eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática

Grado de Ingeniería Informática

▪ **T**rabajo **F**in de **G**rado ▪

Computación

Desarrollo de un sistema de control de un espacio aéreo para múltiples drones autónomos en entorno simulado

---

Jon Garciandia Garaño

Junio 2022

eman ta zabal zazu
Universidad del País Vasco
Euskal Herriko Unibertsitatea
INFORMATIKA FAKULTATEA
FACULTAD DE INFORMÁTICA

Facultad de Informática

Grado de Ingeniería Informática

▪ **T**rabajo **F**in de **G**rado ▪

Computación

Desarrollo de un sistema de control de un espacio aéreo para múltiples drones autónomos en entorno simulado

---

Jon Garciandia Garaño

Junio 2022

Dirección

Manuel Graña Romay

Resumen

La utilidad de los vehículos aéreos no tripulados (UAV) ha ido aumentando significativamente en aplicaciones militares y en diferentes industrias civiles. Especialmente se están considerando seriamente en la logística y distribución en ambientes *outdoor* e *indoor* en el caso de grandes factorías. A medida que la tecnología y las políticas continúan desarrollándose, es probable que esta inserción sea cada vez mayor.

En este proyecto hemos estudiado el problema de la gestión de un espacio aéreo congestionado en el que navegan múltiples drones transportando personas u objetos con destinos diversos e independientes. No se considera que los drones estén cooperando sino que funcionan de forma independiente, con la única restricción de la evitación de colisiones. El objetivo a largo plazo es el desarrollo de sistemas de control autónomos que garanticen la navegación segura en todo tipo de condiciones. El presente proyecto presenta un paso adelante en la medida en que proporciona una plataforma de simulación en la que podemos experimentar con diversas soluciones de forma verosímil. Se presentan en el proyecto resultados de dos alternativas de control.

Para los experimentos hemos creado un entorno de simulación en MatLab. Hemos subido todo el código a zenodo.org: https://zenodo.org/record/6683833#.YrMjnpb9CUk

Índice

Lista de Figuras y Tablas

FIGURAS

TABLAS

1

Introducción y definiciones

1.1. Introducción

Este proyecto se enmarca en una línea de trabajo que tiene por objetivo final desarrollar algoritmos de control robustos que permitan a una población de drones navegar de forma segura en un espacio aéreo congestionado.

El objetivo específico de este proyecto es simular el comportamiento de múltiples drones en un espacio aéreo congestionado. El objetivo final del proyecto es que todos los drones puedan navegar por el espacio de forma simultánea minimizando el riesgo de colisión entre ellos. Para ello hemos creado un entorno de simulación en matlab apoyado en diferentes *toolboxes* (*UAV toolbox*, *Simulink*, *Navigation toolbox*, *aerospace toolbox* y *aerospace blockset*). Además, para poder apreciar mejor el movimiento de los drones, hemos creado un entorno de realidad virtual de simulación 3D.

Primeramente, hemos analizado el comportamiento de los drones con un sistema de control reactivo muy sencillo basado solamente en sensores (LIDAR). Hemos realizado diferentes experimentos para comprobar el funcionamiento de este sistema.

En segundo lugar, buscando mejorar el comportamiento de los drones hemos creado un algoritmo de planificación de trayectorias basado en el algoritmo A*, este algoritmo utiliza una estación de control terrestre central para la comunicación entre drones de la situación actual del conjunto completo de drones. El objetivo de este algoritmo es reducir la probabilidad de colisión entre los drones y optimizar la trayectoria que van a seguir los drones.

Finalmente, hemos comparado los resultados de los experimentos de ambos algoritmos.

1.2. Definiciones

***UAV**:* Un vehículo aéreo no tripulado (VANT), UAV (del inglés unmanned aerial vehicle) comúnmente conocido como **dron**, hace referencia a una aeronave que vuela sin tripulación. Un dron es capaz de mantener de manera autónoma un nivel de vuelo controlado y sostenido, y propulsado por un motor de explosión, eléctrico o de reacción.

***Cuadricóptero**:* del inglés *Quadrotor*, es un helicóptero con cuatro rotores para su sostén y su propulsión. Para mantener el equilibrio dos hélices (contrarias) giran en un sentido y los otros dos en el contrario. El control de vuelo lo proporciona la variación independiente de la velocidad de cada rotor.

La figura 1.1. muestra la arquitectura de un dron cuadricóptero con rotores motorizados con servos eléctricos y alimentado con baterías:

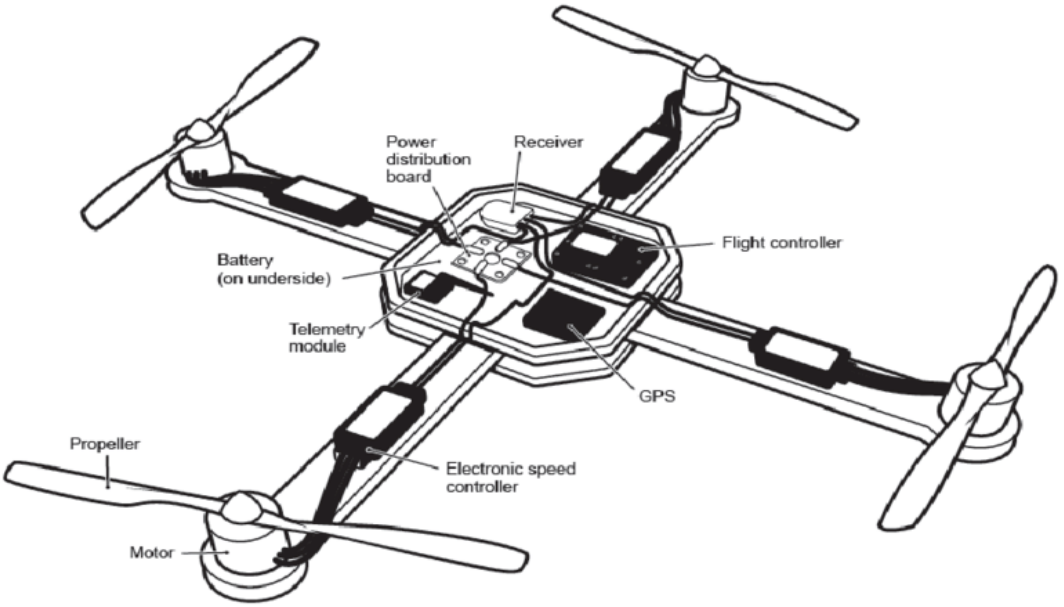

Figura 1.1 Arquitectura del dron cuadricóptero [1]

Sensor LIDAR: es un dispositivo que permite determinar la distancia desde un emisor láser a un objeto o superficie utilizando un haz láser pulsado. La distancia al objeto se determina midiendo el tiempo de retraso entre la emisión del pulso y su detección a través de la señal reflejada.

Un cuadricóptero es libre de rotar en tres dimensiones: *yaw* (guiñada), *pitch* (cabeceo) y *roll* (alabeo). El *yaw* se ajusta aplicando más revoluciones a los rotores que giran en la misma dirección, en cambio el *pitch* y el *roll* se ajustan aplicando diferencia de revoluciones en un par de rotores que giran al mismo lado.

En la figura 1.2 se pueden apreciar cuales son los ejes de rotación:

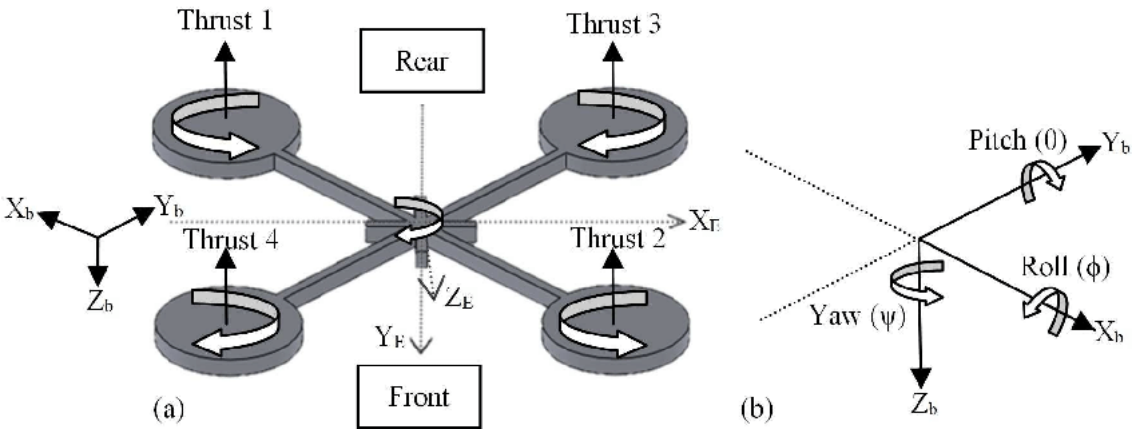

Figura 1.2. Ejes de rotación del dron cuadricóptero [7]

1.3. Software

Para simular el funcionamiento de los drones hemos utilizado el software Matlab, la versión R2021b. Matlab contiene numerosas *ToolBoxes* que facilitan la implementación de ciertas aplicaciones.

Estas son las *ToolBox* que hemos utilizado:

UAV Toolbox: proporciona herramientas y aplicaciones de referencia para diseñar, simular, probar y desplegar vehículos aéreos no tripulados (UAV) y aplicaciones de drones.

Simulink*:* un entorno de diagramas de bloque que se utiliza para diseñar sistemas con modelos multidominio, simular antes de implementar en hardware y desplegar sin necesidad de escribir código.

Simulink 3D Animation*:* enlaza modelos de simulink y algoritmos de matlab con objetos gráficos 3D en escenas de realidad virtual para mejorar la visualización de las simulaciones.

Navigation ToolBox*:* proporciona algoritmos y herramientas de análisis para planificación del movimiento, localización y mapeo simultáneos.

***Aerospace Toolbox y Aerospace Blockset**:* ofrecen herramientas y funciones basadas en estándares para analizar el movimiento, la misión y el entorno de vehículos aeroespaciales.

Hemos utilizado algunas de las funciones que explican en el ejemplo de *UAV Obstacle Avoidance in Simulink* [16] :

- *uavScenario*(): para crear el escenario donde estarán todos los drones.
- *uavPlatform*(): para crear la plataforma UAV, cada dron tendrá una plataforma. Hay que especificar el escenario donde se colocan, la posición inicial y la orientación inicial del dron.
- *updateMesh()*: para añadir el dron a la visualización de la simulación. Hay que especificar que tipo de dron queremos, en nuestro caso un cuadricóptero.
- *uavLidarPointCloudGenerator()*: crea un sensor LIDAR, hay que especificar el rango máximo, rango horizontal, rango vertical, resolución horizontal, resolución vertical y la frecuencia de muestreo.
- *uavSensor()*: añade el sensor LIDAR a la plataforma UAV.
- *addMesh()*: añade obstáculos al escenario, hay que especificar las dimensiones del obstáculo.
- *show3D()*: muestra una previsualización del escenario.

Además hemos utilizado la librería *networkx*[1] de Python (3.8.3) para calcular la trayectoria más corta entre dos puntos mediante el algoritmo A*.

[1] NetworkX es un paquete de Python para la creación, manipulación y estudio de la estructura, dinámica y funciones de redes complejas.

2

Planificación y Gestión del Proyecto

2.1. Tareas y diagrama de Gantt

Este proyecto se ha dividido en diferentes tareas:

- **Aprender el funcionamiento de MatLab y Simulink**: aprender a utilizar un nuevo lenguaje de programación y relacionarse con sus funcionalidades.
- **Estudiar y analizar ToolBoxes**: estudiar y analizar el funcionamiento de las ToolBoxes de MatLab utilizados en la simulación (*UAV toolbox*, *Simulink*, *Navigation toolbox*, *aerospace toolbox* y *aerospace blockset*).
- **Buscar documentación sobre el tema**: buscar y leer información relevante sobre el tema, se han utilizado herramientas como *Google Scholar*, *Research Gate* …
- **Analizar e implementar el algoritmo de comunicación mediante sensores**: analizar e implementar el algoritmo utilizado para la simulación.
- **Crear el entorno de simulación:** crear el entorno de simulación de simulink donde se realizan todos los experimentos.
- **Crear el entorno de simulación 3D**: crear el entorno de realidad virtual en 3D para mejorar la visualización de la simulación.
- **Implementar el algoritmo de control mediante A***: crear un algoritmo de control para evitar la colisión entre drones y optimizar la trayectoria basado en el algoritmo A*.
- **Realizar los experimentos**: realizar diferentes experimentos para comprobar el funcionamiento de los algoritmos.
- **Escribir la memoria**: escribir este documento.

En la figura 2.1 podemos ver el diagrama de Gantt que resume el tiempo empleado para cada una de las tareas.

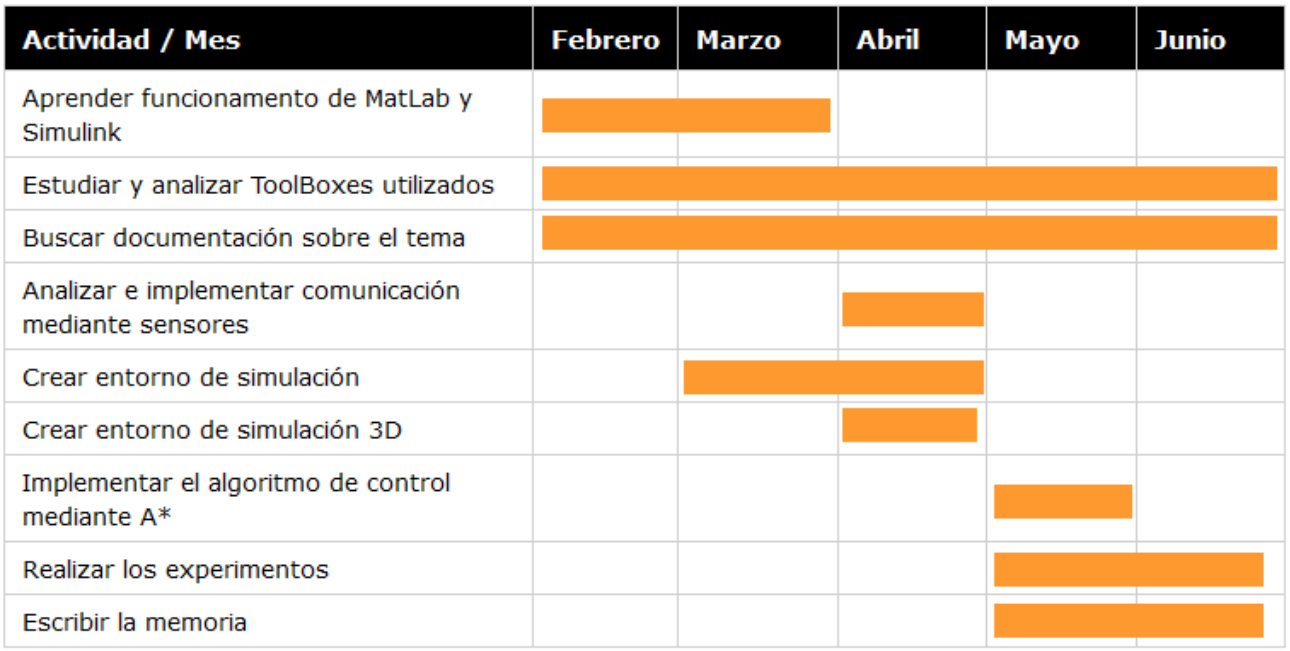

Figura 2.1. Diagrama de Gantt

Además de estas tareas se ha realizado un póster que tiene como objetivo representar el trabajo realizado en una sola imagen y una presentación que se realizará ante el tribunal para que valoren el trabajo y los objetivos conseguidos en este proyecto.

2.2. Análisis de riesgos

En este tipo de proyectos suelen haber riesgos que pueden introducir desviaciones en la planificación del proyecto, para evitar esto hemos intentado prever todos los posibles riesgos que puedan haber. Estos son los riesgos más significativos que hemos encontrado y sus correspondientes soluciones que proponemos:

- **Recursos de hardware limitados**: la simulación requiere grandes costos computacionales, y a veces no es suficiente con las capacidades del ordenador de casa. Para solucionarlo el tutor me ha habilitado el acceso a un aula de la facultad de informática con ordenadores más potentes.
- **Pérdida de datos:** tener todo el contenido creado en un solo ordenador puede ser peligroso, para evitar esto hemos ido guardando el trabajo realizado en *GoogleDrive*.
- **Covid-19**: dada la situación actual en cualquier momento se podría decretar la alerta sanitaria, por ello, hemos tenido que estar preparados para realizar el trabajo plenamente desde casa.
- **Tiempo:** Hemos tenido cinco meses para realizar el proyecto, por ello, es muy importante trabajar día a día y llevar las tareas al día.

3

Entorno de simulación

3.1 Especificaciones

Estas son las características de los elementos de *hardware* y *software* utilizados en la simulación:

UAV:

Tipo de UAV: drones cuadricóptero

Medidas: 60x60x10 cm

Peso: 0.1kg

Sensor LIDAR:

Rango máximo: 7 metros

Rango horizontal (-180, 180), 360 grados.

Rango vertical (-15 ,15), 30 grados.

Resolución horizontal: 0.5

Resolución vertical: 2

Frecuencia de muestreo: 10Hz

3.2 Simulación

Para hacer la simulación hemos utilizado el modelo de simulación de UAV-s de matlab. Este modelo permite crear una simulación a tiempo real y analizar el comportamiento de los drones.

Para simular el movimiento y la evitación de obstáculos de los drones hemos utilizado un algoritmo autónomo basado en las lecturas del sensor LIDAR, en la figura 3.3 se puede ver una imagen del diagrama de bloques de simulink. En el capítulo 4 se analiza en profundidad este algoritmo.

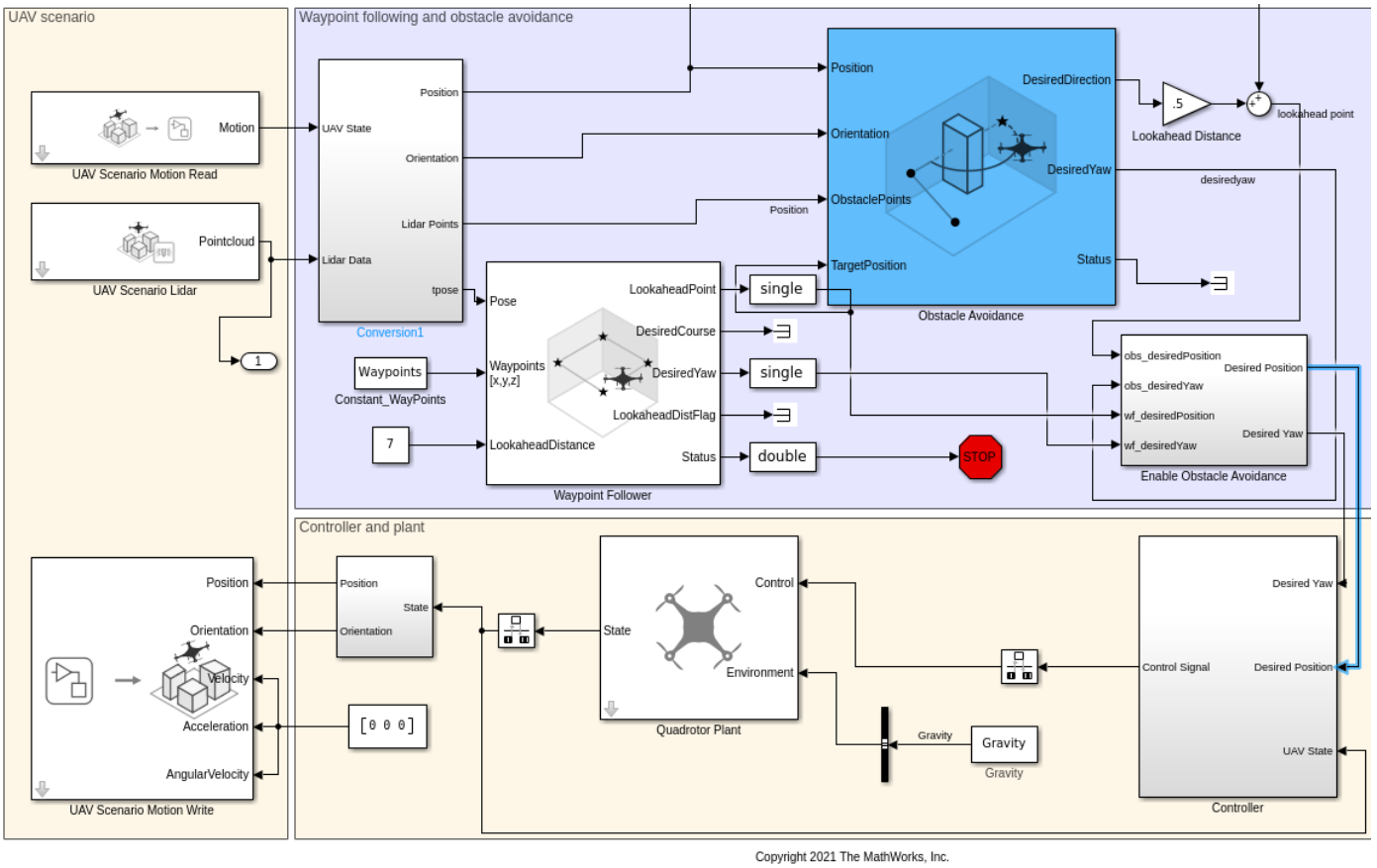

Figura 3.1. Diagrama de bloques de simulink del algoritmo simulación

Para simular el entorno de una nave, que puede corresponder a un almacén, hemos creado un escenario UAV mediante la función de matlab *uavScenario*, en este escenario se pueden añadir bloques (obstáculos) para simular un entorno realista.

El escenario tiene unas dimensiones de 60x40x10 metros y hemos añadido todo tipo de obstáculos de diferentes tipos para simular vigas, paredes, estanterías… En la figura 3.2 se puede ver una imagen del escenario de simulación.

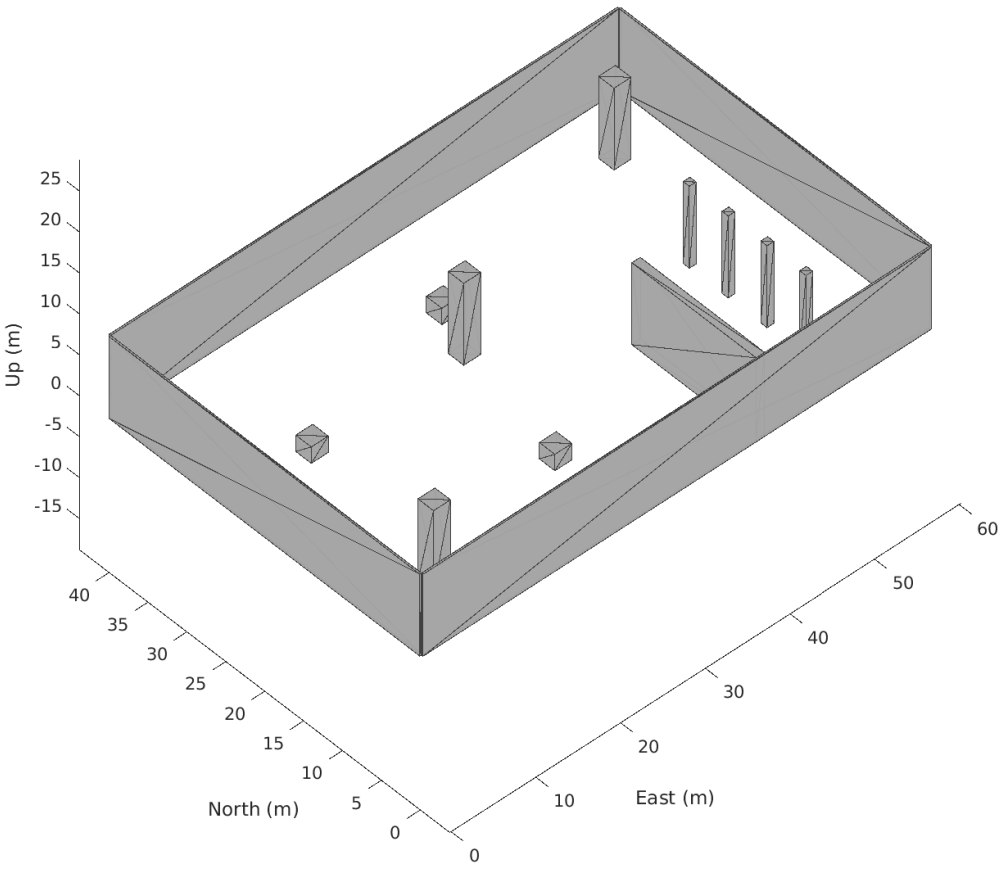

Figura 3.2. Escenario de simulación

3.3 Visualización 3D

Una vez terminada la simulación y calculados las posiciones y los ángulos de los drones en cada momento, hemos utilizado *Simulink 3D Animation* para examinar de forma más cómoda los resultados de la simulación. Esta herramienta nos permite crear un entorno con objetos gráficos 3D en escenas de realidad virtual. Hemos añadido texturas para darle más realismo. Además la visualización 3D nos permite tener diferentes ángulos de visión: visión general, vista superior, vista en primera persona de cada uno de los drones, etc.

En las figuras 3.3 y 3.4 se pueden apreciar snapshots con el resultado final del entorno de visualización 3D. Algunas visualizaciones se han capturado como videos que se han publicado en youtube.

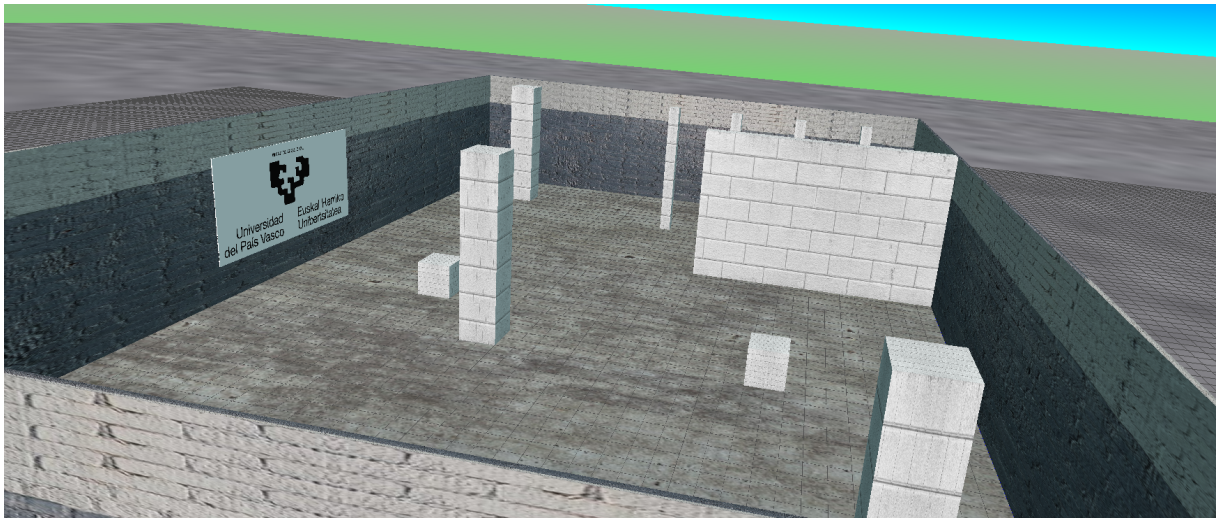

Figura 3.3. Entorno de simulación 3D

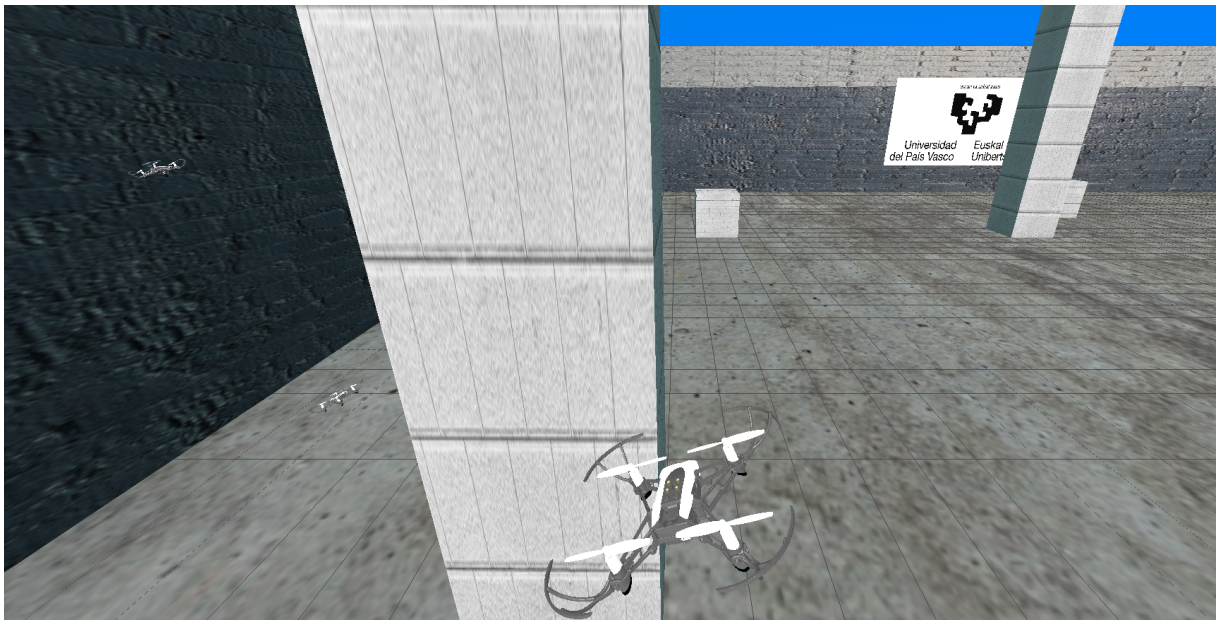

Figura 3.4. Vista primera persona simulación 3D

4

Control mediante sensores

4.1 Introducción

Hemos analizado el comportamiento de un sistema de control de drones autónomos basado en la información que recibe de los sensores. Esta configuración de control no utiliza ningún sistema de control centralizado que estaría situado en tierra, es decir, todas las decisiones que tomen los drones serán independientes entre sí. Los drones navegan de forma autónoma usando como información sobre el entorno únicamente las lecturas de los sensores a bordo. Hemos adaptado el ejemplo *UAV Obstacle Avoidance in Simulink* [16] para poder hacer la simulación con varios drones a la vez. La complejidad del modelado crece de forma exponencial con el número de drones.

El dron recibe la información de su entorno mediante los sensores (sensor de altitud, LIDAR, GPS…), calcula el siguiente punto en su trayectoria teniendo en cuenta la información de los sensores, la posición actual y la localización de destino. El control local calcula la velocidad de rotación de cada motor para llegar a ese punto.

En la figura 4.1 se muestra la cadena de decisión de este sistema compuesto de drones con control local autónomo.

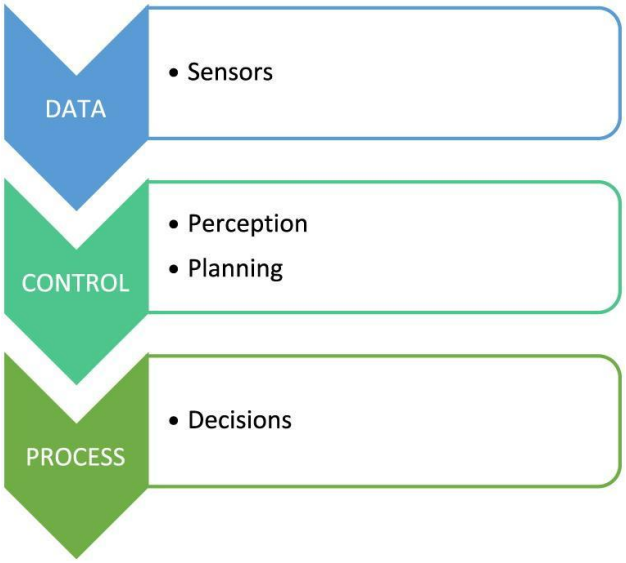

Figura 4.1. Cadena de decisión de dron autónomo con sensores [2]

4.2 Algoritmo

El UAV recibe cada 0.1 segundos los datos del sensor LIDAR. Calcula el siguiente punto al que tiene que ir (*desiredPosition*) aplicando las rutinas *Waypoint follower* y *Obstacle Avoidance*, y finalmente calcula los ángulos *yaw*, *pitch* y *roll* que necesita el dron para llegar a ese punto.

Una vez calculados los ángulos de rotación necesarios se pueden calcular las velocidades de rotación de cada motor.

En la figura 4.2 se puede ver pseudocódigo del algoritmo

$$
\begin{aligned}
&Posicion \leftarrow Posicion\_Inicial \\
&\textbf{while } Posicion \neq Posicion\_Final \textbf{ do} \\
&\quad LookAheadPoint \leftarrow WayPointFollower(Posicion) \\
&\quad desiredPoint \leftarrow ObstacleAvoidance(Posicion, LookAheadPoint, LIDAR) \\
&\quad yaw, pitch, roll \leftarrow getRotation(desiredPoint, Posicion) \\
&\quad Posicion \leftarrow getNewPosition(yaw, pitch, roll, Posicion) \\
&\textbf{end while}
\end{aligned}
$$

Figura 4.2. Pseudocódigo del algoritmo de control mediante sensores

4.2.1 WayPoint Follower

El *Waypoint Follower* es un algoritmo que calcula el siguiente punto al que tiene que ir el UAV (*Look Ahead Point*) siguiendo un conjunto de *waypoints* (puntos de referencia), que en este caso son solo el punto inicial y el punto de llegada, esto es, no se le proporciona una trayectoria deseada, entre los dos puntos, excepto la trivial línea recta.

Para calcular el punto *Look Ahead* hemos utilizado un método no lineal [3], que recibe como entrada los *waypoints,* la posición actual del UAV y el *Look Ahead Distance*, cuanto más pequeña sea el *Look Ahead Distance* el seguimiento de los *waypoints* será mejor pero si es muy pequeño puede provocar oscilaciones en la ruta.

4.2.2 Evitación de Obstáculos

Calcula un punto deseado (*desired Position)* teniendo en cuenta las lecturas del sensor LIDAR, la posición y la orientación actual del dron y el *Look Ahead Point* calculado en el *WayPoint Follower*.

Utiliza el algoritmo 3DVFH+ [4] para el cálculo del siguiente punto, este algoritmo se basa en el algoritmo de evitación de obstáculos 2D VFH+ [5] y utiliza el marco octomap[2] para representar el entorno tridimensional.

El algoritmo es capaz de calcular un nuevo punto para el UAV en un tiempo promedio de 300µs.

4.3 Experimentación

En este apartado analizaremos el funcionamiento de este algoritmo. Haremos tres experimentos diferentes, primero analizaremos la reacción de dos drones que se cruzan para llegar a sus destinos en el apartado 4.3.1, luego analizaremos el comportamiento de un dron que tiene que evitar una pared para llegar a su destino en el apartado 4.3.2, y finalmente analizaremos el comportamiento de diez drones moviéndose por el espacio en el apartado 4.3.3.

4.3.1 Cruce

En este apartado vamos a comprobar cómo se comportan los drones cuando se cruzan, vamos a colocar dos drones uno en el punto A y otro en el punto B, el dron A tendrá como objetivo llegar al punto B, y el dron B tendrá como objetivo llegar al punto A. O sea, están en trayectorias que inevitablemente colisionan.

Habrá un momento en el que los drones se van a cruzar, el objetivo es analizar la respuesta y la distancia que dejan los drones entre ellos para evitar la colisión.

Hemos colocado el dron A en el punto p_A=[10, 5, 4] y el dron B en el punto p_B=[20, 5, 4]. Al analizar la trayectoria los dos drones que se muestra en la figura 4.3, parece que los drones se evitan y no colisionan entre ellos, pero la distancia que dejan entre ellos es pequeña, llegan a estar a menos de 70cm, como se aprecia en el plot de la figura 4.4. Finalmente en la figura 4.5 se puede apreciar mejor la distancia mínima de los drones en la visualización 3D.

[2] La estructura de datos octomap [6] es una forma eficiente de representar un entorno 3D

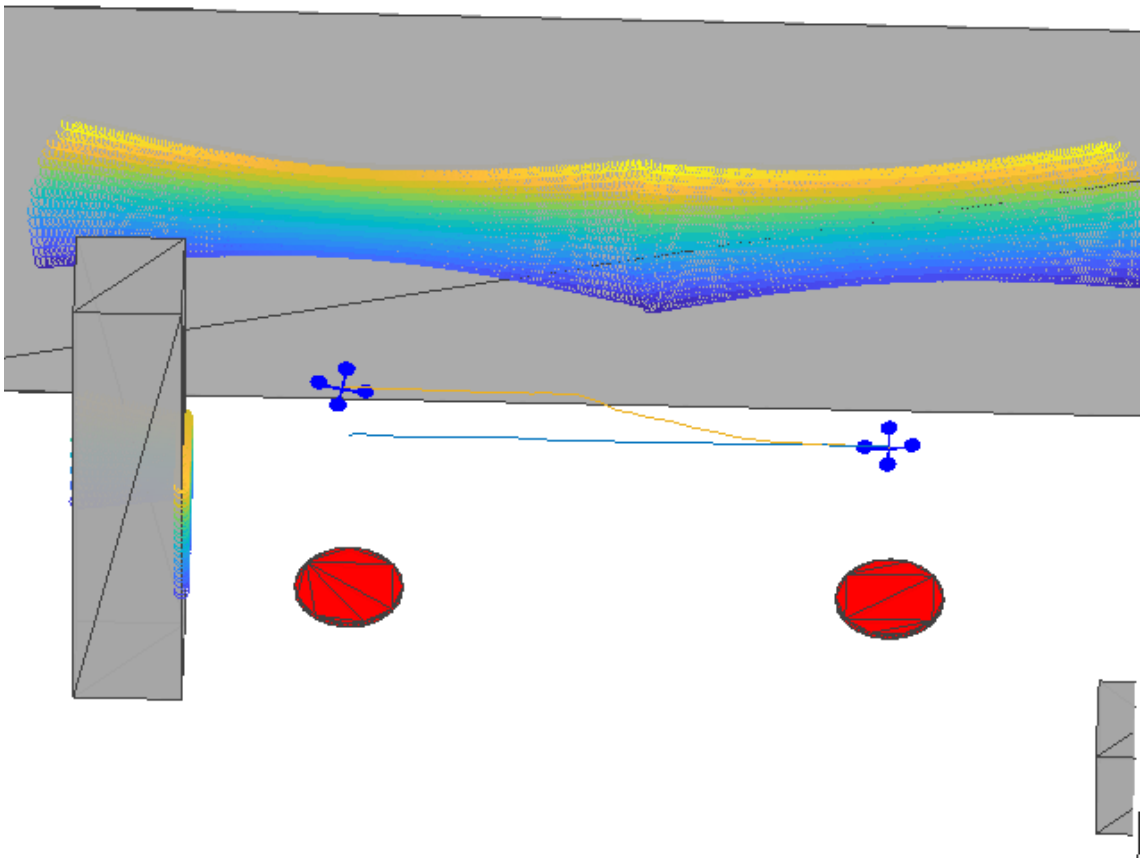

Figura 4.3. Trayectoria del experimento cruce, control únicamente mediante sensores a bordo

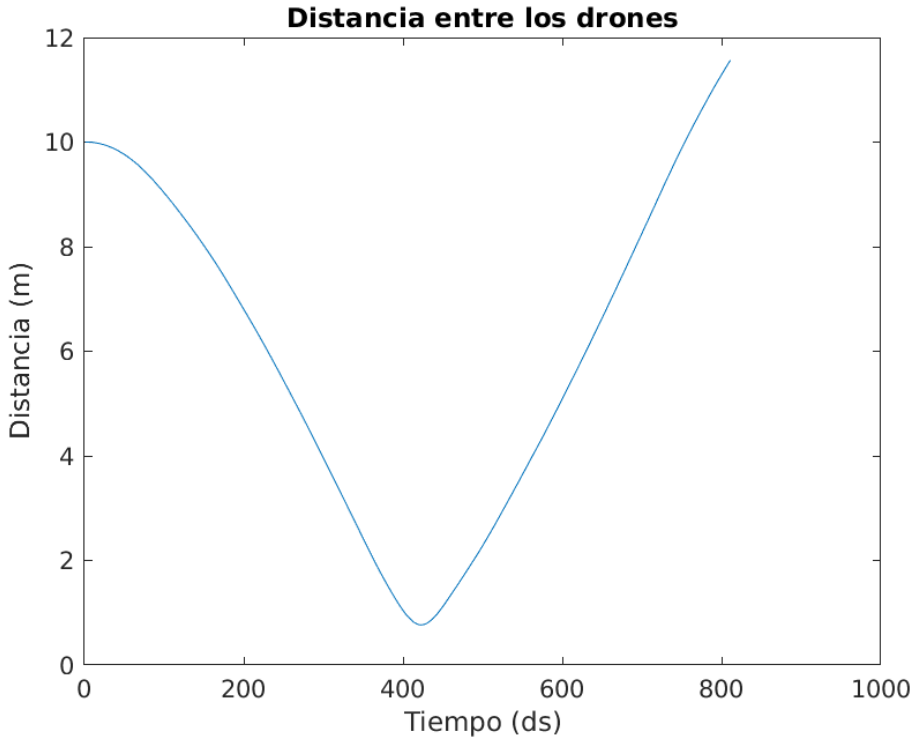

Figura 4.4. Distancia entre drones en el experimento cruce, control únicamente mediante sensores a bordo

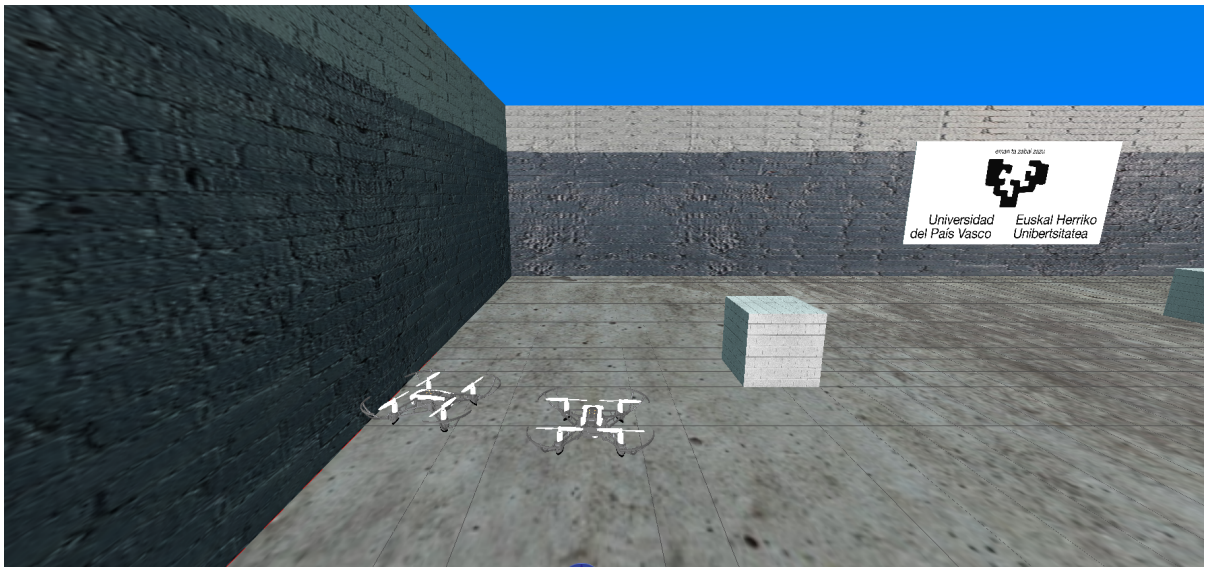

Figura 4.5. Distancia mínima entre los drones en el experimento cruce, control únicamente mediante sensores a bordo

4.3.2 Pared

En este experimento vamos a analizar el comportamiento del control autónomo basado en sensores locales cuando el dron tiene que bordear una pared para llegar a su destino. En este caso lo más importante es que llegue lo antes posible, es decir, que no pierda mucho tiempo encontrando la solución para pasar la pared.

Hemos colocado el dron en el punto p_A=[3, 38, 4] y su destino es el punto p_B=[3, 42, 4]. Como se puede ver en la figura 4.6 el dron no es capaz de bordear la pared para llegar al destino, se queda dando vueltas en el mismo sitio. Este comportamiento se explica de la siguiente manera: cuando el dron se aleja demasiado del punto de destino se da la vuelta porque la distancia por el otro lado es más pequeña. Al volver al origen esta condición cambia y el dron se queda oscilando.

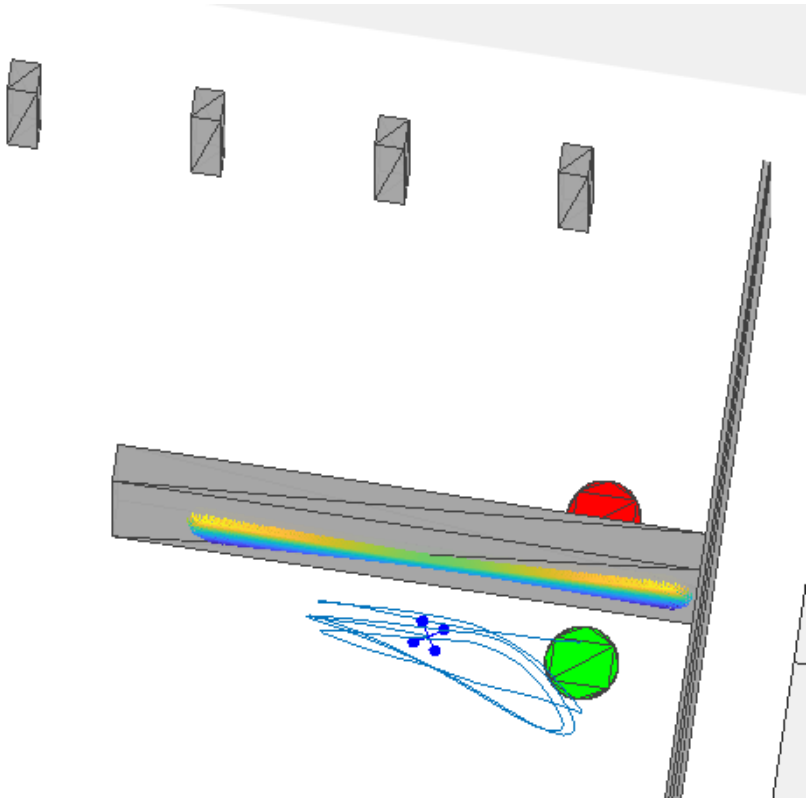

Figura 4.6. Trayectoria del experimento pared, control únicamente mediante sensores a bordo cuando el dron tiene que ir del punto verde al rojo.

4.3.3 Recorrido completo de una población de drones

En este último experimento vamos a analizar el comportamiento de 10 drones moviéndose por el espacio simultáneamente bajo control local usando únicamente sensores a bordo. El objetivo de este experimento es comprobar cómo se comportarían en un entorno de trabajo realista, en el que todos los drones se mueven a la vez por el mismo espacio.

Hemos colocado 5 drones en un extremo de la nave p={[10, 3, 4], [15, 3, 4], [20, 3, 4], [25, 3, 4], [30, 3, 4]} y otros 5 drones en el otro extremo de la nave p={[10, 57, 4], [15, 57, 4], [20, 57, 4], [25, 57, 4], [30, 57, 4]}. Cada dron tendrá como destino la posición simétricamente opuesta. Por ejemplo el dron que sale en la posición p_A=[10, 3, 4] tendrá como destino la posición p_B=[30, 57, 4]. En la figura 4.7 se puede ver la trayectoria que siguen los drones.

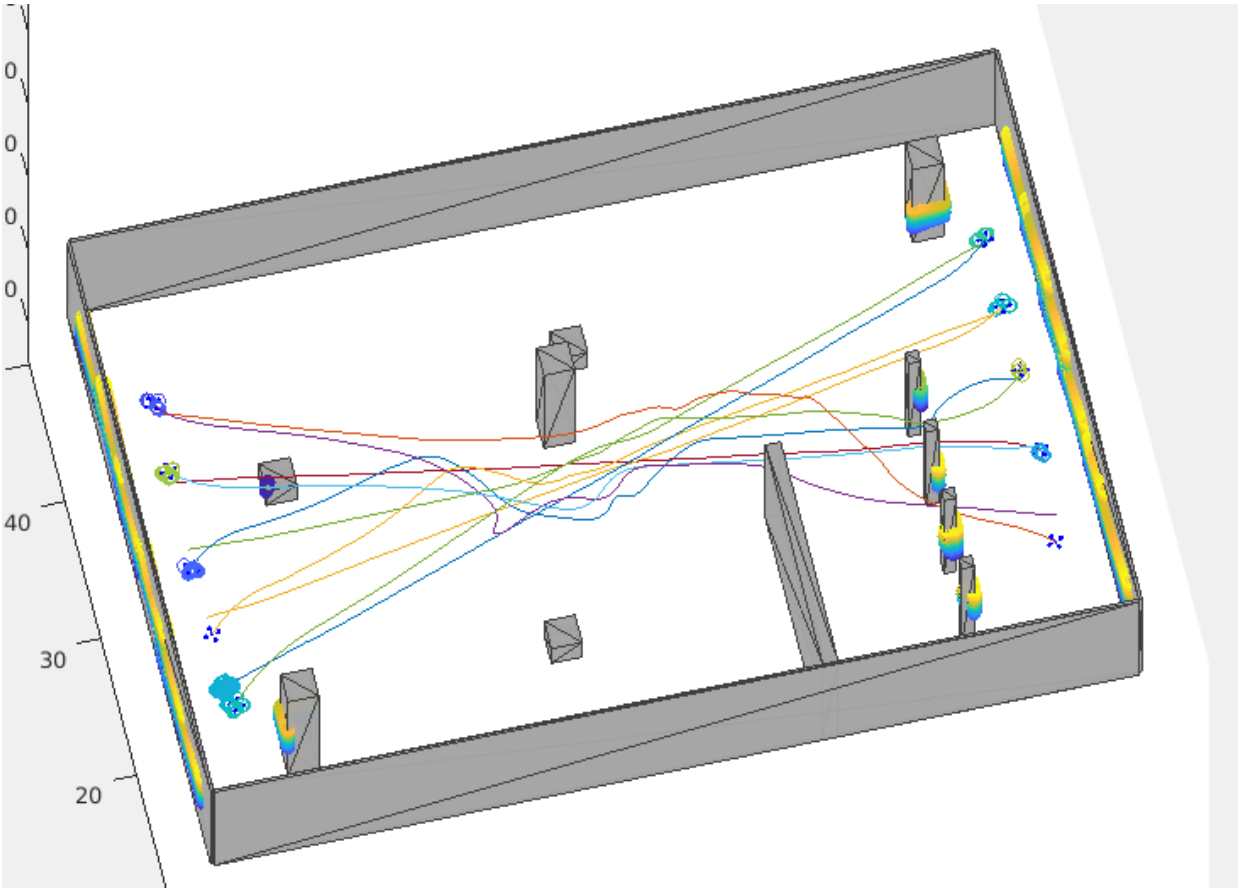

Figura 4.7. Trayectoria del experimento recorrido completo, control únicamente mediante sensores a bordo

Como se puede apreciar en la figura 4.7 todos los drones pasan por la misma zona para llegar a su destino, esto causa que los drones estén demasiado cerca como se aprecia en la figura 4.8. El riesgo de colisión es muy alto ya que los drones no tienes prácticamente espacio para maniobrar. Como se puede ver en la figura 4.8 los drones llegan a estar a menos de 15cm de distancia, esto quiere decir que más de un dron ha colisionado con otro. Finalmente, en la figura 4.9 se muestra una imagen de la visualización 3D cuando todos los drones están en medio de la nave. En el siguiente enlace se puede ver un video de la trayectoria de los drones: https://youtu.be/cTrZNpwdvG8

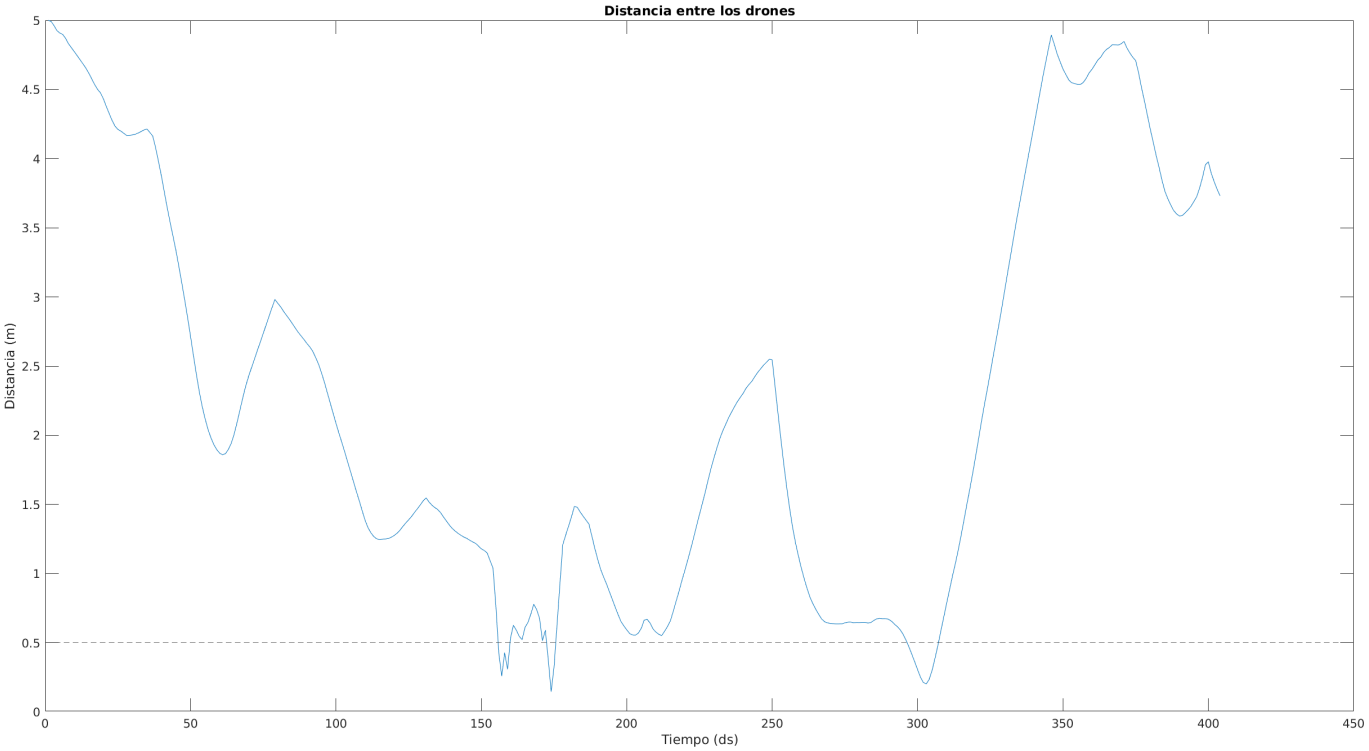

Figura 4.8. Distancia mínima entre drones en el experimento recorrido completo, control únicamente mediante sensores a bordo. Se considera como riesgo de colisión un umbral de 0.5 metros

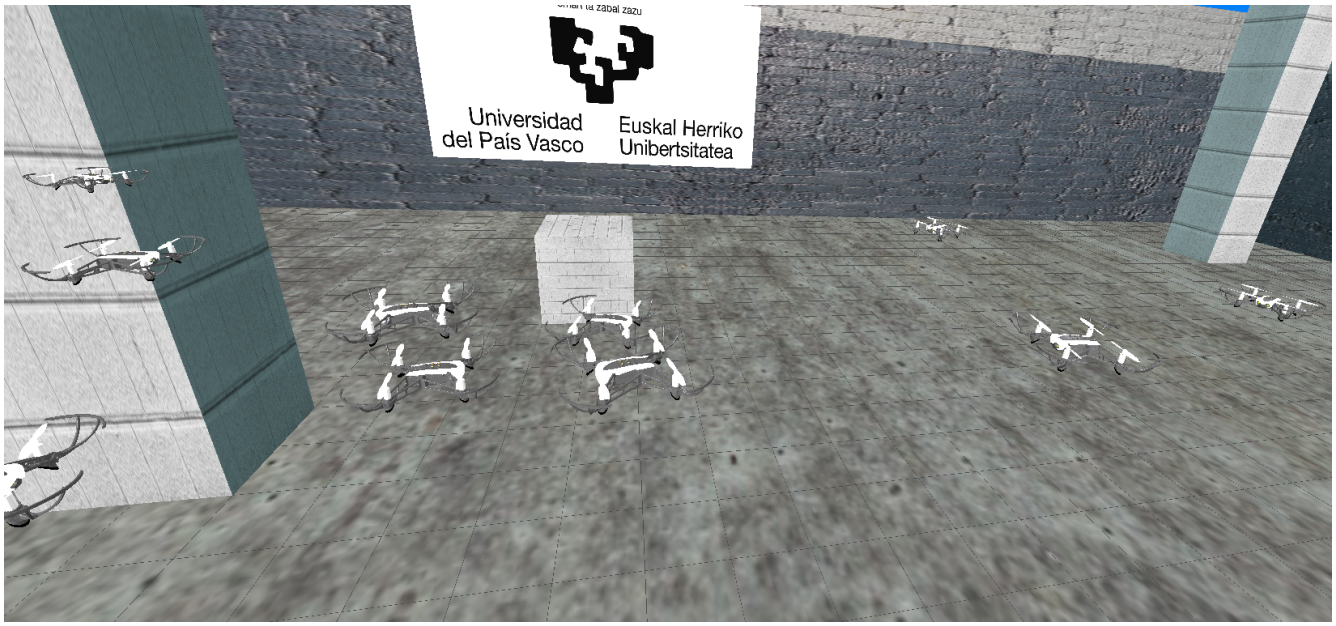

Figura 4.9. Visualización 3D del experimento recorrido completo, control únicamente mediante sensores a bordo

5

Control mediante A*

5.1 Introducción

En este apartado vamos a estudiar el comportamiento de un algoritmo creado para la evitación de colisiones entre drones que utiliza el algoritmo A* para generar un conjunto de trayectorias seguras.

Para este algoritmo utilizaremos una estación de control terrestre o *GCS* (*Ground Control Station)*, esta estación será la encargada de calcular la trayectoria óptima sin colisiones para cada uno de los drones.

Cada vez que uno de los drones tenga un nuevo destino enviará un mensaje a la estación de control terrestre, está calculará la ruta más eficiente para el dron y le enviará la trayectoria que tiene que seguir al dron. La trayectoria viene dada por un conjunto de *waypoints*. Una vez que el dron tenga la trayectoria simplemente tendrá que usar el algoritmo *Waypoint Follower* para seguir la trayectoria y llegar a su destino. La ventaja de este algoritmo es que al calcular la trayectoria de cada uno de los drones **la central terrestre tiene en cuenta la trayectoria de los demás drones**, lo que disminuye el riesgo de colisión entre drones y aumenta la eficiencia de los drones.

Por otro lado, para la utilización de este algoritmo es necesario conocer el mapa del entorno, es decir, conocer los obstáculos que hay en el entorno.

En la figura 5.13 se muestra la cadena de decisiones de este sistema.

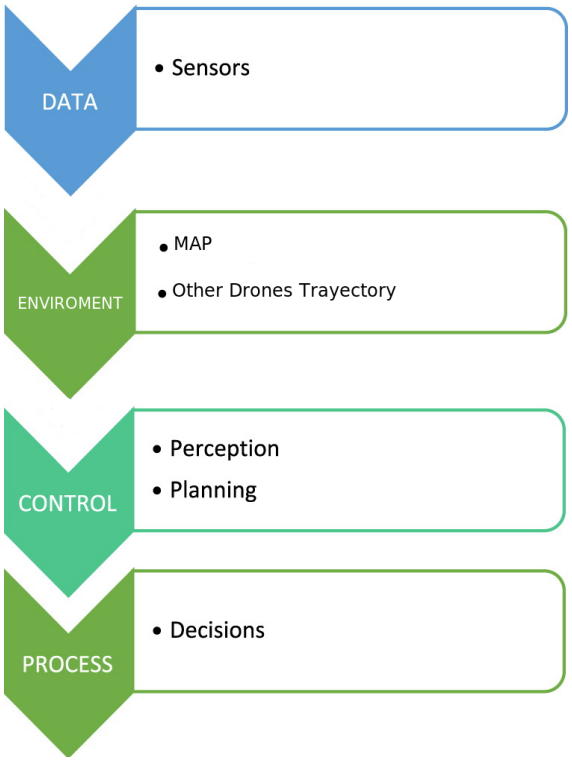

Figura 5.13. Cadena de decisiones, algoritmo control mediante A* para la generación de trayectorias seguras y eficientes

En el apartado 5.2 vamos a analizar el sistema de comunicación entre los drones y la estación de control terrestre, en el apartado 5.3 vamos a explicar el algoritmo utilizado para el cálculo de la trayectoria, en el apartado 5.4 vamos a ver los resultados obtenidos en los diferentes experimentos con este algoritmo y finalmente en el apartado 5.5 vamos a comparar los resultados de este algoritmo con el control local basado únicamente en sensores.

5.2 Comunicación entre drones y estación de control terrestre

Los drones y la estación de control central terrestre se comunican mediante una red Wifi 802.11n[3] de 5GHz [8]. La estación de control terrestre funciona como un servidor, el dron envía la localización actual del dron mediante el protocolo TCP, la estación de control terrestre calcula la trayectoria óptima para el dron y se la manda.

Los drones y la estación de control terrestre están conectados mediante una red en estrella, es decir, todos los drones están conectados con la estación de control terrestre pero los drones no están conectados entre ellos como se ilustra en la figura 5.1.

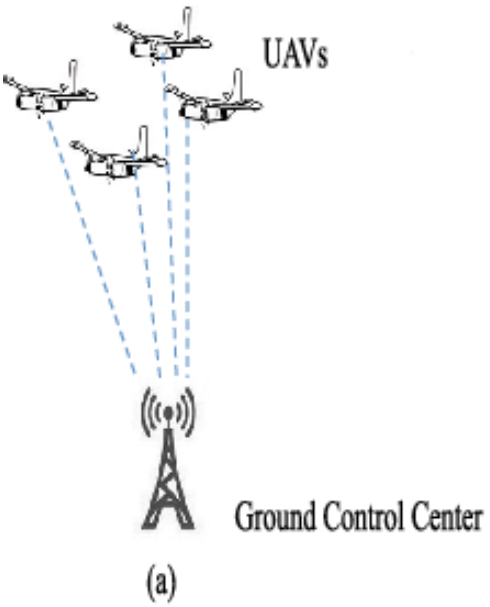

Figura 5.1. Topología estrella del control de estación terrestre [9]

[3] El estándar 802.11 es una familia de normas inalámbricas creada por el Institute of Electrical and Electronics Engineers (IEEE)

5.3 Algoritmo

Este algoritmo combina un mapa tridimensional creado mediante una red de grafos y el algoritmo A*.

Para crear el mapa utilizamos una red de grafos con 24.000 nodos (40x60x10), cada nodo representa un metro cúbico en el espacio, véase la figura 5.2.

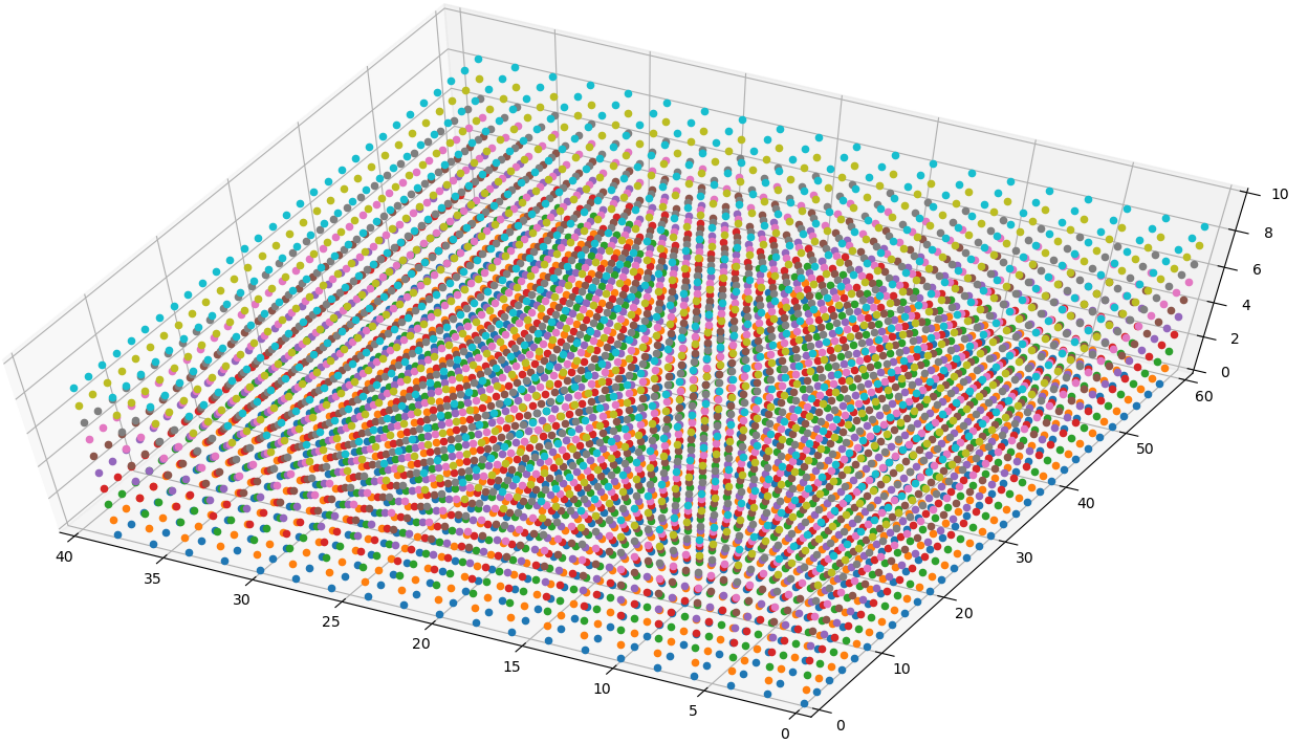

Figura 5.2. Nodos de la red de grafos del mapa 3D

Cada nodo tiene 26 vecinos diferentes [10], es decir, el UAV puede ir a 26 diferentes posiciones desde su posición actual. El coste para llegar a los vecinos se calcula mediante la distancia euclídea. Los nodos pueden estar a 1 metro (solo hay que moverse en un eje), 1.41 metros (dos ejes) y 1.73 metros (tres ejes).

Una vez que el mapa está creado hay que añadir los obstáculos, para ello simplemente hay que quitar todas las conexiones de los nodos donde esté el obstáculo. Por ejemplo, si queremos añadir un obstáculo en la posición [10, 10, 5], tenemos que desconectar todos los vecinos del nodo [10, 10, 5], así cuando el dron este cerca de la posición [10, 10, 5] nunca podrá ir ahí porque no existe ningún camino.

Para calcular la trayectoria de un dron de un punto A a otro punto B, utilizaremos el algoritmo A* [11]. A* es un algoritmo completo, en caso de existir una solución, siempre dará con ella.

Este algoritmo utiliza una función de evaluación f(n) = g(n) + h'(n), donde h'(n) representa el valor heurístico del nodo a evaluar desde el actual, n, hasta el final, y g(n), el costo real del camino recorrido para llegar a dicho nodo, n. A* mantiene dos estructuras de datos auxiliares, que podemos denominar abiertos, implementado como una cola de prioridad ordenada por el valor f(n) de cada nodo, y cerrados, donde se guarda la información de los nodos que ya han sido visitados. En cada paso del algoritmo, se expande el nodo que esté primero en abiertos, y en caso de que no sea un nodo objetivo, calcula la f(n) de todos sus hijos, los inserta en abiertos, y pasa el nodo evaluado a cerrados [12]. Para calcular el valor heurístico h'(n) utilizaremos la distancia euclídea.

Hemos utilizado la función *astar_path()* [13] de la librería *netowrkX* para calcular la trayectoria más corta mediante el algoritmo A*. La función solo necesita el grafo de localizaciones espaciales que hemos creado anteriormente y el nodo de inicio y el nodo de destino.

Una vez calculada la trayectoria óptima para llegar del punto A al punto B **se guarda la trayectoria como si fuera un obstáculo para que el siguiente dron no pase por el mismo lugar** y así bajar la probabilidad de que haya una colisión entre dos drones.

De este modo se calculan todas las trayectorias (*waypoints*) de todos los drones y se ejecuta el algoritmo *WayPointFollower,* descrito anteriormente en el apartado 4.2.1, para hacer la simulación.

En la figura 5.3 está el pseudocódigo del algoritmo completo, para calcular los *waypoints* de todos los drones.

$$
\begin{aligned}
&G \leftarrow crear\_mapa\_grafo()\\
&G \leftarrow añadir\_Obstaculos(G, obstaculos)\\
&WayPoints \leftarrow []\\
&\textbf{for } k \text{ in } Drons \textbf{ do}\\
&\quad wps \leftarrow a\_star(G, inicio[k], final[k])\\
&\quad \mathrm{G} \leftarrow añadir\_obstaculo(G, wps)\\
&\quad WayPoints[k] \leftarrow wps\\
&\textbf{end for}
\end{aligned}
$$

Figura 5.3. Pseudocódigo del algoritmo de control mediante A*

5.4 Experimentación

En este apartado analizaremos el funcionamiento de este algoritmo. Haremos los mismos tres experimentos que hemos hecho con el otro algoritmo, primero analizaremos la reacción de dos drones que se cruzan para llegar a sus destinos, apartado 5.4.1, luego analizaremos el comportamiento de un dron que tiene que evitar una pared para llegar a su destino, apartado 5.4.2, y finalmente analizaremos el comportamiento de diez drones moviéndose por el espacio, apartado 5.4.3.

5.4.1 Cruce

En este apartado vamos a comprobar cómo se comportan los drones cuando se cruzan, vamos a colocar dos drones uno en el punto A y otro en el punto B, el dron A tendrá como objetivo llegar al punto B, y el dron B tendrá como objetivo llegar al punto A.

Habrá un momento en el que los drones se van a cruzar el objetivo es analizar la respuesta y la distancia que dejan los drones entre ellos para evitar la colisión.

Hemos colocado el dron A en el punto p_A=[10, 5, 4] y el dron B en el punto p_B=[20, 5, 4].

Primero hemos calculado la trayectoria de los dos drones A y B mediante el algoritmo explicado en el apartado 5.3. Como se puede ver en la figura 5.4 el dron A (línea azul) usa el camino más rápido (línea recta) para llegar al punto B, en cambio el dron B, calcula su trayectoria (línea naranja) teniendo en cuenta la trayectoria del anterior dron, se desplaza a la derecha desde el principio para evitar la colisión.

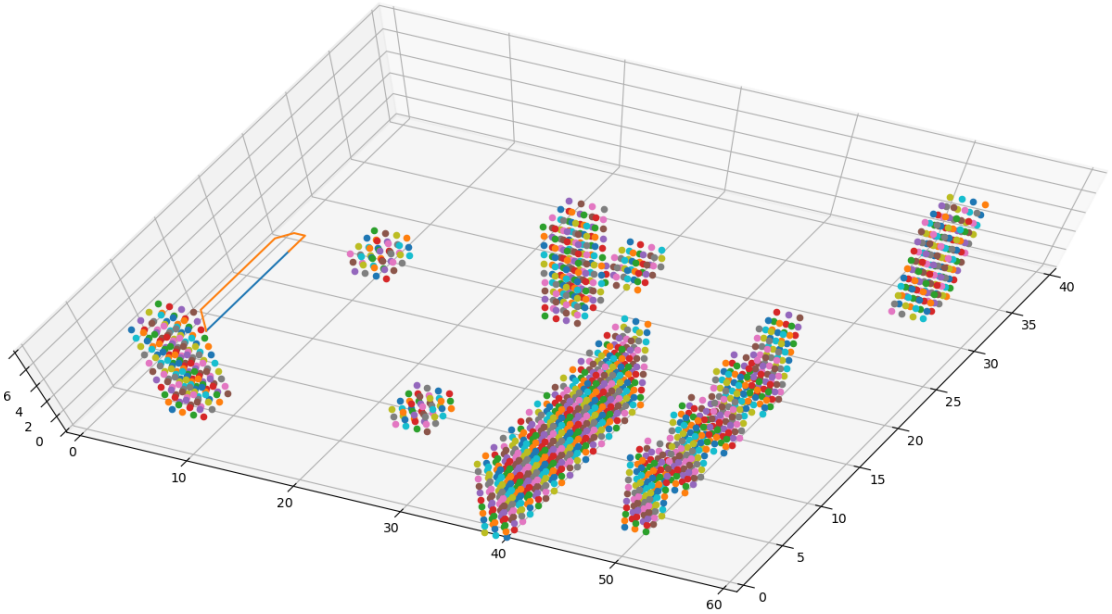

Figura 5.4. Trayectoria calculada por el algoritmo A* para el experimento cruce

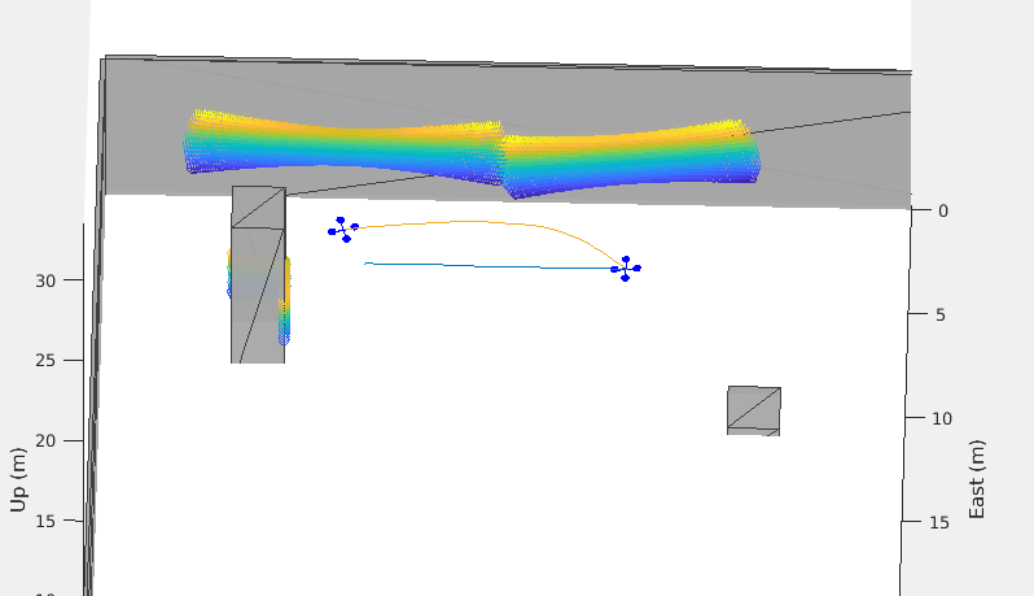

Figura 5.5. Trayectoria del experimento cruce, control mediante A*

Una vez calculada la trayectoria, procedemos a hacer la simulación usando como *waypoints* los puntos de la trayectoria, figura 5.5. Como se puede ver en la figura 5.5 los drones siguen la trayectoria calculada, el dron A sigue el camino recto y el dron B se desplaza hacia la derecha desde el primer momento para que no haya ningún riesgo de colisión.

Hemos calculado las distancias que hay entre los drones en todo el recorrido para confirmar que no hay ningún tipo de riesgo. Como se aprecia en la figura 5.6 la distancia mínima entre los drones es de 1.9 metros, es decir, no hay riesgo de colisión. Finalmente en la figura 5.7 se muestra una imagen de la simulación 3D donde se aprecia bien la distancia entre los dos drones.

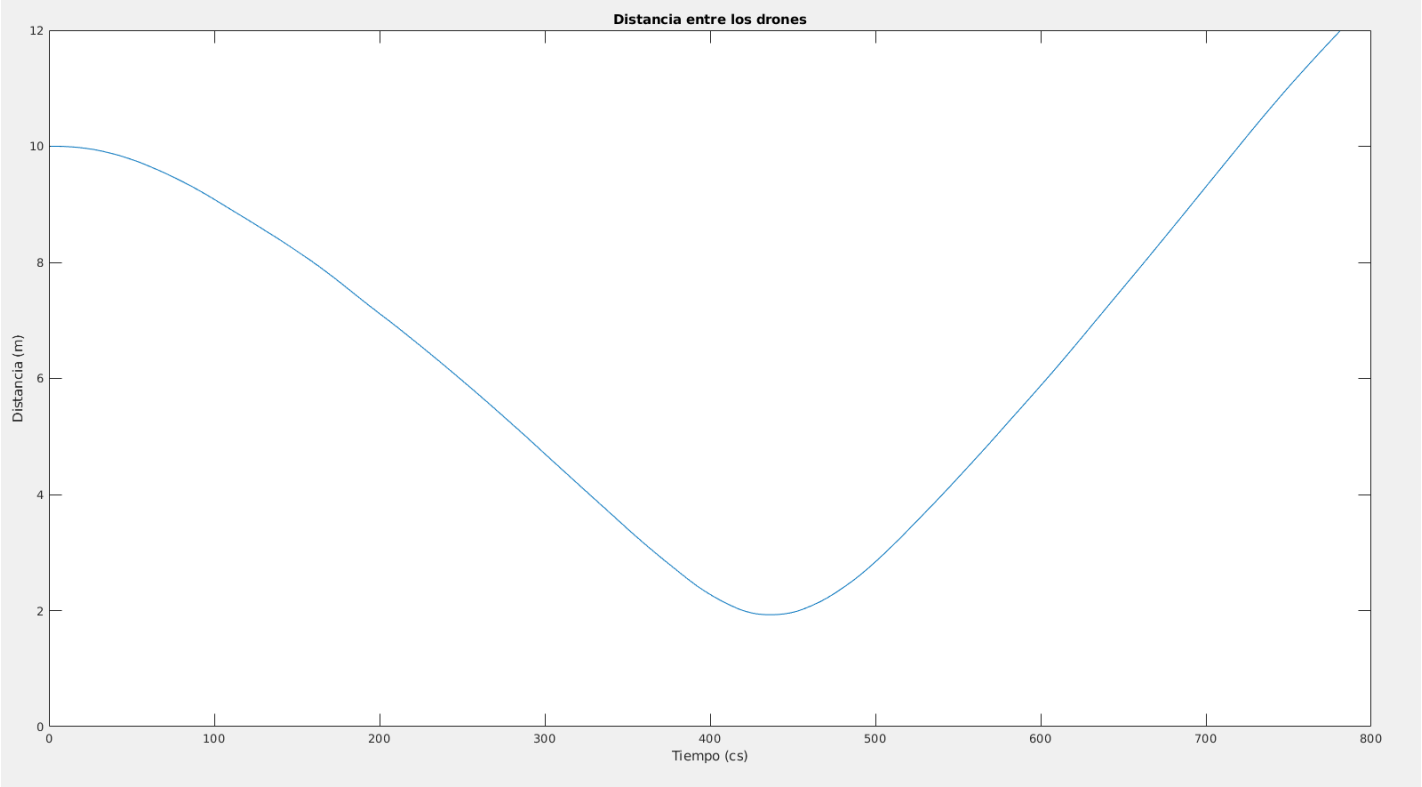

Figura 5.6. Distancia entre drones en el experimento cruce, control mediante A*

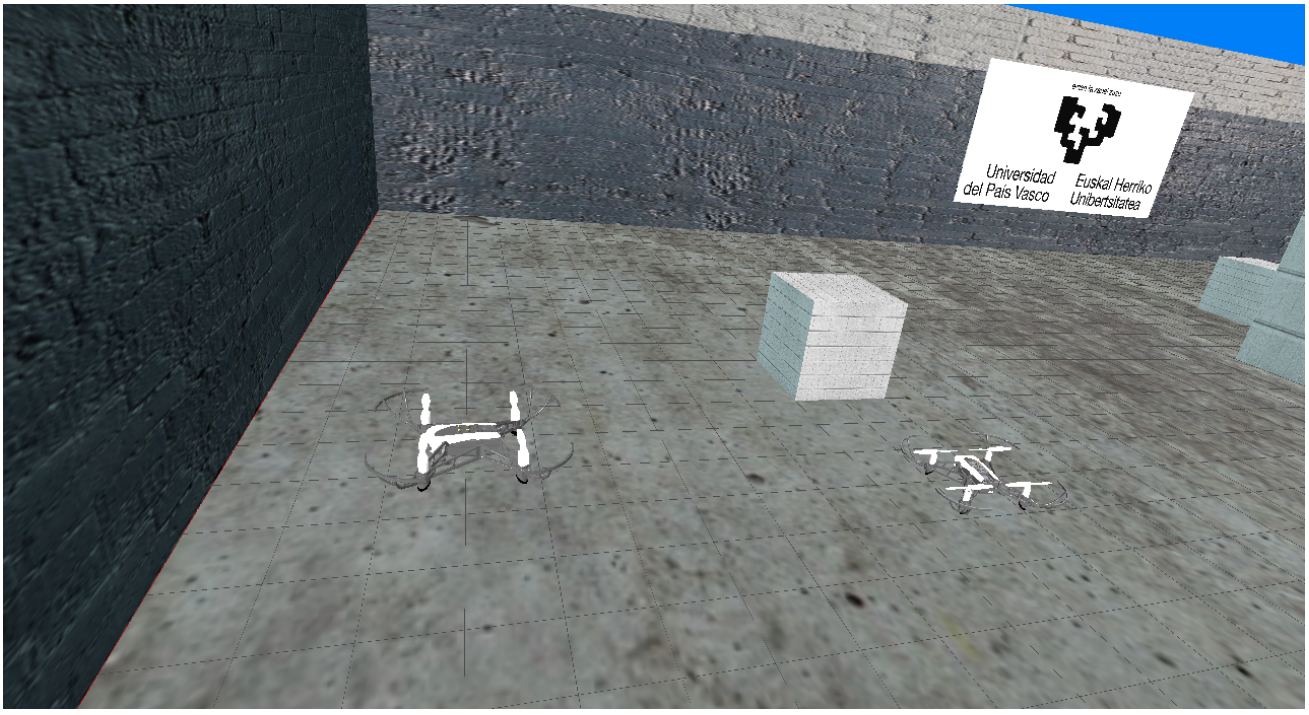

Figura 5.7. Visualización 3D experimento cruce, control mediante A*

5.4.2 Pared

En este experimento vamos a analizar el comportamiento de este algoritmo cuando el dron tiene que bordear una pared para llegar a su destino. En este caso lo más importante es que llegue lo antes posible, es decir, que no pierda mucho tiempo encontrando la solución para pasar la pared.

Hemos colocado el dron en el punto p_A=[3, 38, 4] y su destino es el punto p_B=[3, 42, 4].

Primero hemos calculado los *waypoints* que debe seguir el dron mediante el algoritmo explicado en el apartado 5.3. Estos son los puntos que ha calculado el algoritmo: *{(3, 38, 4), (7, 37, 4), (9, 37, 4), (11, 37, 4), (13, 37, 4), (15, 37, 4), (17, 38, 4), (17, 40, 4), (15, 41, 4), (13, 41, 4), (11, 41, 4), (9, 41, 4), (7, 41, 4), (5, 41, 4), (3, 42, 4)}*

Una vez calculados los *waypoints* simplemente falta hacer la simulación para ver el comportamiento del dron. En la figura 5.8 se puede ver la trayectoria creada por el dron, pasa la pared sin perder ningún tiempo y de la manera más efectiva.

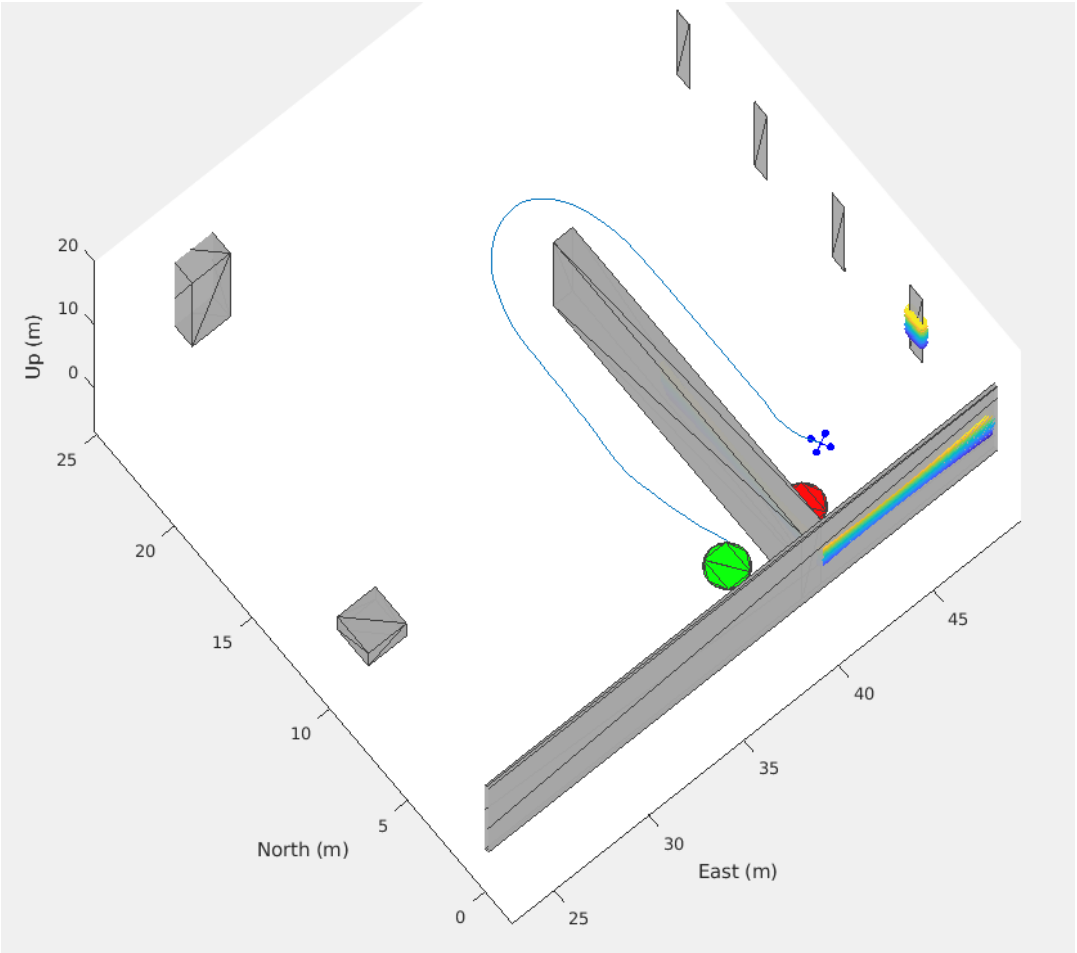

Figura 5.8. Trayectoria del experimento pared, control mediante A*

5.4.3 Recorrido completo de una población de drones

En este último experimento vamos a analizar el comportamiento de 10 drones moviéndose por el espacio simultáneamente. El objetivo de este experimento es comprobar cómo se comportarían en un entorno de trabajo real, en el que todos los drones se mueven a la vez por el mismo espacio.

Hemos colocado 5 drones en un extremo de la nave p={[10, 3, 4], [15, 3, 4], [20, 3, 4], [25, 3, 4], [30, 3, 4]} y otros 5 drones en el otro extremo de la nave p={[10, 57, 4], [15, 57, 4], [20, 57, 4], [25, 57, 4], [30, 57, 4]}. Cada dron tendrá como destino la posición simétricamente opuesta. Por ejemplo el dron que sale en la posición p_A=[10, 3, 4] tendrá como destino la posición p_B=[30, 57, 4].

Como en los anteriores experimentos primero calcularemos los *waypoints* mediante el algoritmo A*, véase la figura 5.9. Todos los drones calculan su camino teniendo en cuenta los anteriores para evitar riesgo de colisión. Después de calcular los *waypoints* simplemente queda hacer la simulación, figura 5.10, y analizar el comportamiento de los drones. En la simulación se aprecia que todos los drones no pasan por el centro en el mismo momento y parece que respetan cierta distancia de seguridad. Para tener datos más exactos hemos calculado la distancia mínima entre todos los drones para detectar posibles colisiones, lo que se muestra en la figura 5.11. Para finalizar, en la figura 5.12 se puede apreciar que los drones van separados, no están todos apelotonados en el centro, cada uno sigue su camino evitando así que se crucen en su camino con otros drones. En el siguiente enlace se puede ver un video de la trayectoria de los drones: https://youtu.be/dOnB4EzZ5jM

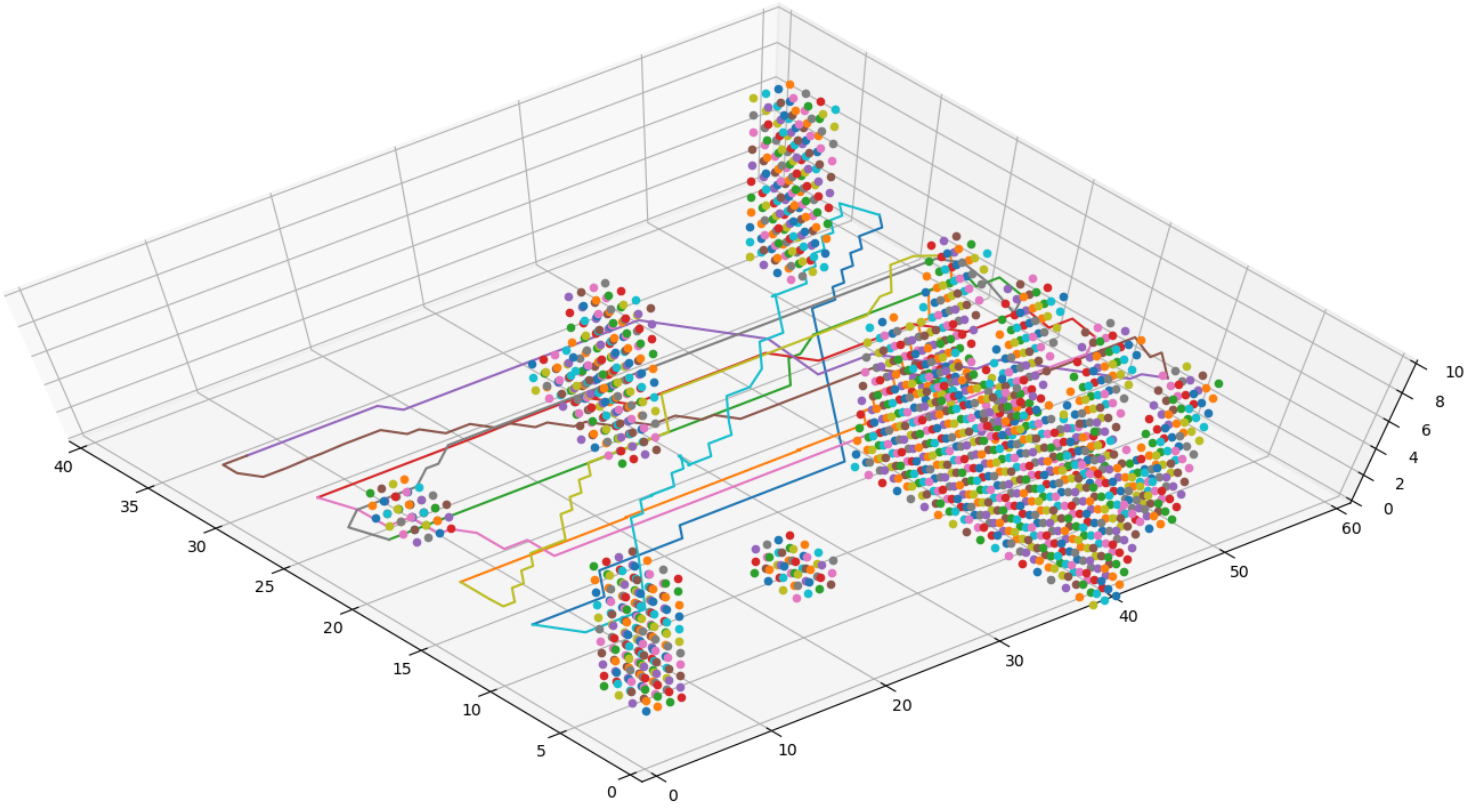

Figura 5.9. Trayectoria calculada por el algoritmo A* para el experimento recorrido completo

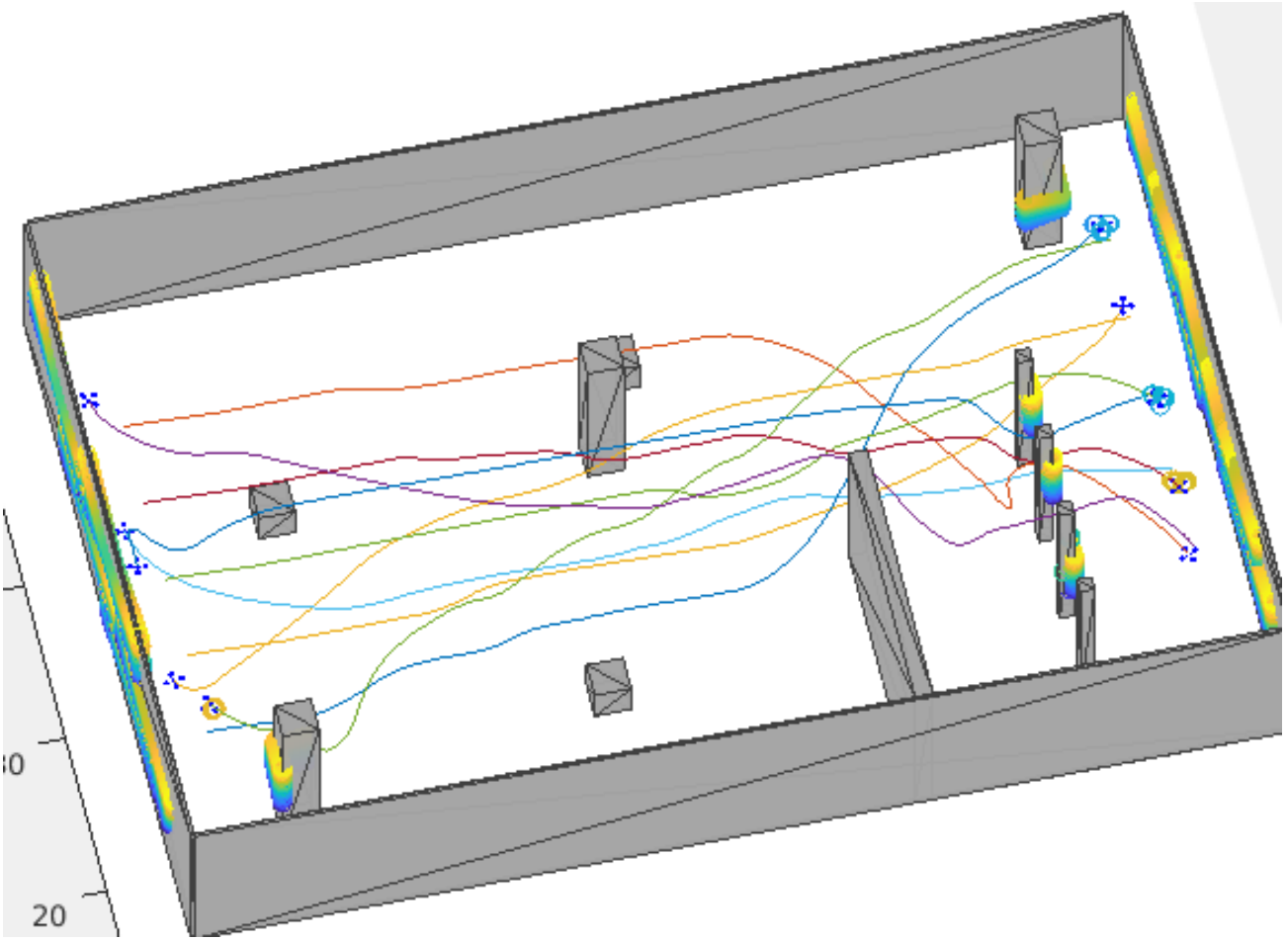

Figura 5.10. Trayectoria del experimento recorrido completo, control mediante A*

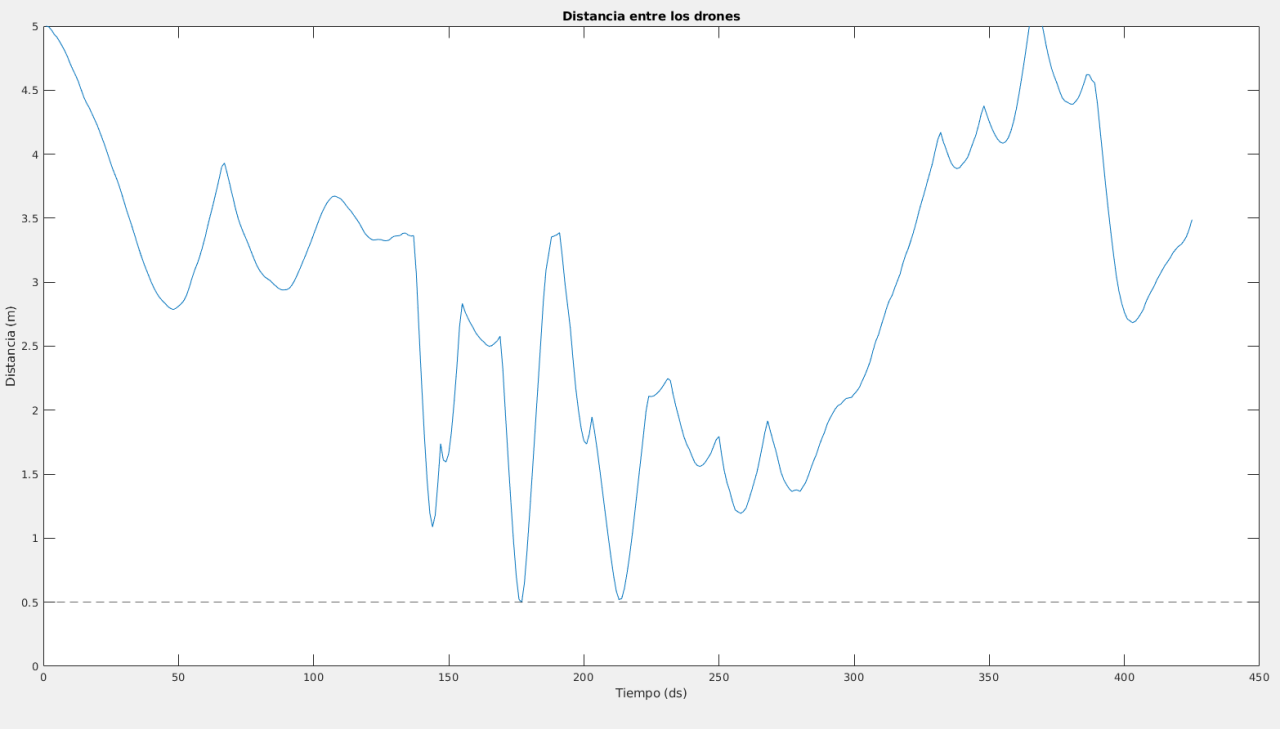

Figura 5.11. Distancia entre drones en el experimento recorrido completo, control mediante A*

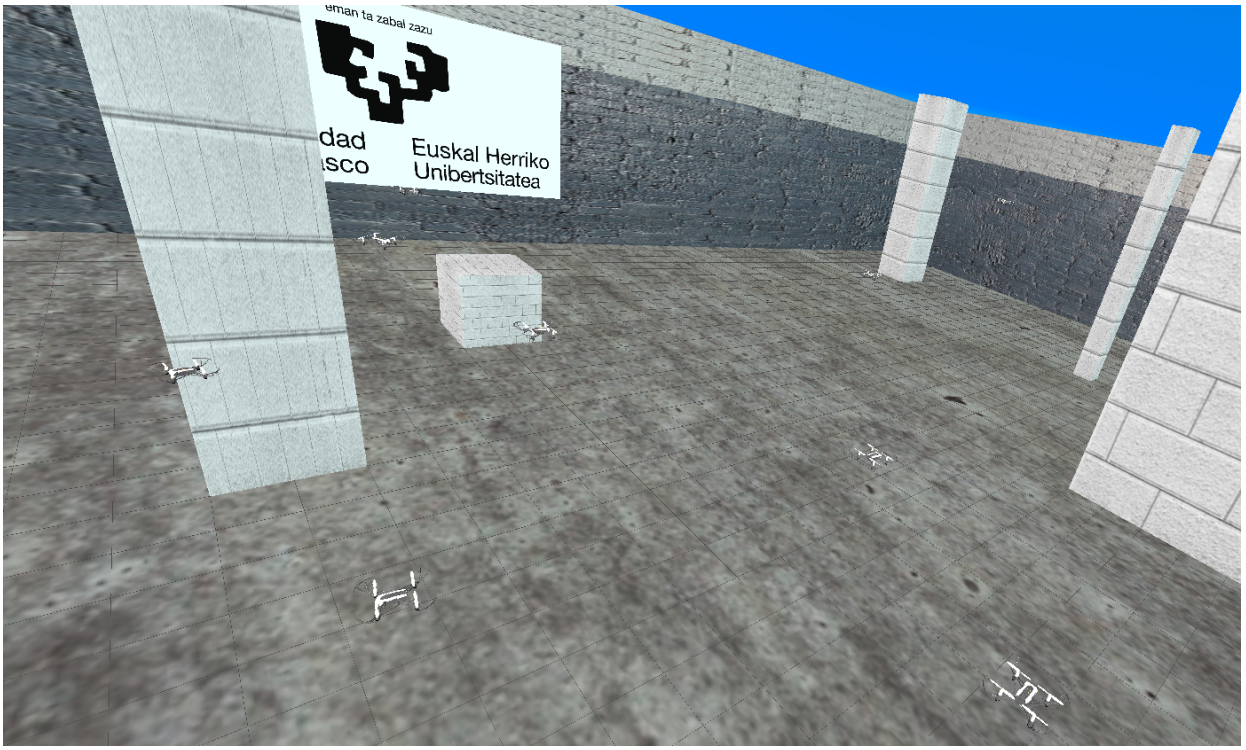

Figura 5.12. Visualización 3D del experimento recorrido completo, control mediante A*

5.5 Comparación

En este apartado vamos a comparar los resultados obtenidos con el algoritmo de control local basado únicamente en sensores y el algoritmo de control mediante A*.

Como hemos visto en los experimentos de pared 4.3.2 y 5.4.2 en ciertos casos es necesario un control mediante una estación de control terrestre para calcular la trayectoria del dron, porque sino el dron puede estar dando vueltas por el espacio sin encontrar el camino hacia el destino.

Además como hemos comprobado con el experimento 4.3.3 el control local basado únicamente en sensores no evita la colisión entre drones cuando hay muchos drones en un espacio reducido, en cambio el algoritmo de control mediante A* reduce significativamente las probabilidades de que ocurra una colisión entre drones. Para confirmar esto hemos repetido el experimento de recorrido completo de una población de drones 5 veces poniendo los drones en posiciones aleatorias y calculado la distancia mínima que hay entre los drones, tabla 5.1.

Distancia minima entre drones (m)		
nº	Control mediante sensores	Control mediante A*
1	0.098	0.91
2	0.107	0.97
3	0.08	0.35
4	0.19	1.01
5	-	0.57

Tabla 5.1. Comparación distancia mínima entre drones

El algoritmo de control mediante A* propuesto en el capítulo 5 consigue unas distancias mínimas entre drones mucho más grandes que el algoritmo simple de control mediante sensores, el riesgo de colisión es mucho más pequeño, además en el algoritmo de control mediante sensores algún dron puede quedarse atascado y no encontrar el camino a su destino (como ocurre en el quinto ejemplo de la tabla 5.1).

6

Conclusiones y trabajo futuro

6.1 Conclusiones

En este proyecto se han conseguido cumplir los objetivos propuestos al principio:

- **Crear un entorno de simulación de múltiples drones simultáneos**: hemos adaptado el ejemplo de *UAV Obstacle Avoidance in Simulink* [16] para poder crear una simulación con varios drones simultáneos.

- **Crear un entorno de visualización de realidad virtual en 3D**: hemos creado un mundo de realidad virtual mediante el software *3D World Editor* de MatLab para mejorar y dar realismo a la visualización de las trayectorias de los drones.

- **Analizar el comportamiento de los drones con un control simple mediante sensores**: hemos analizado y estudiado el comportamiento de los drones con un control simple mediante sensores [16], hemos comprobado que este control no es efectivo en un espacio aéreo con muchos drones moviéndose simultáneamente.

- **Implementar un algoritmo de control que evita la colisión entre drones y optimiza la trayectoria de los drones**: hemos implementado un algoritmo de control que utiliza el algoritmo A* para calcular la trayectoria de los drones. Este algoritmo ha resultado ser muy efectivo para la evitación de colisiones entre drones.

También hemos tenido algunas complicaciones mientras desarrollamos el proyecto:

- **Compatibilidad de software**: la simulación se ha hecho en MatLab pero el cálculo de trayectorias mediante A* se ha hecho en Python, por suerte MatLab tiene la opción de llamar y utilizar módulos de Python [17], pero este sistema es solo compatible con algunas versiones de Python y Matlab. Nosotros hemos utilizado la versión *R2021-b* de MatLab y la versión 3.8.3 de Python.

- **Costo computacional**: la simulación de varios drones requiere un costo computacional muy grande, simular el comportamiento de 10 drones a la vez es prácticamente imposible en el ordenador de casa. Por ello el tutor me ha dado acceso a un aula de la facultad con ordenadores potentes para realizar los experimentos.

- **Sistemas de coordenadas**: para calcular la orientación de los drones la simulación de simulink utiliza coordenadas del tipo ángulos de Euler ZYX, en cambio la visualización 3D utiliza la notación axial-angular. Por ello, para conseguir la orientación real de los drones en la visualización 3D hemos tenido que transformar las orientaciones de Euler ZYX a axial-angular.

6.2 Trabajos futuros

Todo el trabajo realizado se ha basado en un entorno virtual tratando de imitar a un entorno real del día a día. Sería muy interesante analizar el funcionamiento del algoritmo con drones reales en un entorno real como una nave, un taller, o una fábrica.

Nosotros hemos optado por el algoritmo A* para calcular la ruta más corta entre dos puntos pero este algoritmo tiene un alto coste computacional, sobre todo por el espacio de memoria que ocupa. Se pueden utilizar otros algoritmos para el cálculo de la trayectoria [15]: algoritmos genéticos [14], PSO (*Particle Swarm Optimization*) [14], RRT (*Rapidly Exploring Random Trees*).

Con el algoritmo de control mediante A* los drones calculan sus trayectorias al principio del recorrido y lo siguen hasta el final mediante el *WayPointFollower,* sería interesante probar el algoritmo de control mediante A* pero calculando la trayectoría de los drones cada X tiempo y no solamente al principio.

Bibliografía

[1] Pothuganti, Karunakar & Jariso, Mesfin & Kale, Pradeep. (2017). A Review on Geo Mapping with Unmanned Aerial Vehicles. International Journal of Innovative Research in Computer and Communication Engineering. 3297.

[2] Campion, Mitchell & Prakash, Ranganathan & Faruque, Saleh. (2018). UAV Swarm Communication and Control Architectures: A Review. Journal of Unmanned Vehicle Systems. 7. 10.1139/juvs-2018-0009.

[3] Park, Sanghyuk & Deyst, John & How, Jonathan. (2004). A New Nonlinear Guidance Logic for Trajectory Tracking. 10.2514/6.2004-4900.

[4] Vanneste, Simon & Bellekens, Ben & Weyn, Maarten. (2014). 3DVFH+: Real-Time Three-Dimensional Obstacle Avoidance Using an Octomap. CEUR Workshop Proceedings. 1319.

[5] I. Ulrich and J. Borenstein, ”Vfh+: Reliable obstacle avoidance for fast mobile robots,” in Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on, vol. 2. IEEE, 1998, pp. 1572-1577.

[6] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, ”Octomap: An efficient probabilistic 3d mapping framework based on octrees,” Autonomous Robots, vol. 34, no. 3, pp. 189-206, 2013.

[7] Kuantama, Endrowednes & Vesselenyi, Tiberiu & Dzitac, Simona & Tarca, Radu. (2017). PID and Fuzzy-PID Control Model for Quadcopter Attitude with Disturbance Parameter. International Journal of Computers, Communications and Control. 12. 519-532. 10.15837/ijccc.2017.4.2962.

[8] F. Boehm and A. Schulte, "Air to ground sensor data distribution using IEEE802.11N Wi-Fi network," 2013 IEEE/AIAA 32nd Digital Avionics Systems Conference (DASC), 2013, pp. 4B2-1-4B2-10, doi: 10.1109/DASC.2013.6712581.

[9] Gupta, L., Jain, R., & Vaszkun, G. (2016). Survey of Important Issues in UAV Communication Networks. IEEE Communications Surveys and Tutorials, 18(2), 1123–1152. https://doi.org/10.1109/comst.2015.2495297

[10] Dario Calogero Guastella, Nunzio Dario Cavallaro, Carmelo Donato Melita, Maurizio Savasta, and Giovanni Muscato. 2018. 3D path planning for UAV swarm missions. In Proceedings of the 2018 2nd International Conference on Mechatronics Systems and Control Engineering (ICMSCE 2018). Association for Computing Machinery, New York, NY, USA, 33–37. https://doi.org/10.1145/3185066.3185069

[11] Hart, P. E., Nilsson, N. J., & Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics, 4(2), 100-107

[12] *¿Como funciona A*?* https://www.ecured.cu/Algoritmo_de_B%C3%BAsqueda_Heur%C3%ADstica_A*

[13] Función astar_path de la librería *networkX* https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.astar.astar_path.html

[14] V. Roberge, M. Tarbouchi and G. Labonte, "Comparison of Parallel Genetic Algorithm and Particle Swarm Optimization for Real-Time UAV Path Planning," in *IEEE Transactions on Industrial Informatics*, vol. 9, no. 1, pp. 132-141, Feb. 2013, doi: 10.1109/TII.2012.2198665.

[15] Liang Yang, Juntong Qi, J. Xiao and Xia Yong, "A literature review of UAV 3D path planning," Proceeding of the 11th World Congress on Intelligent Control and Automation, 2014, pp. 2376-2381, doi: 10.1109/WCICA.2014.7053093.

[16] UAV Obstacle Avoidance in Simulink https://es.mathworks.com/help/uav/ug/uav-obstacle-avoidance-in-simulink.html

[17] Access Python Modules from MATLAB - Getting Started https://es.mathworks.com/help/matlab/matlab_external/create-object-from-python-class.html

Anexo A: Modelos y programas

CÁLCULO DE TRAYECTORIAS MEDIANTE ALGORITMO DE CONTROL A*:

```
"""
Created on Mon May  2 09:51:55 2022

@author: jongar
"""

import networkx as nx
import random
import matplotlib.pyplot as plt

def calculate(start,end):


   MAX_X=40
   MAX_Y=60
   MAX_Z=10
   G = nx.grid_graph(dim = [MAX_Z,MAX_Y,MAX_X]) ##Z,Y,X
   #nx.draw(G)
   for i in range(0,MAX_X):
       for j in range(0,MAX_Y):
           for k in range(0,MAX_Z):
               for z in [-1,0,1]:
                   for s in [-1,0,1]:
                       for l in [-1,0,1]:
                           i_=i+z
                           j_=j+s
                           k_=k+l
                           if(i_>=MAX_X or j_>=MAX_Y or k_ >= MAX_Z or
i_<0 or j_<0 or k_<0 or(i_== i and j_==j and k==k_)):
                               continue
                           d=(z**2+s**2+l**2)**(0.5) ##distantzia
kalkulatu
                           G.add_edge((i,j,k),(i_,j_,k_),weight=d)



   add_obstacle(G, .8,.8, 10, (5,50))
   add_obstacle(G, .8,.8, 10, (10,50))
   add_obstacle(G, .8,.8, 10, (15,50))
```

```
    add_obstacle(G, .8,.8, 10, (20,50))

    add_obstacle(G, 2,2,2, (10,25))
    add_obstacle(G, 2,2,2, (25,10))
    add_obstacle(G, 2,2,2, (30,30))

    add_obstacle(G, 2,2,10, (6,7))
    add_obstacle(G, 2,2,10, (34,54))
    add_obstacle(G, 2,2,10, (25,28))

    add_obstacle(G, 16,1,10, (8,40))



    wps=[]
    for i in range(0,len(start)-2,3): ##for each dron

        nondik =(start[i],start[i+1],start[i+2])
        nora =(end[i],end[i+1],end[i+2]) #
        if G[nondik]=={}:
            nondik = move_position(G,nondik,MAX_X,MAX_Y,MAX_Z)
        if G[nora]=={}:
            nora = move_position(G,nora,MAX_X,MAX_Y,MAX_Z)

        wp=nx.astar_path(G,nondik,nora,heuristic=distance_h)
        print(wp)

        wp2=[]
        for j in range(3,len(wp)-2,2):
            wp2.append(wp[j])
        wp2.insert(0,((start[i],start[i+1],start[i+2]) ))
        wp2.append((end[i],end[i+1],end[i+2]))

        wps.append(wp2)

        wp.insert(0,((start[i],start[i+1],start[i+2]) ))
        wp.append((end[i],end[i+1],end[i+2]))

        add_obsacle_dron(G,wp)##add dron tratectory as obstacle for
other drones

    return wps


def distance_h(node1,node2):
    a = node1[0]-node2[0]
    b = node1[1]-node2[1]
```

```
    c = node1[2]-node2[2]
    return (a**2+b**2+c**2)**(0.5)

def add_obsacle_dron(G,w_pts):
    for i in w_pts:
        node = i
        x=node[0]
        y=node[1]
        z=node[2]
        for d in [-1,0,1]:
            for s in [-1,0,1]:
                for l in [-1,0,1]:
                    try:
                        G.remove_edge((x+d,y+s,l+z), (x,y,z))
                    except:
                        continue #ELEMENT ISNT IN THE GRAPH

def add_obstacle(G,w,h,altuera,pos):

    x = pos[0] - w/2
    y = pos[1] - h/2
    for i in range(int(x)-1,int(x+w)+1):
        for j in range(int(y)-1,int(y+h)+1):
            for z in range(0,altuera+1):
                add_obsacle_dron(G,[(i,j,z)])

def move_position(G,pos,MAX_X,MAX_Y,MAX_Z):
    x_=random.randint(-1, 1);
    y_=random.randint(-1, 1);
    z_=random.randint(0,0);
    x=x_+pos[0]
    y=y_+pos[1]
    z=z_+pos[2]
    if(x <0 or y<0 or z<0 or x>MAX_X or y>MAX_Y or z>MAX_Z):
        return move_position(G,pos,MAX_X,MAX_Y,MAX_Z)
    if G[x,y,z] != {}:
        return (x,y,z)
    return move_position(G,(x,y,z),MAX_X,MAX_Y,MAX_Z)
```