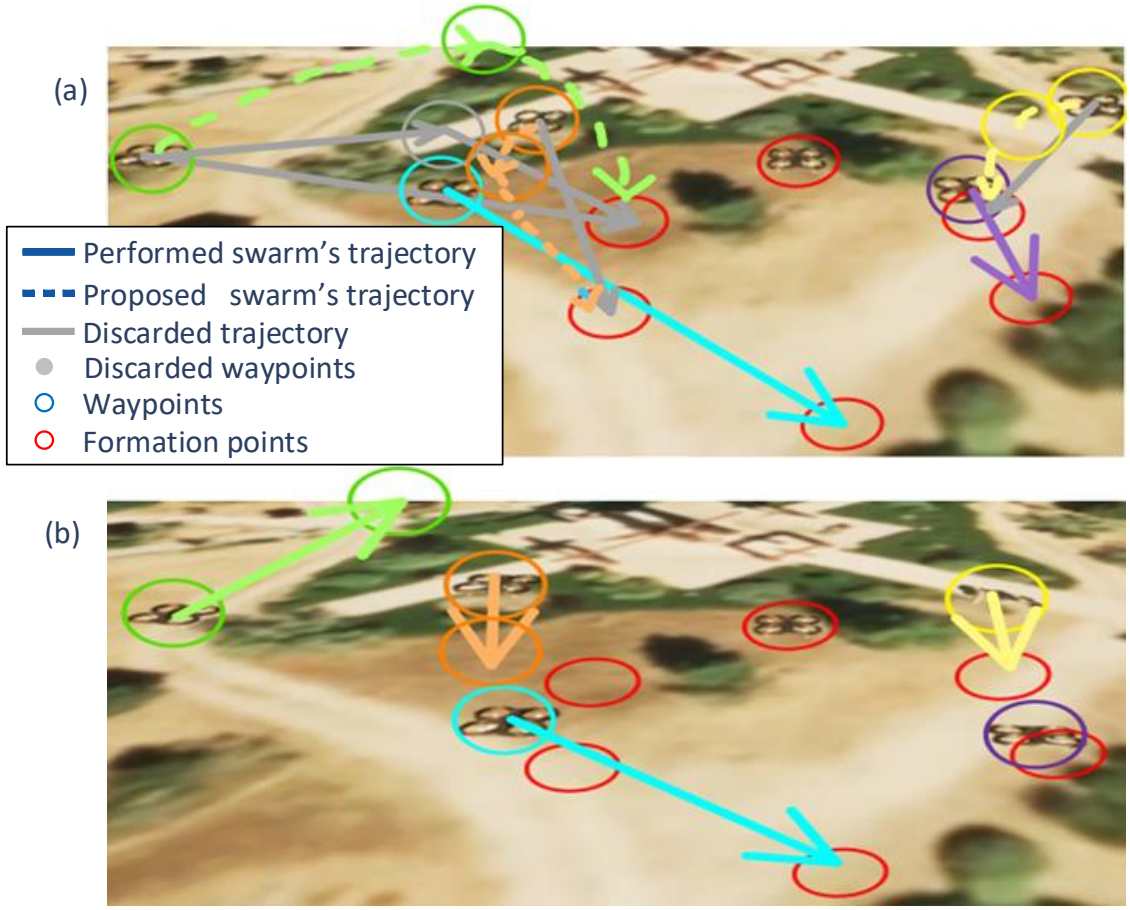



*Figura 50. Miembros del enjambre se evitan entre sí para realizar la formación*

Una vez adoptada la formación, esta se encontrará orientada hacia el objetivo, e iniciarán su viaje todos al mismo tiempo, por lo que deberán de sincronizarse. En la Figura 51 se muestra la lectura de un mensaje de sincronización donde el dron 3 pide sincronizarse en el tiempo indicado en nanosegundos en el campo "sync with me in".

```
|16:39:59,898|Drone 2|READ a message from Drone 3:
{
    "3": {
        "to": ["6","1","4","2","5"],
        "msg": {
            "sync with me in": 1.6559088007649334e+18
```

```
            },
            "read by": ["1","6",4","2"],
            "time_ns": 1655908799265570393
        }
```


*Figura 51. Mensaje de sincronización recibido*

Una vez sincronizados, iniciarán su vuelo en formación, evitando en todo lo posible la deformación de la misma durante el trayecto, como podemos observar en la Figura 52.

*Figura 52. Trayecto hacia el objetivo en formación*

## 6.3 Caso de uso 2: dron inhabilitado durante el trayecto

Para este caso de uso haremos que el enjambre, compuesto por 6 drones, se dirija hacia otro objetivo, representado en la Figura 53.

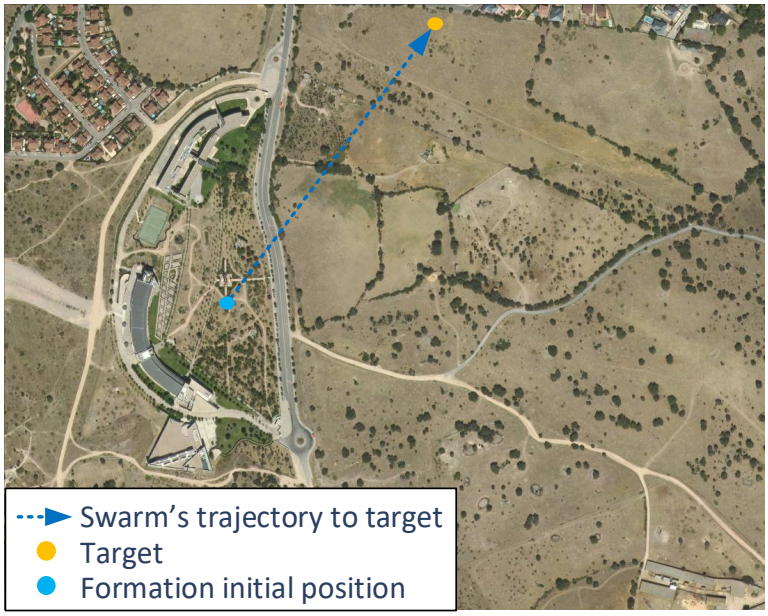



*Figura 53. Trayectoria del enjambre hacia el objetivo del caso de uso 2*

La formación que adopten en el viaje será la establecida por defecto para el numero de drones que participe, es decir, un hexágono en un plano paralelo al XY. Durante el trayecto, desconectaremos un dron al azar, por lo que este caerá simulando ser abatido como se muestra en la Figura 55 (a) y (b). El enjambre debe de percatarse, como se ilustra en la Figura 55 (c), donde se observa como el resto de los miembros del enjambre frenan al darse cuenta de que su compañero no actualiza su posición desde hace "mucho" tiempo (milisegundos). Además, en esta misma figura, se aprecia débilmente el dron caído en la parte inferior izquierda de la imagen. En la Figura 54 se expone el mensaje obtenido en el log de ejecución cuando ocurrió la mencionada situación.

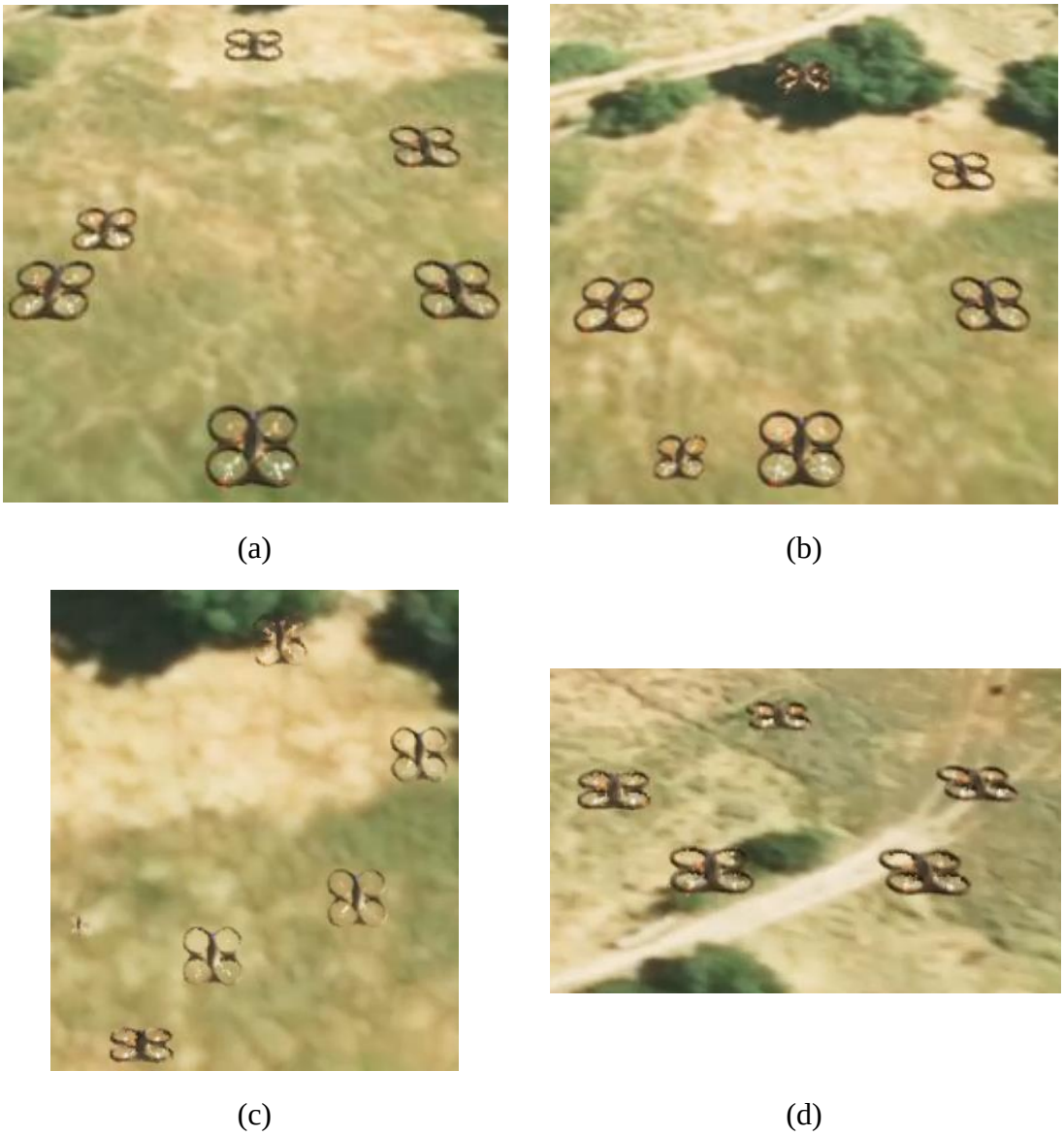
```
|18:12:07,940|Drone 4|Drone 2 is DEAD

|18:12:07,940|Drone 4|changes formation:
{
    "type": "5",
    "in": ["4"],
    "adopted_by": ["4"]
}
```

*Figura 54. Miembro del enjambre percibe que su compañero ha sido neutralizado*

En dicha secuencia, el dron 4 se percata de que el dron 2 lleva demasiado tiempo sin enviar su posición, por lo que le da por muerto, cambiando automáticamente la formación del enjambre al tipo 5. Esta nueva formación se realizará en base a él ya que ha sido el

primero en inscribirse en la lista de formación. Después, el enjambre continúa con su misión como se muestra en la Figura 55 (d).



(a) (b)

(c) (d)

*Figura 55. Enjambre reacciona ante un miembro caído*

## 6.4 Caso de uso 3: dron evita obstáculo gracias a su compañero

En este caso de uso, se ha preparado un escenario con un obstáculo. Concretamente un muro de 50 metros de alto, 20 metros de ancho y 1 metro de profundidad, que se encontrará únicamente en la trayectoria de uno de los drones. Este deberá sortearlo gracias a la información proporcionada por sus compañeros de enjambre, quienes tienen sus trayectorias libres.

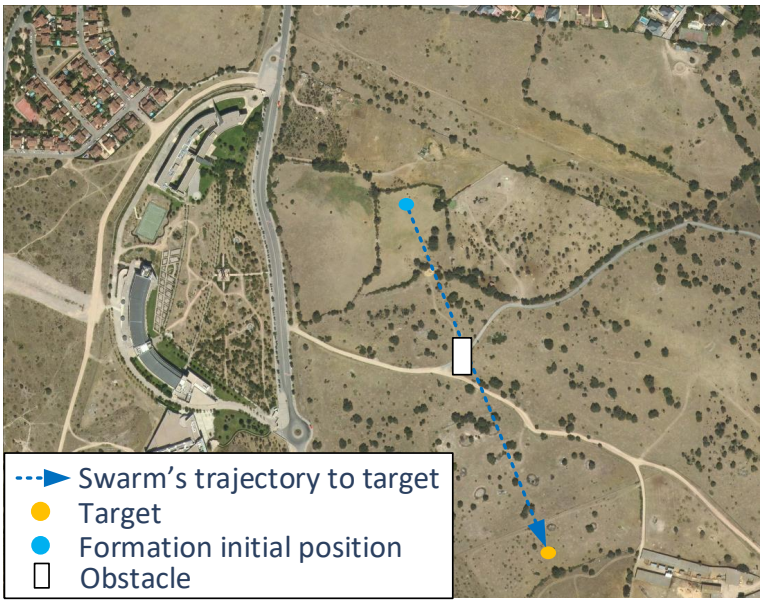
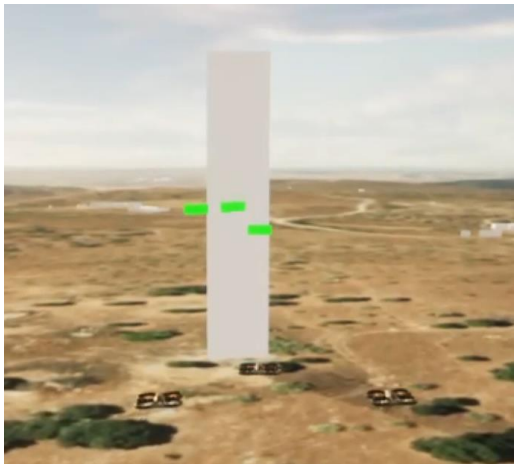



*Figura 56. Trayectoria del enjambre hacia el objetivo del caso de uso 3*

Para esta ejecución, se ha seleccionado un objetivo que se encuentra a 60 metros a partir del muro, por lo que deberán de evitarlo para completar la misión, como se muestra en la Figura 56. El enjambre consta de 3 drones con la formación por defecto para ese caso (triángulo en el plano XY').

En la Figura 57 (a) se observa cómo han creado la formación y se dirigen al obstáculo. Cuando llegan al muro en la Figura 57 (b), solo el dron de la derecha lo detecta. Por tanto, este lee los mensajes de sus compañeros y encuentra que alguno tiene su trayectoria libre. Entonces, opta por seguir al dron que le aporte el punto seguro más cercano, es decir, el dron de su izquierda. En la Figura 57 (c) se observa cómo ha realizado el movimiento de evasión y se coloca detrás del mismo, y finalmente en la Figura 57 (d) todos han superado el obstáculo y se disponen a reagruparse para continuar con la misión.



(a) (b)

(c) (d)

*Figura 57. Dron evita obstáculo gracias a su compañero*

## 6.5 Caso de uso 4: enjambre completo evita un obstáculo

La formación que usaremos para este caso de uso será la misma que en el anterior, un triángulo en un plano paralelo al XY, de manera que se facilite la comparación de ambos casos de uso. Sin embargo, en este caso necesitamos un obstáculo que interrumpa la trayectoria de toda la formación. Por lo que se ha elegido un muro de 20 metros de alto, 50 metros de ancho y 5 metros de profundidad, representado en blanco en la Figura 58.

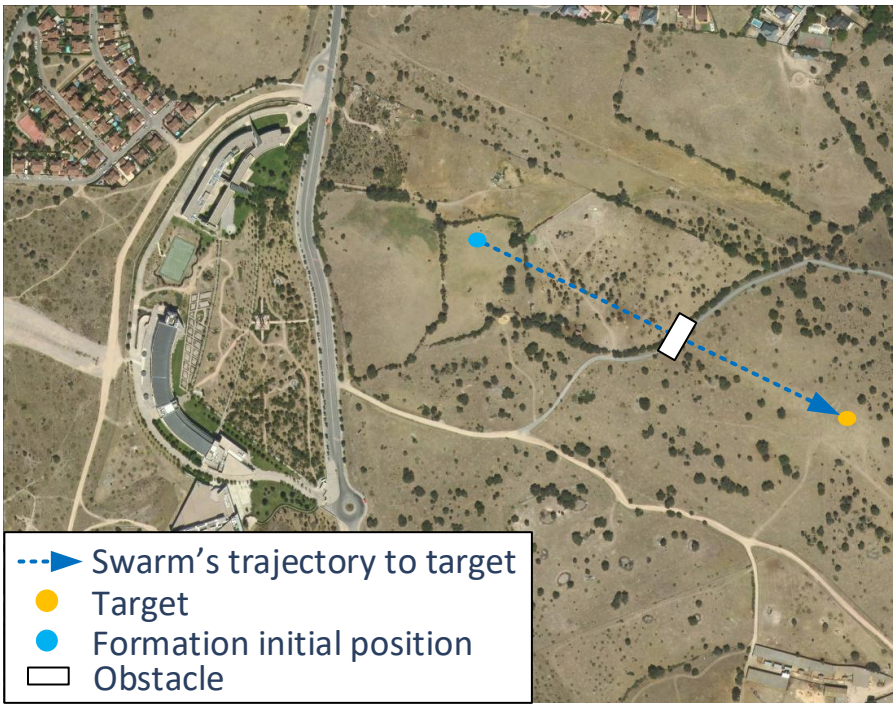
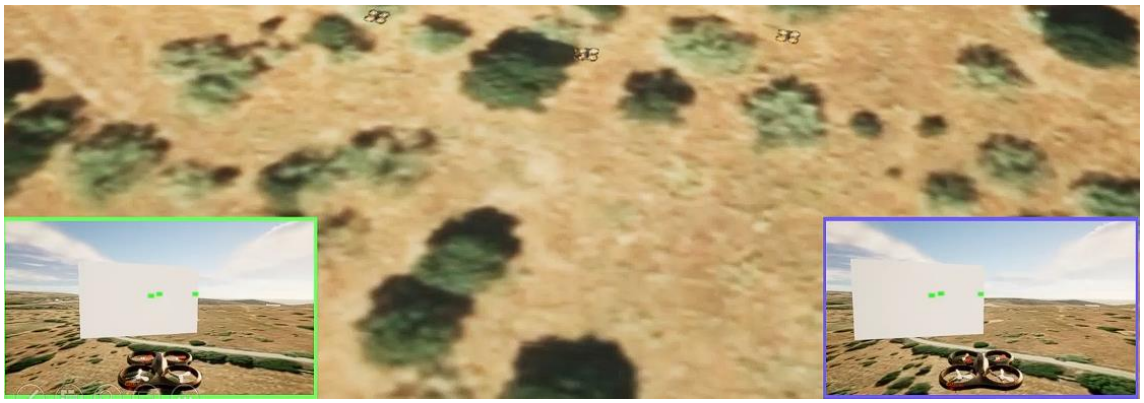



*Figura 58. Trayectoria del enjambre hacia el objetivo del caso de uso 4*

Los drones ubicados en los extremos de la formación lo sortearán por el lateral del obstáculo más próximo a ellos, que encontrarán gracias al barrido realizado con el sensor de distancia. Antes de que esto ocurra, el dron que se encuentra a la cabeza de la formación habrá determinado que no puede evitar el obstáculo por ningún lateral del mismo, ya que su sensor de distancia en ningún momento ha marcado una distancia máxima a lo largo del barrido realizado sobre la superficie del obstáculo, por lo que decide evitarlo por encima. Tras determinar el punto por el que evitan el obstáculo, los drones lo compartirán con sus compañeros por si estos les fuese de utilidad, como en el caso de uso anterior.

La Figura 59 (a) muestra cómo los drones comienzan el barrido con sus sensores de distancia, apreciable en las imágenes de las esquinas inferiores, que hacen referencia a las cámaras colocadas en los drones de los laterales de la formación. En la Figura 59 (b), uno de los drones ha localizado un punto seguro y comienza a evadir el obstáculo, mientras el resto continua con el barrido. En la Figura 59 (c), se observa el dron del centro superando el objeto por encima, mientras sus compañeros lo evitan por los laterales donde han localizado sus respectivos puntos seguros. Finalmente, en la Figura 59 (d), todos los drones han solventado el obstáculo y se reagrupan para continuar con la misión.



(a)

(b)

(c)

(d)

*Figura 59. Enjambre completo evita un obstáculo*

## 6.6 Evaluación de la solución

Mediante la matriz de trazabilidad verificaremos los requisitos que han sido satisfechos durante la ejecución de los casos de uso propuestos para nuestra plataforma. En la Tabla 15, se muestra como han sido satisfechos todos y cada uno de los requisitos funcionales, mientras que en la Tabla 16 se observa que los únicos requisitos no funcionales no cumplidos son los siguientes requisitos opcionales: el RNF03, detallado en la Tabla 30, que trata el cifrado de los mensajes enviados entre los drones; los requisitos RNF16 y RNF17, detallados en la Tabla 43 y Tabla 44 respectivamente, referentes a una interfaz de usuario que aumentase la usabilidad del sistema; y el RNF06, detallado en la Tabla 33. Este último, es la razón por la que el trabajo no se denomina "arquitectura real de enjambres de drones descentralizados", ya que propone que dicha comunicación se realice mediante mensajes de la API MAVSDK y sería la forma de continuar con este trabajo, consiguiendo que todo lo implementado pueda ser directamente trasladado a la realidad.

| RF / CU | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | | |
| 2 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | | |
| 3 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ |
| 4 | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | |

*Tabla 15. Matriz de trazabilidad de requisitos funcionales*

| RNF / CU | 01 | 02 | 03 | 04 | 05 | 06 | 07 | 08 | 09 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | ✓ | ✓ | X | ✓ | ✓ | X | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | X | X | ✓ |
| 2 | ✓ | ✓ | X | ✓ | ✓ | X | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | X | X | ✓ |
| 3 | ✓ | ✓ | X | ✓ | ✓ | X | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | X | X | ✓ |
| 4 | ✓ | ✓ | X | ✓ | ✓ | X | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | X | X | ✓ |

*Tabla 16. Matriz de trazabilidad de requisitos no funcionales*

## 6.7 Alternativas a la solución

Para realizar la sincronización, explicada en el Apartado 0, en un primer momento se pensó en tratar de aproximar la función de delay o retraso que experimentaban los drones en base a la posición en la que llegaban a un punto concreto del código. De esta forma, podríamos determinar el tiempo que debía de esperar cada dron en ese punto para sincronizarse con los demás. Al obtener los tiempos en los que los drones llegaban a dicho punto y representarlos gráficamente, se observó una correlación lineal positiva al orden inverso de llegada, por lo que se trató de aproximar por el método de mínimos cuadrados, el resultado se muestra en la Figura 60.

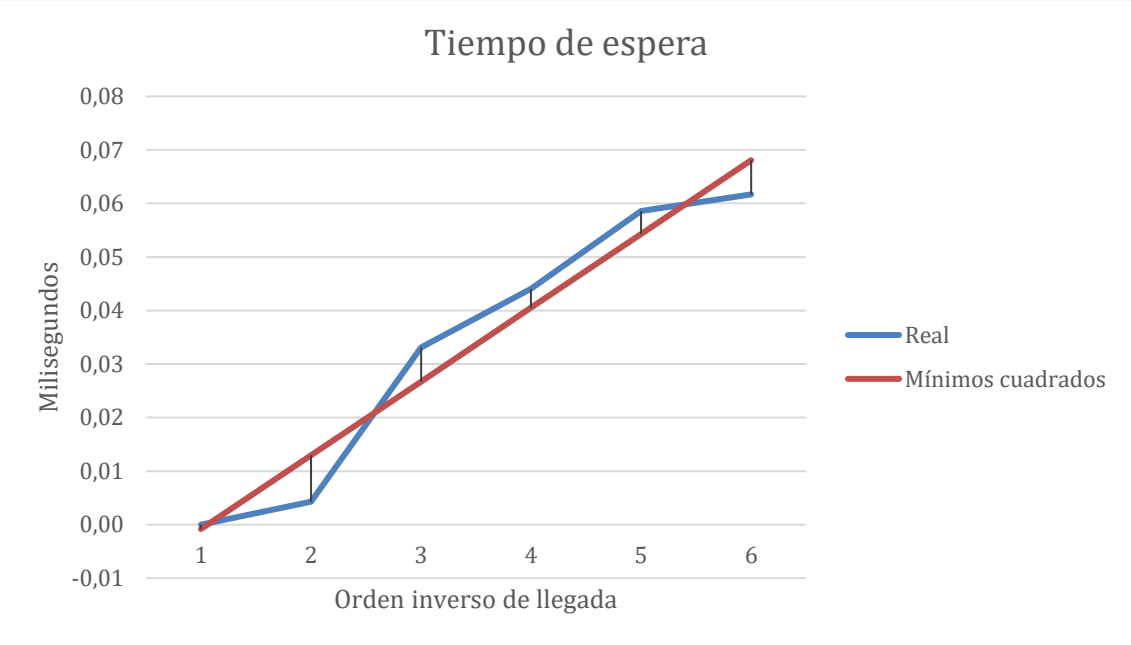



*Figura 60. Aproximación por mínimos cuadrados del tiempo de espera de cada dron para la sincronización del enjambre*

Tras su implementación, y la realización de varias pruebas, se observó que el problema persistía. Pese a que se probaron otras aproximaciones no lineales como las logarítmicas, no se podía considerar que se hubiera conseguido sincronizar a los drones.

Por otro lado, durante la implementación del proceso descentralizado, se encontraron ciertas limitaciones de la API MAVSDK-Python al usar programación multi-hilo. Diferentes hilos invocaban simultáneamente al método de esta librería encargado de armar el UAV antes de su despegue, y este retornaba una excepción de comando denegado. Tras consultar a los desarrolladores de dicha librería, el problema no pudo resolverse con éxito.

# 7 CONCLUSIONES Y TRABAJOS FUTUROS

Dado que los requisitos no satisfechos son opcionales, se puede determinar que se ha encontrado una solución al problema planteado, pues se ha desarrollado un entorno de simulación de código abierto donde se pueden realizar implementaciones para investigar, mejorar y probar enjambres de drones descentralizados. Como demostración de las posibilidades de la arquitectura desarrollada, se ha verificado que permite al enjambre adoptar diferentes formaciones, en diferentes planos geométricos, en las que pueden viajar coordinados hacia un objetivo común. Además, se garantiza la seguridad del enjambre ante obstáculos inmóviles que encuentre en su camino y ante errores o ataques que inhabiliten a alguno de sus miembros.

## 7.1 Trabajos futuros

Para hablar de los trabajos futuros a partir del presente, debemos de hablar de los requisitos no cumplidos. En una versión futura el entorno debería asegurar el cifrado de las comunicaciones entre los drones, de esta forma se aumenta la seguridad del enjambre antes posibles ataques sobre dichas comunicaciones. Estos mensajes deberían ser transmitidos a través de MAVSDK, lo que facilitaría la futura migración de los proyectos simulados a la realidad. Igualmente, sería de gran utilidad poder contar con una interfaz gráfica multi-idioma que facilitase la configuración del entorno ya que agilizaría las labores de investigación.

También, podría realizarse una solución similar a este problema, pero basada en la programación multi-hilo que presentaría un mejor rendimiento ante tareas muy exigentes computacionalmente.

Como conclusión, este proyecto sienta las bases para futuros trabajos de enjambres de drones descentralizados diseñados y probados íntegramente en simulación que puedan funcionar con el menor esfuerzo posible en la realidad. Algunos de estos trabajos pueden ser:

- Enjambre de drones distribuido para la generación de ortofotos, los cuales se reparten de manera óptima la zona de interés.
- Enjambre de drones distribuido para detección de incendios y vertidos.
- Enjambre de drones distribuido para labores de búsqueda y vigilancia costera, en parques naturales, en la montaña, etc.

- Enjambre de drones distribuido para realizar fotogrametría de un área de interés o reconstrucción de superficies 3D.
- Enjambre de drones distribuido para el desarrollo de estrategias militares.

# 8 BIBLIOGRAFÍA


[1] H. Ling, H. Luo, H. Chen, L. Bai, T. Zhu, y Y. Wang, «Modelling and Simulation of Distributed UAV Swarm Cooperative Planning and Perception», *International Journal of Aerospace Engineering*, vol. 2021, pp. 1-11, may 2021, doi: 10.1155/2021/9977262.

[2] Luis Tercero, *El Trading de Alta Frecuencia (Documental La Noche Temática)*, (12 de junio de 2013). Accedido: 21 de junio de 2022. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=a3pVqMT9DXA

[3] PorTierraMaryAire podcast, *PTMyA T4E15: Enjambres de drones. Proyecto LISS de Escribano Mechanical & Engineering*, (27 de enero de 2021). Accedido: 21 de junio de 2022. [En línea Video]. Disponible en: https://www.youtube.com/watch?v=w4qMj9tXqzY

[4] C. D. V. López, «El programa LISS de Escribano -», 21 de febrero de 2021. https://www.revistaejercitos.com/2021/02/21/el-programa-liss-long-range-intelligent-security-system-de-escribano/ (accedido 21 de junio de 2022).

[5] J. Wei, H. Li, M. Guo, J. Li, y H. Huang, «Backstepping Control Based on Constrained Command Filter for Hypersonic Flight Vehicles with AOA and Actuator Constraints», *International Journal of Aerospace Engineering*, vol. 2021, pp. 1-16, sep. 2021, doi: 10.1155/2021/8620873.

[6] «Welcome to DigitalSky». https://digitalsky.dgca.gov.in/home (accedido 26 de abril de 2022).

[7] V. Chamola, P. Kotesh, A. Agarwal, Naren, N. Gupta, y M. Guizani, «A Comprehensive Review of Unmanned Aerial Vehicle Attacks and Neutralization Techniques», *Ad Hoc Networks*, vol. 111, p. 102324, feb. 2021, doi: 10.1016/j.adhoc.2020.102324.

[8] P. K. Chittoor, B. Chokkalingam, y L. Mihet-Popa, «A Review on UAV Wireless Charging: Fundamentals, Applications, Charging Techniques and Standards», *IEEE Access*, vol. 9, pp. 69235-69266, 2021, doi: 10.1109/ACCESS.2021.3077041.

[9] T. Chang y H. Yu, «Improving Electric Powered UAVs' Endurance by Incorporating Battery Dumping Concept», *Procedia Engineering*, vol. 99, dic. 2015, doi: 10.1016/j.proeng.2014.12.522.

[10] Electropaedia, «Battery and Energy Technologies». https://www.mpoweruk.com/lithium_failures.htm

[11] B. Galkin, J. Kibilda, y L. Dasilva, «UAVs as Mobile Infrastructure: Addressing Battery Lifetime», *IEEE Communications Magazine*, vol. PP, pp. 2-7, feb. 2019, doi: 10.1109/MCOM.2019.1800545.

[12] A. Malaver, N. Motta, P. Corke, y L. Gonzalez, «Development and Integration of a Solar Powered Unmanned Aerial Vehicle and a Wireless Sensor Network to Monitor Greenhouse Gases», *Sensors*, vol. 15, pp. 4072-4096, feb. 2015, doi: 10.3390/s150204072.

[13] M. Achtelik, J. Stumpf, D. Gurdan, y K.-M. Doth, «Design of a flexible high performance quadcopter platform breaking the MAV endurance record with laser power beaming», sep. 2011, pp. 5166-5172. doi: 10.1109/IROS.2011.6048336.

[14] M. Lu, M. Bagheri, A. James, y T. Phung, «UAV Wireless Charging: A Review, Reconceptualization, and Extension», *IEEE Access*, vol. PP, may 2018, doi: 10.1109/ACCESS.2018.2841376.

[15] M. Deittert, A. Richards, C. Toomer, y A. Pipe, «Engineless Unmanned Aerial Vehicle Propulsion by Dynamic Soaring», *Journal of Guidance Control and Dynamics - J GUID CONTROL DYNAM*, vol. 32, pp. 1446-1457, sep. 2009, doi: 10.2514/1.43270.

[16] P. Richardson, «Upwind Dynamic Soaring of Albatrosses and UAVs», *Progress in Oceanography*, vol. 130, nov. 2014, doi: 10.1016/j.pocean.2014.11.002.

[17] E. Çetin, C. Barrado, y E. Pastor, «Counter a Drone in a Complex Neighborhood Area by Deep Reinforcement Learning», *Sensors*, vol. 20, n.º 8, p. 2320, abr. 2020, doi: 10.3390/s20082320.

[18] P. DeLellis *et al.*, «Collective behaviour across animal species», *Sci Rep*, vol. 4, n.º 1, Art. n.º 1, ene. 2014, doi: 10.1038/srep03723.

[19] J. Buhl *et al.*, «From Disorder to Order in Marching Locusts», *Science (New York, N.Y.)*, vol. 312, pp. 1402-6, jul. 2006, doi: 10.1126/science.1125142.

[20] T. Mora *et al.*, «Local equilibrium in bird flocks», *Nature Physics*, vol. 12, ago. 2016, doi: 10.1038/nphys3846.

[21] M. Ballerini *et al.*, «Interaction ruling animal collective behavior depends on topological rather than metric distance: Evidence from a field study», *Proceedings of the National Academy of Sciences*, vol. 105, n.º 4, pp. 1232-1237, ene. 2008, doi: 10.1073/pnas.0711437105.

[22] U. Lopez, J. Gautrais, I. D. Couzin, y G. Theraulaz, «From behavioural analyses to models of collective motion in fish schools», *Interface Focus*, vol. 2, n.º 6, pp. 693-707, dic. 2012, doi: 10.1098/rsfs.2012.0033.

[23] H. Hildenbrandt, C. Carere, y C. K. Hemelrijk, «Self-organized aerial displays of thousands of starlings: a model», *Behavioral Ecology*, vol. 21, n.º 6, pp. 1349-1359, nov. 2010, doi: 10.1093/beheco/arq149.

[24] T. Vicsek y A. Zafeiris, «Collective motion», *Physics Reports*, vol. 517, n.º 3, pp. 71-140, ago. 2012, doi: 10.1016/j.physrep.2012.03.004.

[25] B. T. Fine y D. A. Shell, «Unifying microscopic flocking motion models for virtual, robotic, and biological flock members», *Auton. Robots*, vol. 35, n.º 2-3, pp. 195-219, oct. 2013, doi: 10.1007/s10514-013-9338-z.

[26] E. Méhes y T. Vicsek, «Collective motion of cells: from experiments to models», *Integr. Biol.*, vol. 6, n.º 9, pp. 831-854, 2014, doi: 10.1039/C4IB00115J.

[27] G. Vásárhelyi, C. Virágh, G. Somorjai, T. Nepusz, A. E. Eiben, y T. Vicsek, «Optimized flocking of autonomous drones in confined environments», *Sci. Robot.*, vol. 3, n.º 20, p. eaat3536, jul. 2018, doi: 10.1126/scirobotics.aat3536.

[28] 14:00-17:00, «ISO 8373:2012», *ISO*. https://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/05/58/55890.html (accedido 21 de junio de 2022).

[29] «Drone Light Shows Powered by Intel», *Intel*. https://www.intel.com/content/www/us/en/technology-innovation/intel-drone-light-shows.html (accedido 21 de junio de 2022).

[30] «EHang | Aerial Media - UAV Formation Light Show». https://www.ehang.com/formation/ (accedido 21 de junio de 2022).

[31] «Drone-shows-Creative-potential-and-best-practices.pdf». Accedido: 21 de junio de 2022. [En línea]. Disponible en: https://www.taittowers.com/wp-content/uploads/2017/03/Drone-shows-Creative-potential-and-best-practices.pdf

[32] N. Jakobi, P. Husbands, y I. Harvey, «“Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics,”» ene. 1995, vol. 929, pp. 704-720.

[33] M. Dorigo, G. Theraulaz, y V. Trianni, «Swarm Robotics: Past, Present, and Future [Point of View]», *Proc. IEEE*, vol. 109, n.º 7, pp. 1152-1165, jul. 2021, doi: 10.1109/JPROC.2021.3072740.

[34] W. Sun, «Distributed Optimal Scheduling in UAV Swarm Network», en *2021 IEEE 18th Annual Consumer Communications & Networking Conference (CCNC)*, Las Vegas, NV, USA, ene. 2021, pp. 1-4. doi: 10.1109/CCNC49032.2021.9369643.

[35] D. H. Stolfi, M. R. Brust, G. Danoy, y P. Bouvry, «Optimising pheromone communication in a UAV swarm», en *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, Lille France, jul. 2021, pp. 323-324. doi: 10.1145/3449726.3459526.

[36] A. Filatov, M. Zaslavskiy, y K. Krinkin, «Multi-Drone 3D Building Reconstruction Method», *Mathematics*, vol. 9, p. 3033, nov. 2021, doi: 10.3390/math9233033.

[37] G. Podevijn, R. O’Grady, N. Mathews, A. Gilles, F. Carole, y M. Dorigo, «Investigating the effect of increasing robot group sizes on the human psychophysiological state in the context of human–swarm interaction», *Swarm Intelligence*, vol. 10, sep. 2016, doi: 10.1007/s11721-016-0124-3.

[38] G. Podevijn, R. O’Grady, F. Carole, y M. Dorigo, «Human Responses to Stimuli Produced by Robot Swarms - the Effect of the Reality-Gap on Psychological State», 2018, pp. 531-543. doi: 10.1007/978-3-319-73008-0_37.

[39] C. Nam, P. Walker, H. Li, M. Lewis, y K. Sycara, «Models of Trust in Human Control of Swarms With Varied Levels of Autonomy», *IEEE Transactions on Human-Machine Systems*, vol. PP, pp. 1-11, feb. 2019, doi: 10.1109/THMS.2019.2896845.

[40] A. Tahir, J. Böling, M.-H. Haghbayan, H. Toivonen, y J. Plosila, «Swarms of Unmanned Aerial Vehicles – A Survey», *Journal of Industrial Information Integration*, vol. 16 (100106), oct. 2019, doi: 10.1016/j.jii.2019.100106.

[41] Y. Wei, M. B. Blake, y G. R. Madey, «An Operation-Time Simulation Framework for UAV Swarm Configuration and Mission Planning», *Procedia Computer Science*, vol. 18, pp. 1949-1958, ene. 2013, doi: 10.1016/j.procs.2013.05.364.

[42] W. Wu *et al.*, «CoUAV: A Cooperative UAV Fleet Control and Monitoring Platform». arXiv, 8 de abril de 2019. doi: 10.48550/arXiv.1904.04046.

[43] B. Schlotfeldt, D. Thakur, N. Atanasov, V. Kumar, y G. J. Pappas, «Anytime Planning for Decentralized Multirobot Active Information Gathering», *IEEE Robot. Autom. Lett.*, vol. 3, n.º 2, pp. 1025-1032, abr. 2018, doi: 10.1109/LRA.2018.2794608.

[44] «Routing · MAVLink Developer Guide». https://mavlink.io/en/guide/routing.html (accedido 21 de junio de 2022).

[45] «Message Signing · MAVLink Developer Guide». https://mavlink.io/en/guide/message_signing.html (accedido 26 de abril de 2022).

[46] G. Zogopoulos-Papaliakos, M. Logothetis, y K. J. Kyriakopoulos, «A Fault Diagnosis Framework for MAVLink-Enabled UAVs Using Structural Analysis», en *2019 International Conference on Robotics and Automation (ICRA)*, Montreal, QC, Canada, may 2019, pp. 676-682. doi: 10.1109/ICRA.2019.8793760.

[47] «License (GPLv3) — Dev documentation». https://ardupilot.org/dev/docs/license-gplv3.html (accedido 22 de junio de 2022).

[48] «Review de Pixhawk 4.» https://crebots.com/review-de-pixhawk-4/ (accedido 22 de junio de 2022).

[49] «Holybro Pixhawk 4 · PX4 v1.9.0 User Guide». https://docs.px4.io/v1.9.0/en/flight_controller/pixhawk4.html (accedido 27 de marzo de 2022).

[50] «Features -- Gazebo». https://gazebosim.org/features (accedido 22 de junio de 2022).

[51] «Home - AirSim». https://microsoft.github.io/AirSim/ (accedido 22 de junio de 2022).

[52] «PX4 Setup for AirSim - AirSim». https://microsoft.github.io/AirSim/px4_setup/ (accedido 22 de junio de 2022).

[53] *SCRIMMAGE Multi-Agent Simulator*. Georgia Tech Research Institute, 2022. Accedido: 21 de junio de 2022. [En línea]. Disponible en:

https://github.com/gtri/scrimmage/blob/7f474ff2f9691e5de33df2505885585e4ef26aff/docs/source/tutorials/airsim-plugin.rst

[54] «Sensors - AirSim». https://microsoft.github.io/AirSim/sensors/ (accedido 22 de junio de 2022).

[55] «Aerial photogrammetry and drone-based LiDAR for surveying missions», *LiDAR drone OnyxScan, UAV 3D laser scanner*, 10 de enero de 2019. https://www.onyxscan-lidar.com/aerial-photogrammetry-and-drone-based-lidar-for-surveying-missions/ (accedido 26 de abril de 2022).

[56] «Infrared Camera - AirSim». https://microsoft.github.io/AirSim/InfraredCamera/ (accedido 22 de junio de 2022).

[57] «Event camera - AirSim». https://microsoft.github.io/AirSim/event_sim/ (accedido 22 de junio de 2022).

[58] «Simulation | PX4 User Guide». https://docs.px4.io/master/en/simulation/ (accedido 29 de marzo de 2022).

[59] R. Python, «Async IO in Python: A Complete Walkthrough – Real Python». https://realpython.com/async-io-python/ (accedido 22 de junio de 2022).

[60] «Multitasking and Context Switching», *Bryan Braun - Frontend Developer*, 25 de junio de 2012. https://www.bryanbraun.com/2012/06/25/multitasking-and-context-switching/ (accedido 29 de marzo de 2022).

[61] «Javi Blog - Hilos, concurrencia y programación asíncrona». https://javiblog.com/2021/hilos-threads-concurrencia-programacion-asincrona.html (accedido 29 de marzo de 2022).

[62] «Tornado Web Server — Tornado 6.1 documentation». https://www.tornadoweb.org/en/stable/index.html (accedido 29 de marzo de 2022).

[63] «What is gevent? — gevent 21.12.1.dev0 documentation». http://www.gevent.org/ (accedido 29 de marzo de 2022).

[64] «Asyncio Tutorial: Async Programming in Python», *Django Stars Blog*, 12 de abril de 2017. https://djangostars.com/blog/asynchronous-programming-in-python-asyncio/ (accedido 29 de marzo de 2022).

[65] «DroneKit», *DroneKit*. http://dronekit.io (accedido 22 de junio de 2022).

[66] «MAVSDK, The Standards Compliant SDK for MAVLink», *Dronecode Foundation*. https://www.dronecode.org/sdk/ (accedido 22 de junio de 2022).

[67] «Introduction · MAVSDK Guide». https://mavsdk.mavlink.io/main/en/index.html (accedido 22 de junio de 2022).

[68] *DroneKit Python*. DroneKit, 2022. Accedido: 22 de junio de 2022. [En línea]. Disponible en: https://github.com/dronekit/dronekit-python

[69] «GitHub: Where the world builds software», *GitHub*. https://github.com/ (accedido 22 de junio de 2022).

[70] «Zotero | Your personal research assistant». https://www.zotero.org/ (accedido 22 de junio de 2022).

[71] «VPN Acceso remoto (solución recomendada) | UC3M». https://www.uc3m.es/sdic/servicios/vpn (accedido 22 de junio de 2022).

[72] «¿Cuánto Cobra un Ingeniero Informático? (Sueldo 2022) | Jobted.es». https://www.jobted.es/salario/ingeniero-inform%C3%A1tico (accedido 22 de junio de 2022).

[73] «REGLAMENTO (UE) 2018/ 1139 DEL PARLAMENTO EUROPEO Y DEL CONSEJO - de 4 de julio de 2018 - sobre normas comunes en el ámbito de la aviación civil y por el que se crea una Agencia de la Unión Europea para la Seguridad Aérea y por el que se modifican los Reglamentos (CE) n.o 2111/ 2005, (CE) n.o 1008/ 2008, (UE) n.o 996/ 2010, (CE) n.o 376/ 2014 y las Directivas 2014/ 30/ UE y 2014/ 53/ UE del Parlamento Europeo y del Consejo y se derogan los Reglamentos (CE) n.o 552/ 2004 y (CE) n.o 216/ 2008 del Parlamento Europeo y del Consejo y el Reglamento (CEE) n.o 3922/ 91 del Consejo», p. 122.

[74] «COMMISSION DELEGATED REGULATION (EU) 2019/ 945 - of 12 March 2019 - on unmanned aircraft systems and on third-country operators of unmanned aircraft systems», p. 40.

[75] Ministerio de la Presidencia y para las Administraciones Territoriales, *Real Decreto 1036/2017, de 15 de diciembre, por el que se regula la utilización civil de las aeronaves pilotadas por control remoto, y se modifican el Real Decreto 552/2014, de 27 de junio, por el que se desarrolla el Reglamento del aire y disposiciones*

*operativas comunes para los servicios y procedimientos de navegación aérea y el Real Decreto 57/2002, de 18 de enero, por el que se aprueba el Reglamento de Circulación Aérea*, vol. BOE-A-2017-15721. 2017, pp. 129609-129641. Accedido: 21 de junio de 2022. [En línea]. Disponible en: https://www.boe.es/eli/es/rd/2017/12/15/1036

[76] «DOC20201214134023 2020+12+14+Contrato+1003220007500+SENDISTAR +firmado.pdf». Accedido: 22 de junio de 2022. [En línea]. Disponible en: https://contrataciondelestado.es/wps/wcm/connect/2a52c5ae-cbcf-46a6-9a05-235c22237028/DOC20201214134023 2020+12+14+Contrato+1003220007500+SENDISTAR+firmado.pdf?MOD=AJPERES

[77] J. Molas, «El vínculo entre la innovación militar y civil: hacia un nuevo marco de relación», *Arbor : Ciencia, Pensamiento y Cultura*, vol. CLXXXIV, dic. 2008, doi: 10.3989/arbor.2008.iA2.346.

[78] J. Díaz, «El Pentágono cree que su nueva arma "manada de lobos" cambiará la guerra», *elconfidencial.com*, 20 de junio de 2022. https://www.elconfidencial.com/tecnologia/novaceno/2022-06-20/superarma-manada-de-lobos-enjambre-drones_3443823/ (accedido 22 de junio de 2022).

[79] J. Díaz, «Rusia está usando una nueva arma secreta en Ucrania», *elconfidencial.com*, 18 de mayo de 2022. https://www.elconfidencial.com/tecnologia/novaceno/2022-05-18/rusia-ucrania-canon-laser-drones_3426766/ (accedido 22 de junio de 2022).

[80] «Drone Services Market Size & Share | Industry Report, 2021-2026 | MarketsandMarkets™». https://www.marketsandmarkets.com/Market-Reports/drone-services-market-80726041.html (accedido 21 de junio de 2022).

[81] García-Aunon, P., Roldán, J.J., De León, J.; Del Cerro, J., Barrientos, A., «Practical applications using multi-UAV systems and aerial robotic swarms.», *Revista Iberoamericana de Automática e Informática Industrial*, n.º 18, pp. 230-241, 2021.

[82] craigloewen-msft, «Instalación de WSL». https://docs.microsoft.com/es-es/windows/wsl/install (accedido 22 de junio de 2022).

[83] «Unreal Engine is an Advanced Training and Simulation Software Platform», *Unreal Engine*. https://www.unrealengine.com/en-US/solutions/simulation (accedido 22 de junio de 2022).

[84] «Cesium for Unreal», *Cesium*. https://cesium.com/platform/cesium-for-unreal/ (accedido 22 de junio de 2022).

[85] «Platform», *Cesium*. https://cesium.com/platform/ (accedido 22 de junio de 2022).

[86] «Ficha». https://aplicaciones.uc3m.es/cpa/generaFicha?est=218&asig=15758&idioma=1 (accedido 22 de junio de 2022).

# Anexo A. REQUISITOS

## A.1 Requisitos funcionales

### A.1.1 Disponibilidad de información

| | |
|---|---|
| **Identificador:** RF01 | |
| **Nombre:** | Sensores |
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | El sistema debe ser capaz de obtener la información recopilada por los sensores de cada dron. |

*Tabla 17. RF01 Sensores*

| | |
|---|---|
| **Identificador:** RF02 | |
| **Nombre:** | Registro y almacenamiento de información |
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | El sistema debe ser capaz de registrar toda la información recibida, así como almacenarla para poder mostrarla posteriormente si es requerido. |

*Tabla 18. RF02 Registro y almacenamiento de información*

### A.1.2 Descentralización

| | |
|---|---|
| **Identificador:** RF03 | |
| **Nombre:** | Descentralización |
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | El sistema debe ser descentralizado, permitiendo que, si se pierde comunicación con uno o varios drones cualesquiera, la misión pueda continuar. |

*Tabla 19. RF03 Descentralización*

### A.1.3 Comunicación

| | |
|---|---|
| **Identificador:** RF04 | |
| **Nombre:** | Comunicación entre los drones |
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | El sistema debe garantizar la comunicación entre los drones. |

*Tabla 20. RF04 Comunicación entre los drones*

| |
|---|
| **Identificador:** RF05 |

| **Nombre:** | Cambiar de objetivo |
|---|---|
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | El sistema debe garantizar que los drones comuniquen diferentes objetivos. |

*Tabla 21. RF05 Cambiar de objetivo*

### A.1.4 Adopción de formaciones

| **Identificador:** RF06 | |
|---|---|
| **Nombre:** | Adopción de formaciones en un plano paralelo al XY |
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | El sistema debe permitir al enjambre la adopción de diferentes formaciones en un plano paralelo al XY. |

*Tabla 22. RF06 Adopción de formaciones en un plano paralelo al XY*

| **Identificador:** RF07 | |
|---|---|
| **Nombre:** | Adopción de formaciones en un plano paralelo al XZ |

| | |
|---|---|
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | El sistema debe permitir al enjambre la adopción de diferentes formaciones en un plano paralelo al XZ. |

*Tabla 23. RF07 Adopción de formaciones en un plano paralelo al XZ*

| | |
|---|---|
| **Identificador:** RF08 | |
| **Nombre:** | Rotar en dirección al objetivo |
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | El sistema debe permitir que la formación del enjambre rote de forma que viaje posicionada en dirección al objetivo. |

*Tabla 24. RF08 Rotar en dirección al objetivo*

### A.1.5 Evasión de colisiones

| | |
|---|---|
| **Identificador:** RF09 | |
| **Nombre:** | Evasión de colisiones entre drones del enjambre |
| **Prioridad:** Alta | |

| | |
|---|---|
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | El sistema debe garantizar que los drones que componen el enjambre no colisionarán entre ellos. |

*Tabla 25. RF09 Evasión de colisiones entre drones del enjambre*

| | |
|---|---|
| **Identificador:** RF10 | |
| **Nombre:** | Evasión de colisiones con obstáculos |
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | El sistema debe garantizar que los drones del enjambre eviten colisionar con obstáculos del medio. |

*Tabla 26. RF10 Evasión de colisiones con obstáculos*

| | |
|---|---|
| **Identificador:** RF11 | |
| **Nombre:** | Evasión de colisiones por información de otros miembros |
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |

| | |
|---|---|
| **Descripción:** | El sistema debe garantizar que los drones del enjambre eviten colisionar con obstáculos del medio gracias a la información proporcionada por el resto de los miembros del enjambre. |

*Tabla 27. RF11 Evasión de colisiones por información de otros miembros*

## A.2 Requisitos no funcionales

### A.2.1 Disponibilidad

| | | |
|---|---|---|
| **Identificador:** RNF01 | | |
| **Nombre:** | Disponibilidad | |
| **Prioridad:** Alta | | |
| **Necesidad:** Obligatorio | | |
| **Claridad:** Alta | | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre | |
| **Descripción:** | El sistema tiene que estar disponible siempre que lo desee el usuario. | |

*Tabla 28. RNF01 Disponibilidad*

### A.2.2 Seguridad

| | | |
|---|---|---|
| **Identificador:** RNF02 | | |
| **Nombre:** | Integridad | |
| **Prioridad:** Alta | | |
| **Necesidad:** Obligatorio | | |
| **Claridad:** Alta | | **Verificabilidad:** Alta |

| | |
|---|---|
| **Estabilidad:** | Siempre |
| **Descripción:** | Se tiene que garantizar la integridad de las comunicaciones entre los drones. El sistema tiene que comprobar que cada vez que recibe un fichero de datos, éste sea el original y no haya sido manipulado o se haya corrompido. |

*Tabla 29. RNF02 Integridad*

| | |
|---|---|
| **Identificador:** RNF03 | |
| **Nombre:** | Cifrado |
| **Prioridad:** Media | |
| **Necesidad:** Opcional | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | Los datos comunicados entre los drones deben estar cifrados para garantizar la seguridad de las comunicaciones y dificultar el éxito de ataques sobre las mismas. |

*Tabla 30. RNF03 Cifrado*

### A.2.3 Restricciones

| | |
|---|---|
| **Identificador:** RNF04 | |
| **Nombre:** | Escalabilidad |
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |

| | |
|---|---|
| **Descripción:** | El sistema debe ser escalable, de manera que pueda variar el número de drones que formen el enjambre. |

*Tabla 31. RNF04 Escalabilidad*

| | | |
|---|---|---|
| **Identificador:** RNF05 | | |
| **Nombre:** | Compatibilidad con PX4 | |
| **Prioridad:** Alta | | |
| **Necesidad:** Obligatorio | | |
| **Claridad:** Alta | | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre | |
| **Descripción:** | El sistema debe ser compatible con el controlador de vuelo PX4. | |

*Tabla 32. RNF05 Compatibilidad con PX4*

| | | |
|---|---|---|
| **Identificador:** RNF06 | | |
| **Nombre:** | Comunicación vía MAVSDK | |
| **Prioridad:** Media | | |
| **Necesidad:** Opcional | | |
| **Claridad:** Alta | | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre | |
| **Descripción:** | El sistema debe realizar la comunicación entre los drones mediante MAVSDK. | |

*Tabla 33. RNF06 Comunicación vía MAVSDK*

| | |
|---|---|
| **Identificador:** RNF07 | |
| **Nombre:** | Compatibilidad de coordenadas |

| | |
|---|---|
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | El sistema debe ser compatible con las coordenadas GPS (latitud y longitud). |

*Tabla 34. RNF07 Compatibilidad de coordenadas*

| | |
|---|---|
| **Identificador:** RNF08 | |
| **Nombre:** | Cualquier escenario |
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | El sistema debe funcionar en cualquier escenario. |

*Tabla 35. RNF08 Cualquier escenario*

### A.2.4 Fiabilidad

| | |
|---|---|
| **Identificador:** RNF09 | |
| **Nombre:** | Fiabilidad |
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |

| | |
|---|---|
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | La comunicación entre los drones tiene que ser fiable, reduciendo la tasa de fallos. La probabilidad de fallo deberá ser inferior al 0,1%. |

*Tabla 36. RNF09 Fiabilidad*

### A.2.5 Rendimiento

| | |
|---|---|
| **Identificador:** RNF10 | |
| **Nombre:** | Drones soportados por el sistema |
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | El sistema debe soportar como mínimo un enjambre formado por 6 drones. |

*Tabla 37. RNF10 Drones soportados por el sistema*

| | |
|---|---|
| **Identificador:** RNF11 | |
| **Nombre:** | Tiempos de carga |
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |

| | |
|---|---|
| **Descripción:** | La carga del sistema y del entorno de simulación no debe ser superior a 1 minuto en un equipo de gama alta. |

*Tabla 38. RNF11 Tiempos de carga*

| | | |
|---|---|---|
| **Identificador:** RNF12 | | |
| **Nombre:** | Calidad de los gráficos | |
| **Prioridad:** Alta | | |
| **Necesidad:** Obligatorio | | |
| **Claridad:** Alta | | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre | |
| **Descripción:** | El entorno de visualización debe ofrecer una calidad mínima de 480p. | |

*Tabla 39. RNF12 Calidad de los gráficos*

### A.2.6 Mantenibilidad

| | | |
|---|---|---|
| **Identificador:** RNF13 | | |
| **Nombre:** | Mantenimiento del sistema | |
| **Prioridad:** Alta | | |
| **Necesidad:** Obligatorio | | |
| **Claridad:** Alta | | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre | |
| **Descripción:** | El sistema deberá mantenerse de manera constante a lo largo del tiempo, asumiendo posibles problemas que puedan surgir y plantear soluciones de prevención de los mismos. Ofrecer un mantenimiento adecuado para que el sistema pueda seguir en funcionamiento a largo plazo. | |

*Tabla 40. RNF13 Mantenimiento del sistema*

| **Identificador:** RNF14 | |
|---|---|
| **Nombre:** | Estado del sistema |
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | Dado el gran volumen de datos que maneja el sistema, se generarán "logs" del estado de todos los integrantes del mismo. |

*Tabla 41. RNF14 Estado del sistema*

### A.2.7 Portabilidad

| **Identificador:** RNF15 | |
|---|---|
| **Nombre:** | Lenguaje de programación |
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | El sistema se implementará en Python debido a que es un lenguaje de más alto nivel que C++, la otra posibilidad que nos ofrece MAVSDK. |

*Tabla 42. RNF15 Lenguaje de programación*

### A.2.8 Interfaz

| | |
|---|---|
| **Identificador:** RNF16 | |
| **Nombre:** | Interfaz |
| **Prioridad:** Media | |
| **Necesidad:** Opcional | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | Se desarrollará una interfaz intuitiva que permita modificar fácilmente la configuración de cada dron. |

*Tabla 43. RNF16 Interfaz*

| | |
|---|---|
| **Identificador:** RNF17 | |
| **Nombre:** | Multi idioma |
| **Prioridad:** Media | |
| **Necesidad:** Opcional | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | Esta interfaz podrá visualizarse en varios idiomas entre los cuales se encontrarán: español e inglés. |

*Tabla 44. RNF17 Multi idioma*

### A.2.9Legislativos

| |
|---|
| **Identificador:** RNF18 |

| | |
|---|---|
| **Nombre:** | Open source |
| **Prioridad:** Alta | |
| **Necesidad:** Obligatorio | |
| **Claridad:** Alta | **Verificabilidad:** Alta |
| **Estabilidad:** | Siempre |
| **Descripción:** | El sistema será open source, es decir, será publicado para que cualquier usuario pueda beneficiarse del mismo, así como ayudar a su mejora y actualización. |

*Tabla 45. RNF18 Open source*