Proyecto Fin de Grado
Ingeniería Electrónica, Robótica y Mecatrónica

Planificación eficiente de trayectorias libres de colisiones para un equipo de drones. Comparación de algoritmos.

Autor: Ana Victoria Canterla Martín

Tutor: José Miguel Díaz Báñez

Dpto. Matemática Aplicada II
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Proyecto Fin de Grado

Ingeniería Electrónica, Robótica y Mecatrónica

Planificación eficiente de trayectorias libres de colisiones para un equipo de drones. Comparación de algoritmos.

Autor:

Ana Victoria Canterla Martín

Tutor:

José Miguel Díaz Báñez

Catedrático de Universidad

Dpto. Matemática Aplicada II

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2022

Proyecto Fin de Grado: Planificación eficiente de trayectorias libres de colisiones para un equipo de drones. Comparación de algoritmos.

Autor: Ana Victoria Canterla Martín
Tutor: José Miguel Díaz Báñez

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

La realización de este trabajo no supone un trabajo más en sí, significa el fin de una etapa dura pero a la vez emocionante en la que una vez llegado aquí, puedo sentirme orgullosa de la elección que tomé en su día para hacer esta carrera y vivir todos los momentos que ello ha conllevado para hacerme la persona que soy a día de hoy.

En primer lugar, me gustaría agradecer a mi tutor el Prof. José Miguel Díaz-Báñez por guiarme y ayudarme en cada paso de este trabajo. También agradecer a Fabio Rodríguez, por su dedicación constante a ofrecerme el soporte y la ayuda que he necesitado, y al Prof. Jesús Capitán, por aportar sus conocimientos para un trabajo más fructífero.

Terminar esta etapa y con ello este Trabajo de Fin de Grado no habría sido igual sin el apoyo de toda mi familia y amigos que siempre han estado ahí a pesar de cualquier circunstancia. Por ello quiero agradecerles enormemente a mi familia la gran oportunidad que me han regalado para conseguirlo y a mis amigos por ser quienes me han ayudado y motivado siempre a continuar.

Ana Victoria Canterla Martín
Sevilla, 2022

Resumen

Los vehículos aéreos no tripulados o drones se han convertido en una tecnología de gran interés mundial en los últimos años debido a su potencial para realizar un gran número de aplicaciones, lo que revierte no solo en aspectos económicos sino también en la protección del medio ambiente y en la seguridad de los operarios. Un problema crucial en este campo es el diseño de algoritmos eficientes que planifican trayectorias libres de colisiones entre los drones. Este Trabajo de Fin de Grado muestra un análisis comparativo entre dos algoritmos que resuelven dicho problema en un entorno 2D. Un primer algoritmo, conocido como ORCA, es un modelo descentralizado y diseñado para cualquier tipo de escenario y un segundo algoritmo, (CA-*nk*), es centralizado y diseñado para aplicaciones donde se utilizan un equipo pequeño de drones de autonomía limitada y se requieren trayectorias con un consumo eficiente de energía. Realizando un estudio computacional en distintos escenarios, se muestra el comportamiento de dichos algoritmos, llegando a la conclusión de que el método CA-*nk* resulta más eficiente en consumo de energía y en tiempo de ejecución de la tarea.

Abstract

Unmanned aerial vehicles or drones have become a technology of great global interest in recent years due to their potential to perform a large number of applications, which has implications not only for economic aspects but also for environmental protection and operator safety. A crucial problem in this field is the design of efficient algorithms that plan collision-free trajectories between drones. This work shows a comparative analysis between two algorithms that solve this problem in a 2D environment. A first algorithm, known as ORCA, is a decentralized model designed for any type of scenario. A second algorithm, (CA-nk), is centralized and designed for applications where a small team of drones with limited autonomy is used and energy efficient trajectories are required. By performing a computational study in different scenarios, a comparison between these algorithms is shown, concluding that the CA-nk is more efficient both in energy consumption and completion time for their tasks.

Índice Abreviado

Índice

Índice de Figuras

Índice de Tablas

Notación

$\mathbb{R}$	Cuerpo de los números reales
$\mathbb{C}$	Cuerpo de los números complejos
$\|\mathbf{v}\|$	Norma del vector $\mathbf{v}$
$\langle \mathbf{v}, \mathbf{w} \rangle$	Producto escalar de los vectores $\mathbf{v}$ y $\mathbf{w}$
$\|\mathbf{A}\|$	Determinante de la matriz cuadrada $\mathbf{A}$
$\det(\mathbf{A})$	Determinante de la matriz (cuadrada) $\mathbf{A}$
$\mathbf{A}^\top$	Transpuesto de $\mathbf{A}$
$\mathbf{A}^{-1}$	Inversa de la matriz $\mathbf{A}$
$\mathbf{A}^*$	Conjugado
sen	Función seno
tg	Función tangente
arc tg	Función arco tangente
arc cos	Función arco coseno
:	Tal que
$\stackrel{\text{def}}{=}$	Igual por definición
$\|\mathbf{x}\|$	Norma-2 del vector $\mathbf{x}$
$\|\mathbf{A}\|$	Cardinal, número de elementos del conjunto $\mathbf{A}$
$\mathbf{x}_i, i = 1,2,\ldots,n$	Elementos i, de 1 a n, del vector $\mathbf{x}$
$\mathrm{d}x$	Diferencial de x
$\leqslant$	Menor o igual
$\geqslant$	Mayor o igual
$\setminus$	Backslash
$\Leftrightarrow$	Si y sólo si
$\frac{a}{b}$	Fracción con estilo pequeño, a/b
Δ	Incremento
σ	Desviación típica
$\sim$	Equivalencia
ρ_0	Distancia límite de la influencia del campo potencial
ρ	Distancia más corta al obstáculo O
$\oplus$	Suma de espacios vectoriales
::	Proporción
$\exists$	Cuantificación existencial
$\in$	Relación de pertenencia
$\bar{\alpha}$	Media aritmética

1 Introducción

El desarrollo de los vehículos aéreos no tripulados, conocidos como drones, posee un amplio abanico de ventajas, siendo la más fundamental poder realizar sus tareas sin llevar un piloto humano, evitando así los riesgos que pueden tomar operaciones en zonas de difícil acceso [21].

En los últimos años se ha hecho una gran inversión en el campo de los drones y se ha observado una gran expansión en el uso no sólo de vehículos aéreos no tripulados, sino también de vehículos terrestres y de superficie marítima (UGV y USV), principalmente para vigilancia, cartografía e inspección.

Es este trabajo se abarca únicamente los vehículos cuyas funciones se desarrollan en el aire. El nivel de autonomía de estos vehículos depende de las tareas que puedan realizar o las decisiones que puedan tomar sin estar controlados de manera remota. Para ello, de manera general, disponen de diferentes tipos de sensores a bordo o se basan en la retroalimentación recibida de una cámara instalada en ellos, los cuales se utilizan para conocer la situación del entorno que les rodea y de esta forma toman decisiones autónomas en el tiempo de ejecución. Debido a su autonomía y capacidad para desplazarse lejos de las estaciones base o de sus operadores, es evidente la necesidad de disponer de un mecanismo a bordo para evitar colisiones con objetos y otros vehículos.

Dentro de las múltiples aplicaciones profesionales a las que se destinan estos tipos de vehículos caben destacar:

Cinematografía

Un interesante dominio de aplicación para UAVS es la industria del cine en la que los cineastas confían cada vez más en estos dispositivos para componer tomas que serían imposibles o extremadamente caras de producir [5].

Los drones ofrecen grados de libertad que ningún otro dispositivo de cámara podría proporcionar - grúas de cámara, *steadycam* o pistas de cámara cada uno tiene limitaciones físicas específicas.

Figura 1.1 Aplicación de un dron en una escena cinematográfica.

Mapeo y vigilancia

Demostrando su versatilidad, los drones se usan en temas de seguridad y vigilancia. Tomando como ejemplo una petrolera, el ahorro en recurso humano y tiempo es significativo si se trata de vigilar tramos de varios kilómetros. Los drones ya están equipados con sistemas de detección facial que pueden reconocer y alarmar sobre una persona, a larga distancia. Empresas de seguridad y sector militar son otros de sus usuarios más importantes.

Figura 1.2 Aplicación de un dron en la vigilancia del cumplimiento de distancia social en playas durante la pandemia por COVID-19.

Búsqueda y rescate

En caso de desastre, hay una necesidad inminente de asistencia robótica para llevar a cabo una operación de búsqueda y rescate eficaz, debido a su despliegue inmediato permisible [11]. La principal ventaja que tiene un robot de búsqueda y rescate sobre su homólogo humano es la velocidad relativa a la que puede entrar en un lugar de desastre y comenzar a recopilar información. Los robots pueden desplegarse en un lugar de desastre casi inmediatamente porque son prescindibles en relación con la vida humana.

Gracias a la optimización de las baterías y el poder de almacenamiento de información, los drones son los aliados perfectos cuando se trata de operaciones de rescate porque tienen la capacidad de cubrir grandes distancias y hacer el reconocimiento de personas extraviadas, gracias a sofisticados programas de detección y análisis de patrones físicos de la naturaleza [9].

Figura 1.3 Aplicación de un dron de búsqueda y rescate.

Agricultura de precisión

Abonar un terreno adecuadamente, realizar un análisis de suelos o una aspersión contra plagas, son algunas de las tareas que se pueden realizar mediante drones, evitando en este último caso, por ejemplo, los problemas de salud

que conllevan respirar los plaguicidas cuando se aplican manualmente [10].

Figura 1.4 Aplicación de un dron en agricultura de precisión.

Inspección de infraestructuras y construcciones

Estas plataformas pueden inspeccionar con frecuencia las obras de construcción, supervisar el trabajo en curso, crear documentos de seguridad e inspeccionar las estructuras existentes, en particular para las zonas de difícil acceso, evitando tragedias [6].

Figura 1.5 Aplicación de un dron en inspección de infraestructuras.

Manejo de desastres y salvamento público

Las aplicaciones de seguridad, como la gestión de los desastres naturales y los incidentes provocados por el hombre, dependen fundamentalmente de que se disponga rápidamente de un panorama de la situación de la zona afectada. Los sistemas de teleobservación basados en UAVs pueden constituir un instrumento esencial para captar imágenes aéreas en esos escenarios [7].

Figura 1.6 Aplicación de un dron en la vigilancia de un incendio forestal.

Transporte

Los drones tienen la capacidad de realizar registros de colisiones, monitoreo de vías y vigilancia ciudadana con la capacidad de vincularse a las cámaras de seguridad de la ciudad y al centro de control de la policía.

Figura 1.7 Aplicación de un dron en la vigilancia del tráfico.

1.1 Motivación y objetivo

La motivación de este trabajo está basada en el uso de un equipo pequeño de drones que tienen batería limitada y se requiere ahorro de energía en las tareas a realizar. Por ejemplo, el caso en el que se quiere usar un equipo de drones para cubrir un evento deportivo en tiempo real, realizando cada uno de ellos los desplazamientos que le ordene el director de TV. Las restricción fundamental del problema es la limitación de autonomía de cada dron (son pequeños multirotores) y el hecho de que viajen por lugares comunes exige un algoritmo eficiente en energía que evite colisiones entre ellos.

El objeto de estudio es realizar la comparación de dos algoritmos de la literatura para ver cuál ofrece trayectorias más satisfactorias para el escenario planteado.

1.2 Estructura del trabajo

Tras la breve introducción en la que se han presentado diferentes tipos de aplicaciones profesionales para drones, se realiza una revisión del estado del arte en el que se verán difentes métodos de evitación de colisiones y se expone

una breve introducción a los dos algoritmos que serán utilizados, con acrónimos ORCA y CA-*nk*. Posteriormente, se desarrollará más en profundidad cada uno de los algoritmos utilizados (ORCA y CA-*nk*), explicando la base teórica de cada uno de ellos y su implementación. Finalmente, se explicarán los escenarios diseñados donde se ejecutarán los experimentos, las métricas que se utilizarán para realizar la comparación, se llevará a cabo el análisis de los resultados obtenidos en los experimentos y se describirán las conclusiones.

2 Estado del Arte

2.1 Introducción

La capacidad de evitar colisiones es un requisito crítico para los vehículos aéreos no tripulados cuando comparten un espacio común donde desarrollan sus tareas. Se trata de un problema de planificación donde la detección y evitación de colisiones sigue siendo un desafío por resolver de la manera más eficiente posible. Los primeros métodos de evitación fueron estudiados para los vehículos móviles terrestres, por lo que esta área de investigación ha sido muy activa en lo relacionado con la robótica trabajando en dos dimensiones. En cambio, para los métodos aplicados a sistemas de UAVs, podrá requerirse tanto dos dimensiones (dimensión utilizada en los algoritmos que serán comparados en este trabajo) como tres dimensiones, además de requerir poder ser ejecutado con bajas necesidades computacionales, debido a las necesidades de tiempo real que estos vehículos aéreos plantean. En este capítulo se realizará una introducción a algunos de los diferentes métodos de evitación de colisiones, inclusive los dos protagonistas de este trabajo. Dentro de la gran variedad de estos métodos están los de optimización geométrica, métodos probabilísticos, control predictivo [2], replanificación de ruta con A*[1], potenciales [12], etc. Algunos de estos métodos están pensados para ser estudiados en un escenario con únicamente dos UAVs, sin embargo, el desarrollo de este trabajo está basado en escenarios en lo que existen múltiples UAVs.

2.2 Métodos basados en evitación de obstáculos para vehículos aéreos no tripulados

2.2.1 Mediante optimización geométrica

ORCA (*Optimal Reciprocal n-Body Collision Avoidance*)

Se trata de un algoritmo con enfoque descentralizado, es decir, cada UAV resuelve el conflicto de colisión de manera independiente, dando por hecho que, en cada conflicto, el otro UAV responsable de la colisión tomará la mitad de la responsabilidad de resolver la evitación. Se consideran múltiples UAVs en el mismo escenario, aunque la evitación se realice por parejas. Está formulado en el espacio de velocidades y como su propio nombre indica, es la versión optimizada de Reciprocal Collision Avoidance, en la que se introducen las velocidades de optimización, de modo que los UAVs tendrán que desviarse lo menos posible de sus trayectorias en cada momento para evitar colisiones [14].

CA-nk (*Collision Avoidance for n drones and k turn angles*)

Se enfoca en equipos de alrededor de 4-6 UAVs. El método consiste en verificar iterativamente los conflictos en cortos intervalos de tiempo y resolver los conflictos eficientemente de manera centralizada para esos intervalos, es decir, todas las situaciones son tratadas simultáneamente y se resuelve el conflicto de una sola vez (todos los UAVs negocian con todos los demás UAVs para tomar una decisión). De esta manera se asignan direcciones libres de colisiones a todos los UAVs. Cada UAV genera un conjunto discreto de direcciones posibles (cono de direcciones) y uno de ellos envía a los demás una solución óptima para la asignación de ángulo de giro de los múltiples vehículos [3].

2.2.2 Método probabilístico en 3D

En este método, estudiado en un espacio de tres dimensiones en [8], se considera cada UAV como un sistema de masa puntual con velocidad constante, que asume realizar siempre maniobras coordinadas y que se encuentra en un escenario plano no giratorio. Considera el caso no cooperativo (descentralizado), por lo que cada UAV tiene total autoridad para evitar la colisión.

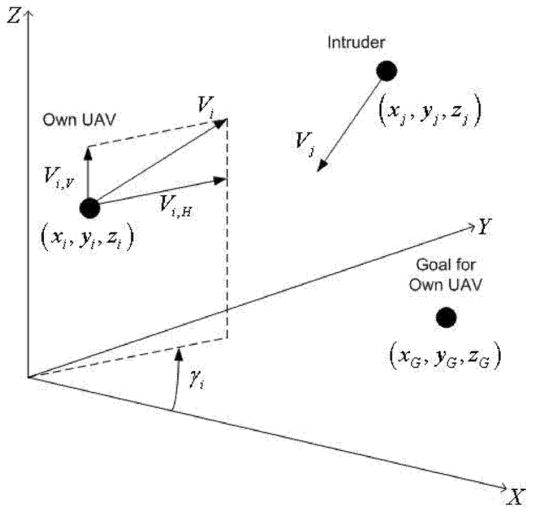


Figura 2.1 Vista geométrica en 3-D tomada de [8].

Se considera colisión al encontrarse dos o más UAVs entre una distancia mínima. Al ser esta colisión tridimensional, se descompone en colisiones horizontales y verticales. De esta manera, las colisiones ahora se considerarán ocasionadas cuando se pierda una separación mínima simultáneamente en las direcciones horizontales y verticales. La colisión es evitada a la vez que el UAV se dirige a su punto destino, por lo que la distancia entre este punto y el UAV se irá decrementando hasta cero en el último momento. En este caso se utiliza un algoritmo basado en los algoritmos de Monte Carlo, que otorgan la respuesta exacta con elevada probabilidad. En este algoritmo, cada UAV obtiene información de sí mismo y del UAV intruso en cada intervalo de tiempo mediante ADS-B (Automatic Dependent Surveillance Broadcast). El UAV principal es el que realiza la evitación guiada, mientras que el intruso sigue su trayectoria predeterminada en el escenario.

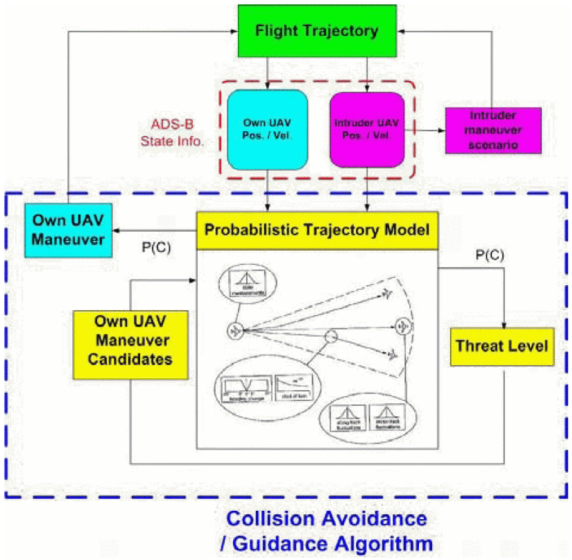
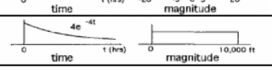


Figura 2.2 Algoritmo de evitación de colisiones tomado de [8].

La información adquirida mediante ADS-B también incluye errores de posición y estimaciones futuras, con los que se podrá calcular la probabilidad de que se produzca una colisión calculando previamente el modelado probabilístico de la trayectoria. Para ello se hace uso de la información adquirida por ADS-B de los errores.

Tabla 2.1 Parámetros del modelo de trayectoria tomados de [8].

	Uncertainty Factor	Modeled Distribution
Own and Intruder UAV	Lateral Position Error	Gaussian σ = 50 m
	Vertical Position Error	Gaussian σ = 30 m
	Along-Track Speed Error	Gaussian σ = 15 kts
	Cross-Track Position Error	Gaussian σ = 1 nmi
Intruder UAV Only	Heading Change	
	Altitude Change	

(15 kts = 7.72m/s , 1 nmi = 1.852 km)

Finalmente se calcula la probabilidad de colisión utilizando el algoritmo de Monte Carlo.

$$P(C) = \frac{\#ofCollisions}{\#ofMonteCarloSimulations} \tag{2.1}$$

Tabla 2.2 Nivel de alerta tomado de [8].

P(C)	Threat Level
$0 \sim 5\%$	Level 0
$5 \sim 10\%$	Level 1
$10 \sim 20\%$	Level 2
$20 \sim 30\%$	Level 3
$30 \sim 100\%$	Level 4

Si está en el nivel 0, se considera que la colisión no tendrá lugar y el proceso continuará guiando al UAV hacia su punto destino, mientras que si el nivel está por encima de 0 se prevé que se producirá una colisión por lo que se ejecutará el proceso de evitación.

2.2.3 Métodos potenciales

En este tipo de método, los UAVs y los obstáculos son considerados como partículas con carga del mismo signo que se mueven en un campo de fuerzas, por lo que entre ellos se producirá un efecto repulsivo. Cada punto objetivo se considera un polo atractivo para cada UAV correspondiente, por lo que se atraerán entre ellos [12]. Cuando aumenta la velocidad de un UAV hacia un obstáculo, su fuerza de repulsión se incrementará y esta dependerá tanto de la velocidad del UAV como de la velocidad relativa a los obstáculos.

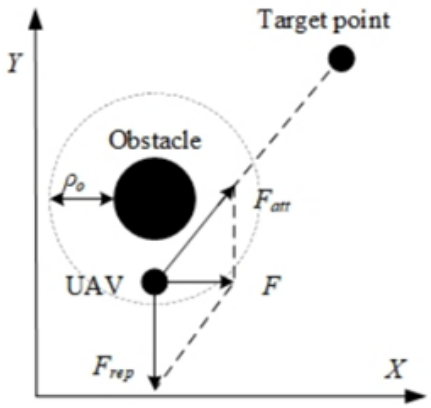


Figura 2.3 Diagrama de fuerzas del método tradicional de campos de potencial artificial tomado de [12].

2.2.4 Replanificación de ruta con A*

En este método, el escenario está discretizado dividiendo el mapa en una cuadrícula. Esta cuadrícula está representada por una gráfica ponderada en la que cada celda de la cuadrícula se corresponde con cuatro puntos de la gráfica. Los bordes conectan los puntos de manera ortogonal o vertical y la ponderación de cada borde se corresponde con la distancia entre los puntos que lo delimitan. Si un obstáculo es detectado, los bordes que conectan las celdas que lo forman y las celdas dentro de la distancia de seguridad son eliminadas de la gráfica. Tras esta eliminación, se realiza una replanificación de la ruta desde la posición actual del UAV hasta el punto objetivo, eligiendo la ruta más corta.

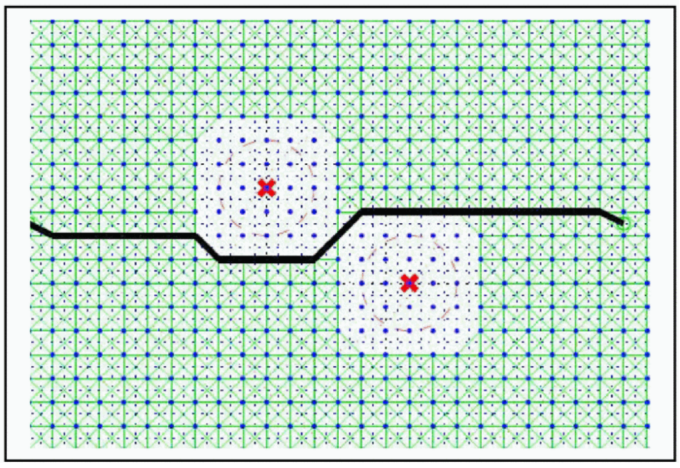


Figura 2.4 Evitación de obstáculos por replanificación de ruta con búsqueda A* tomado de [1].

3 Algoritmos

En este capítulo se va a tratar el desarrollo teórico de los dos algoritmos a comparar. Básicamente, hacemos una exposición basada en los artículos correspondientes, [14] para ORCA y [3] para CA-*nk*.

En ORCA, se presenta un enfoque formal para RCA (*Reciprocal n-body collision avoidance*) en el cual varios robots móviles que se mueven en un espacio común con más robots, evitan las colisiones entre sí de manera independiente y sin comunicación con los demás robots. Gracias a la definición de *Velocity Obstacle* [4], la resolución del problema de planificación de trayectorias se reduce a un programa lineal de baja dimensión.

Este problema es fundamental en robótica, que normalmente se define en el contexto de un robot móvil autónomo que navega en un espacio con obstáculos en movimiento, donde el robot móvil funciona empleando un ciclo continuo de detección y actuación. En cada uno de estos ciclos el robot debe dirigirse hacia su objetivo a la vez que calcula una acción basada en las observaciones del entorno de forma que no colisione con ningún obstáculo u otras entidades móviles.

En CA-*nk*, se propone una novedosa estrategia de asignación de ángulos de giro para la planificación de trayectorias sin colisiones en sistemas con múltiples UAVs. El algoritmo permite a cada UAV alcanzar el destino asignado de forma segura. Establece intervalos de tiempos cortos y consecutivos, y en cada intervalo se resuelven los posibles conflictos de manera centralizada. Para ello, se genera un conjunto discreto de direcciones posibles para cada UAV y se resuelve de forma eficiente la asignación de ángulos de giro para una trayectoria libre de colisiones que minimice las desviaciones del plan de vuelo original. Aunque las trayectorias completas no son óptimas, el sistema puede reaccionar ante posibles fallos durante la ejecución, ya que los conflictos se resuelven en cada intervalo de tiempo.

3.1 ORCA

En [14] se presenta un enfoque formal para RCA en el que varios robots móviles deben evitar colisiones entre sí mientras se mueven en un espacio de trabajo común. En este enfoque, cada robot actúa de forma totalmente independiente, sin comunicación con los demás robots.

El problema de planificación de trayectorias libres de colisión ha sido muy estudiado para el caso de un solo robot que evita obstáculos, pero en el desarrollo de este algoritmo, se aborda un problema más complejo y menos estudiado como es RCA, en el que se evitan colisiones, ahora entre múltiples robots.

Este problema se define como sigue. Sea un conjunto n de robots que comparten un entorno. Para simplificar, suponemos que los robots tienen forma de disco y se mueven en el plano $\mathbb{R}^2$. Cada robot A tiene una posición actual p_A (el centro de su disco), una velocidad actual V_A y un radio r_A. Estos parámetros forman parte del estado externo del robot, es decir, suponemos que pueden ser observados por otros robots. Además, cada robot tiene una velocidad máxima v^{max} y una velocidad preferida v^{pref}, que es la velocidad que el robot asumiría si no hubiera otros robots en su camino (velocidad dirigida hacia el objetivo del robot con un magnitud igual a la velocidad preferida del robot). Consideramos que estos parámetros forman parte del estado interno del robot y, por tanto, no pueden ser observados por otros robots.

La tarea consiste en que cada robot A seleccione de forma independiente y simultánea una nueva velocidad v^{new} para sí mismo, de forma que se garantice que todos los robots estén libre de colisiones durante al menos una cantidad de tiempo preestablecida cuando continúen moviéndose a su nueva velocidad. Como objetivo secundario,

los robots deben seleccionar su nueva velocidad lo más cerca posible de su velocidad preferida. Los robots no pueden comunicarse entre sí y sólo pueden utilizar las observaciones de la posición y velocidad actuales del otro robot. Sin embargo, cada uno de los robots puede suponer que los demás robots utilizan la misma estrategia que él para seleccionar una nueva velocidad.

A este problema se le llama RCA. Obsérvese que este problema no puede resolverse utilizando la coordinación central, ya que la velocidad preferida de cada robot sólo la conoce el propio robot. A continuación se presenta una condición suficiente para que cada robot seleccione una velocidad que esté libre de colisiones durante al menos un tiempo predefinido.

3.1.1 Reciprocal Collision Avoidance

Velocity Obstacle

Sean A y B dos robots, se define *Velocity Obstacle* como el conjunto de velocidades relativas de A con respecto a B que resultarán en una colisión entre A y B antes de un tiempo τ.

Sea $D(p,r)$ un disco de radio r y centrado en p:

$$D(p,r) = \{q, ||q-p|| < r\} \tag{3.1}$$

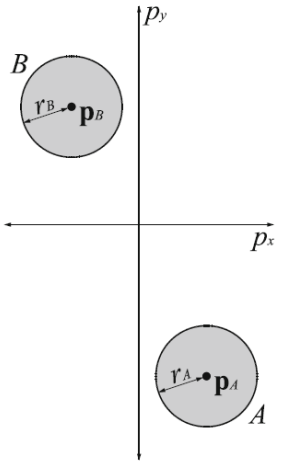


Figura 3.1 Configuración de dos UAVs A y B tomado de [14].

Se define Velocity Obstacle de A inducida por B en el tiempo τ, denotada $VO^{\tau}_{A|B}$, como:

$$VO^{\tau}_{A|B} = \{v|\exists t \in [0,\tau] :: tv \in D(p_B - p_A, r_A + r_B)\} \tag{3.2}$$

Sean v_A y v_B las velocidades actuales de los drones A y B respectivamente. La definición de *Velocity Obstacle* (ver Fig. 3.2) implica que si la velocidad relativa entre A y B pertenece al conjunto ($v_A - v_B \in VO^{\tau}_{A|B}$), colisionarán antes de un tiempo τ si continúan moviéndose a esas velocidades, respectivamente.

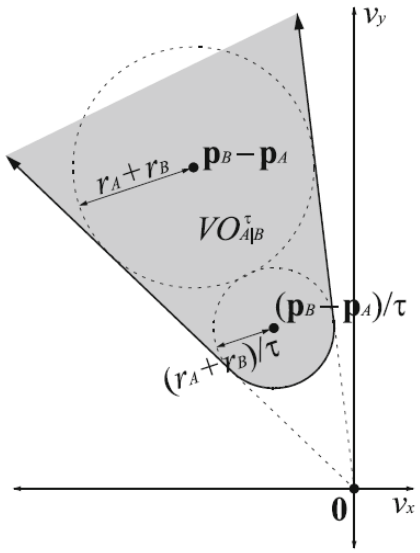


Figura 3.2 *Velocity Obstacle* $VO^{\tau}_{A|B}$ (gris) tomado de [14].

Sea $X + Y$ la suma de Minkowski de X e Y:

$$X \oplus Y = \{x + y \mid x \in X, y \in Y\}, \tag{3.3}$$

Entonces, para cualquier conjunto V_B, si $v_B \in V_B$ y $v_A \notin VO^{\tau}_{A|B} \oplus V_B$, entonces A y B no colisionarán en al menos un tiempo τ.

Esto nos lleva a la definición del conjunto de velocidades que evitan la colisión para A, $CA^{\tau}_{A|B}(V_B)$, tomando B su velocidad V_B:

$$CA^{\tau}_{A|B}(V_B) = \{v | v \notin VO^{\tau}_{A|B} \oplus V_B\} \tag{3.4}$$

3.1.2 Optimal Reciprocal Collision Avoidance

Una vez definido el concepto de Velocity Obstacle, elegiremos conjuntos de velocidades permitidas, V_A para A y V_B para B de manera que $CA^{\tau}_{A|B}(V_B) = V_A$ y $CA^{\tau}_{B|A}(V_A) = V_B$, lo que implica que éstas evitarán la colisión de manera recíproca. Además, garantizan que A y B no colisionarán en al menos un tiempo τ.

En este algoritmo, mejorando la versión anterior, se elige la velocidad óptima, que es la que se considera con menor desviación hacia su punto objetivo.

Para velocidades óptimas v_A^{opt} para A y v_B^{opt} para B, definimos los conjuntos $ORCA^{\tau}_{A|B}$ para A y $ORCA^{\tau}_{B|A}$ para B, respectivamente, como sigue:

Asumiendo que los drones A y B toman sus velocidades óptimas, v_A^{opt} y v_B^{opt}, respectivamente, y que vayan encaminándose hacia una colisión, es decir, $v_A^{opt} - v_B^{opt} \in VO^{\tau}_{A|B}$, se define $\vec{u}$ como el vector desde el punto que define la velocidad relativa entre A y B ($v_A^{opt} - v_B^{opt}$) hasta el punto más cercano en el límite del cono representante de Velocity Obstacle, esto es,

$$\vec{u} = \arg \min_{v \in \partial VO^{\tau}_{A|B}} (||v - (v_A^{opt} - v_B^{opt})||) - (v_A^{opt} - v_B^{opt}) \tag{3.5}$$

Se define $\vec{n}$ como el vector normal hacia fuera del límite de *Velocity Obstacle* en el punto $(v_A^{opt} - v_B^{opt}) + \vec{u}$.

De esta manera, $\vec{u}$ es el cambio más pequeño requerido a la velocidad relativa entre A y B para evitar una colisión en al menos un tiempo τ. Para ello el dron A adaptará su velocidad en $\frac{1}{2}$ de $\vec{u}$ y asumirá que B lo hará en la otra mitad, repartiendo así la responsabilidad de evitar la colisión.

Por lo tanto, el conjunto $ORCA^{\tau}_{A|B}$ de velocidades permitidas para A :

$$ORCA^{\tau}_{A|B} = \{v|(v - (v^{opt}_{A} + \frac{1}{2}\vec{u})) \cdot \vec{n} \geq 0\} \tag{3.6}$$

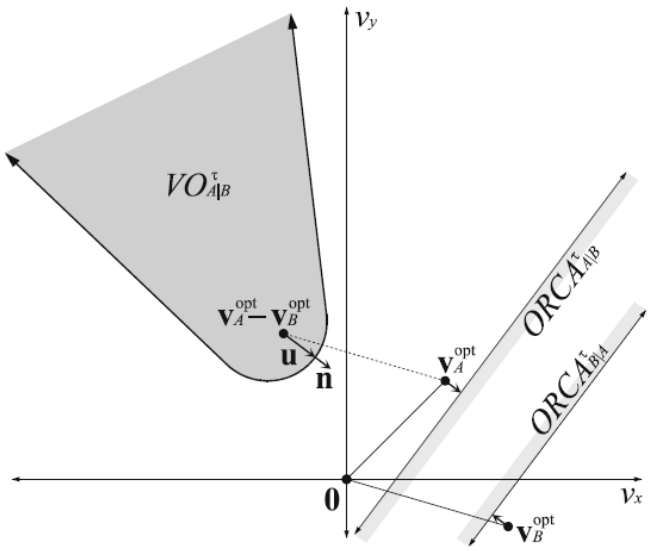


Figura 3.3 El conjunto $ORCA^{\tau}_{A|B}$ de velocidades permitidas para A para evitar la colisión recíproca óptima con B es un semiplano delimitado por la línea perpendicular a u a través del punto $v^{opt}_{A} + \frac{1}{2}\vec{u}$, donde $\vec{u}$ es el vector de $v^{opt}_{A} - v^{opt}_{B}$ al punto más cercano en el límite de $VO^{\tau}_{A|B}$. Tomado de [14].

Esto es aplicable para un par de drones. A continuación, se tratará con el problema de cómo aplicar el algoritmo ORCA para evitar la colisión de n drones en un escenario con múltiples UAVs en movimiento (n-Body Collision Avoidance). Además, se desarrollará la incorporación de obstáculos estáticos.

Se muestra el esquema de ciclo continuo que ejecuta cada UAV independientemente para la detección de colisiones y actuación (ver Fig. 4.4).

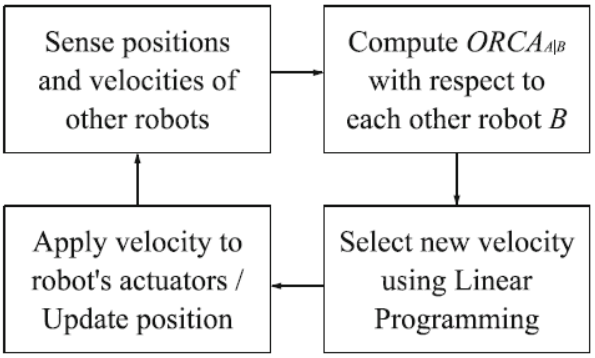


Figura 3.4 Ciclo ejecutado por cada UAV tomado de [14].

De esta manera, cada UAV realiza este ciclo dentro de un tiempo τ. En cada iteración, cada UAV adquiere la información del radio, la posición actual y la velocidad de optimización actual de todos los demás UAVs y de sí mismo.

Basado en esta información, el UAV deduce el semiplano de velocidades permitidas $ORCA^{\tau}_{A|B}$ con respecto a cada otro UAV B.

Por lo tanto, el conjunto de velocidades para A que estarán permitidas de manera que no se produzca colisión con los demás drones en al menos un tiempo τ es la intersección de los semiplanos de las velocidades permitidas inducidas por los demás drones, y este conjunto se denomina $ORCA_A^\tau$:

$$ORCA_A^\tau = D(0, v_A^{max}) \cap ORCA_{A|B}^\tau \tag{3.7}$$

Nótese que esta definición también incluye $D(0, v_A^{max})$, siendo esta la restricción de velocidad máxima del dron A formada por un disco centrado en 0 y radio v_A^{max} (ver fig. 3.5)

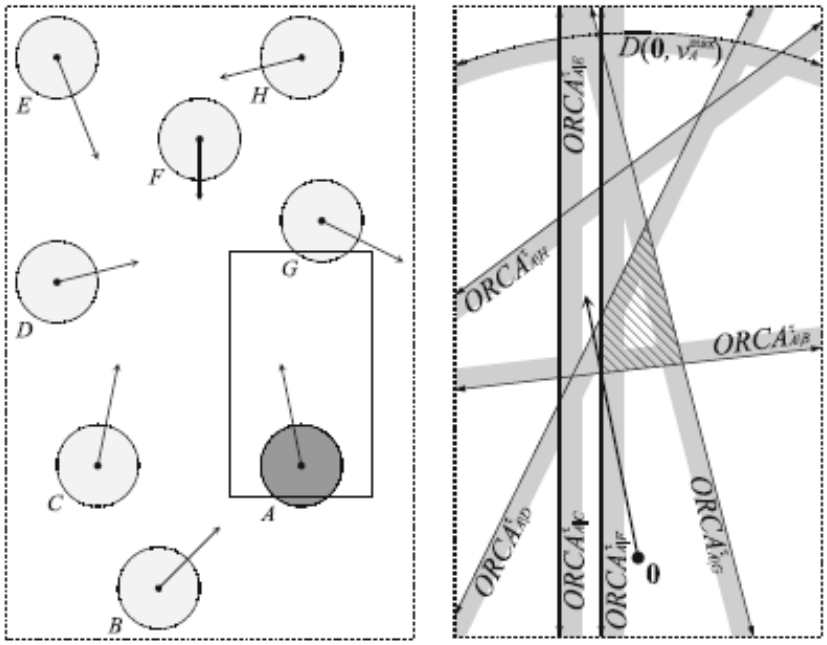


Figura 3.5 Restricciones para ocho robots con algoritmo ORCA tomado de [14].

El siguiente paso es que A selecciona una nueva velocidad denominada V_A^{NEW} por sí mismo. Esta velocidad será la que esté más cerca de su velocidad preferida V_A^{PREF} dentro de todas las velocidades permitidas. Para ello, calcula el valor v que dentro de la región de las velocidades permitidas, minimice el módulo entre la velocidad calculada y la velocidad preferida:

$$v_A^{new} = \arg \min_{v \in ORCA_A^\tau} ||v - v_A^{pref}|| \tag{3.8}$$

Este proceso se puede implementar de manera eficiente utilizando programación lineal [14].

3.1.3 Implementación

Para este trabajo se han adaptado los códigos publicados en el repositorio de GitHub [1], programados en C++. Estos códigos se basan en el uso de las funciones de la librería RVO2 (*Reciprocal Collision Avoidance for Real-Time Multi-Agent Simulation*) [13].

Un programa que realice la simulación de esta biblioteca requiere la siguiente estructura:

Primero debe incluirse la librería RVO.h y crear una instancia del simulador RVO::RVOSimulator. Tras realizar el paso anterior, se declaran dos etapas:

PRIMERA ETAPA: se prepara el escenario, utilizando la función setupScenario(sim) (Fig.3.6) en el que se definirán las posiciones origen (con la función `addAgent()`) y objetivo (`goals.pushback()`) de cada dron, además de especificar tanto los parámetros predeterminados para los drones (`setAgentDefaults()`) como el paso temporal

[1] *https://github.com/snape/RVO2*

global de la simulación (`setTimeStep()`)[2]. Si el escenario está formado también por obstáculos estáticos, estos se crearán en esta función, definiendo los vértices de cada uno (`vértices.pushback()`). Para que los obstáculos se tengan en cuenta en la simulación se llama a la función `processObstacles()`.

```
void setupScenario(RVO::RVOSimulator* sim) {
// Specify global time step of the simulation.
sim->setTimeStep(0.25f);

// Specify default parameters for agents that are subsequently added.
sim->setAgentDefaults(15.0f, 10, 10.0f, 5.0f, 2.0f, 2.0f);

// Add agents, specifying their start position.
sim->addAgent(RVO::Vector2(-50.0f, -50.0f));
sim->addAgent(RVO::Vector2(50.0f, -50.0f));
sim->addAgent(RVO::Vector2(50.0f, 50.0f));
sim->addAgent(RVO::Vector2(-50.0f, 50.0f));

// Create goals (simulator is unaware of these).
for (size_t i = 0; i < sim->getNumAgents(); ++i) {
goals.push_back(-sim->getAgentPosition(i));
}

// Add (polygonal) obstacle(s), specifying vertices in counterclockwise order.
std::vector<RVO::Vector2> vertices;
vertices.push_back(RVO::Vector2(-7.0f, -20.0f));
vertices.push_back(RVO::Vector2(7.0f, -20.0f));
vertices.push_back(RVO::Vector2(7.0f, 20.0f));
vertices.push_back(RVO::Vector2(-7.0f, 20.0f));

sim->addObstacle(vertices);

// Process obstacles so that they are accounted for in the simulation.
sim->processObstacles();
}
```

Figura 3.6 Definición del escenario.

SEGUNDA ETAPA: se ejecuta la realización real de la simulación. En ella se procesan las siguientes funciones en bucle, actualizando cada vez la posición y velocidad bidimensional de cada dron, hasta que todos llegan a su punto objetivo: updateVisualization(sim) (Fig. 3.7), cuya finalidad es mostrar en el terminal la hora global actual y la posición de todos los drones en cada instante; setPreferredVelocities(sim) (Fig. 3.8), en la cual el simulador calcula las velocidades reales de los drones e intenta seguir lo más cerca posible a las velocidades preferidas mientras garantiza que no hay colisiones. Estas velocidades preferidas se establecen para cada dron de manera que, si este se encuentra aún lejos de su objetivo, la velocidad preferida se establece como un vector unitario hacia el punto objetivo de ese dron. Por el contrario, si el dron está dentro de un radio de su objetivo definido, la velocidad preferida se establece en cero;reachedGoal(sim) (Fig. 3.9), utilizada como condición del bucle principal que comprueba si todos los drones han llegado a sus objetivos, verificando las distancias con respecto a estos. En ese caso, terminaría la simulación.

```
void updateVisualization(RVO::RVOSimulator* sim) {
// Output the current global time.
std::cout << sim->getGlobalTime() << " ";

// Output the position for all the agents.
for (size_t i = 0; i < sim->getNumAgents(); ++i) {
std::cout << sim->getAgentPosition(i) << " ";
}

std::cout << std::endl;
}
```

Figura 3.7 Visualización actualizada de las posiciones de los drones.

[2] En cada escenario se selecciona un *TimeStep* para el UAV. Cada UAV vuela durante un *TimeStep* con una dirección y velocidad seleccionadas. Después de cada *TimeStep*, los UAV buscan colisiones y las evitan utilizando giros y maniobras de variación de velocidad.

```
void setPreferredVelocities(RVO::RVOSimulator* sim) {
// Set the preferred velocity for each agent.
for (size_t i = 0; i < sim->getNumAgents(); ++i) {
if (absSq(goals[i] - sim->getAgentPosition(i)) < sim->getAgentRadius(i) * sim->getAgentRadius(i) ) {
// Agent is within one radius of its goal, set preferred velocity to zero
sim->setAgentPrefVelocity(i, RVO::Vector2(0.0f, 0.0f));
} else {
// Agent is far away from its goal, set preferred velocity as unit vector towards agent's goal.
sim->setAgentPrefVelocity(i, normalize(goals[i] - sim->getAgentPosition(i)));
}
}
}
```

Figura 3.8 Asignación de las velocidades preferidas.

```
bool reachedGoal(RVO::RVOSimulator* sim) {
// Check whether all agents have arrived at their goals.
for (size_t i = 0; i < sim->getNumAgents(); ++i) {
if (absSq(goals[i] - sim->getAgentPosition(i)) > sim->getAgentRadius(i) * sim->getAgentRadius(i)) {
// Agent is further away from its goal than one radius.
return false;
}
}
return true;
}
```

Figura 3.9 Cálculo del fin de las trayectorias.

Tras el análisis profundo de los códigos, se procedió a modificar el ejemplo Circle.cpp (ubicado en la carpeta de ejemplos del repositorio) utilizando la herramienta Visual Studio y añadiendo todos los cálculos requeridos para la obtención de las medidas necesarias para la comparación. Su estructura principal engloba las funciones anteriormente descritas (ver Fig. 3.10).

```
#include <RVO.h>

std::vector<RVO::Vector2> goals;

int main()
{
// Create a new simulator instance.
RVO::RVOSimulator* sim = new RVO::RVOSimulator();

// Set up the scenario.
setupScenario(sim);

// Perform (and manipulate) the simulation.
do {
updateVisualization(sim);
setPreferredVelocities(sim);
sim->doStep();
} while (!reachedGoal(sim));

delete sim;
}
```

Figura 3.10 Estructura principal.

3.2 CA-*nk*

El algoritmo CA-*nk* está enfocado está para escenarios con un equipo pequeño de agents donde se requiere mínimo gasto de energía en la ejecución de las tareas, al tener los drones una autonomía limitada. Por ejemplo, el caso de un equipo de vehículos aéreos no tripulados que cooperan para la producción de eventos al aire libre. En concreto, un equipo entre cuatro y seis drones que comparten un espacio de trabajo, en el que a cada uno se le pueda asignar un objetivo diferente para rastrear y grabar durante el evento. Para ello, evitar las colisiones en el espacio compartido es primordial. En estos contextos, se puede preferir métodos que varíen las direcciones de velocidad en lugar de variar la celeridad, tanto por el ahorro de energía como por la calidad de las imágenes en la grabación. Además, se espera que estas grabaciones del evento sean retransmitidas en directo, por lo que se puede suponer una comunicación que

posea un nodo central, y de esta manera se permita el uso de enfoques centralizados, siempre que la ejecución sea eficiente y pueda realizarse en tiempo real.

En este escenario, en el trabajo [3] se propone un método novedoso de planificación de trayectorias sin colisiones. A continuación desarrolamos las ideas del procedimiento propuesto en [3].

El método consiste en la comprobación iterativa de conflictos en intervalos cortos de tiempo y en la resolución de conflictos de forma eficiente y centralizada para esos intervalos, asignando direcciones libres de colisiones a todos los drones. Cada dron genera un conjunto discreto de direcciones posibles (cono de direcciones) y uno de ellos envía a los demás una solución óptima (según el criterio elegido) para la asignación de ángulos de giro de varios vehículos.

En el desarrollo de este algoritmo se asume velocidad constante (no necesariamente la misma para todos) y la maniobra para evitar la colisión se centra únicamente en cambios de direcciones de velocidad. Por otra parte, se minimiza la desviación máxima con respecto al segmento que une los puntos de partida y llegada; de esta forma se pretende obtener ángulos de giro similares para todos los UAVs.

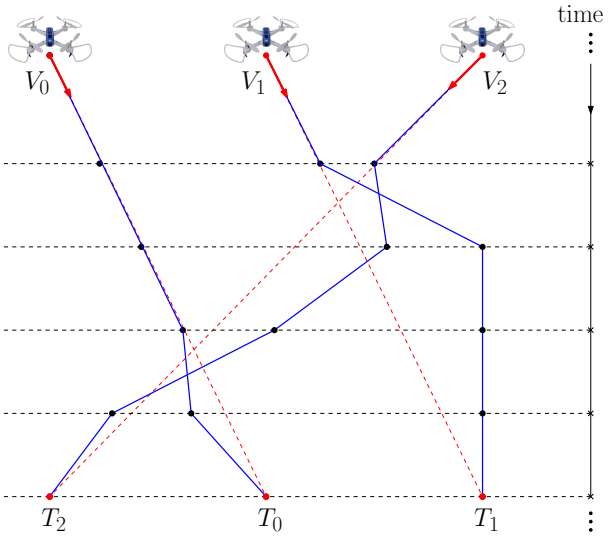


Figura 3.11 Ejemplo de planificación de trayectorias sin colisiones para 3 UAVs, resolviendo el problema de CA-*nk* en 5 intervalos de tiempo tomado de [3]. La línea discontinua es la recta entre el punto inicial y objetivo para cada UAV (treyectoria óptima si no hubiera colisiones).

El problema abordado en este trabajo considera un equipo de n UAVs, cada uno modelado como vehículo V_i, con una posición inicial y un punto objetivo T_i a visitar en un espacio común (Fig. 3.11). El objetivo es calcular trayectorias libres de colisiones para los UAVs, de forma que se minimice la máxima desviación con respecto al plan de vuelo original.

Se define $V = \{V_0, V_1, ..., V_{n-1}\}$ como un conjunto de n UAVs que trabajan en una región Q del espacio bimensional. Cada UAV tiene una posición inicial y un vector de velocidad, una velocidad constante pero no necesariamente igual para todos los UAVs y un conjunto de k vectores de velocidad posibles $H_i = v_{i,0}, v_{i,1}, ..., v_{i,k-1}$, esto es, un rango alrededor de su dirección inicial. Su objetivo es asignar a cada UAV un vector de velocidad de tal manera que no se produzcan colisiones en esa región Q.

Se modela cada UAV como una esfera de radio r, por lo que se considera colisión si la distancia euclidiana entre sus centros es menor a $2r$. Se propone un algoritmo centralizado que asume que cada UAV posee información de los demás UAVS como posición y velocidad. Cada UAV en cada intervalo de tiempo, inicia su trayectoria con una velocidad inicial que apunta hacia su objetivo y en caso de detectar una colisión, el algoritmo *CollisionAvoidance(n,k)* lo resuelve. En este caso centralizado, un UAV puede ser el que se encargue de computar y comunicar la solución a los demás.

Para aplicar la resolución del problema de colisiones con múltiples UAVs, asumiremos que los UAVs tienen forma de disco y que se mueven en un mismo plano. Esto puede ampliarse también para un escenario 3D. Primero introducimos una transformación geométrica y derivamos los resultados teóricos que son clave para resolver el problema de forma eficiente.

Asumimos un movimiento rectilíneo con velocidad constante para los UAVs, por lo que, si un vehículo V se mueve desde la posición p con velocidad v entonces, después del tiempo t su posición es: $V(t) = p + tv$. No se modela la dinámica de los vehículos ni los factores externos como el viento, sino que se replantea cada lapso de tiempo en un horizonte de tiempo determinado, con el fin de tener en cuenta esas incertidumbres y reaccionar a tiempo.

Consideramos la transformación geométrica $M(v)$ que mapea $V(t)$ en $\tilde{V}(t) = V(t) - tv$ (una traslación con dirección -v en cada instante t). Aplicando el mapeo $M(v_{0,1})$ a dos vehículos V_0 y V_1 con velocidades $v_{0,i}$ y $v_{1,j}$, $1 \leq i,j \leq k$, obtenemos las nuevas ecuaciones de movimiento de los UAVs:

$$\tilde{V_0}(t) = V_0(t) - t_{v0,i} = p_0; \tag{3.9}$$

$$\tilde{V_1}(t) = V_1(t) - t_{v0,i} = p_1 + t(v_{1,i} - v_{0,i}); \tag{3.10}$$

Sea p_i la posición inicial de cada dron. Obtenemos las ecuaciones de movimiento de los UAVs 0 y 1, realizando una traslación correspondiente a cada uno en un instante t (aplicando la transformación geométrica $M(v_0,i)$).

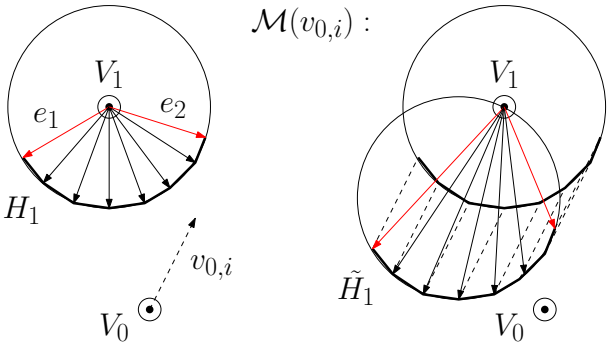


Figura 3.12 Transformación geométrica $M(v_0,i)$ para vehículos V_0 y V_1 tomado de [3].

Obsérvese (Fig. 3.12) que con la transformación geométrica $M(v_{0,i})$ el vehículo V_1 se mueve con dirección $v_{1,j}$ - $v_{0,1}$ y el vehículo V_0 permanece quieto, es decir, se convierte en un obstáculo estático para V_1.
Sea s_i el módulo de la velocidad del vehículo V_i, ($s_i = \|v_{i,j}\|$ para todo $v_{i,j} \in H_i$). Obsérvese que si $s_i < s_j$, entonces V_j es más rápido que V_i. Entonces, obtenemos el resultado que se muestra en la Fig. 3.12 .

El siguiente resultado se demuestra en [3]:

Lema 1: Sean V_0 y V_1 dos vehículos que vuelan en el mismo plano tales que $s_0 \leq s_1$. Tomamos $v_{0,i} \in H_0$ y dejamos que $\tilde{H}_1$ sea el conjunto de velocidades de V_i tras aplicar el mapeo $M(v_{0,i})$. Entonces, el orden radial de H_1 y $\tilde{H}_1$ con respecto a p_1 es el mismo.

La siguiente definición va a permitir encontrar un algoritmo efficiente.

Definición 1: Sean V_0 y V_1 dos vehículos con centros p_0 y p_1 respectivamente. Supongamos que V_0 es un vehículo estático y que V_1 tiene un conjunto de posibles vectores de velocidad H_1 (los vectores de velocidad pueden tener diferente módulo). Las velocidades extremas de V_1 con respecto a V_0 son los vectores vecinos en H_1 del vector -$\vec{p_1 p_0}$. Podemos ver en la Fig. 3.13 que H_1 está totalmente contenida en la cuña gris definida por las velocidades extremas.

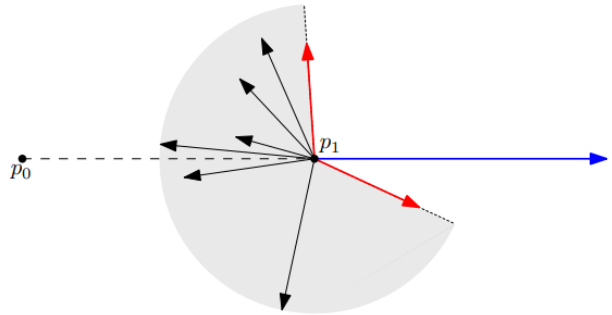


Figura 3.13 Ilustración de la definición de velocidades extremas tomado de [3]. La flecha azul es el vector -$p_1\vec{p}_0$. Las otras flechas conforman el conjunto H_1 de vectores de velocidad de V_1. Las flechas rojas son las velocidades extremas de V_1 con respecto a V_0.

Lema 2: Sea V_0 un vehículo estático con posición p_0, V_1 un vehículo con posición p_1 y H_1 el conjunto de posibles velocidades de V_1. Si V_1 colisiona con V_0 usando sus velocidades extremas, entonces V_1 colisiona con V_0 usando $v_{1,j}$ para todas las velocidades $v_{1,j} \in H_1$.

La definición 1 puede ampliarse cuando V_0 tiene un vector de velocidad sin velocidad cero de la siguiente manera:

Definición 2: Sea V_0 un vehículo con velocidad $v_{0,i}$. Sea V_1 un vehículo con el conjunto H_1 de posibles vectores de velocidad. Las velocidades extremas de V_1 con respecto a V_0 son los vectores $e_1 \in H_1$ y $e_2 \in H_1$ tales que e_1-$v_{0,i}$ y e_2-$v_{0,i}$ son velocidades extremas de V_1 con respecto a V_0 en la transformación $M(v_{0,i})$.

A partir de esta definición y del Lema 2 es fácil comprobar que:

Lema 3: Sea V_0 un vehículo con velocidad $v_{0,i}$. Sea V_1 un vehículo con el conjunto H_1 de posibles vectores de velocidad. Sean $e_1 \in H_1$ y $e_2 \in H_1$ las velocidades extremas de V_1 con respecto a V_0. Si V_1 y V_0 colisionan usando e_1 y $v_{0,i}$, respectivamente, entonces, para todo $v_{1,j} \in H_1$, V_1 y V_0 colisionan usando $v_{1,j}$ y $v_{0,i}$, respectivamente.

Lema 4: Sea V_0 un vehículo con posición p_0 con un conjunto de velocidades H_0 (todas las velocidades tienen el mismo módulo). Sea V_1 un vehículo con posición p_1 con un conjunto de velocidades H_1 (todas las velocidades tienen el mismo módulo). Si $s_0 \leq s_1$ entonces las velocidades extremas de V_1 con respecto a V_0 son invariantes independientemente de la velocidad $v_{0,i} \in H_0$ utilizada por V_0. Además, las velocidades extremas de V_1 con respecto a V_0 son vectores vecinos en H_1 del vector -$p_1\vec{p}_0$.

Teniendo en cuenta todo lo anterior, puede probarse el resultado fundamental en el que se basa el método:

Teorema 1: Sean V_0 y V_1 dos vehículos tales que $s_0 \leq s_1$. Sean e_1 y e_2 las velocidades extremas de V_1 con respecto a V_0. Sean $w_{1,1}$ (resp. $w_{1,2}$) y $w_{2,1}$ (resp. $w_{2,2}$) las velocidades extremas de V_0 con respecto a V_1 si V_1 utiliza e_1 (resp. e_2). Si V_0 y V_1 colisionan utilizando todas las siguientes combinaciones (e_1,$w_{1,1}$),(e_1,$w_{1,2}$),(e_2,$w_{2,1}$) y (e_2,$w_{2,2}$), entonces V_0 y V_1 colisionarán para todos los pares de velocidades $v_{0,i} \in H_0$ y $v_{1,j} \in H_1$, respectivamente.

El resultado anterior garantiza que basta comprobar la colisión tomando las velocidades extremas. El método comprobará así si hay solución y, en este caso buscará la velocidad más adecuada según la función objetivo.

La estrategia consiste en resolver el problema de planificación de trayectorias libres de colisión en ventanas de tiempo consecutivas, dando como resultado trayectorias pseudo-óptimas que minimizan la desviación de los UAVs con respecto a sus planes de vuelo originales. En cada ventana de tiempo, se asigna a cada agente una velocidad que minimiza el desplazamiento con respecto a la direccón del segmento que une la posición actual con el objetivo de forma que no se den colisiones.

La aplicación para tareas de filmación autónoma motivó el desarrollo de este algoritmo en [3], ya que los pequeños equipos de vehículos aéreos no tripulados deben evitar las colisiones mientras filman objetivos en movimiento. Por ello, se prefieren las trayectorias suaves sin aceleraciones y eficientes en energía.

3.2.1 Implementación

Se presentan aquí los códigos publicados en el repositorio de GitHub [3]. En este caso, fueron programados en Python. Dentro del repositorio, en el código principal experiments.py (Fig. 3.14) se definen todos los experimentos y se desarrolla para cada uno como sigue:

Primero se define y añade cada dron en una lista mediante la función `UAV()` definida en classes.py (Fig. 3.15), la cual posee como parámetros de entrada la posición inicial del dron (bidimensional), la velocidad preferida, el radio del dron, su dirección inicial, el punto objetivo (bidimensional), la distancia mínima del objetivo a la que se considera que el dron ha llegado a su destino y la máxima amplitud, que define el rango en el que el dron puede generar las k direcciones de velocidad para evitar la colisión (si es 60°, las k direcciones estarán en un rango entre 60° y -60° con respecto a la dirección que lleva el dron).

```python
import json
import numpy as np
import os

from classes import UAV
from utils import euclidian_distance, get_normalized_vector, plot_history
from math import *
from simulations import simulate

K_DIR = 7
TIMESTEP = 0.25
TIMERANGE = 10
SPEED = 1.5
SPEEDRATE = 20
RADIO = 1

MAXAMPLITUDE=radians(60)

def experiment1(k=K_DIR, speed = SPEED, radio = RADIO, timestep=TIMESTEP, ca_timerange=TIMERANGE):
    ''' 6 drones antipodales'''

    drone_positions = []
    for i in range(6):
        x = 10*cos(i*2*pi/6)
        y = 10*sin(i*2*pi/6)
        drone_positions.append((x,y))

    goals = [(-x, -y) for x,y in drone_positions]

    uavs = []
    for position, goal in zip(drone_positions, goals):

        direction = get_normalized_vector(np.array(goal)-np.array(position))
        uav = UAV(position, speed, radio, direction, goal, goal_distance=0.5, max_amp=MAXAMPLITUDE)
        uavs.append(uav)


    measures = simulate(uavs, k, ca_timerange, timestep)
    with open("results/experiments1.json", "w") as f:
        f.write(json.dumps(measures))

    plot_history(uavs, name="results/experiments1")
```

(a) Funcion experiment1().

```python
if __name__ == "__main__":



    experiment1()

    print("****Experimento 1 terminado****")
    print()
```

(b) Funcion main().

Figura 3.14 experiments.py para uno de los experimentos.

[3] *https://github.com/fabio-rodriguez/CollisionAvoidance*

```python
class UAV:

    def __init__(self, position, speed, radio, direction, goal_point, goal_distance=1, max_amp=radians(45)):
        self.position = np.array(position)
        self.initial_position = np.array(position)
        self.speed = speed
        self.radio = radio
        self.direction = np.array(get_normalized_vector(direction))
        self.goal_point = np.array(goal_point)
        self.is_in_goal = False
        self.max_amp = max_amp
        self.history = [self.position]
        ## Minimum distance to reach the goal
        self.goal_distance = goal_distance

    def fly(self, timestep):

        goal = Point(self.goal_point[0], self.goal_point[1])
        res = detect_collision_point(self.position, self.direction, goal,self.goal_distance+self.radio)
        try:
            p1, p2 = res
            p1 = np.array([p1.x, p1.y])
            p2 = np.array([p2.x, p2.y])
            d1, d2 = euclidian_distance(self.position, p1), euclidian_distance(self.position, p2)
            near = p1 if d1 < d2 else p2
        except:
            near = None if res == None else np.array([res.x, res.y])

        if not (near is None):
            time = time_from_displacement(self.direction, self.speed, euclidian_distance(self.position, near))
            if time < timestep:
                self.is_in_goal = True
                self.history.append(self.goal_point)
                return

        self.position = self.position +timestep*self.speed*self.direction
        self.history.append(self.position)

        if euclidian_distance(self.position, self.goal_point) < (self.radio+self.goal_distance):
            self.is_in_goal = True

    def generate_directions(self, k, stop_cost = None):

        if k%2!=0:
            k-=1
        goal_dir = get_normalized_vector(self.goal_point - self.position)

        d = []
        amp = 2*self.max_amp/k
        currentamp, _ = vector_2angles(self.direction)
        togoal = True
        for i in range(k+1):
            newamp = -self.max_amp + amp*i
            newdir = get_normalized_vector(angles_2vector(currentamp+newamp))
            cost = euclidian_distance(newdir, goal_dir)
            d.append((newdir, cost))
            if goal_dir[0] == newdir[0] and goal_dir[1] == newdir[1]:
                togoal = False

        goalamp, _ = vector_2angles(goal_dir)
        if togoal and angle_in_range(currentamp-self.max_amp, currentamp+self.max_amp, goalamp):

        return d

    def __str__(self) -> str:

        string = f''' DRONE PROPERTIES
                postion: {self.position}\n
                speed: {self.speed}
                radio: {self.radio}
                direction: {self.direction}
                goal: {self.goal_point}
                max_amp: {self.max_amp}
            '''

        return string

    def copy(self):
        uav = UAV((self.position[0], self.position[1]), self.speed, self.radio, (self.direction[0], self.direction[1]), self.goal_point)
        return uav
```

Figura 3.15 class UAV en classes.py.

Tras definir todos los drones del experimento, se ejecuta la función `simulate()`, definida en simulations.py (Fig. 3.16), la cual se encarga de hacer llegar a todos los drones a sus metas sin colisiones y devuelve el valor de las métricas calculadas que este trabajo necesita para llevar a cabo la comparativa de los dos algoritmos. La función `simulate()` toma como parámetros de entrada la lista de drones creada anteriormente, el número de maniobras a generar en caso de colisión (k), el rango de tiempo de detección de colisiones, el paso temporal del algoritmo (*time step*) y el número máximo de iteraciones antes de detenerse la simulación.

```python
import time
from utils import *
from solver import resolve_collision

def simulate(uavs, k, ca_timerange, timestep, max_iterations=10**5):

    flying_uavs = uavs[:]
    count = 0
    no_solution = False

    t = time.time()
    min_cost = 10**6
    max_iterations=10**3
    while flying_uavs and count<max_iterations:
        print("iteration", count)

        # Las direcciones actuales que se supone que no tienen colision
        current_dirs = [uav.direction for uav in flying_uavs]

        new_directions, cost = resolve_collision(flying_uavs, k, ca_timerange)

        if not new_directions:

            if list_collide(flying_uavs, current_dirs, ca_timerange):
                no_solution = True
                break

            for uav, d in zip(flying_uavs, current_dirs):
                uav.direction = d

        else:
            assert len(new_directions) == len(flying_uavs), "Directions and UAVs must have the same len"
            min_cost = min_cost if min_cost < cost else cost

            for uav, direction in zip(flying_uavs, new_directions):
                uav.direction = direction[0]


        aux = []
        for uav in flying_uavs:
            uav.fly(timestep)
            if not uav.is_in_goal:
                aux.append(uav)

        flying_uavs = aux[:]
        for i in range(len(flying_uavs)):
            for j in range(i+1, len(flying_uavs)):
                v = round(sum((flying_uavs[i].position - flying_uavs[j].position)**2)**0.5, 2)
                assert v >= (flying_uavs[i].radio + flying_uavs[j].radio), f"Drones TOO close: {v}"

        count+=1

    tf = time.time() - t

    measures = {i: calc_measures(uav) for i, uav in enumerate(uavs)}
    measures["waypoints"] = {i: {
        "X": [float(point[0]) for point in uav.history],
        "Y": [float(point[1]) for point in uav.history]
        } for i, uav in enumerate(uavs)}
    measures["total_time"] = tf
    measures["min_cost"] = min_cost

    if no_solution:
        return None

    return measures
```

Figura 3.16 simulations.py.

4 Comparación de los algoritmos

4.1 Escenarios

Se han diseñado cuatro escenarios para comparar las técnicas CA-*nk* y ORCA, junto con 100 escenarios aleatorios para una prueba ampliada. A continuación se describen los escenarios (para una visualización véase la Figura 5.1):

1. Cinco UAVs Antipodal (5UA): Cinco UAVs ubicados uniformemente en el perímetro de un círculo con un radio de 10 m. El objetivo de cada dron es su punto antipodal en el perímetro del círculo.
2. Seis UAVs Antipodal (6UA): El mismo escenario que 5AU, pero con seis drones
3. Un UAV Evitando Cinco (1UE5): Un dron se dirige hacia su punto de destino, mientras otros cinco drones se cruzan en su camino. Los cinco drones también se cruzan en su camino entre sí, de forma que sus trayectorias interfieran en la del dron principal.
4. Cinco UAVs Abajo hacia Arriba (5UAA): Cinco UAVs uniformemente situados en una línea recta cuyos objetivos se encuentran uniformemente situados en una línea recta paralela. Los objetivos están situados de tal manera que cada segmento que une origen y destino de los UAVs colisiona con al menos otros dos segmentos. La distancia entre líneas rectas paralelas es de 17 m.
5. Aleatorios: Un conjunto de 100 escenarios en los que los puntos inicial y objetivo de los UAVs se generan aleatoriamente dentro de un cuadrado de 50 m^2. Cada experimento contiene cinco drones. Para ello, se genera un algoritmo (programado en c++) que crea un número N (en este caso 100) de escenarios diferentes con posiciones iniciales y finales de forma aleatoria para 5 drones. De esta forma, se obtiene un número elevado de datos resultantes de cada escenario para comparar con ambos algoritmos de planificación.

   Se generan posiciones aleatorias iniciales y finales para cada dron con el uso de la función `rand()`, en un espacio de trabajo dentro de un rango de -25 a 25 metros. Todo ello teniendo en cuenta una distancia de seguridad mínima entre posiciones iniciales y entre posiciones finales de manera que los drones no colisionen entre ellos al intentar colocarse en sus posiciones asignadas.

   Las posiciones de cada experimento aleatorio quedan registradas en archivos de texto, los cuales son leídos posteriormente desde los códigos de simulación de los algoritmos, para darle las posiciones iniciales a cada dron.

Para poder visualizar cada experimento, se ha hecho uso de la herramienta Matlab. Se ha creado un sencillo programa que utiliza los archivos de texto (.txt) que guardan las trayectorias de cada dron y las representa gráficamente.

4.2 Medidas

Una vez diseñados los escenarios, se seleccionaron diferentes medidas para comparar el rendimiento de ambos algoritmos con respecto al consumo de energía y se procedió con las modificaciones necesarias en los códigos para obtenerlas. Factores cruciales en el consumo de energía son: el tiempo de vuelo, las variaciones de velocidad y los giros realizados por el dron. Por lo tanto, las medidas seleccionadas consideran estos factores para comparar las técnicas CA-*nk* y ORCA en cuanto al consumo de energía en el vuelo de los UAVs. Hay que notar que una fórmula cerrada de energía depende de las especificaciones del dron, por lo que hemos preferido tener en cuenta este conjunto

de medidas que influyen directamente en el gasto energético. Las medidas seleccionadas son las siguientes:

- Tiempo (s): El tiempo que tardan todos los UAVs en llegar al objetivo.
- Suma de aceleración (SA) (m/s^2): La suma de las variaciones de velocidad durante la navegación. La variación de la velocidad se calcula como el valor absoluto de la diferencia entre cada dos velocidades consecutivas.
- Desviación (m): Desviación del UAV de su trayectoria óptima (segmento que une los puntos origen y destino).
- Ángulo máximo (rad): El máximo ángulo de giro realizado durante la navegación.
- Ángulo medio (rad): La media de todos los giros realizados durante la navegación.
- Giros totales: El número de maniobras de giro totales utilizadas durante la navegación.

4.2.1 Cálculo de medidas

Tiempo

Para ORCA, puesto que el código permite definir el valor de *TimeStep*, se obtiene para cada dron el número de *Steps* que tarda en llegar al punto objetivo y finalmente se obtendrá el tiempo que ha empleado en recorrer la trayectoria mediante el producto de *TimeStep* por el número de *Steps*.

$$Tiempo = TimeStep \cdot Steps \tag{4.1}$$

Para CA-*nk*, se ha calculado mediante la fórmula:

$$Tiempo = d_{real} \cdot v, \tag{4.2}$$

siendo d_{real} la distancia real recorrida por cada dron (definida más abajo) y v la velocidad del dron (constante para CA-*nk*).

Suma de Aceleraciones (SA)

Para el cálculo de esta medida se realiza un sumatorio de la diferencia entre el módulo de la velocidad actual y la inmediatamente anterior, dividido por el *TimeStep* en cada iteración.

$$SA = \sum_{i=1}^{n} \frac{||\vec{v}_i^{\,act}| - |\vec{v}_i^{\,ant}||}{\tau} \tag{4.3}$$

Siendo $\vec{v}_i^{\,act}$ y $\vec{v}_i^{\,ant}$ las velocidades i-ésimas actual y anterior respectivamente y τ el *TimeStep*.

Desviación

Para el cálculo de la desviación (D) se obtiene en primer lugar la distancia real recorrida por cada dron. Para ello se ha realizado en el código un sumatorio de los módulos de todos los vectores de posición que forman la trayectoria.

$$\vec{v} = (v_1, v_2) \tag{4.4}$$

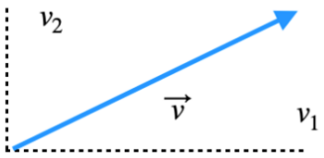


Figura 4.1 Vector de posición entre dos puntos de una trayectoria.

$$|\vec{v}| = \sqrt{v_1^2 + v_2^2} \tag{4.5}$$

$$d_{real} = \sum_{i=1}^{n} |\vec{v}_i| \tag{4.6}$$

La desviación para cada vehículo será la diferencia entre la distancia real recorrida y la mínima distancia entre los puntos inicial (A) y objetivo (B).

$$d_{min} = \sqrt{(x_B - x_A)^2 + (y_B - y_A)^2} \tag{4.7}$$

$$D = d_{real} - d_{min} \tag{4.8}$$

Siendo d_{real} la distancia real y d_{min} la distancia mínima entre el punto inicial y final.

Ángulo Máximo

Para obtener el ángulo máximo de todos los giros que realiza el dron durante la trayectoria sobre su eje central, se hace una comparación consecutiva entre ellos en cada *Step*, eligiendo cada vez el mayor.

Ángulo Medio

Para obtener el ángulo de giro medio, primero se calculan los ángulos de giro. Para ello se obtienen los ángulos que forman cada dos vectores consecutivos, obtenidos de los puntos de la trayectoria de cada dron. Estos se han calculado utilizando la fórmula del coseno por el cálculo vectorial, siendo $\vec{p}$ y $\vec{q}$ dos vectores consecutivos de la trayectoria y |p| y |q| sus módulos respectivamente.

$$\vec{p} \cdot \vec{q} = |\vec{p}| \cdot |\vec{q}| \cdot cos(\alpha) \tag{4.9}$$

$$cos(\alpha) = |\frac{\vec{p} \cdot \vec{q}}{|\vec{p}| \cdot |\vec{q}|}| \tag{4.10}$$

De esta manera, tomando el valor absoluto, obtenemos el coseno de Alpha (ángulo formado por los dos vectores) y calculando el arcocoseno obtenemos finalmente el ángulo de giro.

$$arcos(cos(\alpha)) = angulo(^\circ) \tag{4.11}$$

Para el cálculo de ángulo medio de giro, se realiza un sumatorio de todos los águlos de giros que realiza el dron durante la trayectoria sobre su eje central y se divide entre el número de giros totales.

$$\bar{\alpha} = \frac{\sum_{i=1}^{n} \alpha_i}{N} \tag{4.12}$$

Siendo α_i el ángulo i-ésimo de la trayectoria y N la cantidad de giros totales.

Giros Totales

Para calcular el número de giros totales, se incrementa un contador cada vez que detecte que el ángulo girado es distinto de cero.

4.3 Parámetros

En esta sección se describen los parámetros utilizados, similares para ambos algoritmos (Tabla 4.1):

- Radio: El radio del dron.
- Horizonte de tiempo: La cantidad mínima de tiempo durante la cual la nueva velocidad calculada de un UAV está a salvo de la colisión con respecto a otros UAVs.
- Paso de tiempo: El paso en segundos entre las iteraciones de los planificadores (*TimeStep*).
- Rango de velocidades: El intervalo de velocidades de los vehículos aéreos no tripulados.

- Ángulo Máximo: El ángulo máximo de giro para un UAV (sólo para CA-*nk*)
- Número de ángulos (k): El número de ángulos generados cuando se produce una colisión (sólo para CA-*nk*)
- Distancia vecinos: La distancia máxima (de punto central a punto central de cada dron) a otros UAVs que un nuevo UAV tiene en cuenta en la navegación (sólo para ORCA)
- Número vecinos: El número máximo de otros UAV que un nuevo UAV tiene en cuenta en la navegación (sólo para ORCA)

Tabla 4.1 Parámetros para CA-*nk* y ORCA usadas en la comparación.

Parámetros	Valor	
	$CA-nk$	$ORCA$
Radio	1 m	1 m
Horizonte de tiempo	10 s	10 s
TimeStep	{0.25 s, 2 s}	{0.25 s, 2 s}
Rango de velocidades	$[1.5\ m/s, 1.5\ m/s]$	$(0\ m/s, 1.5\ m/s]$
Ángulo máximo	60°	-
Número de ángulos (K)	8	-
Distancia vecinos	-	15 m
Número vecinos	-	10 UAVs

Para estudiar las diferencias de ambas técnicas, todos los experimentos se realizaron para dos TimeSteps diferentes, 0,25s y 2s. La velocidad de los drones en ORCA está en el rango (0m/s, 1,5m/s], mientras que en CA-*nk*, los UAVs vuelan a velocidad constante utilizando la máxima velocidad de 1,5m/s. Teniendo ocho maniobras de giro uniformemente seleccionadas y 60º máximo de giro, los ángulos de giro que se han considerado para el método CA-*nk* son $\{-60.00^\circ, -42.86^\circ, -25.71^\circ, -8.57^\circ, 8.57^\circ, 25.71^\circ, 42.86^\circ, 60^\circ\}$.

4.4 Resultados

Tabla 4.2 Tabla de valores medios y desviaciones de las medidas de comparación de Tiempo, SA, y Desviación. Radio = 1 m y TimeStep 0.25 s.

Experimentos	Tiempo (s)		SA (m/s)		Desviación (m)	
	$CA-nk$	$ORCA$	$CA-nk$	$ORCA$	$CA-nk$	$ORCA$
5UA	13.65 ± 0.02	593894.25 ± 0.00	0.00 ± 0.00	64.81 ± 7.71	0.48 ± 0.03	1.68 ± 0.24
6UA	15.16 ± 1.38	303222.75 ± 0.00	0.00 ± 0.00	62.89 ± 12.09	2.74 ± 2.08	2.04 ± 0.23
1UE5	13.43 ± 3.59	24.75 ± 0.00	0.00 ± 0.00	19.78 ± 1.91	1.98 ± 2.84	0.31 ± 0.42
5UAA	14.18 ± 1.83	26.50 ± 0.00	0.00 ± 0.00	15.27 ± 2.44	0.34 ± 0.43	0.52 ± 0.62
Aleatorios	17.56 ± 8.24	40.04 ± 7.48	0.00 ± 0.00	17.80 ± 2.87	0.10 ± 0.52	0.27 ± 1.47

Tabla 4.3 Tabla de valores medios y desviaciones de las medidas de comparación Giros Totales, Ángulo Máximo y Ángulo medio. Radio = 1 m y Timestep 0.25 s.

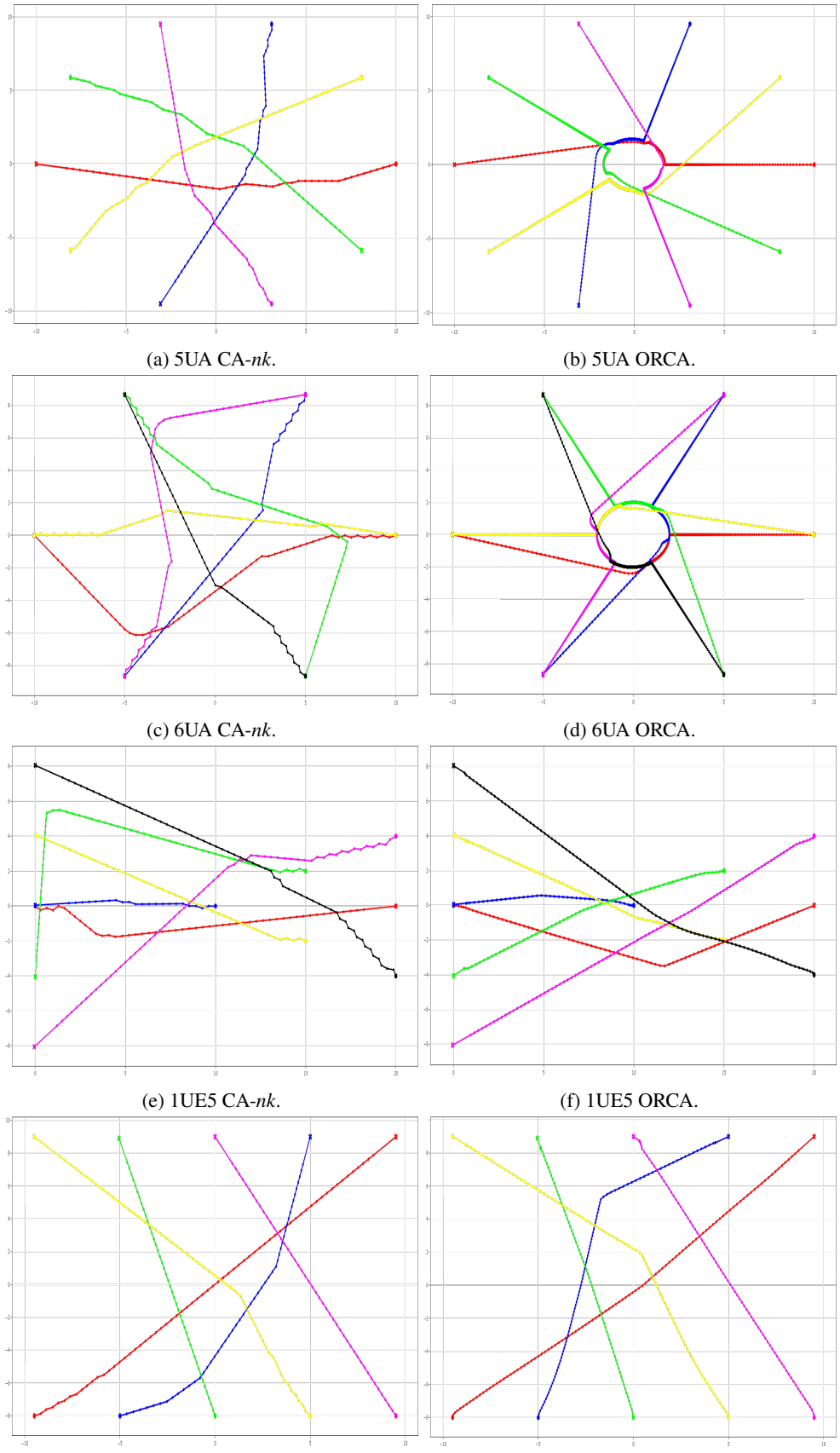
Experimentos	Giros Totales		Ángulo Máximo (rad)		Ángulo medio (rad)	
	$CA-nk$	$ORCA$	$CA-nk$	$ORCA$	$CA-nk$	$ORCA$
5UA	9.40 ± 1.85	170.80 ± 65.46	0.47: 0.41 ± 0.05	1.43: 0.49 ± 0.47	0.33 ± 0.04	0.01 ± 0.01
6UA	30.33 ± 2.56	166.17 ± 84.25	1.05: 0.70 ± 0.00	1.21: 0.48 ± 0.34	0.63 ± 0.05	0.30 ± 0.12
1UE5	11.17 ± 5.46	67.17 ± 20.14	1.05: 0.93 ± 0.16	0.85: 0.48 ± 0.18	0.61 ± 0.11	0.03 ± 0.01
5UAA	4.60 ± 4.59	84.60 ± 14.11	0.38: 0.22 ± 0.18	0.93: 0.52 ± 0.21	0.20 ± 0.17	0.02 ± 0.00
Aleatorios	2.70 ± 7.58	80.23 ± 42.10	1,05: 0.11 ± 0.25	1.56 : 0.29 ± 0.37	0.08 ± 0.18	0.02 ± 0.06

Tabla 4.4 Tabla de valores medios y desviaciones de las medidas de comparación de Tiempo, SA, y Desviación. Radio = 1 m y TimeStep 2 s.

Experimentos	Tiempo (s)		SA (m/s)		Desviación (m)	
	$CA-nk$	$ORCA$	$CA-nk$	$ORCA$	$CA-nk$	$ORCA$
5UA	13.75 ± 0.07	2979.20 ± 0.98	0.00 ± 0.00	64.81 ± 7.71	0.63 ± 0.10	1.89 ± 0.48
6UA	13.79 ± 0.12	76400.67 ± 1.49	0.00 ± 0.00	1.20 ± 0.20	0.69 ± 0.18	1.94 ± 0.62
1UE5	13.42 ± 3.56	26.33 ± 5.47	0.00 ± 0.00	0.77 ± 0.13	1.95 ± 3.12	0.38 ± 0.23
5UAA	14.20 ± 1.83	26.00 ± 4.56	0.00 ± 0.00	0.31 ± 0.14	0.36 ± 0.52	0.31 ± 0.18
Aleatorios	17.57 ± 8.24	43.56 ± 8.52	0.00 ± 0.00	0.65 ± 0.37	0.11 ± 0.63	1.33 ± 2.93

Tabla 4.5 Tabla de valores medios y desviaciones de las medidas de comparación Giros Totales, Ángulo Máximo y Ángulo medio. Radio = 1 m y Timestep 2 s.

Experimentos	Giros totales		Ángulo Máximo (rad)		Ángulo medio (rad)	
	$CA-nk$	$ORCA$	$CA-nk$	$ORCA$	$CA-nk$	$ORCA$
5UA	2.80 ± 0.40	652.40 ± 69.64	0.53: 0.51 ± 0.02	1.11: 0.73 ± 0.21	0.43 ± 0.01	0.01 ± 0.00
6UA	2.83 ± 0.90	46.33 ± 3.73	0.69: 0.41 ± 0.13	1.02: 0.89 ± 0.14	0.33 ± 0.06	0.05 ± 0.01
1UE5	1.50 ± 0.96	10.83 ± 2.41	1.05: 0.64 ± 0.32	0.44: 0.34 ± 0.08	0.53 ± 0.27	0.10 ± 0.02
5UAA	1.40 ± 1.36	10.60 ± 2.15	0.39: 0.22 ± 0.18	0.53: 0.34 ± 0.15	0.20 ± 0.17	0.06 ± 0.02
Aleatorios	0.37 ± 1.06	11.59 ± 5.96	0.88: 0.08 ± 0.21	1.56 : 0.33 ± 0.43	0.06 ± 0.16	0.07 ± 0.13



(a) 5UA CA-*nk*.

(b) 5UA ORCA.

(c) 6UA CA-*nk*.

(d) 6UA ORCA.

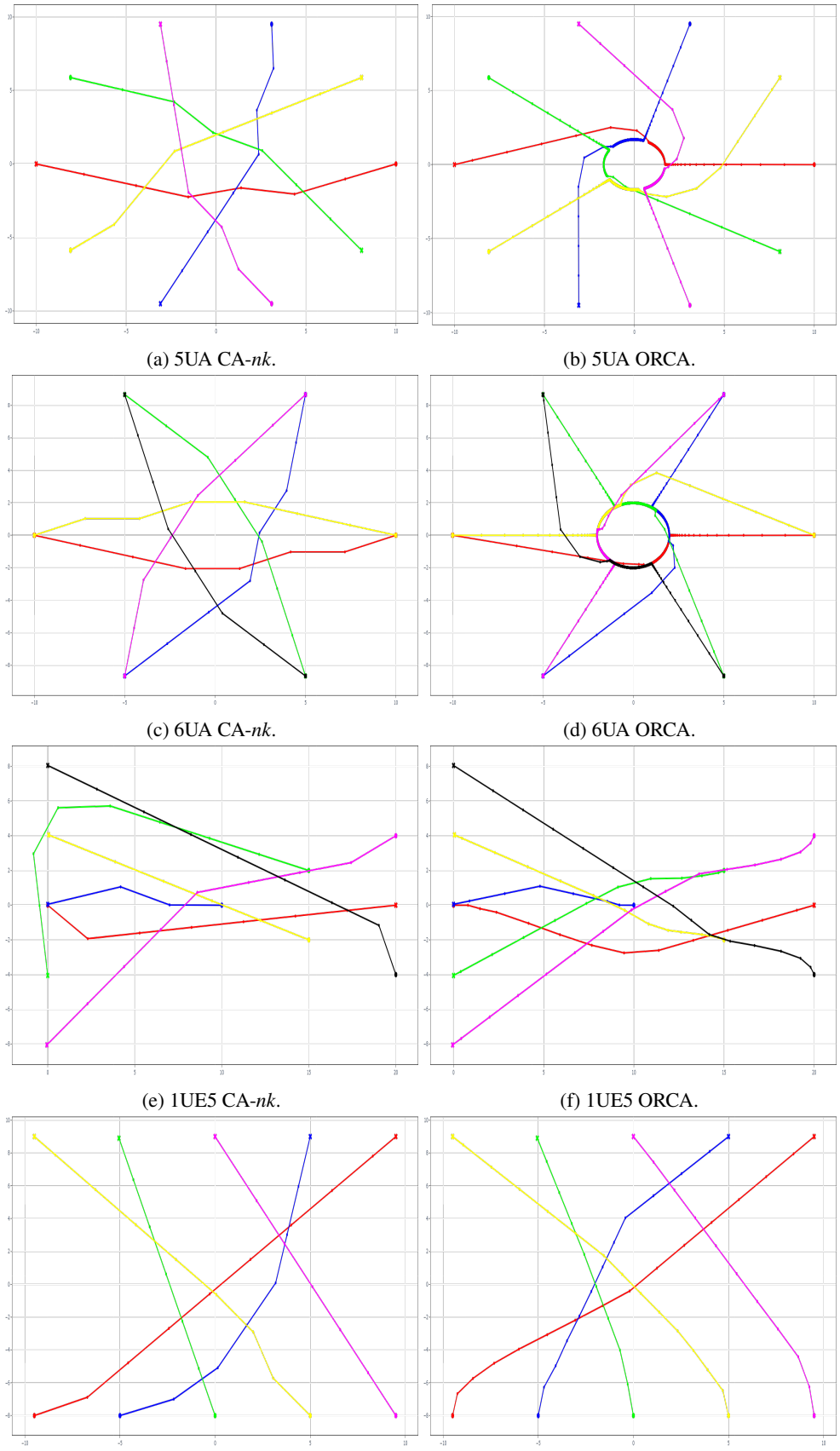
(e) 1UE5 CA-*nk*.

(f) 1UE5 ORCA.

(g) 5UAA CA-*nk*.

(h) 5UAA ORCA.

Figura 4.2 Imagenes de las trayectorias de los UAVs en escenarios específicos usando un TimeStep de 0.25 segundos.



(a) 5UA CA-*nk*.

(b) 5UA ORCA.

(c) 6UA CA-*nk*.

(d) 6UA ORCA.

(e) 1UE5 CA-*nk*.

(f) 1UE5 ORCA.

(g) 5UAA CA-*nk*.

(h) 5UAA ORCA.

Figura 4.3 Imagenes de las trayectorias de los UAVs en escenarios específicos usando un TimeStep de 2 segundos.

4.5 Análisis

Las figuras 4.2 y 4.3 muestran la navegación de todos los UAVs en los escenarios específicos para las técnicas CA-*nk* y ORCA, utilizando un TimeStep de 0,25 s y 2 s, respectivamente. Las figuras muestran que, en general, los UAVs en ORCA tienen trayectorias más suaves que en CA-*nk*. Sin embargo, las trayectorias de los UAVs usando CA-*nk* tienen menos puntos de giro *waypoints* que las de ORCA. Esto es debido a la decisión que toman los UAVs en ORCA de frenar ante una posible colisión. Las tablas 4.2 y 4.3 muestran los resultados de la evaluación de las medidas utilizando ambas técnicas en cada escenario utilizando el TimeStep 0,25 s; así mismo, las Tablas 4.4 y 4.5 muestran los resultados utilizando el TimeStep 2 s. Las tablas muestran la media y la desviación estándar de los resultados alcanzados por todos los drones en los experimentos.

Las tablas 4.2 y 4.4 para 0,25 s y 2 s, respectivamente, muestran que, en cuanto al tiempo se refiere, CA-*nk* tiene mejores resultados que ORCA, ya que este algoritmo, y en especial en los experimentos antipodales, requiere una gran cantidad de tiempo para realizar la tarea en comparación con CA-*nk*. Esto se produce, como se dijo anteriormente, por las respuestas de ORCA de reducir su velocidad hasta casi 0 cuando detecta posibles colisiones, por lo que aumenta considerablemente la demora en llegar al punto objetivo. Por su lado, CA-*nk* mantiene la velocidad constante y no se ve afectado su tiempo de finalización.

Por otra parte, puesto que en CA-*nk* los UAVs navegan a velocidad constante, la aceleración de los drones es nula, y por tanto, Suma de Aceleraciones es 0. Por el contrario, en ORCA, observamos que se consumiría una cantidad de energía por el cúmulo de aceleraciónes que con CA-*nk* se ahorraría.

Para la desviación podemos observar que la diferencia de los resultados obtenidos para ambos algoritmos no es significativa.

Las tablas 4.3 y 4.5 para 0,25 s y 2 s, respectivamente, muestran los resultados de las medidas relacionadas con las maniobras de giro. En cuanto al número de giros totales que se realizan, se ve un claro resultado favorable para CA-*nk*. El número elevado de giros en ORCA podría ser debido en gran parte a la tardanza de los drones para realizar las trayectorias. En el tiempo que están frenando y calculando nuevas trayectorias libres de colisiones se producen giros que a la larga acaban sumando una cantidad bastante mayor que en CA-*nk*, obteniendo éste un tiempo de finalización de tarea inferior que ORCA. Esto se ve mayormente reflejado para los experimentos antipodales, que son los que más difieren en tiempo para los dos algoritmos. Sin embargo, para los demás experimentos en los que los resultados de tiempo se asemejan más, ORCA sigue teniendo peores resultados en cantidad de giros, por lo que para esta medida es menos eficiente en el consumo de energía.

Por otra parte, CA-*nk* tiene límite en el máximo ángulo que puede girar. Esto ayuda a obtener ángulos máximos menores que en ORCA para el cuál no hay límite en el ángulo de giro. Sin embargo, vemos que para el experimento 1UE5 en concreto, se obtienen resultados mejores en ORCA. Este escenario está diseñado para ver la capacidad del algoritmo al enfrentarse a muchas posibles colisiones seguidas una detrás de otra, enfrentando un dron a cinco drones. Por lo que podemos decir que en experimentos en los que se da una situación crítica, CA-*nk*, al ir a velocidad constante, se ve obligado a dar cambios de direcciones más bruscos con giros de ángulos mayores; por su parte, ORCA, al tener la opción de frenar, evita en mayor medida los giros bruscos, dando como resultado trayectorias más suaves. Sin embargo, para este experimento, CA-*nk* sigue tomando menos tiempo pararealizar la tarea y realiza menos giros que ORCA, siendo más eficiente para nuestro objetivo.

Aunque hemos observado que, en general, ORCA da resultados con el ángulo de giro máximo mayor que en CA-*nk*, en la medida Ángulo Medio vemos que ORCA da como resultados valores más pequeños que CA-*nk*. Esto es debido a que los UAVs en CA-*nk* realizan pocos giros con un valor mínimo de 8,57º, mientras que ORCA realiza bastantes más giros, pero la mayoría de ellos con un ángulo de giro muy pequeño.

4.6 Conclusión y futuras líneas de investigación

Como resultado de la comparación realizada en este trabajo, podemos concluir decir que CA-*nk* da buenos resultados cuando el objetivo es realizar planificación de trayectorias de manera eficiente con respecto a la energía requerida, el tiempo en ejecutar la tarea y el tiempo de computación necesario para dar las soluciones. Todo esto, asumiendo un número reducido de drones.

Por el contrario, ORCA sería buena elección si el objetivo es evitar colisiones por encima de la eficiencia de la tarea y de la cantidad de drones que coexistan en el escenario.

En este trabajo se ha considerado un escenario en dos dimensiones y sin incluir obstáculos en los experimentos.

Como trabajo futuro se puede plantear realizar los experimentos en un entorno de tres dimensiones así como añadir obstáculos en los escenarios. Para ello, habría que adaptar los algoritmos (que serían menos eficientes) y, para el caso de CA-*nk*, habría que precalcular las trayectorias óptimas que unen origen y destino teniendo en cuanta los obstáculos existentes. Finalmente, una comparación con otras técnicas de planificación supondría una línea de extensión del trabajo interesante.

Bibliografía

[1] A. Alexopoulos, A. Kandil, P. Orzechowski, and E. Badreddin. A comparative study of collision avoidance techniques for unmanned aerial vehicles. In *2013 IEEE international conference on systems, man, and cybernetics*, pages 1969–1974. IEEE, 2013.

[2] E. Boivin, A. Desbiens, and E. Gagnon. Uav collision avoidance using cooperative predictive control. In *2008 16th Mediterranean Conference on Control and Automation*, pages 682–688. IEEE, 2008.

[3] L. E. Caraballo, J. M. Díaz-Báñez, R. Fabila-Monroy, A. Fernández, and F. Rodríguez. Collision avoidance for aerial vehicles using turn-angles. In *XVIII Spanish Meeting on Computational Geometry*. University of Girona, 2019.

[4] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *The international journal of robotics research*, 17(7):760–772, 1998.

[5] Q. Galvane, J. Fleureau, F.-L. Tariolle, and P. Guillotel. Automated cinematography with unmanned aerial vehicles. *arXiv preprint arXiv:1712.04353*, 2017.

[6] Y. Ham, K. K. Han, J. J. Lin, and M. Golparvar-Fard. Visual monitoring of civil infrastructure systems via camera-equipped unmanned aerial vehicles (uavs): a review of related works. *Visualization in Engineering*, 4(1), 2016.

[7] D. Hein, T. Kraft, J. Brauchle, and R. Berger. Integrated uav-based real-time mapping for security applications. *ISPRS International Journal of Geo-Information*, 8(5):219, 2019.

[8] K.-Y. Kim, J.-W. Park, and M.-J. Tahk. Uav collision avoidance using probabilistic method in 3-d. In *2007 International Conference on Control, Automation and Systems*, pages 826–829. IEEE, 2007.

[9] E. . LABORAL. Aplicaciones profesionales dron, 2021. Aplicaciones.

[10] U. R. Mogili and B. Deepak. Review on application of drone systems in precision agriculture. *Procedia computer science*, 133:502–509, 2018.

[11] Y. Naidoo, R. Stopforth, and G. Bright. Development of an uav for search & rescue applications. In *IEEE Africon'11*, pages 1–6. IEEE, 2011.

[12] J. Sun, J. Tang, and S. Lao. Collision avoidance for cooperative uavs with optimized artificial potential field algorithm. *IEEE Access*, 5:18382–18390, 2017.

[13] J. van den Berg. Rvo2 library documentation, 2008-16. Jur van den Berg, Stephen J. Guy, Jamie Snape, Ming C. Lin, and Dinesh Manocha.

[14] J. Van Den Berg, S. J. Guy, M. Lin, and D. Manocha. Reciprocal n-body collision avoidance. In *Robotics research*, pages 3–19. Springer, 2011.

[15] Wikipedia. Github — wikipedia, la enciclopedia libre, 2021.

[16] Wikipedia. Matlab — wikipedia, la enciclopedia libre, 2021.

[17] Wikipedia. Microsoft visual studio — wikipedia, la enciclopedia libre, 2021. [Internet].

[18] Wikipedia. Python — wikipedia, la enciclopedia libre, 2021.

[19] Wikipedia. C++ — wikipedia, la enciclopedia libre, 2022. [Internet; descargado 13-septiembre-2022].

[20] Wikipedia. Visual studio code — wikipedia, la enciclopedia libre, 2022. [Internet; descargado 13-septiembre-2022].

[21] J. N. Yasin, S. A. Mohamed, M.-H. Haghbayan, J. Heikkonen, H. Tenhunen, and J. Plosila. Unmanned aerial vehicles (uavs): Collision avoidance systems and approaches. *IEEE Access*, 8:105139–105155, 2020.

Apéndice: Herramientas Utilizadas

1 Visual Studio

Microsoft Visual Studio es un entorno de desarrollo integrado para macOS y Windows. Este entorno es compatible con distintos lenguajes de programación como Visual Basic .NET, Java, Python, C++, etc. Para este trabajo se ha utilizado Python y C++.

Figura 1 Logotipo de Visual Studio.

La versión utilizada para este trabajo ha sido Visual Studio 2017 [17]. En este entorno se desarrolla el algoritmo ORCA.

2 Visual Studio Code

Visual Studio Code es un editor de código fuente desarrollado por Microsoft para Windows, Linux, macOS y Web. Incluye soporte para la depuración, control integrado de Git, resaltado de sintaxis, finalización inteligente de código, fragmentos y refactorización de código [20].

En este entorno se desarrolla el algoritmo CA-*nk*.

Figura 2 Logotipo de Visual Studio Code.

3 Python

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta parcialmente la orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma [18].

En este lenguaje se desarrolla el algoritmo *CA-nk*.

Figura 3 Logotipo de Python.

4 C++

C++ es un lenguaje de programación diseñado en 1979 por Bjarne Stroustrup. La intención de su creación fue extender al lenguaje de programación C mecanismos que permiten la manipulación de objetos. En ese sentido, desde el punto de vista de los lenguajes orientados a objetos, C++ es un lenguaje híbrido [19].

En este lenguaje se desarrolla el algoritmo *ORCA*.

Figura 4 Logotipo de C++.

5 Matlab

Es un software matemático con entorno de desarrollo integrado (IDE), el cual tiene un lenguaje de programación propio, llamado Lenguaje M. Este software es multiplataforma pudiendo utilizarse en Unix, Windows (en este caso) y Apple Mac Os X. [16]

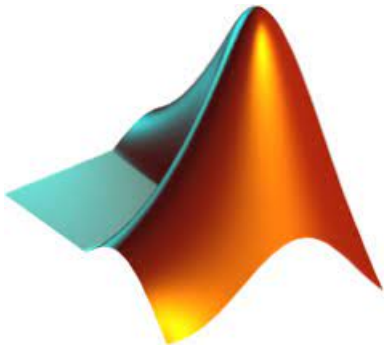


Figura 5 Logotipo de Matlab.

Sus principales funciones son:

- Manipulación de matrices
- Representación de datos y funciones
- Implementacion de algoritmos
- Creación de interfaces de
- Comunicación con programas en otros lenguajes y con otros dispositivos Hardware

En este trabajo, esta herramienta será utilizada únicamente para ejecutar la segunda de sus funciones.

6 GitHub

GitHub es una plataforma de desarrollo colaborativo para alojar proyectos utilizando el sistema de control de versiones Git. Se utiliza principalmente para la creación de código fuente de programas de ordenador. El software que opera GitHub fue escrito en Ruby on Rails. Desde enero de 2010, GitHub opera bajo el nombre de GitHub, Inc. Anteriormente era conocida como Logical Awesome LLC. El código de los proyectos alojados en GitHub se almacena típicamente de forma pública. [15].

La realización de este trabajo ha partido de la base de obtener el conjunto de códigos que ejecutan los algoritmos ORCA y CA-*nk* de este repositorio.

En el caso del algoritmo ORCA, se ha hecho un previo análisis para interpretar y entender el ejemplo **Circle.cpp** que es en el que se han realizado las modificaciones necesarias para la construcción de los diferentes escenarios y por consiguiente, la toma de datos.

Figura 6 Logotipo de GitHub.