Máster en Ingeniería de Sistemas y Control

PLANIFICADOR DE TRAYECTORIAS DE VEHÍCULOS AÉREOS NO TRIPULADOS BASADO EN ALGORITMOS EVOLUTIVOS MULTI-OBJETIVO COMPUTACIONALMENTE EFICIENTES

Autor: Fernando Puech Helguero

Directores: Eva Besada Portas

Luis de la Torre Cubillo

Febrero 2016

Autorización

Autorizamos a la Universidad Complutense de Madrid (UCM) y a la Universidad Nacional de Educación a Distancia (UNED) a difundir y utilizar con fines académicos, no comerciales y mencionando expresamente a sus autores, tanto la memoria de este Trabajo Fin de Máster, como el código, la documentación y/o el prototipo desarrollado.

Firmado:

Resumen

Este Trabajo Fin de Máster consiste tanto en el estudio y análisis de algoritmos de optimización computacionalmente eficientes aplicados a un problema de cálculo de trayectorias de aviones no tripulados, como en su desarrollo e implementación. La principal problemática a resolver radica en la preparación de un algoritmo de optimización evolutivo que obtenga resultados en el menor tiempo posible.

El algoritmo de optimización objeto de estudio es de carácter evolutivo ya que artículos previos a este proyecto han demostrado que se comporta mejor que otros algoritmos de optimización como el algoritmo genético. Nuestro algoritmo optimiza la ruta a seguir por el o los aviones no tripulados con la información conocida antes del comienzo de la misión.

Para alcanzar los objetivos del proyecto, es decir, obtener un algoritmo que de soluciones en un corto espacio de tiempo y que, además, sea fácil de ejecutar e integrar en otros sistemas, se han utilizado tecnologías estándar como son el lenguaje de programación C/C++ y sus extensiones Open Multi-Processing (OpenMP), facilitando estas últimas el aprovechamiento de todos los recursos que ofrecen las CPU (unidades centrales de proceso) actuales.

Durante el desarrollo del proyecto se ha preparado una infraestructura con una serie de funciones, clases y tipos de datos sobre las que se ha construido y paralelizado el algoritmo de optimización diferencial evolutivo aplicado al problema de optimización mencionado anteriormente. Por último, cabe destacar que se ha construido una infraestructura extensible para que pueda ser reutilizada y ampliada en el futuro.

Palabras clave

- Optimización (Optimization)
- Planificación de trayectorias (Trajectory planning)
- Avión no tripulado (Unmanned Aircraft Vehicle, UAV)
- Algoritmo evolutivo diferencial (Differential evolution)
- Paralelización (Parallelization)
- OpenMP

Índice

Figuras

Tablas

1 Introducción

Vivimos en un mundo en el que cada vez se utilizan más vehículos no tripulados que deben ser capaces de llegar a sus destinos en entornos con situaciones cuya complejidad se incrementa día a día. Este incremento en la complejidad de las situaciones a las que se tiene que hacer frente, ha derivado en que los problemas de optimización de rutas cada vez cuenten con más objetivos y restricciones, hecho que complica enormemente la obtención de soluciones óptimas en tiempos razonables. De los múltiples algoritmos de optimización existentes, los siguientes tipos de algoritmos de optimización heurística han sido aplicados exitosamente en el cálculo de rutas de vehículos aéreos no tripulados:

- Algoritmos Genéticos [1] [2] [3]
- Nubes de partículas [4] [5] [6] [7]
- Evolución diferencial [8] [9] [10] [11]

Los algoritmos genéticos [12] parten de las ideas evolutivas de Darwin en las que una población de individuos mejora a partir de los individuos más adaptados al medio, puesto que estos sobreviven y se cruzan con otros individuos bien adaptados para crear una nueva población más adaptada todavía. De esta forma, las poblaciones evolucionan a mejor.

Los algoritmos de optimización de nubes de partículas, presentados por primera vez en un artículo de Kennedy y Eberhart [13], se basan en el comportamiento social de una bandada de pájaros que busca alimento. La optimización se consigue a partir de una población de soluciones candidatas, o partículas, moviendo éstas por todo el espacio de búsqueda según reglas matemáticas que tienen en cuenta la posición y la velocidad de las partículas. El movimiento de cada partícula se ve influido por la mejor posición hallada hasta el momento por dicha partícula, así como por las mejores posiciones globales encontradas por todas las partículas a medida que recorren el espacio de búsqueda.

Los algoritmos de evolución diferencial también surgieron en 1994-1996 y se deben a Price y Storn [14]. La idea detrás de estos algoritmos surgió cuando intentaban resolver el problema del ajuste de los coeficientes de polinomios de Chevicheff a un conjunto de datos y se les ocurrió aplicar la filosofía de perturbar una población de vectores utilizando diferencias entre los vectores de la propia población. La evolución diferencial ha resultado ser muy eficaz en la optimización de funciones objetivo cuyas variables son de tipo real.

En la Universidad Nacional de Educación a Distancia (UNED) y la Universidad Complutense de Madrid (UCM) se han desarrollado una serie de trabajos que abordan el problema de búsqueda de rutas óptimas de aviones no tripulados en escenarios de combate, con el objetivo de garantizar la supervivencia de los aparatos mientras estos realizan su misión en el menor tiempo posible. Entre estos trabajos, destaca [15] que describe un planificador de rutas para aviones no tripulados en el que se tienen en cuenta múltiples objetivos y una serie de restricciones que los aviones no tripulados han de cumplir en escenarios realistas. Además, dicho trabajo tiene en cuenta la optimización de las rutas antes de que los aviones estén volando (problema offline o desconectado) y la adaptación de las rutas una vez los aviones están volando (problema online o conectado). Además, en [16] se analiza la bondad de diversos algoritmos de optimización aplicados al problema de optimización de rutas no tripulados, comparando la eficiencia de los

algoritmos desde el punto de vista de su convergencia y de lo realista y realizable que puede ser la ruta obtenida para un modelo de avión no tripulado determinado. Además, en el artículo [17] se presenta un algoritmo paralelo y evolutivo que busca aprovechar mejor los recursos de los ordenadores con el objetivo de reducir el tiempo de búsqueda de soluciones sub-óptimas.

Partiendo de las ideas descritas en los mencionados trabajos, y con el fin de obtener un sistema computacionalmente eficiente, escalable y reutilizable en múltiples sistemas, en este trabajo se busca diseñar e implementar un planificador de rutas desconectado (offline) de alto rendimiento, aplicando tecnologías que permiten aprovechar los recursos de las unidades centrales de proceso (CPU por sus siglas en inglés) actuales para mejorar la eficiencia del algoritmo diferencial evolutivo sobre el que se sustenta el planificador.

1.1 Objetivos

El objetivo principal de este trabajo es la obtención de algoritmos de optimización computacionalmente eficientes, valorando las tecnologías de paralelismo disponibles y su facilidad de aplicación a los algoritmos de optimización. La idea que se persigue es obtener una respuesta óptima al cálculo de rutas offline en el menor tiempo posible. La importancia de la reducción de los tiempos de cálculo radica en el interés que existe a la hora de poder aplicar el algoritmo a situaciones cada vez más complejas, que permitan evaluar la bondad tanto del algoritmo como de las rutas obtenidas.

Como objetivo secundario se quiere que el algoritmo sea escalable, es decir, que en función de la carga de trabajo se adapte a los recursos de cómputo de los que se disponga.

Por último, este trabajo tiene como objetivo lograr algoritmos que, si bien, sólo engloban el cálculo de rutas en modo desconectado, sean extensibles y fáciles de integrar en otros sistemas más complejos. Por extensible se entiende que tenga una estructura que permita en el futuro introducir nuevos algoritmos de optimización y/o nuevos objetivos a cumplir.

Para lograr estos objetivos se estudiará y evaluará el algoritmo de planificación descrito en [15] e implementado por los autores de dicho trabajo en Matlab, y se determinarán las partes cuyo rendimiento puede ser significativamente mejorado utilizando técnicas de paralelización.

1.2 Organización de la memoria

Esta memoria se organiza en los siguientes capítulos:

- El capítulo 2 describe en detalle el problema de optimización que se quiere resolver y la representación matemática de los diferentes objetivos, además, se detalla el algoritmo de evolución diferencial y sus diferentes fases. Por último, se introducen una serie de herramientas que permiten la representación de algoritmos de manera computacionalmente eficiente.
- El capítulo 3 es el corazón de la memoria y en él se analizan los problemas encontrados para representar de manera computacionalmente eficiente el planificador de rutas y las soluciones y herramientas aplicadas en cada caso.
- El capítulo 4 presenta diferentes casos de planificación de rutas resueltos con los planificadores secuencial y paralelo obtenidos tras aplicar las soluciones algorítmicas descritas en el capítulo 0. Además, se realiza una comparación de los tiempos de

obtención de rutas sub-óptimas de los planificadores secuencial y paralelo desarrollados y de una versión previa del planificador desarrollada en Matlab.

- El capítulo 5 presenta las conclusiones de la memoria, así como líneas futuras de desarrollo.

2 Algoritmo de optimización y problema de optimización

Este capítulo describe en detalle el problema de optimización que hay que resolver, el algoritmo evolutivo diferencial y sus características y, por último, introduce una serie de herramientas que son útiles para representar de manera computacionalmente eficiente tanto el problema de optimización como el algoritmo evolutivo.

2.1 Problema de optimización

El problema que se va a resolver presenta una serie de aviones no tripulados que han de cumplir una misión consistente en partir de puntos conocidos sobre un mapa y llegar a destinos dados visitando una serie de localizaciones intermedias. Además, el trayecto que hay que recorrer puede estar localizado en una zona hostil donde existen múltiples sistemas antiaéreos que hay que evitar. Los sistemas antiaéreos tienen un radio de detección conocido y un radio de acción en los que son capaces de destruir el avión no tripulado. Por su parte, los aviones no tripulados han de llegar a su destino sin ser destruidos y tratando de no ser detectados. Para reducir la posibilidad de destrucción y/o detección el avión puede volar a baja altura para parapetarse con la orografía del terreno, pero sin chocar contra el suelo. Además, existen zonas de exclusión aérea en las que, por diversos motivos, el avión no puede entrar bajo ningún concepto, forzando que la ruta generada por el algoritmo evite dichas áreas.

El avión no tripulado tiene a su vez una serie de características técnicas que limitan las trayectorias que puede seguir y la distancia máxima que puede recorrer dada su capacidad de combustible y al consumo de éste. Estas limitaciones hacen que un objetivo de la misión sea llegar al destino utilizando el trayecto más corto posible.

Dado que existe más de un avión no tripulado que ha de cumplir su misión sobre la misma región del mapa, los aviones deberán esquivarse entre sí para no chocar. De aquí se deduce que no puede haber dos aviones no tripulados en el mismo instante en el mismo punto.

El problema de optimización tiene dos partes de cálculo diferenciadas, una parte desconectada, objeto de estudio en este proyecto y una parte conectada. La parte desconectada aborda la optimización de las rutas de los aviones no tripulados con la información de la que se dispone a priori, es decir, antes de que los aviones comiencen el vuelo, como puede ser la orografía de la zona a sobrevolar, las defensas antiaéreas ya conocidas o las zonas prohibidas. Tras haber calculado las mejores rutas con los datos de los que se dispone a priori, los aviones empiezan a volar y se resuelve el problema conectado. En la parte conectada los aviones no tripulados tienen que hacer frente a situaciones imprevistas, como es la aparición de defensas antiaéreas que previamente no se conocían, y que requieren que se adapten las rutas óptimas obtenidas durante la resolución de la parte desconectada del problema. En este proyecto no se aborda la resolución de la parte conectada del problema, pero hemos querido mencionarla para que el lector tenga una visión global del problema.

2.2 Problema desconectado (off-line) de optimización de rutas

En este apartado se detallan las propiedades más relevantes del problema de planificación elegido, las funciones objetivo utilizadas para formularlo como un problema de minimización multi-objetivo y el método que compara la bondad de diferentes soluciones del problema.

2.2.1 Representación del problema de optimización

Para formular nuestro problema de planificación como un problema de optimización multi-objetivo se consideran once objetivos distintos. Los diez primeros miden la calidad de la trayectoria que sigue cada avión no tripulado de manera independiente; seis de ellos son restricciones, mientras que los otros cuatro deben ser minimizados para obtener la mejor trayectoria o solución al problema de minimización. El último de los once objetivos es una restricción que mide la validez de una trayectoria cuando más de un avión no tripulado está sobrevolando el mapa o, dicho de otro modo, es una restricción que indica si las trayectorias calculadas chocan o no en algún punto.

Las trayectorias de los aviones no tripulados se representan como curvas s*pline* cúbicas [18] obtenidas a partir de la interpolación de una lista de puntos tridimensionales que cada avión no tripulado ha de visitar durante la misión. Esta lista de puntos, pertenecientes al espacio cartesiano 3D, es la codificación elegida en nuestro problema para definir de forma unívoca una posible solución del problema o trayectoria 3D.

La evaluación de la función objetivo requiere que las curvas se representen de manera discretizada sobre un número adecuado de N puntos cartesianos (x_i, y_i, z_i). Además, la función objetivo debe de tener en cuenta las propiedades de los diferentes elementos, es decir, el terreno, las zonas que no se pueden sobrevolar, las defensas antiaéreas y las características dinámicas propias del avión no tripulado. Éstas últimas se han extraído experimentalmente de un modelo dinámico complejo del avión para que la función objetivo sea más manejable.

2.2.2 Modelos matemáticos de las funciones objetivo

En este apartado se describe, siguiendo la nomenclatura recogida en [15], cada una de las funciones matemáticas utilizadas para representar los objetivos del problema.

2.2.2.1 Radio de giro mínimo

El avión no tripulado está limitado por diferentes características físicas, entre las que se encuentra su capacidad de giro. El radio mínimo de giro R_i^{min} de cada avión no tripulado en cada punto i en los que se ha discretizado su trayectoria depende del factor de carga máximo n_i^{max} en ese punto, la altitud z_i y la velocidad v_i.

R_i^{min} y n_i^{max} se calculan con las ecuaciones:

$$R_i^{min} = \frac{v_i^2}{g\sqrt{(n_i^{max})^2 - 1}} \quad (1)$$

$$n_i^{max} = 5.3809 \times 10^{-9} z_i^2 - 4.4291 \times 10^{-4} z_i + 6.1 \quad (2)$$

donde g es la fuerza de la gravedad y los coeficientes de la ecuación (2) se han extraído tras realizar múltiples simulaciones con el modelo matemático anteriormente mencionado.

Todos aquellos puntos de la trayectoria en los que el radio de giro es menor que el mínimo radio permitido, son penalizados según la expresión (3), donde el primer caso indica que no se cumple la restricción de que el radio de giro en el punto es mayor que el radio de giro mínimo en ese punto y N es el número de puntos sobre los que se ha discretizado la trayectoria.

$$\sum_{i=1}^{N} c_i^1 \text{ con } c_i^1 = \begin{cases} 1, & n_i \geq n_i^{\max} \\ 0, & \text{en cualquier otro caso} \end{cases} \quad (3)$$

2.2.2.2 Angulo máximo de ataque y ángulo mínimo de planeo

Los aviones no tripulados, dadas sus características físicas, están limitados por el máximo ángulo de ataque con el que pueden volar sin entrar en pérdida α_i y el mínimo ángulo de planeo β_i. De la simulación del modelo observa que ambos dependen de la altura siguiendo las ecuaciones:

$$\alpha_i = -1.5377 \times 10^{-10} z_i^2 - 2.6997 \times 10^{-5} z_i + 0.4211 \quad (4)$$

$$\beta_i = 2.5063 \times 10^{-9} z_i^2 - 6.3014 \times 10^{-6} z_i - 0.3257 \quad (5)$$

La pendiente que sigue el avión no tripulado en el punto i de la trayectoria se calcula como:

$$S_i = \frac{z_{i+1} - z_i}{\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2}} \quad (6)$$

Donde x_i, y_i, z_i son las componentes cartesianas del punto i, y $x_{i+1}, y_{i+1}, z_{i+1}$ son las componentes del siguiente punto de la trayectoria.

Todos los puntos de la trayectoria que no están en el rango (β_i, α_i) se penalizan siguiendo la siguiente ecuación que garantiza que el mínimo valor representa el cumplimiento de la restricción.

$$\sum_{i=1}^{N} c_i^2 \text{ con } c_i^2 = \begin{cases} 0, & \beta_i < S_i < \alpha_i \\ 1, & \text{en cualquier otro caso} \end{cases} \quad (7)$$

2.2.2.3 Consumo de combustible

Cada avión no tripulado tiene una capacidad limitada de combustible con la que llegar a su destino y pasar por todos los puntos intermedios especificados en la misión. Para calcular el consumo en cada punto de la trayectoria se distinguen las siguientes situaciones:

1. Cuando el avión vuela horizontalmente el consumo es dependiente de la altura z_i, según la ecuación:

$$F_i^{CH} = 9.553 \times 10^{-8} z_i^2 - 2.4524 \times 10^{-3} z_i + 29.5 \quad (8)$$

2. Cuando el avión vuela con su máximo ángulo de ataque el consumo sigue siendo dependiente de la altura z_i, según la ecuación:

$$F_i^{MS} = 1.6679 \times 10^{-11} z_i^3 - 2.4832 \times 10^{-7} z_i^2 - 4.259 \times 10^{-3} z_i + 87.881 \quad (9)$$

3. Cuando el avión vuela con su máximo ángulo de inclinación lateral el consumo se define según la ecuación:

$$F_i^{MT} = -3.0435 \times 10^{-1} (n_i^{\max})^2 + 16.552 \times n_i^{\max} + 0.3565 \quad (10)$$

4. Cuando el avión está ascendiendo se usa la expresión (11), que combina valores obtenidos en los tres casos anteriores.

$$FC_i^A = F_i^{CH} + \frac{S_i}{\alpha_i}\left(F_i^{MS} - F_i^{CH}\right) \tag{11}$$

5. Cuando el avión está girando, pero no ascendiendo, se utiliza:

$$FC_i^T = F_i^{CH} + \frac{n_i}{n_i^{\max}}\left(F_i^{MT} - F_i^{CH}\right) \tag{12}$$

6. Cuando el avión vuela sin girar y descendiendo el consumo se considera igual al consumo vuelo horizontal F_i^{CH}.

Se supone que el consumo entre dos puntos de la trayectoria es constante y se suman los productos del consumo en cada punto por el tiempo necesario para llegar de un punto al siguiente de la trayectoria. El tiempo se calcula como:

$$\Delta t_i = \frac{\sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 + (z_{i+1} - z_i)^2}}{v} \tag{13}$$

Como restricción se utiliza la siguiente expresión que sólo penaliza la función objetivo siempre y cuando el consumo sea mayor que la cantidad de combustible embarcado inicialmente en el avión no tripulado.

$$\max\left(\sum_{i=1}^{N} FC_i \Delta t_i - CombustibleEmbarcado, 0\right) \tag{14}$$

2.2.2.4 Evitar el choque contra el suelo

Se comprueba la validez cada ruta comprobando que la altitud de vuelo del avión no tripulado es siempre superior a la altura del suelo:

$$\sum_{i=1}^{N} c_i^4 \text{ con } c_i^4 = \begin{cases} 1, z_i \leq \text{mapa}(x_i, y_i) \\ 0, \text{en otro caso} \end{cases} \tag{15}$$

donde la función $\text{mapa}(x_i, y_i)$ obtiene la altura del terreno en el punto (x_i, y_i).

2.2.2.5 Permanecer en el área de vuelo

Esta función objetivo penaliza los puntos de la trayectoria que están fuera de los límites definidos por el mapa sobre el que se están optimizando las rutas.

$$\sum_{i=1}^{N} c_i^5 \text{ con } c_i^5 = \begin{cases} 0, \text{ dentroDelMapa}(x_i, y_i) \\ 1, \text{en otro caso} \end{cases} \tag{16}$$

$$\text{dentroDelMapa}(x_i, y_i) = (x_l^m \leq x_i \leq x_u^m) \wedge (\text{y}_\text{l}^\text{m} \leq y_i \leq y_u^m) \quad (17)$$

donde x_l^m es el límite izquierdo del mapa, x_u^m es el límite derecho del mapa, y_l^m es el límite inferior del mapa e y_u^m es el límite superior del mapa.

2.2.2.6 Evitar las zonas de exclusión aérea

Las zonas de exclusión aérea, en inglés No Flying Zones (NFZ), son zonas rectangulares del mapa en las que el avión no tripulado no debe entrar bajo ningún concepto. Para tener en cuenta este objetivo se calcula, en los puntos de la discretización de la trayectoria del UAV que caen dentro de la NFZ, la distancia del punto a la frontera más cercana de la NFZ. De esta forma, este objetivo puede distinguir entre dos rutas con el mismo número de puntos dentro de la NFZ, pero con distancias diferentes a sus bordes:

$$\sum_{i=1}^{N}\sum_{j=1}^{M} d_i^j \text{ con } d_i^j = \begin{cases} \min_{k,a}\left(d_i^{j,k,a}\right), \text{ si enNFZ}(i,j) \\ 0, \text{ en cualquier otro caso} \end{cases} \quad (18)$$

$$M = \text{número de NFZs} \quad (19)$$
$$d_i^{j,k,a} = \left|a_k^{NFZ,j} < a_i\right|, \text{con } k = l, u \wedge a = x, y$$

$$enNFZ(i,j) = \left(x_l^{NFZ,j} \leq x_i \leq x_u^{NFZ,j}\right) \wedge \left(y_l^{NFZ,j} \leq y_i \leq y_u^{NFZ,j}\right) \quad (20)$$

donde $x_l^{NFZ,j}$ es el extremo izquierdo del área que abarca la j-esima NFZ, $x_u^{NFZ,j}$ su extremo derecho, $y_l^{NFZ,j}$ su extremo inferior e $y_u^{NFZ,j}$ su extremo superior.

2.2.2.7 Camino más corto

Se busca el camino más corto porque supone menor consumo de combustible, completar la misión en menor tiempo y un menor riesgo de encontrarse con amenazas no conocidas. Se minimiza la longitud de la ruta normalizada por la distancia mínima entre el origen y el destino.

$$PLR = \frac{\sum_{i=1}^{N-1}\sqrt{(x_{i+1}-x_i)^2 + (y_{i+1}-y_i)^2 + (z_{i+1}-z_i)^2}}{l_{\min}} \quad (21)$$

Con $l_{\min}$ la distancia mínima entre el punto de origen de la misión y el punto de destino pasando por todos los puntos marcados para la misión.

2.2.2.8 Probabilidad de destrucción mínima

Las trayectorias más seguras son las que acumulan menor probabilidad de destrucción y detección del UAV. La función de probabilidad de destrucción depende de las unidades de defensa antiaérea consideradas, ADUs por sus siglas en inglés. Para cada punto de la trayectoria cada ADU tiene una cierta posibilidad de destruir el avión no tripulado. Las defensas antiaéreas tienen un radio de máximo riesgo para el avión no tripulado y un radio de detección máximo.

El valor de la probabilidad de destrucción que hay que minimizar se calcula con la siguiente expresión:

$$PDestrucción = 1 - \prod_{i=1}^{N}\prod_{j=1}^{A}\left(1 - PK_i^j \Delta t_i\right) \quad (22)$$

donde

$$A = \text{número de ADUs}$$
$$N = \text{número de puntos de la trayectoria} \quad (23)$$

$$PK_i^j = \begin{cases} PK_i^j, \text{ si } enZonaAltoRiesgo(i,j) \\ 0, \text{ en cualquier otro caso} \end{cases} \quad (24)$$

$$enZonaAltoRiesgo(i,j) = \left(R_{ij} \leq R_{PKmax}^j\right) \quad (25)$$

$$PK_i^j = \begin{cases} \text{coefDeteccion , si existeLV(i,j)} \\ 0, \text{ en cualquier otro caso} \end{cases} \quad (26)$$

$$existeLV(i,j) = \begin{cases} 1, \text{ si el ADU j ve la posición i del UAV} \\ 0, \text{ en cualquier otro caso} \end{cases} \quad (27)$$

El coeficiente de destrucción se calcula en función de la distancia entre el punto de la trayectoria y el sistema antiaéreo, y el número de veces que la trayectoria entra en la zona de máximo riesgo.

2.2.2.9 Probabilidad de detección mínima

En este punto se calcula la probabilidad de detección del avión en función de su sección equivalente de radar, Radar Cross Section (RCS) en inglés, siempre y cuando esté dentro del radio de detección del sistema antiaéreo. Para el cálculo de la sección radar en cada punto i de la trayectoria se considera que el avión es un elipsoide de semiejes a, b, c

$$RCS_{ij} = \frac{\pi a^2 b^2 c^2}{\sqrt{\left(a\alpha_z\beta_\phi\right)^2 + \left(b\alpha_z\alpha_\phi\right)^2 + (c\beta_z)^2}} \quad (28)$$

$$\text{con} \quad \begin{matrix} \alpha_z = \sin\left(az_{ij}^e\right) \text{ y } \beta_z = \cos\left(az_{ij}^e\right) \\ \alpha_\phi = \sin\left(\phi_{ij}^e\right) \text{ y } \beta_\phi = \cos\left(\phi_{ij}^e\right) \end{matrix} \quad (29)$$

Donde az_{ij}^e es el ángulo entre la velocidad del avión en el punto i y el segmento que une dicho punto con la posición del radar j, y ϕ_{ij}^e se obtiene con la siguiente función:

$$\phi_{ij}^e = \phi_{ij} - \arctan\left(\frac{tan\left(el_{ij}\right)}{sin\left(az_{ij}\right)}\right) \quad (30)$$

Con ϕ_{ij} la inclinación, el_{ij} la altura y az_{ij} el acimut del avión en su posición i respecto al ADU j. El objetivo a minimizar se calcula como

$$PDR = 1 - \prod_{i=1}^{N}\prod_{(j=1)}^{A}\left(1 - PD_i^j \Delta t_i\right) \quad (31)$$

$$A = \text{número de ADUs}$$
$$N = \text{número de puntos de la trayectoria} \quad (32)$$

$$\text{con } PD_i^j = \begin{cases} 0, \text{ si } R_{ij} > R_{Dmax}^j \vee \neg existeLV(i,j) \\ \dfrac{1}{1+\zeta_2^j\left(\dfrac{R_{ij}^4}{RCS_{ij}}\right)^{\zeta_1^j}}, \text{ en cualquier otro caso} \end{cases} \quad (33)$$

Donde ζ_2^j y ζ_1^j son parámetros propios del radar.

2.2.2.10 Altitud mínima de vuelo

Con este objetivo se busca favorecer que los aviones no tripulados vuelen lo más bajo posible para aprovechar la orografía y ocultarse de las defensas antiaéreas. Para logarlo, se acumulan las diferencias de altura entre los puntos de la trayectoria y el terreno.

$$\sum_{i=1}^{N} c_i^{10} \quad \text{con } c_i^{10} = \begin{cases} z_i - mapa(x_i, y_i), \text{ si } z_i \geq mapa(x_i, y_i) \\ 0, \text{ en cualquier otro caso} \end{cases} \quad (34)$$

El segundo caso, con valor 0, evita que los UAV intente ocultarse de las ADU volando por debajo del terreno.

2.2.2.11 Evitar colisiones entre los aviones

Para garantizar que los aviones no tripulados no chocan entre sí, se comprueba que la distancia entre cada punto de las trayectorias de cada par de UAV sea mayor que un mínimo aceptable. Si no es así entonces se procede a comprobar que los instantes de tiempo en los que cada avión pasa por el punto crítico sean diferentes.

Para realizar las comprobaciones se compara la trayectoria del avión no tripulado u con las trayectorias del resto de aviones no tripulados v. La comparación se realiza con cada punto de las trayectorias; si la distancia d_{ij}^{uv} entre el punto i de la trayectoria del avión u y el punto j de la trayectoria del avión v es menor que un valor mínimo $d^{\min}$, se comprueba que los tiempos de llegada a esos puntos (t_i^u y t_j^v) son lo suficientemente distintos, es decir, que su diferencia sea mayor que $t_{\min}$. El tiempo mínimo y la distancia mínima son parámetros configurables del optimizador.

Este objetivo se calcula a partir de las trayectorias Pareto óptimas que tienen en cuenta los diez primeros objetivos y siguiendo las siguientes expresiones:

$$\sum_{\forall v \neq u} \sum_{i=1}^{N^u} \sum_{j=1}^{N^v} c_{ij}^{11} = \begin{cases} 1, \text{ si } \ d_{ij}^{uv} < d_{\min} \wedge \left|t_i^u - t_j^v\right| < t_{\min} \\ 0, \text{ en cualquier otro caso} \end{cases} \quad (35)$$

$$d_{ij}^{uv} = \sqrt{\left(x_i^u - x_j^v\right)^2 + \left(y_i^u - y_j^v\right)^2 + \left(z_i^u - z_j^v\right)^2} \quad (36)$$

Donde N^u es el número de puntos de la trayectoria del UAV u y N^v el número de puntos de la mejor trayectoria del UAV v.

2.2.3 Combinación de los objetivos

Para comparar dos trayectorias de un mismo UAV en función de los 11 objetivos anteriores se podría utilizar una combinación lineal de los valores de los objetivos asociados a cada trayectoria

para obtener un valor indicativo de su bondad. Sin embargo, esta forma de proceder, que convierte el problema multi-objetivo en un problema mono-objetivo, requiere la normalización de los objetivos y conduce hacia las soluciones que tienen una mejor combinación lineal, por lo que se utiliza una alternativa basada en la definición de comparación Pareto con prioridades diferenciadas para los diferentes grupos de objetivos [19].

Las prioridades que se utilizan son parámetros del planificador, donde el menor valor numérico asociado a un objetivo indica una prioridad más elevada. Por lo tanto, es conveniente poner en el primer nivel los objetivos asociados a las restricciones y en los niveles sucesivos los objetivos asociados a cosas que se quieren mejorar en la medida de lo posible, siempre y cuando se hayan mejorado los objetivos situados en niveles superiores.

Las prioridades que el planificador utiliza por defecto son las siguientes:

- Prioridad 1:
  - Choque contra el suelo
  - Ángulo de giro mínimo
  - No rebasar los límites del mapa
  - Máximo ángulo de ataque y ángulo mínimo de planeo
  - Probabilidad de destrucción
  - Consumo de combustible
  - No entrar en las NFZs
  - No chocar contra otros aviones
- Prioridad 2:
  - Longitud de la trayectoria
- Prioridad 3:
  - Altitud mínima de vuelo
  - Probabilidad de detección

A la hora de ordenar las trayectorias obtenidas para un UAV según los valores obtenidos con sus funciones objetivo, se construyen diferentes frente de Pareto comprobando para cada trayectoria qué otras trayectorias de la comparativa, lo dominan (son mejores que ella), teniendo en cuenta las prioridades descritas anteriormente. Una vez que se ha calculado para cada trayectoria el valor de la dominancia respecto al resto de las trayectorias, se utiliza como bondad de ordenación de los individuos la cuenta el número total de trayectorias que dominan a cada una.

2.3 Algoritmo de evolución diferencial

Para resolver el problema de optimización se ha utilizado un algoritmo de evolución diferencial. El algoritmo diferencial consiste en perturbar los individuos (soluciones) de una población (conjunto de soluciones) inicial utilizando las diferencias de valores existentes entre parejas de individuos de la misma población. Al adaptar las perturbaciones a las diferencias de parejas de individuos existentes, las direcciones de búsqueda y el tamaño máximo de los incrementos evolucionan de acuerdo con los individuos existentes en la población en cada iteración del algoritmo. Además, su funcionalidad se complementa con una etapa de combinación de los valores de las soluciones existentes y las perturbadas y otra etapa en la que se seleccionan, de

entre las soluciones perturbadas y las ya obtenidas, los individuos que formarán parte de la población que se utilizará en la siguiente iteración.

En la población que hay que generar o que es manipulada por el algoritmo evolutivo se codifican las coordenadas de los puntos de control de los splines que definen la trayectoria que un UAV tiene que seguir. Por lo tanto, cada individuo o solución del problema se encontrará codificado, a bajo nivel, como un vector en el que hay tantos elementos como el producto de la longitud de la lista de puntos de control de la trayectoria y el número de dimensiones que del punto (en nuestro caso 3). En este apartado, y de forma genérica, utilizaremos la siguiente nomenclatura: $g_r^{q,k}$ representa el *r*-ésimo elemento (gen) del q-ésimo vector de soluciones (individuo) de la población formada en la *k*-ésima iteración del algoritmo, $I^{q,k}$ al q-ésimo vector de soluciones (individuo) de la población formada en la *k*-ésima iteración del algoritmo, y $V^{q,k}$ al valor que almacena el valor de las 11 funciones objetivo asociadas a la trayectoria definida mediante $I^{q,k}$.

Las fases del algoritmo de evolución diferencial se pueden resumir en:

- Generación y evaluación de la población inicial.
- Iteración sobre las siguientes tareas:
  - Inclusión de inmigrantes.
  - Perturbación de la población.
  - Combinación de las soluciones perturbadas con las existentes en la iteración anterior.
  - Evaluación de las nuevas soluciones.
  - Elección de los individuos.

En los siguientes apartados se describen en detalle las labores realizadas en cada etapa.

2.3.1 Generación y evaluación de la población inicial

Aunque la población de puntos de control (es decir los valores de $g_r^{q,0}$) se pueden generar de manera totalmente aleatoria, existe una información previa, determinada por las características de la misión del UAV, que puede ser utilizada para restringir el espacio de búsqueda del algoritmo. Entre dicha información se definen, como ya se ha explicado, una serie de localizaciones en el mapa, entre las que se incluyen el punto de partida y el punto de destino. Dado que los puntos predefinidos han de ser visitados por el avión, la generación de la población inicial utiliza dicha información en una fase de configuración previa, en la que calculan una serie de parámetros para generar, indirectamente, los valores de $g_r^{q,0}$ (y por lo tanto sus correspondientes trayectorias) de manera aleatoria.

La generación de valores se realiza en dos etapas. En la primera se generan aleatoriamente, utilizando los rangos pre-calculados para la misión, los puntos de control de las trayectorias en coordenadas polares. En la segunda, se realiza la conversión de los puntos codificados en coordenadas polares en su representación cartesiana en 3 dimensiones. Se sigue esta forma de proceder, consistente en una generación aleatoria entre rangos pre-calculados en polares y codificación de la solución en cartesianas, ya que en los estudios realizador para [15] demostraron que producía las mejores soluciones.

Una vez que se ha generado la población inicial (valores de $g_r^{q,0}$ y por lo tanto de $I^{q,0}$), se calcula el valor de la función objetivo para dicha población (es decir los valores de $V^{q,k}$), que se utilizará para comparar las soluciones de la población inicial con las soluciones de la población perturbada.

2.3.2 Inclusión de inmigrantes

En esta etapa se genera una pequeña población de inmigrantes de manera aleatoria y se evalúan sus individuos. Esta población y sus valores objetivos se incluyen en la población inicial o en la población de la iteración $k-1$, y ayuda a renovar la información existente en la población, reduciendo la posibilidad de que el algoritmo evolutivo quede capturado en mínimos locales.

2.3.3 Perturbación de la población

Se obtienen los valores iniciales de los genes $g_r^{q,k}$ de una nueva población a partir de la mutación de los valores de los genes $g_r^{q,k-1}$ de la población previa. El proceso de mutación se puede realizar con cualquiera de los siguientes métodos:

1. Mutación aleatoria fija. Se mutan el valor de los genes eligiendo de manera aleatoria triadas de individuos ($I^{a,k-1}, I^{b,k-1}, I^{c,k-1}$) pertenecientes a la población anterior $k-1$, y modificando el valor del individuo $I^{a,k-1}$ con un valor proporcional a la diferencia de los valores entre $I^{b,k-1}$ y $I^{c,k-1}$. Es decir, el valor de cada gen $g_r^{q,k}$ se calcula con la siguiente expresión :

$$g_r^{q,k} = g_r^{a,k-1} + fm(g_r^{b,k-1} - g_r^{c,k-1}) \tag{37}$$

Donde

$$\begin{aligned} g_r^{q,k} &= \text{gen } r \text{ del individuo } q \text{ de la nueva población } k \\ g_r^{a,k-1} &= \text{gen } r \text{ del individuo aleatorio } a \text{ de la población } k-1 \\ g_r^{b,k-1} &= \text{gen } r \text{ del individuo aleatorio } b \text{ de la población } k-1 \\ g_r^{c,k-1} &= \text{gen } r \text{ del individuo aleatorio } c \text{ de la población } k-1 \\ fm &= \text{factor de mutación} \end{aligned} \tag{38}$$

2. Mutación aleatoria con tramado (dithering). Se sigue un procedimiento similar al anterior, salvo a la hora de elegir el valor del factor de proporcionalidad por el que se multiplica la diferencia entre los genes. En este tipo de mutación, dicho factor toma un valor generado de forma aleatoria para todos los genes asociados al mismo individuo q. Por lo tanto, el valor de cada gen $g_r^{q,k}$ se calcula con la siguiente expresión :

$$g_r^{q,k} = g_r^{a,k-1} + fm^q(g_r^{b,k-1} - g_r^{c,k-1}) \tag{39}$$

Donde

$$fm^q = \text{aleatorio uniforme} \in [fm_{min}, fm_{max}] \text{ para el individuo } q \tag{40}$$

3. Mutación aleatoria con ruido (jitter). El procedimiento es similar a los anteriores, aunque en este caso se genera un factor de proporcionalidad aleatorio diferente para cada individuo q y gen r. Es decir, el valor de cada gen $g_r^{q,k}$ se calcula con la siguiente expresión:

$$g_r^{q,k} = g_r^{a,k-1} + fm_r^q(g_r^{b,k-1} - g_r^{c,k-1}) \quad (41)$$

Donde

$$fm_r^q = \text{aleatorio uniforme} \in [fm_{min}, fm_{max}] \text{ para el individuo } q \text{ y el gen } r \quad (42)$$

4. Perturbación fija sobre los mejores individuos. Se sigue un procedimiento similar al de mutación aleatoria fija, salvo a la hora de elegir el individuo base $I^{d,k-1}$ cuyo valor se ve modificado por una diferencia escalada de los dos restantes ($I^{b,k-1}$ y $I^{c,k-1}$). En este caso, el individuo $I^{a,k-1}$ se elige entre aquellos que pertenecen al primer frente Pareto de la población en $k-1$. Por lo tanto, el valor de cada gen $g_r^{q,k}$ se calcula con la siguiente expresión:

$$g_r^{q,k} = g_r^{d,k-1} + fm(g_r^{b,k-1} - g_r^{c,k-1}) \quad (43)$$

Donde

$$\begin{aligned} g_r^{q,k} &= \text{gen } r \text{ del individuo } q \text{ de la nueva población } k \\ g_r^{d,k-1} &= \text{gen } r \text{ del individuo aleatorio } a \\ &\quad \in \{\text{mejores individuos de la población } k-1\} \\ g_r^{b,k-1} &= \text{gen } r \text{ del individuo aleatorio } b \text{ de la población } k-1 \\ g_r^{c,k-1} &= \text{gen } r \text{ del individuo aleatorio } c \text{ de la población } k-1 \\ fm &= \text{factor de mutación} \end{aligned} \quad (44)$$

5. Perturbación con tramado (dither) sobre los mejores individuos. Es un híbrido entre la perturbación fija sobre los mejores individuos y la mutación aleatoria con tramado. De la primera hereda el procedimiento de elegir el individuo base y de la segunda el método de elegir el factor de proporcionalidad. En este caso, el procedimiento a seguir es:

$$g_r^{q,k} = g_r^{d,k-1} + fm^i(g_r^{b,k-1} - g_r^{c,k-1}) \quad (45)$$

6. Perturbación con ruido (jitter) sobre los mejores individuos. Híbrida la perturbación fija sobre los mejores individuos y la mutación aleatoria con ruido heredando el procedimiento de elegir el individuo base de la primera y el método de elegir el factor de proporcionalidad de la segunda. En este caso, el procedimiento a seguir es:

$$g_r^{q,k} = g_r^{d,k-1} + fm_r^i(g_r^{b,k-1} - g_r^{c,k-1}) \quad (46)$$

2.3.4 Combinación de las soluciones

Tras haber obtenido los valores de todos los genes de la nueva población se procede a cruzar sus valores con los de la población original, es decir, los de la población obtenida en la iteración anterior $(k-1)$

Para llevarlo a cabo, se utiliza un operador de cruce uniforme. Este operador tiene un factor aleatorio de cruce que controla los genes que perduran en la nueva población. Se procede generando de manera aleatoria un peso p_r^q para cada gen $g_r^{q,k}$ de la nueva población y si el valor

del peso es mayor que el factor de cruce f_c se sustituye el valor del gen $g_r^{q,k}$ de la población generada por el valor del gen $g_r^{q,k-1}$ de la población anterior:

$$g_r^{q,k} = \begin{cases} g_r^{q,k}, \text{si } p_r^q < f_c \\ g_r^{q,k-1}, \text{si } p_r^q \geq f_c \end{cases} \quad (47)$$

Con

$$p_r^q = aleatorio\ uniforme \in [0..1] \quad (48)$$

2.3.5 Evaluación de la función objetivo

En esta fase se obtienen los valores de la función objetivo $V^{q,k}$ para todos y cada uno de los individuos $I^{q,k}$ de la población k obtenida tras la perturbación y el cruce con la población $k-1$.

2.3.6 Elección de los individuos

En esta fase se eligen los individuos $I^{q,k}$ que van a formar definitivamente parte de la población de la iteración k. La elección se realiza entre parejas de individuos pertenecientes a la población anterior y los pertenecientes a la población perturbada en la iteración actual, quedándonos de esta manera como individuo $I^{q,k}$ de la nueva población:

$$I^{q,k} = \begin{cases} I^{q,k}, \text{si} \quad V^{q,k} \text{ mejor que } V^{q,k-1} \\ I^{q,k-1}, \text{ en cualquier otro caso} \end{cases} \quad (49)$$

Para saber si el individuo $I^{q,k}$ es mejor que $I^{q,k-1}$, en función de los valores de sus múltiples objetivos ($V^{q,k}$ y $V^{q,k-1}$) , se calcula una matriz de dominancia $M_{DOM}(V^{q,k}, V^{q,k-1})$ para cada pareja de individuos de las poblaciones k y $k-1$ con el objeto de detectar qué individuo domina en la pareja. La matriz de dominancia es una matriz $m \times m$ con:

$$m = \text{número de individuos comparados} \quad (50)$$

y cada elemento de la matriz:

$$M_{DOM}\left(V^{1,*}, \ldots, V^{m,*}\right)_{\text{s,t}} = \begin{cases} 1, \text{si } V^{s,*} \text{ es mejor que} V^{t,*} \text{ con } s \neq t \\ 0, \text{en cualquier otro caso} \end{cases} \quad (51)$$

La comprobación de qué individuo de cada pareja posible es mejor, se realiza teniendo en cuenta las prioridades de los diferentes objetivos.

La comprobación dentro de un grupo de objetivos con la misma prioridad se realiza de la siguiente manera:

1. Se buscan los objetivos que el primer individuo no cumple y se compara lo lejos que están de cumplirse con lo lejos que están de cumplirse los mismos objetivos del segundo individuo. Si la distancia del primer individuo es menor que la del segundo individuo, y no existe ningún objetivo cuya distancia del segundo individuo sea menor que la del primero, entonces el primer individuo domina al segundo y se marca el segundo individuo de forma que no haya que seguir comparándolo con el primero.
2. Se busca el número de objetivos que cumple el primer individuo y se compara con el número de objetivos que cumple el segundo. Si el primer individuo cumple más

objetivos que el segundo, el primero domina al segundo y se marca el segundo de forma que no haya que seguir comparándolo con el primero.

3. Se comparan aquellos objetivos que ambos individuos cumplen. Para ello se compara la diferencia entre los valores de los objetivos del primer individuo y los del segundo. Si existe algún objetivo cuya diferencia sea mayor que cero y ningún objetivo cuya diferencia sea menor que cero, entonces el primer individuo domina al segundo y se marca el segundo de forma que no haya que seguir comparándolo con el primero.
4. Antes de comenzar el proceso con el siguiente grupo de objetivos se marcan todos aquellos individuos que dominan al primero, o que son dominados por el primero, para que no se estudien en el siguiente grupo de objetivos.

Como resultado de este proceso se acaba determinando qué individuos dominan a otros, son dominados por otros, o no mantienen ninguna relación de dominancia con otros.

2.4 Herramientas para mejorar el rendimiento

En este capítulo se presentan una serie de tecnologías que se pueden utilizar para obtener algoritmos computacionalmente eficientes.

2.4.1 Lenguajes de programación

Existen múltiples lenguajes de programación que permiten desarrollar cualquier algoritmo, pero, de entre todos ellos, se suele considerar que C o C++ son los más indicados para obtener un alto rendimiento. Esto es debido a que C y C++ son lenguajes de alto nivel muy flexibles y adaptables que ofrecen al programador un control muy fino del uso de la memoria y otros recursos hardware del ordenador. La prueba de ello es que C y C++ suelen ser los lenguajes utilizados en la industria [20] para el desarrollo de controladores hardware, además de ser C el lenguaje de referencia en los sistemas embebidos dada su compatibilidad y sencillez [21].

2.4.2 Paralelismo

Un programa paralelo es aquél que ejecuta varias instrucciones simultáneamente. Para lograr ejecutar más de una instrucción simultáneamente, el programa se debe dividir en subprogramas que se ejecuten en diferentes procesadores. Para lograr la ejecución en paralelo se puede utilizar un ordenador con varios procesadores, varios ordenadores con un solo procesador conectados por una red o una solución intermedia.

La razón principal por la que se busca ejecutar código en paralelo es reducir el tiempo que este tarda en finalizar su tarea. Si consideramos que el tiempo (*T)* que tarda en ejecutarse un programa en un procesador es función del número de instrucciones (*I)* que tiene el programa por el tiempo medio (t_m) que tarda el procesador en ejecutar una instrucción:

$$T = I \times t_m \tag{52}$$

Sería lógico pensar que utilizando más procesadores podríamos dividir la cantidad de instrucciones a ejecutar entre el número de procesadores (N_p) y, en el mejor de los casos, el nuevo tiempo para ejecutar el programa (T_p) sería:

$$T_p = \frac{I}{N_p} \times t_m = \frac{T}{N_p} \tag{53}$$

La realidad es que no todas las instrucciones de un programa o algoritmo se pueden ejecutar en paralelo y por eso no basta con añadir más procesadores si no que hay que adaptar los algoritmos para que estos puedan aprovechar al máximo los procesadores disponibles. Al cambiar los algoritmos a su versión paralela hay que tener en cuenta que se gasta tiempo en sincronizar los diferentes procesadores, por lo que, en algunas situaciones, o cuando la paralelización no se ha llevado a cabo de manera cuidadosa, las versiones secuenciales del algoritmo pueden ser más eficientes que la versión paralela.

Se ha mencionado que para ejecutar un programa en paralelo éste ha de dividirse en varios procesadores dentro de un mismo ordenador o en varios ordenadores conectados mediante una red. Esta distinción evidencia dos paradigmas a la hora de desarrollar programas de manera paralela, uno es el paradigma de memoria compartida y el otro del paradigma de memoria distribuida [22].

En el paradigma de memoria distribuida cada procesador tiene su zona de memoria independiente del resto de procesadores. En este paradigma, cuando hay que compartir información, ésta se ha de enviar de un procesador a otro mediante algún tipo de mensajería. La lógica encargada de repartir la información ha de ser añadida a los algoritmos, hecho que complica sensiblemente la traducción de un programa secuencial a uno paralelo. La gran ventaja del modelo de memoria distribuida es que tanto el número de procesadores como la memoria son escalables, es decir, un programa podría tener acceso a más recursos de manera sencilla.

En el paradigma de memoria compartida todos los procesadores tienen acceso al mismo espacio de memoria, lo que facilita el acceso a la información desde los diferentes procesadores. Este modelo hace que cada procesador pueda trabajar de manera independiente mientras no tiene que compartir información mediante complejos esquemas de mensajería ni sufre de la latencia que introducen las redes de comunicación. Este modelo permite hacer evolucionar códigos secuenciales a su versión paralela de manera más sencilla que en un paradigma de memoria distribuida. El problema de la memoria compartida es que, si bien, no hay que encargarse de pasar la información de un procesador a otro, hay que tener cuidado de que dos procesadores no intenten escribir a la vez en la misma zona de memoria. Además, en los esquemas de memoria compartida ésta no escala con el incremento de procesadores por lo que existen mayores restricciones en el tamaño de los programas y la cantidad de datos que pueden gestionar.

2.4.2.1 Historia

A principios de la década de 1990 surgió Message Passing Interface (MPI) con el objeto de definir un estándar para la programación paralela en el mundo de la supercomputación. El primer estándar MPI 1.0 data de 1994 y está enfocado a entornos de computación con memoria distribuida.

En 1997 surgió otro estándar de programación paralela denominado Open Multi-Processing (OpenMP) orientado a entornos de memoria compartida. Sus primeras versiones eran compatibles con el lenguaje de programación FORTRAN únicamente y hubo que esperar al año 2002 para que surgiese una nueva versión del estándar OpenMP compatible con C/C++, un

lenguaje ampliamente utilizados en el mundo de la computación de alto rendimiento. Desde entonces el estándar ha evolucionado siendo su versión 4.5 de muy reciente creación [23].

En los últimos quince años la capacidad de cálculo de las CPU y otros componentes se ha incrementado exponencialmente, surgiendo diversas tecnologías de programación paralela como son CUDA (Compute Unified Device Architecture, [24] [25]), OpenCL (Open Computing Language, [26] [27]) o Intel TBB (Intel Threading Building Blocks, [28]). La primera es una tecnología propietaria orientada a explotar la capacidad de cálculo de las tarjetas gráficas nVidia, la segunda busca la misma idea sin atarse a ningún fabricante hardware concreto y la tercera es una tecnología propietaria de Intel orientada a exprimir al máximo los recursos de la CPU con un modelo de orientación a objetos.

A continuación detallamos las características más relevantes de cada una de ellas.

2.4.2.2 Open Multi-Processing (OpenMP)

OpenMP es un API (Interfaz de Programación de Aplicaciones) para desarrollar programas paralelos siguiendo el paradigma de memoria compartida utilizando los lenguajes de programación C, C++ o Fortran. Este API utiliza el modelo "Fork and Join" (dividir y unir) en el cual los programas comienzan con un hilo de ejecución secuencial (hilo maestro) hasta que llegan a una sección paralela en la que se crean múltiples hilos que procesan el trabajo en paralelo para, al final de la sección paralela, sincronizarse y unirse al hilo principal que continúa la ejecución secuencial hasta la siguiente sección paralela o el final del programa tal y como se ve en la Figura 1.

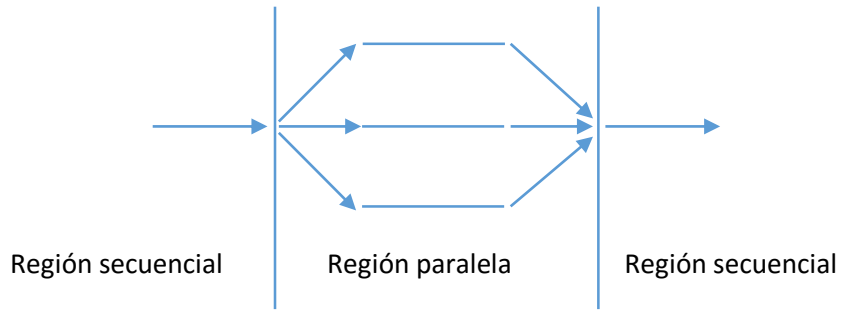

FIGURA 1

La gran ventaja que tiene OpenMP es que es muy fácil de introducir en algoritmos con una implementación secuencial ya existente. Además, es muy utilizado, portable y se adapta muy bien a las arquitecturas multi-núcleo de las CPU actuales o futuras ya que abstrae totalmente al programador de la arquitectura hardware del sistema. Adicionalmente, como es un estándar no depende de hardware o librerías propietarias para funcionar en múltiples plataformas y por lo tanto es muy fácil de mantener. Finalmente, es importante destacar que cualquier equipo con más de un núcleo en su CPU es susceptible de utilizar programas que usen OpenMP sin necesidad de cambiar su sistema operativo actual ni instalar complejos paquetes de librerías multiproceso.

La forma de especificar el paralelismo se realiza mediante directivas del compilador en el código fuente (como puede verse en la Figura 2) y habilitando la compatibilidad con las mismas en las opciones del compilador (como se ve en la Figura 3). Para tener acceso a las funciones que,

además de las directivas, ofrece OpenMP basta con incluir el fichero de cabeceros de funciones “omp.h”.

Entre otras, OpenMP ofrece las siguientes extensiones al lenguaje C o C++:

- Parallel for: ejecuta un bucle for de C, C++ o FORTRAN en paralelo utilizando tantos hilos como estén disponibles o los que haya especificado el usuario.
- Parallel: sección de código que se ejecuta en paralelo.
- Critical: sección que obliga a que los hilos que la ejecuten lo hagan de manera sincronizada, es decir, sólo un hilo ejecuta la sección mientras el resto espera para poder ejecutarla.
- Atomic: esta directiva permite que varios hilos actualicen el valor de una variable de manera sincronizada con un coste de sincronización mucho menor que la sección “Critical”.

Además de las directivas mencionadas son interesantes las siguientes funciones:

- omp_in_parallel: función que permite conocer si se está o no en una sección paralela, es decir, si una sección de código se está ejecutando por más de un hilo a la vez.
- omp_get_wtime: función que obtiene el tiempo que ha pasado en segundos desde un punto arbitrario pero consistente.

```
#pragma omp parallel for
for(UAVPO_omp_size_t i = 0; i < static_cast<UAVPO_omp_size_t>(totalRows); ++i)
```

FIGURA 2 ESPECIFICACIÓN DE UN BUCLE PARALELO UTILIZANDO OPENMP

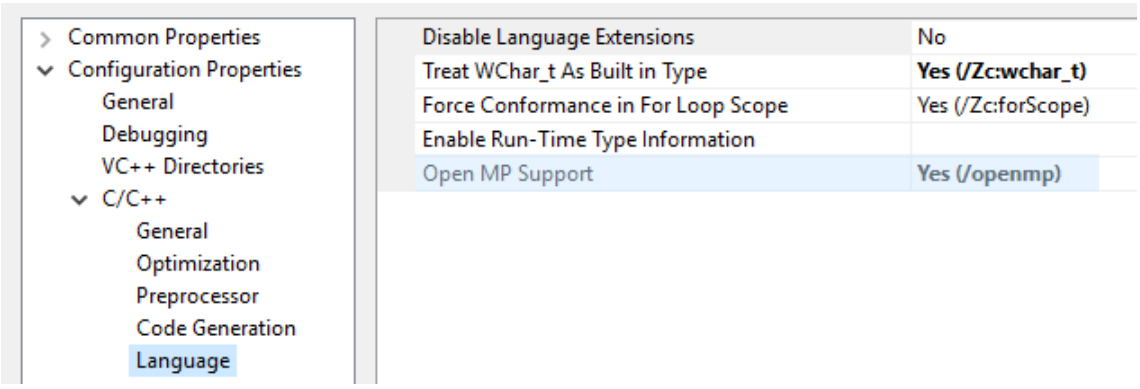

FIGURA 3 OPCIONES DE OPENMP EN VISUAL STUDIO 2010

2.4.2.3 AVX

Advanced Vector Extensions (AVX, [29]) es una extensión de las instrucciones de la arquitectura x86 que permite el manejo de hasta cuatro variables de precisión doble en una sola instrucción de manera que se podría dividir entre cuatro el número de iteraciones necesarias para procesar un vector. El uso de estas instrucciones requiere que tanto la CPU como el compilador utilizado las soporten. Su uso complica sensiblemente la legibilidad del código y su portabilidad, por lo

que suele ser mejor dejar que el compilador utilice y optimice dichas instrucciones a su libre albedrío durante el proceso de compilación.

2.4.2.4 Intel TBB

Intel Threading Building Blocks (Intel TBB, [28]) es una biblioteca basada en plantillas de C++ que facilita la escritura de programas que aprovechen las capacidades de paralelismo de las CPU actuales. La biblioteca ofrece algoritmos y estructuras de datos pensadas para su uso en situaciones multi-hilo; liberando al programador de algunas tareas de sincronización a la hora de acceder a la información; simplificando la creación, sincronización y destrucción de los hilos; y balanceando automáticamente la carga de trabajo entre estos. El defecto de esta librería es que, si bien es muy flexible y potente, hace que el programa dependa del compilador de Intel.

2.4.2.5 CUDA

Compute Unified Device Architecture (CUDA, [24] [25]) es un conjunto de herramientas de desarrollo entre las que se encuentra un compilador desarrollado por la compañía nVidia que permiten a los programadores acceder a los recursos de las GPU (Unidad de Procesamiento Grafico) que fabrica nVidia utilizando el lenguaje C.

El propósito de CUDA es aprovechar las ventajas que ofrecen las GPU frente a las CPU tradicionales, ya que las primeras tienen un altísimo número de núcleos en los que repartir la carga computacional, frente a uno o unos pocos núcleos que tienen las CPU. En teoría si una aplicación está diseñada para funcionar con un número elevado de hilos se beneficiaría enormemente de la arquitectura multi-núcleo de las GPU, además del gran ancho de banda que tiene la memoria de las GPU.

Los defectos de esta herramienta son que se limitan a explotar las capacidades del hardware nVidia, que introducen un trabajo extra a la hora de pasar información de la memoria del procesador a la de la tarjeta gráfica y vice versa por lo que son útiles en cargas de trabajo en las que se pueda dividir el trabajo en un gran número de hilos, y que tienen ciertas limitaciones que hace que no puedan aprovechar todas las capacidades del lenguaje C y sus tipos de datos.

2.4.2.6 OpenCL

Open Computing Language (OpenCL, [26] [27]), un interfaz de programación de aplicaciones y un lenguaje de programación que surgió como un estándar abierto para competir con CUDA, permite crear aplicaciones con paralelismo a nivel de datos y de tareas que pueden ejecutarse tanto en CPU tradicionales como en GPU. Es decir, es un conjunto de herramientas de programación que busca unificar el acceso a los recursos de computación disponibles con un único modelo de programación. El lenguaje está basado en C, eliminando parte de la funcionalidad, pero extendiéndolo con operaciones vectoriales.

3 Aplicación del paralelismo y otras técnicas de mejora algorítmica al planificador de trayectorias

En este capítulo se analizan los puntos computacionalmente problemáticos de las funciones de evaluación de objetivos y del algoritmo de evolución diferencial, y se describen como han sido implementados de una forma computacionalmente eficiente utilizando herramientas de paralelismo y otras técnicas de mejora algorítmica.

3.1 Puntos computacionalmente problemáticos de la función objetivo

La función objetivo es en sí pesada, primero porque hay que realizar muchos cálculos en ella y en segundo lugar porque hay que realizarlos por cada individuo (trayectoria) de cada UAV y durante todas las iteraciones del algoritmo de evolución diferencial.

Hasta ahora se han descrito las diferentes componentes de la función objetivo sin analizar las posibilidades de mejora que existen a la hora de implementar un algoritmo eficiente que calcule los diferentes objetivos.

Cada modelo matemático descrito en el apartado 2.2.2 representa la consecución de un objetivo, agrupable en uno de los dos subconjuntos siguientes: los objetivos que cada avión ha de cumplir de manera independiente y el objetivo de colisión que ha de calcularse a partir de los resultados obtenidos en el primer conjunto. A la hora de implementar la evaluación de todos ellos, es relevante al conjunto al que pertenecen, ya que aquellos que son propios de cada UAV e independientes de otros se pueden evaluar simultáneamente, mientras que el objetivo que depende de los resultados obtenidos con el primer conjunto tiene que esperar a que la evaluación de los objetivos restantes haya finalizado. Esta diferencia hace que en los siguientes apartados analicemos y propongamos por separado las mejoras que se pueden realizar en cada grupo.

3.1.1 Primer conjunto: objetivos independientes de otros UAVs

Los objetivos asociados a cada UAV que son independientes del resto de UAVs se enumeran a continuación y se evalúan en una función de objetivos independientes:

- Radio de giro mayor o igual que el mínimo ángulo de giro, descrito en el apartado 2.2.2.1
- Ángulo de ataque y ángulo de planeo dentro del rango de máximo ángulo de ataque y mínimo ángulo de planeo, descrito en el apartado 2.2.2.2
- Consumo de combustible, descrito en el apartado 2.2.2.3
- Evitar el choque con el suelo, descrito en el apartado 2.2.2.4
- Evitar rebasar los límites del mapa, descrito en el apartado 2.2.2.5
- Evitar sobrevolar las zonas de exclusión aérea, descrito en el apartado 2.2.2.6
- Camino más corto, descrito en el apartado 2.2.2.7
- Mínima probabilidad de destrucción, descrito en el apartado 2.2.2.8
- Probabilidad de detección, descrito en el apartado 2.2.2.9
- Altitud mínima, descrito en el apartado 2.2.2.10

Analizando los cálculos que hay que realizar para obtener la información de la función objetivo independiente se observa que es crítico evitar duplicar información y, por lo tanto, siempre que

sea posible se evita copiar argumentos, siendo en su mayoría referencias a información común. Además se observa que el cálculo del objetivo de altitud media de vuelo mínima se puede realizar en la misma iteración que el objetivo de evitar el choque contra el suelo, de esta manera evitamos tener que calcular las diferencias de alturas dos veces.

Para calcular los objetivos de radio de giro y ángulo de ataque, camino más corto, mínima probabilidad de destrucción, mínima probabilidad de detección y consumo de combustible mínimo hace falta obtener la evolución del comportamiento del avión a partir de los puntos de control del spline que representa su trayectoria. Los parámetros comunes de estas funciones son las velocidades en cada eje para cada punto de la trayectoria, los ángulos de giro, los ángulos de ataque y las distancias entre cada punto de la trayectoria. Para mejorar el rendimiento, dicha información se calcula una vez por trayectoria de manera que se reutiliza en el análisis de cada objetivo, por ejemplo, los ángulos de ataque y planeo entre cada punto de la trayectoria se necesitan para saber si se superan los límites permitidos y durante el cálculo de la sección radar del avión no tripulado. Al calcularlos sólo una vez reducimos sensiblemente el coste computacional de la función objetivo. Siguiendo esta filosofía se han implementado todos los cálculos de los objetivos tratando de reutilizar cálculos para minimizar el tiempo de ejecución.

No sólo se deben reusar cálculos sino también evitar calcular aquello que no sea necesario, especialmente si se sabe que es costoso computacionalmente. Por ejemplo, el cálculo de la sección equivalente del radar es computacionalmente costoso y por ello se calcula exclusivamente en los puntos de la trayectoria que están dentro del área de detección del radar. Pasa lo mismo con el cálculo de la probabilidad de destrucción por un sistema antiaéreo, que sólo se realiza si se está dentro del área con capacidad de matar de la unidad antiaérea.

Otro cuello de botella se encuentra en el cálculo de los objetivos probabilísticos (de detección y destrucción) en los que hay que calcular una función que determina si existe una la línea de visión entre dos puntos situados sobre la región definida por un determinado mapa. Esta función contiene uno de los cálculos computacionalmente más elevados entre las funciones objetivo independientes, motivo por el que el detalle del problema asociado a su cómputo y la solución eficiente utilizada se presenta en una sección independiente.

Por último, es importante destacar que los objetivos independientes de otros UAVs se pueden evaluar de forma simultánea, no solo para cada uno de los UAVs, sino también para todos los individuos de la población asociada a un determinado UAV. Es decir, el cómputo de los objetivos de este grupo se puede paralelizar sobre los UAVs o sobre los individuos de las poblaciones de trayectorias que se generan para cada avión no tripulado y, por lo tanto, se puede acelerar sensiblemente el cómputo de los mismos.

3.1.1.1 Línea de visión: Algoritmo de Bresenham

El cálculo de la línea de visión se realiza trazando un segmento entre dos puntos dados y comprobando si el segmento intersecta con algún punto del terreno que hay entre los dos puntos de estudio. Si no hay intersección, los dos puntos se ven mutuamente y por lo tanto existe línea de visión. En el ejemplo de la Figura 4 existe línea de visión entre el punto negro y el verde, ya que no hay ningún obstáculo con el que el segmento entre los dos puntos intersecte, mientras que entre el punto negro y el naranja no existe línea de visión ya que el segmento intersecta con el terreno.

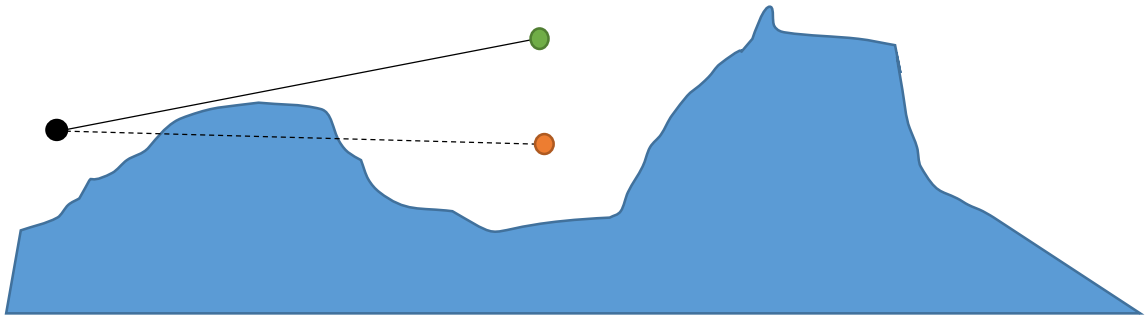

FIGURA 4 LÍNEA DE VISIÓN ENTRE DOS PUNTOS

La línea de visión se utiliza durante el cálculo del objetivo de probabilidad de destrucción y de detección de la unidad de defensa aérea ya que, si ésta no tiene línea de visión con el avión no tripulado, este último no puede ser detectado ni destruido. Su uso es muy costoso por dos motivos. El primero porque hay que ejecutarla sobre cada punto de cada trayectoria de cada UAV que entra en cada zona de influencia del ADU, y el segundo, porque puede ser implementada siguiendo estrategias de diferente complejidad computacional.

Por ejemplo, en el planificador disponible, desarrollado en Matlab, se usa una función propia de la Mapping Toolbox denominada *los2,* que cada vez que es llamada tiene que validar los parámetros del mapa de alturas y realizar una serie de interpolaciones para obtener un perfil de visibilidad que son bastante costosas.

Como alternativa, para nuestro planificador computacionalmente eficiente, hemos decidido utilizar el algoritmo de Bresenham [30], que determina qué elementos de una matriz n-dimensional, en nuestro caso bidimensional, han de seleccionarse para obtener una representación aproximada de una línea recta entre dos puntos dados. Es un algoritmo sencillo y ampliamente utilizado en computación ya que utiliza aritmética de enteros que es muy barata computacionalmente hablando. La Figura 5 muestra el pseudo-código del algoritmo de Bresenham utilizando aritmética de enteros y la Figura 6 muestra gráficamente el resultado de representar la recta $y = 0{,}5x + 1$ utilizando el mencionado código.

```
plotLine(x0,y0, x1,y1)
  dx=x1-x0
  dy=y1-y0

  D = 2*dy - dx
  plot(x0,y0)
  y=y0
  if D > 0
    y = y+1
    D = D - (2*dx)
  for x from x0+1 to x1
    plot(x,y)
    D = D + (2*dy)
    if D > 0
      y = y+1
      D = D - (2*dx)
```

FIGURA 5 ALGORITMO DE BRESENHAM UTILIZANDO ARITMÉTICA DE ENTEROS

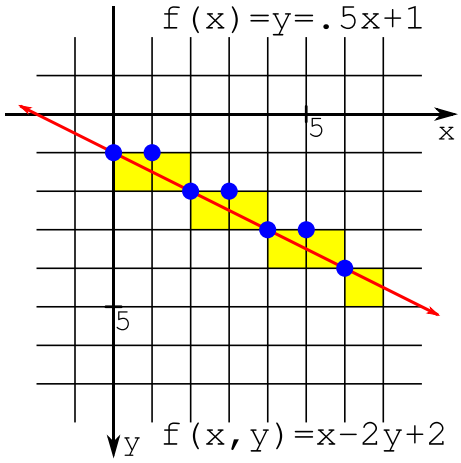

FIGURA 6 ALGORITMO DE BRESENHAM

En un artículo [31] sobre diversas aplicaciones del algoritmo de Bresenham, se describe en detalle la aplicación al cálculo de la línea de visión. Dado que en nuestro caso el terreno se representa como una matriz bidimensional cuyos valores representan las alturas de una serie de coordenadas cartesianas x e y, arbitrarias y ordenadas, podemos aplicar de manera directa el algoritmo de Bresenham para obtener de manera rápida una discretización del segmento que une los dos puntos entre los que se está calculando la línea de visión.

El algoritmo de Bresenham en sí no es suficiente para calcular la línea de visión, hace falta, además detectar si existe intersección del segmento obtenido con el terreno [32]. Para ello se calcula el ángulo de inclinación (tilt en inglés) entre el punto de origen del segmento y cada punto de la discretización de dicho segmento, que termina en un punto de la trayectoria sobre la que se está calculando la línea de visión. Si existe un ángulo de inclinación mayor que el ángulo de inclinación entre el origen y el destino significa que hay una intersección con el terreno y por lo tanto no hay línea de visión. Por el contrario, si el mayor ángulo de inclinación es el del punto de destino entonces no hay intersección con el terreno y existe línea de visión.

La expresión utilizada para el cálculo del ángulo de inclinación entre dos puntos arbitrarios del mapa es:

$$\alpha = \arctan\left(\frac{h_{pd} - h_c - h_{po}}{d}\right) \tag{54}$$

Donde

$$d = \sqrt{\left(x_{pd} - x_{po}\right)^2 + \left(y_{pd} - y_{po}\right)^2} \tag{55}$$

$$\begin{aligned} h_{pd} &= \text{altura del punto de destino} \\ h_{po} &= \text{altura del punto de origen} \\ h_c &= \text{factor de corrección debido a la curvatura de la Tierra} \end{aligned} \tag{56}$$

$$(h_c + r_{tierra})^2 = d^2 + r_{tierra}^2$$
$$h_c = \sqrt{d^2 + r_{tierra}^2} - r_{tierra} \quad (57)$$
$$r_{tierra} = \text{radio de la Tierra}$$

La discretización del segmento mencionado nos da K puntos sobre los que se aplica la expresión (54). La función $existLOS$ se calcula entonces según la expresión (58).

$$existLOS = \begin{cases} 1, \text{si } \forall k \in [1, K)\ \alpha_k < \alpha_K \\ 0, \text{en cualquier otro caso} \end{cases} \quad (58)$$

El problema es geométricamente sencillo, pero puede suponer mucho tiempo de computación ya que el tiempo de cálculo necesario crece de manera cuadrática $O(n^2)$ con el número de puntos a comprobar. La gran ventaja de este algoritmo es que el trazado de los segmentos y la comprobación de las intersecciones se pueden realizar de manera independiente por cada par de puntos y por lo tanto es fácil ejecutarlo en paralelo. En nuestro caso hemos preferido no paralelizar el algoritmo en sí, sino reducir las llamadas a la función de cálculo de visión al mínimo imprescindible, puesto que ya se paraleliza la función objetivo por cada trayectoria y porque sólo se realiza el cálculo de la línea de visión para aquellos puntos que están dentro del área en la que la unidad antiaérea puede derribar el avión no tripulado. Además, como ya se ha comentado, el algoritmo de Bresenham es muy rápido porque sólo utiliza sumas y restas de números enteros para obtener una aproximación del segmento, lo que supone una discretización muy veloz a la hora de obtener el perfil de alturas entre los puntos de estudio. El hecho de que el algoritmo de Bresenham sea especialmente rápido para calcular la línea de visión se apoya en que no se realizan interpolaciones complejas sobre el mapa si no que se itera sobre sus componentes, puesto que éstas ya representan una discretización del terreno en estudio.

Para mejorar aún más la velocidad del cálculo de la línea de visión se pre-procesa el mapa durante el arranque del algoritmo de forma que no haya que realizar transformaciones del mismo en cada cálculo de línea de visión. Además, el mapa es el mismo para todos los aviones no tripulados de manera que al compartir la información se reduce sensiblemente el consumo de memoria. Adicionalmente, para reducir el tiempo de carga se ha paralelizado el análisis de los datos del fichero de manera que se lee en bloques lo más grandes posibles y se procesa la información una vez está en memoria de manera paralela.

Finalmente, es importante destacar que según el artículo de Tuft [33], para reducir el tiempo de computación del algoritmo de línea de visión se puede aplicar una técnica denominada “culling” que consiste en determinar áreas en las que no existe línea de visión para reducir el número de segmentos a los que aplicar el cálculo de la línea de visión. La implementación que se ha realizado de la función objetivo sólo calcula la línea de visión en el caso de que el avión no tripulado se encuentre en el área en la que el radar sería capaz de destruirlo y si éste ha sido detectado, por lo que se están aplicando técnicas de “culling” implícitamente.

3.1.2 Segundo conjunto: objetivos dependientes

Ya se ha indicado que de los 11 objetivos a minimizar uno se ha de realizar de manera conjunta a partir de las trayectorias que minimizan los objetivos que son independientes para cada avión no tripulado. En el cálculo de colisiones se tienen en cuenta las trayectorias que son Pareto óptimas para los primeros 10 objetivos del resto de aviones no tripulados, ya que no tiene sentido buscar colisiones con todas las trayectorias que se generan para el resto de aviones no tripulados porque estos seguirán una de sus trayectorias Pareto óptimas. El cálculo de número de colisiones por trayectoria se puede realizar en paralelo, pero se ha preferido mantener de manera secuencial ya que la función objetivo conjunta se calculará en paralelo para cada avión no tripulado debida a que de esta forma se logra una mayor eficiencia al asignar recursos a cada avión no tripulado que asignándolos a cada individuo de la población de trayectorias.

3.2 Puntos computacionalmente problemáticos del algoritmo de optimización

El algoritmo de optimización tiene cuatro fases que podemos mejorar desde el punto de vista computacional: primero, la generación de la población inicial; segundo, la perturbación de la población; tercero, la combinación de las soluciones; y cuarto la selección de las mejores soluciones. Existe, además, una quinta fase que se encarga de mostrar y almacenar los datos que se obtienen durante las diversas iteraciones del algoritmo.

El algoritmo de optimización se puede aplicar a uno o más aviones no tripulados, por esto hay que tener en cuenta que la mejora de la eficiencia del algoritmo pasa por realizar un tratamiento diferente en los dos casos (un solo avión o múltiples aviones).

Por lo tanto, en los siguientes epígrafes analizamos tanto la problemática y soluciones generales de todo el algoritmo como aquellas asociadas a cada una de las cuatro fases, teniendo siempre en cuenta la diferencias que se dan en el caso de planificar la trayectoria de uno o varios aviones no tripulados.

3.2.1 Problemas generales del algoritmo

Cuando se comenzó el desarrollo de un planificador computacionalmente eficiente se hizo evidente la necesidad de utilizar operaciones matriciales que fueran eficientes y flexibles sobre las que construir los operadores del algoritmo evolutivo. Como el lenguaje utilizado para el desarrollo, que es C++, no ofrece en su implementación estándar ninguna clase matriz, se analizaron diversas librerías de matrices ya existentes, como son Eigen [34] o la librería ViennaCL [35]. Aunque ambas ofrecen flexibilidad y alto rendimiento, su uso supondría arrastrar dependencias de mucha funcionalidad innecesaria para el problema que estamos abordando. Por eso se ha desarrollado una clase denominada “Matrix” que es la base de las estructuras de datos utilizadas en todo el algoritmo. Se ha desarrollado como un patrón de C++ (“template” en inglés), lo que ha ahorrado tiempo de desarrollo y facilitado su reutilización. Además, la clase tiene un algoritmo de reserva dinámica de memoria que busca ser lo más amigable posible con las cachés de las CPUs, incrementando la memoria en un factor de 1.5, para favorecer la reutilización de memoria previamente utilizada por la matriz cuando no se conoce a priori el tamaño que va a tener la misma y por ende mejorar sensiblemente el rendimiento en el manejo de la estructura de datos [36].

Por otra parte, la lectura de los ficheros de configuración y mapa de alturas es un procedimiento lento, pero presenta la ventaja de que sólo hay que realizarlo una vez. Para obtener el mejor rendimiento posible se ha utilizado la librería RapidXML [37] para el manejo de ficheros en formato extended markup language (XML) porque la implementación de un procesador de ficheros XML estaba fuera del ámbito de desarrollo de este proyecto. A este respecto, se ha procurado utilizar funcionalidad que primaba el alto rendimiento sobre la flexibilidad a la hora de manejar el formato XML de los ficheros de configuración del algoritmo de optimización. Como ya se ha mencionado anteriormente, la carga del mapa, debido a su gran tamaño, supone una parte elevada del proceso de inicialización del planificador. Por lo tanto, esta operación se realiza leyendo en bloques de bytes la información del fichero hasta que está totalmente cargado en memoria y una vez está en memoria se analiza la información para extraer los datos de alturas de manera paralela.

Finalmente, para poder realizar el análisis a posteriori de los tiempos de ejecución se ha desarrollado un sistema de análisis temporal y de rendimiento (también conocido como "profiling" en inglés) integrado en los algoritmos y que sirve para conocer la carga de trabajo que supone cada parte del algoritmo y ayudar a detectar puntos problemáticos. El sistema desarrollado, utilizado la función omp_gt_wtime OpenMP, mide la carga de trabajo en cantidad de llamadas, tiempo de ejecución y porcentaje de tiempo del total de la optimización. Resuelve la problemática de medición de tiempo asociado a los algoritmos paralizados, gestionando una pila de ejecución independiente por cada hilo de ejecución, de forma que es posible calcular correctamente los tiempos de ejecución de cada función, incluyendo y excluyendo el tiempo asociado a las llamadas de otras funciones. De esta forma se pueden determinar con bastante precisión y de forma experimental cuales son las funciones más costosas del planificador. El sistema de análisis temporal, cuando está activado, mide los tiempos y muestra un informe con la siguiente información:

Total duration in seconds	Tiempo total que ha durado la optimización en segundos				
Function	***Call count***	***Inclusive (s)***	***Inclusive (%)***	***Exclusive(s)***	***Exclusive (%)***
Nombre de la función o método de la clase	Número de veces que se ejecuta la función	Tiempo total de ejecución en segundos de la función incluyendo todos los hilos y las llamadas a las funciones que usa esta función	El dato anterior, pero como porcentaje del total	Tiempo real de ejecución del cuerpo de la función teniendo en cuenta que el tiempo de ejecución en diferentes hilos se solapa y eliminando el tiempo de ejecución de las funciones a las que se llama desde esta función	El dato anterior, pero como porcentaje del tiempo total de ejecución.

TABLA 1 FORMATO DE LA INFORMACIÓN GENERADA POR EL ANALIZADOR DE TIEMPOS

3.2.2 Problemas durante la generación de la población inicial (fase I)

La función que se encarga de la generación de la población inicial ha de ser eficiente porque, aunque a priori uno pensaría que sólo se utiliza una vez al inicio del algoritmo, realmente se está utilizando en cada iteración para generar poblaciones de inmigrantes que sirven para perturbar la población. La generación de los inmigrantes es importante puesto que ayuda a renovar una población y a reducir los estancamientos en los mínimos locales.

La población inicial se genera de manera aleatoria aplicando una configuración especificada por el usuario mediante ficheros de configuración.

En el caso de optimizarse la trayectoria de un único vehículo aéreo, la generación aleatoria de los valores de un conjunto de individuos trata de paralelizar los cálculos en función del tamaño de la población, de manera que, si la población es menor que un determinado umbral, se genera de manera secuencial, puesto que es más eficiente, y a partir del mencionado umbral la población se genera de manera paralela. En esta fase se tiene en cuenta la codificación de las trayectorias y sus parámetros, habiéndose paralelizado las funciones que traducen entre coordenadas polares y cartesianas, además de las transformaciones entre las diferentes codificaciones posibles. En nuestro caso nos interesa principalmente la codificación cartesiana y la transformación de coordenadas polares a cartesianas ya que el procedimiento de configuración descrito en apartados anteriores se calcula en coordenadas polares, pero en el resto del algoritmo se utilizan las coordenadas cartesianas.

En el caso de que haya más de un avión no tripulado, el algoritmo se ha desarrollado para que todos los aviones calculen simultáneamente sus poblaciones iniciales, es decir, de manera paralela ya que es más eficiente que calcular en paralelo cada elemento de la población inicial y de manera secuencial cada avión no tripulado.

Utilizando estas dos estrategias dependientes del tipo de planificación, mono-avión o multi-avión, se logra que el algoritmo adapte los recursos de los que dispone a cada tipo de carga de trabajo y de manera dinámica sin que el usuario tenga que especificar nada.

Como ya se ha indicado en apartados anteriores es importante que la lógica desarrollada sea eficiente, y dicha eficiencia se logra entre otras cosas reduciendo el número de veces que se copia o duplica la información. Por lo tanto, la población generada se comparte por todas las funciones del algoritmo de manera que se logra un menor consumo de memoria y se evita que el planificador pierda el tiempo copiando información innecesariamente.

3.2.3 Problemas durante la perturbación y cruce de la población (fases II y III)

La perturbación o mutación es una operación pesada debido el gran número de individuos que debe modificar. Como las perturbaciones de los diferentes individuos son independientes entre sí y entre los diferentes aviones no tripulados, al igual que en el caso anterior, se puede realizar una paralelización en ambas vías, de manera que el algoritmo paraleliza el cálculo para cada avión no tripulado, si hay más de uno, y si sólo hay un avión, entonces paraleliza la perturbación en sí.

El caso del cruce podría considerarse similar. Sin embargo, como el cálculo que se realiza por cada individuo no es muy costoso, se tiene en cuenta el tamaño de la población a la hora de

activar el cálculo paralelo para el caso de un único UAV, ya que para poblaciones pequeñas el coste de sincronización de los múltiples hilos puede ser mayor que el tiempo necesario para el cálculo secuencial.

Finalmente, es importante destacar que en estas dos fases se puede ganar mucho tiempo evitando copias de información, al trabajar directamente sobre las matrices que contienen la población anterior y la actual.

3.2.4 Problemas durante la selección de los mejores individuos (fase V)

Esta fase es la más costosa computacionalmente de todo el algoritmo porque incluye los siguientes elementos:

- Obtención de los puntos que representan, de forma discreta, al spline codificado en los genes de los individuos de la población.
- Evaluación de la función objetivo, de acuerdo con las directrices recogidas en el apartado 3.1.
- Selección de los individuos a partir de la dominancia por parejas.

3.2.4.1 Discretización de los splines cúbicos codificados genéticamente

Los splines utilizados para representar las trayectorias son "natural cubic splines" [18] cuya peculiaridad es que la derivada en los puntos inicial y final es 0. Dado que un spline es una curva continua hace falta obtener una discretización de la misma para poder evaluar la función objetivo, que, en nuestro caso busca que todos los puntos de control del spline no superen los 10 km. El cálculo comienza creando un spline cuyos puntos de control son los definidos por el individuo de la población, después se interpola sobre el mismo hasta obtener L puntos cuya distancia entre si no supere una distancia dada. Para reducir el número de veces que hay que interpolar sobre el spline, se analiza la distancia entre los puntos de control que se usaron para generar el spline y se divide entre la distancia máxima permitida lo que nos da una estimación inicial del número de puntos necesarios L. Con la estimación inicial se interpola sobre el spline y se calcula la distancia máxima entre los puntos obtenidos. Dicha distancia se utiliza para obtener un factor con el que incrementar L para la siguiente interpolación hasta que obtengamos puntos cuya distancia entre sí no supere la distancia máxima. El factor de incremento ΔL se calcula:

$$d_j = \sqrt{\left(x_j - x_{j-1}\right)^2 + \left(y_j - y_{j-1}\right)^2 + \left(z_j - z_{j-1}\right)^2}, j \in (1, L] \tag{59}$$

$$\Delta L = \frac{\max\left(d_j\right)}{d_{\max}} \tag{60}$$

Al calcularse así el número de puntos que ha de tener la trayectoria se reduce el número de veces que hay que interpolar y además se ajusta el número de puntos interpolados a los estrictamente necesarios, ya que si utilizásemos un incremento arbitrario (como se hace en la implementación original en Matlab) estaríamos calculando la distancia entre puntos a ciegas y muy probablemente generando trayectorias más largas que ralentizarían innecesariamente la función objetivo. El tiempo de cálculo de la función objetivo depende del número de individuos

sobre los que se tenga que evaluar y de la longitud de las trayectorias de cada individuo, es decir, del número de puntos que éstas tengan.

Una vez se conocen todos los puntos de la trayectoria de cada individuo se calculan los tiempos de llegada a cada punto de la trayectoria suponiendo que la velocidad entre dos puntos de la discretización obtenida es constante:

$$t_{llegada,j} = \frac{d_j}{v} + t_0 \tag{61}$$

En nuestro caso t_0 es 0 pero podría parametrizarse para cada avión no tripulado.

Tras el cálculo de los tiempos se asignan los tipos de los puntos de control de la trayectoria, siendo "waypoints" normales o "waypoints de repostaje" los puntos que coinciden con los "waypoints" definidos por el problema y "waypoints optimizados" todos los demás.

La generación de la trayectoria de cada individuo de la población es independiente del resto por lo que es un cálculo candidato a ser realizado en paralelo para así reducir el coste computacional de la generación de los coeficientes de interpolación y las interpolaciones realizadas para obtener las trayectorias finales de cada individuo de la población.

3.2.4.2 Selección de los mejores individuos

Hay tres funciones implicadas en la selección de individuos, dos de las cuales se basan en la salida de una tercera.

La función en la que se basan las otras dos calcula la matriz de dominancia de los individuos en función de los resultados de la función objetivo de cada individuo y de una serie de prioridades. Esta función es bastante costosa porque se llama en muchas ocasiones y porque, además, cada cálculo se realiza sobre una gran cantidad de datos. En el cálculo de la matriz de dominancia primero hay que ordenar los datos en grupos de prioridades. Los algoritmos de ordenación son pesados y más cuando hay gran cantidad de datos a ordenar. Para evitar que la ordenación sea un proceso costoso se crea un diccionario de prioridades cuya clave es la prioridad y cuyo dato es la columna de la matriz de datos. Con dicho diccionario se referencian los datos de manera ordenada en función de las prioridades sin tener que ordenar toda la matriz de datos, lo que supone un ahorro importante de tiempo de cálculo. La segunda fase para el cálculo de la matriz de dominancia consiste en, para cada individuo y objetivo, calcular si cumple los límites inferior y superior y de no cumplirse la distancia que hay a los límites. Este proceso se realiza de manera paralela para reducir el tiempo que se tarda en obtener la información que se utilizará en la tercera y última fase del cálculo de la matriz de dominancia. La última fase consiste en comparar el cumplimiento de los objetivos de los individuos contra el resto de individuos y las diferencias entre los valores obtenidos por cada individuo en el caso de que ambos cumplan los objetivos para así determinar quién domina a quién. Evidentemente la comparación se puede hacer de manera independiente y por ende de manera paralela.

De las otras dos funciones, una calcula la bondad de cada individuo como una suma de la cantidad de individuos que lo dominan, para luego invertir el valor obtenido de manera que los individuos con mayor resultado obtenido son los mejores del frente de Pareto, y la otra realiza una selección de individuos calculando la matriz de dominancia comparando los individuos de

dos en dos. La función de selección de parejas utiliza la función del cálculo de dominancia, como esta última ya se ha paralelizado, se ha preferido no paralelizar la selección de parejas en sí.

La función que obtiene la bondad de cada individuo se paraleliza siempre y cuando haya un único avión no tripulado y está diseñada para que su cálculo involucre el menor número de iteraciones posibles sobre la matriz de dominancia previamente calculada.

3.2.5 Fase de salida de datos (fase V)

La quinta fase, que consiste en pintar y almacenar datos no pertenece propiamente al proceso de optimización, pero si no es eficiente puede penalizar en exceso el rendimiento del algoritmo retrasando la obtención de resultados.

Para obtener un equilibrio entre cantidad de información mostrada y velocidad de cómputo se ha desarrollado una función de salida que muestra la información en pantalla cada 2 iteraciones. Además, la función de salida almacena en el disco duro las componentes de la trayectoria y los valores de salida de la función objetivo para los individuos Pareto óptimos de cada iteración.

La fase de salida en pantalla no puede realizarse en paralelo porque se intercalaría la información de cada avión no tripulado y por eso se sincronizan los hilos de ejecución en el punto de salida a pantalla. Por otro lado, la salida a disco se realiza en un fichero de datos que es diferente para cada fichero por lo que se realiza totalmente en paralelo. Como sólo hay que sincronizar en el momento que se imprime en pantalla no se pierde mucho tiempo ya que es un proceso breve.

Durante la implementación del planificador se ha descubierto que el uso del stream de salida de C++, $std::cout$, era muy ineficiente en la implementación del compilador Microsoft Visual Studio 2010 y que, usando la función alternativa en C, $printf$, se reducían los tiempos de mostrado de mensajes hasta 20 veces. Por lo tanto, en nuestro planificador se utiliza la orden $printf$ para mostrar los datos por pantalla.

4 Resultados

En este apartado se describen los casos probados con el planificador, mostrándose las rutas óptimas obtenidas y comparando los tiempos de ejecución entre las versiones secuenciales y versiones paralelas del mismo.

Para poder mostrar de forma gráfica los resultados de las trayectorias resultantes, se ha recurrido a la herramienta gnuplot [38] [39], que permite la generación de gráficos de alta calidad a partir de una serie de ficheros de datos. En nuestro caso, el planificador se conecta dinámicamente a gnuplot a través de una tubería (conocidas normalmente por su nombre inglés "pipe") por la que se le envían los comandos necesarios para dibujar las trayectorias y generar así la imagen final del resultado final que se presenta en esta memoria.

Se han representado las trayectorias optimizadas en dos dimensiones ya que una representación estática (no re-orientable) en 3D de las mismas, impide en muchas ocasiones ver parte de la información relevante del problema. Además, durante la fase de dibujado se ha preparado un algoritmo de selección de colores que genera los mismos aleatoriamente, pero de manera que contrasten entre sí. El algoritmo se basa en las ideas que se presentan en [40]. Las ecuaciones siguientes nos dan el valor del color en formato RGB:

$$hue_0 = rand() \tag{62}$$

$$hue_n = (hue_{n-1} + \phi)\ mod\ 1 \tag{63}$$

$$rgb_n = hsvToRgb(hue_n, 0.99, 0.99) \tag{64}$$

Donde la función $hsvToRgb(hue, saturation, value)$ convierte de la representación de color HSV (siglas en inglés de Hue, Saturation and Value) a RGB (siglas en inglés de Red, Green and Blue).

Aunque en la representación gráfica no se muestra la orografía del mapa sobre el que se realiza cada misión, el planificador evolutivo utiliza el mapa para calcular aquellos objetivos (incluyendo los de línea de visión) que lo requieren.

4.1 Ejemplos con un único avión no tripulado

En este apartado se analizan los resultados que ofrece el planificador para misiones en las que se utiliza un solo avión no tripulado.

Para obtener las rutas óptimas de cada caso, el algoritmo evolutivo implementado realiza quinientas iteraciones sobre una población inicial de treinta individuos. En cada iteración se perturba la población de la iteración anterior con cinco inmigrantes y se utiliza la mutación aleatoria con ruido. De esta forma, se caracteriza el algoritmo de forma análoga a la utilizada en el artículo [16]

4.1.1 Caso 1

En este caso el UAV tiene que llegar a su destino atravesando la barrera de defensas antiaéreas representadas en la Figura 7, cuya leyenda tiene el siguiente significado:

- ADU detection range: área en la que la probabilidad de detección puede ser mayor que 0
- ADU kill range: área en la que la probabilidad de destrucción puede ser mayor que 0
- ADU position: la situación en el mapa de los diferentes ADUs
- UAV1 waypoints: puntos que debe visitar obligatoriamente el UAV durante el desarrollo de la misión.
- UAV1 shortest path: representa la trayectoria más corta obtenida trazando líneas rectas entre los waypoints que ha de visitar el UAV.
- UAV1 first wpt: representa el punto del que parte el UAV.
- UAV1 optimized path: representa la trayectoria obtenida por el planificador.

Los resultados obtenidos para los diferentes objetivos se pueden ver en la Tabla 2, en la que se muestra la siguiente información:

- Floor: el valor minimizado del objetivo "Evitar el choque contra el suelo". Su valor representa el número de veces que la trayectoria generada por el planificador choca contra el suelo.
- Slope: el valor minimizado del objetivo "Angulo máximo de ataque y ángulo mínimo de planeo". Su valor representa el número de veces que se incumple el objetivo.
- Map: el valor minimizado del objetivo "Permanecer en el área de vuelo". Su valor representa el número de veces que se ha salido del mapa.
- Radius: el valor minimizado del objetivo "Radio de giro mínimo". Su valor representa el número de veces que se hace un giró más brusco de lo que permiten las características del avión.
- NFZ: el valor minimizado del objetivo "Evitar las zonas de exclusión aérea". Su valor representa cuanto se ha entrado en las NFZs.
- Fuel: el valor minimizado del objetivo "Consumo de combustible". Un valor de 0 indica que no se ha consumido todo el combustible disponible.
- Altitude: el valor minimizado del objetivo "Altitud mínima de vuelo". Representa el valor medio de la altitud de la trayectoria en metros.
- Distance: el valor minimizado del objetivo "Camino más corto". Representa lo larga que es la trayectoria generada por el planificador con respecto a la trayectoria que une los puntos de paso obligatorios utilizando líneas rectas. Por lo tanto, este objetivo siempre tiene un valor mayor 1.
- Kill: el valor minimizado del objetivo "Probabilidad de destrucción". Probabilidad de destrucción en tanto por 1.
- Detec: el valor minimizado del objetivo "Probabilidad de detección". Probabilidad de detección en tanto por 1.

UAV	Floor	Slope	Map	Radius	NFZ	Fuel	Altitude (m)	Distance (ratio)	Kill	Detec
UAV1	0	0	0	0	0	0	3.634,37	1,10394	0	1

TABLA 2 VALORES DE LOS OBJETIVOS PARA LA TRAYECTORIA OBTENIDA EN EL CASO 1

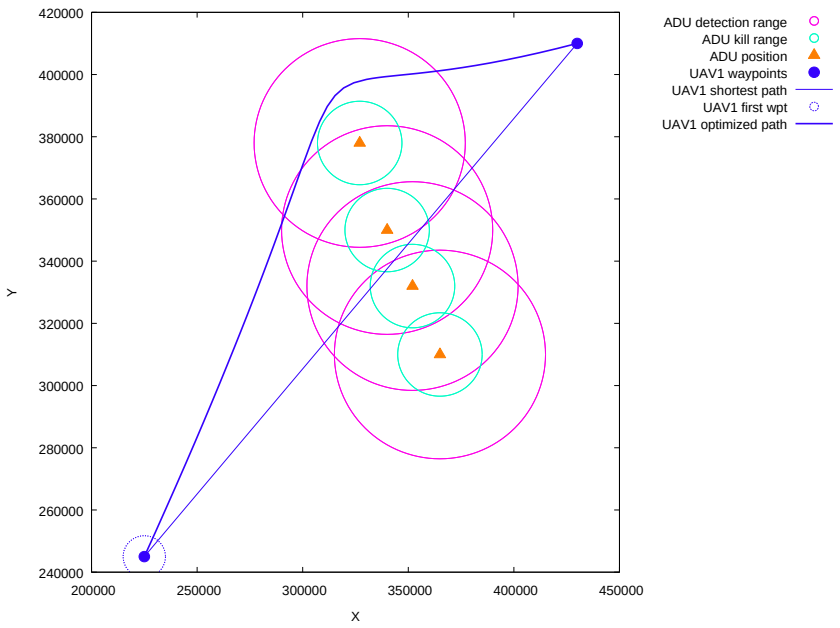

FIGURA 7 RESULTADO GRÁFICO DEL CASO 1

Como cabía esperar el planificador genera una ruta diferente de la más corta (línea recta), que sobrevuela los dos puntos de paso obligatorio, evita las zonas en las que los ADUs podrían destruir el UAV, y cumple todas las restricciones (las 6 primeros columnas de la Tabla 2). Además, en la Tabla 2 se puede observar, como en algunos casos posteriores, que el avión es detectado, ya que el objetivo de detección tiene una prioridad menor que la distancia recorrida por el avión.

4.1.2 Caso 2

En este ejemplo el UAV ha de visitar cuatro puntos sobrevolando una zona en la que hay tres ADUs y dos NFZs. Además de las leyendas introducidas en el ejemplo anterior, las zonas de exclusión aérea se representan con un rectángulo tal y como se puede observar en la Figura 8.

En este caso el planificador evita en la medida de lo posible las zonas de alto riesgo y se apoya en el algoritmo de línea de visión para ocultar el avión con la orografía del terreno, como demuestra el número de llamadas a la función existsLOS que se observa en la Tabla 4 (véase la Tabla 1 para conocer el significado de las columnas). En la Tabla 3 se puede observar que se cumplen todas las restricciones (6 primeras columnas).

UAV	Floor	Slope	Map	Radius	NFZ	Fuel	Altitude (m)	Distance (ratio)	Kill	Detec
UAV1	0	0	0	0	0	0	3.027,98	1,17171	0	1

TABLA 3 VALORES DE LOS OBJETIVOS PARA LA TRAYECTORIA OBTENIDA EN EL CASO 2

Function	Call count	Inclusive (s)	Inclusive (%)	Exclusive (s)	Exclusive (%)
existeLV	30.952	0,0302983	0,493426	0,00476598	0,466982

TABLA 4 NÚMERO DE LLAMADAS Y TIEMPOS DE EJECUCIÓN DE LA FUNCIÓN EXISTELOS EN EL CASO 2

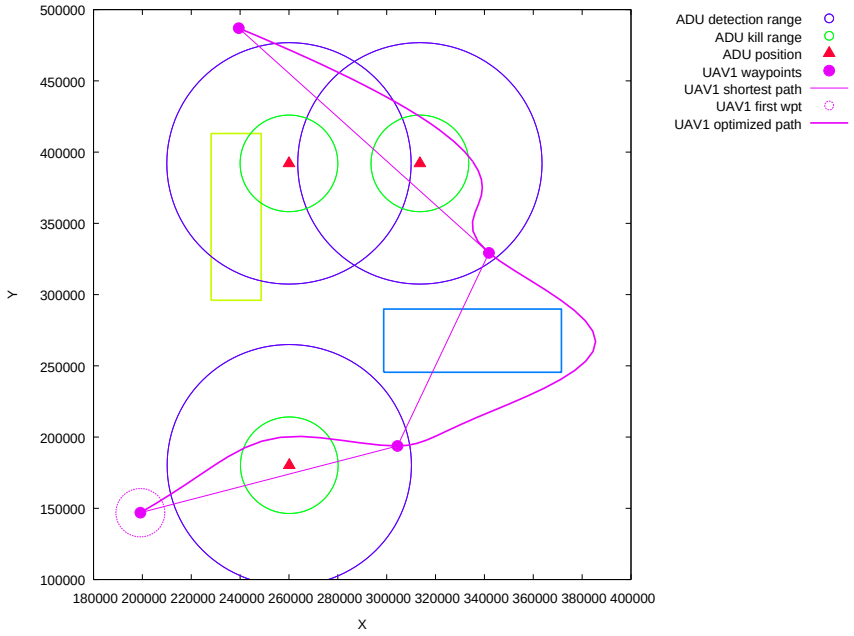

FIGURA 8 RESULTADO GRÁFICO DEL CASO 2

4.1.3 Caso 3

El UAV tiene que visitar cuatro puntos, de los cuales tres están relativamente bien alineados y el cuarto supone que el UAV tiene que realizar un giro brusco hacia atrás. En este caso, como se ve en la Figura 9, se demuestra que el planificador genera rutas en las que los giros están suavizados y por lo tanto se cumple el objetivo "Radio de giro mínimo". La bondad de los resultados se corrobora en la Tabla 5, en la que se observa que se cumplen todas las restricciones (primeros 6 columnas) y que las probabilidades de destrucción y detección son cero al no existir unidades de defensa aérea.

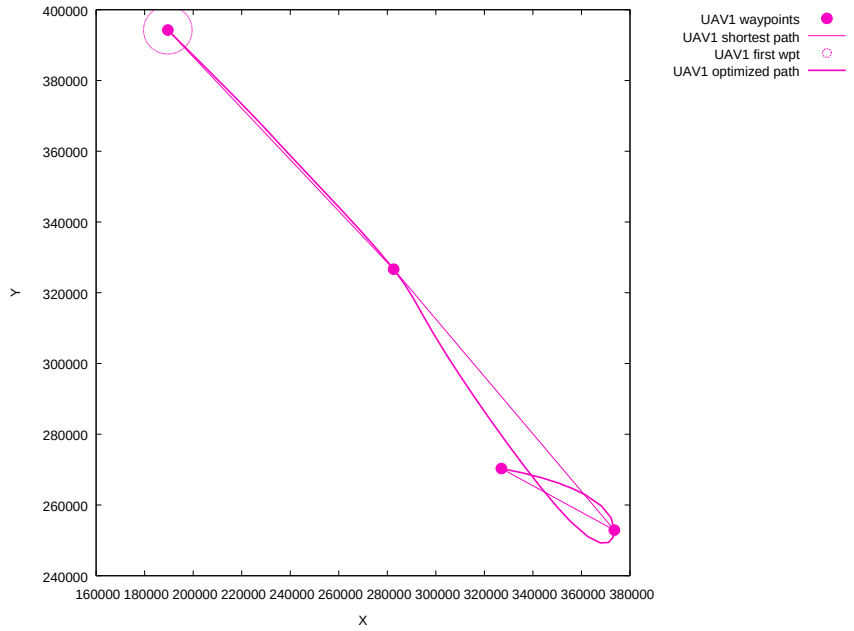

FIGURA 9 RESULTADO GRÁFICO DEL CASO 3

UAV	Floor	Slope	Map	Radius	NFZ	Fuel	Altitude (m)	Distance (ratio)	Kill	Detec
UAV1	0	0	0	0	0	0	1.441,68	1,03154	0	0

TABLA 5 VALORES DE LOS OBJETIVOS PARA LA TRAYECTORIA OBTENIDA EN EL CASO 3

4.1.4 Caso 4

En el cuarto caso mono-UAV, el UAV tiene que visitar nueve puntos en los que el origen y el destino están muy cerca (por lo que el UAV ha de hacer un circuito de ida y vuelta). La zona a sobrevolar contiene seis ADUs y tres NFZs. En la Figura 10 se observa que la trayectoria óptima obtenida evita las zonas en las que el avión puede ser destruido. Además, se observa que la trayectoria generada es bastante suave y no realiza giros bruscos. Además, en la Tabla 6 se puede observar cómo se cumplen todas las restricciones.

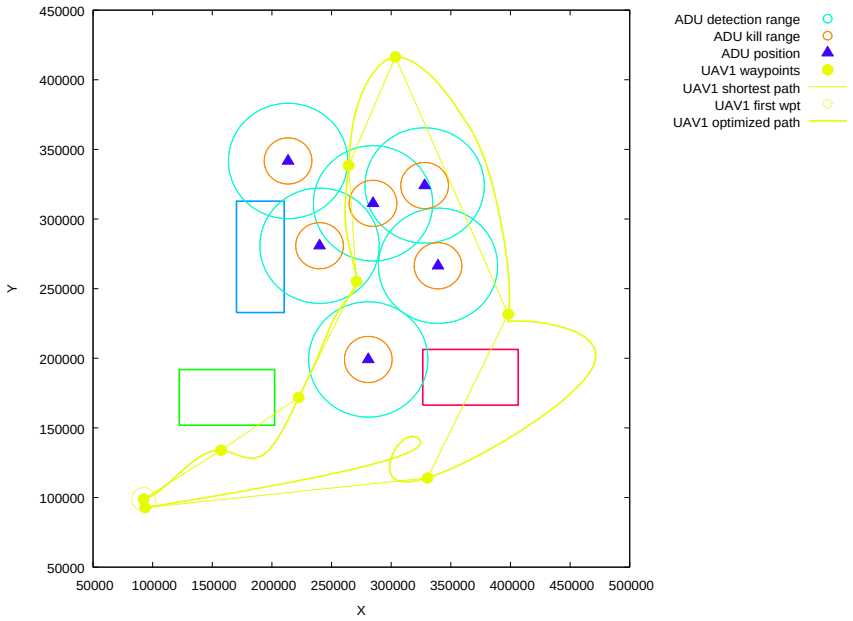

FIGURA 10 RESULTADO GRÁFICO DEL CASO 4

UAV	Floor	Slope	Map	Radius	NFZ	Fuel	Altitude (m)	Distance (ratio)	Kill	Detec
UAV1	0	0	0	0	0	0	1.441,68	1,03154	0	1

TABLA 6 VALORES DE LOS OBJETIVOS PARA LA TRAYECTORIA OBTENIDA EN EL CASO 4

4.1.5 Comparación de tiempos de ejecución

Para comprobar la eficiencia computacional del planificador desarrollado en este Trabajo Fin de Master, cada uno de los casos se ha ejecutado utilizando tres implementaciones diferentes del mismo:

1) La implementación secuencial desarrollada en Matlab por los directores del Proyecto y otros profesores de la UCM.
2) Una implementación secuencial en C++ desarrollada en este Trabajo Fin de Master como una aproximación intermedia entre la implementación secuencial de Matlab y la final paralela. En ella ya se incluyen todas las mejoras algorítmicas de reutilización de cálculos, así como gestión optimizada de la memoria que se han comentados en el capítulo 0.

3) La implementación paralela en C++ resultante de este Trabajo Fin de Master. En ella se incluyen todas las mejoras de la implementación secuencial en C++ y se aprovechan las capacidades de cálculo paralelo de OpenMP.

Las optimizaciones se han realizado en un ordenador con 6 GB de memoria RAM y un procesador Intel Core i7 3630QM que ofrece hasta 8 hilos de ejecución simultánea.

Los resultados obtenidos como promedio de 3 ejecuciones consecutivas en cada caso y sobre cada implementación se muestran en las tres primeras columnas con datos de la Tabla 7. Los valores recogidos en las 3 últimas columnas son los tiempos normalizados tomando como referencia, para cada caso, los tiempos de la implementación computacionalmente menos eficiente (Matlab).

Caso	Matlab (s)	C++ Secuencial (s)	C++ Paralelo (s)	Matlab (%)	C++ Secuencial (%)	C++ Paralelo (%)
1	184,43	1,74145	0,780967	100,00%	0,94%	0,42%
2	75,114	2,58708	1,01566	100,00%	3,44%	1,35%
3	70,225	1,10179	0,60452	100,00%	1,57%	0,86%
4	358,46	5,57417	1,70031	100,00%	1,56%	0,47%

TABLA 7 TIEMPOS DE EJECUCIÓN EN EJEMPLOS CON UN ÚNICO UAV

En la Figura 11 se representan de forma gráfica los tiempos normalizados, sobre una escala máxima en el eje de ordenadas del 10%, ya que sobre una escala del 100% las diferencias entre las versiones en Matlab y en C++ son tan elevadas, que no era posible ver la representación de los tiempos del planificador implementado en C++.

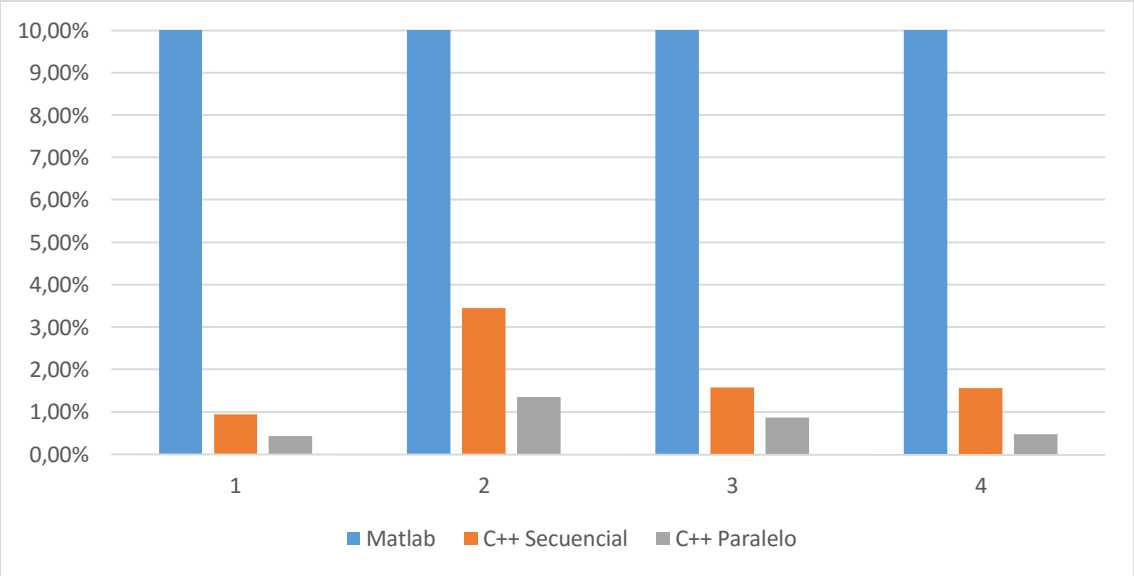

FIGURA 11 TIEMPOS DE EJECUCIÓN NORMALIZADOS EN EJEMPLOS CON UN ÚNICO UAV. EL 100% ES EL TIEMPO DE EJECUCIÓN DE LA VERSIÓN SECUENCIAL DESARROLLADA EN MATLAB

Los resultados muestran que el planificador que se ha desarrollado en C++, tanto en su versión secuencial como en su versión paralela, es significativamente más eficiente, desde el punto de vista computacional, que una implementación del planificador desarrollada previamente en Matlab.

Como las diferencias de tiempo son sustanciales, se han ejecutado los casos varias veces para ver si había algún fallo de configuración que forzase a la versión en Matlab a realizar más operaciones (por ejemplo, un dibujado de resultados de manera innecesaria). Tras comprobar que los resultados (trayectorias y valores de la función objetivo) obtenidos con las tres implementaciones son similares y que la versión de Matlab no está realizando operaciones innecesarias, se puede concluir que a pesar de que la implementación en C++ es por si más eficiente, y que su paralelización mejora aún más los tiempos de computo.

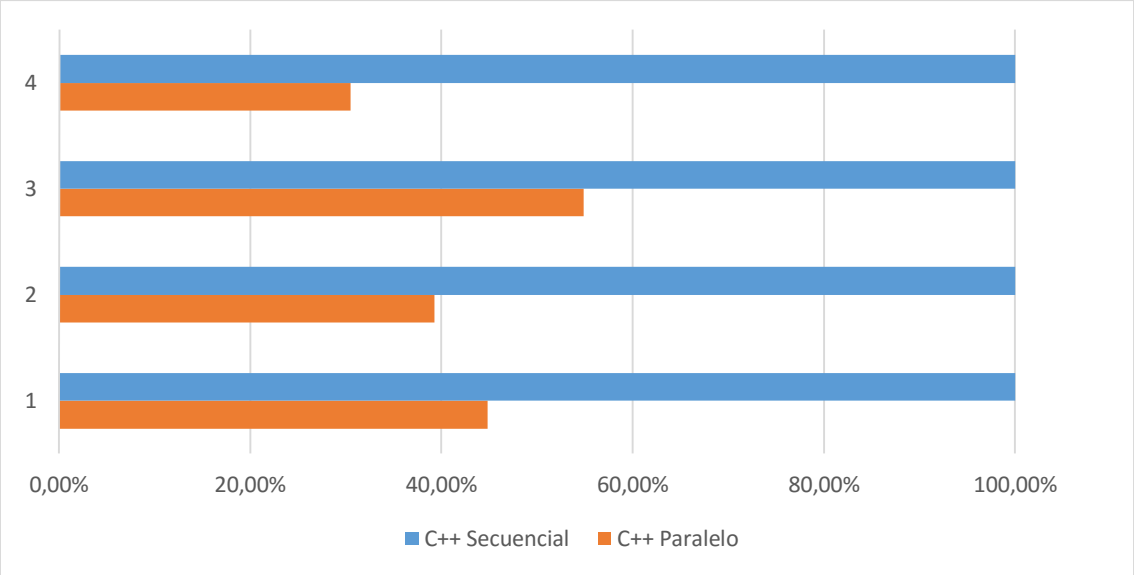

FIGURA 12 COMPARACIÓN DEL TIEMPO ENTRE LA VERSIÓN SECUENCIAL EN C++ Y LA VERSIÓN PARALELA EN C++. EL 100% ES EL TIEMPO DE EJECUCIÓN DE LA VERSIÓN SECUENCIAL

En la Figura 12 se observa que cuando el algoritmo tiene que paralelizar los cálculos para un único UAV obtenemos una reducción media del 58% en los tiempos de ejecución. Esto demuestra que el hecho de utilizar técnicas de paralelismo a la hora de tratar los individuos de las poblaciones de trayectorias para un avión dado mejora sensiblemente los tiempos de cálculo. Los tiempos que se muestran se han medido en problemas que generan poblaciones de 30 individuos, y, a raíz de los resultados obtenidos, se puede extrapolar que cuanto más grande sean las mencionadas poblaciones más evidente será la mejora.

4.2 Ejemplos con múltiples aviones no tripulados

En este apartado se analizan los casos en los que el planificador optimiza las trayectorias de un conjunto de aviones no tripulado.

Para obtener las rutas óptimas de cada caso, el algoritmo evolutivo implementado realiza quinientas iteraciones sobre una población inicial de treinta individuos. En cada iteración se perturba la población de la iteración anterior con cinco inmigrantes y se utiliza la mutación aleatoria con ruido. Por lo tanto, utilizamos en los ejemplos multi-UAVs los mismos parámetros de configuración del algoritmo evolutivo que en los casos mono-UAV.

4.2.1 Caso 1

En este caso tres UAVs sobrevuelan un área con tres ADUs y dos NFZs. Además, dos de los UAVs comparten uno de los puntos de obligada visita (waypoints), que no pueden ser visitados de forma simultanea por ambos vehículos.

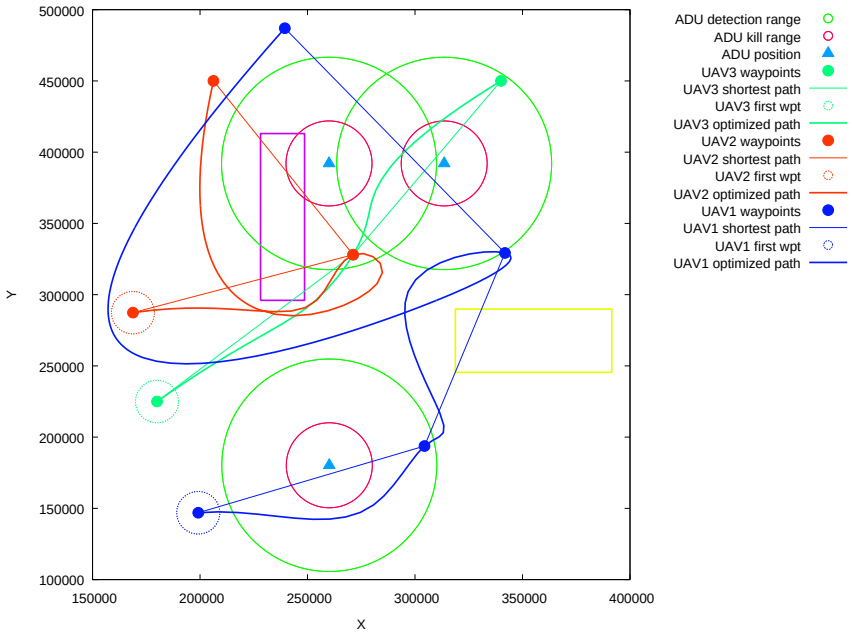

FIGURA 13 RESULTADOS GRÁFICOS DEL CASO 1

En la Figura 13 se muestra que el planificador obtiene rutas que visitan correctamente todos los puntos definidos como obligatorios para cada UAV con trayectorias de giros suaves y que además evitan las zonas de alto riesgo en la medida de lo posible, aprovechando la orografía si han de entrar en las zonas de alto riesgo de los radares.

La Tabla 8 muestra los valores de los objetivos para cada avión no tripulado que corroboran el cumplimiento de todas las restricciones (incluyéndose, en el caso multi-UAV el objetivo de colisión).

UAV	Floor	Slope	Map	Radius	NFZ	Fuel	Altitude (m)	Distance (ratio)	Kill	Detec	Collision
UAV1	0	0	0	0	0	0	1.666,90	1,7065	0	1	0
UAV2	0	0	0	0	0	0	1.370,77	1,55988	0	1	0
UAV3	0	0	0	0	0	0	1.594,85	1,01969	0	1	0

TABLA 8 VALORES DE LOS OBJETIVOS PARA LA TRAYECTORIA OBTENIDA EN EL CASO 1

4.2.2 Caso 2

En este caso dos UAVs vuelan con un waypoint común equidistante de sus puntos de salida y llegada. Además, se fuerza que los aviones vuelen a altitud y velocidad constante para comprobar que el planificador desvía una de las trayectorias de manera que no lleguen al waypoint común al mismo tiempo.

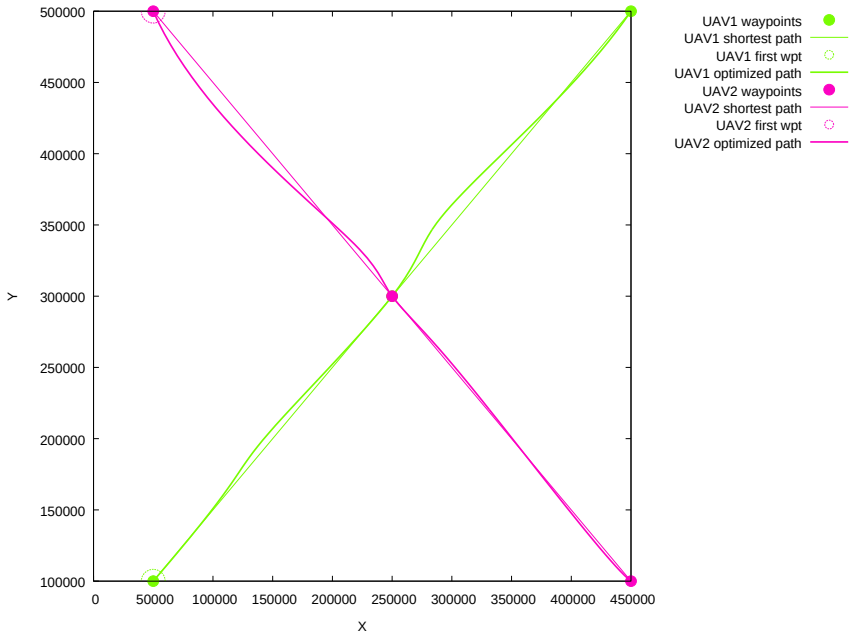

FIGURA 14 RESULTADOS GRÁFICOS DEL CASO 2

El planificador hace que las trayectorias sean lo más rectas posibles, pero para evitar la colisión en el punto central se desvían ligeramente, como era de esperar, lo que demuestra una vez más que el planificador funciona correctamente.

Como se observa en la Tabla 9, los aviones no chocan, las restricciones se cumplen y la distancia alcanzada únicamente es un poco mayor que la mínima (de ratio 1).

UAV	Floor	Slope	Map	Radius	NFZ	Fuel	Altitude (m)	Distance (ratio)	Kill	Detec	Collision
UAV1	0	0	0	0	0	0	2.700	1,00315	0	0	0
UAV2	0	0	0	0	0	0	2.700	1,00413	0	0	0

TABLA 9 VALORES DE LOS OBJETIVOS PARA LA TRAYECTORIA OBTENIDA EN EL CASO 2

4.2.3 Caso 3

En este caso hay dos UAVs cuyos destinos se cruzan y que se encuentran situados tras dos zonas de exclusión aérea.

En este caso se observa, sobre los resultados recogidos en la Tabla 10 y la Figura 15, que el planificador hace que las trayectorias eviten las zonas de exclusión aérea y el choque de los UAVs cambiando su altura de vuelo.

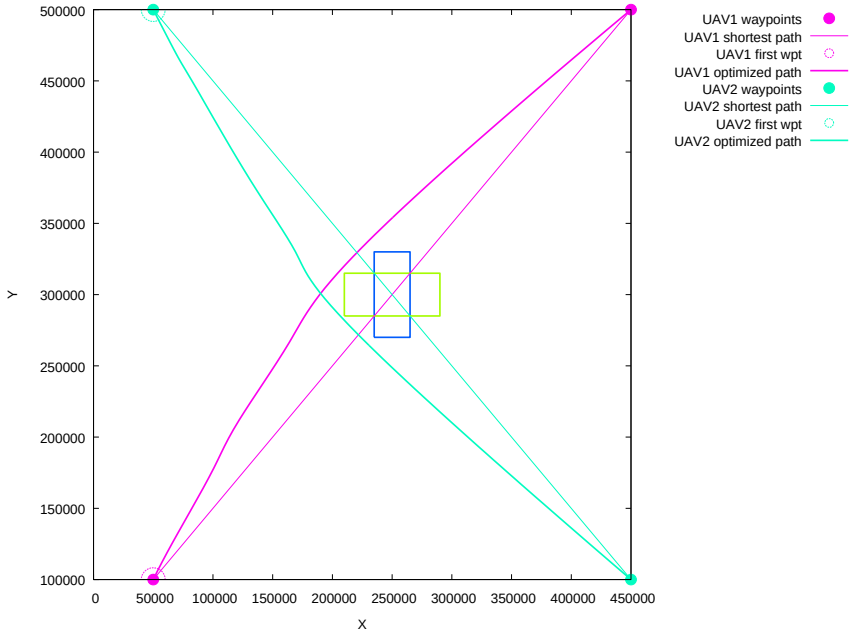

FIGURA 15 RESULTADOS GRÁFICOS DEL CASO 3

UAV	Floor	Slope	Map	Radius	NFZ	Fuel	Altitude (m)	Distance (ratio)	Kill	Detec	Collision
UAV1	0	0	0	0	0	0	1.845,6	1,0104	0	0	0
UAV2	0	0	0	0	0	0	2.077,5	1,009	0	0	0

TABLA 10 VALORES DE LOS OBJETIVOS PARA LA TRAYECTORIA OBTENIDA EN EL CASO 3

4.2.4 Caso 4

En este caso nos encontramos con cuatro UAVs volando en paralelo, 2 ADUs y 2 NFZs. Dos de los aviones han de evitar un ADU y los otros dos una NFZ.

Los resultados del planificador que se recogen en Figura 16 y la Tabla 11 muestran como las trayectorias generadas no chocan entre si y evitan las NFZ y la zona de alto riesgo del ADU.

UAV	Floor	Slope	Map	Radius	NFZ	Fuel	Altitude (m)	Distance (ratio)	Kill	Detec	Collision
UAV1	0	0	0	0	0	0	2.176,57	1,00549	0	1	0
UAV2	0	0	0	0	0	0	1.939,33	1,05702	0	0	0
UAV3	0	0	0	0	0	0	1.472,42	1,05886	0	0	0
UAV4	0	0	0	0	0	0	2.201,28	1,00531	0	1	0

TABLA 11 VALORES DE LOS OBJETIVOS PARA LA TRAYECTORIA OBTENIDA EN EL CASO 4

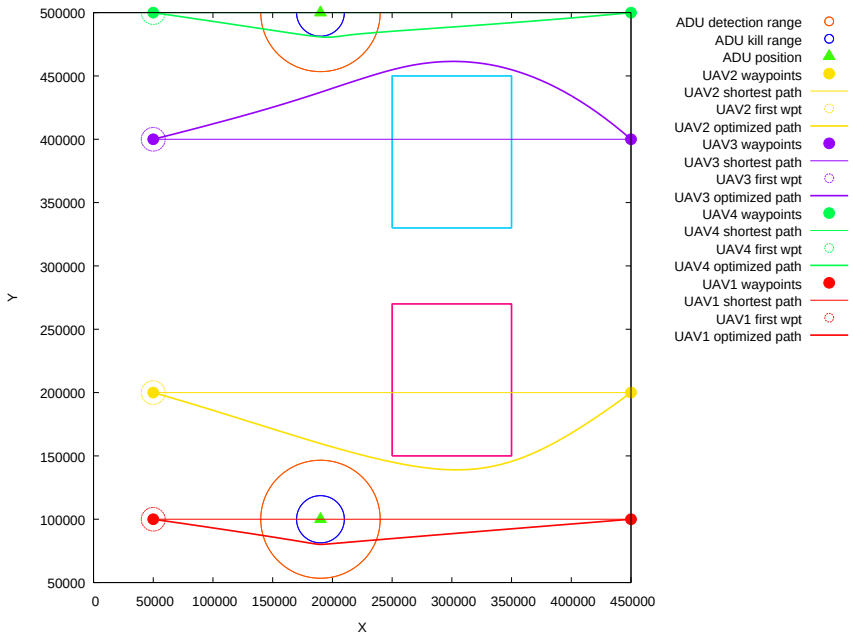

FIGURA 16 RESULTADO GRÁFICO DEL CASO 4

4.2.5 Caso 5

En este caso ocho UAVs vuelan con destinos opuestos dos a dos.

Los resultados del planificador, que se recogen en la Figura 17 y en la Tabla 12 Valores de los objetivos para la trayectoria , demuestran que aun en los casos en los que existen múltiples aviones sobrevolando la misma zona, el planificador es capaz de generar rutas en las que los aviones no chocan. En este caso la leyenda de la Figura 17 se ha representado en kilómetros.

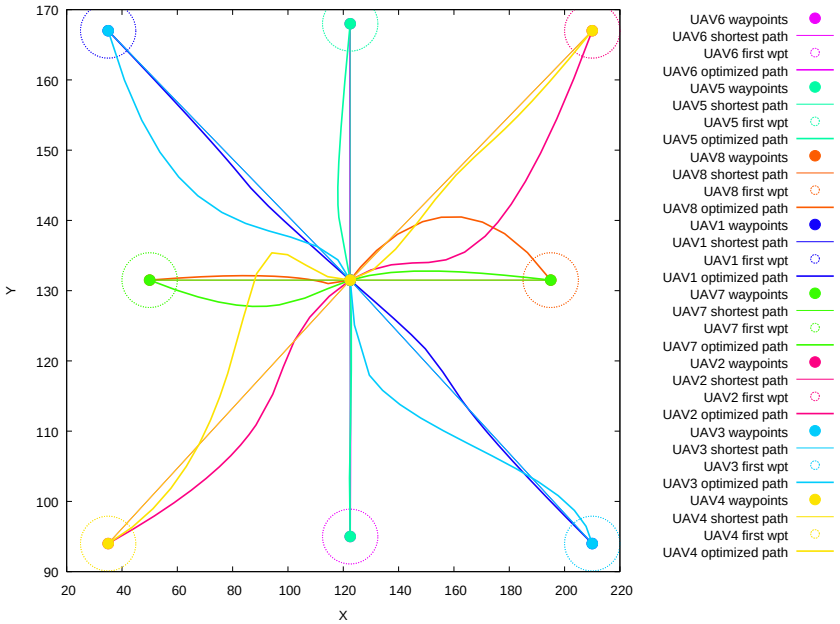

FIGURA 17 RESULTADO GRÁFICO DEL CASO 5

UAV	Floor	Slope	Map	Radius	NFZ	Fuel	Altitude (m)	Distance (ratio)	Kill	Detec	Collision
UAV1	0	0	0	0	0	0	2.231,68	1,00114	0	0	0
UAV2	0	0	0	0	0	0	2.177,39	1,0283	0	0	0
UAV3	0	0	0	0	0	0	2.175,04	1,04364	0	0	0
UAV4	0	0	0	0	0	0	2.220,10	1,04307	0	0	0
UAV5	0	0	0	0	0	0	1.583,54	1,0193	0	0	0
UAV6	0	0	0	0	0	0	1.756,82	1,00003	0	0	0
UAV7	0	0	0	0	0	0	2.160,45	1,00371	0	0	0
UAV8	0	0	0	0	0	0	2.005,13	1,02069	0	0	0

TABLA 12 VALORES DE LOS OBJETIVOS PARA LA TRAYECTORIA OBTENIDA EN EL CASO 5

4.2.6 Caso 6

En este caso dos UAVs sobrevuelan un mapa con dos ADUs. La peculiaridad de este caso es que uno de los UAV tiene su origen y destino dentro de las zonas de detección de los radares.

La Figura 18 y la Tabla 13 demuestran que como cabía esperar el planificador genera trayectorias que evitan el choque y las zonas con probabilidad de destrucción.

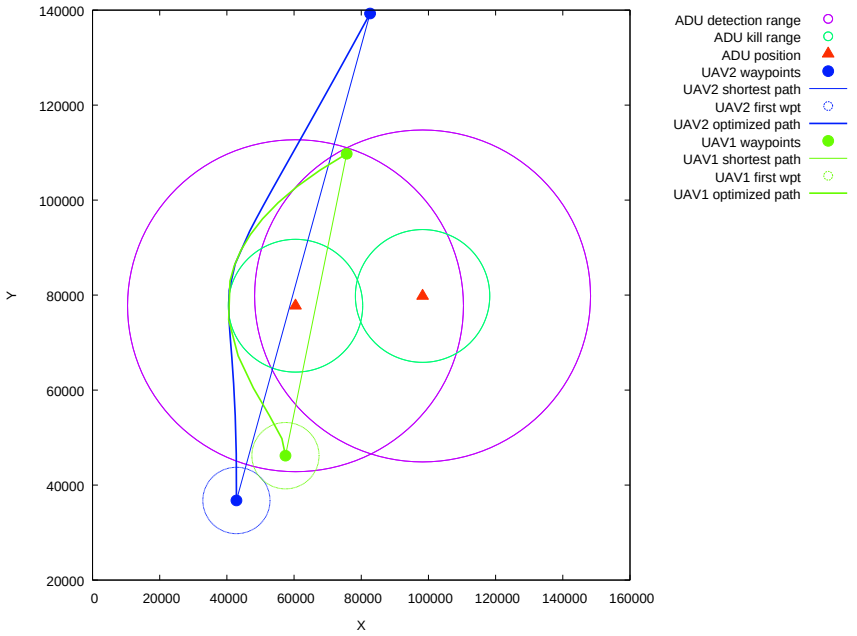

FIGURA 18 RESULTADO GRÁFICO DEL CASO 6

UAV	Floor	Slope	Map	Radius	NFZ	Fuel	Altitude (m)	Distance (ratio)	Kill	Detec	Collision
UAV1	0	0	0	0	0	0	1.524,22	1,31508	0	1	0
UAV2	0	0	0	0	0	0	897,435	1,0593	0	1	0

TABLA 13 VALORES DE LOS OBJETIVOS PARA LA TRAYECTORIA OBTENIDA EN EL CASO 6

4.2.7 Caso 7

En este caso tres UAVs sobrevuelan un mapa con seis ADUs y tres zonas de exclusión aérea.

En la Figura 19 se puede ver que el planificador evita las NFZ, los ADU y las colisiones a base de alargar las rutas. En la Tabla 14 se muestra el cumplimiento de las restricciones y la variabilidad en las distancias de las rutas.

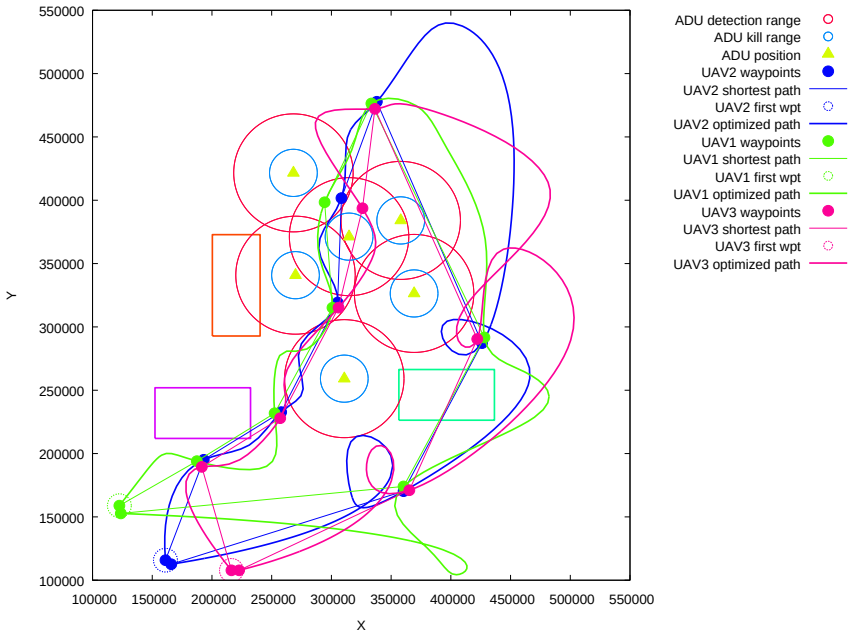

FIGURA 19 RESULTADO GRÁFICO DEL CASO 7

UAV	Floor	Slope	Map	Radius	NFZ	Fuel	Altitude (m)	Distance (ratio)	Kill	Detec	Collision
UAV1	0	0	0	0	0	0	3.741,04	1,34027	0	1	0
UAV2	0	0	0	0	0	0	2.883,30	1,51507	0	1	0
UAV3	0	0	0	0	0	0	2.951,02	1,61719	0	1	0

TABLA 14 VALORES DE LOS OBJETIVOS PARA LA TRAYECTORIA OBTENIDA EN EL CASO 7

4.2.8 Comparación de tiempos de ejecución

De nuevo, todos los casos se han ejecutado con tres implementaciones diferentes, la primera secuencial y desarrollada en Matlab, la segunda secuencial desarrollada en C++ con todas las mejoras del capítulo 0 salvo las relacionadas con la paralelización del algoritmo; y la tercera paralela desarrollada en C++, que constituye el algoritmo computacionalmente eficiente desarrollado en este Trabajo Fin de Master.

Los casos multi-UAV se han ejecutado sobre el mismo ordenador: un portatil con 6 GB de memoria RAM y un procesador Intel Core i7 3630QM que ofrece hasta 8 hilos de ejecución simultánea.

Los tiempos que se muestran a continuación, en las tres primeras columnas con tiempo de la Tabla 15, se obtienen promediando los valores obtenidos durante tres ejecuciones consecutivas de cada implementación sobre cada caso. Los valores normalizados de las tres últimas columnas, se obtienen respecto al caso peor recogido en cada caso.

Caso	Matlab	C++ Secuencial	C++ Paralelo	Matlab	C++ Secuencial	C++ Paralelo
1	382,7	13,0868	8,25873	100,00%	3,42%	2,16%
2	202,44	6,18242	4,68693	100,00%	3,05%	2,32%
3	192,57	8,83944	6,96107	100,00%	4,59%	3,61%
4	432,97	14,0428	6,26053	100,00%	3,24%	1,45%
5		14,4603	4,28681		100,00%	29,65%
6	355,09	2,61138	2,12349	100,00%	0,74%	0,60%
7	526,59	120,777	76,5948	100,00%	22,94%	14,55%

TABLA 15 TIEMPOS DE EJECUCIÓN Y SU NORMALIZACIÓN EN EJEMPLOS CON MULTIPLES UAV

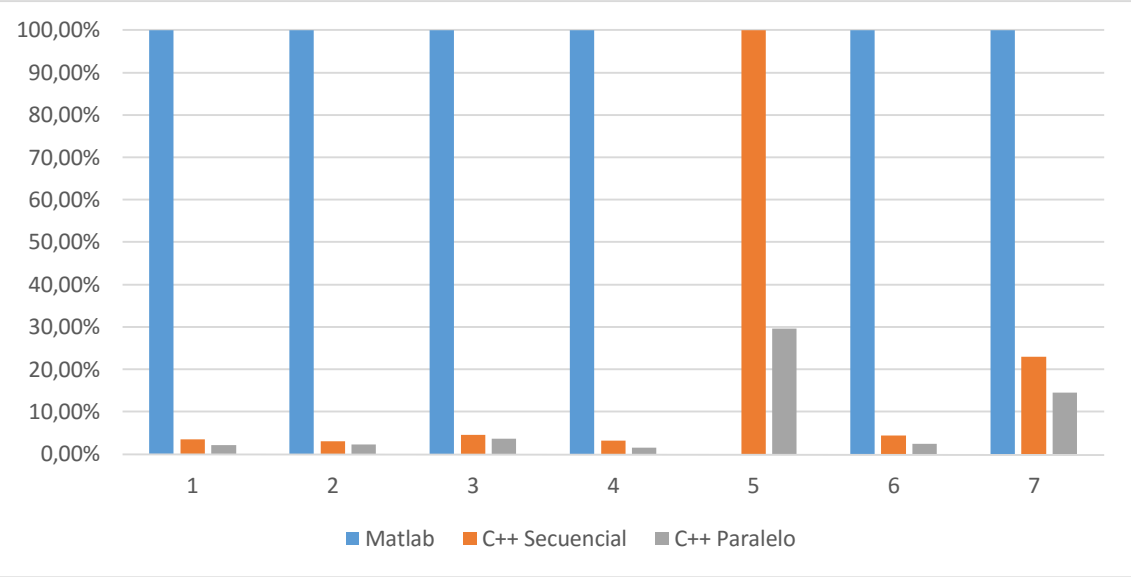

FIGURA 20 TIEMPOS DE EJECUCIÓN NORMALIZADOS EN EJEMPLOS CON MULTIPLES UAVS (BARRAS MÁS PEQUEÑAS SON MEJORES)

En los casos con múltiples aviones no tripulados se observa una mejoría sustancial con respecto a la implementación secuencial de Matlab. Al igual que en los casos con un solo avión no tripulado, se han comprobado varias veces las configuraciones de todas las implementaciones para descartar errores que justifiquen la elevada ganancia computacional de los planificadores implementado en C++.

El Caso 5 es un caso especial ya que la versión implementada en Matlab no era capaz de resolverlo. Las versiones en C++ sí lo logran, ya que contienen una variación en el algoritmo para el cálculo de las colisiones. El cambio consiste en que en vez de elegir la mejor ruta de cada frente de Pareto para los diferentes UAV y con dichas rutas calcular el objetivo de colisiones, las implementaciones en C++ tienen en cuenta todas las trayectorias Pareto óptimas y calculan el objetivo sumando la cantidad de colisiones en vez de si hay colisión o no.

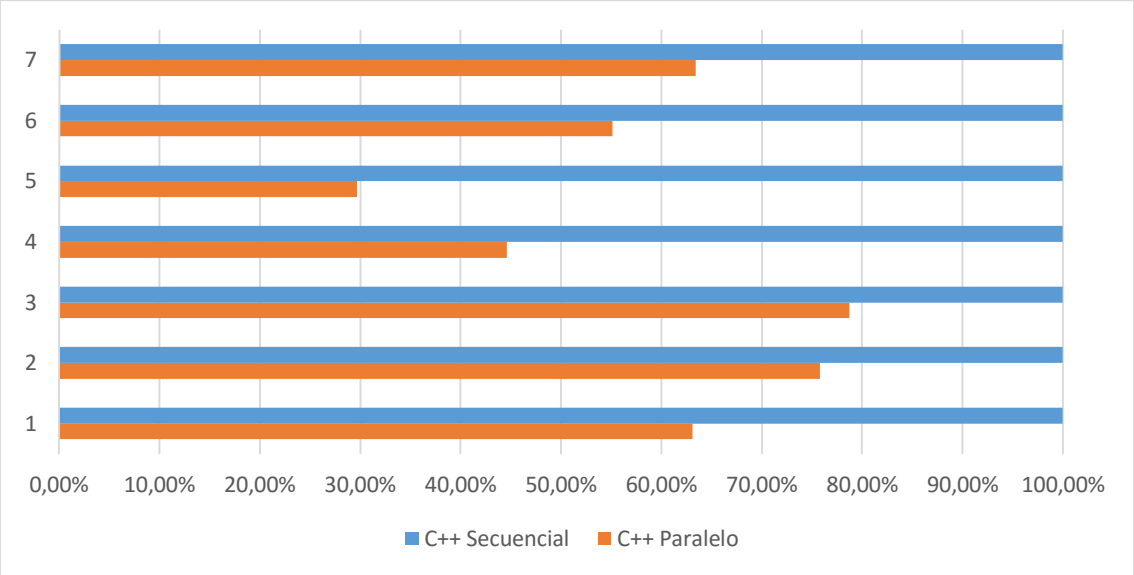

FIGURA 21 COMPARACIÓN DEL TIEMPO ENTRE LA VERSIÓN SECUENCIAL EN C++ Y LA VERSIÓN PARALELA EN C++. EL 100% ES EL TIEMPO DE EJECUCIÓN DE LA VERSIÓN SECUENCIAL

En la Figura 21 se observa que la versión paralela del algoritmo reduce los tiempos de ejecución de manera sensible, si bien, como el algoritmo paraleliza el cálculo para cada avión en vez de paralelizar el tratamiento de las poblaciones, como se hace cuando hay un solo UAV, la mejora se hace más evidente cuantos más aviones no tripulados hay en el escenario. El caso paradigmático es el caso 5 en el que hay 8 UAVs volando al mismo tiempo, obteniéndose una reducción del 70% en el tiempo de ejecución con respecto a la versión secuencial.

5 Conclusiones

Los objetivos planteados al comienzo de esta memoria eran:

1. Generar algoritmos evolutivos computacionalmente eficientes para la planificación de trayectorias en misiones con múltiples UAVs.
2. Generar algoritmos escalables.
3. Generar algoritmos que fueran fáciles de integrar en otros sistemas.

La implementación del planificador que se ha desarrollado cumple el primer y segundo objetivo como demuestran los resultados de tiempos de los apartados 4.1.5 y 4.2.8 en los que queda patente una mejora de la eficiencia algorítmica sensible con respecto a la implementación original en Matlab. La mejora en la eficiencia se ha logrado, por una parte, gracias a la aplicación de herramientas que optimizaban las partes secuenciales a base de no repetir tareas y aprovechando los cálculos ya realizados; además, allá donde era posible, se ha mejorado la eficiencia del uso de memoria evitando copias innecesarias de datos. Por otra parte, se han paralelizado aquellos elementos independientes del resto, hecho que redunda en la escalabilidad del algoritmo y en su capacidad para resolver casos más complejos.

Además, las nuevas implementaciones escalan correctamente en función de la carga de trabajo y recursos disponibles como se observa en la Figura 12 y en la Figura 21.

El desarrollo cumple además con el tercer objetivo definido ya que el nuevo planificador puede ser integrado fácilmente en otros sistemas porque utiliza exclusivamente tecnologías estandarizadas como son el lenguaje C++ y sus extensiones OpenMP, y no depender de librerías complejas ni hardware específico que introducen complejidad a la hora de utilizar el planificador en otros sistemas.

Entre las líneas de trabajo futuras, cabe destacar las siguientes:

- Durante la implementación se ha utilizado el compilador Microsoft Visual Studio 2010 que es compatible con la versión 2.0 de OpenMP. La mencionada versión de OpenMP no tiene toda la funcionalidad que ofrecen las versiones de reciente creación 4.0 ó 4.5, y su principal problema es que no permite el anidamiento de la paralelización lo que ha obligado a plantear dos formas de orientar el paralelismo en función del número de aviones no tripulados. Una posible vía de desarrollo sería adaptar los algoritmos desarrollados para permitir el mencionado anidamiento o la utilización de la tarjeta gráfica con tecnologías como CUDA para tener acceso a un número mayor de núcleos en los que repartir las cargas de trabajo.
- En el trabajo realizado se han obtenido resultados muy satisfactorios, pero no se ha abordado la parte “online” del problema en la que los aviones comienzan a volar y tienen que adaptar las rutas ante situaciones inesperadas o no conocidas durante la fase “offline”. Sería muy interesante continuar este trabajo extendiendo el desarrollo con la mencionada parte “online” del problema. Considerando la drástica reducción de tiempos en la ejecución del planificador, una replanificación local de las trayectorias, efectuada de manera online para evitar obstáculos y peligros imprevistos, sería más que viable.

6 Referencias

[1] C. Zheng, L. Li, F. Xu, F. Sun y M. Ding, «Evolutionary Route Planner for Unmanned Air Vehicles,» *IEEE Transactions on Robotics,* vol. 21, 2005.

[2] A. Sonmez, E. Kocyigit y E. Kugu, «Optimal path planning for UAVs using Genetic Algorithm,» *2015 International Conference on Unmanned Aircraft Systems (ICUAS),* pp. 50-55, 2015.

[3] Y. Qu, Q. Pan y J. Yan, «Flight path planning of UAV based on heuristically search and genetic algorithms,» *Proceedings of 31st Annual Conference of IEEE Industrial Electronics Society,* pp. 1-5, 2005.

[4] X. Liu, H. Wei, C. Zhou y S. Li, «The path planning of UAV based on orthogonal particle swarm optimization,» *Proceedings of SPIE,* vol. 8921, 2013.

[5] Z. Cheng, E. Wang, Y. Tang y Y. Wang, «Real-time Path Planning Strategy for UAV Based on Improved Particle Swarm Optimization,» *Journal of Computers,* vol. 9, nº 1, p. 209, 2014.

[6] Y. V. Pehlivanoglu, «A NEW PARTICLE SWARM OPTIMIZATION METHOD FOR THE PATH PLANNING OF UAV IN 3D ENVIRONMENT,» *JOURNAL OF AERONAUTICS AND SPACE TECHNOLOGIES,* vol. 5, nº 4, pp. 1-14, 2012.

[7] S. Li, X. Sun y Y. Xu, «Particle Swarm Optimization for Route Planning of Unmanned Aerial Vehicles,» *2006 IEEE International Conference on Information Acquisition,* pp. 1213-1218, 2006.

[8] A. N. Brintaki y I. K. Nikolos, «Coordinated UAV Path Planning Using Differential Evolution,» *Operational Research,* vol. 5, nº 3, pp. 487-502, 2005.

[9] H. D. Xiangyin Zhang, «An improved constrained differential evolution algorithm for unmanned aerial vehicle global route planning,» *Applied Soft Computing,* vol. 26, pp. 270-284, 2015.

[10] P. Yang, K. Tang, J. Lozano y X. Cao, «Path Planning for Single Unmanned Aerial Vehicle by Separately Evolving Waypoints,» *IEEE Transactions on Robotics,* vol. 31, nº 5, pp. 1130-1146, 2015.

[11] Y. Fu, M. Ding, C. Zhou y H. Hu, «Route Planning for Unmanned Aerial Vehicle (UAV) on the Sea Using Hybrid Differential Evolution and Quantum-Behaved Particle Swarm Optimization,» *IEEE Transactions on Systems, Man, and Cybernetics,* vol. 43, nº 6, pp. 1451-1465, 2013.

[12] D. Goldberg, Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, 1989.

[13] J. Kennedy y R. Eberhart, «Particle Swarm Optimization,» *IEEE,* pp. 1942-1948, 1995.

[14] «Differential Evolution (DE),» [En línea]. Available: http://www1.icsi.berkeley.edu/~storn/code.html#hist. [Último acceso: 2 Enero 2016].

[15] E. Besada-Portas, L. de la Torre, J. M. de la Cruz y B. de Andres-Toro, «Evolutionary Trajectory Planner for Multiple UAVs,» *IEEE Transactions on Robotics,* 2010.

[16] E. Besada-Portas, L. de la Torre, A. Moreno y J. L. Risco-Martin, «On the performance comparison of multi-objective evolutionary,» *Information Sciences,* vol. 238, pp. 111-125, 2013.

[17] A. Moreno, L. de la Torre, J. L. Risco Martin, E. Besada Portas, J. Aranda y J. L. Ayala Rodrigo, «DEVs based parallel framework for multi-objective evolutionary algorithms,» *Proc. of the fourth international workshop on parallel architectures and bioinspired algorithms,* vol. 10, pp. 39-48, 2011.

[18] C. de Boor, A Practical Guide to Splines, New York: Springer-Verlag, 1978.

[19] C. M. Fonseca y P. J. Fleming, «Multiobjective optimization and multiple constraint handling with evolutionary algorithms - Part I: Unified formulation,» *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans,* vol. 18, nº 1, pp. 26-37, 1988.

[20] Microsoft, «Getting started with Windows drivers,» Microsoft, [En línea]. Available: https://msdn.microsoft.com/en-us/library/windows/hardware/ff554690%28v=vs.100%29.aspx. [Último acceso: 10 Enero 2016].

[21] Modelica Association, «Functional Mock-up Interface,» [En línea]. Available: https://www.fmi-standard.org/. [Último acceso: 4 Enero 2016].

[22] A. Kiessling, «An Introduction to Parallel Programming with OpenMP,» Edinburgo, 2009.

[23] «OpenMP,» [En línea]. Available: http://openmp.org/wp/. [Último acceso: Noviembre-Diciembre 2015].

[24] W. contributors, «CUDA,» [En línea]. Available: https://en.wikipedia.org/w/index.php?title=Special:CiteThisPage&page=CUDA&id=697481561. [Último acceso: 6 Diciembre 2015].

[25] nVidia, «CUDA,» [En línea]. Available: http://www.nvidia.com/object/cuda_home_new.html. [Último acceso: 12 Diciembre 2015].

[26] W. contributors, «OpenCL,» [En línea]. Available: https://en.wikipedia.org/w/index.php?title=OpenCL&oldid=701900314. [Último acceso: 8 Diciembre 2015].

[27] «OpenCL,» [En línea]. Available: https://www.khronos.org/opencl/. [Último acceso: 8 Diciembre 2015].

[28] «Intel® Threading Building Blocks (Intel® TBB),» [En línea]. Available: https://software.intel.com/en-us/intel-tbb. [Último acceso: 8 Octubre 2015].

[29] C. Lomont, «Introduction to Intel® Advanced Vector Extensions,» [En línea]. Available: https://software.intel.com/en-us/articles/introduction-to-intel-advanced-vector-extensions. [Último acceso: 7 Octubre 2015].

[30] W. contributors, «Bresenham's line algorithm,» [En línea]. Available: https://en.wikipedia.org/w/index.php?title=Bresenham%27s_line_algorithm&oldid=699165594. [Último acceso: 24 12 2015].

[31] «Bresenham magic: raycasting, line of sight…,» [En línea]. Available: http://deepnight.net/bresenham-magic-raycasting-line-of-sight-pathfinding/. [Último acceso: 20 Diciembre 2015].

[32] A. Osterman, «Implementation of the r.cuda.los module in the open source GRASS GIS by using parallel computation on the NVIDIA CUDA graphic cards,» *ELEKTROTEHNISKI VESTNIK ENGLISH EDITION,* nº 2012, pp. 19-24, 2012.

[33] D. Tuft, B. Salomon, S. Hanlon y D. Manocha, «Fast Line-of-Sight Computations in Complex Environments,» [En línea]. Available: gamma.cs.unc.edu/LOS/RBVLos.pdf. [Último acceso: 3 Enero 2016].

[34] «EIGEN,» [En línea]. Available: http://eigen.tuxfamily.org/index.php?title=Main_Page. [Último acceso: 25 Septiembre 2015].

[35] «ViennaCL,» [En línea]. Available: http://viennacl.sourceforge.net/. [Último acceso: 30 Septiembre 2015].

[36] J. Delong y E. Baunton, «GitHub FBVector.md,» [En línea]. Available: https://github.com/facebook/folly/blob/master/folly/docs/FBVector.md. [Último acceso: 15 Octubre 2015].

[37] M. Kalicinski, «RAPIDXML,» [En línea]. Available: http://rapidxml.sourceforge.net/. [Último acceso: 3 Octubre 2015].

[38] «gnuplot homepage,» [En línea]. Available: http://www.gnuplot.info/. [Último acceso: 10 Enero 2016].

[39] «Gnuplotting,» [En línea]. Available: http://www.gnuplotting.org/. [Último acceso: 10 Enero 2016].

[40] M. Ankerl. [En línea]. Available: http://martin.ankerl.com/2009/12/09/how-to-create-random-colors-programmatically/. [Último acceso: 15 Enero 2016].

[41] I. K. Nikolos, E. S. Zografos y A. N. Brintaki, «UAV Path Planning Using Evolutionary Algorithms,» de *Innovations in Intelligent Machines - 1*, Springer Berlin Heidelberg, 2007, pp. 77-111.