Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Algoritmo descentralizado para evitación de colisiones de múltiples UAVs en 3D

Autor: Alfonso Alcántara Marín
Tutor: Jesús Capitán Fernández

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2017

Trabajo Fin de Grado
Grado en Ingeniería de las Tecnologías Industriales

Algoritmo descentralizado para evitación de colisiones de múltiples UAVs en 3D

Autor:
Alfonso Ángel Alcántara Marín

Tutor:
Jesús Capitán Fernández
Profesor Ayudante Doctor

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2017

Trabajo Fin de Grado: Algoritmo descentralizado para evitación de colisiones de múltiples UAVs en 3D

Autor: Alfonso Alcántara Marín

Tutor: Jesús Capitán Fernández

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2017

El Secretario del Tribunal

Agradecimientos

En primer lugar, me gustaría agradecer a mis padres. Sin su apoyo llegar hasta aquí no hubiera sido posible.
A mi tutor, Jesús, por proporcionarme la ayuda y herramientas necesarias para realizar este trabajo.
A mis amigos más cercanos, por estar ahí en los momentos difíciles.

Alfonso Alcántara Marín
Sevilla, 2017

Resumen

Este trabajo presenta un algoritmo de evitación de colisiones para un sistema de múltiples vehículos aéreos en 3D que es ejecutado en tiempo real. El algoritmo propuesto está basado en un algoritmo de evitación de obstáculos en dos dimensiones llamado SWAP.

Se ha implementado una extensión del algoritmo SWAP para funcionar en tres dimensiones y en vehículos aéreos no tripulados, bajo el ecosistema ROS. De manera que pueda ser ejecutado de forma descentralizada para evitar obstáculos estáticos y dinámicos.

La implementación del algoritmo, realizada en el entorno ROS, ha sido simulada en GAZEBO, software que permite realizar simulaciones realistas. Siendo simuladas las situaciones con mayor carácter conflictivo, en entornos realistas y analizando las variables que permiten mostrar el funcionamiento del algoritmo.

Abstract

This work presents a collision avoidance algorithm for multiple aerial vehicle systems to be applied in real time. The proposed algorithm is based on SWAP: Safety-Enhanced Avoidance Policy, a descentralized algorithm which works on 2D. This thesis presents an extension of SWAP which works on 3D. This algorithm has been implemented considering dynamic constraints of the UAV model and static obstacles, so it can be used in realistic environments. The algorithm works with the ROS framework and has been tested using GAZEBO, which is a software that makes simulations in realistic environment.

Índice

Índice de Figuras

Notación

UAV	Unmanned Aircraft Vehicle. Vehículo aéreo no tripulado
RCA	Reciprocal Collision Avoidance. Evitación recíproca de colisión
ORCA	Optimal Reciprocal Collision Avoidance. Evitación óptima y reciproca de colisión.
RRT	Rapidly exploring Random Trees. Algoritmo de exploración aleatoria.
SWAP	Descentralized safe conflict resolution for multiple robots in dense scenarios. Resolución descentralizada de conflictos para múltiples robots en escenarios densos.
MBZIRC	International Robotics Challenge. Concurso internacional de robótica
PX4	Autopiloto pixhawk
SITL	Software In The Loop Simulación que integra las ecuaciones que rigen la dinámica de un vehículo y su entorno
ROS	Robotic Operating System
GAZEBO	Software de simulación realista

1 INTRODUCCIÓN

1.1 Motivación

El uso de Vehículos Aéreos No Tripulados (UAV, por sus siglas en inglés, Unnmaned Aerial Vehicle) ha aumentado exponencialmente en los últimos años. Dadas sus características de bajo coste, autonomía, y especialmente, la práctica ausencia de riesgo humano; ofrecen un gran potencial para aplicaciones, tanto civiles como militares. Recientemente, gracias a los avances tecnológicos en dinámicas, navegación y sensores, un gran número de aplicaciones civiles han salido a la luz. Aplicaciones de vigilancia, fotografía, búsqueda y rescate, agricultura, topología, etc. En definitiva, los UAVs están suponiendo una alternativa para disminuir el coste y el riesgo de numerosas necesidades.

Ilustración 1 – Sistema multi-UAVs en incendios

Ilustración 2 - Prototipo de sistema multi-UAV para mensajería

Las misiones cooperativas realizadas por sistemas de varios vehículos han sido muy estudiadas en los últimos años, dadas las ventajas que presentan frente a las realizadas por un solo robot. La complejidad de tareas tales como el montaje de partes de un objeto, transporte de materiales industriales, detección de incendios, monitorización, necesitan del empleo de grupos de robots que ofrecen mayor versatilidad, rapidez y eficiencia. La comunidad científica sabe que el empleo de grupos de UAVs para realizar determinadas aplicaciones presenta numerosas ventajas.

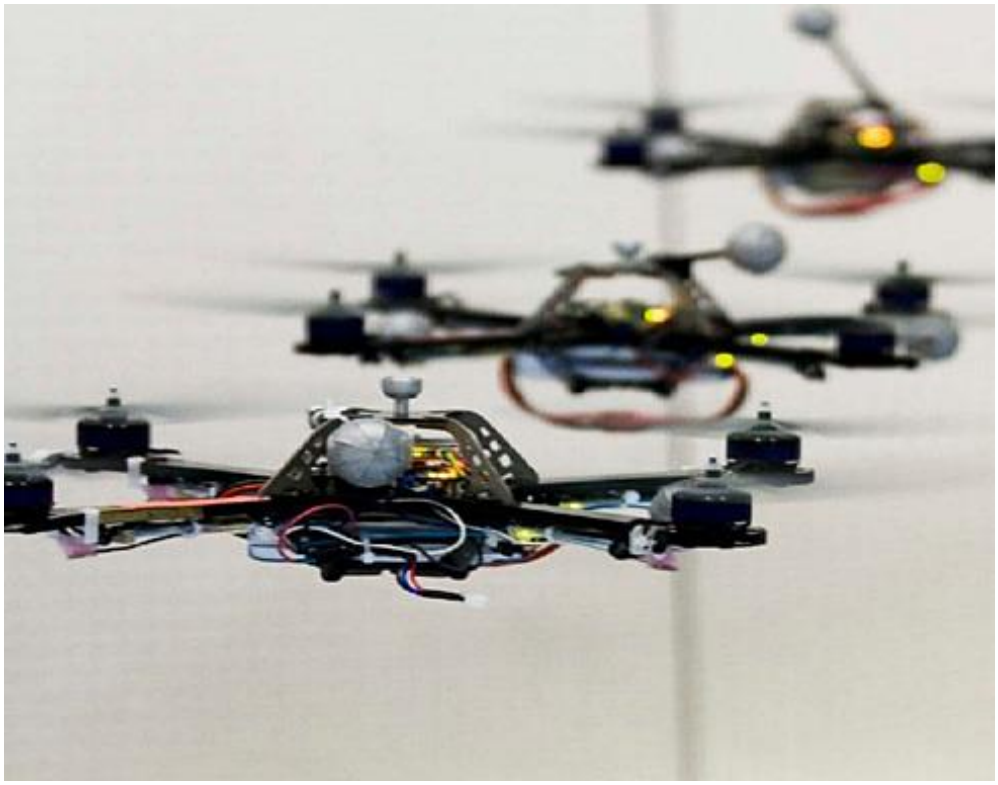

Ilustración 3 - Sistema de UAVs volando en formación

La coordinación y evitación de colisiones juega un papel fundamental en este tipo de aplicaciones donde el número de robots implicados aumenta. En esos casos, los algoritmos de planificación de trayectorias son usados para calcular las trayectorias que permiten realizar la misión. Y de forma paralela, algoritmos reactivos calcularán trayectorias en tiempo real que permita evitar un obstáculo cuando una posible colisión sea detectada.

La capacidad de evitación de obstáculos será una característica fundamental para que los UAVs puedan obtener permisos para realizar diferentes actividades, asegurando un vuelo seguro, aparte del que asegura la vigilancia humana. En un futuro, un UAV requerirá de un sofisticado sistema de evitación para que pueda volar, junto a otros, en escenarios con una gran densidad de obstáculos fijos.

El problema de evitación de obstáculos ha sido ampliamente estudiado en robótica. Pero este tipo de vehículos presentan nuevas necesidades: como actuación en tiempo real, espacio tridimensional, baja capacidad de carga o robustez sensorial y de comunicaciones, que suponen un nuevo desafío para la comunidad científica.

En este trabajo se pretende analizar el problema de evitación de obstáculos e implementar un algoritmo que, ejecutado de una manera descentralizada, permita a un UAV evitar obstáculos fijos y móviles en el espacio tridimensional.

Dado el riesgo, coste y tiempo que supone testear las nuevas aplicaciones desarrolladas en este tipo de vehículos en el campo de vuelo, son necesarias herramientas de simulación que permitan observar las variables críticas del desarrollo de este tipo de aplicaciones en entornos muy realistas. De esta forma, se minimiza el riesgo, coste y tiempo necesario para poner a punto las nuevas aplicaciones desarrolladas. En la ilustración 4 se puede ver una herramienta que permite realizar simulaciones en entornos realistas ofreciendo

la posibilidad de utilizar el mismo hardware que se utilizaría en el vehículo aéreo.

Ilustración 4 - Ejemplo de simulación realista

En este trabajo se utiliza una herramienta de simulación realista tanto para implementar el algoritmo como para comprobar su funcionamiento en las situaciones más conflictivas.

1.2 Objetivo

El desarrollo e implementación de un algoritmo de evitación de obstáculos no podría darse sin un estudio o análisis de los trabajos de evitación de obstáculos para sistemas de UAVs existentes en la literatura. Para ello se realiza un estudio del estado del arte de este problema, expuesto en el capítulo 2.

El algoritmo que se presenta en este trabajo es una extensión de un algoritmo previo denominado SWAP. Este algoritmo previo fue desarrollado para robots móviles y entornos de dos dimensiones. Para entender el algoritmo propuesto es necesario conocer el algoritmo a partir del cual surge, por ello, este algoritmo previo es detallado en el capítulo 3. Otro conocimiento preliminar necesario para la posterior implementación del algoritmo es el funcionamiento de ROS. La estructura y el funcionamiento general de ROS se desarrollan en el capítulo 4.

El objetivo principal de este trabajo es realizar una extensión del algoritmo SWAP para funcionar en tres dimensiones y en vehículos aéreos no tripulados. Los detalles de esta extensión y su funcionamiento en vehículos aéreos no tripulados son desarrollados en el capítulo 5.

No solo se pretende extender de manera teórica el algoritmo de evitación de colisiones, el segundo objetivo es implementarlo para funcionar en UAVs. Este algoritmo será programado en C++ bajo el ecosistema ROS. Aparte de las numerosas herramientas que ROS aporta, se aprovecha la estructura de funcionamiento de ROS y todas las facilidades para el desarrollo de aplicaciones de robótica que nos ofrece este entorno.

Para comprobar que el algoritmo sigue los comportamientos adecuados, tanto en situaciones habituales como en situaciones críticas, se realizan simulaciones en un software realista. De estas simulaciones se extraen variables de interés para sacar las conclusiones apropiadas. Todas las simulaciones se desarrollan detalladamente en el capítulo 6.

Finalmente, las conclusiones provenientes del desarrollo y análisis del algoritmo de evitación de colisiones

implementado en este trabajo, son detalladas en el capítulo 7. De la misma manera que son analizados los posibles trabajos futuros con este algoritmo y sus limitaciones.

2 ESTADO DEL ARTE

El problema de detección y evitación de colisiones en un sistema de vehículos aéreos no tripulados ha sido ampliamente estudiado y es todavía un desafío para la comunidad científica. Se trata de un problema complejo que debe solucionarse en tiempo real, por tanto, debe resolverse eficientemente. Además, es un proceso crítico para la seguridad del sistema. De hecho, una de las exigencias para que se puedan introducir UAVs autónomos en el espacio aéreo no restringido es que dispongan de un sistema de "sense and avoid" que obtenga prestaciones similares o mejores a las proporcionadas por un piloto.

La comunidad científica comenzó a estudiar el problema de evitación de obstáculos para aplicarlo en vehículos móviles terrestres. La evitación de obstáculos para vehículos móviles ha sido un área de investigación muy activa en el campo de la robótica y multitud de trabajos fueron expuestos para trabajar en dos dimensiones. La principal limitación de estos trabajos es su aplicación a dos dimensiones y el coste computacional que requieren. Un UAV requerirá un algoritmo que pueda ejecutarse en tres dimensiones y que pueda ser ejecutado con bajas necesidades computacionales, dadas las necesidades de tiempo real que este tipo de vehículos plantean.

Existe una gran variedad de métodos para solucionar el problema de evitación de obstáculos en UAVs. Algunas de las técnicas empleadas para mantener un UAV seguro de colisión incluyen modelos de optimización geométrica, enfoques probabilísticos, enfoques mediante consenso, algoritmos genéticos, filtros de kalman, control predictivo etc. [1]. Cada método tiene sus ventajas y características particulares. Todos tratan de mantener una mínima distancia entre el UAV que ejecuta el algoritmo y el obstáculo. Sin embargo, muchos de esos estudios solo se basan en un escenario en el que solo existen dos UAVs. Este trabajo se centra en escenarios donde existen más de dos UAVs. Este tipo de sistemas requieren de algoritmos de evitación que escalen adecuadamente con el número de vehículos implicados en el sistema.

De forma general, podemos decir que hay dos enfoques mediante el cual se pueden encarar la resolución de conflictos entre más de dos UAVS: de una forma descentralizada o de una forma centralizada.

En el enfoque centralizado todas las situaciones son simultáneamente tratadas y el conflicto es resuelto de una vez. Cada UAV negocia con todos los UAVs de su entorno para tomar una decisión. En algunas ocasiones, los grupos de UAVs se agrupan para realizar el movimiento de evitación. Este tipo de algoritmos ofrecen una solución más robusta que los métodos descentralizados, pero requieren un mayor coste computacional.

En el caso de los algoritmos descentralizados, los conflictos son tratados de manera independiente por cada vehículo, tomando cada UAV una decisión según su entorno sin negociar con otros vehículos. Este tipo de enfoque es mucho más simple que el enfoque centralizado, aunque en ocasiones puede llevar a situaciones menos seguras. Otra característica atractiva de este enfoque es su coste computacional.

En este apartado se va a describir, de forma general, algunos de los enfoques existentes en la literatura para resolver el problema de evitación de obstáculos en sistemas con múltiples vehículos aéreos.

2.1 Planificación de trayectorias

La mayoría de los trabajos publicados en la literatura sobre planificación de trayectorias no son buenos candidatos para el tipo de problema desarrollado en este trabajo, a causa del bajo coste computacional que requiere una solución en tiempo real para un sistema multi robot. Los métodos de planificación de trayectorias incluyen algoritmos RRT (Rapidly exploring Random Trees), optimización de partículas en enjambre, algoritmos de evolución, grafos como A* y D*, etc.

Existen otros métodos, basados en la planificación de trayectorias que, de manera centralizada, ofrecen una solución. Se basan en predecir una colisión mediante la intersección de las trayectorias de los UAVs implicados en el sistema. Como se puede ver en la ilustración 5, los UAV definen una trayectoria dependiendo del objetivo y, comparando estas, predicen una trayectoria que hará evitar los obstáculos en el camino.

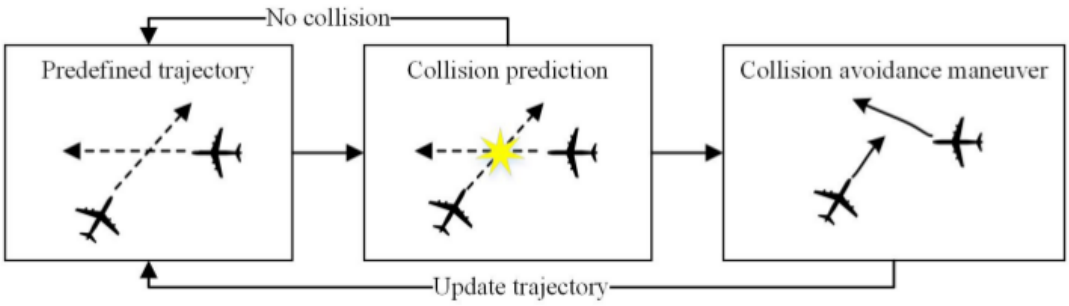

Ilustración 5 - Esquema de evitación en algoritmos de planificación

En [2] se incluye este procedimiento. Una vez detectada la colisión de trayectorias, se ejecuta un algoritmo ROA (en inglés, Rolling Optimization Algorithm) para actualizar y mejorar los puntos de referencia iniciales y minimizar la distancia total. Este algoritmo se basa en una función de coste en la que cada punto tiene un peso, siendo mayor el peso cuanto más alejado esté de la trayectoria.

Este tipo de algoritmos tiene los inconvenientes que anteriormente se mencionan de los enfoques centralizados. Requieren de un sistema, que de manera centralizada, ejecute los algoritmos de evitación y de planificación de todos los vehículos involucrados, con el coste computacional que conlleva.

2.2 Métodos potenciales

En estos métodos, los UAVs son tratados como partículas cargadas con la misma polaridad, por lo tanto, se repelen unas a otras. El destino de un UAV es modelado como una carga de signo opuesto que atrae al UAV haciéndole navegar hasta su destino. Por lo tanto, el UAV se moverá en la dirección de la fuerza resultante que crea estos campos potenciales. Las fuerzas van multiplicadas por constantes para ajustar el comportamiento del vehículo.

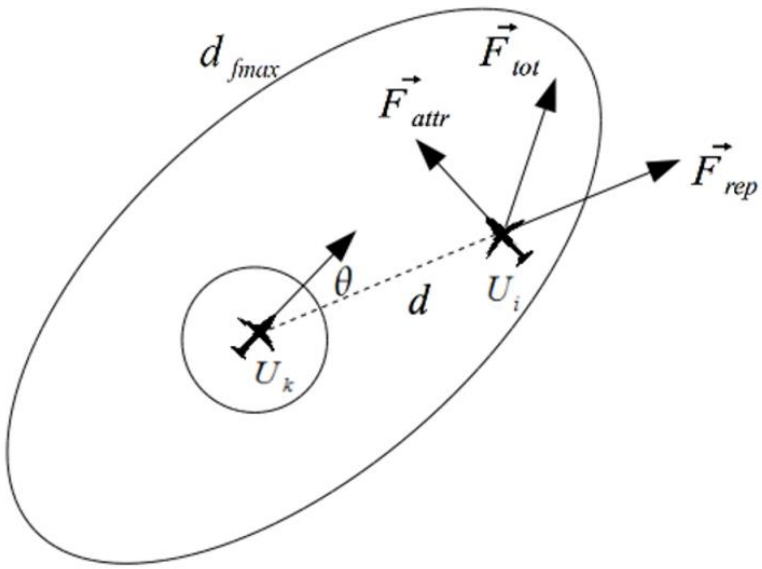

Ilustración 6 - Fuerzas en métodos potenciales

En [3] se presenta un algoritmo optimizado, basado en los campos potenciales, para operaciones multi-UAV en el espacio 3D. Gestiona tanto el problema de planificación de trayectoria como el de evitación de obstáculos en el mismo algoritmo y de manera centralizada, requiriendo un coste computacional alto. Ofrece

una solución exitosa del problema en 3D y con múltiples vehículos aunque no está simulado en un software realista.

2.3 Teoría de juegos

Existen algoritmos que se basan en predecir la trayectoria de determinados obstáculos en un futuro próximo dependiendo de la decisión o acción que este realice. Esto lo realizan siguiendo reglas de teoría de juegos.

En la teoría de juegos clásica se supone que los agentes se pueden anticipar correctamente a lo que sus oponentes harán aprendiendo patrones de comportamiento.

En [4] se usa satisfactoriamente la teoría de juegos para solucionar el problema de la evitación de colisiones con múltiples UAVs. Presentan simulaciones en las que utilizan escenarios con una gran densidad de vehículos aéreos para testear el algoritmo propuesto.

Aunque funciona de manera descentralizada, el gran problema de este tipo de algoritmos es que son muy costosos computacionalmente y no están simulados en entornos realistas.

2.4 ORCA

Los algoritmos RCA (en inglés, Reciprocal Collision Avoidance) consideran múltiples vehículos navegando en un escenario común. En este tipo de algoritmos la evitación se realiza por parejas y cada vehículo tiene la mitad de responsabilidad de evitar al otro. Este tipo de algoritmos está formulado en el espacio de velocidades. Un ejemplo se encuentra en [5].

El algoritmo ORCA de sus siglas en inglés (Optimal Reciprocal Collision Avoidance) [6] está basado en mejorar el comportamiento de los algoritmos RCA. Este algoritmo elige la velocidad óptima del campo de velocidades que permite evitar el obstáculo.

Se desarrolla una versión en 3D de este algoritmo en [7]. Y en [8] se desarrolla una implementación de la versión 3D-ORCA para sistemas en ROS. En [9] se implementa la versión 3D de ORCA en ROS para un sistema de multiples vehículos. En este último trabajo se ofrece una solución descentralizada, en tiempo real y 3D, simulada en escenarios realistas y con un coste computacional medio.

2.5 Cono de colisión

El enfoque del cono de colisión para el problema de evitación de colisiones toma un sistema de dos agentes y lo convierte en un sistema compuesto de un solo agente y un obstáculo estático. El cono de colisión es una función de coste que es ejecutada y usada para determinar si dos UAVs colisionarán.

Considerando los dos UAVs de la ilustración 7; si el vehículo B es tratado como un obstáculo fijo, A puede ser tratado como un agente dinámico cuyo movimiento será definido por la velocidad relativa entre A y B, V_{rel}. V_{ref} es tangencial al área circular de B. El rango de las velocidades relativas entre los dos vectores V_{rel} y V'_{rel} es conocido como cono de colisión. El cono de colisión será una función de coste que representará el riesgo de un UAV a efectuar una colisión.

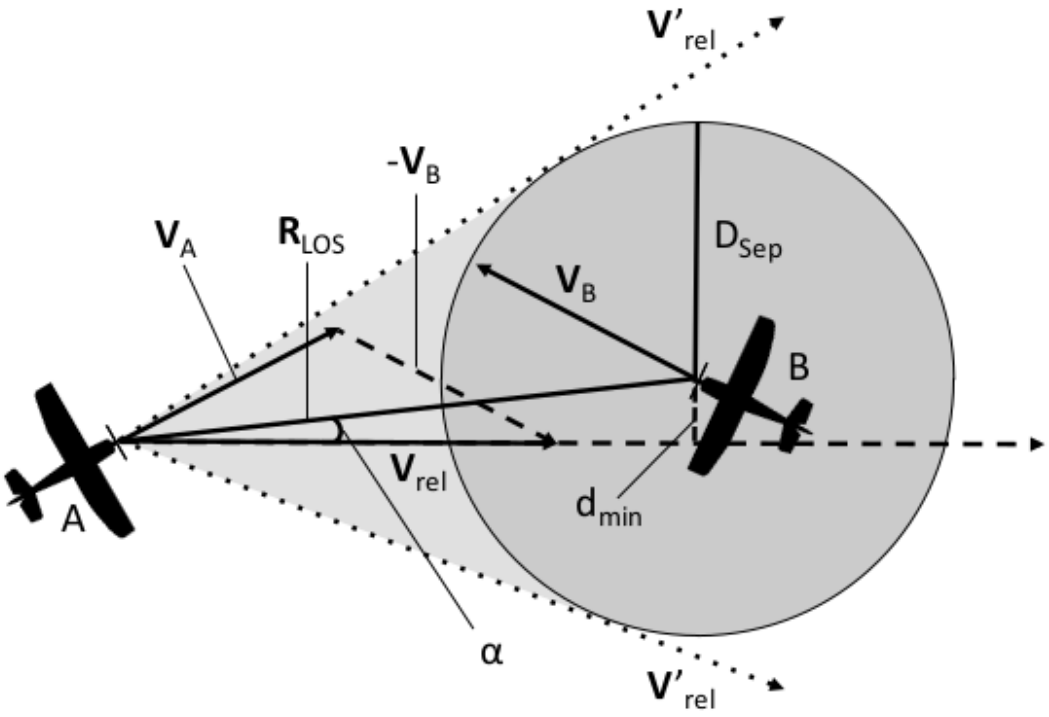

Ilustración 7 - Cono de colisión

En [10], el concepto de cono de colisión es extendido para trabajar con múltiples vehículos aéreos. Los UAVs son agrupados por su peligro relativo a otros. El cono de colisión se forma tomando como base la circunferencia que engloba a varios vehículos conflictivos. El cono de colisión entre un UAV y sus agresores es usado para crear una función de coste, que a su vez es usada para estimar la seguridad de la posición del UAV en el siguiente lapso de tiempo.

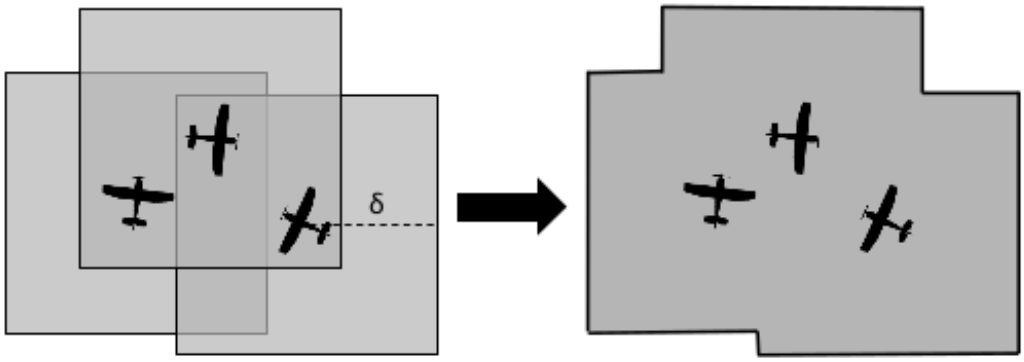

Ilustración 8 – Agrupación de obstáculos en cono de colisión

Este estudio [10] se basa en un sistema de múltiples vehículos aéreos, dinámicos, simulado con 8, 16 y 32 UAV. El problema de este algoritmo es que pierde eficiencia cuando el número de UAVs implicados en el sistema crece. Mantener una solución óptima hace que crezca su coste computacional. Otro inconveniente es que funciona en dos dimensiones. Es un algoritmo hibrido entre global y descentralizado, utilizando ambos enfoques.

2.6 Propuesta

Otra característica importante a señalar del estado del arte de la evitación de colisiones, es la forma en la que se simula un algoritmo de estas características. No es común, en la literatura de evitación de colisiones, realizar una simulación realista del algoritmo propuesto. Ni en un entorno de simulación realista ni en el campo de vuelo. Dada las restricciones de seguridad del sector aéreo, una resolución de un conflicto como este no puede ser aceptada y desarrollada sin un riguroso testeo y evaluación. La simulación en el campo de vuelo es costosa, por lo que utilizar un software que permita la simulación en entornos realistas y con las dinámicas propias del vehículo cobra especial importancia.

Tras una revisión de la literatura existente, se puede concluir que este tipo de vehículos necesitan una solución descentralizada que tenga pocas dependencias de comunicación, su coste computacional sea bajo y no crezca exponencialmente con el número de vehículos implicados en el sistema. Estas son algunas de las características que se persiguen en este trabajo. A su vez, se emplea un método de simulación realista para testear el algoritmo propuesto.

3 PRELIMINARES

Este trabajo parte de un algoritmo de evitación de obstáculos para sistemas multi-robots. El trabajo preliminar fue publicado en un estudio llamado "Descentralized safe conflict resolution for multiple robots in dense scenarios" por Eduardo Ferrera, Jesús Capitán, Ángel Castaño y Pedro J. Marrón [11]. En este estudio, se propone un algoritmo que consigue un comportamiento seguro de los robots que lo ejecutan. En escenarios con una alta densidad de vehículos y/o obstáculos fijos. Ejecutado, de forma reactiva y descentralizada, resuelve conflictos en escenarios 2D con múltiples vehículos móviles.

El algoritmo previo se denomina SWAP. El cual prioriza la seguridad del vehículo manteniendo una mínima distancia entre el robot y el obstáculo. Es capaz de llegar a su destino reaccionando a los obstáculos detectados usando información local.

Con la ayuda de sensores de rango o, cuando sea posible, recibiendo las posiciones de los robots ubicados a su alrededor, se construye una representación polar de su entorno y se define un sector angular de orientaciones que permitirían al vehículo detectar los conflictos que sean necesarios para navegar de forma segura.

El algoritmo utiliza determinadas reglas – explicadas en los siguientes apartados - que hacen que el vehículo, al encontrarse con un obstáculo, lo rodee en el sentido contrario a las agujas del reloj, hasta que el camino en dirección a su destino esté libre de colisiones.

Esto se realizará gracias a una serie de comportamientos que irán alternándose dependiendo de la situación en la que se encuentre el robot. Estos comportamientos serán gobernados por una máquina de estados que estará ejecutándose en cada robot de forma descentralizada.

3.1. Detección de conflictos

Cada robot usa una representación polar de su alrededor gracias a la lectura de sensores de rango. Este sensor de rango proporciona la distancia y orientación a la que se encuentra el obstáculo respecto del robot, medidas que el robot emplea para realizar un diagrama polar de su entorno. A su vez, el robot es capaz de localizarse a sí mismo.

Por lo tanto, siguiendo una serie de reglas, el robot buscará la orientación necesaria para evitar el obstáculo, detectado mediante el sensor de rango y almacenado en su diagrama polar. A continuación, se describirá cómo trata cada robot los obstáculos representados.

Cada robot se representa mediante una Región de Seguridad. Esta Región de Seguridad es un circulo de radio r_{sr} que está centrado en el medio del robot y que contiene todo su contorno. Para tener en cuenta la distancia que necesita un robot para pararse totalmente, el error del sistema de posicionamiento y otros errores como pueden ser los retrasos de comunicación, se define la región de inflado. La región de inflado es un circulo de radio r_{ir} que resulta de aumentar el radio de la región de seguridad de los obstáculos almacenados en el diagrama polar. A continuación, se definen los parámetros utilizados para construir la región de inflado:

- $\mathbf{r_{sr}}$ la región de seguridad de un robot es un circulo de radio r_{sr} con centro en su zona media y que contiene todo el contorno del robot
- $\mathbf{d_{br}}$ representa la distancia que un robot necesita para parar, en el peor de los casos, cuando navegue a v_{max}
- $\mathbf{e_r}$ considera el máximo error en el sistema de posicionamiento
- $\mathbf{\Omega}$ un parámetro que representa un offset diseñado para tener en cuenta otros posibles errores, retrasos en comunicación, etc.

La región de inflado se construye para permitir detectar un obstáculo con la suficiente antelación como para

que pueda evitarse en el peor de los casos; es decir, cuando dos robots naveguen en sentido contrario a velocidad máxima. En la imagen se pueden ver los parámetros que formarán la región de inflado para que dos robots puedan parar, en el peor de los casos, sin colisionar. Por lo tanto, se define el radio de la región de inflado como $r_{ir} = d_{br} + e_r + \Omega$

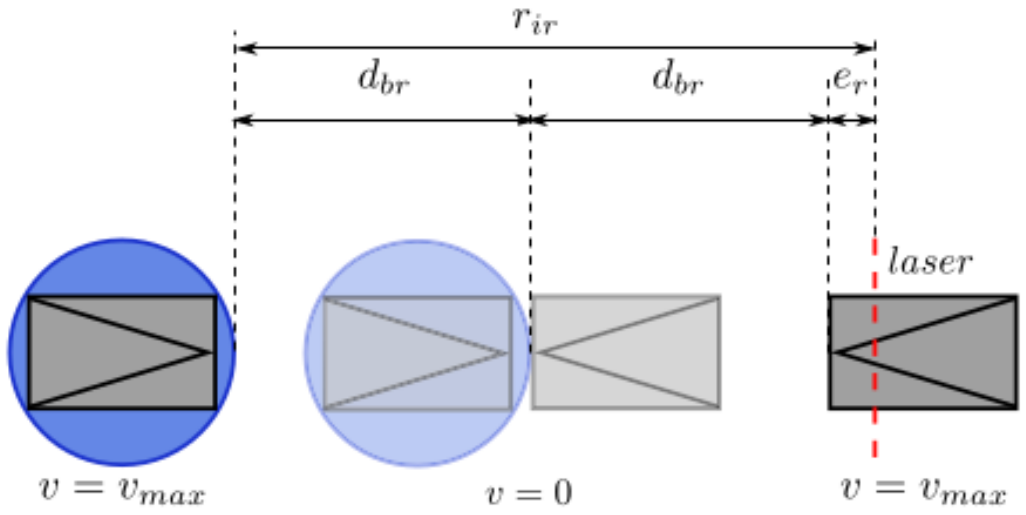

Ilustración 9 - Radio de inflado en situación de frenado

Existirá una colisión, si la región de seguridad del robot que está ejecutando el algoritmo SWAP, intersecta con alguna de las regiones infladas en el diagrama polar. La figura de la derecha de la ilustración 10, representa el diagrama polar de un robot que se encuentra en la situación representada a su izquierda. En azul, la región de seguridad, y en naranja, la región de inflado. La intersección entre la región de inflado y la región de seguridad (cruces en rojo), representa los puntos conflictivos.

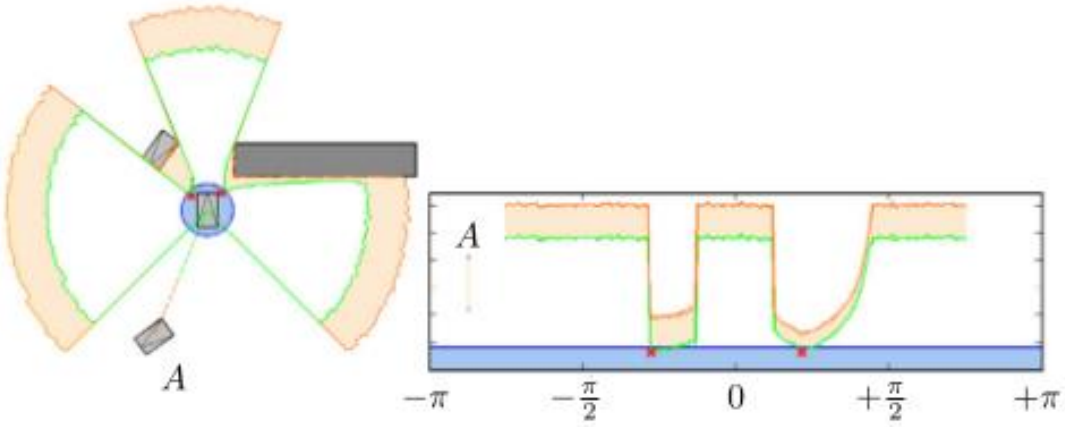

Ilustración 10 - Diagrama polar en situación conflictiva

3.1 Direcciones prohibidas

En esta sección se explicará cómo analizan los robots sus conflictos para definir una dirección que asegure una navegación libre de colisión.

Para cada obstáculo, el robot define el ángulo de dirección φ_k de ese conflicto como la orientación del punto C_k que se encuentra más cerca del robot. En la ilustración 11 se muestra un ejemplo de como el robot define el sector de direcciones prohibidas entre ($\varphi - \pi/2$, $\varphi + \pi/2$), el cual significa que navegar hacia una dirección fuera

de ese intervalo sería navegar en una dirección segura. De la misma manera, en la ilustración 11, se describe geométricamente cómo un robot calcula la región de direcciones prohibidas para evitarla y navegar en una dirección segura. La dirección de la línea perpendicular a la dirección del conflicto hará que el robot pueda navegar de forma que evite los obstáculos a su alrededor.

Como se puede ver en la ilustración 11, existen dos ángulos que permiten evitar el obstáculo. Solo uno evitará el obstáculo en el sentido contrario de las agujas del reloj. Siguiendo esta misma regla, todos los robots se evitarán unos a otros en el sentido contrario de las agujas del reloj.

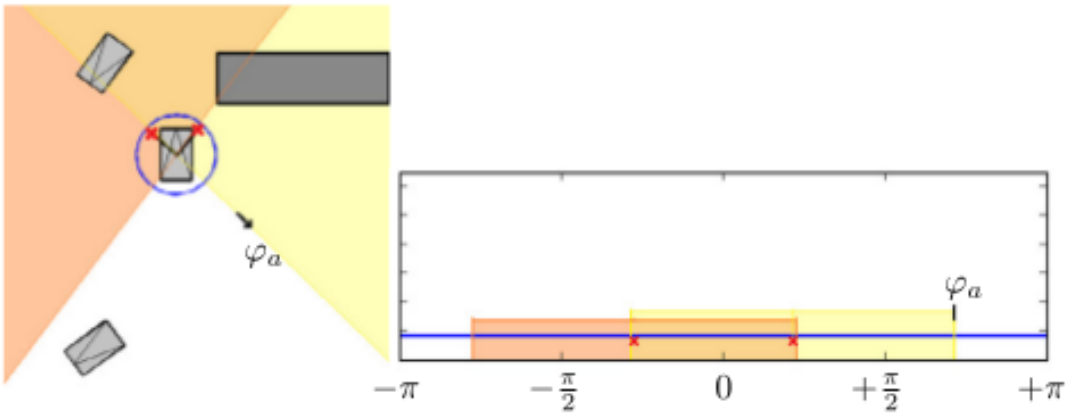

Ilustración 11 - Ángulos de evitación de región conflictiva

3.2 Máquina de estados algoritmo SWAP

Los robots intentan alcanzar su punto de destino de forma segura, resolviendo los conflictos que encuentran en su camino. Esto se realiza siguiendo una serie de comportamientos gobernados por una máquina de estados. Cada robot ejecuta localmente y de una forma descentralizada la misma máquina de estados, la cual está a cargo de comandar diferentes velocidades y orientaciones. Así pues, todos los robots siguen el mismo comportamiento que permite solventar las situaciones conflictivas. A continuación se describirán los diferentes estados.

- **Free**: este es el comportamiento normal en navegación donde el robot puede ir libremente hacia su destino. El robot está en el estado *FREE* si no existen conflictos a su alrededor o si existen conflictos pero se dan dos condiciones: (1) φ_g no pertenece al sector prohibido y (2) el punto de destino no está detrás del robot. En este estado navega a toda velocidad a su destino, comandando una velocidad $v_{ref} = v_{max}$.
- **Blocked**: el robot entra en este estado cuando detecta conflictos a su alrededor y no hay ninguna orientación de evitación fuera del sector prohibido. Esto significa que el robot está bloqueado y rodeado de obstáculos; por tanto, el robot parará ($v_{ref}=0$) y esperará hasta que algún obstáculo se mueva y deje una orientación φ_a que permita escapar.
- **Rencontre**: El robot entra en este estado cuando detecta un conflicto que debe y puede evitar antes de seguir navegando en la dirección de su destino. En esta situación, el robot necesita resolver el conflicto escapando con la orientación del ángulo de evitación; por lo tanto, comienza parando para prevenir colisiones ($v_{ref}=0$) y al mismo tiempo que intenta alcanzar la orientación de evitación $\varphi_{ref} = \varphi_a$. Una vez está orientado correctamente, la maniobra de evitación está a salvo y el robot puede cambiar al estado Rendezvous.
- **Rendezvous**: Este estado ocurre cuando el robot está realizando una maniobra para evitar un obstáculo. El robot está orientado hacia φ_a y puede rodear el obstáculo en el sentido contrario de las agujas del reloj. Durante esta operación, una velocidad lineal $v_{ref} = v_a < v_{max}$ es comandada por razones de seguridad, manteniéndose φ_a.

Se pueden dar cualquier transición entre estados, pero existen algunas que se darán en la mayoría de condiciones. Un robot navegará normalmente hacia su destino en el estado *FREE*, y entonces, encontrará un conflicto. Si el conflicto está en frente del robot (no es ignorable), este cambiará al estado *RENCONTRE* para posteriormente ir a *RENDEZVOUS*, rodeando el obstáculo en el sentido contrario a las agujas del reloj hasta volver al estado *FREE*. Si dos robots se encuentran el uno al otro y realizan el movimiento de evitación simultáneamente, ellos intercambiarán sus posiciones. Si más de dos robots convergen en el mismo punto o llegan a estar rodeados por otros, el robot que esté bloqueado pasará al estado BLOCKED hasta que deje de estarlo.

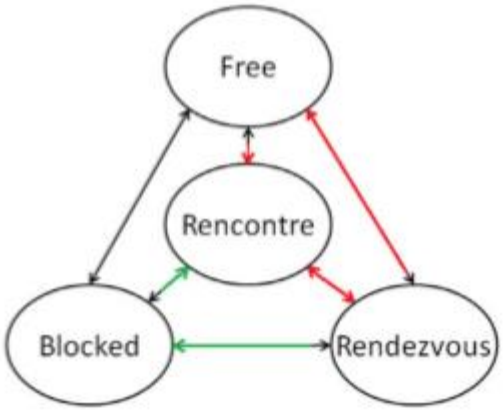

Ilustración 12 - Transiciones entre estados

3.3 Descripción del algoritmo

El algoritmo 1 describe todos los pasos de una iteración SWAP que cada robot del sistema ejecutará de una forma descentralizada. Primero, se calcula la región de inflado de los datos leídos por el sensor de rango. Si el robot recibe información de otros robots, esta se incorpora al diagrama polar. Luego, se calcula el ángulo de evitación y la máquina de estados selecciona el siguiente estado comandando la velocidad y orientación requerida.

Algorithm 1 SWAP($p, \theta, \mathcal{R}$)

$r_{ir} \leftarrow 2d_{br} + e_r + \gamma$
IR $\leftarrow$ `inflate`$(\mathcal{R}, r_{ir})$
`sendInfo`(p)
$\mathcal{P} \leftarrow$ `receiveInfo`$()$
if $\mathcal{P} \neq \emptyset$ **then**
  $r_{ir} \leftarrow 2d_{br} + 2e_l + r_{sr} + \gamma$
  IR $\leftarrow$ IR $\cup$ `inflate`$(\mathcal{P}, r_{ir})$
end if
$\mathcal{C} \leftarrow$ IR $\cap$ SR
for all $C_k \in \mathcal{C}$ **do**
  $\varphi_k \leftarrow$ `closestPointAngle`(C_k)
end for
$\varphi_a \leftarrow$ `computeAvoidanceAngle`$(\{\varphi_k\}_{k=1,\ldots,|\mathcal{C}|})$
$v_{ref}, \theta_{ref} \leftarrow$ FSM$(\varphi_a, \varphi_g, \theta)$
return v_{ref}, θ_{ref}

Ilustración 13 - Algoritmo SWAP

3.4 Características del algoritmo

A continuación se van a nombrar algunas de las características que hacen este algoritmo un buen trabajo preliminar de cara a desarrollar un algoritmo descentralizado en 3D para múltiples vehículos aéreos.

- Es un algoritmo descentralizado. Se busca que el coste computacional necesario para ejecutar el algoritmo en cada robot escale adecuadamente con el número de robots implicados en el sistema.
- Algoritmo reactivo. Es importante que el algoritmo esté preparado para funcionar con información local.
- Es robusto frente a incertidumbres en los sensores, dinámicas de segundo orden y ante dificultad en las comunicaciones, proporcionando soluciones seguras.
- Funciona ante mínimos locales.

Algunas características deben ser desarrolladas en la nueva implementación. Tales como:

- Funcionamiento en 3D. Debe tenerse en cuenta que existen obstáculos en el espacio tridimensional, por lo que la altura será una nueva variable a introducir en el diagrama polar.
- Funcionamiento en vehículos aéreos no tripulados.
- Implementación de la solución expuesta en un entorno de programación de robótica, el cuál esté listo para funcionar en vehículos reales.
- Adaptación de parámetros para implementación en UAV.

4 HERRAMIENTAS UTILIZADAS

Dada las restricciones de seguridad del sector aéreo, una resolución de un conflicto como este no puede ser aceptada y desarrollada sin un riguroso testeo y evaluación. La simulación en el campo de vuelo es costosa, por lo que conviene realizar simulaciones en un entorno de simulación realista que permita testear las diferentes situaciones de riesgo. Es una herramienta fundamental antes de realizar pruebas en el campo de vuelo. Esto abaratará costes y permitirá realizar un mayor número de pruebas.

Para realizar la simulación de este algoritmo en UAVs, concretamente en un quadrotor, se utiliza ROS y GAZEBO. A continuación, se describirá de forma generalizada, por qué se han elegido estas herramientas, cómo funcionan y cuál es el sistema de paquetes de código utilizado para este trabajo.

4.1 ROS

4.1.1 ¿Por qué ROS?

Se ha elegido ROS para implementar el algoritmo propuesto debido a que proporciona multitud de librerías y herramientas para ayudar a desarrolladores de software a crear aplicaciones para robots. ROS provee abstracción de hardware, controladores de dispositivos, librerías, herramientas de visualización, comunicación por paso de mensajes, administración de paquetes, etc. Además, está bajo la licencia de código abierto. A continuación, se describen algunas de las características más importantes que hacen que ROS sea una de las herramientas más importantes en robótica.

- El sistema de comunicación es, a menudo, una de las primeras necesidades cuando se implementa una nueva aplicación de robótica. El sistema de mensajería que ROS incorpora permite ahorrar tiempo, ya que administra los detalles de comunicación. Otro beneficio de usar este sistema de paso de mensajes es que obliga a implementar interfaces claras entre los nodos del sistema, mejorando así la encapsulación y promoviendo la reutilización del código.
- Debido a que el sistema de paso de mensajes es anónimo y asíncrono, los datos pueden ser fácilmente capturados y reproducidos sin ningún cambio en el código. Si existe una tarea A que lee los datos de un sensor y se está desarrollando una Tarea B que procesa los datos producidos por la Tarea A; ROS facilita la captura de los datos publicados por la Tarea A en un archivo y los vuelve a publicar en un momento posterior. La abstracción de paso de mensajes permite que la tarea B sea independiente de la fuente de los datos, que podría ser la tarea A o el archivo de registro de datos. Este es un potente patrón de diseño que puede reducir significativamente el esfuerzo de desarrollo y promover la flexibilidad y la modularidad en el sistema.
- El sistema de publicación y subcripción de mensajes mencionado anteriormente es de naturaleza asíncrona, pero a veces se requieren interacciones síncronas de petición y respuesta entre procesos. ROS proporciona esta capacidad utilizando servicios.
- La licencia de código abierto, entre otras cosas, proporciona bibliotecas y herramientas específicas para robots que harán que un robot funcione rápidamente sin necesidad de reinventar la rueda.
- ROS se puede utilizar sin estar conectado al robot, ni tampoco, tener la necesidad de utilizar una interfaz gráfica de usuario. Todas las funcionalidades básicas y las herramientas son utilizables desde la línea de comandos. Hay comandos para lanzar grupos de nodos, grabar, reproducir datos, etc. Si se prefiere, también ofrece herramientas gráficas.

4.1.2 Estructura de ROS

Para entender mejor el funcionamiento de ROS se realizará una breve descripción de los componentes más destacados que forman el ecosistema ROS.

- **Master**: El ROS Master registra todos los nodos, servicios y topics que se encuentran en ejecución. Sin el maestro, los nodos no serían capaces de encontrar al resto de nodos, intercambiar mensajes o invocar servicios.

- **Nodos**: Los nodos son procesos que realizan la computación. ROS está diseñado para ser modular; un sistema de control de robots comprende por lo general muchos nodos. Por ejemplo, un nodo controla un láser, un nodo controla los motores de hélices, un nodo lleva a cabo la localización, un nodo realiza planificación de trayectoria, un nodo proporciona una vista gráfica del sistema, y así sucesivamente. Cada nodo pregunta al master por la localización de otro nodo y se comunica mediante el paso de mensajes y servicios.

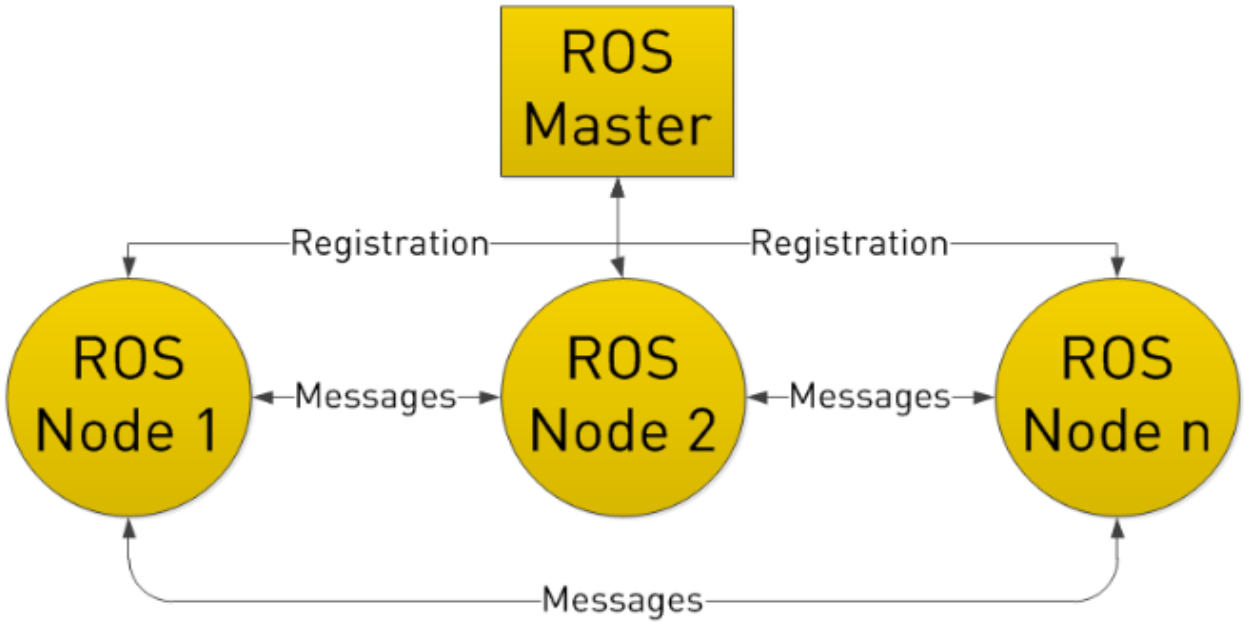

Ilustración 14 - Estructura de nodos ROS

- **Mensajes**: Un mensaje es una estructura de datos. Los mensajes pueden incluir tipos de datos primitivos (integer, floating point, boolean, etc.), arrays de tipos de datos o estructuras.

- **Tópicos**: Los mensajes son enrutados por un sistema de transporte publicador/subscriptor. Un nodo envía mensajes publicando en un tópico o *topic*. Un nodo que está interesado en cierto tipo de datos se subscribirá al topic apropiado. Pueden existir muchos publicadores concurrentemente para un solo topic y un nodo puede publicar y/o subscribirse a múltiples topics. En general, los publicadores y subscriptores no son conscientes de los otros nodos que están subscritos o está publicando en ese topic. La idea es desacoplar la producción de información de su consumidor.

- **Servicios**: El modelo de publicación y subscripción es una forma de comunicación muy flexible, pero en muchos casos, el transporte en un único sentido no es suficiente para las interacciones de petición y respuesta que a menudo se requieren en un sistema distribuido. La petición y respuesta se realiza a través de los servicios, que se definen a partir de una estructura de mensajes: una para la petición y otra para la respuesta. Un nodo proporciona un servicio con un nombre y un cliente utiliza dicho servicio mediante el envío del mensaje de petición y respuesta.

- **Bag**: Un bag es un formato para guardar y reproducir mensajes de ROS. Los bags son un mecanismo importante para el almacenamiento de datos. Permiten grabar la lectura de sensores, que pueden ser difícilmente adquiridas, pero son necesarias para el desarrollo y testeo de algoritmos.

4.1.3 Sistema de paquetes

Para la realización de este trabajo se han utilizado dos paquetes de ROS principales: *grvcTEAM* y *collision_avoidance*.

El primero de ellos es un paquete proporcionado por el departamento de Robótica y Automática de la Universidad de Sevilla. Este paquete permite simular un quadrotor que dispone de un autopiloto PX4. Contiene todas las herramientas necesarias para realizar un "software in the loop". En este conjunto de paquetes también se dispone de una capa de abstracción que sirve de interfaz para comandar determinadas órdenes al UAV. Algunas de ellas son: despegue, aterrizaje, referencia de posición, referencia de velocidad, etc. Esto facilita considerablemente la realización de la simulación.

Además, este paquete contiene una versión light de un quadrotor, permite simular sin elementos que harían necesario un hardware más potente. La versión light es adecuada para simular el algoritmo de evitación de colisiones en 3D.

El segundo de ellos es el paquete que contiene todos los elementos que permiten la ejecución de la implementación del algoritmo SWAP. La programación de este paquete fue iniciada, por Grupo de Robótica, Visión y Control (GRVC), para realizar una implementación del algoritmo SWAP en el MBZIRC. Finalmente no dio tiempo a terminarse y fue retomada para finalizarla en este trabajo.

En este paquete se encuentra el código que ejecuta el nodo *swap_avoidance*, y además, el nodo que va ejecutando las diferentes órdenes de la misión y comunicándose con el nodo swap. También se encuentran las dos librerías que hacen que el el nodo SWAP pueda ejecutar todas las funciones necesarias: la librería *swap* y la librería *polarobstacle*. La primera de ellas contiene la programación de las funciones encargadas de gestionar los conflictos y evitarlos, y la segunda contiene las funciones necesarias para construir el diagrama polar explicado anteriormente.

En la ilustración 15 se puede ver un esquema de los paquetes que permiten la simulación. MAVROS es un paquete de ROS que proporciona los drivers de comunicación para varios autopilotos con el protocolo de comunicación MAVlink.

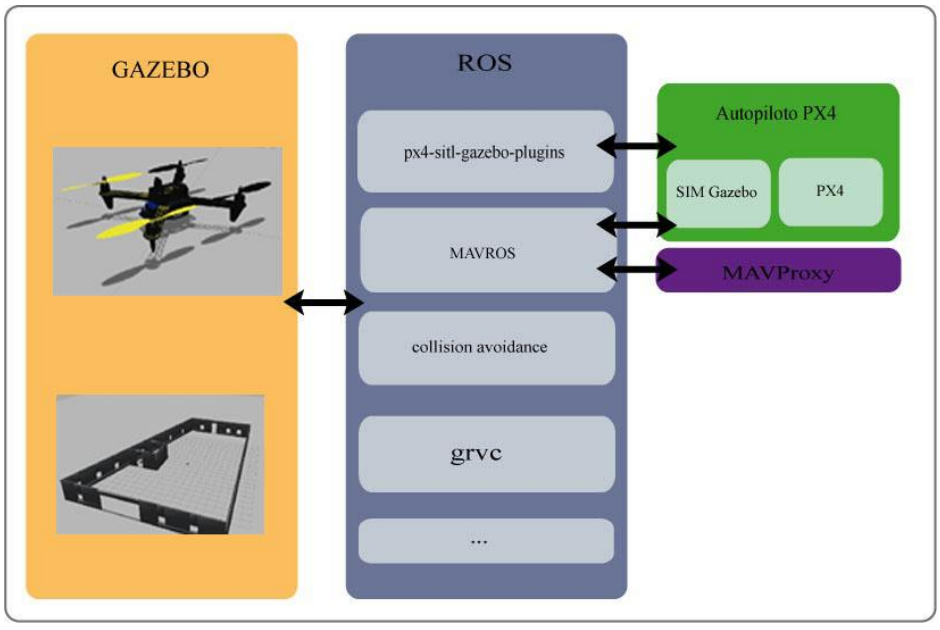

Ilustración 15 - Esquema de paquetes en ROS

4.2 GAZEBO

Gazebo ofrece la capacidad de simular robots, de forma precisa y eficiente, en complejos ambientes interiores y exteriores. Posee un robusto motor de física, gráficos de alta calidad y una cómoda interfaz gráfica. Además, GAZEBO se integra con ROS de manera independiente a través de una serie de paquetes que permiten simular robots que están programados bajo el ecosistema ROS. Para ello, utiliza el sistema de paso de mensajes de ROS. A parte de lo mencionado, algunas de las características que hacen interesante este simulador son:

- Motores físicos de alto rendimiento.
- Proporciona rendering realista de entornos. Incluyendo iluminación de alta calidad, sombras y texturas.
- Permite el uso de sensores, opcionalmente con ruído. Láser, cámaras 2D y 3D, Kinect, sensores de contacto, etc.
- Permite grabar simulaciones realizadas y accede a ellas de forma similar a los bags de ROS.

Como se ha mencionado anteriormente, GAZEBO utiliza un sistema de nodos distribuidos similar al de ROS. ROS dispone de los paquetes necesarios para ejecutar GAZEBO en su sistema, de esta forma, se ejecutan una serie de nodos que permiten realizar el paso de mensajes necesario para una simulación en GAZEBO.

4.3 MATLAB

MATLAB (abreviatura de MATrix LABoratory, "laboratorio de matrices") es una popular herramienta matemática con lenguaje de programación propio. Algunas de las prestaciones que la hacen interesante para este trabajo son: la manipulación de matrices, la representación de datos y funciones, o la implementación de algoritmos.

En cada iteración, el programa guarda un archivo de texto con todas las medidas interesantes, para visualizarlas y compararlas posteriormente. Mediante un script de MATLAB, se extraen los datos guardados en el archivo de texto y se manipulan para ser visualizadas de una forma diferente a la que ofrece GAZEBO.

Esto es de especial interés y ha servido para depurar multitud de fallos a la hora de la implementación del algoritmo. Por ejemplo, se programó un script en el lenguaje de programación de MATLAB para representar como iban evolucionando la posición del vehículo, su región de seguridad y la región de inflado de los obstáculos que evitaba. De esta manera, se podía aislar el problema a un problema de 2D si se tenía la certeza de que el fallo de implementación no estaba en la altura. Esto es un ejemplo de cómo se puede visualizar las variables en MATLAB, grabadas en el archivo de texto durante la simulación.

No solo se ha utilizado MATLAB como ayuda para detectar fallos en la implementación, también se ha utilizado para representar los datos de manera que puedan justificarse las conclusiones que se desarrollarán en capítulos posteriores. Son representadas variables como la distancia entre UAVs, el tiempo de la misión, distancia al objetivo, etc.

5 EXTENSION SWAP

En este capítulo se desarrolla la extensión del algoritmo SWAP para que pueda ser utilizado exitosamente en vehículos aéreos, con las necesidades que este tipo de vehículos plantean. Se analizan las dificultades que se han encontrado y la forma de solucionarlas.

El principal desafío que supone extender el algoritmo SWAP a este tipo de vehículos es la extensión a tres dimensiones. No solo hay que tener en cuenta la extensión a 3D, las características dinámicas de este tipo de vehículos son también fundamentales. El algoritmo SWAP funciona en 2D y está pensando para uniciclos con características dinámicas diferentes a las de los UAVs.

Este algoritmo se ha implementado en el ecosistema ROS, mediante el lenguaje de programación C++. Como se explicará en posteriores apartados, ROS ofrece un conjunto de herramientas de gran utilidad para desarrollar aplicaciones robóticas. ROS ofrece la posibilidad de ejecutarse en el mismo UAV de la misma forma que se ejecuta en un ordenador personal, esto lo hace especialmente interesante a la hora de realizar simulaciones realistas.

5.1 Extension 3D

Una de las características más interesantes de un UAV es su capacidad de navegar en el espacio tridimensional. Esto lo hace especialmente atractivo para determinadas aplicaciones. Por lo tanto, ya que este tipo de vehículo puede navegar en el eje Z, un algoritmo de evitación de obstáculos destinado a funcionar en UAVs debe ser modificado para tener en cuenta la altura. Para ello, la altura de los obstáculos se almacenará junto al diagrama polar, visto en el algoritmo preliminar, para clasificar el tipo de obstáculo según su altura.

Los UAVs que ejecuten este algoritmo serán capaces de localizarse y evitarse en diferentes alturas. Pero, para una primera extensión, por simplicidad, se ha decidido que el movimiento de evitación que realiza un UAV sea en el plano x-y. Por lo tanto, cuando un UAV encuentre un obstáculo con riesgo de colisión en el espacio 3D, lo evitará manteniendo su altura constante.

Cabe destacar que, en este tipo de vehículos, no solo existe peligro de que colisionen sus partes entre sí, sino que, un UAV volando debajo de otro, debido al par que generan sus hélices, podría crear una diferencia de presiones que perturbara el vuelo de otro UAV, si está lo suficiente cerca en altura. Este caso también cuenta como colisión.

Por lo tanto, a diferencia del algoritmo preliminar, en el que los vehículos se aproximaban mediante una circunferencia de centro su medio y que ocupa todo su contorno, ahora, cada UAV se aproxima mediante un cilindro de una altura determinada y una base del mismo radio que la región de seguridad del algoritmo preliminar. En la ilustración 16, en verde, la región de seguridad en la extensión 3D. En rojo, puede visualizarse la región de inflado que hará que una posible colisión sea detectada con el suficiente tiempo para que el UAV pueda evitarlo, teniendo en cuenta la distancia de frenado, el error de posicionamiento y los posibles retrasos de comunicación.

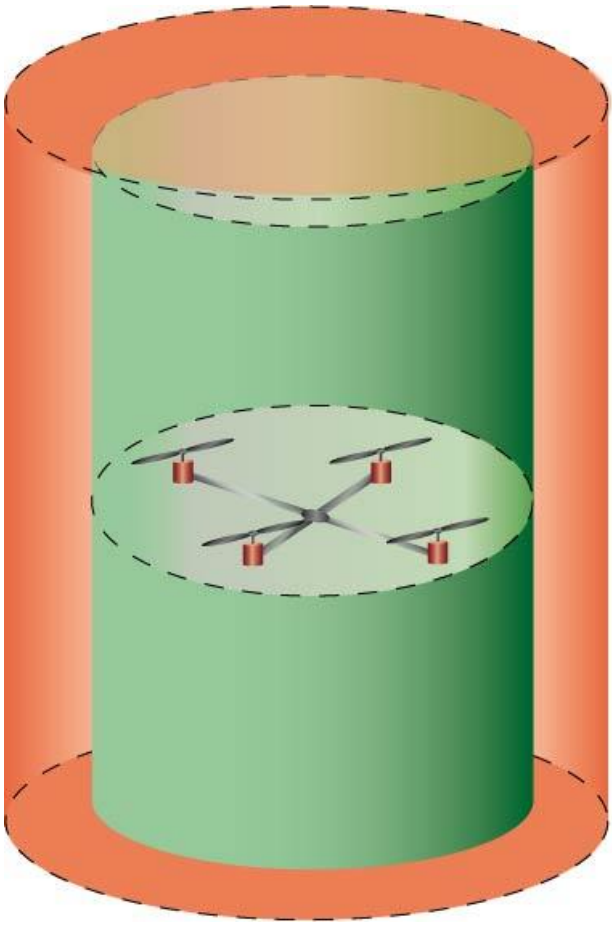

Ilustración 16 – Cilindro

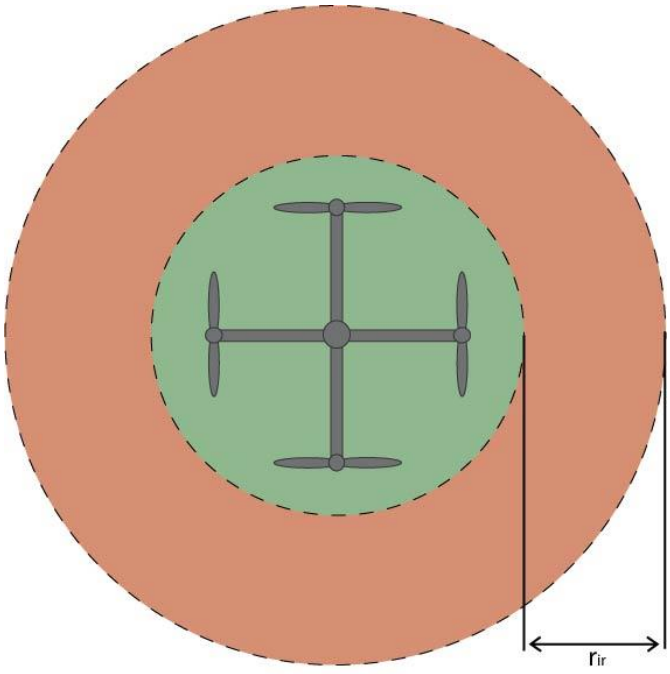

Ilustración 17 - Base del cilindro

Existe una situación particular en la que dos o varios UAVs están navegando hacia su destino y solo van a colisionar en una parte superior o inferior de sus cilindros. En este caso, el comportamiento óptimo será aquel que les haga navegar hacia su destino sin variar la altura y sin entrar en el movimiento característico de evitación, es decir, sin rodearse. De esta forma, se prevé una posible colisión en altura que haría retrasarse a ambos UAVs, realizando el movimiento de evitación SWAP cuando no es estrictamente necesario. En la ilustración 18, se ilustra un caso en el que sería especialmente interesante este tipo de comportamiento. Los dos UAVs al intersectar en una región superior fuera de la región de seguridad, siguen a su destino manteniendo la altura constante y evitando la colisión en altura.

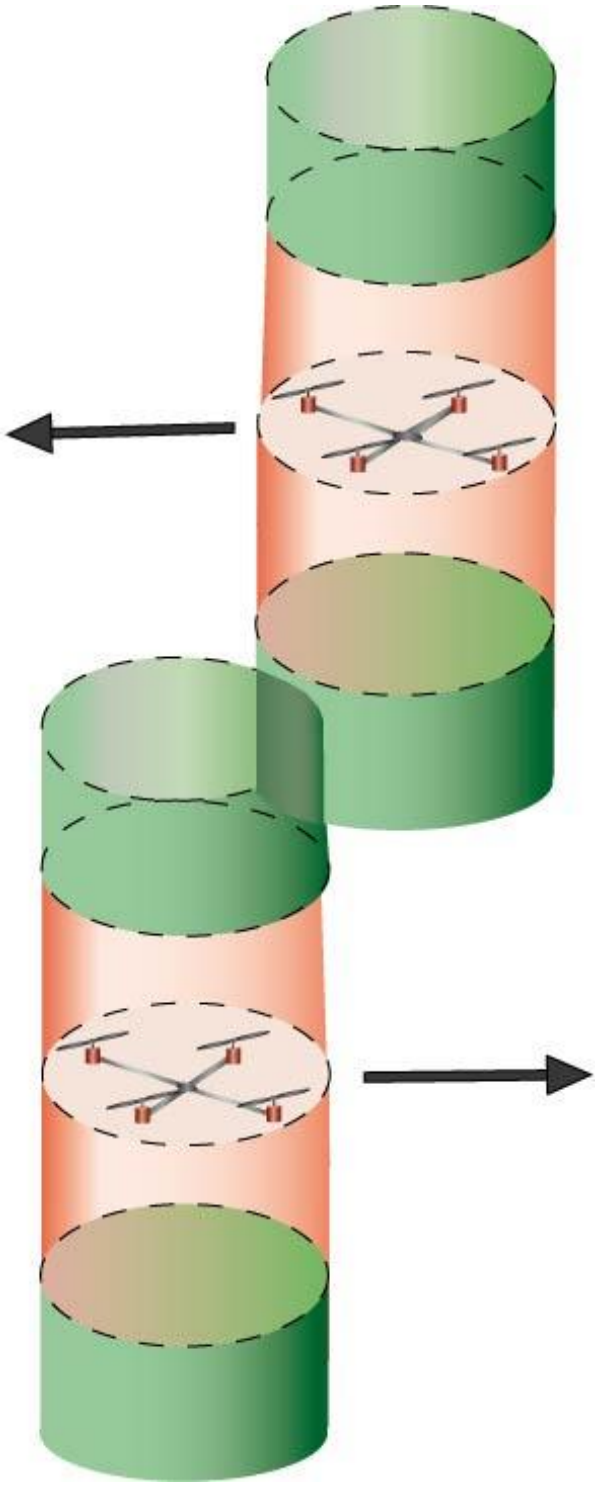

Ilustración 18 - Situación z-blocked

Para conseguir el comportamiento descrito en el párrafo anterior, se añade una zona de seguridad en la parte superior e inferior del cilindro. En la ilustración 19, se puede ver los dos parámetros que definirán el cilindro mencionado, con la posibilidad de variarlo en cada vuelo y dependiendo su valor del tipo de UAV que lo ejecute. De esta manera, los UAVs son capaces de optimizar su vuelo hacia el punto de destino, sobre todo cuando existan múltiples UAVs navegando a diferentes alturas.

- ***z-swap***: este parámetro define la altura del cilindro que representa la región de seguridad de un UAV. Cuando algún obstáculo o UAV intersecte con este cilindro se considerará colisión.
- ***z-range***: este parámetro define el rango de seguridad en los extremos superior e inferior del algoritmo. Este intervalo de los cilindros hará que se eviten las posibles colisiones en altura.

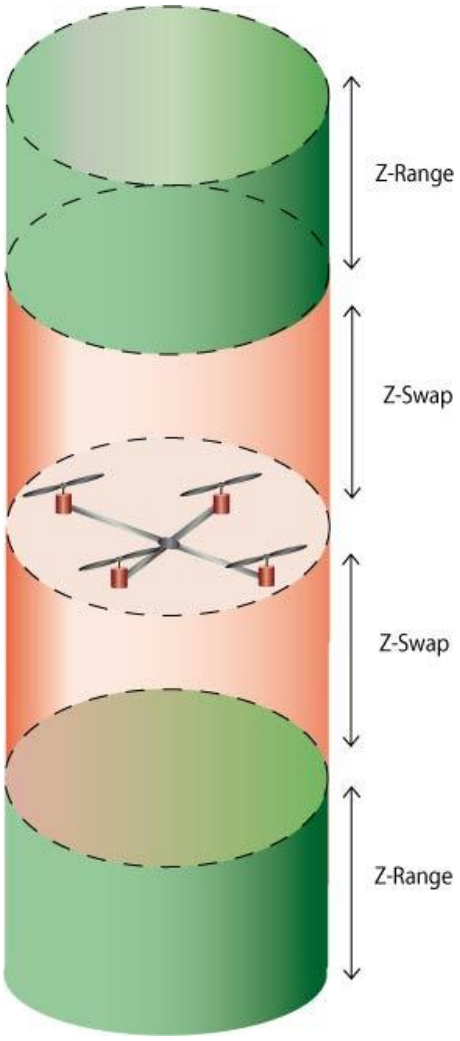

Ilustración 19 - Parámetros de altura

5.2 Extensión a vehículo aéreo no tripulado (UAV)

Este algoritmo se podría utilizar en cualquier tipo de UAV. En el algoritmo preliminar, se utilizaba en uniciclos. Estos tienen una dinámica muy diferente a la de los UAVs. Son especialmente diferentes a la hora de realizar la parada y evitación. Un quadrotor, por ejemplo, cambia su orientación modificando la velocidad de rotación de sus hélices, la transición de una orientación a otra opuesta es mucho más rápida. Por lo tanto, no tiene la misma necesidad de bajar su velocidad mientras cambia su orientación.

En simulación se vio que cuando el vehículo entra en estado *RENCONTRE,* reacciona mejor si se mantiene la velocidad $v_{ref} = v_{max}$ a si se le comanda $v_{ref} = 0$. Esto ocurre porque el UAV cambia de dirección distribuyendo diferentes velocidades en los rotores, no necesita rotar ningún eje de guiado.

Este algoritmo se podría ejecutar en cualquier vehículo aéreo cambiando los parámetros del mismo. En este caso se simula mediante el SITL del quadrotor utilizado para el MBZIRC. Este quadrotor dispone de un controlador PX4 al que se le comandarán referencias de velocidad en las tres componentes locales del robot, V*x*, V*y* y Vz. Esto permitirá gobernar el comportamiento del UAV tanto en velocidad como en orientación. De la misma manera, existe la posibilidad de gobernar el comportamiento del UAV dándole las referencias de control en posición, pero tal como se vio en simulación, funcionaba mejor si las referencias de control eran en velocidad.

Otro de los trabajos interesantes, desarrollados en simulación, era seleccionar unos parámetros adecuados para que el algoritmo funcionase correctamente. Parámetros como la distancia de seguridad o el offset que prevé errores de comunicación y de retraso, son ajustados para que funcionen de una forma adecuada en el quadrotor. Estos parámetros dependerán del tipo de vehículo que ejecute la implementación del algoritmo. Siendo de gran interés sintonizar estos parámetros en simulaciones realistas antes que realizarlos en el campo de vuelo. Los parámetros sintonizados son los siguientes:

- **Distancia de parada**: al igual que el algoritmo preliminar, la distancia de parada es la distancia que un UAV necesita para pararse por completo navegando a velocidad máxima.
- **Error de posicionamiento**: parámetro para asumir el máximo error posible en el sistema de posicionamiento
- **Offset Ω**: parámetro que permite aumentar la seguridad del sistema
- **Radio de seguridad**: radio de la base del cilindro que representa la región de seguridad de un UAV. Al igual que en el algoritmo preliminar, este será el radio del cilindro que engloba todo el contorno del UAV.
- **Control de rotación**: este parámetro actúa como la constante K de un control proporcional para mantener la distancia del UAV al obstáculo y, de esta manera, rotar en círculo.
- **Velocidad del UAV:** cuando el algoritmo comande la velocidad a la que debe navegar el UAV, el módulo de esta velocidad será igual a este parámetro.

5.3 Sensores utilizados

En este trabajo, los UAVs serán capaces de posicionarse a sí mismos y se enviarán las posiciones entre sí. Por lo tanto, un UAV recibirá de una manera descentralizada las posiciones de los UAVs a su alrededor y él enviará, a su vez, su posición a todos los UAVs que vuelan en su entorno.

Para probar y detectar el algoritmo propuesto en obstáculos fijos, se añade un sensor láser 2D. Este sensor dispone de n rayos laser que, cada uno, lee la orientación y distancia a la que el obstáculo intersecta con el láser. El algoritmo almacena la lectura del láser directamente en el diagrama polar, ya que, al proporcionar la distancia y el ángulo de orientación, está proporcionando una medida polar. En la ilustración 20 (en azul), se puede ver el láser implementado en GAZEBO para detectar obstáculos fijos.

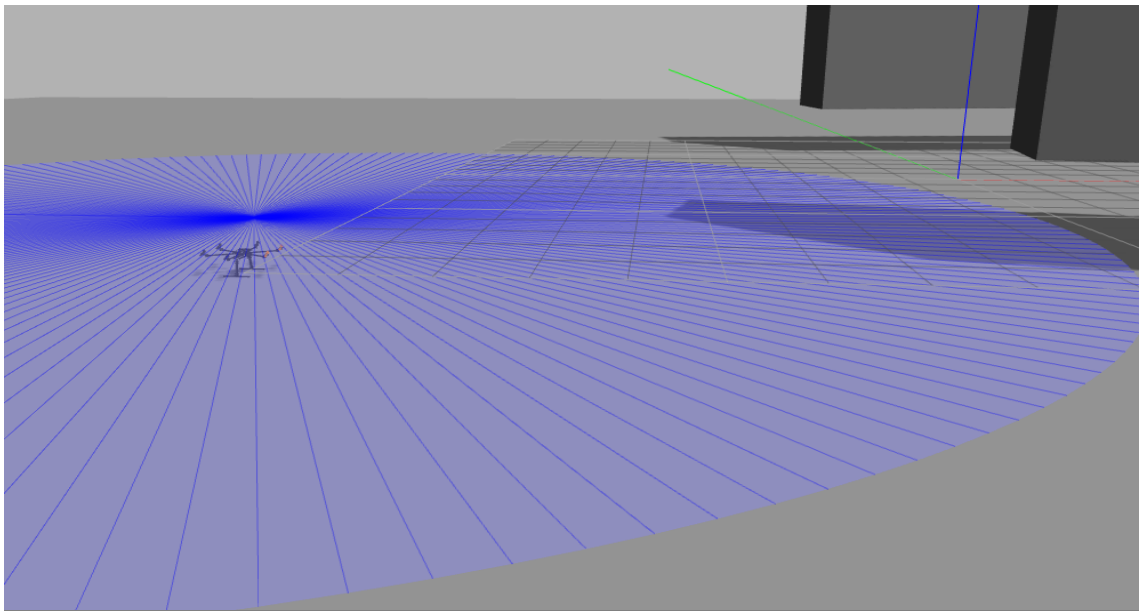

Ilustración 20 - Quadrotor con láser 2D

También cabe destacar que, igual que el algoritmo puede funcionar fácilmente con un láser 2D, también sería fácilmente adaptable para funcionar con un quadrotor que disponga de un láser 3D, por ejemplo, un LIDAR. Esto puede ser un posible trabajo posterior que permita al quadrotor detectar los obstáculos con un LIDAR sin necesidad de depender del posicionamiento de otros.

5.4 Máquina de estados

Tal como ocurre en el algoritmo preliminar, cada robot ejecuta, de una forma descentralizada, una máquina de estados que se encarga de asignar los comportamientos necesarios para que un UAV navegue, de forma segura y sin colisiones, hacia su destino.

Como novedad, en la extensión desarrollada en este trabajo, se añade un nuevo estado que se denomina *Z_BLOCKED*. A continuación, se van a describir cada uno de los estados que la máquina irá asignando dependiendo de la posición del UAV que ejecuta el algoritmo, y de la de los obstáculos y vehículos a su alrededor.

- **Free**: Es el comportamiento normal en navegación; donde el robot puede ir libremente hacia su destino. A diferencia del algoritmo preliminar, en este estado el robot navega en tres dimensiones hacia su destino. El robot está en estado FREE si no existen conflictos a su alrededor o si existen conflictos pero dándose dos condiciones: (1) φ_g no pertenece al sector prohibido; (2) el punto de destino no está detrás del robot. En este estado navega a toda velocidad a su destino, comandando una velocidad $v_{ref} = v_{max}$.
- **Blocked**: el robot entra en este estado cuando detecta conflictos y no hay ninguna orientación posible fuera del sector prohibido. Esto significa que el estado está bloqueado y rodeado de obstáculos, por lo tanto, el robot mantendrá la posición constante ($v_{ref} = 0$) y esperará hasta que algún obstáculo se mueva y deje una orientación φ_a que permita escapar.
- **Rencontre**: El robot entra en este estado cuando detecta un conflicto que debe y puede evitar antes de seguir navegando en la dirección de su destino. En este estado, la orientación será la del ángulo de evitación y la velocidad del UAV se reducirá por seguridad.
- **Rendezvous**: En este estado el obstáculo ya no se encuentra de frente, es decir, ya se ha pasado una parte crítica, aunque el UAV debe seguir evitando el obstáculo. Se aumenta la velocidad y se mantiene la orientación φ_a que permitirá evitar el obstáculo. El UAV no dejará este estado hasta que su región de seguridad deje de intersectar con la región de inflado del obstáculo que está evitando o

hasta que el punto de destino se encuentre visible sin obstáculos en su camino.

- **Z-blocked**: este estado ocurre cuando un UAV colisiona con un obstáculo solo en el intervalo superior o inferior de su cilindro. El UAV puede seguir navegando hacia su destino solo si mantiene su altura constante. Por lo tanto, $Vz_{ref}= 0$ y solo se modifican Vx_{ref} y Vy_{ref}.

5.5 Casos conflictivos

Existe un caso conflictivo que es necesario mencionar ya que no es tratado por ningún estado de forma particular. Este caso conflictivo se dará cuando dos UAVs estén navegando variando solo la altura y en sentido contrario. Como se puede ver en la ilustración 21, si los dos UAVs en la imagen se encuentran, entrarán en estado *z_blocked* e intentarán navegar solo en el plano X-Y, ya que sus puntos de destino están en el mismo plano x-y, los UAVs quedarán parados hasta que uno de los dos salga de esa posición.

Este caso es conflictivo ya que no existe un estado específico para evitarlo. Aun así, este caso no es peligroso, ya que los UAVs mantienen su posición, como si estuvieran en estado *blocked*, hasta que uno de los dos deje espacio para que el otro pueda navegar hasta su destino. Por lo tanto, no existe riesgo de colisión.

Hay otro caso conflictivo, relacionado con el láser 2D. Este láser permite detectar obstáculos fijos solo en el plano x-y que contiene al UAV. Si existe un obstáculo situado justo por encima o por debajo del plano mencionado anteriormente, este obstáculo no será detectado. Si el UAV está navegando en altura podría colisionar con un obstáculo de este tipo sin capacidad para detectarlo.

La incorporación de un láser 3D solucionaría este problema. Así pues, se incluye como un posible trabajo futuro.

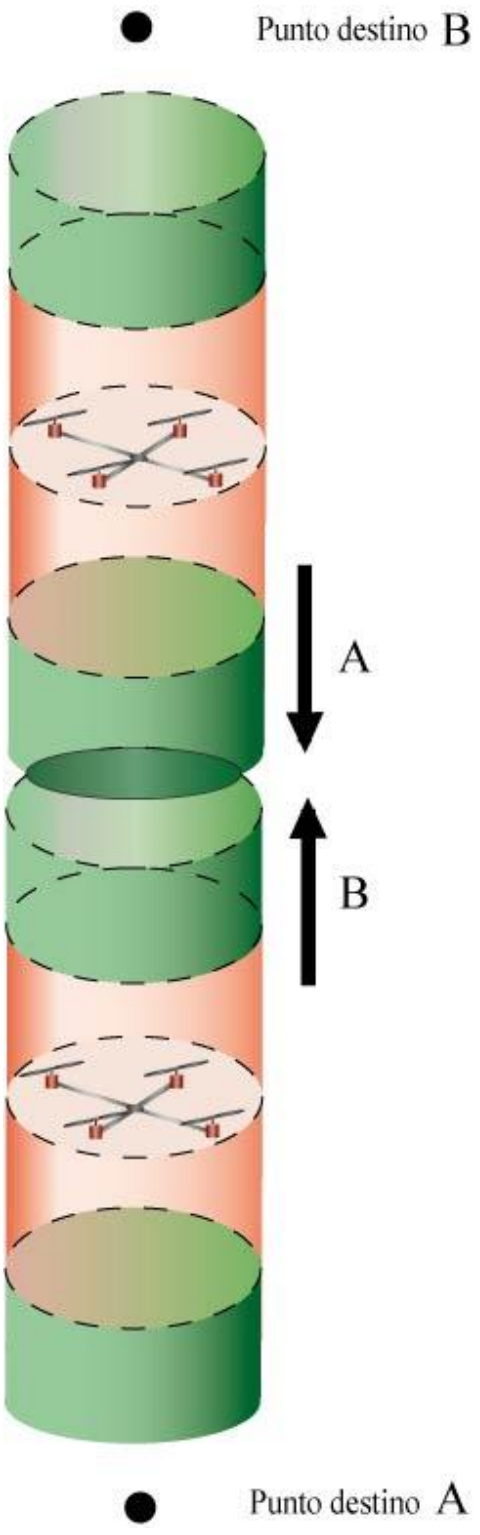

Ilustración 21 – Esquema de caso conflictivo

5.6 Implementación en ROS

5.6.1 Funcionamiento general y paso de mensajes

El algoritmo desarrollado en este trabajo está implementado bajo el ecosistema ROS y programado en el lenguaje C++. Se escogió este lenguaje ya que es, junto a Phyton, el que soporta ROS. Como se explicará más adelante, el entorno ROS funciona mediante nodos que publican y se subscriben a determinados topics

Existen dos nodos principales: *avoidance_swap* y *mission_planner*. El primero de ellos ejecuta la extensión del algoritmo SWAP y por ello será detallado posteriormente. El nodo *mission_planner* se creó para realizar la planificación de la misión. Este último se encarga de ordenar el despegue y aterrizaje del vehículo, así como de comandar la orientación hacia el siguiente punto de referencia o la orientación de evitación si una colisión fuese detectada por el nodo *avoidance_swap*. Es necesario destacar que cada vehículo ejecuta este sistema de nodos y paso de mensajes de manera individual y descentralizada.

Por lo tanto, el nodo *mission_planner* despega el vehículo y le comanda el error de posición en cada instante para que navegue hacia el siguiente punto de referencia. Cuando una colisión es detectada por el nodo *avoidance_swap*, este se lo comunica al nodo *mission_planner* mediante el topic *collision_warning*. Entonces, el nodo *mission_planner* leerá la orientación y velocidad necesaria para evitar la colisión mediante el topic *avoid_movement_direction* y se la comandará al UAV. Previamente, el nodo *avoidance_swap*, había calculado una velocidad y orientación que sirviera para evitar el obstáculo y la había publicado en el topic *avoid_movement_direction*.

El topic *wished_movement_direction* se utiliza para que el nodo *mission_planner* informe al nodo *avoidance_swap* de la orientación que le comandará al UAV para hacerlo navegar a su destino. Esto se realiza para que si existe un obstáculo y esta orientación es suficiente para evitarlo, el nodo *avoidance_swap* no lo identifique como colisión.

En la ilustración 22, se puede ver un esquema de los nodos y paso de mensajes descritos anteriormente.

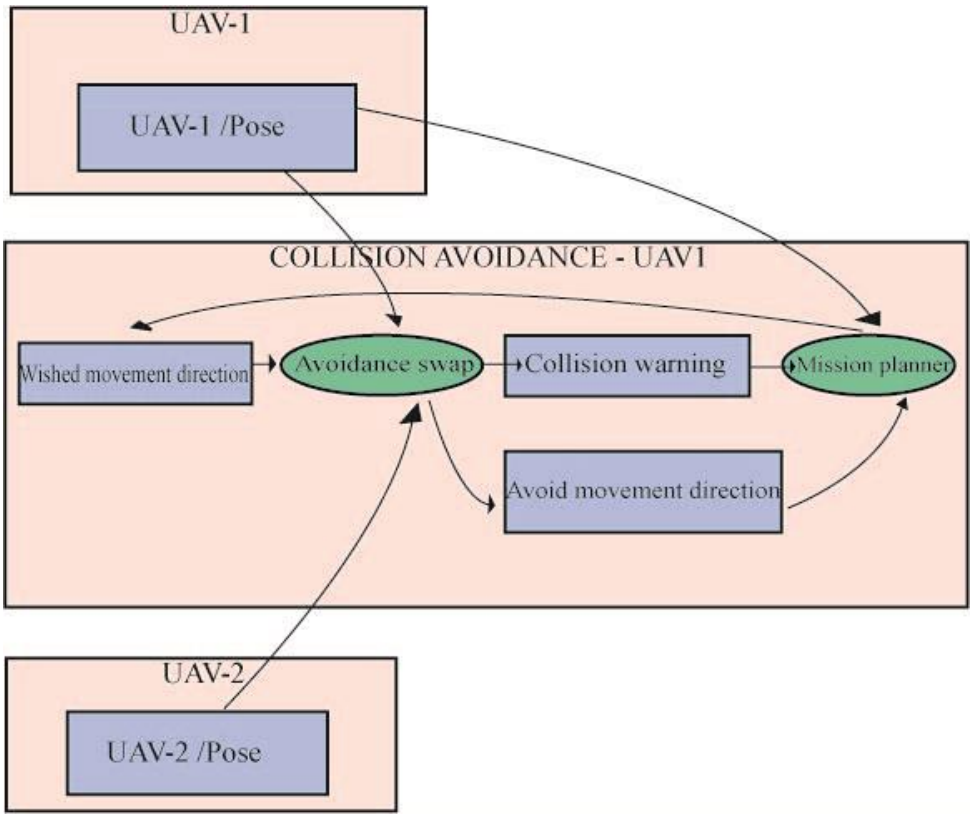

Ilustración 22 - Nodos y topics en simulación

5.6.2 Avoidance_swap

El nodo *avoidance_swap* es el engargado de ejecutar el algoritmo de evitación de colisiones en 3D. Este nodo está subscrito a los topics de posición de cada UAV (incluyéndose a si mismo) y al topic *wished_movement_direction*, usado para que el nodo que planea la misión le envíe la siguiente orientación prevista. A su vez, publica si existe colisión en el topic *collision_warning*, y la dirección y velocidad necesaria para evitar la colisión en el topic *avoid_movement_direction*.

Una vez leídos los parámetros del algoritmo, inicializado el nodo y subscrito a los tópics de interés, el nodo se mantenderá ejecutando iteraciones. A continuación, se describirá de manera secuencial una iteración del nodo que ejecuta el algoritmo.

1. Se reciben las medidas de los UAVs y del láser para almacenarlas en el Polar Obstacle Diagram (POD). El POD lo formarán vectores con, además de las componentes mencionadas en el SWAP preliminar (angulo y distancia), otra componente más según la diferencia de altura entre el obstáculo y el UAV. Se almacenan solo los puntos que son conflictivos en altura, dejando los que no intersectan sus cilindros fuera del POD.

2. Se construye la región de inflado

3. Se chequean los conflictos. Se llamará conflicto a aquel que, tras construir la región de inflado, intersecte con la región de seguridad del robot. Se elije como punto conflictivo, de todos los puntos de un mismo obstáculo, aquel que esté más próximo al robot. Se guardan, también, la altura de los puntos conflictivos.

4. A continuación, se ejecuta la función que evalúa el tipo de conflicto, llamada conflict manager.

a. Si el conflicto está solo en el rango de z_range, se comandará la velocidad y orientación del estado *z_blocked*.

b. Si hay conflictos, pero son evitables sin necesidad de cambiar de orientación, se comanda la velocidad y orientación del estado free, y no se publica en el topic *collision_warning*.

c. Si existe conflicto y es necesario cambiar la orientación. Se publica la orientación y velocidad del estado *rencontre* o *rendezvouz* en el topic *avoid_movement_direction.* Se publica que existe una colisión mediante el topic *collision_warning*.

d. Si existe conflicto, pero no existe ninguna forma de resolverlo, se publica que existe colisión mediante el topic *collision_warning*, y la velocidad y orientación del estado *blocked* (v_{ref}=0).

5. Se ejecuta una función encargada de construir un archivo de texto para guardar información que luego será tratada mediante un script de MATLAB. Como se explicará posteriormente, esto se realiza para recoger datos y representar la simulación de forma distinta a la que se representa en GAZEBO.

5.6.3 Mission_planner

Este es el nodo encargado de ir comandando al UAV las diferentes órdenes que conforman la misión. En primer lugar, leerá de un archivo de texto, los puntos de referencia que deberá ir siguiendo cada simulación, posteriormente comanda las siguientes órdenes:

1. Despega el UAV.

2. Comanda la velocidad y orientación necesaria. Si ha recibido un mensaje mediante el topic *conflict_warning* comandará la orientación y velocidad necesaria para evitar el obstáculo. Ésta se encuentra en el topic *avoid_movement_direction.* Si no hay conflicto publicará la dirección y velocidad necesarias para el siguiente waypoint.

3. Comprueba si se ha llegado al siguiente waypoint. Si es así actualiza el siguiente waypoint de la lista. Esto se realiza hasta que se haya completado la lista de waypoints. Entonces, pasará al punto 4.

4. Aterriza el UAV

6 SIMULACIONES

6.1 Introducción

La implementación del algoritmo desarrollado en este trabajo no podría haber sido realizada sin un trabajo de simulación. Para comprobar el funcionamiento, depurar fallos y elegir los parámetros adecuados es fundamental disponer de herramientas que permitan una rápida visualización, de determinadas variables o comportamientos, mostrando el funcionamiento del algoritmo.

Del mismo modo, este capítulo resulta esencial para exponer el comportamiento del algoritmo, sacando conclusiones que permitan demostrar que se cumplen los objetivos de este trabajo y relacionando determinadas variables con el comportamiento del robot.

Para ello, como se explicó en capítulos anteriores, la implementación del algoritmo estará ejecutándose bajo el ecosistema ROS, del mismo modo que el programa que ejecuta la misión y el paquete que contiene las constantes dinámicas que rigen el comportamiento del quadrotor. El comportamiento realista del quadrotor será visualizado mediante el software GAZEBO. A su vez, de cada simulación se extraerá un archivo de texto con las variables de interés necesarias para representar gráficamente, mediante la herramiento MATLAB, el comportamiento del robot.

En este capítulo se mostrarán cuatro simulaciones elegidas por englobar todos los estados que rigen el comportamiento del robot que ejecuta el algoritmo de evitación, y por comprobar que el robot evita las colisiones con éxito en las situaciones más conflictivas.

Cada apartado se corresponderá a un tipo de simulación. En cada uno de ellos, en primer lugar, se describirá el escenario en el que se realiza la misión, los vehículos que componen el sistema en esa simulación y el objetivo de esta. Posteriormente, se presentará una secuencia de imágenes para mostrar el desarrollo de la simulación en GAZEBO. Por último, el comportamiento del robot se relacionará con representaciones gráficas de variables de interés, que permitirán sacar las conclusiones oportunas.

Las variables a mostrar dependerán de la simulación que se realice, ya que unas serán más representativas que otras dependiendo del comportamiento del robot. Entre otras: tiempo de la misión, distancia al obstáculo, distancia entre UAVs, distancia al objetivo, distancia recorrida, etc.

6.2 Simulación Cubo

El objetivo de esta misión es comprobar el funcionamiento del algoritmo cuando existen multitud de vehículos ejecutando el proceso de evitación, a la vez y de manera descentralizada. Para ello se hará coincidir a cuatro vehículos en un mismo espacio, de manera que tengan que evitarse al mismo tiempo.

Para forzar esta situación, las posiciones de los cuatro UAVs serán las de los cuatro vértices de la base de un cubo imaginario. La idea es que los UAVs se crucen en el centro del cubo. Para ello, intercambiarán sus posiciones siendo su posición final los vertices opuestos de la cara superior de ese cubo imaginario. En la ilustración 23 y 24 que se muestran a continuación, se pueden ver las posiciones iniciales y finales de la simulación

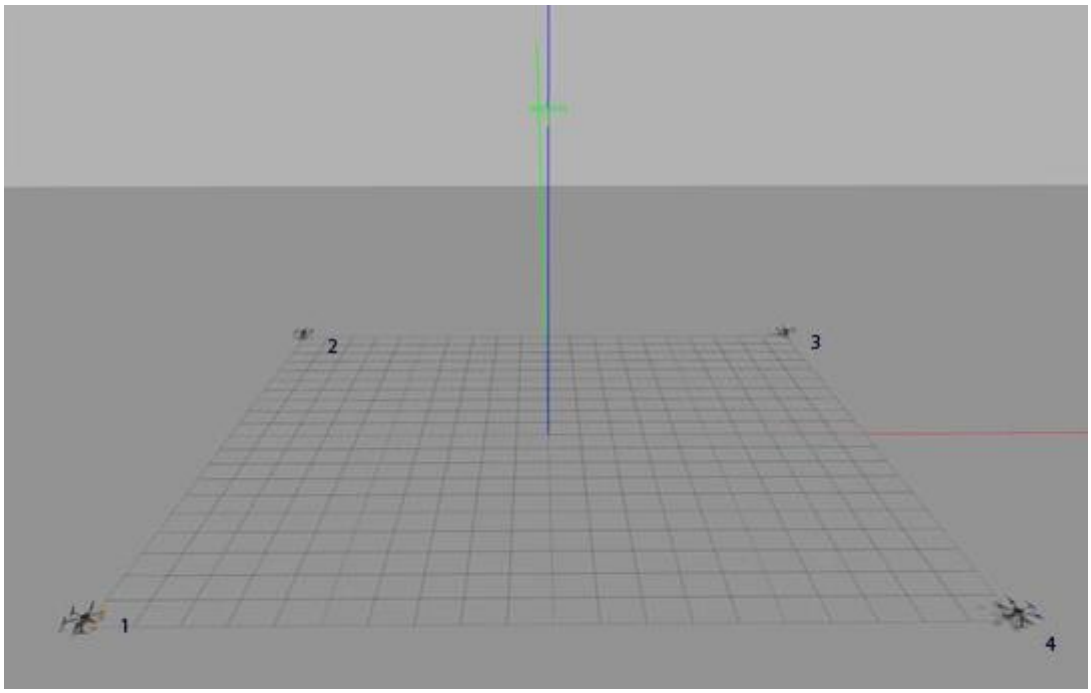

Ilustración 23- Posición inicial

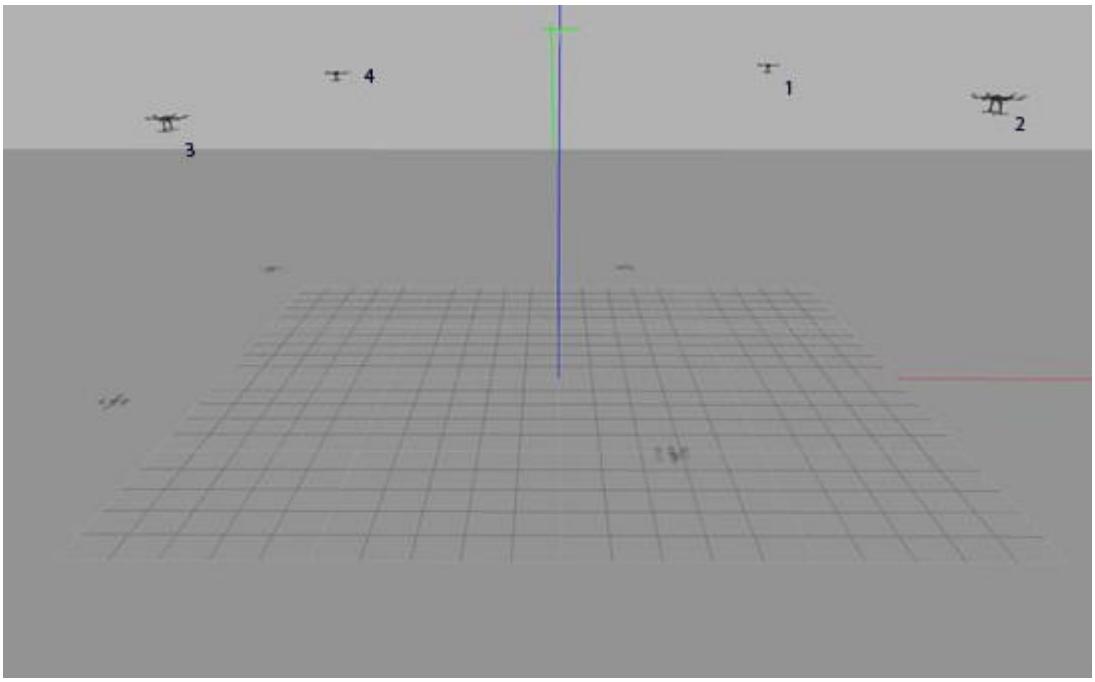

Ilustración 24 - Posición final

De esta manera, si todos los UAVs despegan a la vez, los cuatro convergerán en el centro al mismo tiempo y realizarán el movimiento de evitación formando una circunferencia y rotando cada uno en el sentido contrario de las agujas del reloj. A continuación, se expone una secuencia de imágenes con la simulación completa.

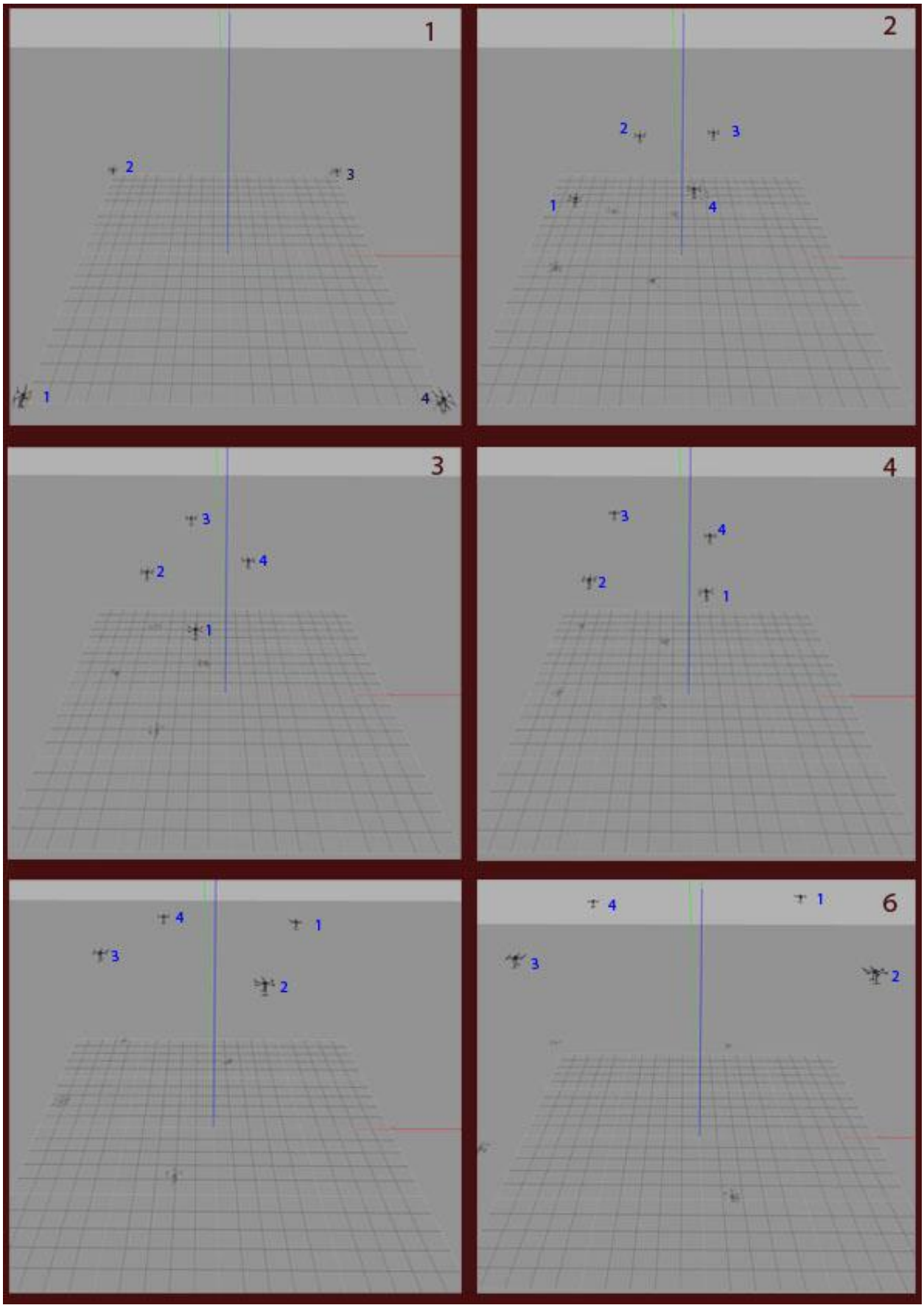

Ilustración 25 - Secuencia de imágenes simulación cubo en GAZEBO

En la ilustración 26 se comparan las distancias que los UAVs mantienen en el plano horizontal. Se analiza el comportamiento del UAV-1 relacionando su posición con la del resto de vehículos del sistema, ya que los únicos obstáculos que existen en su entorno son los tres UAVs restantes. Cabe destacar que nunca se produce colisión, ya que la distancia entre UAVs nunca baja de los dos metros (línea negra en la gráfica).

Como se puede ver en la gráfica, los UAVs despegan, se aproximan en el centro del cubo reduciendo sus distancias, mantienen las distancias mientras el movimiento de evitación se produce y cuando encuentran el camino libre navegan hacia el punto final manteniendo la misma distancia que al principio; ya que las distancia entre ellos en la posición inicial y final son las mismas.

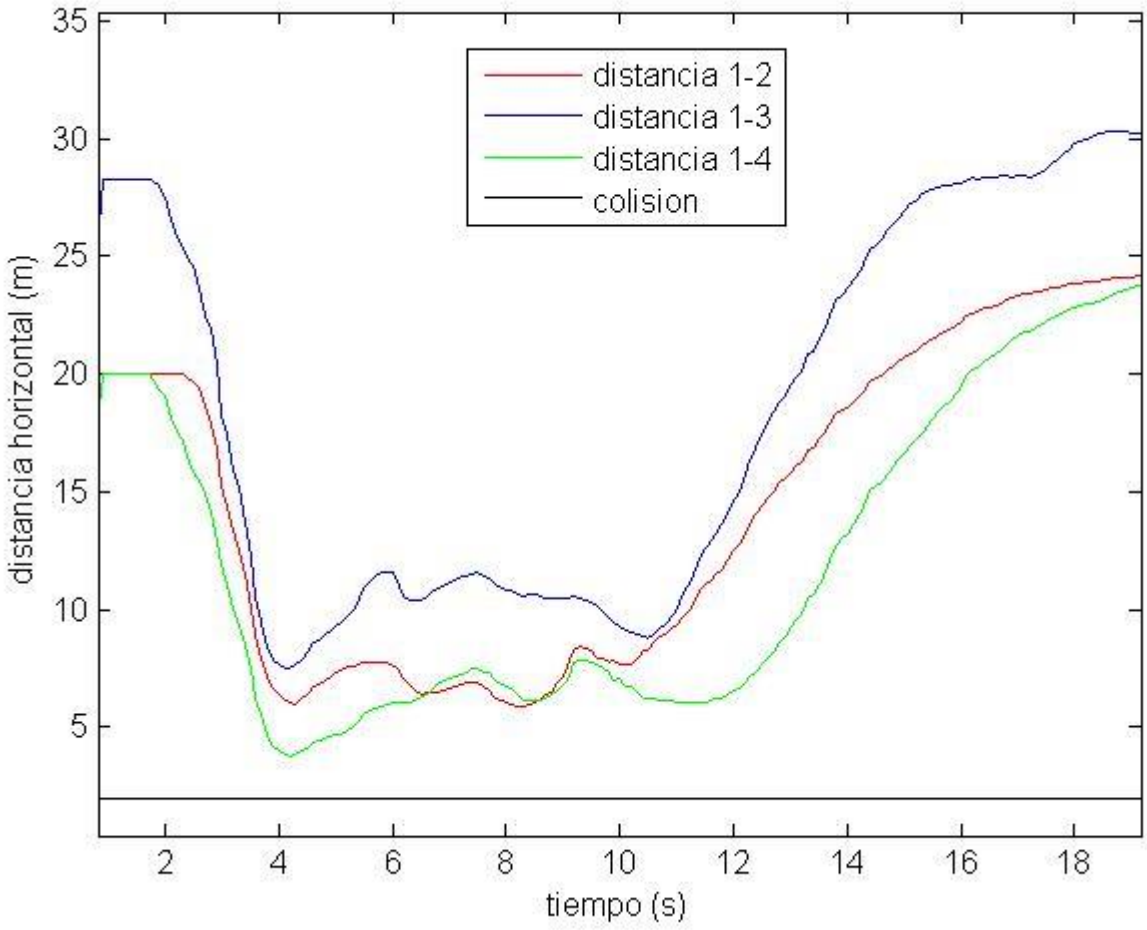

Ilustración 26 - Distancia horizontal entre UAVs

La ilustración 27 refleja la distancia vertical que los vehículos mantienen. En este caso no es muy relevante, ya que los UAVs nunca interseccionan en el plano, por lo tanto, aunque viajen a la misma altura, sus aproximaciones cilindricas nunca se tocarán. En la gráfica se puede ver como, en este caso, todos los UAVs viajan a la misma altura, ya que todos intentan la misma trayectoria a la vez: intercambiar los vertices del cubo.

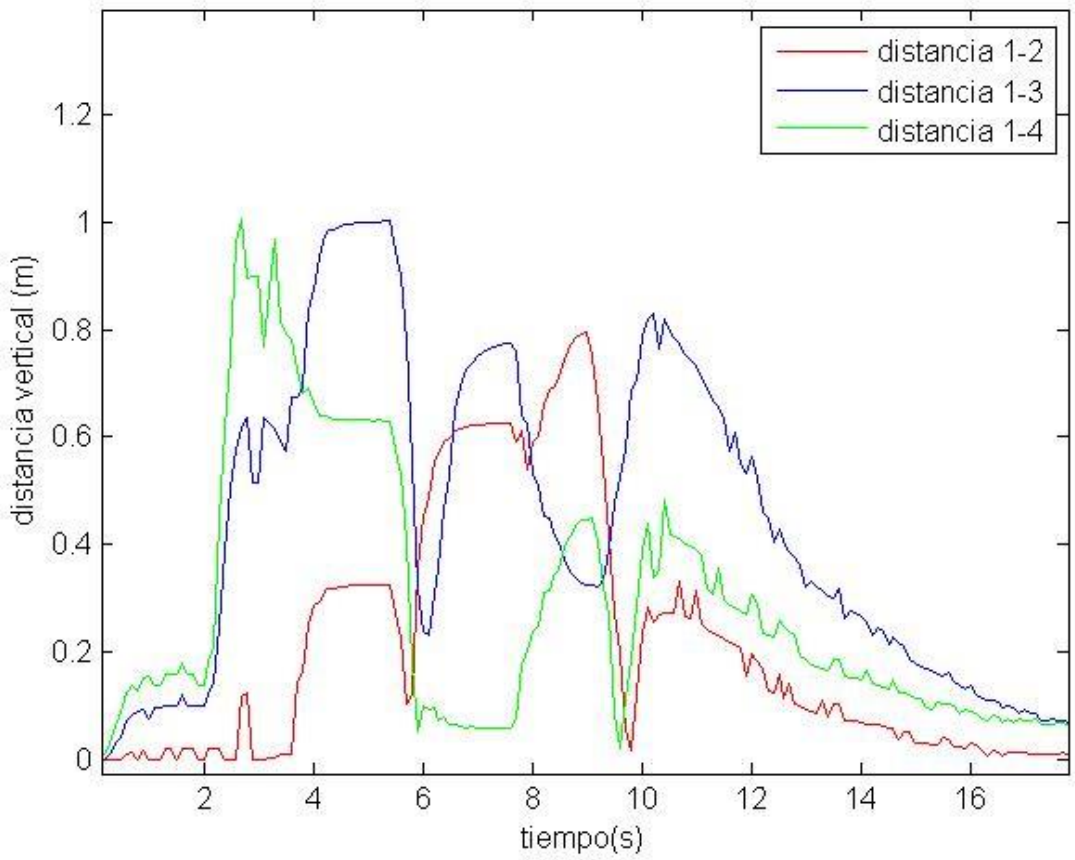

Ilustración 27 – Distancia vertical entre UAVs

A continuación, se expone una tabla con las distancias recorridas por los vehículos, el tiempo en el que se efectúa la misión y la distancia que recorrerían si no encontraran ningún obstáculo en su camino (distancia nominal). Los UAVs recorren distancias similares aun realizando el movimiento de evitación, ya que las distancias entre sus puntos inicial y final son las mismas. Del mismo modo, comparando las distancias recorridas con la distancia nominal, es posible asegurarse de que el UAV no realiza más de una vuelta en el movimiento de evitación. Un posible fallo de la implementación del algoritmo sería que el UAV permaneciera dando vueltas al obstáculo.

	UAV -1	UAV-2	UAV-3	UAV-4	Distancia nominal
Tiempo misión (s)	18,2	18,5	19	18,7	11
Distancia total (m)	43,349	43,6024	44,4755	43,6359	35

Tabla 1 - Distancias recorridas

6.3 Simulación U-Shape

El objetivo de esta simulación es comprobar que la implementación de este algoritmo funciona con éxito para detectar y evitar obstáculos en forma de U. Este tipo de obstáculos tienen la particularidad de obligar al vehículo a ir en dirección contraria al punto de destino para salir de la región que encierra el obstáculo en forma de U. Esta situación es propensa para que, un vehículo que ejecuta un algoritmo de evitación de colisiones, pueda quedarse atrapado.

Para detectar el obstáculo, el UAV lleva incorporado un láser 2D que detecta la posición de los obstáculos a su alrededor. Al láser se le asigna un rango de seis metros. Por lo tanto, enviará la orientación y distancia de todos los obstáculos, que situados en el plano del UAV, estén a menos de seis metros del robot.

En el escenario de esta simulación un UAV parte de una posición inicial hasta encontrarse rodeado por tres paredes A, B y C. El UAV tendrá que rodear el obstáculo, volviendo hacia atrás, para ser capaz de llegar a su destino evitando la colisión. En las ilustraciones 28 y 29 se observa el escenario en el que se realiza la simulación.

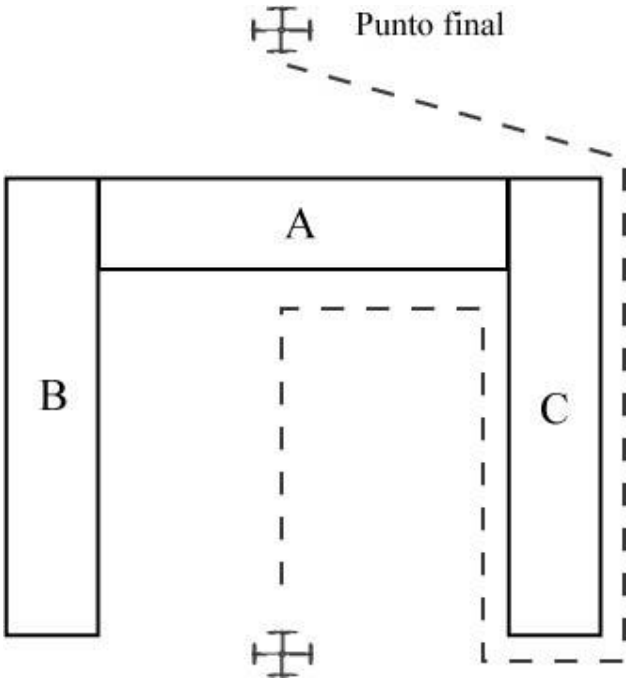

Ilustración 28 - Esquema del escenario

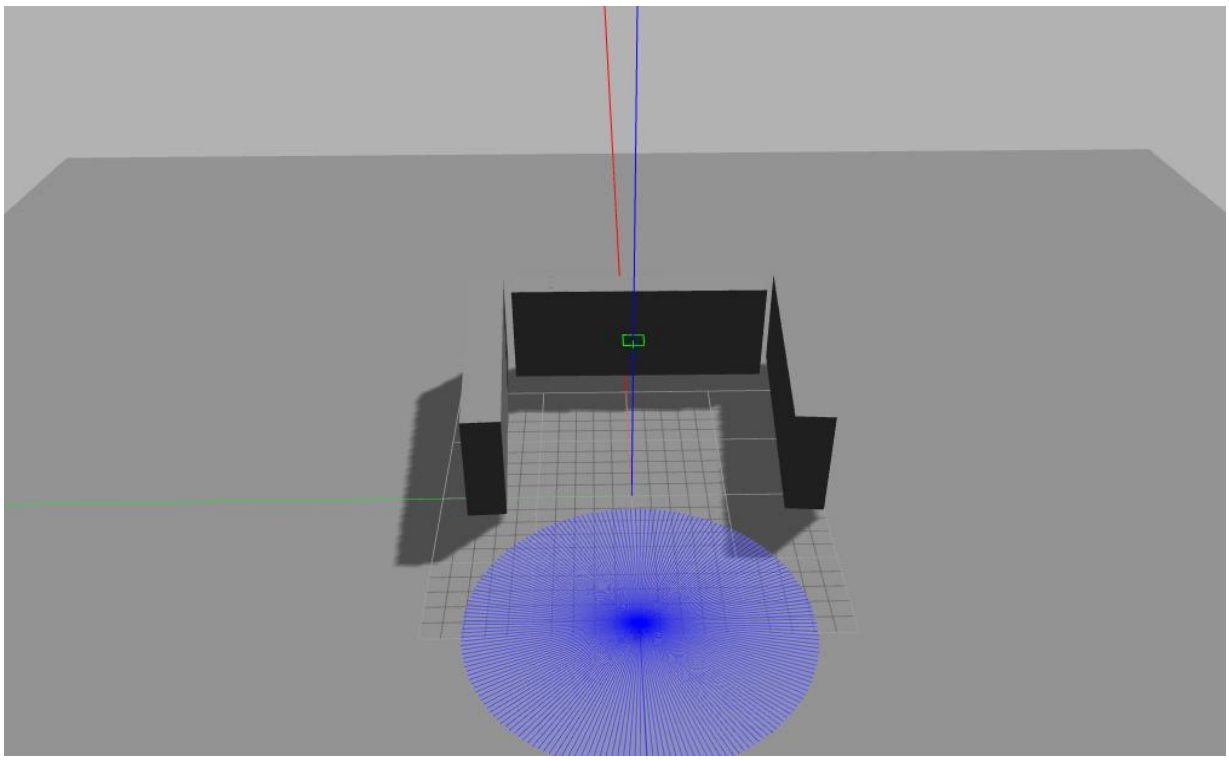

Ilustración 29 – Posición inicial

En la ilustración 28, se representa un esquemático del entorno en el que se realiza la simulación. Con línea discontinua está dibujada la trayectoria esperada por el robot para evitar el obstáculo. Detectando la pared del fondo y rodeando el obstáculo en el sentido de las agujas del reloj. En la siguiente figura se muestra una secuencia de imágenes de la simulación donde el robot realiza la trayectoria esperada.

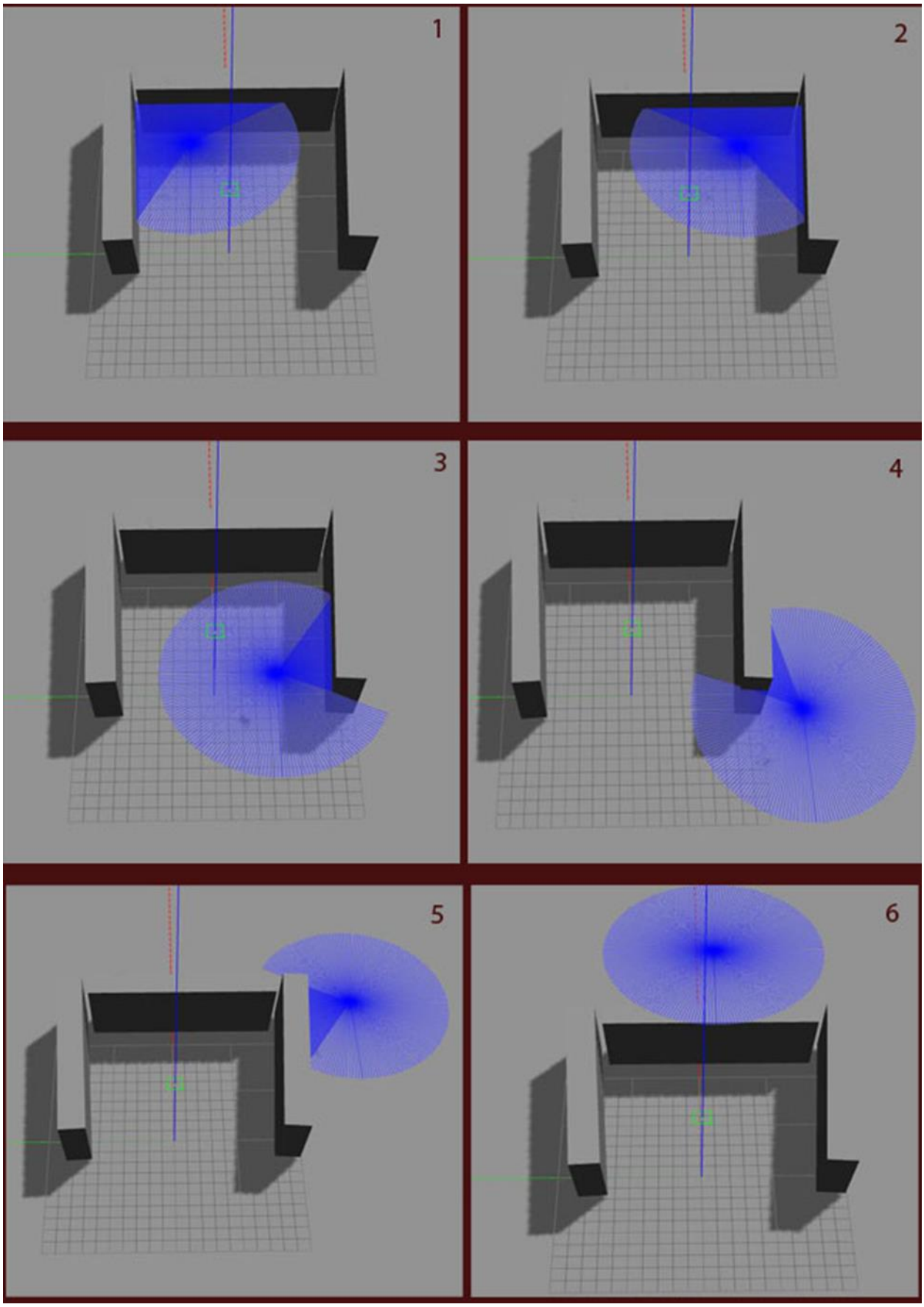

Ilustración 30 - Secuencia de imagenes en obstáculo U-Shape

Los parámetros que se le asignan al vehículo para realizar la misión son:

Distancia de frenado d_{br}: 3 metros

Error de posicionamiento e_r : nulo. (No existe ruido en las medidas)

Offset de seguridad Ω: 1 metro

Según los parámetros anteriores, el UAV debería detectar el obstáculo a una distancia de 4 metros. En ese instante frenaría y comenzaría el movimiento de evitación. En las ilustraciones 31, 32 y 33 es posible observar las distancias que mantiene el UAV con las paredes A, B y C. En ningún momento, se cruza la línea roja de colisión con ninguna de las paredes del obstáculo. Como se aprecia en la ilustración 31, solo está a una distancia de menos de dos metros cuando el vehículo ya ha salido del obstáculo en U y se dispone a navegar hacia su destino.

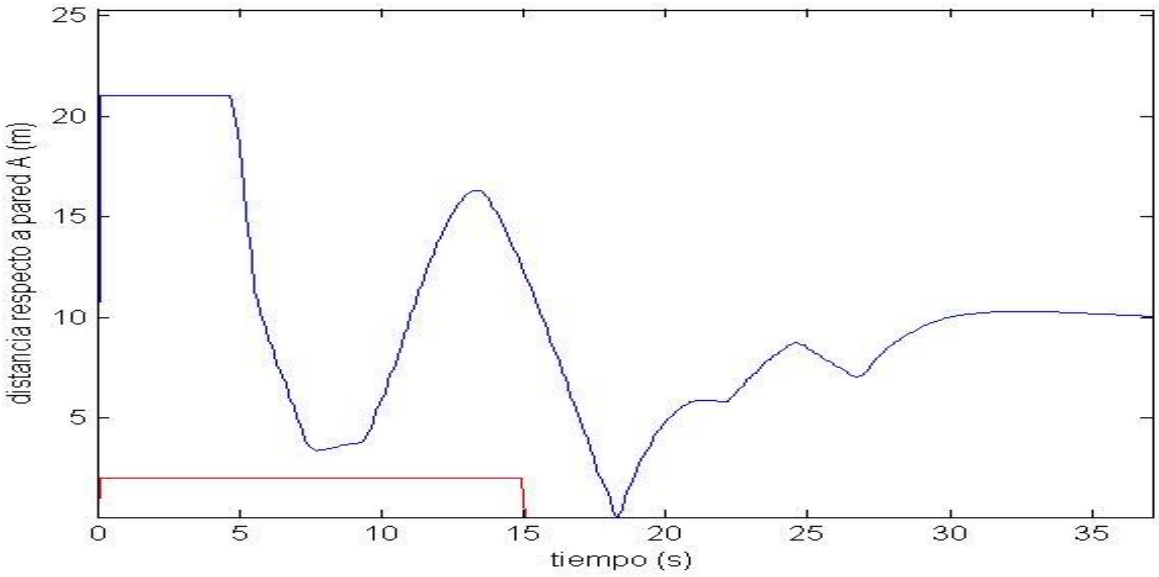

Ilustración 31 – Distancia del UAV respecto a la pared A

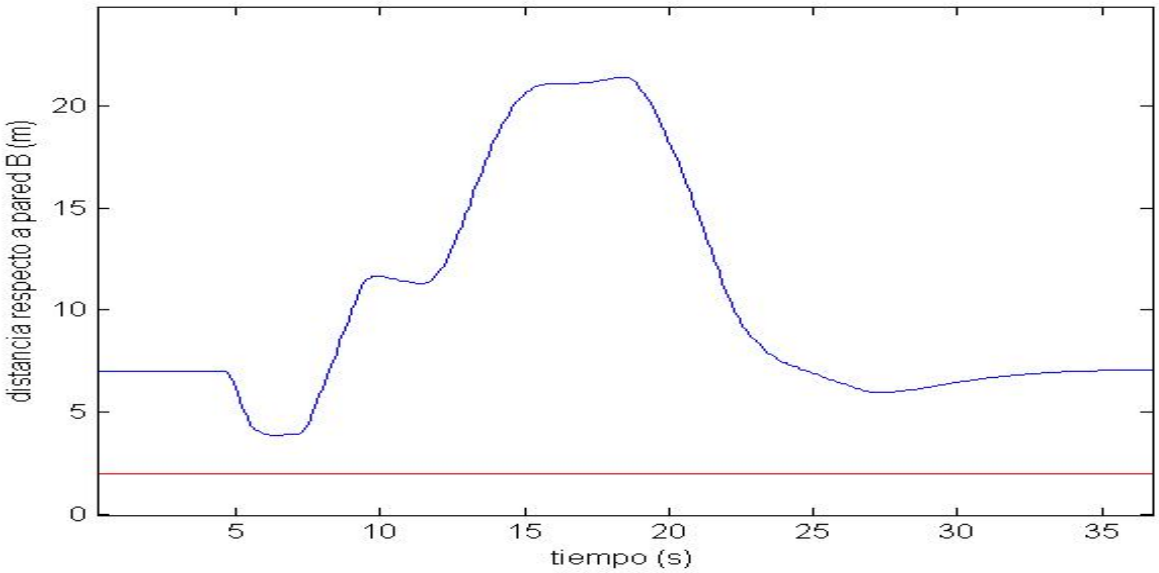

Ilustración 32 – Distancia del UAV respecto de la pared B

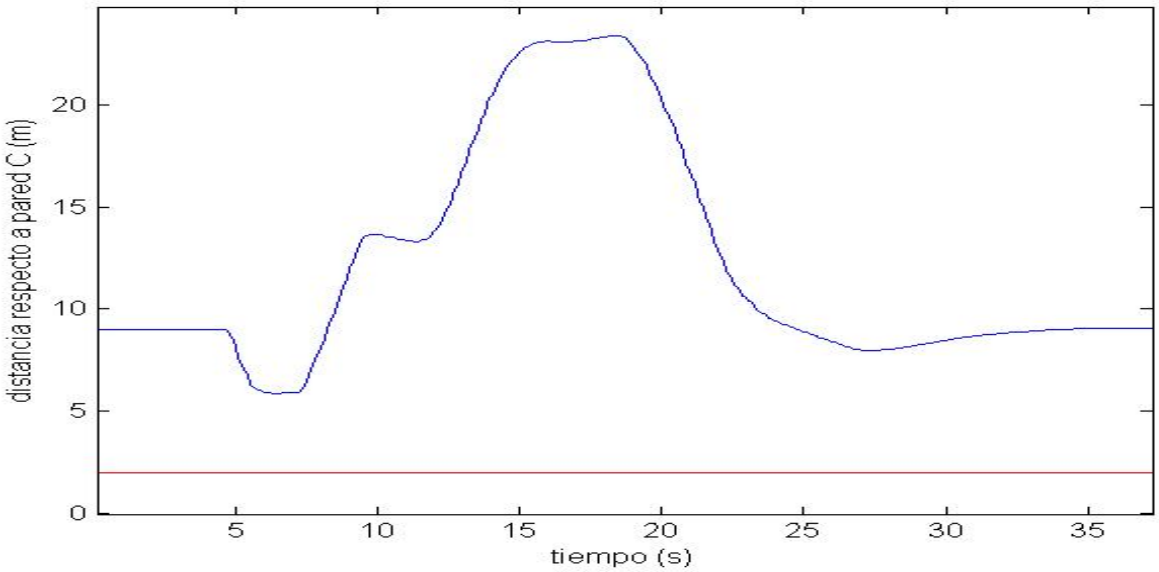

Ilustración 33 - Distancia del UAV respecto de la pared C

En la ilustración 34 se representa la distancia que mantiene el UAV con el objetivo. Si se relaciona con la secuencia de simulación expuesta en la ilustración 30, se puede apreciar como el UAV detecta la pared A en el punto [8 12.5] de la gráfica, y tiene que retroceder. El UAV rodeará la pared B hasta que ha salido de la región en forma de U encerrada por el obstáculo, punto [13.6 27]. Posteriormente se dirigirá hacia el objetivo, alcanzándolo en el punto [25 0.4].

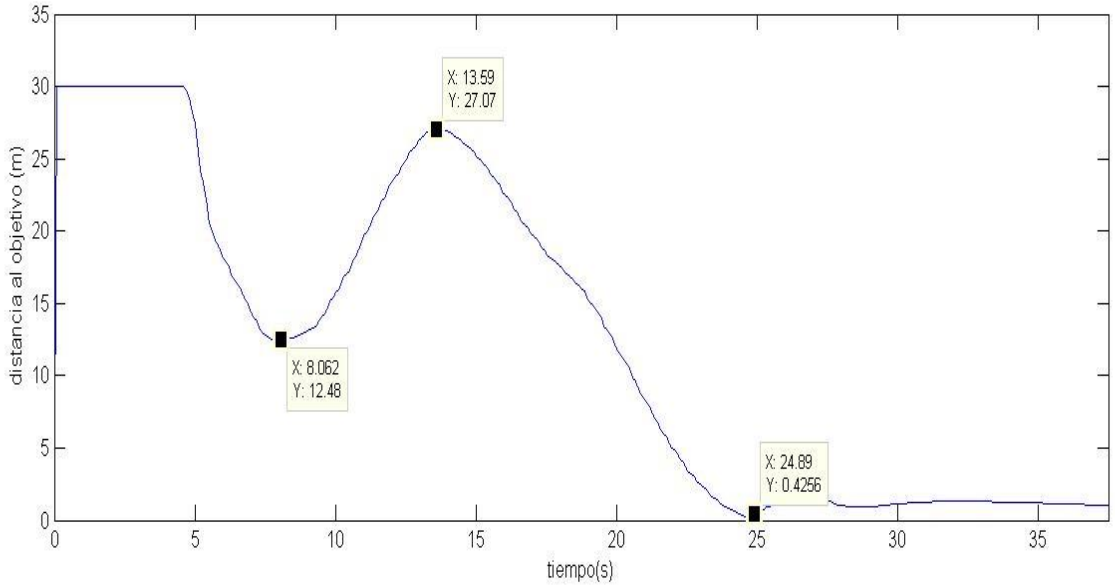

Ilustración 34 - Distancia respecto al objetivo

6.4 Simulaciones en diferentes alturas

Este apartado se incluye para comprobar los diferentes comportamientos que un UAV puede realizar dependiendo de la diferencia de altura entre obstáculos. Podrían darse tres situaciones diferentes:

- Dos o más UAVs están a una altura que no es peligrosa para que sus cilindros se intersecten. Estado *free*.
- UAVs se aproximan en el plano y a una altura peligrosa en la que pueden seguir navegando hacia su destino pero manteniendo la altura constante, previniendo el movimiento de evitación. Estado *z_blocked*.
- UAVs se aproximan en el plano a una altura en la que se va a producir colisión, sus cilindros van a colisionar, por lo que realizan el movimiento natural de evitación SWAP. Estado *rencontre/rendezvous.*

En este capítulo se analizarán los comportamientos de los estados free, rencontre, rendezvous y z-blocked; relacionando la altura de los UAVs implicados en la posible colisión.

6.4.1 Rencontre/rendezvous y Free

En esta simulación, un UAV navega hacia su destino (UAV-1) y en su entorno existen dos UAVs; uno de ellos a una altura similar (UAV-2) y otro a una altura mucho mayor (UAV-3).

Primero, se va a analizar la distancia entre los UAV 1 y 2, línea verde en la ilustración 35. Como se puede comprobar en la gráfica, la línea verde nunca cruza la línea roja de colisión ya que los UAVs están a alturas similares y esto provocaría la colisión. Durante los segundos 5 a 18 de la simulación, la línea verde se dispone cerca de la línea roja, esto se produce debido al movimiento de evitación SWAP mediante el estado *recontre/rendezvouz*. La distancia en el plano x-y se mantiene, evitando de esta forma la colisión.

Sin embargo, en la segunda gráfica, la línea azul sí cruza la línea roja de colisión, sin producirse esta, ya que los UAVs 1 y 3 están a una altura en la que no son peligrosos. Esto indica que, tanto el UAV-1 como el UAV-3, estaban en el estado *free* navegando hacia su destino, y aun cruzándose en el plano x-y, no cambian de estado debido a su diferencia de altura.

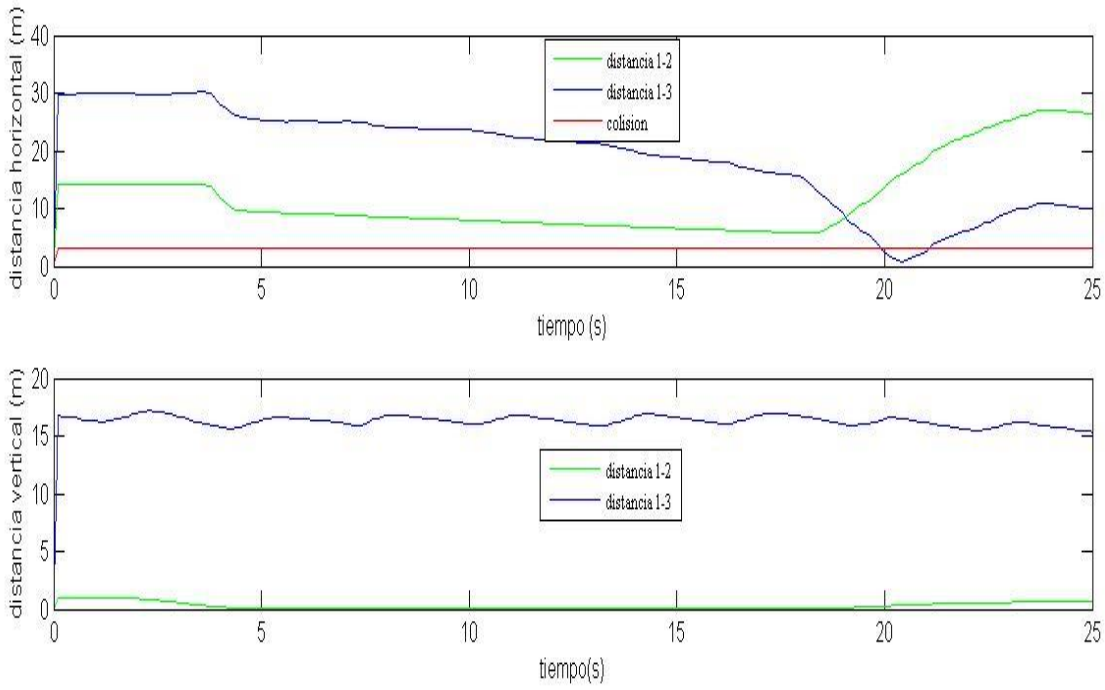

Ilustración 35 – Distancia entre UAVs en simulaciones de alturas

6.4.2 Estado z_blocked

En esta situación existen dos UAVs implicados. Estos dos UAVs navegan hacia su destino a una altura en la que los extremos de sus cilindros intersectan. Como se explicó en capítulos anteriores, esta altura no es peligrosa, pero si siguen acercándose podrían entrar en el estado *rencontré/rendezvouz.* Para anticiparse a esa situación, los UAVs entran en estado *z-blocked.* De esta manera, navegarán hacia su destino sin variar la altura hasta que sus cilindros no intersecten.

En la ilustración 36, es posible observar que en el intervalo de tiempo de 15 a 30 segundos de simulación, los UAVs mantienen su distancia en altura mientras se cruzan en el plano. En este intervalo los UAVs navegan en estado z-blocked.

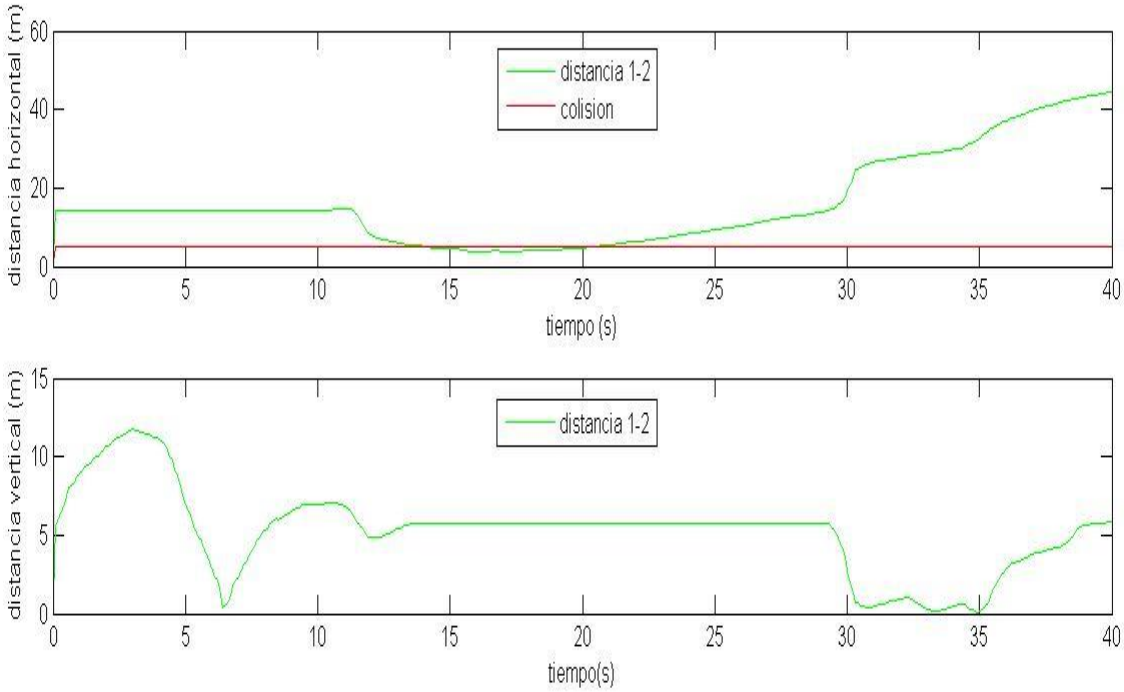

Ilustración 36 - Distancia entre UAVs en simulaciones de alturas

En la ilustración 37 se representa la distancia que mantiene uno de los UAVs respecto al punto de destino. En la gráfica superior, la distancia horizontal, y en la gráfica inferior, la distancia vertical. Se puede observar como el UAV reduce la distancia respecto al destino de forma lineal, manteniendo constante la altura. Este comportamiento de navegación es identificativo del estado z-blocked.

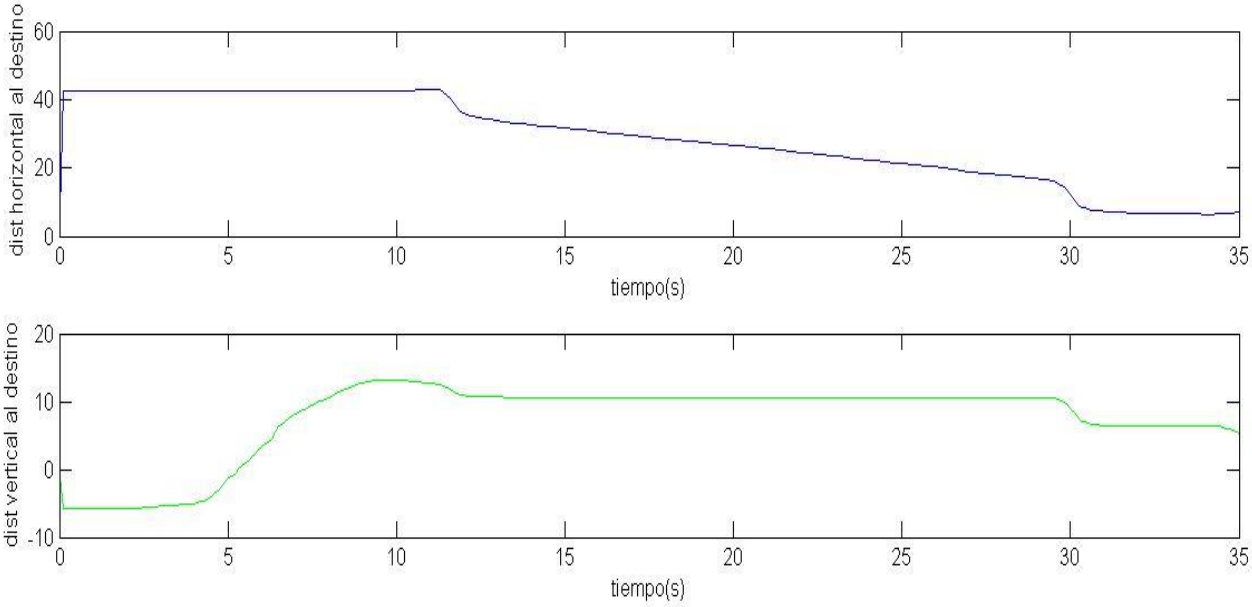

Ilustración 37 - Distancia entre UAVs en simulaciones de alturas

6.5 Simulación UAV bloqueado

Mediante esta simulación se pretende comprobar que un UAV rodeado de obstáculos permanece en estado bloqueado a la espera de que alguno de los obstáculos deje su posición, permitiéndole así salir del bloqueo y disponer de un camino para evitar los demás obstáculos que existan a su alrededor.

Para esto, se simulará un UAV principal rodeado de tres UAVs que permanezcan en una posición que haga que el vehículo objeto de la simulación no tenga ningún camino libre por donde realizar el movimiento de evitación. Posteriormente, uno de los UAVs cambiará de posición y el UAV principal deberá evitar el grupo de obstáculos "saliendo" por el camino libre que había dejado el UAV que había cambiado de posición. A continuación, se muestra una secuencia de imágenes con la simulación en GAZEBO.

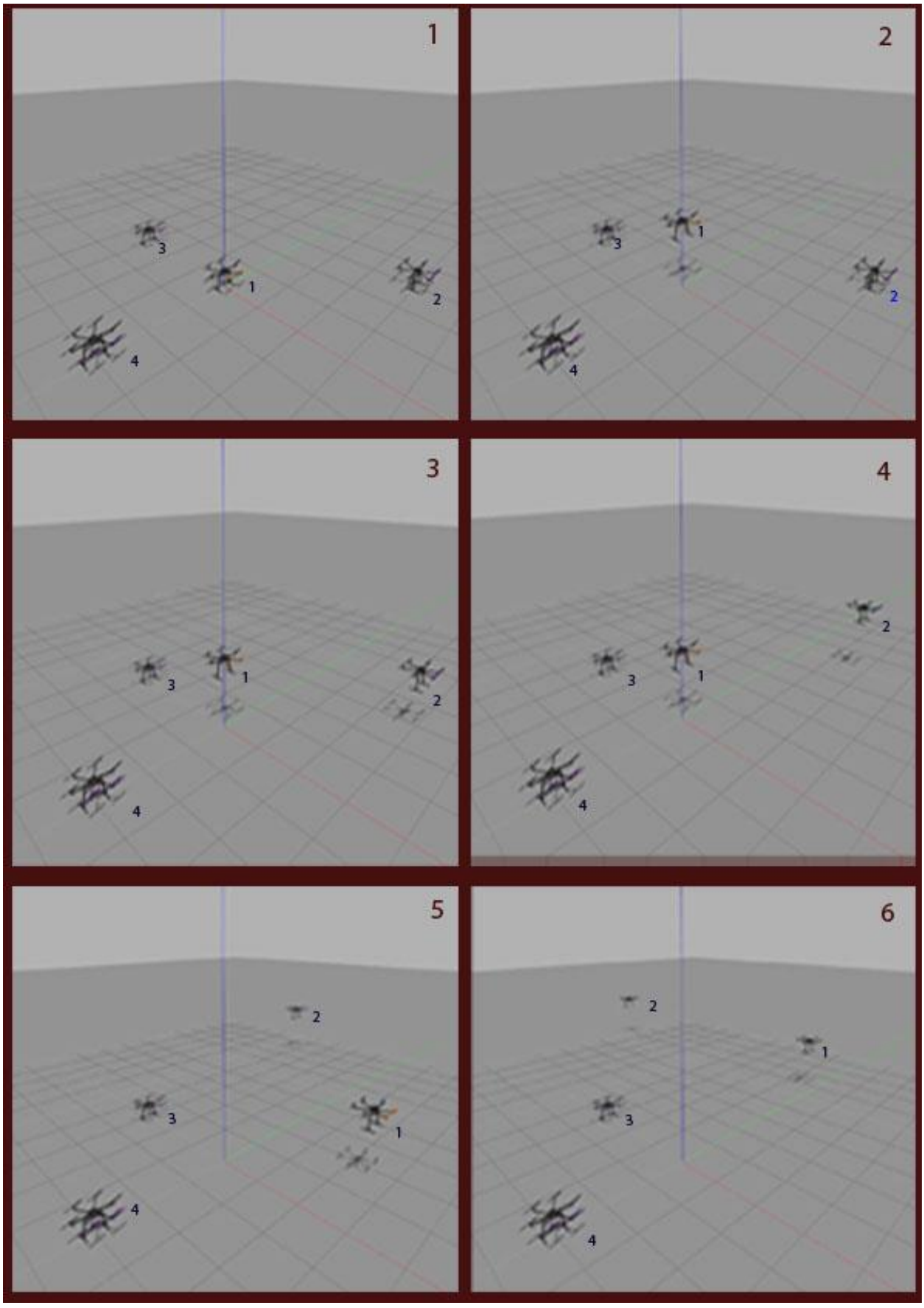

Ilustración 38 - Secuencia de imágenes en simulación de bloqueo

7 CONCLUSIONES

7.1 Conclusiones

En el estudio previo, realizado en el capítulo dos, para conocer el estado del problema de la evitación de colisiones en sistemas con múltiples vehículos aéreos no tripulados, se apreció que la mayoría de trabajos en la literatura eran teóricos y existían pocas implementaciones de los mismos. Por ello, se marcó como objetivo realizar una implementación del algoritmo en ROS, que posteriormente sería simulado en un entorno de simulación realista y que permitiría mostrar el funcionamiento de la implementación. Esta implementación se realizó con vistas a ser usada posteriormente en vehículos reales. Se decidió que la mejor forma de usar un algoritmo para funcionar en tiempo real era que fuera descentralizado y reactivo, sin cálculos excesivos que requirieran costes computacionales altos.

El algoritmo SWAP previo reúne las características necesarias para funcionar en tiempo real, mencionadas anteriormente. Por ello el algoritmo expuesto en este trabajo es una extensión del mismo.

El primer paso para realizar una implementación de este algoritmo fue exponer un planteamiento teórico que solucionase los problemas y necesidades de los vehículos aéreos no tripulados. Este planteamiento teórico, expuesto en el capítulo tres, incluyó la extensión del algoritmo tanto para funcionar en tres dimensiones como para funcionar en vehículos con características dinámicas propias de los UAV.

Se realizó una implementación del algoritmo de evitación en ROS, que funcionara de la mano de un nodo que ejecutara la misión. Se utilizó este último para realizar misiones en simulación que mostraran el comportamiento de múltiples UAVs ejecutando el algoritmo de forma descentralizada. Para ello se comandaron distintos puntos de referencia, forzando las siguientes situaciones:

- Múltiples UAVs realizando, de manera simultánea, el movimiento de evitación
- UAV frente obstáculo en forma de U
- Diferentes tipos de obstáculos a diferentes alturas
- UAVs rodeado de obstáculos sin camino libre para escapar

Se realizaron simulaciones en cada una de estas situaciones, extrayendo datos de variables de posición de los UAVs implicados y de los obstáculos fijos. Esto permitía comprobar el funcionamiento del algoritmo y de la implementación del mismo. Permitiendo sacar las siguientes conclusiones:

- Funciona de manera adecuada con múltiples vehículos, ejecutándolo al mismo tiempo y de manera descentralizada en cada uno de ellos. Al estar ejecutándose en cada uno de los vehículos, el movimiento de evitación se realiza más rápido, ya que intercambian sus posiciones. De ahí el termino SWAP.
- Es un algoritmo que trabaja bien en situaciones de riesgo que puedan provocar mínimos locales, como puede ser obstáculos en forma de U. Se comprobó que un UAV puede navegar en dirección contraria a su punto de destino para evitar el obstáculo.
- Cada UAV que ejecutó la implementación gestionó los obstáculos dependiendo de la altura en la que se encontrasen. Sin tener en cuenta los que estaban a demasiada altura, previniendo una posible colisión en altura navegando en el plano x-y o evitando el obstáculo.
- Se comprobó que cuando un UAV está rodeado de obstáculos, espera a que exista un camino libre para seguir realizando la misión.
- Además, se implementó un láser, comprobando la facilidad de este tipo de algoritmo para incluir la información de este tipo de sensores.
- Mediante la distancia recorrida y la visualización en GAZEBO, se comprobó que los UAVs no permanecían dando vueltas al obstáculo. Problema que apareció al inicio de la implementación.

- Se comprobaron los diferentes parámetros que harían aumentar la seguridad del vuelo. Siendo esto esencial para posteriores pruebas en el campo de vuelo.

Gracias a todas las comprobaciones realizadas en simulación, se concluyó que la extensión del algoritmo SWAP es válida para funcionar en 3D y, en concreto, en vehículos aéreos no tripulados.

7.2 Trabajos futuros

Una vez comprobado el funcionamiento de la implementación del algoritmo en un software realista, el trabajo futuro más atractivo sería probar este algoritmo en vehículos reales. Para esto se podrían realizar pruebas de forma gradual, es decir, eligiendo parámetros que hagan que nuestro sistema sea seguro. Por ejemplo, volando los UAVs a alturas que nunca interfieran en sus respectivos vuelos pero que cuando se crucen en el plano x-y realicen el movimiento de evitación. Esto se podría realizar dando un valor alto al parámetro relacionado con la altura del cilindro de aproximación. De esta manera se podría comprobar el movimiento de evitación en un vehículo real sin ponerlo en riesgo.

Aun así, todavía hay trabajos en simulación que son posibles trabajos futuros. Actualmente se encuentra en proceso añadir ruido a las medidas de las posiciones de los UAVs para ver su comportamiento en simulación. Esto haría aun más realista la simulación ya que en el vuelo real existirán ruidos de sensores como los de comunicación o posicionamiento GPS.

Otro trabajo, en el que la herramienta de simulación es fundamental, es incluir un sensor 3D para detectar obstáculos. Es un posible trabajo futuro que podría hacer que el vehículo evitara obstáculos solo con medidas locales.

REFERENCIAS

[1] M. Lao y J. Tang, «Cooperative Multi-UAV Collision Avoidance Based on Distributed Dynamic Optimization and Causal Analysis».

[2] J. Tang, Y. Tianyuan, L. Bai y S. Lao, «Collision Avoidance for Cooperative UAVs with Rolling Optimization Algorithm Based on Predictive State Space,» *MDPI*, 2017.

[3] J. Sun y S. Lao, «Collision Avoidance for Cooperative UAVs with Optimized Artificial Potencial Field Algorithm,» 2016.

[4] J.K. Archibald, J.C. Hill, N.A Jepsen, W.C. Stirling y R.L. Frost, «A satisficing Approach to Aircraft Conflict Resolution,» Brigham Young University.

[5] J.P. van den Berg, M. Lin y D. Manocha, «Reciprocal velocity obstacles for real-time multi-agent navigation,» *ICRA*, pp. 1928-1935, 2008.

[6] J. van der Berg, S. J. Guy, M. C. Lin y D. Manocha, «Reciprocal n-body Collision Avoidance,» de *INTERNATIONAL SYMPOSIUM ON ROBTICS RESEARCH*, 2009.

[7] M. D., S. G. Guy, J. Snape, M. C. Lin y J. v. d. Berg, «RVO2 Library: Reciprocal Collision Avoidance for Real-Time Multi-Agent Simulation».

[8] J. Meyer, «Hector quadrotor ros package website,» www., Available: http://wiki.ros.org/hector_quadrotor, 2014.

[9] D. Alejo, J. A. Cobano, G. Heredia y A. Ollero, «Optimal Reciprocal Collision Avoidance with Mobile and Static Obstacles for Multi-UAV System,» de *International Conference on Unmanned Aircraft System (ICUAS)*, Orlando, FL, USA, 2014.

[10] Z. A. Daniels, L. Wright, J. M. Holt y S. Biaz, «Collision Avoidance of Multiple UAS Using a Collision Cone-Based Cost Function,» 2012.

[11] E. Ferrera, J. Capitán, Á. R. Castaño y P. J. Marrón, «Descentralized safe conflict resolution for multiple robots in dense scenarios,» *ELSEVIER*, 2017.

[12] L. Zhongjie y L. Hugh, «Consensus based on learning game theory with a UAV rendezvouz application,» *CSAA*, 2015.

[13] Rutchi Jason, R. Senkbeil, J. Carrol, J. Dickinson, J. Holt y S. Biaz, «UAV Collision Avoidance Using Artificial Potencial Fields».