UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DISCA
DEPARTAMENTO DE INFORMÁTICA DE SISTEMAS Y COMPUTADORES

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Dpto. de Informática de Sistemas y Computadores

Gestión eficiente de conflictos entre aeronaves no tripuladas mediante mecanismos de atracción y respulsión direccionales

Trabajo Fin de Máster

Máster Universitario en Ingeniería de Computadores y Redes

AUTOR/A: Arenillas Pozas, Julián

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

Cotutor/a: Hernández Orallo, Enrique

Director/a Experimental: WUBBEN, JAMIE

CURSO ACADÉMICO: 2022/2023

Abstract

In recent years, unmanned aerial vehicles (UAVs) or drones have demonstrated their versatility and utility in various sectors of society. However, securing these devices in urban environments presents significant challenges as the number of aircraft sharing the same airspace increases. To deal with this matter, virtual force fields have been proposed as an efficient tool for conflict management. This research proposes the development of an aerial protocol based on directional force fields (D-FFP), which utilizes repulsion and attraction vectors to resolve conflicts among multiple drones. This protocol is compared to a previous version (FFP), and the improvements obtained compared to the standard force field approach are evaluated. The results, obtained through the execution of representative scenarios in a realistic simulator, demonstrate an average reduction of 69,1 % in time overhead while maintaining a minimum safety distance between drones.

Keywords: UAVs, conflict management, U-Space, directional force field, attraction/repulsion

Resumen

En los últimos años, los vehículos aéreos no tripulados (VANTs o UAVs) han demostrado su versatilidad y utilidad en diversos ámbitos de la sociedad. Sin embargo, la integración segura de estos dispositivos en entornos urbanos plantea desafíos significativos, a medida que aumenta el número de aeronaves que comparten el mismo espacio aéreo. Para abordar esta problemática, se ha propuesto el uso de campos de fuerza virtuales como herramienta eficiente de gestión de conflictos. En esta investigación, se propone el desarrollo de un protocolo aéreo basado en campos de fuerza direccionales (D-FFP), que utiliza vectores de repulsión y atracción para resolver conflictos surgidos entre múltiples drones. Este protocolo se compara con una versión anterior (FFP), y se evalúan las mejoras obtenidas en comparación con dicho enfoque estándar de campo de fuerza. Los resultados obtenidos, mediante la ejecución de escenarios representativos en un simulador realista, demuestran una reducción del 69,1 % en la sobrecarga del tiempo (en promedio), manteniendo en todo momento una distancia de seguridad mínima entre drones.

Palabras clave: UAVs, gestión de conflictos, U-Space, campo de fuerza direccional, atracción/repulsión

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

En la última década, los Vehículos Aéreos No Tripulados (VANTs) han aportado significantes mejoras a diversos sectores de la sociedad, desde la fotografía y el entretenimiento hasta la logística y agricultura. Coloquialmente conocidos como drones, este tipo de vehículos ha tenido una constante perspectiva militar en el pasado, y gradualmente se ha enfocado su uso para facilitar tareas cotidianas que puedan aportar a la comunidad, desde transporte de recursos hasta misiones de rescate forestal.

Su uso en entornos urbanos plantea nuevos problemas, ya que procede garantizar una integración segura y fiable de los drones respecto a los distintos elementos que puedan compartir el mismo espacio aéreo. De aquí surge nuevas infraestructuras y regulaciones que permitan la gestión, seguridad y privacidad de aeronaves no tripuladas con el resto de la población. Concretamente, en Europa se está integrando el proyecto U-Space, que intenta lograr la coexistencia y cooperación de las aeronaves dentro de la totalidad del espacio aéreo de una población.

Uno de los principales problemas a resolver de este proyecto es la gestión de conflictos entre varios drones que pueda surgir cuando estos mismos se acerquen entre sí al realizar misiones con trayectorias similares. Para resolver este tipo de problemas, existen distintos enfoques que eviten colisiones desafortunadas, como los campos de fuerza, metodología clave para ofrecer una evasión entre varios drones haciendo el uso de vectores de repulsión, siendo esta una de las temáticas principales de esta memoria.

En esta memoria se desarrollará un novedoso protocolo aéreo, que se basará en uno anterior llamado *Force Field Protocol* (FFP) [1], desarrollado por el Grupo de Redes de Computadoras (GRC) de la Universitat Politècnica de

València (UPV), que presenta un campo de fuerza omnidireccional para la repulsión; el objetivo del nuevo protocolo a desarrollar será transformar la propiedad de este campo de cara a lograr direccionalidad, con la intención de lograr una mejoría de prestaciones. Además, se utilizará como referencia una versión preliminar del campo de fuerza direccional implementado en el simulador MATLAB [2] de cara a obtener unos parámetros de partida.

Este tipo de simuladores permiten el desarrollo y ejecución eficiente de algoritmos matemáticos que pueda presentar distintos proyectos, pero a menudo se suele omitir la complejidad real de un sistema físico, como la inercia o el retardo en la comunicación. Es por eso mismo que, para lograr una experimentación y validez de los resultados que deseamos obtener, utilizaremos la herramienta de simulación ArduSim [3], perteneciente también al GRC, que permite la emulación virtual de misiones de vuelo acomodadas al gusto del usuario, teniendo en cuenta parámetros físicos y características reales que alteran el resultado si se ejecutará en un entorno real. Además, el código desarrollado para ArduSim se podrá ejecutar directamente en drones reales.

Así pues, el trabajo a realizar en esta memoria será un protocolo aéreo basado en campos de fuerza direccionales con uso de vectores de repulsión y atracción que permita lograr una resolución de conflictos simple y escalable a varios drones, a la que denominaremos *Directional-Force Field Protocol* (D-FFP).

1.1. Objetivos

La principal intención de esta memoria es detallar el proceso de creación de un protocolo aéreo enfocado a la gestión de conflictos entre aeronaves, haciendo el uso de la metodología de campos de fuerza con propiedad direccional; concretamente, se calculan vectores de atracción y repulsión, cuya fuerza combinada permite alcanzar su destino y, a la vez, evadir cualquier obstáculo estático o dinámico que esté presente en su recorrido. Para lograr este propósito, se deberá seguir una serie de metas para cumplir con el desarrollo eficiente del proyecto:

- Estudio y comprensión del protocolo anteriormente creado, FFP, del que parte este proyecto, y la implementación y simulación preliminar del protocolo D-FFP utilizando MATLAB.
- Comprensión del funcionamiento y del código interno del simulador ArduSim [3].

- Implementación y desarrollo del protocolo D-FFP, además de la sintonización de parámetros para ajustar las propiedades direccionales.
- Experimentación de distintos escenarios propuestos para representar casos de conflicto entre VANTs, y comparación de datos respecto a los trabajos anteriores.

1.2. Estructura de la memoria

Esta memoria está organizada en los capítulos que se detallan a continuación:

- **Capítulo 2. Estado del arte.** Presentamos una vista general de distintas publicaciones que pueden ayudar a entender mejor la temática que queremos abordar, mostrando un enfoque hacia las aeronaves no tripuladas, la gestión de conflictos, y los campos de fuerza.
- **Capítulo 3. Bases del proyecto.** Mostramos las diferentes herramientas elegidas para desarrollar este proyecto, como el simulador ArduSim [3] anteriormente mencionado, además de los distintos desarrollos que sirven de punto de partida para este trabajo, para tener una mejor idea de los avances a lograr.
- **Capítulo 4. Diseño y desarrollo del problema.** Detallamos los pasos y características a tener en cuenta a la hora de realizar este proyecto, y desarrollamos el código necesario para poder realizar una implementación correcta y funcional.
- **Capítulo 5. Experimentos y pruebas finales.** Verificamos el correcto funcionamiento del proyecto en este capítulo, haciendo uso de distintos escenarios representativos y tablas con datos que respaldan nuestros resultados.
- **Capítulo 6. Conclusiones y trabajo futuro.** Acabamos la memoria explicando un resumen de lo logrado a lo largo de este proyecto, y posibles trabajos futuros que puedan partir de este proyecto.

Capítulo 2

Estado del arte

En el ámbito de la aviación, las aeronaves no tripuladas, coloquialmente conocidas como drones, han experimentado un auge significativo de su uso, aumentando el número de estos dispositivos que deben compartir un mismo espacio aéreo. Esto plantea nuevos problemas, como la gestión segura y eficiente de posibles colisiones entre ellos, donde una posible estrategia a utilizar es el uso de campos de fuerza virtuales. Estos campos utilizan principios físicos y matemáticos para generar repulsiones entre los distintos drones ante un peligro de colisión. En este capítulo, comentaremos distintos enfoques dentro de la gestión de colisiones, detallando distintos tipos de estrategias utilizadas en este campo, y profundizando un poco más en desarrollos que utilizan campos de fuerza como técnica de evasión; de esta forma lograremos conocer mejor el contexto detrás de este proyecto.

El campo de los sistemas de anticolisión ha sido objeto de estudio para todo tipo de vehículos, no solo aeronaves. Se han propuesto múltiples soluciones y enfoques para abordar este problema, los cuales se suelen dividir en dos categorías [4]: la gestión estratégica, y la gestión táctica de conflictos. La primera trata de analizar y estudiar todos los distintos obstáculos que puedan alterar la viabilidad del vuelo y la trayectoria del dron antes de despegar, modificando la misión aérea para evadir dichos obstáculos. Por otra parte, la segunda trata de realizar lo mismo, pero a tiempo real, mientras ya vuelan dichos drones, modificando la trayectoria de vuelo en el momento que se detecta una posible colisión. Nuestro nuevo protocolo a desarrollar entraría en esta segunda categoría.

Entrando en más detalle sobre la metodología usada, el siguiente artículo [5] presenta una clasificación de las distintas estrategias de evitación de colisiones, donde se incluyen las siguientes categorías: enfoques geométricos, la

generación de trayectorias de escape optimizadas, la implementación de sistemas de detección y evitación, y el uso de campos de fuerza. Adelantándonos un poco, nuestro protocolo usa esta última metodología.

Empezando por la primera estrategia, los enfoques geométricos estudian varios aspectos en la información del dron y del obstáculo, como la posición, velocidad, y/o trayectoria, de forma que se pueda reajustar la trayectoria de dicho dron para evadir el obstáculo correctamente. Cuando se detecta una colisión, el dron reajusta la trayectoria intentando disminuir la desviación resultante, consultando la información geométrica disponible en el dron y el objeto a colisionar. Park et al. [6] presentan un sistema de Vigilancia Dependiente Automática (ADS-B), donde, calculando el punto mínimo que puedan alcanzar dos *Unmanned Aerial Vehicles* (UAVs) al acercarse entre sí, se evalúa la peor condición de colisión entre estos dos y se resuelve el conflicto que pueda ocurrir.

Pasando ahora a las trayectorias de escape optimizadas, estas estrategias consideran los conflictos como problemas que requieren el uso de la optimización para alcanzar un resultado funcional. En esta estrategia entran como datos clave el uso de información geográfica, y las posiciones y tamaño de los obstáculos, al igual que en el enfoque geométrico, pero a la hora de recalcular la maniobra de evasión, se decide elegir la maniobra menos agresiva que ofrezca una protección adecuada [7]. Esto se debe a la limitada capacidad de computación que pueda tener un *Unmanned Aerial Vehicle* (UAV), ya que no siempre se puede dotar a una aeronave de componentes potentes por problemas de precio, aumento de peso, aumento del consumo, y reducción de la batería. Aquí es donde la optimización gana un importante papel a la hora de gestionar los posibles conflictos y sus evasiones. El artículo anteriormente mencionado [5] define varios enfoques dentro de esta estrategia como, por ejemplo, algoritmos inspirados en el comportamiento de las hormigas, la optimización bayesiana, y la optimización por enjambre de partículas. En este ámbito, Pérez-Carabaza et al. [8] definen un algoritmo de búsqueda de tiempo mínimo inspirado en la conducta de búsqueda de alimento de las hormigas, implementada en drones para establecer una ruta libre de colisiones, y manteniendo a la vez una comunicación estable con la estación de control terrestre.

Continuando con los sistemas de detección y evitación, generalmente se utilizan sensores implantados en los UAVs que facilitan la detección al momento de enfrentarse a posibles obstáculos en el recorrido. El uso de estos sensores aligera considerablemente la carga de computación necesaria para

detectarlos, cosa que mejora notablemente los tiempos de respuesta. Esta estrategia trabaja muy bien con obstáculos que puedan estar en movimiento sin conocer las trayectorias o rutas de vuelo que puedan tener. Wang et al. [9] implementan un sensor LiDAR utilizado en un algoritmo de detección de obstáculos, con el posible seguimiento de la trayectoria y velocidad si presentan movimiento. Además, la incorporación de este sensor al sistema permite reducir la carga de memoria necesaria para el correcto funcionamiento del algoritmo implementado. El uso de sensores LiDAR es bastante común en este tipo de sistemas gracias a su alta precisión; no obstante, estos sensores pueden presentar un precio un poco elevado. Sensores de proximidad vía ultrasonidos entran en juego para sustituir a los sensores anteriores, con un precio más reducido, y siendo más accesibles; sin embargo, se requiere un manejo especial de los datos recogidos por estos sensores. Balemans et al. [10] presentan un sistema capaz de predecir datos que ofrece un sensor LiDAR, reemplazándolo con un sensor de ultrasonidos, y utilizando un autocodificador convolucional apilado.

Por último, en esta lista, las estrategias de uso de campo de fuerza, o campos potenciales artificiales, aplica un enfoque de partículas cargadas que se repelen entre sí, imitando la naturaleza de dos polos iguales. Utilizando fuerzas de atracción y repulsión, el dron que aplique esta estrategia pueden ser atraído a distintos puntos en un espacio aéreo, mientras es repelido por objetos que puedan causar una colisión. Estos obstáculos pueden tener una naturaleza estática o dinámica, es decir, pueden permanecer inmóviles o en constante movimiento en el mismo espacio aéreo que el dron, respectivamente. Los entornos estáticos presentan menos complejidad respecto a los dinámicos, ya que estos últimos necesitan una mejor precisión en la información geométrica de los drones, y en la dirección que presentan en el movimiento. Si analizamos la literatura alrededor de este campo podemos observar que, aparte de no haber mucha cantidad de trabajos al respecto, la mayoría de autores consideran entornos estáticos en sus soluciones, como [11, 12, 13, 14]. En nuestro proyecto vamos a suponer que los objetos a los que nos enfrentamos están en constante movimiento, presentando un entorno dinámico. Es por eso que, trabajos como el de Choi et al. [15] son más interesantes en este estudio donde, considerando tanto objetos estáticos como dinámicos a evitar, utilizan un campo vectorial independiente del operador rotacional, y reducen el problema del mínimo local existente en obstáculos estáticos. Kownacki et al. [16] hacen uso de UAVs no holonómicos, como los drones de ala fija, para implementar un campo de fuerza que repele los obstáculos desde las zonas situadas detrás y delante de este, a lo largo de una línea definida por el vector velocidad del obstáculo. Burgos et al. [17] implementan una metodología de

campo de flujo potencial para el desarrollo de su campo de fuerza, cambiando el ángulo de rumbo de la aeronave respecto al vector velocidad resultante.

De este modo, el GRC propuso en el pasado el protocolo FFP [1], que utiliza un campo de fuerza omnidireccional, mostrando como las colisiones pueden ser evitadas con una pequeña sobrecarga temporal. Utilizando vectores de atracción y repulsión, las aeronaves consiguen evitarse entre sí, actuando como dos cargas eléctricas iguales. Sin embargo, no se tuvo en cuenta factores como la dirección del obstáculo con respecto al rumbo de la aeronave principal. Esto implica que la magnitud en la fuerza de repulsión de los drones era la misma, independientemente de si estaban situados uno delante del otro, o si, por ejemplo, viajaban en trayectorias paralelas sin causar peligro. En situaciones donde ambos drones mantienen una cierta distancia, y sus trayectorias no presentan una intersección, llegarían a alejarse una distancia innecesaria frente a obstáculos que no necesitan una evasión tan drástica. De aquí nace el protocolo D-FFP, que se encargará de trasladar esta omnidireccionalidad del anterior protocolo a un campo de fuerza direccional, evitando colisiones entre drones, al mismo tiempo que se reduce al mínimo posible la trayectoria de evasión realizada, reduciendo pues el tiempo total de la misión, y aumentando la autonomía del dron en el aire.

Capítulo 3

Bases del proyecto

Para contextualizar mejor al lector, se comentarán las diferentes herramientas y proyectos anteriores que forman las bases de este proyecto, empezando por el simulador de vuelo ArduSim proporcionado por el GRC, detallando el protocolo anterior del método de campo de fuerza a desarrollar y, por último, una versión de esta implementada en MATLAB y tomada como referencia.

3.1. ArduSim

En casi todos los artículos mencionados en el estado del arte, MATLAB [2] es utilizado frecuentemente como herramienta para realizar experimentos de validez. Con este tipo de simuladores se pueden ejecutar grandes tandas de experimentos de forma rápida y eficiente, pero se suelen omitir complejidades reales de un sistema físico, como por ejemplo la inercia. La herramienta ArduSim [3] ofrece al usuario la capacidad de realizar varias ejecuciones a tiempo real de una misión aérea configurada para realizar una tarea en específico. Estas misiones pueden tener múltiples UAVs en funcionamiento simultáneamente, ligado este número a las capacidades y rendimiento del ordenador donde se ejecute. Respecto a las comunicaciones, ArduSim simula una red ad-hoc inalámbrica que permite el paso de mensajes entre los distintos drones en funcionamiento. Como información adicional, ArduSim genera la trayectoria que debe seguir cada UAV en formato OMNeT++ y NS2, permitiendo su simulación o incluso su ejecución en un dron multirrotor físico.

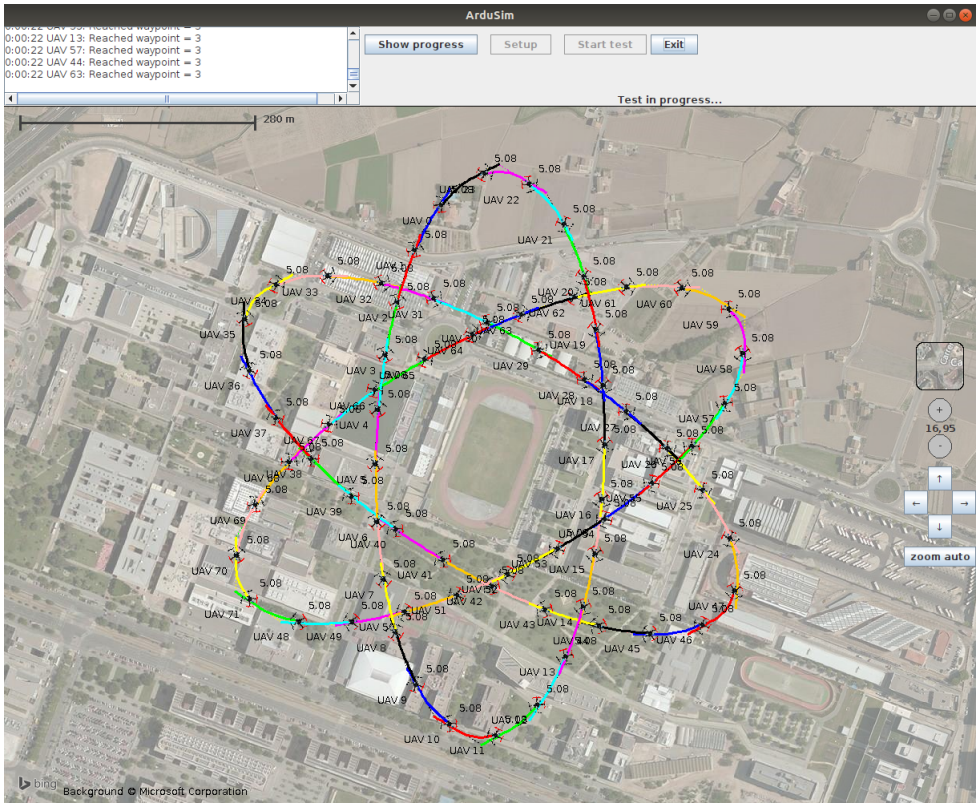


Figura 3.1: Ejecución visual en ArduSim.

Esta herramienta está a disposición del público en GitHub [18]. Este repositorio ofrece un manual donde se explica el funcionamiento del simulador, su instalación y ejecución en una máquina personal, la implementación en un dron real y incluso el como programar tus propios protocolos aéreos. A continuación, explicaremos resumidamente el comportamiento del simulador desde el enfoque de la arquitectura, su capa física y los distintos protocolos aéreos implementados y ofrecidos al usuario.

3.1.1. Arquitectura

SITL, o *Software In The Loop*, es el enfoque utilizado en la aplicación ArduCopter [19], perteneciente a ArduPilot [20], como módulo de desarrollo básico para simular un gran número de UAVs dentro de ArduSim. Esta herramienta contiene código que permite al UAV asemejarse a uno real, simulando sus propiedades físicas de vuelo con una buena precisión. Cada UAV virtual simulado ejecuta su propia instancia SITL debido a su limitación de simulación a un solo proceso por dron. Esto lo convierte en una herramienta

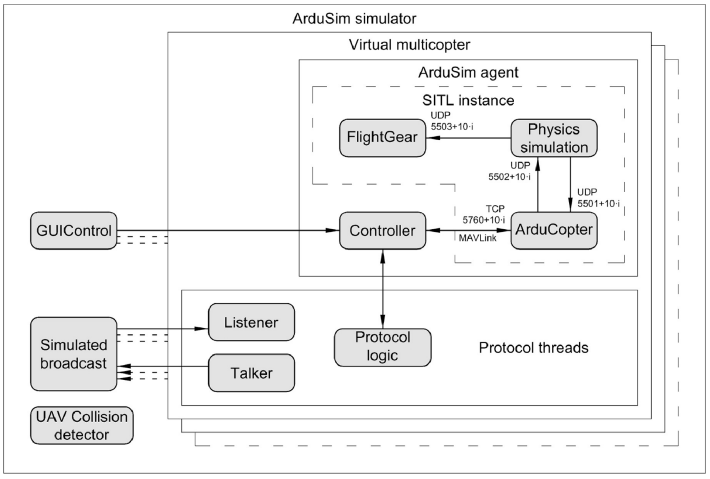


Figura 3.2: Arquitectura de ArduSim.

no adecuada para implementar protocolos de comunicación entre los drones.

Por esta razón, en la figura 3.2 se detalla una arquitectura de simulación multiagente que implementa una lógica de control de alto nivel sobre SITL. Mediante la interfaz gráfica *GUIControl*, ArduSim permite configurar los UAVs y realizar los experimentos de forma directa (también permitida a través de una línea de comandos). Además, se incluyen varias características de difusión de mensajes entre drones con *Simulated broadcast*, y la detección de posibles colisiones con *UAV Collision detector*.

Cada dron virtual está compuesto por un agente que se encarga de controlar los diferentes hilos requeridos. Varios de estos hilos son el *Listener* y el *Talker*, procesos importantes para el envío de paquetes de datos y la recepción de estos, respectivamente. Adicionalmente, se presenta un hilo adicional *Protocol logic* que comanda al UAV teniendo en cuenta el funcionamiento del protocolo ejecutado en la simulación, así como las comunicaciones de otros drones circundantes.

Por último, un agente ArduSim contiene a su vez una instancia SITL y un hilo *Controller*, encargados de enviar mensajes al dron y recibir la información que genera para su posterior manejo. Esta instancia SITL ofrece un identificador de 8 bits para cada dron, lo que significa el número de drones virtuales de la simulación se ve limitada a un máximo de 256 UAVs. Sin embargo, en ArduSim pueden crearse varias instancias SITL, permitiendo exceder el número de drones en el aire para grandes pruebas.

3.1.2. Protocolos aéreos

Como se ha comentado anteriormente, el simulador permite la propia programación e implementación de protocolos aéreos que se ajusten para realizar diversas tareas específicas. ArduSim ofrece de por sí varios protocolos a utilizar, donde cada uno es independiente del otro. A continuación, se comenta el funcionamiento de estos, dejando para el final el protocolo en el que se basará este proyecto, y que se explicara con más detalle en el siguiente apartado:

- *Mission*: Se selecciona una ruta planificada y un grupo de drones la siguen.
- *MBCAP*: Se realizan evasiones entre un grupo de drones que siguen una ruta planificada.
- *MUSCOP*: Un dron maestro realiza una misión mientras un enjambre mantiene una formación y sigue al maestro, con conocimiento de dicha misión, despegando todos los drones de forma segura hasta alcanzar la formación en el aire.
- *Follow Me*: Un enjambre de drones sigue a un dron maestro controlado de forma manual por un piloto.
- *Vision*: Un dron singular equipado con una cámara utiliza datos de visualización para aterrizar en un marcador visual Aruco-marker [21].
- *shakeup*: Permite a un enjambre reconfigurar su formación en vuelo evitando colisiones.
- *compareTakeOff*: Protocolo que compara distintos algoritmos de asignación para despegar de forma segura.
- *magnetics*: Se implementa un campo de fuerza omnidireccional para evitar colisiones en vuelo entre varios drones.

Este último protocolo ha sido formalmente presentando como *Force Field Protocol* (FFP), al que detallaremos mejor en el siguiente apartado.

3.2. Protocolo FFP

Como bien se ha ido comentando a lo largo de la memoria, el protocolo FFP [1] toma un importante papel en este proyecto, siendo el protocolo base de todo el trabajo a desarrollar. Su concepto principal es la generación de

un campo de fuerza alrededor de un dron, que es compartida con el resto de drones. Utilizando mensajes inalámbricos periódicos, todos los drones realizan una difusión de su propia posición y la fuerza de repulsión de su campo cada 200 milisegundos, valor que ofrece suficiente resistencia a las pérdidas de señal mientras se evita ocupar el canal de comunicación de forma excesiva. Esta comunicación se basa en el estándar 802.11ac, basándose en comunicación Wi-Fi ad-hoc en la banda de 5 GHz.

Estos campos de fuerza no solo se aplican a los UAVs, sino que también podemos aplicar estas propiedades a diferentes puntos del mapa, ya sea para simular objetos estáticos que generen una repulsión, o bien objetivos que deban pasar los drones generando una fuerza de atracción.

Algorithm 1 FFP:collisionAvoidance()

Require: $targetLocation, locationObstacles$

```
while !targetReached do
    Vector attraction = getAttractionVector()
    Vector repulsion = getRepulsionVector()
    Vector resulting = Vector.add(attraction, 2*repulsion)
    resulting = resulting.scalarProduct(maxSpeed)
    resulting = reduceToMaxSpeed(resulting)
    moveUAV(resulting)
    if distance(UAV,target) ≤ 1 then
        targetReached = true
    end if
end while
```

En el algoritmo 1, se representa el algoritmo principal usado en este protocolo. Asumiendo que los drones tienen planificadas unas rutas con distintos objetivos a sobrepasar y que permanecen en el aire, el algoritmo primero calcula las fuerzas de atracción de dichos objetivos. Después, se calcula el vector de repulsión afectado por los distintos obstáculos estáticos o dinámicos presentes en el espacio aéreo. Al vector resultante se le suma ambos vectores, lo cual permitirá determinar la trayectoria y velocidad que debe seguir el UAV. El vector repulsión es multiplicado por dos para que predomine sobre el vector atracción, evitando el peor caso donde ambos vectores tengan el mismo valor, manteniendo quieto el UAV e impidiéndole seguir su rumbo. Para asegurarnos de que el UAV no vaya más rápido que la velocidad máxima (establecida por el usuario), se reduce la longitud del vector resultante en caso de que se supere. Finalmente, este vector se pasa al controlador de vuelo para asegurarse de que dicho UAV vuele en la dirección prevista. Todo este

proceso se repite hasta que el dron haya completado su trayectoria entera. Si la misión presenta varios objetivos o puntos a los que volar, la trayectoria del dron es reemplazada por el siguiente punto una vez haya cumplido con el anterior.

Algorithm 2 FFP:getAttractionVector()

Require: $UAVlocation, targetLocation$

```
1: Vector attraction = targetLocation - UAVLocation
2: attraction.normalize()
3: β = attractionFunction(distance(UAVLocation, targetLocation))
4: attraction.scalarProduct(β)
5: return attraction
```

Algorithm 3 FFP:getRepulsionVector()

Require: $UAVlocation, locationObstacles$

```
1: Vector totalRepulsion = new Vector();
2: for obstacle in obstaclesList do
3:     Vector repulsion = UAVLocation - obstacleLocation
4:     repulsion.normalize()
5:     γ = repulsionFunction(distance(UAVLocation, obstacleLocation)
6:     repulsion.scalarProduct(γ)
7:     totalRepulsion += repulsion
8: end for
9: return totalRepulsion
```

Respecto al cálculo del vector de atracción, podemos observar con mejor detalle la implementación en el algoritmo 2. Primero, se calcula un vector con la distancia entre la posición del UAV y la ubicación objetivo apuntando hacia este último, y se normaliza el valor para reducirlo a su unidad unitaria. Después, se aplica una función que afecta a este vector cuando se localiza cerca de la ubicación objetivo. Esto reduce la velocidad del dron a la mitad para evitar que pase volando más allá del punto.

El vector de repulsión es similar al cálculo del vector de atracción, pero con varias diferencias, tal y como se ilustra en el algoritmo 3. Una de ellas es el vector obtenido según la distancia, que es el mismo, pero debe apuntar hacia el dron. Igualmente, se debe tener en cuenta que puede existir más de un obstáculo, en cuyo caso se debe calcular cada vector de repulsión de

cada obstáculo sobre el dron, y acumularlo en un vector total. Al igual que el de atracción, también se le aplica una función para determinar la fuerza de este vector. Concretamente, debe alcanzar valores más altos cuanto más cerca esté del obstáculo a evadir:

$$Repulsion : \gamma = \begin{cases} 1 & \text{if } x \leq frd \\ \text{máx}\left(1 - \left(\frac{x-frd}{\alpha}\right)^2; 0\right) & \text{otherwise} \end{cases}$$

Usando esta ecuación, el UAV sufrirá repulsión del obstáculo al máximo valor cuando este esté a menor valor que la distancia de máxima de repulsión (variable "Full Repulsion Distance" , o frd). Cuando el obstáculo se vea alejado, la fuerza del vector será disminuida en base al valor α seleccionado.

Dicha función es la responsable de determinar la propiedad de omnidireccionalidad del campo de fuerza. Uno de los objetivos principales del desarrollo de nuestro proyecto es modificar la ecuación anterior para insertar la dirección y trayectoria del obstáculo como variable que modifique la fuerza de repulsión.

A continuación, mostramos como se ha logrado en una versión anterior implementada en MATLAB.

3.3. Implementación D-FFP en MATLAB

La versión anterior FFP proponía un enfoque de campo de fuerza con propiedades omnidireccionales de repulsión, simulando el comportamiento de dos cargas eléctricas similares. En esta versión se quiere variar este comportamiento, aplicando los principios de repulsión entre dos imanes con la misma polaridad, insertando como variable a la fuerza de repulsión la dirección (θ):

$$R(\theta, \mu) = \begin{cases} cos(\mu \cdot \theta) : \theta \in [-\frac{\pi}{2 \cdot \mu}, \frac{\pi}{2 \cdot \mu}] \\ C : \theta \notin [-\frac{\pi}{2 \cdot \mu}, \frac{\pi}{2 \cdot \mu}] \end{cases}$$

Ajustando los parámetros C y μ, se pueden modificar las propiedades del campo de fuerza que, concretamente, determinan el tamaño de la componente omnidireccional y la anchura del lóbulo principal, respectivamente.

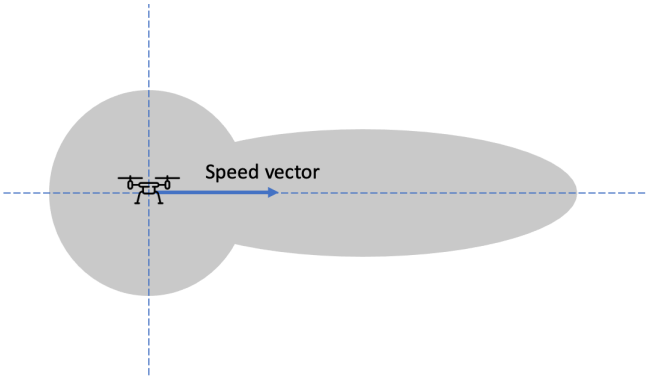


Figura 3.3: Patrón de repulsión D-FFP.

El patrón de repulsión que se genera con el uso de esta ecuación se puede observar en la figura 3.3, donde ángulos próximos a cero proporcionan un valor alto, y obstáculos que se encuentren en dicha línea marcada por el vector velocidad obtienen una mayor repulsión, a diferencia del resto que obtienen una repulsión de valor constante en el resto de direcciones.

Para el desarrollo de este proyecto, partiremos de esta ecuación, modificándola para añadir más funcionalidades, como la posibilidad de evadir obstáculos en el eje z, además de mejorar su implementación para tener en cuenta más complejidades de una situación real. En el capítulo de la experimentación, uno de los objetivos a cumplir es comparar los resultados que se ofrecen en ambas versiones (FFP y la implementación MATLAB de D-FFP) para determinar las mejoras obtenidas en nuestro proyecto respecto a anteriores trabajos. Ahora sí, procederemos a explicar en el siguiente capítulo el diseño y desarrollo del protocolo D-FFP, implementado en ArduSim.

Capítulo 4

Diseño y desarrollo del problema

Este capítulo se enfocará al análisis, diseño y desarrollo del protocolo aéreo que se pretende implementar. Concretamente, se comentarán los requisitos necesarios para utilizar las herramientas proporcionadas, las clases y el código desarrollado dentro del simulador ArduSim, y pruebas preliminares que confirman el funcionamiento del protocolo para asegurarse de que se realizan bien los experimentos propuestos en el capítulo siguiente.

4.1. Análisis del problema

Como bien se ha comentado a lo largo de esta memoria, el propósito de este trabajo es la creación de un nuevo protocolo aéreo tomando las bases construidas por versiones anteriores, e implementarlo dentro de un simulador de vuelo a tiempo real.

A partir de este objetivo, se puede dividir el plan de desarrollo en varias partes: estudio de las herramientas utilizadas para llevar a cabo el trabajo, programación del protocolo D-FFP, y ajustes de parámetros para afinar las posteriores pruebas.

- Empezando por las herramientas utilizadas, el lenguaje de programación a utilizar es Java (openJDK 17 [22]), utilizando IntelliJ [23] como plataforma de desarrollo. Esto surge de las recomendaciones de uso de ArduSim, al igual que el uso del sistema operativo Ubuntu [24](Ubuntu 22.04.2 LTS). Por último, se han utilizado herramientas como Google Earth [25] para la creación de misiones de vuelo, y scripts hechos en Python3 [26] para el ajuste de parámetros.

- Para la programación del protocolo, existe una guía en github de ArduSim [18] que explica paso a paso como estructurar las distintas clases dentro del simulador y sus finalidades. Nos basaremos en técnicas ya conocidas por las versiones anteriores, y se adaptará el código para su uso en ArduSim, a la misma vez que se insertarán nuevas funcionalidades como la evitación en el eje z, que permite al algoritmo observar y actuar en las tres dimensiones.

- Por último, habrá que afinar los distintos parámetros que forman la geometría de radiación del campo de fuerza para minimizar los movimientos de los drones lo máximo posible, sin violar la clausula de la distancia mínima de 10 metros que aplica ArduSim para detectar que ha ocurrido una posible colisión. Este ajuste lo logramos con los scripts escritos en Python mencionados anteriormente, ejecutando el simulador por línea de comando para realizar tandas de experimentos rápidamente, y encontrar la mejor combinación posible.

Una vez analizada la situación en la que nos encontramos, tenemos una clara idea de qué pasos realizar para completar el desarrollo del proyecto. En el siguiente apartado, se detallarán los requisitos necesarios para la ejecución de las herramientas empleadas en el proyecto.

4.2. Requisitos necesarios

La herramienta que más características de rendimiento necesita es el simulador de vuelo. Por lo tanto, siempre que se cumplan los requisitos mínimos de ArduSim, uno tendrá suficientes recursos para trabajar con el resto de herramientas utilizadas.

En el GitHub de ArduSim existe un apartado que comenta los requisitos mínimos y necesarios para utilizar este simulador:

Requisitos mínimos:

- Intel core i5 (versión de 4 núcleos)
- 6 GB de RAM
- S.O. Windows, Linux o MacOS
- Java SE 17
- Cygwin y ImDisk Virtual Disk Driver (si se usa Windows)

Requisitos recomendados:

- Intel core i7 (4 núcleos *Hyper-Threading*)
- 16 GB de RAM
- Linux (Ubuntu 22.04)
- Java SE 17

Como información adicional, este proyecto está desarrollado en un portátil Lenovo con un procesador i7 de onceava generación con 4 núcleos *Hyper-Thread*, 16 GB de RAM, sistema operativo Ubuntu 22.04.2 LTS y openJDK 17.

Observando los requisitos recomendados, nuestro entorno de trabajo cumple sobradamente con lo necesario para trabajar con este simulador.

4.3. Desarrollo de la solución

Procedemos ahora a uno de los hitos más importantes de esta memoria: el desarrollo del nuevo protocolo llamado *Directional-Force Field Protocol* (D-FFP), siguiendo los pasos de creación de un protocolo aéreo en ArduSim. Esta sección se divide en dos partes, la estructura que componen todas las clases relacionadas con la ejecución del nuevo protocolo, una explicación detallada de la clase que contiene el nuevo algoritmo, y cómo se ha realizado la automatización de ajuste de parámetros para elegir la mínima radiación posible emitida por el campo de fuerza, respetando la distancia mínima de 10 metros entre drones.

4.3.1. Estructuración de clases del protocolo

Antes de hacer hincapié en el desarrollo, hay que entender la estructura de clases dentro del código interno del simulador. Cada protocolo está organizado en tres carpetas con propósitos distintos: *gui*, *logic* y *pojo*.

- En la carpeta *gui* residen las distintas clases que ofrecerán una visualización gráfica de la configuración del protocolo dentro del simulador, e inicializaran los distintos parámetros que el usuario ha establecido para la misión.
- Las distintas clases y algoritmos necesarios para el correcto funcionamiento del protocolo residen en la carpeta *logic*, que, como bien refleja su nombre, contienen toda la lógica detrás del este.

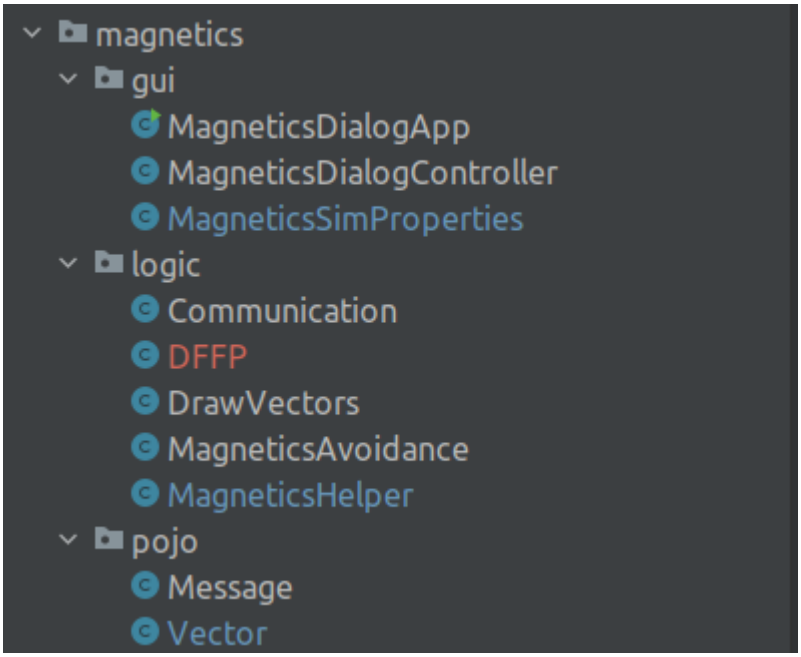


Figura 4.1: Distribución de las clases en ArduSim

- Por último, la carpeta *pojo* contiene distintas funcionalidades que sirven de apoyo para las clases de *logic*.

Partiendo del protocolo FFP ya implementado, nosotros hemos modificado y añadido clases dentro de su estructura. En la figura 4.1 podemos observar la distribución de clases del protocolo, siendo la clase resaltada en rojo el principal algoritmo D-FFP nuevo añadido. Las clases en blanco pertenecen al anterior protocolo, y las azules han sido modificadas para el comportamiento de la nueva versión.

La carpeta *gui* contiene tres clases para el control de la interfaz gráfica y la carga de parámetros:

- **MagneticsDialogApp**: Muestra al usuario la escena de configuración creada para que el usuario pueda modificar distintos parámetros a gusto de la misión a ejecutar.
- **MagneticsDialogController**: Carga las propiedades gráficas de un archivo .fxml que contiene el diseño de la ventana de configuración, posteriormente inicializada en *MagneticsDialogApp*.
- **MagneticsSimProperties**: Una vez se configuran los parámetros en la ventana gráfica, estos se almacenan con los nuevos valores para su posterior uso en el protocolo.

Esta última clase ha sido modificada para tener en cuenta los nuevos parámetros que se utilizaran en la versión D-FFP.

La carpeta *pojo* contiene dos clases objeto que se usarán para facilitar la programación:

- **Message**: Se encarga de convertir el formato de los datos enviados y/o recibidos por los distintos UAVs en vuelo. Concretamente, permite la conversión de datos que representan la posición 3D de un dron a un "JSONObject" para enviarlos como mensaje, o la inversa para recibirlos de otro dron.

- **Vector**: Permite la creación de vectores y su llamada a distintas operaciones matemáticas para su modificación, como la multiplicación escalar o la normalización del vector. Se ha modificado esta clase para incluir las operaciones del producto escalar y el producto vectorial, que se utilizarán en los cálculos del algoritmo D-FFP.

Por último, en la estructura, la carpeta logic contiene distintas clases utilizadas para el funcionamiento del protocolo D-FFP:

- **Communication**: Esta clase es la que se encarga de establecer las comunicaciones entre los distintos drones ejecutados en una misión. Esta comunicación se basa en el envío y recepción de la posición de cada uno de ellos, donde se utiliza el formato "JSONObject" para los mensajes, utilizando la clase *Message* para facilitar y agilizar el proceso de conversación.

- **DFFP**: Aquí reside todo el funcionamiento del algoritmo que se ha desarrollado como propósito principal de esta memoria. Se procederá a su explicación en el siguiente apartado.

- **DrawVectors**: Para facilitar la visualización de las fuerzas de los vectores de atracción y repulsión, esta clase ofrece la posibilidad de mostrar dichos vectores en una misión, dando un aspecto visual al usuario de como actúa el campo de fuerza.

- **MagneticsAvoidance**: Perteneciente al protocolo FFP, este es el algoritmo que desarrolla la idea explicada en el capítulo 3.2. No se utiliza esta clase para el desarrollo del protocolo D-FFP, pero se mantiene su uso por si el usuario quiere revertir el comportamiento, como haremos en la experimentación para comparar resultados.

- **MagneticsHelper**: Esta clase es la que se encarga de que los drones ejecuten el protocolo deseado. Se ha modificado para cambiar el algoritmo y comportamiento del protocolo FFP al D-FFP.

4.3.2. Protocolo *Directional-Force Field Protocol* (D-FFP)

La clase D-FFP actúa como un hilo que se ejecuta de manera independiente entre cada UAV que ejecuta este protocolo, utilizando como método de conversación el paso de mensajes entre cada uno para conocer la localización puntual de cada dron.

```
1  @Override
2  public void run(){
3      takeoff();
4      communication.start();
5      long start = System.currentTimeMillis();
6      while (waypoints.size() > 0) {
7          while (!waypointReached()) {
8              updateVirtualLimits();
9              Vector attraction = getAttractionVector();
10             Vector totalRepulsion = getRepulsionVector();
11             Vector resulting = Vector.add(attraction,
                   totalRepulsion);
12             moveUAV(resulting);
13             API.getArduSim().sleep(200);
14         }
15         waypoints.poll();
16     }
17     long protocolTime = System.currentTimeMillis() - start;
18     communication.stopCommunication();
19     land();
20     saveData(protocolTime);
21 }
```

Figura 4.2: Función *run()* en DFFP.java

El código mostrado en la figura 4.2 es el principal algoritmo a ejecutar. Como primero paso, los drones despegarán de sus posiciones iniciales que ha creado el usuario en el fichero de la misión, después se iniciarán las comunicaciones y se creará una variable para cronometrar el tiempo de ejecución. El algoritmo de evasión funcionará hasta que se cumplan todos los objetivos a sobrepasar que residen en el archivo de la misión. Como bien se indica en el primer y segundo bucle, se ejecutará el algoritmo hasta alcanzar el objetivo

marcado y, una vez cumplido, este se verá eliminado de la lista de objetivos (*waypoints.poll()*) para dar paso al siguiente hasta que todos hayan sido sobrevolados. Esto marcará el fin del protocolo, al que dará paso al cronometraje del tiempo total, el paro de las comunicaciones, y el aterrizaje de los drones en el último objetivo alcanzado, seguido de un guardado en un archivo aparte que ofrece información variada sobre la misión ejecutada.

Entrando en el segundo bucle, donde reside la funcionalidad del algoritmo, primero se realiza el cálculo de límites que se establecerán para dar lugar a unos bordes virtuales que definen el espacio de evasión en el eje z. Dicho código es mostrado en la figura 4.3, con un esquema del principio en la figura 4.4.

```
1  private void setUpVirtualLimits(){
2      targetAltitude = waypoints.peek().z;
3
4      double cA = copter.getAltitude();
5      double tA = targetAltitude;
6
7      double vF = virtualFloor;
8      double vC = virtualCeiling;
9
10     boolean belowTarget = cA < tA - 3;
11     boolean aboveTarget = cA > tA + 3;
12
13     if(belowTarget){
14         vF = Math.max(vF, cA - 10);
15         vC = tA + 10;
16     } else if(aboveTarget){
17         vF = tA - 10;
18         vC = Math.min(vC, cA + 10);
19     } else {
20         vF = tA - 10;
21         vC = tA + 10;
22     }
23
24     vF = Math.max(10, vF);
25     vC = Math.min(120, vC);
26
27     virtualCeiling = vC;
28     virtualFloor = vF;
29 }
```

Figura 4.3: Función *setUpVirtualLimits()* en DFFP.java

La idea de estos límites es prohibir al dron aumentar o disminuir su altura

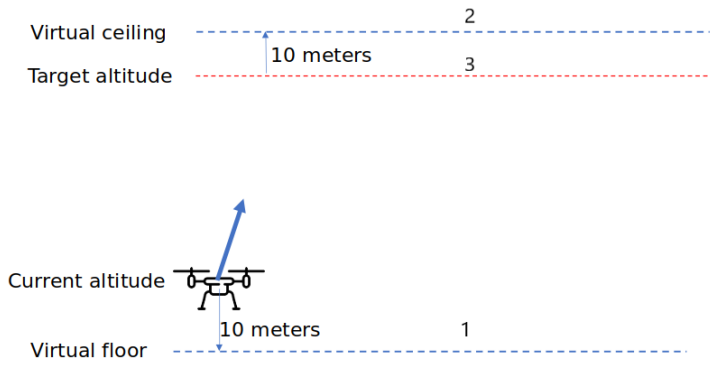


Figura 4.4: Principio de límites de vuelo virtuales.

más allá del techo y suelo virtual cuando se realiza una maniobra de evasión. Este techo y suelo serán valores que se modificarán, a lo largo del protocolo, respecto a la altura del dron y del objetivo que debe alcanzar. Concretamente, estos límites se definen respecto a si la altura del dron está por encima, por debajo, o cerca de la altura objetivo:

1. Si la altitud del dron está por debajo de los 3 metros respecto a la altura objetivo, el suelo virtual será el máximo entre el valor anterior del suelo y la altura del dron reducida en 10 metros. El valor del techo se establecerá como 10 metros sobre la altura objetivo.

2. Si la altitud del dron está por encima de los 3 metros respecto a la altura objetivo, el suelo obtendrá un valor de menos 10 metros respecto a la altura objetivo. El techo virtual será el mínimo entre el anterior valor del techo y la altura del dron aumentada en 10 metros.

3. Si no se cumple ninguno de los anteriores casos, el dron está a una altura óptima para alcanzar el objetivo. El suelo tendrá un valor de 10 metros inferior a la altura objetivo, y el techo 10 metros superior a esta misma altura.

Una vez se establezcan estos límites, se utilizarán como dato a tener en cuenta en la reducción del vector de repulsión, eludiendo al dron al reducir o aumentar su altura por encima o por debajo de estos límites. Con este principio podemos prohibir que el dron, al realizar una maniobra de evasión, reduzca su altura a unos niveles cercanos del suelo físico, o un aumento innecesario de esta altura, evitando problemas de colisiones contra dicho suelo y/o violaciones por sobrepasar la altura máxima que estipula la ley. Para

evitar estos problemas, se ha puesto el valor mínimo del suelo como 10 y el valor máximo de 120 para el techo.

Una vez calculado los límites virtuales, se procede ahora a calcular el vector de atracción hacia el siguiente objetivo al que volar. El cálculo de este vector es representado en el código de la figura 4.5 con la función *getAttractionVector()*.

```
1  private Vector getAttractionVector(){
2      Location3DUTM UAVLoc = getCopterLocation();
3      Location3DUTM WP = waypoints.peek();
4
5      Vector attr = new Vector(UAVLoc, WP);
6
7      attr.normalize();
8      attr.scalarProduct(maxSpeed);
9
10     double beta = attractionFunction(WP, UAVLoc);
11     attr.scalarProduct(beta);
12
13     return attr;
14 }
```

```
1  private double attractionFunction(Location3DUTM WP,
       Location3DUTM UAVLoc){
2      double beta;
3      double distance = new Vector(UAVLoc, WP).magnitude();
4
5      if (distance >= tDist){
6          beta = 1;
7      } else {
8          beta = 0.5;
9      }
10
11     return beta;
12 }
```

Figura 4.5: Funciones *getAttractionVector()* y *attractionFunction(...)* en DFFP.java para el cálculo del vector de atracción.

Para obtener este vector, se calculan primero la posición tanto del dron en marcha como del objetivo "waypoint", y se restan las coordenadas de dichas posiciones para que el vector apunte hacia el objetivo. Esta operación de resta es facilitada por la clase *Vector* anteriormente comentada, al igual que su posterior reducción al valor unitario y su multiplicación por la velocidad

correspondiente del dron en vuelo. Por último, este vector es alterado por la variable beta, valor resultado de la función *attractionFunction(...)* de la figura 4.5. Esta función sirve para reducir la velocidad de dicho vector si el dron está cerca del objetivo. Con esto se evita que el dron pase el objetivo sin haber alcanzado su posición debido a la velocidad y la inercia que provoca este vector de atracción. Dicha distancia se calcula con la magnitud del vector anteriormente calculado y, si es mayor que la variable "tDist" (parámetro definido por el usuario), se reduce dicho vector a la mitad. Al modificar los componentes del vector por la variable beta, obtenemos el vector de atracción necesario para mover el dron hacia el objetivo.

Pasando ahora al cálculo del vector de repulsión, en la figura 4.6 se muestra el código implementado. Esta función es la más importante del algoritmo, ya que aquí reside la decisión del dron de hacia donde debe dirigirse para esquivar una posible colisión. Primero, se explicará todo el funcionamiento del código de dicha función, dejando para más tarde el cálculo de la variable gamma, que es la indicadora del ángulo que debe tomar el vector de repulsión.

```
1  private Vector getRepulsionVector(){
2      Vector totalRepulsion = new Vector(0,0,0);
3
4      for(obstacle:obstacleList){
5          Location3DUTM obstLoc = new Location3DUTM(obstacle.
               getLocationUTM(), obstacle.getAltitude());
6          double obstHeading = obstacle.getHeading();
7
8          Location3DUTM UAVLoc = getCopterLocation();
9          double UAVHeading = copter.getHeading();
10
11         Vector repulsion = new Vector(obstLoc, UAVLoc);
12         repulsion.normalize();
13
14         double forceFactor = repulsionFunction(
15                 UAVLoc, UAVHeading,
16                 obstLoc, obstHeading
17         );
18
19         repulsion.scalarProduct(forceFactor);
20
21         double cA = copter.getAltitude();
22         if(cA != targetAltitude){
23             double reductionFactor = Math.min(Math.abs(cA -
                   virtualCeiling), Math.abs(cA - virtualFloor))
24                       / Math.abs(cA - targetAltitude);
25             if(reductionFactor < 1) {repulsion.z *=
                   reductionFactor;}
26         }
27
28         totalRepulsion.add(repulsion);
29     }
30
31     totalRepulsion.scalarProduct(2*maxSpeed);
32
33     return totalRepulsion;
34 }
```

Figura 4.6: Función *getRepulsionVector()* en DFFP.java para el cálculo del vector de repulsión.

Empezando con un vector total vacío, está función tratara de sumar todos los vectores de repulsión de cada obstáculo respecto al dron, siempre que el obstáculo en cuestión no haya aterrizado, ya que se dejaría de constar como tal. Primero, se calcula el vector repulsión restando las posiciones del dron evasor y del obstáculo de forma que el vector tenga una dirección opuesta al obstáculo, se normaliza este vector para reducirlo al valor unitario, y se le

aplica un producto interior proporcionado por la variable "forceFactor", que indica la fuerza del vector de repulsión resultante de aplicar la radiación del campo de fuerza respecto a la posición y distancia del obstáculo. Después, la coordenada z de este vector será afectada por un factor de reducción establecido por la diferencia de altura entre el dron y el objetivo, además de su distancia a los límites virtuales establecidos previamente. Una representación visual de la fórmula aplicada al eje z es la siguiente:

$$reductionFactor = \frac{\text{mín}\left(|currentAltitude - virtualCeiling|, |currentAltitude - virtualFloor|\right)}{|currentAltitude - targetAltitude|}$$

El factor de reducción presenta un comportamiento que tiende a cero cuanto mayor sea la diferencia entre la altura del dron y la altura objetivo, y tiene como numerador el valor mínimo entre las diferencias absolutas del dron con los límites virtuales. Esto presenta un comportamiento balanceado, aumentando el valor del factor cuando más cerca está de la altura objetivo, y reduciéndolo cuanto más se aleja. Como medidas adicionales, no realizamos dicha operación cuando la altura del dron y del objetivo es la misma, ya que dicha resulta en una división por cero. Además, tampoco queremos multiplicar el vector cuando el valor del factor de reducción es mayor que 1, ya que no nos interesa aumentar la repulsión, solo disminuirla cuando se acerque a un límite virtual.

Por último, el vector de repulsión calculado es acumulado al vector total de todos los obstáculos y, una vez se acabe dicho cálculo, se escala dicho vector a la velocidad actual del dron, y se multiplica por dos para priorizar la fuerza de la repulsión sobre la atracción, evitando un caso donde ambos sean iguales y se contrarresten, cosa que fuerza al dron a adoptar una posición estática.

Como bien hemos dicho antes, el vector de repulsión disminuye dependiendo de la posición del obstáculo respecto al campo de fuerza aplicado. La función "repulsionFunction" se encarga de calcular esta reducción observando la posición entre el dron que ejecuta el protocolo y el obstáculo a evitar. En la siguiente figura 4.7 mostramos dicho código:

```
1  private double repulsionFunction(Location3DUTM UAVLoc1, double
       Heading1, Location3DUTM UAVLoc2, double Heading2){
2      double repulsion, gamma;
3      double theta1, theta2, F1, F2;
4      double distance = new Vector(UAVLoc2, UAVLoc1).magnitude();
5
6      Vector vH1 = new Vector(
7              Math.sin(Heading1),
8              Math.cos(Heading1),
9              0);
10     Vector vD1 = new Vector(UAVLoc1, UAVLoc2);
11     //vD1.z = 0;     // Plano 2D
12
13     // Plano 3D
14     theta1 = Math.atan2((Vector.crossProduct(vH1,vD1)).magnitude
           (),Vector.dotProduct(vH1,vD1));
15     // Plano 2D
16     //theta1 = Math.acos(Vector.dotProduct(vH1, vD1)/(vH1.
           magnitude()*vD1.magnitude()));
17
18     F1 = fieldDirectionForce(theta1);
19
20     Vector vH2 = new Vector(
21             Math.sin(Heading2),
22             Math.cos(Heading2),
23             0);
24     Vector vD2 = new Vector(UAVLoc2, UAVLoc1);
25     //vD2.z = 0;          // Plano 2D
26
27     //Plano 3D
28     theta2 = Math.atan2((Vector.crossProduct(vH2,vD2)).magnitude
           (),Vector.dotProduct(vH2,vD2));
29     //Plano 2D
30     //theta2 = Math.acos(Vector.dotProduct(vH2, vD2)/(vH2.
           magnitude()*vD2.magnitude()));
31
32     F2 = fieldDirectionForce(theta2);
33
34     gamma = gammaFunction(distance);
35
36     repulsion = gamma * Math.max(F1,F2);
37
38     return repulsion;
39 }
```

Figura 4.7: Función *repulsionFunction(...)* en DFFP.java para el cálculo de la fuerza del vector de repulsión.

Para calcular dicha reducción, lo primero que debemos saber es el ángulo del obstáculo respecto al dron, y si este está comprendido dentro del campo de fuerza o no. Para ello es necesario saber la posición y dirección de ambos, además de la distancia que presentan entre sí. Dichas posiciones son usadas para crear un vector distancia entre el dron y el obstáculo, y otro entre el obstáculo y el dron, vectores iguales pero con trayectorias inversas. Las direcciones, con un ángulo tomado en radianes, se utilizan para calcular dos vectores dirección que representan hacia donde miran tanto el dron como el obstáculo. Ambos vectores sirven para conocer el ángulo exacto en radianes hacia donde apunta el vector distancia:

$$\theta_{2D} = \arccos\left(\frac{\vec{H}\cdot\vec{D}}{|\vec{H}|*|\vec{D}|}\right) \qquad \theta_{3D} = \arctan\left(\frac{|\vec{H}\times\vec{D}|}{\vec{H}\cdot\vec{D}}\right)$$

Representamos el ángulo θ calculado a partir del vector dirección $\overrightarrow{H}$ y del vector distancia $\overrightarrow{D}$ de dos formas, θ_{2D} y θ_{3D}. Esto se debe a la funcionalidad de evasión en tres dimensiones, que fue implementada una vez se desarrolló el protocolo cuando solo tenia en cuenta dos dimensiones. En el código, se ha comentado la ecuación del cálculo en dos dimensiones para una reversión de cambios posible, calculando el ángulo en las tres dimensiones por defecto. Esto lo conseguimos realizando la arcotangente del cociente entre la magnitud del producto vectorial y el producto escalar de ambos vectores dirección y distancia. El producto vectorial indica el vector perpendicular situado en el eje z de los vectores dirección y distancia, mientras que el producto escalar indica el ángulo comprendido entre los dos vectores.

```
1  private double fieldDirectionForce(double theta){
2      double dForce, D;
3
4      if(Math.abs(theta) < Math.PI/(dirFactor*2)){
5          D = Math.cos(dirFactor*theta);
6      } else {
7          D = 0;
8      }
9
10     dForce = Math.max(D, dirRatio);
11     return dForce;
12 }
```

Figura 4.8: Función *fieldDirectionForce(...)* en DFFP.java.

Con el ángulo dirección calculado, se utilizará para determinar la posición del vector respecto a la radiación del campo de fuerza, donde la figura

4.8 representa el cálculo. Si dicho ángulo está comprendido en el intervalo $\left[-\frac{\pi}{\text{dirFactor}*2}, \frac{\pi}{\text{dirFactor}*2}\right]$, entonces este se encuentra apuntando a la radiación direccional y, si este no es el caso, apuntará hacia el campo omnidireccional. El valor devuelto por esta función será el máximo entre el parámetro "dirRatio" y la variable "D", que será el coseno del ángulo por el parámetro "dirFactor" si apunta a la parte direccional, o 0 en el caso contrario. Ambos parámetros permiten al usuario ajustar varios aspectos del campo de fuerza, donde "dirFactor" delimita la anchura que abarca la elipse direccional y "dirRatio" establece la reducción base aplicada al vector de repulsión. Con el valor máximo calculado, el vector de repulsión será reducido con el producto interior del valor resultante de aplicar esta función.

La función "gammaFunction" también se encarga de reducir la fuerza del vector de repulsión salvo que, en vez de ser por la posición del obstáculo respecto al dron, se tiene en cuenta la distancia del obstáculo respecto al campo de fuerza.

```
1  private double gammaFunction(double distance){
2      double gamma;
3
4      if (distance <= frDist){
5          gamma = 1;
6      } else {
7          gamma = Math.max(1 - Math.pow((distance-frDist)/alpha,
               2), 0);
8      }
9
10     return gamma;
11 }
```

Figura 4.9: Función *gammaFunction(...)* en DFFP.java

En la figura 4.9 se muestra el código de esta función, presentando dos parámetros adicionales como en la anterior función. Mientras "frDist" representa el radio del campo omnidireccional, el parámetro "alpha" representa la longitud de la elipse direccional. Para el cálculo de esta reducción, el vector de repulsión no se verá afectado si la distancia del dron con el obstáculo es igual o menor al parámetro "frDist". Si este no es el caso, la reducción tendrá un valor máximo entre 0 y $1 - \left(\frac{\text{distance}-\text{frDist}}{\text{alpha}}\right)^2$. Con esto obtenemos que el vector de repulsión sea reducido a mayores distancias, siendo el vector de repulsión reducido a 0 si el obstáculo no está dentro del circulo omnidireccional ni de la elipse direccional, no moviendo al dron de su rumbo planeado.

Como último punto en la reducción del vector de repulsión de la figura 4.7, el valor que devolverá esta función para posteriormente reducir el vector de repulsión será el resultado de multiplicar y valor de la "gammaFunction" y el máximo entre la reducción aplicada respecto al ángulo calculado del dron hacia el obstáculo y del obstáculo hacia el dron. Aunque ambos valores deben ser iguales, ya que tienen el mismo ángulo, se coge el máximo valor devuelto por si pudiera ocurrir un ligero error en los datos de posición y/o error que se consultan del dron y del obstáculo.

Volviendo al código del protocolo D-FFP 4.2, los últimos pasos a realizar son obtener el vector resultado de añadir ambos vectores de atracción y repulsión, y comandar al dron para que se desplace en la dirección del vector resultante. Dicho desplazamiento lo logramos en la función "moveUAV" representada en la figura 4.10.

```
1  private void moveUAV(Vector resulting) {
2      if(resulting.magnitude() >= maxSpeed){
3          resulting.normalize();
4          resulting.scalarProduct(maxSpeed);
5      }
6
7      double resultYaw = Math.atan2(resulting.x, resulting.y);
8
9      copter.moveTo(resulting.y, resulting.x, -resulting.z,
           resultYaw, 0);
10 }
```

Figura 4.10: Función *moveUAV(...)* en DFFP.java.

Antes de mandar la orden de movimiento al dron, reducimos el vector resultado si su magnitud sobrepasa la velocidad máxima del dron. Esto se logra normalizando el vector a su valor unitario y realizando su multiplicación escalar sobre la velocidad del dron en el vuelo. Después, calculamos el ángulo del vector resultado con la arcotangente de sus parámetros coordenadas "x" e "y", y enviamos una orden para que el dron se mueva con las mismas velocidad y ángulo que las coordenadas del vector resultante. Concretamente, "moveTo" toma como parámetros en su función la velocidad objetivo apuntando al norte, la velocidad objetivo apuntando al este, la velocidad objetivo apuntando al suelo, el ángulo de rumbo del dron, y la velocidad de giro para alcanzar el ángulo. Como los vectores de velocidad a indicar en el comando de movimiento apuntan hacia el norte, este, y al suelo, se invierten las dos

primeras coordenadas del vector resultante y se cambia el signo del eje z. Se aplica el ángulo anterior calculado del vector resultado para hacer girar el dron hacia ese sentido, y se le aplica un valor por defecto 0 a la velocidad de giro (el valor a 0 indica no modificar la velocidad ya predefinida para girar el dron).

Todo este proceso se verá repetido cada 200 milisegundos para indicar constantemente la trayectoria que debe seguir el dron y los obstáculos a evadir sin agotar los recursos del ordenador ni del simulador ejecutado. Como bien se ha comentado al principio de este apartado, este proceso se repetirá constantemente hasta que se agoten los puntos objetivos que debe volar el dron, marcando el final de su misión, aterrizando en el suelo y finalizando todo el proceso del protocolo D-FFP.

4.3.3. Ajuste de parámetros

En la sección anterior, se ha comentado constantemente el uso de variables parametrizadas que el usuario puede modificar para alterar las propiedades del campo de fuerza. Concretamente, han sido los parámetros "tDist", "dirFactor", "dirRatio", "frDist", "dirFactor", "dirRatio" y "alpha" los que se han nombrado:

- "tDist": Delimita un radio circular comprendido dentro de los objetivos de la misión. Si el dron alcanza el valor del radio, el vector de atracción se reduce a la mitad. Esto evita que el dron no alcance el objetivo debido a la inercia provocada por la velocidad.
- "dirFactor": Establece la longitud del semieje menor de la elipse direccional.
- "dirRatio": Proporción constante aplicada a la reducción posicional del vector de repulsión.
- "frDist": Radio del circulo omnidireccional. Si el dron se encuentra dentro, el vector de repulsión alcanzar su máximo valor.
- "alpha": Establece la longitud del semieje mayor de la elipse direccional.

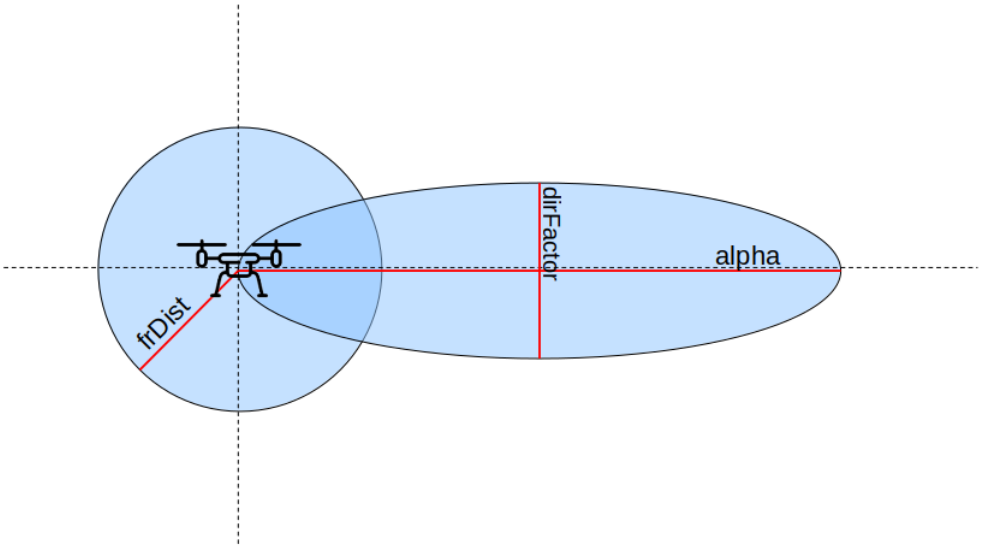


Figura 4.11: Radiación del campo D-FFP y los parámetros que la modifican.

Todos estos parámetros permiten definir el comportamiento que tendrá el protocolo en una misión, estableciendo como de grande o pequeño será el campo total creado por el protocolo. En nuestro caso, lo que nos interesa es que los drones obtengan el menor desplazamiento posible a la hora de realizar maniobras evasivas, pero también debemos respetar la distancia mínima de 10 metros establecida para el dron, que asegura que no existe una posible colisión entre drones. Para ello se debe realizar una tanda de experimentos y probar a aumentar y disminuir valores en base a pruebas de vuelo. Realizar estos ajustes, e ir probando los parámetros uno a uno, puede ser una tarea ardua para el usuario que lo realice a manualmente. Para programar tandas de misiones, el simulador ArduSim ofrece su ejecución mediante terminal, haciendo falta su versión compilada, y el uso de un script para realizar las misiones, cambiando parámetros a medida que acaben.

El lenguaje Python ha sido de ayuda para esto, permitiéndonos crear un script (ver figura 4.12) que altere los parámetros en intervalos de crecimiento, mientras ejecuta una tanda de misiones por cada ajuste que realiza:

Parámetro	**Rango**	**Intervalo**
frDist	[50, 70]	10
alpha	[10, 30]	10
dirFactor	[2, 6]	2
dirRatio	[0.2, 0.4]	0.1

Cuadro 4.1: Rangos y intervalos de muestreo para ajustar los parámetros del campo direccional

La elección de los rangos e intervalos viene dada al previo ajuste de parámetros en la versión anterior del protocolo D-FFP realizada en MATLAB. El ajuste en MATLAB realiza un mayor rango con menores intervalos debido a la rapidez de experimentación de la herramienta, sin embargo, los valores óptimos de esta versión no concordaban al utilizarlos en el simulador, dando resultados peores. Mediante prueba y error y de forma manual, se consiguió establecer un patrón valido de los parámetros que ofrecían resultados favorables, gracias a las pruebas anteriores de la versión MATLAB.

El script 4.12 realizó un total de 324 misiones ejecutadas a lo largo de 12,3 horas. Cuando finalizaba una misión, distintos datos como el tiempo total y la distancia mínima entre drones, se almacenaban en un archivo csv. Comparando y filtrando resultados, se pudo obtener una conclusión respeto los parámetros ideales, que son los siguientes:

- tDist = 50
- frDist = 70
- alpha = 30
- dirFactor = 6
- dirRatio = 0.2

Como información adicional, el parámetro "tDist" no ha sido incluido en estas pruebas debido a que su alteración no afecta al comportamiento del campo de fuerza, y se ha puesto por defecto a 50 en todos los casos. Además, estos parámetros dan lugar a una futura mejora de precisión, que en este trabajo no se ha podido llevar a cabo debido a la gran cantidad de tiempo necesaria para probar rangos mayores y con mayor granularidad de valores.

Los valores de los distintos parámetros, tal y como indicados arriba, serán aplicados para todas las pruebas que utilicen este protocolo en el siguiente capítulo de experimentación.

```
import subprocess
import glob
import os.path
import time

missions= ["cross", "angleSameDir", "angleDifferentDir", "headon"]
protocolparametersFilePath = "magnetics.properties"
ardusimParametersFilePath = "SimulationParam.properties"
logFilePath = "logFileDFFP.txt"
sequential = "true"

def writeProtocolParameters(mission, frDist, alpha, dirFactor, dirRatio):
    with open(protocolparametersFilePath, "w") as f:
        f.write("tDist=50\n")
        f.write("frDist=" + str(frDist) + "\n")
        f.write("alpha=" + str(alpha) + "\n")
        f.write("dirFactor=" + str(dirFactor) + "\n")
        f.write("dirRatio=" + str(dirRatio) + "\n")
        f.write("missionFile=" + mission + ".kml\n")
        f.write("beaconingTime=200\n")

start = time.time()
for frDist in range(50, 71, 10):
    for alpha in range(10, 31, 10):
        for dirFactor in [2,4,6]:
            for dirRatio in range(2, 5, 1):
                for mission in missions:
                    writeProtocolParameters(mission, frDist, alpha, dirFactor, dirRatio/10)
                    cmd = ['java', '-jar', 'ArduSim.jar', 'simulator-cli', ardusimParametersFilePath]
                    print(cmd)
                    print("running mission "+ mission + "\n")
                    subprocess.run(cmd)
                    end = time.time()
                    print("Elapsed time: " + str(end-start) + " seconds")
                    with open(logFilePath, 'a') as file:
                        file.write(mission + ";executed correctly\n")
print("simulation Done\n")
end = time.time()
print("Simulation lasted " + str(end-start) + " seconds")
```

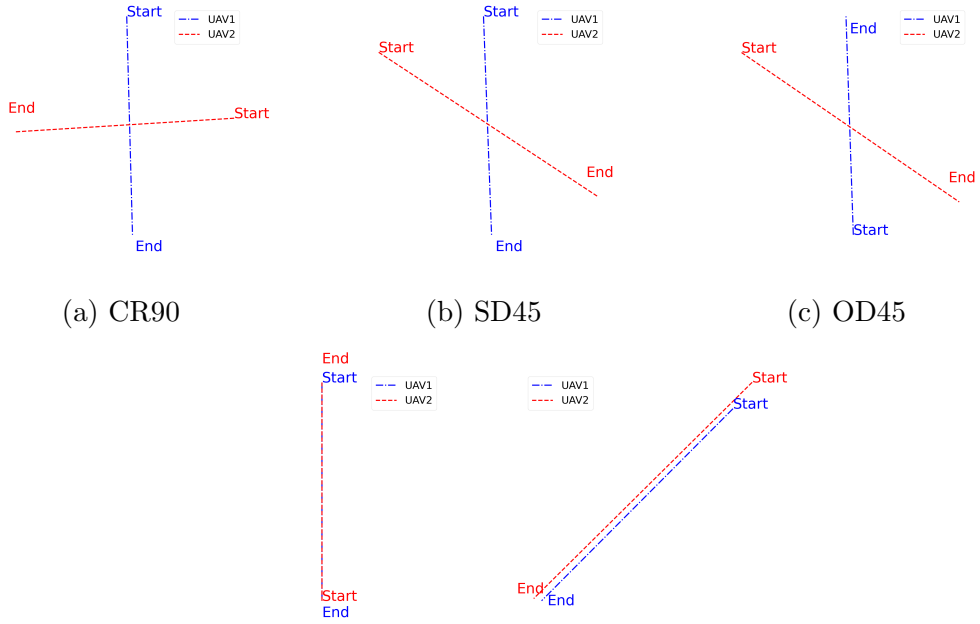
Figura 4.12: Script DFFP.py para la realización de tandas de misiones con modificación de parámetros en intervalos.

Capítulo 5

Experimentos y pruebas finales

Este penúltimo capítulo será dedicado a la realización de varias pruebas y experimentos con la finalidad de validar el funcionamiento del protocolo *Directional-Force Field Protocol* (D-FFP), presentando y recopilando información de distintas misiones creadas para abarcar la mayoría de casos problemáticos. Dando una vista general de lo que se espera hablar en este capítulo, primero se mostrarán los escenarios planteados en los casos de prueba, presentando distintos ángulos de colisión que afrontará el campo de fuerza implementado. Después se realizarán pruebas de vuelo de dron contra dron para observar las evasiones resultantes, donde ambos drones implementaran el nuevo protocolo. Seguidamente, se compararán los resultados obtenidos en estas pruebas junto a los datos y las trayectorias generadas por la versión del protocolo anterior *Force Field Protocol* (FFP). Todas estas pruebas definidas se realizarán teniendo en cuenta solo dos dimensiones, dando paso a la siguiente tanda de experimentos donde se probará la implementación de la funcionalidad en tres dimensiones. Estos experimentos constarán de una prueba de vuelo de un dron que recorrerá una trayectoria lineal, mientras varios drones posicionados en dicha trayectoria se desplazarán verticalmente, obstaculizando al dron principal en el eje z mientras realiza su recorrido. Por último, se realizará una última prueba aérea de 10, 25 y 50 drones con recorridos aleatorios a lo largo de una misión, ejecutando el protocolo final desarrollado.

Ahora sí, procedemos a representar gráficamente las distintas misiones de vuelo creadas con el propósito de abarcar al máximo las posibles situaciones que puedan generar una colisión. La figura 5.1 presenta estos casos:



(a) CR90 (b) SD45 (c) OD45

(d) HO (e) TO

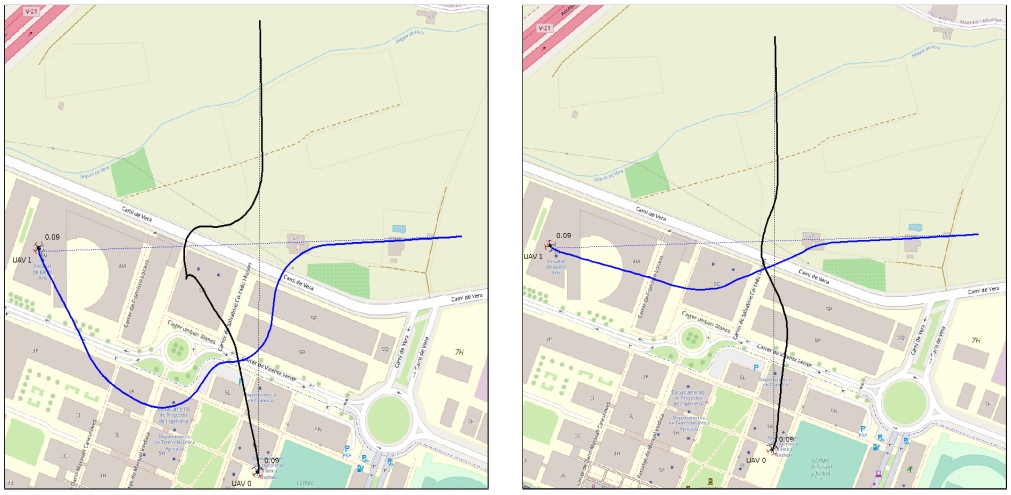
Figura 5.1: Representación de los cinco escenarios usados en las pruebas.

- CR90: Ambos drones presentan unas trayectorias cruzadas, alcanzando una colisión de 90 grados en la intersección.
- SD45: Se genera una colisión de 45 grados donde ambos drones presentan trayectorias cruzadas con similar dirección.
- OD45: Se genera una colisión de 45 grados donde ambos drones presentan trayectorias cruzadas con direcciones opuestas.
- HO: Las trayectorias de los dos drones son las mismas, pero con direcciones inversas, presentando una colisión frontal.
- TO: Ambos drones realizan la misma trayectoria, pero el primero empieza desde más atrás, persiguiendo al segundo dron y alcanzándolo mientras este aterriza.

Estos cinco escenarios se utilizarán para realizar las rutas aéreas de los drones en el simulador ArduSim. Lo que se desea conseguir de estos escenarios son datos que nos permitan analizar un equilibrio óptimo entre la sobrecarga de tiempo del vuelo y la distancia de seguridad entre ambos drones. La sobrecarga de tiempo hace referencia a la diferencia del tiempo de ejecución de

la misión usando el protocolo respecto a no usarlo, es decir, cuanto tiempo se tarda de más en terminar la misión. Respecto a las distancias mínimas, el simulador ArduSim establece una distancia de seguridad de 10 metros para garantizar la no colisión entre drones; por lo tanto, se intentará reducir la distancia entre drones lo máximo posible a este valor para minimizar cambios en la trayectoria.

Ahora se procederá a mostrar los resultados comparativos de las trayectorias tras ejecutar los protocolos FFP y D-FFP, representando en el simulador al primer dron con una trayectoria realizada de color negro, y el segundo con una trayectoria de color azul:



(a) FFP (b) D-FFP

Figura 5.2: Primer escenario comparativo: CR90.

En la figura 5.2, representamos el primer escenario (CR90) con dos imágenes de su simulación. La imagen de la izquierda muestra el patrón de vuelo logrado por el protocolo FFP, mientras que el de la derecha muestra el patrón de vuelo de D-FFP. Como primera observación, las diferencias en las trayectorias de ambas ejecuciones son bastantes diferentes, presentando evasiones exageradas en el primer caso respecto al segundo. Esto se debe a que, al entrar en contacto ambos drones con sus respectivos campos de fuerza, el vector de repulsión generado aleja a los drones en direcciones opuestas. En la ejecución D-FFP también hay un alejamiento por el campo de fuerza omnidireccional, pero de menor fuerza y, además, cuando entra en contacto el campo de fuerza direccional, el vector de repulsión del segundo dron es opuesto a su trayectoria, generando una fuerza contraria a la de atracción, reduciendo la

velocidad del dron y dejando pasar al primer dron.

(a) FFP (b) D-FFP

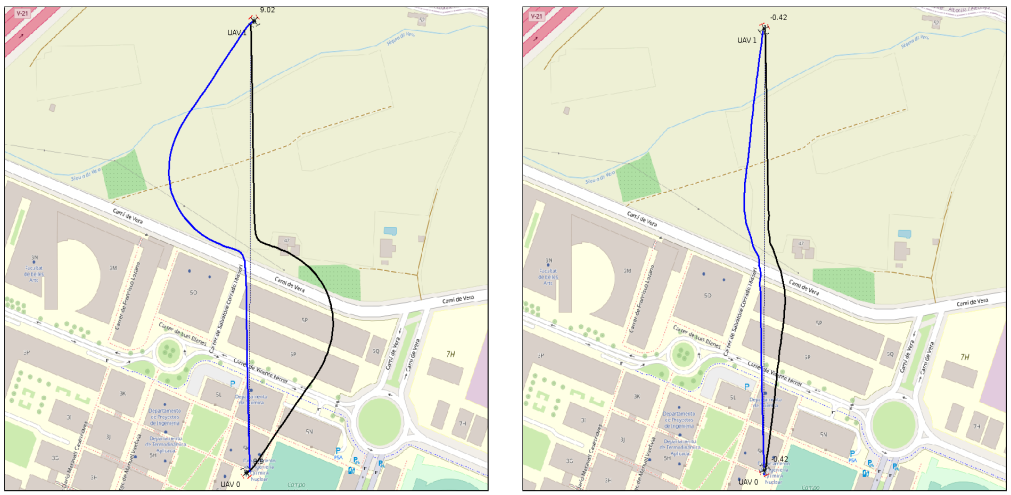
Figura 5.3: Segundo escenario comparativo: SD45.

Pasando ahora a las ejecuciones del escenario SD45 5.3, el principal problema a solucionar es la constante repulsión entre ambos drones en el intento de cruzarse entre sí. En la ejecución con FFP vemos que la fuerza de repulsión aleja constantemente a ambos drones creando ondulaciones en la trayectoria, hasta que finalmente el segundo dron realiza una evasión con una curva interior para cruzar al primer dron por detrás suya. Respecto a la ejecución D-FFP, ambos drones reciben una fuerza de repulsión constante que los mantiene alejados, pero como la fuerza de atracción es más fuerte que la repulsión omnidireccional, estos se van acercando lentamente a lo largo de la trayectoria. Esto ocurre hasta que el primer dron detecta al segundo en su campo direccional, recibiendo una fuerza de repulsión que reduce su velocidad, dejando pasar al segundo dron (con trayectoria azul).

(a) FFP (b) D-FFP

Figura 5.4: Tercer escenario comparativo: OD45.

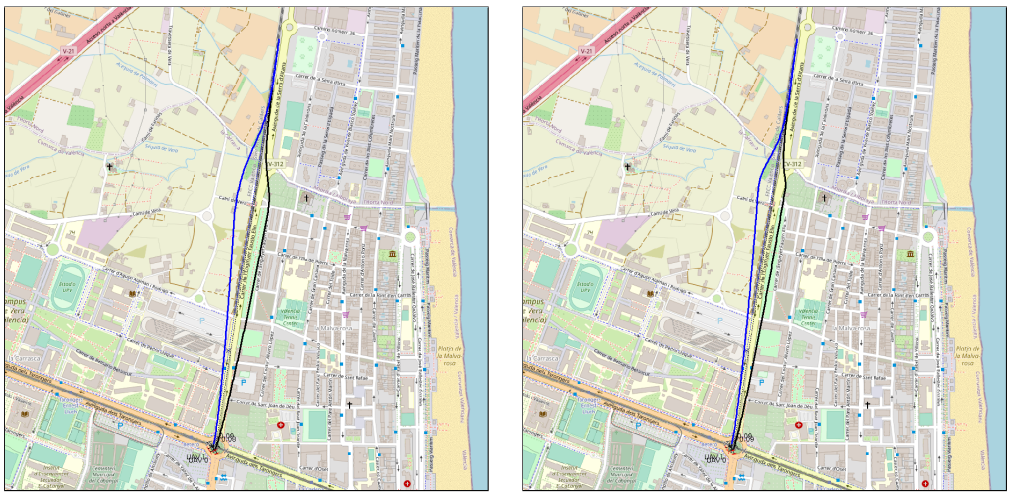
Como tercer escenario (OD45), la figura 5.4 presenta un conflicto donde el primer dron se encuentra de forma frontal al segundo, mientras este último lo tiene presente en un lateral. El protocolo FFP gestiona este conflicto aplicando un vector de repulsión fuerte cuando ambos drones entran en contacto con sus respectivos campos de fuerza, resultando en un giro de 360 grados para el primer dron y una curvatura exagerada de evasión para el segundo. En el protocolo D-FFP, el campo direccional del primer dron entra en contacto con el segundo en el punto de intersección, reduciendo su velocidad debido a un vector de repulsión contrario al de atracción. Adicionalmente, ambos drones se desplazan ligeramente al final de la evasión debido a la componente omnidireccional.



(a) FFP (b) D-FFP

Figura 5.5: Cuarto escenario comparativo: HO.

El escenario HO (ver figura 5.5)representa una mejor visualización del uso de un protocolo respecto al otro. En la primera implementación, ambos drones siguen su rumbo hasta hacer contacto con sus respectivos campos de fuerza, alejándose lateralmente. En la implementación D-FFP, ambos drones se reconocen en sus respectivos campos direccionales y se alejan ligeramente reduciendo sus velocidades hasta alcanzar el punto omnidireccional y realizar una curva un poco más pronunciada. La reducción de velocidad del vector de repulsión en este escenario no es conveniente, ya que se aumenta el tiempo de misión de forma innecesaria; sin embargo, al no pronunciar tanto la curvatura de evasión antes de la colisión, se balancea esta pérdida, lográndose una reducción del tiempo superior a la pérdida.



(a) FFP (b) D-FFP

Figura 5.6: Quinto escenario comparativo: TO.

Como último caso, el escenario TO (ver figura 5.6) representa una situación donde la ejecución de ambas implementaciones dan resultados similares. Tanto la ejecución del protocolo FFP como la D-FFP presentan una trayectoria casi duplicada, con excepción de una pequeña desviación difícil de apreciar al final de la trayectoria azul del segundo dron. El resultado del tiempo de la segunda implementación recibe un aumento de casi una décima de segundo respecto a la primera. Estos resultados los observaremos en la siguiente tabla, donde describiremos cada escenario con su tiempo de ejecución sin implementar ningún protocolo, junto a la sobrecarga de tiempo "ST" y la distancia mínima alcanzada de ambas implementaciones. Volviendo a recalcar, la sobrecarga de tiempo hará referencia a la diferencia entre el tiempo total alcanzado de la misión con una implementación menos el tiempo mínimo alcanzado sin ejecutar ninguna implementación, tal y como se detalla en la tabla 5.1.

Esta tabla resume la sobrecarga temporal y la distancia mínima alcanzada para todas las ejecuciones de las dos implementaciones en los cinco escenarios. Podemos observar como se reduce considerablemente la sobrecarga de tiempo en los cuatro primeros escenarios. El promedio de los cinco escenarios ejecutados en el protocolo D-FFP se ve reducido en 17 segundos aproximadamente, presentando un 69,1 % de porcentaje de disminución respecto al protocolo FFP. Además, el promedio de la distancia mínima se reduce a casi la mitad en comparación con la implementación FFP, nunca alcanzando menos de 10 metros para respetar la distancia de seguridad.

Cuadro 5.1: Resultados de la comparación de los 5 escenarios ejecutados con los protocolos FFP y D-FFP, en términos de distancia mínima entre drones y sobrecarga de tiempo (ST) respecto al tiempo mínimo.

Escenario	Tiempo mín. [s]	**FFP**		**D-FFP**	
		ST (s)	**Dist. Mín. [m]**	**ST [s]**	**Dist. Mín [m]**
CR90	46,45	27,07	62,62	5,8	20,79
SD45	45,05	34,07	92,52	10,21	22,13
OD45	44,85	24,87	26,57	4,21	12,08
HO	45,09	16,02	20,68	3,57	13,87
TO	155,77	23,27	11,02	14,91	64,31
Promedio	***67,44***	***25,06***	***42,68***	***7,74***	***26,64***

Con estos resultados podemos demostrar la eficacia de nuestro protocolo desarrollado en comparación con la versión anterior principalmente centrada en la omnidireccionalidad. Al reducir la sobrecarga de tiempo y la distancia mínima, logramos acabar una misión de forma más temprana, y se disminuye las distancias de separación provocadas por la maniobra de evasión, lográndose así menos consumo eléctrico, y permitiendo al dron estar más tiempo en el aire.

Ahora pasaremos a mostrar un escenario creado específicamente para probar el comportamiento en tres dimensiones implementado como característica adicional de gestión de conflictos del protocolo *Directional-Force Field Protocol* (D-FFP).

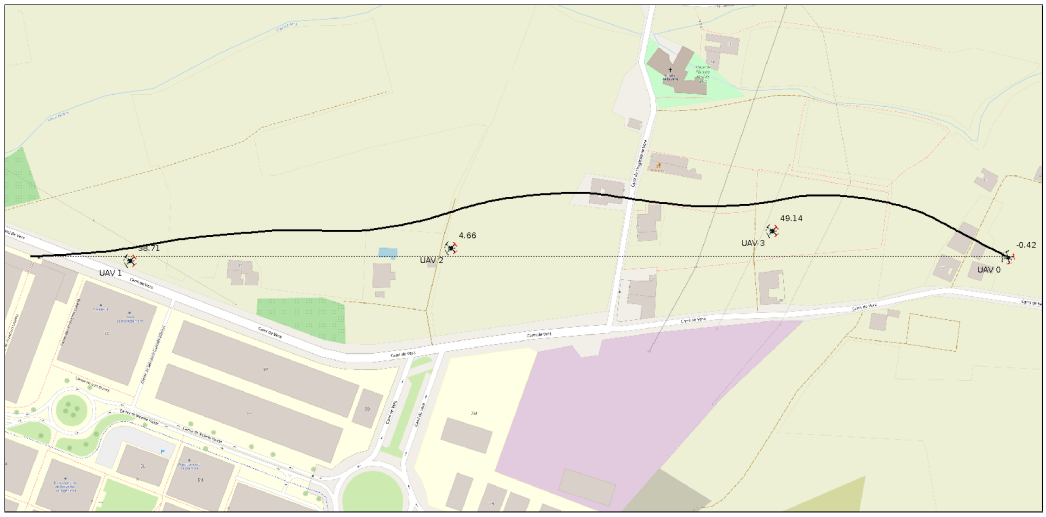


Figura 5.7: Escenario planteado para la prueba 3D de D-FFP.

Para esta prueba se lanzarán 4 drones, donde solo el primero ejecuta el protocolo D-FFP. El dron principal tendrá a recorrer una trayectoria en linea recta con una altitud de 30 metros. El resto de drones estarán ubicados a lo largo de la trayectoria del primero, y se dedicarán a incrementar o decrecer la altitud cuando el dron principal se acerque a sus proximidades. Concretamente, los drones ubicados al principio y final (sin contar el dron que realiza la trayectoria) aumentarán en 40 metros su altitud cuando el dron principal se aproxime, y el dron ubicado por la mitad del recorrido disminuirá su altitud en 40 metros. La trayectoria resultante de ejecutar este escenario esta representado en la figura 5.7. Sin embargo, lo que realmente importa de este escenario es observar la altura de vuelo del dron a lo largo del recorrido, y desde la imagen del simulador no es posible observarlo. Para ello, se ha guardado la información de la altura del dron principal cada 200 milisegundos, desde que acaba de despegar hasta que inicia la maniobra de aterrizaje.

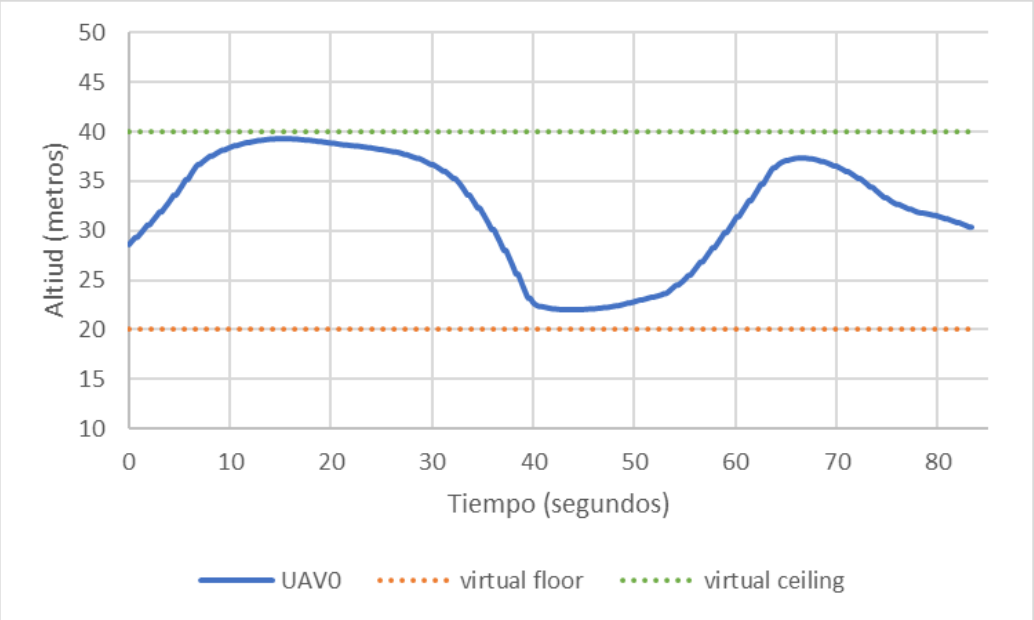


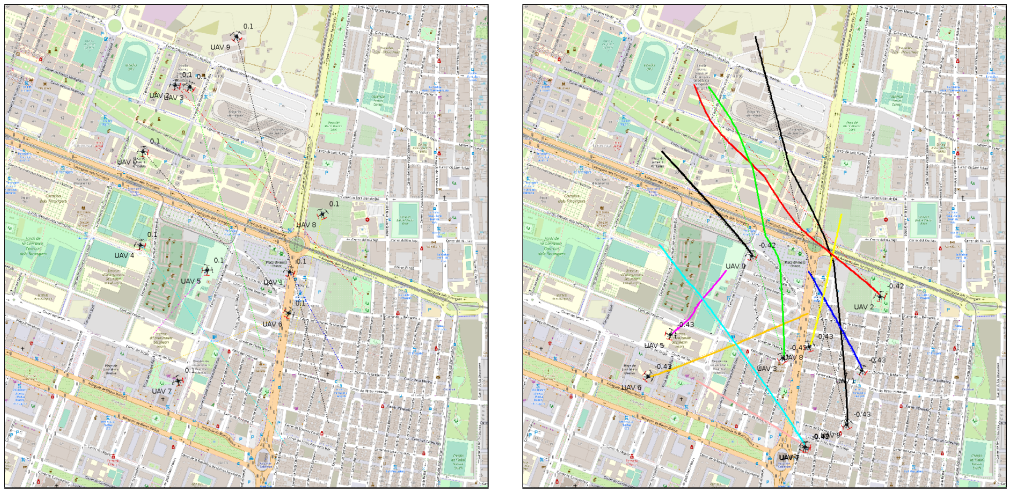
Figura 5.8: Gráfica de altura del dron principal para la prueba 3D de D-FFP.

Tal y como se ilustra en la figura 5.8, el dron 0 realiza una trayectoria que dura 83,4 segundos desde el final del despegue hasta el principio del aterrizaje, con límites virtuales establecidos de 20 y 40 metros. Podemos observar en la gráfica como tiene lugar una ondulación vertical en su trayectória, aumentando la altura a 40 metros después de despegar para esquivar el primer dron (obstáculo ascendente), disminuyendo su altura hasta casi 20 metros para esquivar el segundo dron (obstáculo descendente), y volviendo a aumentar su

altura para esquivar el tercer dron que asciende y, por último, retomando a la altura objetivo de 30 metros antes de aterrizar.

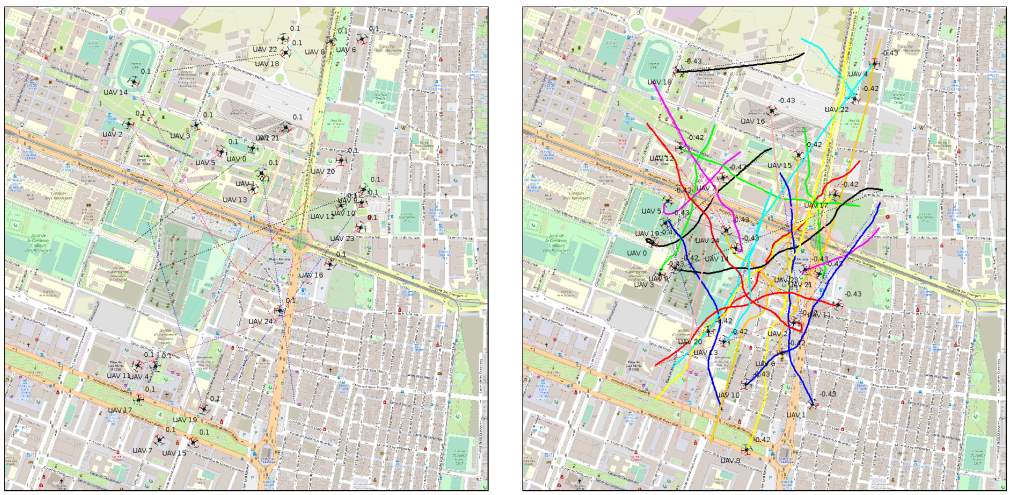
Una vez confirmada la funcionalidad de evasión en el eje z, se ha procedido a observar si, al incluir esta nueva característica, se alteran los resultados obtenidos en los cinco escenarios anteriores, comparando ambas trayectorias antes y después de implementar la evasión 3D. Los resultados de volver a ejecutar dichas misiones son muy similares a la anterior implementación en dos dimensiones del protocolo D-FFP, y la sobrecarga de tiempo y las distancias mínimas se mejoran por muy poco. Es por esto mismo que no hemos incluido estos resultados, ya que nos parecen redundantes.

Como último experimento, la herramienta de simulación ArduSim permite generar un escenario donde cada dron tendrá una trayectoria aleatoria en una zona predefinida. Esto nos permite simular con números elevados de drones usando nuestro protocolo, y observar el comportamiento caótico de gestionar tantos conflictos en un espacio aéreo compartido.



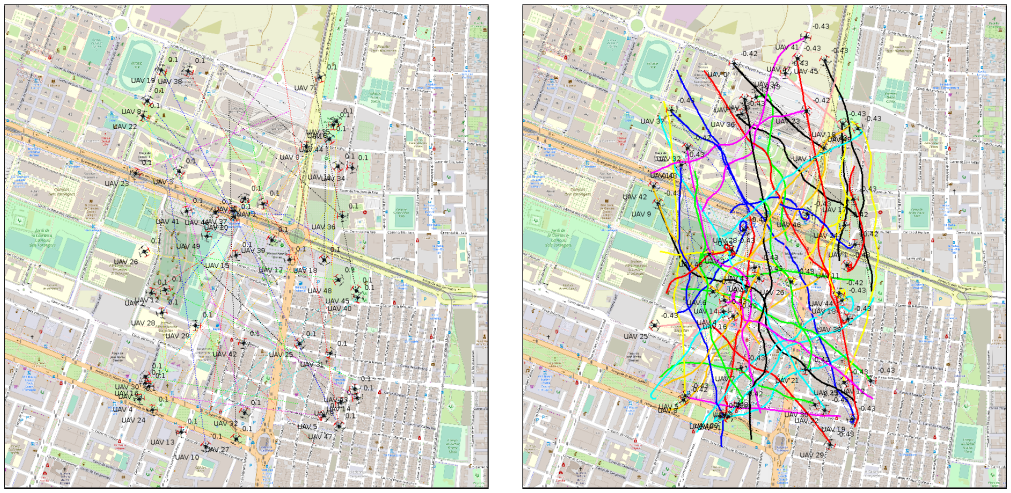
(a) Inicio. (b) Final.

Figura 5.9: Escenario con 10 drones y trayectorias aleatorias.



(a) Inicio. (b) Final.

Figura 5.10: Escenario con 25 drones y trayectorias aleatorias.



(a) Inicio. (b) Final.

Figura 5.11: Escenario con 50 drones y trayectorias aleatorias.

Cuadro 5.2: Resultados de los 3 escenarios con trayectorias aleatorias utilizando 10, 25 y 50 drones.

	10 UAVs	25 UAVs	50 UAVs
Tiempo total (s)	137,72	152,71	176,53
Promedio dist. mín. (m)	112,7	33,62	18,11
Posibles colisiones	0	4	18

Se han realizado tres experimentos con este tipo de escenarios, para 10 5.9, 25 5.10 y 50 5.11 drones. Representamos el posicionamiento inicial del escenario en la figura de la izquierda, antes de que despeguen los drones, y en la derecha el final de la misión con sus respectivas trayectorias. También hemos calculado tres características de cada escenario con su ejecución: el tiempo total de la misión, el promedio de la distancia mínima entre todos los drones, y el número de potenciales colisiones que se han detectado. Este último es un poco ambiguo, ya que han habido varios drones que se han contado como colisión por el simulador, donde la creación aleatoria de sus trayectorias han hecho que los puntos de despegue/aterrizaje de una pareja de drones tengan una ubicación muy cercana. Esto resulta en una aproximación nada más despegar y/o una aproximación al aterrizar.

Todos estos datos se representan en la tabla 5.2. Podemos observar como, a medida que aumenta el número de drones, el tiempo de misión también aumenta. Inversamente, el promedio de las distancias mínimas se reduce al haber aumentado el número de drones que comparten el mismo espacio aéreo. Sin embargo, al aumentar el número de drones en dicho espacio de tamaño predefinido para todos los casos, existen más colisiones potenciales al estar presentes más trayectorias y obstáculos a evadir.

Con estos datos podemos afirmar que el protocolo no garantiza al 100 % de la gestión de colisiones en un espacio muy reducido y con mucho tráfico; sin embargo, si aumentamos dicho espacio con trayectorias de mayor distancia, y separamos más los puntos de aterrizaje y despegue, podemos evitar este problema y realizamos dicha gestión con más garantías.

Capítulo 6

Conclusiones y trabajo futuro

El aumento de uso de aeronaves no tripuladas dentro de un mismo espacio urbano ha generado nuevos problemas a desarrollar por distintos campos tecnológicos. Concretamente, la gestión del espacio aéreo es compleja, y disponer de protocolos que permitan una evasión táctica antes colisiones inminentes es necesario. En esta memoria, se ha propuesto una nueva implementación para solucionar este problema, incorporando un novedoso protocolo aéreo de gestión de conflictos entre UAVs, que utiliza un campo direccional como principio.

Concretamente, se ha mejorado una solución anterior que establecía un campo de fuerza omnidireccional para la evitación y resolución de colisiones. Se ha estudiado como se podría mejorar el protocolo, reduciendo las fuerzas de repulsión generadas por dicho campo y añadiendo otro campo adicional con la propiedad direccional. Adicionalmente, se ha incorporado una nueva funcionalidad al nuevo protocolo para que tenga en cuenta las posiciones en el eje z de los obstáculos, transformando la radiación de los campos de fuerza a un plano en tres dimensiones, y estableciendo unos límites virtuales para evitar variaciones de altitud en contra de la legislación, o colisiones contra el suelo por un descenso excesivo.

Todas estas mejoras se han experimentado aplicando distintos escenarios que intentan abarcar una amplia variedad de situaciones de colisión entre drones, guardando información comparativa como la sobrecarga de tiempo de una misión, y la distancia mínima alcanzada. De esta forma, se ha comprobado la reducción en el tiempo total de vuelo en varios escenarios, respetando las distancias mínimas entre drones, en comparación con el uso del campo omnidireccional. Estas mejoras ofrecen una optimización del tiempo de vida de las baterías y su consumo, permitiendo al dron operar en el aire durante

más tiempo. También se han realizado experimentos con un número definido de drones que comparten un espacio aéreo predefinido y realizan una trayectoria aleatoria en dicho espacio, simulando así un caso real dentro de una población de espacio reducido.

Como posibles mejoras y trabajos futuros, se plantean las posibles soluciones y desarrollos:

- Ajuste más preciso de los parámetros del campo de fuerza para reducir las distancias lo máximo posible (valores cercanos a 10 metros). Debido al tiempo elevado que se necesita para automatizar las ejecuciones de varias tandas de misiones, no se ha logrado garantizar el mejor ajuste de parámetros posible, creando un margen de mejora para el futuro.

- Como se ha observado en la experimentación, hay una posible mejora en las distancias mínimas obtenidas de ejecutar un número elevado de drones dentro de un espacio aéreo reducido. Creemos que, siendo más estricto en los vectores de repulsión, y teniendo una mejor observación del comportamiento de la inercia, el número de colisiones obtenidos puede reducirse considerablemente.

- Para probar si realmente este protocolo ofrece unas buenas expectativas, podemos pasar la experimentación de un entorno de simulación a un entorno físico, realizando pruebas reales con drones físicos del tipo multirrotor.

Siglas

D-FFP *Directional-Force Field Protocol*

FFP *Force Field Protocol*

GRC Grupo de Redes de Computadoras

UAV *Unmanned Aerial Vehicle*

UAVs *Unmanned Aerial Vehicles*

UPV Universitat Politècnica de València

VANTs Vehículos Aéreos No Tripulados

Bibliografía

[1] Jamie Wubben et al. «FFP: A Force Field Protocol for the tactical management of UAV conflicts». En: *Ad Hoc Networks* 140 (2023), pág. 103078. ISSN: 1570-8705. DOI: https://doi.org/10.1016/j.adhoc.2022.103078. URL: https://www.sciencedirect.com/science/article/pii/S1570870522002505.

[2] *MATLAB - El lenguaje del cálculo técnico - MathWorks*. Accedido el 8 de Junio, 2023. URL: https://es.mathworks.com/products/matlab.html.

[3] Francisco Fabra et al. «ArduSim: Accurate and real-time multicopter simulation». En: *Simulation Modelling Practice and Theory* 87 (2018), págs. 170-190. ISSN: 1569-190X. DOI: https://doi.org/10.1016/j.simpat.2018.06.009. URL: https://www.sciencedirect.com/science/article/pii/S1569190X18300893.

[4] Flavia Causa, Armando Franzone y Giancarmine Fasano. «Strategic and Tactical Path Planning for Urban Air Mobility: Overview and Application to Real-World Use Cases». En: *Drones* 7.1 (2023). ISSN: 2504-446X. DOI: 10.3390/drones7010011. URL: https://www.mdpi.com/2504-446X/7/1/11.

[5] Jawad N. Yasin et al. «Unmanned Aerial Vehicles (UAVs): Collision Avoidance Systems and Approaches». En: *IEEE Access* 8 (2020), págs. 105139-105155. DOI: 10.1109/ACCESS.2020.3000064.

[6] Jung-Woo Park, Hyondong Oh y Min-Jea Tahk. «UAV collision avoidance based on geometric approach». En: *2008 SICE Annual Conference*. Loughborough, England: Loughborough University, sep. de 2008, págs. 2122-2126. DOI: 10.1109/SICE.2008.4655013.

[7] B. M. Albaker y N. A. Rahim. «A survey of collision avoidance approaches for unmanned aerial vehicles». En: *2009 International Conference for Technical Postgraduates (TECHPOS)*. Manhattan, New York, U.S: IEEE, 2009, págs. 1-7. DOI: 10.1109/TECHPOS.2009.5412074.

[8] Sara Pérez-Carabaza et al. «UAV trajectory optimization for Minimum Time Search with communication constraints and collision avoidance». En: *Engineering Applications of Artificial Intelligence* 85 (oct. de 2019), págs. 357-371. ISSN: 09521976. DOI: 10.1016/j.engappai.2019.06.002.

[9] Wang Min, Voos Holger y Su Daobilige. «Robust Online Obstacle Detection and Tracking for Collision-freeNavigation of Multirotor UAVs in Complex Environments». En: *2018 15th International Conference on Control, Automation, Robotics and Vision (ICARCV)* (2018), págs. 1228-1234. DOI: 10.1109/ICARCV.2018.8581330.

[10] Niels Balemans, Peter Hellinckx y Jan Steckel. «Predicting LiDAR Data From Sonar Images». En: *IEEE Access* 9 (2021), págs. 57897-57906.

[11] Jiayi Sun, Jun Tang y Songyang Lao. «Collision Avoidance for Cooperative UAVs With Optimized Artificial Potential Field Algorithm». En: *IEEE Access* 5 (2017), págs. 18382-18390. DOI: 10.1109/ACCESS.2017.2746752.

[12] Enming Wu et al. «Multi UAV Cluster Control Method Based on Virtual Core in Improved Artificial Potential Field». En: *IEEE Access* 8 (2020), págs. 131647-131661. DOI: 10.1109/ACCESS.2020.3009972.

[13] Ameni Azzabi y Khaled Nouri. «Path planning for autonomous mobile robot using the Potential Field method». En: *2017 International Conference on Advanced Systems and Electric Technologies (IC_ASET)*. 2017, págs. 389-394. DOI: 10.1109/ASET.2017.7983725.

[14] Changxin Huang et al. «Potential field method for persistent surveillance of multiple unmanned aerial vehicle sensors». En: *International Journal of Distributed Sensor Networks* 14.1 (2018), pág. 1550147718755069. DOI: 10.1177/1550147718755069. eprint: https://doi.org/10.1177/1550147718755069. URL: https://doi.org/10.1177/1550147718755069.

[15] Daegyun Choi, Kyuman Lee y Donghoon Kim. «Enhanced Potential Field-Based Collision Avoidance for Unmanned Aerial Vehicles in a Dynamic Environment». En: *AIAA Scitech 2020 Forum*. DOI: 10.2514/6.2020-0487. eprint: https://arc.aiaa.org/doi/pdf/10.2514/6.2020-0487. URL: https://arc.aiaa.org/doi/abs/10.2514/6.2020-0487.

[16] Cezary Kownacki y Leszek Ambroziak. «A New Multidimensional Repulsive Potential Field to Avoid Obstacles by Nonholonomic UAVs in Dynamic Environments». En: *Sensors* 21.22 (2021). ISSN: 1424-8220.

DOI: 10.3390/s21227495. URL: https://www.mdpi.com/1424-8220/21/22/7495.

[17] Edward Burgos y Subodh Bhandari. «Potential flow field navigation with virtual force field for UAS collision avoidance». En: *2016 International Conference on Unmanned Aircraft Systems (ICUAS)*. 2016, págs. 505-513. DOI: 10.1109/ICUAS.2016.7502641.

[18] Francisco Fabra et al. *ArduSim: Accurate and real-time multicopter simulation*. Accedido el 13 de Junio, 2023. URL: https://github.com/GRCDEV/ArduSim.

[19] *ArduCopter*. Accedido el 29 de Junio, 2023. URL: https://ardupilot.org/copter/.

[20] *ArduPilot*. Accedido el 29 de Junio, 2023. URL: https://ardupilot.org/.

[21] Jamie Wubben et al. «Accurate Landing of Unmanned Aerial Vehicles Using Ground Pattern Recognition». En: *Electronics* 8.12 (2019). ISSN: 2079-9292. DOI: 10.3390/electronics8121532. URL: https://www.mdpi.com/2079-9292/8/12/1532.

[22] *OpenJDK 17*. Accedido el 13 de Junio, 2023. URL: https://openjdk.org/projects/jdk/17/.

[23] *IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains*. Accedido el 13 de Junio, 2023. URL: https://www.jetbrains.com/idea/.

[24] *Ubuntu: Enterprise Open Source and Linux*. Accedido el 13 de Junio, 2023. URL: https://ubuntu.com/.

[25] *Google Earth*. Accedido el 13 de Junio, 2023. URL: https://www.google.es/intl/es/earth/index.html.

[26] *Python*. Accedido el 13 de Junio, 2023. URL: https://www.python.org/.