Trabajo Fin de Grado
Grado en Ingeniería Aeroespacial

Integración de un ESDF generado a través de redes neuronales en planificación para robots aéreos

Autor: Guillermo Gil García
Tutor: Jesús Capitán Fernández
Cotutor: José Antonio Cobano Suárez

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

Sevilla, 2024

Trabajo Fin de Grado
en Ingeniería Aeroespacial

Integración de un ESDF generado a través de redes neuronales en planificación para robots aéreos

Autor:
Guillermo Gil García

Tutor:
Jesús Capitán Fernández
Profesor titular

Cotutor:
José Antonio Cobano Suárez
Profesor Ayudante doctor (UPO)

Dpto. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2024

Trabajo Fin de Grado: Integración de un ESDF generado a través de redes neuronales en planificación para robots aéreos

Autor: Guillermo Gil García

Tutor: Jesús Capitán Fernández

Cotutor: José Antonio Cobano Suárez

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

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2024

El Secretario del Tribunal

Agradecimientos

Gracias a mi familia y amigos por apoyarme cada día, a mis tutores por el trabajo realizado y al Service Robotics Lab por permitirme embarcarme en este proyecto tan interesante.

Guillermo Gil García
Sevilla, 2024

Resumen

Este documento presenta el proceso de diseño y validación de un planificador de trayectorias para drones aéreos basado en un Campo de Distancia Euclídea con Signo (o ESDF, siglas en inglés de *Euclidean Signed Distance Field*) generado de forma *online* con ayuda de una red neuronal.

Para ello, en primer lugar se detalla la estructura del sistema, que se compone de diversos bloques cuyos objetivos son obtener datos del entorno, procesarlos y utilizarlos para entrenar una red neuronal. Dicha red neuronal proporciona la información necesaria para representar el entorno a partir de los datos de los sensores, la cual se utiliza posteriormente para la planificación de trayectorias en conjunto con un algoritmo de planificación basado en grafos, como es el A*.

El sistema se ha desarrollado utilizando ROS (Robot Operating System), habiéndose programado de forma híbrida tanto en C++ como en Python.

Los resultados se han obtenido a través de diferentes simulaciones llevadas a cabo con un dron en un entorno real. Para ilustrar los resultados, se han usado herramientas como Gazebo y RViz.

Abstract

This document presents the design and validation process of a path planning system for aerial drones based on online generated Euclidean Signed Distance Fields with the help of a neural network.

To do this, the structure of the system is first detailed, which is made up of various blocks whose objectives are to obtain data from the environment, process it and use it to train a neural network. This neural network is then used for trajectory planning in conjunction with a graph-based planning algorithm, such as A*.

The system has been developed using ROS (Robot Operating System), having been programmed in a hybrid way in both C++ and Python.

The results have been obtained through simulations using ROS, representing the results with Gazebo and RViz.

Índice Abreviado

Índice

1 Introducción

En las últimas décadas, el desarrollo de la robótica y todos sus campos derivados han supuesto grandes avances tecnológicos que permiten preveer cambios de enormes proporciones en muchos ámbitos de la industria.

Una gran parte de los avances tecnológicos en el amplio espectro de la robótica están comprendidos dentro la robótica aérea, basada principalmente en el desarrollo de vehículos aéreos no tripulados (VANT, por sus siglas en español, o UAV/RPAS, por sus siglas en inglés), también llamados drones. Desde la primera mención de los UAV en el ámbito militar en 1849 [6], su utilización se ha extendido también a tareas civiles, las cuales pueden subdividirse en cuatro grandes áreas de aplicación [30]:

- Tareas de investigación y desarrollo científico como la monitorización ambiental y de ecosistemas [5]
- Tareas de vigilancia [38]
- Aplicaciones industriales, principalmente en la agricultura y en la construcción [13]
- Aplicaciones para la protección civil, como la monitorización de eventos climatológicos extremos (huracanes) y sus consecuencias [37]

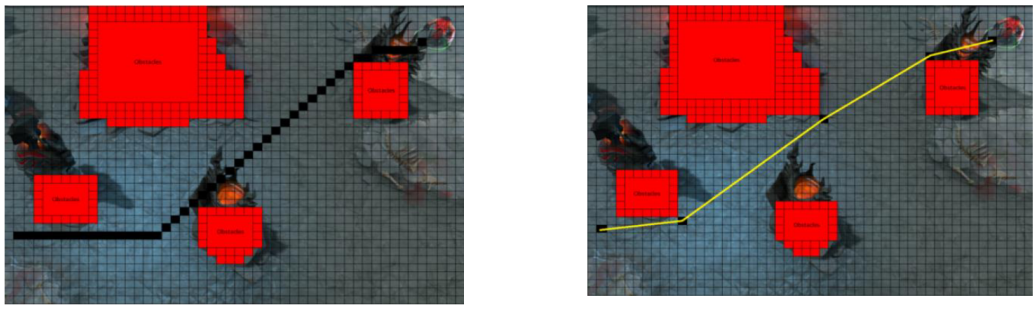
La aplicación de los drones para tareas que pueden llegar a ser de gran dificultad técnica suponen un reto no solo a nivel puramente constructivo (con la necesidad de realizar diseños cada vez más complejos donde la seguridad juega un papel más importante), sino también a la hora de programar planificadores de trayectorias eficaces que puedan hacer frente a entornos difíciles o a requisitos cada vez más complejos. Este Trabajo Fin de Grado se centrará en el desarrollo y la implementación de uno de estos algoritmos de planificación de trayectorias para drones aéreos, fomentando la seguridad de los caminos generados.

1.1 Planificación de trayectorias en UAVs

En el ámbito de la robótica, y en este caso centrándonos en los drones no tripulados, la planificación de trayectorias tiene como objetivo el cálculo de caminos que eviten la colisión del robot con el entorno y que, adicionalmente, cumplan una serie de requisitos basados en especificaciones dependientes de las tareas a realizar.

Existen una gran cantidad de algoritmos, siendo la mayoría clasificables en uno de varios grandes subgrupos según su metodología principal de planificación. Los algoritmos basados en cuadrículas de costes, como el A* [12] o el D* [33], son los que cuentan con mayor longevidad, si bien se siguen empleando en mayor o menor medida gracias a que garantizan encontrar trayectorias óptimas [35]. En la Figura 1.1 se ofrece una comparación entre dos algoritmos ampliamente utilizados.

Otro subgrupo de algoritmos son los basados en campos potenciales, como el APF (siglas en inglés de campo potencial artificial) [15], los cuales utilizan una función potencial para todo el espacio de configuración considerado y realizan la planificación basándose en ellos. Estos métodos tienen ciertos problemas relacionados con mínimos locales, por lo que es común su aplicación integrándolos con otros planificadores, como será el caso de este trabajo. En la Figura 1.2 se puede ver un ejemplo de camino obtenido con un



(a) (b)

Figura 1.1 Comparación entre el algoritmo A* (a) y el Theta* (b), dos algoritmos basados en cuadrículas de costes [17].

algoritmo basado en campos potenciales.

Un último subgrupo de algoritmos de planificación son los basados en muestreo, siendo los casos más conocidos el algoritmo RRT [16] o el PRM [14], los cuales se han vuelto muy populares en la actualidad gracias a que funcionan excepcionalmente bien en problemas de muchos grados de libertad o con gran cantidad de restricciones [35], si bien también se caracterizan por no encontrar siempre soluciones óptimas. En la Figura 1.3 se puede ver un ejemplo gráfico del algoritmo de planificación RRT en funcionamiento.

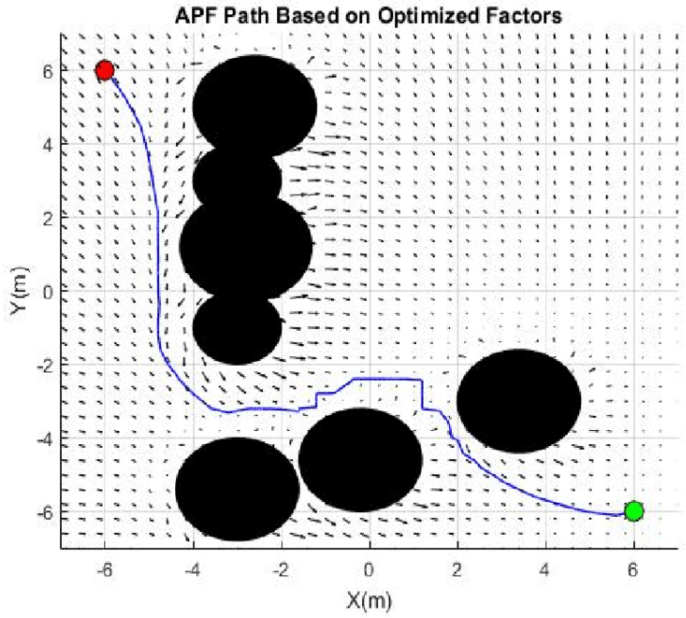


Figura 1.2 Algoritmo de planificación basado en APF [27].

1.2 Evitación de obstáculos y representación del entorno en UAVs: ESDF

Pasando al problema específico a tratar en este Trabajo Fin de Grado, generar una trayectoria en tiempo real para un UAV que sea tanto eficiente como segura es una tarea crítica en algunas aplicaciones de los vehículos aéreos, especialmente cuando la navegación debe darse a través de entornos con una gran densidad de obstáculos. Esta casuística ocurre y es de especial importancia en ciertos ámbitos donde la utilización de drones resulta de especial interés para evitar poner en peligro vidas humanas, como en la gestión de desastres naturales [19], las operaciones de rescate [31], la inspección de minas subterráneas [20] y la extinción de

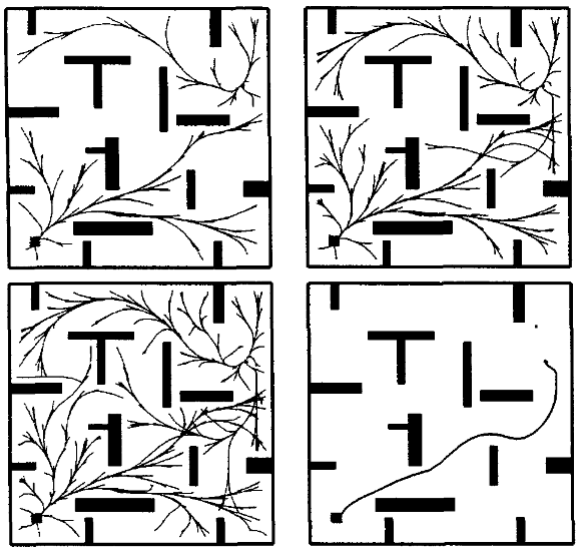


Figura 1.3 Etapas de exploración del algoritmo RRT, con la trayectoria final abajo a la derecha [16].

incendios urbanos [21].

Planificar en tiempo real de forma autónoma requiere de una representación lo más exacta posible del entorno por el que se quiere navegar, para así evitar colisionar con obstáculos. Esta representación debe ser generada y actualizada de forma contínua, utilizando los sensores de abordo, ya sea con cámaras o con LiDARs.

Entre los métodos utilizados para la representación del entorno, se ha popularizado el uso de los Campos de Distancia Euclídea con Signo (*ESDF* por sus siglas en inglés), los cuales son particularmente útiles a la hora de representar de forma eficiente el entorno para tareas de evitación de obstáculos [11]. Este método, descrito en detalle en el Capítulo 2, cuenta con diversas propiedades que lo convierten en una herramienta muy útil para tareas de planificación.

Otra alternativa utilizada en la bibliografía son los campos neuronales (*neural fields*) [22], los cuales utilizan redes neuronales para la representación del mapa de ocupación del entorno del dron. Este tipo de representaciones cuentan con una serie de ventajas que resultan de gran interés, siendo una de ellas que la representación resultante es una función contínua en el espacio y que, por tanto, no es necesario interpolar en ningún momento [24] [34].

Si bien la literatura cuenta con numerosos trabajos que emplean ESDFs para la representación del entorno para tareas de planificación [23] [10] [9], no parece haber trabajos que hayan analizado la efectividad de una posible integración de ESDFs (habitualmente implementados de forma discreta, como se verá en capítulos posteriores) en una red neuronal. Este Trabajo Fin de Grado pretende integrar un ESDF generado de esta manera en un sistema de planificación de trayectorias para drones.

1.3 Objetivos

Los objetivos de este trabajo son la síntesis y la posterior evaluación de un sistema de planificación de trayectorias para drones aéreos que utilice un ESDF generado a través de una red neuronal entrenada de forma contínua.

El objetivo general de este Trabajo Fin de Grado no es solo el diseño del propio sistema que integre un ESDF generado por una red neuronal con los planificadores, sino también la simulación posterior de este y la obtención datos experimentales que permitan verificar el correcto funcionamiento del planificador. Para cumplir con este objetivo, se procederá de forma secuencial. En el Capítulo 2 se describirán conceptos clave como los ESDF y las redes neuronales, planteando el problema y justificando la solución propuesta. El Capítulo 3 estará relacionado con la estructura del sistema en su totalidad, y se ofrecerá una descripción en

detalle de cada una de sus partes. En el Capítulo 4 se detallará la implementación completa del sistema en una plataforma adecuada para su uso y simulación (ROS). El Capítulo 5 se corresponderá a la experimentación y la batería de pruebas utilizadas para validar el funcionamiento de lo implementado. Por último, el Capítulo 6 ofrecerá algunas conclusiones, así como propuestas para posibles líneas de investigación a futuro.

2 Enfoque Propuesto

En este capítulo se plantea un primer acercamiento a qué se quiere abordar en este Trabajo Fin de Grado, explicando conceptos clave relacionados que permitirán una fácil comprensión de capítulos posteriores. Como se describió en el último apartado del capítulo anterior, el objetivo último del trabajo es desarrollar un algoritmo de planificación basado en cuadrículas de costes (en este caso, un A*) que utilice Campos de Distancia Euclídea con Signo (ESDF desde ahora, al ser las siglas de *Euclidean Signed Distance Field*) generados por una red neuronal entrenada mientras el dron navega por el entorno.

En primer lugar se va a explicar en qué consiste el algoritmo básico A*, para pasar posteriormente a detallar qué es un ESDF, para qué se utilizará en el algoritmo de planificación planteado y qué ventajas podría tener su implementación en una red neuronal.

2.1 Algoritmo A*

El algoritmo de planificación de trayectorias A* (pronunciado "A estrella") fue publicado por primera vez en el año 1968 por parte de los investigadores Peter Hart, Nils Nilsson y Bertram Raphael, del Stanford Research Institute [12]. Es usado con frecuencia en muchos campos de la computación, gracias a su completitud, optimalidad y eficiencia óptima [29]. De manera general, dado un grafo cuyos nodos tienen ciertos pesos asociados, el algoritmo es capaz de encontrar el camino de menor coste entre un nodo de inicio y un nodo final.

En la bibliografía se define el A* como un algoritmo "*informed search*" o "*best-first search*", es decir, que encuentra el camino de menor coste construyendo un árbol de caminos desde el nodo de inicio, el cual se va extendiendo nodo a nodo hasta que se llega al nodo final. Para ello, en cada iteración se debe determinar qué nodo debe ser el siguiente en ser anexado al árbol de caminos, el cual se determina minimizando la siguiente función de coste:

$$f(n) = g(n) + h'(n) \tag{2.1}$$

El término $g(n)$ representa el coste real que supone viajar desde el nodo de inicio hasta el nodo n. Este coste no tiene por qué calcularse únicamente como la distancia entre un nodo y otro, ya que pueden incluirse otros costes relacionados con especificaciones deseadas para el vehículo. El algoritmo que se implementará en este trabajo, explicado más adelante, contiene un término adicional para el cálculo de $g(n)$ basado en la distancia del nodo al obstáculo más cercano.

Por otro lado, el término $h'(n)$ representa una estimación del coste de viajar desde el nodo n hasta el nodo final. El apóstrofe tras la h se coloca debido a que este término debe ser estimado heurísticamente, pues no se conoce a priori. Existen múltiples formas de determinar $h(n)$, si bien se debe asegurar que sea heurísticamente admisible (es decir, que nunca sobreestime el coste de llegada) para que la solución final del algoritmo pueda considerarse óptima [29]. Si esto no se cumpliera, no estaríamos hablando del algoritmo

A*, sino de un algoritmo llamado simplemente A.

A continuación se presenta, en forma de pseudocódigo, el funcionamiento del algoritmo A*:

Algorithm 1 Algoritmo de planificación A*

```
function A*(start, goal)
    OpenList ← {start}
    ClosedList ← {}
    for every node n do
        g(n) ← ∞
        f(n) ← ∞
        previous_node(n) ← ∅
    end for
    g(start) ← 0
    f(start) ← h'(start)
    while OpenList is not empty do
        current ← node in OpenList with lowest f(current)
        if current = goal then
            return ReconstructPath(previous_node, current)
        end if
        OpenList ← OpenList \ {current}
        ClosedList ← ClosedList ∪ {current}
        for each neighbor neighbor of current do
            g'(neighbor) ← g(current) + cost(current, neighbor)
            if g'(neighbor) < g(neighbor) then
                previous_node(neighbor) ← current
                g(neighbor) ← g'(neighbor)
                f(neighbor) ← g(neighbor) + h'(neighbor)
                if neighbor ∉ OpenList ∪ ClosedList then
                    OpenList ← OpenList ∪ {neighbor}
                end if
            end if
        end for
    end while
end function

function ReconstructPath(previous_node, current)
    Path ← {current}
    while previous_node(current) ≠ ∅ do
        current ← previous_node(current)
        Path ← {current} ∪ Path
    end while
    return Path
end function
```

Como se puede ver, el algoritmo funciona manteniendo dos listas: una abierta, que contiene los posibles puntos a explorar, y una cerrada, que está formada por los puntos ya explorados. En cada iteración del algoritmo (y tras la inicialización), se toma el punto de menor coste total entre todos los de la lista abierta, metiéndose en la lista cerrada. Posteriormente, para cada vecino de dicho punto se recalculan los costes desde el inicio como si el punto actual fuera su punto "padre". La función de coste para ello (que en el algoritmo aparece como *cost*) puede variar enormemente según la aplicación del algoritmo.

Una vez calculado el nuevo coste, si es menor que el que ya tenía el punto vecino, el punto actual se convierte en su nuevo nodo padre, información que se almacena en la variable *previous_node*. Una vez se llega al nodo considerado final, reconstruir la trayectoria completa es tan sencillo como comprobar los nodos

padre de cada nodo, empezando por el final y finalizando en el nodo de partida.

2.2 Euclidean Signed Distance Fields

A la hora de sintetizar el planificador, otro concepto clave que debe ser explorado es el del ESDF. Los ESDF son campos escalares capaces de representar un espacio de cualquier dimensión. Para cada punto de dicho espacio, el campo le asigna un escalar con valor absoluto igual a la distancia ortogonal entre dicho punto y el obstáculo más cercano. De esta manera, el valor del campo escalar en la superficie de un obstáculo será de 0, e irá aumentando paulatinamente según se toman puntos más alejados de este [7].

Si bien esta breve descripción coincide con un EDF (*Euclidean Distance Field*), su principal diferencia con respecto al ESDF es que este último asigna a cada punto un signo además del valor escalar mencionado anteriormente. Los puntos considerados inaccesibles desde el exterior (es decir, los puntos que están "dentro" de un obstáculo) tendrán signo negativo, mientras que a los demás se les asignará el signo positivo. En la Figura 2.1 se puede ver un ejemplo de aplicación del ESDF a un espacio bidimensional con un obstáculo de forma circular. Este ejemplo podría tomarse, también, como una lasca bidimensional de un ESDF tridimensional con un obstáculo en forma de columna. Como el espacio está dividido en forma de cuadrícula, el valor de cada una de ellas se calcula teniendo en cuenta su punto central. Es decir, se trata de un campo discretizado.

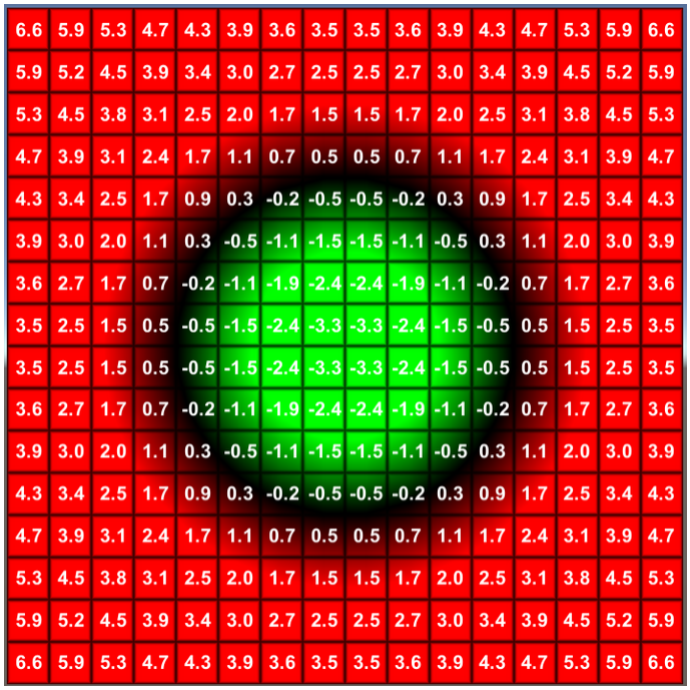


Figura 2.1 Ejemplo de ESDF aplicado a un obstáculo circular (en negro) [8].

Este tipo de campo escalar cumple con una serie de propiedades matemáticas que lo hacen muy susceptible a ser usado en tareas de planificación de trayectorias. El campo es derivable en casi todos sus puntos (siendo la única excepción las líneas medias entre obstáculos, donde existen singularidades no derivables) y su gradiente cumple la ecuación de la eikonal $\|\nabla f\| = 1$.

La forma de integrar los ESDF en planificadores como el A* es introduciendo un término adicional en la función de coste descrita en el apartado anterior (en concreto, un término adicional en $g(n)$). Como el término añadido es dependiente de la inversa del valor del ESDF en el nodo del que se quiere calcular el coste, se penalizará con un mayor coste que el dron pase cerca de un obstáculo, lo cual es especialmente deseable cuando se prioriza la seguridad del vehículo aéreo y de los objetos o personas que lo rodean. La

estructura exacta de la función de coste se explicará en posteriores apartados.

2.3 Integración del ESDF en una red neuronal

La cuestión que puede considerarse más novedosa de este trabajo es, sin duda, la integración del ESDF que se empleará para la planificación en una red neuronal que se entrenará durante el proceso de navegación. Sin embargo, esto también plantea la necesidad de justificar el uso de redes neuronales cuando, en la actualidad, basta con implementar el ESDF como una matriz tridimensional precargada en el dron donde se implementa el planificador. En este apartado se van a detallar las principales razones por las que esta implementación tiene sentido desde el punto de vista funcional.

En primer lugar, implementar un ESDF a través de una matriz con valores precalculados implica discretizar el espacio en un cierto número finito de celdas tridimensionales cúbicas de una cierta dimensión. De esta forma, solo se tienen valores del ESDF para ciertos puntos concretos, y la obtención de valores intermedios radica en la interpolación, resultando en valores no siempre exactos según la geometría del mapa. Este problema no existe cuando se implementa el ESDF a través de una red neuronal, pues esta es, a todos los efectos, una función que toma un cierto valor en todos los puntos del espacio, lo que proporciona una resolución teóricamente ilimitada.

Un segundo punto a favor de la implementación en una red neuronal radica en la cantidad de memoria empleada. Implementar un ESDF en forma de matriz de valores necesita de una gran cantidad de espacio en memoria debido a la necesidad de guardar todos los valores de cada uno de los puntos de la matriz. Este gasto de memoria, además, no es constante y depende en gran medida del tamaño del mapa y de la resolución con la que se quiera implementar el campo escalar. En cambio, la memoria ocupada por una red neuronal es constante para una estructura de la red fija, y no crece según se vaya explorando una mayor parte del espacio. Lo único que debe ser guardado en la usualmente muy limitada memoria del dron aéreo es un número constante y conocido de parámetros de la red.

Por último, la mayor ventaja de la implementación de la red neuronal radica en la posibilidad de realizar replanificación. Cuando un ESDF se implementa en forma de matriz numérica, actualizarlo cuando se detecta un nuevo obstáculo puede llegar a ser inviable, debido a que es necesario recalcular todos y cada uno de los puntos de la matriz. El gasto en cuanto a recursos y tiempo de computación de esta tarea no solo es enorme, sino que crece tanto con el tamaño del mapa como con la resolución. Sin embargo, este problema no ocurre con una red neuronal en constante entrenamiento, la cual puede adaptarse a nuevos obstáculos rápidamente y recalcular sus pesos para representar el mapa más actualizado posible, todo ello sin ningún coste computacional añadido, debido a que el entrenamiento sigue el mismo procedimiento haya nuevos obstáculos o no. Esto abre puertas en el ámbito de la planificación local, permitiendo trazar rutas en ambientes en permanente cambio de forma rápida y, como se verá en posteriores apartados, eficaz.

3 Estructura del sistema

Una vez introducido y justificado el objetivo de este trabajo, este capítulo se centra en la arquitectura que se plantea para la implementación del planificador basado en un A* que utiliza un ESDF almacenado en una red neuronal como función de coste. En primer lugar se presenta el esquema general, que puede verse en la Figura 3.1. A lo largo de este capítulo se describirán secuencialmente cada uno de los bloques que lo forman.

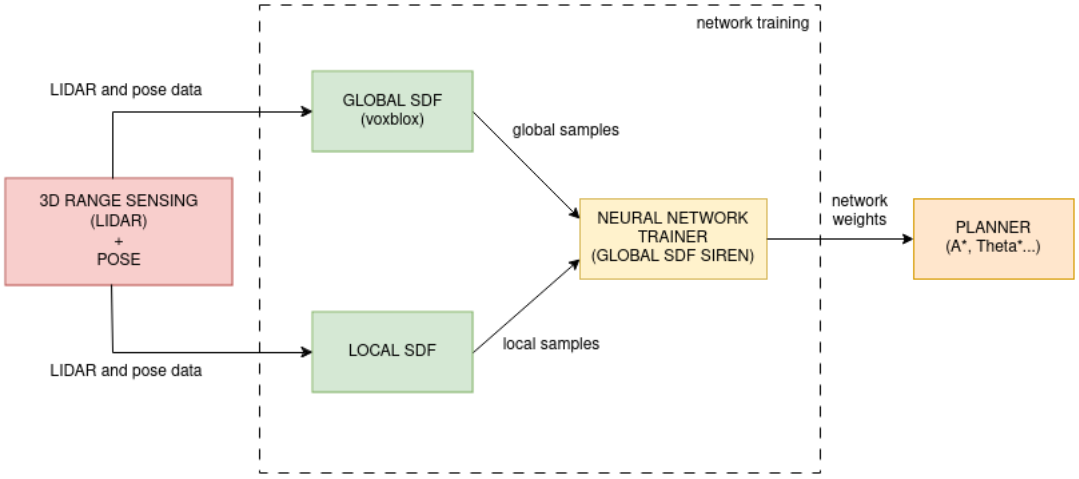


Figura 3.1 Esquema de la arquitectura del sistema a implementar.

El esquema, además de mostrar los bloques principales que componen el sistema de planificación en su conjunto, también muestran el flujo de datos entre ellos. En primer lugar se encuentran los sensores necesarios para llevar a cabo la planificación, en este caso un LiDAR y un sensor inercial de posición. Los datos de estos sensores son, posteriormente, tratados por dos bloques, que tienen como objetivo obtener las muestras necesarias para entrenar la red neuronal de forma efectiva. Por último, y una vez entrenada la red, los parámetros de esta son extraídos para usarlos en la función de coste del planificador.

El sistema está planteado para que sea iterativo. Esto quiere decir que, cada cierto periodo a determinar (pudiendo este ser fijo o variable en función del coste temporal de las operaciones llevadas a cabo) se obtienen los datos de los sensores y se pasan a través de todo el sistema. De esta forma, tras cada iteración se obtiene una red más entrenada que puede ser utilizada para tareas de planificación.

En los siguientes subapartados se detalla el funcionamiento interno de cada bloque.

3.1 Sensores

Como en todo sistema de planificación *on-line*, el presentado en este trabajo necesita de datos actualizados del entorno para llevar a cabo la planificación. Como ya se explicó en el capítulo anterior, el cálculo de un ESDF se basa única y exclusivamente en la disposición de los obstáculos en el espacio. Por tanto, el primer reto del sistema que se quiere construir es plantear qué conjunto de sensores embebidos en el dron son los necesarios para obtener una representación de su entorno. En este caso se ha optado por utilizar un LiDAR y un sensor de posición.

Los sensores LiDAR (siglas en inglés de *Light Detection and Ranging* o de *Laser Imaging Detection and Ranging*) son dispositivos capaces de determinar la distancia desde un emisor láser hasta un objeto, obstáculo o superficie. Si bien existen varios métodos para calcular esta distancia, todos se basan en las propiedades conocidas del haz láser que emiten para ello, ya sea midiendo el retraso o la diferencia de fase entre el haz de luz emitido y el recibido [18].

Existen una gran cantidad de sensores LiDAR diferentes, los cuales difieren en características clave como la precisión en la medida, el rango de detección, la resolución, la frecuencia de muestreo y el campo de visión (FOV), entre otras. A la hora de elegir el sensor adecuado para el dron que va a emplear el sistema de planificación presentado, hay que recordar que el objetivo es obtener una representación lo más completa del espacio circundante, por lo que se propone utilizar un LiDAR montado en la parte inferior del vehículo, con un ángulo de visión de 360º y una apertura de haz lo mayor posible, el cual deberá realizar un barrido periódico de su entorno. En la Figura 3.2 se muestran algunos ejemplos reales de sensores LiDAR en el mercado, siendo el más similar al que se quiere emplear el segundo empezando por la izquierda.

Figura 3.2 Diferentes sensores LiDAR en el mercado actual [4].

Continuando con los sensores necesarios para el sistema, es necesario remarcar que, si bien el LiDAR proporciona la posición de los obstáculos que son visibles por parte del dron, esta distancia viene dada respecto al propio LiDAR. Sin embargo, la construcción de un ESDF del entorno del dron requiere referir los obstáculos a un sistema de coordenadas global e inercial. Por ello, y teniendo en cuenta que el sensor LiDAR está montado en el vehículo aéreo, se necesita conocer la posición y orientación del sistema de referencia del dron si se quiere obtener la posición de los obstáculos en coordenadas globales.

Como en este trabajo no se realiza una implementación real del sistema y todos los datos experimentales se obtendrán de simulaciones, el método de obtención de la posición y orientación del dron no es relevante en sí mismo, si bien existen multitud de opciones (no exluyentes y que pueden ser utilizadas en conjunto mediante fusión sensorial) entre las que destacan las siguientes:

- Sistemas de posicionamiento global como GPS, GLONASS, Galileo, BeiDou...
- Integración de la posición y orientación con IMU
- Medidas de altitud por barómetro
- SLAM

El cálculo preciso de la posición del dron (y, por tanto, del LiDAR) si bien no resulta un problema en simulación, es de vital importancia a la hora de realizar pruebas en entornos reales.

Con estos dos sensores clave, se obtienen como resultado nubes de puntos que pertenecen a los obstáculos detectados por el LiDAR. Estos datos son suficientes para realizar la navegación con el sistema que se plantea.

3.2 SDF Local

Para que la red neuronal, explicada con mayor detalle más adelante, sea capaz de predecir correctamente los valores del ESDF en cada punto del espacio, es necesario entrenarla. Este entrenamiento debe ser realizado con una lista de puntos de los cuales se conozca el valor real del ESDF. Tanto esta sección como la siguiente se centran justo en resolver es problema: determinar qué puntos del espacio necesita la red neuronal para entrenar y cómo obtener el ESDF real de dichos puntos.

Existen diversos artículos donde se presentan soluciones para el entrenamiento de redes neuronales que almacenan información sobre campos escalares como el ESDF, entre los que destacan *iSDF: Real-Time Neural Signed Distance Fields for Robot Perception* [24] y *HIO-SDF: Hierarchical Incremental Online Signed Distance Fields* [34]. Ambos artículos ofrecen un acercamiento idéntico a la forma de entrenar la red. En ambos se emplean dos métodos que se complementan a la hora de obtener datos para cada etapa de entrenamiento. La premisa principal es que se necesitan dos tipos de puntos:

- Puntos locales, próximos al dron, que aseguren un entrenamiento exhaustivo del entorno inmediato del vehículo aéreo.
- Puntos globales, distribuidos a lo largo de todo el mapa ya explorado, para evitar que la red "olvide" obstáculos pasados en la medida de lo posible.

Comenzando con el cálculo de los puntos locales, este se realiza de forma directa gracias a los datos ofrecidos por los sensores previamente detallados. Para una mejor comprensión del método de cálculo de estos puntos, en la Figura 3.3 se expone un esquema explicativo que lo ilustra al completo.

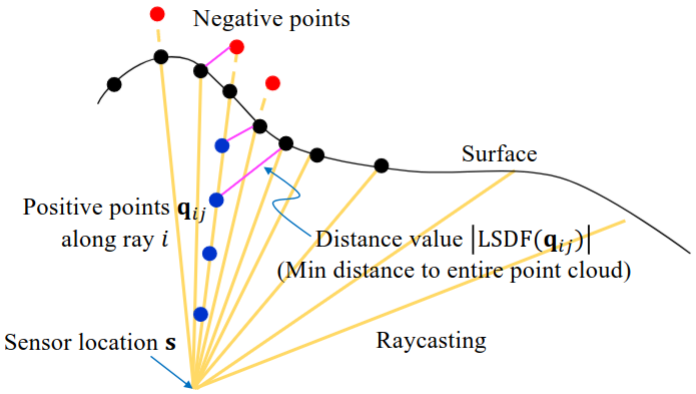


Figura 3.3 Esquema de obtención de puntos locales [34].

En primer lugar, y conociendo tanto la posición del LiDAR como las coordenadas de los obstáculos detectados por el sensor, se puede reconstruir cada haz láser (en la figura, las líneas amarillas). Para cada uno de estos "rayos" del LiDAR, se realiza posteriormente un muestreo de cierto número de puntos, situados aleatoriamente a lo largo de ese vector a una distancia no mayor a $(||p_i - s|| + t)$ desde el sensor, siendo s la posición del sensor, p_i el punto en el obstáculo y t una "distancia de truncado" que asegura que también haya puntos con ESDF negativo en la muestra. Estos puntos muestreados son los que se representan en color azul (los que tienen un ESDF con signo positivo) y rojo (en el caso de los que tienen valor ESDF negativo) en la figura.

Una vez se tienen todos esos puntos, denotados en el artículo y en la figura como q_{ij}, el siguiente paso es obtener su valor ESDF. El cálculo para ello se debe subdividir en dos partes: obtener el valor absoluto y

el signo. La obtención del signo resulta bastante sencilla, pues basta con comparar las distancias del punto muestreado y el punto en el obstáculo al LiDAR. De esta forma:

$$\text{signo}(ESDF(q_{ij})) := \begin{cases} 1 & \text{si } ||q_{ij} - s|| < ||p_i - s|| \\ -1 & \text{en otro caso} \end{cases} \tag{3.1}$$

Para calcular la mejor estimación posible del valor absoluto del ESDF existen diversos métodos. Uno de ellos, de muy bajo coste computacional, podría consistir en aceptar la distancia entre el punto muestreado q_{ij} y el punto en el obstáculo de su mismo rayo p_i como una aproximación aceptable del ESDF. Este método, sin embargo, solo da una aproximación aceptable con puntos muy cerca del obstáculo, y podría llegar a ser problemático en entornos con obstáculos de geometría compleja, donde el valor estimado del ESDF de puntos relativamente alejados del obstáculo podría distar mucho del valor real real.

Para evitar este problema, y dado que el cálculo de estos puntos puede ser paralelizado en GPU para ahorrar tiempo de computación [34](ya que cada uno es independiente de los demás), el método que se utilizará en este trabajo se basa en comparar la distancia de cada punto muestreado a todos los puntos detectados, asumiendo que la menor entre estas distancias es una muy buena aproximación del ESDF. Esto se puede representar matemáticamente de la siguiente forma:

$$||ESDF(q_{ij})|| := \min_{k \in \{1,2,3,\ldots,P\}} ||q_{ij} - p_k|| \tag{3.2}$$

Por último, basta con multiplicar ambos términos para obtener el valor final de la estimación del ESDF de cada punto muestreado:

$$ESDF(q_{ij}) := \text{signo}(ESDF(q_{ij})) * ||ESDF(q_{ij})|| \tag{3.3}$$

El algoritmo que se ha detallado en esta sección tiene muchos grados de libertad que deben ser decididos previamente a la implementación en simulación del sistema completo. Estos grados de libertad son los siguientes:

- El número de rayos del LiDAR donde se realizará el muestreo de puntos de entrenamiento, S
- El número de puntos muestreados en cada rayo, Q
- La distancia de truncado, t

Para la experimentación en simulación que se realizará en capítulos posteriores, se ha decidido respetar los valores propuestos en [34]. En dicho artículo se escoge $S = 1.000$ y $Q = 20$, para un total de 20.000 puntos locales. Por otro lado, el valor propuesto de la distancia de truncado es $t = 0.2m$. Una vez obtenidas las duplas compuestas por las coordenadas de un punto y su ESDF estimado, estas están listas para ser utilizadas en el entrenamiento de la red neuronal.

3.3 SDF Global

Como se explicó en el anterior apartado, los artículos consultados (y en especial [34] y [24]) coinciden en que no basta únicamente con alimentar el entrenamiento de la red con puntos locales y cercanos al dron, sino que es necesario introducir una serie de puntos que representen todo el mapa explorado hasta el momento, con el objetivo de evitar que la red "olvide" zonas por las que ya ha volado con anterioridad. Para muestrear estos puntos, será necesario contar con cualquier tipo de representación global y discreta del ESDF del mapa al completo. Si bien el hecho de guardar un ESDF discreto del mapa en su totalidad parece una contradicción

respecto a lo que se está intentando hacer (pues debería ocupar una gran cantidad memoria), esto no ocurre debido a que la resolución con la que podemos permitirnos guardar este ESDF es bastante reducida, gracias a que solo se muestrearán algunos puntos que pueden estar muy espaciados entre sí.

Para la representación del mapa se va a emplear una herramienta llamada Voxfield [25], siendo la preferida por toda la literatura consultada gracias a su buen rendimiento y consumo de recursos respecto a otras alternativas. En cada iteración de entrenamiento de la red, Voxfield emplea la nube de puntos detectada por el LiDAR para actualizar su versión de baja resolución del ESDF, la cual guarda como una cuadrícula de voxels con una resolución que puede ser modulada según los recursos que podamos destinar a esta tarea.

Para conocer mejor el funcionamiento de Voxfield, se presentan en la Figura 3.4 dos imágenes que muestran diferentes etapas de una de las pruebas en simulación realizadas con el objetivo de probar el sistema al completo (las cuales se detallarán en capítulos posteriores). En ellas se muestran las superficies de los obstáculos que considera Voxfield a la hora de actualizar su ESDF particular. Mientras más cantidad de mapa sea explorado, mayor será el número de obstáculos conocidos por Voxfield y, por tanto, mayor será la exactitud del ESDF calculado para los puntos que obtenidos a partir del muestreo.

(a) Inicio de la simulación.

(b) Final de la simulación.

Figura 3.4 Evolución de la representación de los obstáculos en Voxfield durante una simulación.

A la hora de realizar el muestreo de puntos globales, que debe hacerse considerando todo el mapa explorado, es necesario seguir una estrategia. En el caso del sistema que se va a implementar, y en base a los resultados obtenidos en [34] se ha decidido muestrear, en primer lugar, $N = 10.000$ puntos que se encuentren próximos a la superficie de los obstáculos detectados. Como la resolución con la que se va a trabajar en simulación es de 10 cm (lo que, en este caso, significa que cada voxel tiene 10 cm de lado), se caracterizan como puntos "cercanos" a un obstáculo aquellos cuyo valor estimado del ESDF no supere el límite de los 5 cm. Por otro lado, también se muestrean $M = 30.000$ puntos que se consideran alejados de la superficie, es decir, puntos cuyo ESDF estimado es mayor al límite propuesto anteriormente. El parámetro M, como se verá en el apartado de simulación, es especialmente crítico a la hora de que la red neuronal resultante pueda ser útil para la planificación, pues de él depende que la red en el espacio abierto tenga mayor o menor ruido. Los cambios en la actuación del planificador según el valor de este parámetro M serán tratados con detalle en el Capítulo 5.

Los 40.000 puntos globales obtenidos con este sistema deben ser añadidos a los 20.000 puntos obtenidos a nivel local, obteniendo un total de 60.000 puntos que serán utilizados en cada iteración para entrenar la red neuronal que se describirá a continuación.

3.4 Red Neuronal

3.4.1 Estructura de la red

Una vez explicados los dos métodos para obtener los puntos de entrenamiento, es el momento de detallar la propia red neuronal que se utilizará. Tanto en [24] como en [34], que han sido los dos artículos de referencia

en la síntesis del sistema al completo, los investigadores coinciden completamente en cuanto al tipo y a la estructura principal: una red neuronal densamente conectada, con cuatro capas ocultas de 256 neuronas cada una. Sin embargo, una gran diferencia radica en que en [34] se utiliza una estructura de red tipo SIREN [32] (siglas de *Sinusoidal Representation Network*), es decir, se emplean capas de activación sinusoidales tras cada una de las capas ocultas.

Este tipo de capas de activación parecen dar mejores resultados respecto a las tradicionales (como las ReLUs, que son las que se utilizan en [24]) a la hora de representar campos escalares como los ESDF, gracias a algunas de sus propiedades matemáticas, como que el gradiente de una red SIREN es otra SIREN, lo que permite una supervisión más sencilla del cumplimiento de la condición de Eikonal, explicada en el Capítulo 2 [34]. Además, si bien en la red presentada en [24] es estrictamente necesario modificar las coordenadas de entrada para un buen comportamiento del sistema (con técnicas como el *positional embedding*), con una SIREN esto no resulta imprescindible, pudiendo introducir las coordenadas de un punto sin previo procesado. Por todo esto, se ha decidido emplear una red tipo SIREN en este trabajo.

En la Figura 3.5 se presenta un esquema de la estructura de la red neuronal que se ha empleado. La entrada, representada arriba, se corresponde con las tres coordenadas de un punto del espacio, y la salida es un único valor que, una vez entrenada la red, debería corresponderse con una estimación lo suficientemente buena del valor del ESDF en dicho punto.

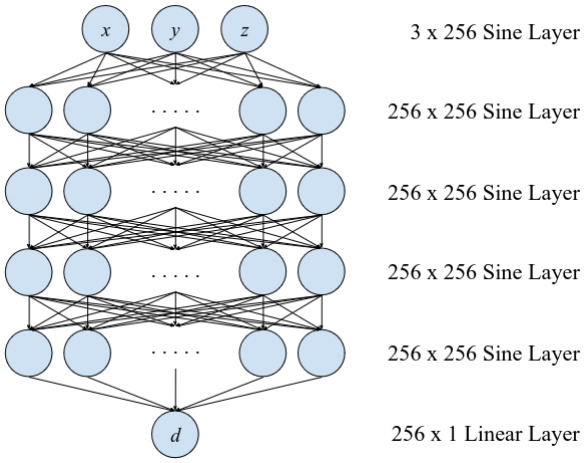


Figura 3.5 Esquema de la red empleada para estimar el ESDF [28].

Como ya se comentó durante el Capítulo 2, la red neuronal tendrá una estructura fija independientemente del tamaño del mapa, lo cual se explica teniendo en cuenta que el objetivo de este sistema es su utilización para planificación local, donde el tamaño de lo que se puede considerar el entorno local del dron sería constante y conocido. Como la red es invariable, la cantidad de términos que deben se guardados en memoria también lo es, y puede ser calculada fácilmente con anterioridad con el objetivo de determinar la memoria ocupada. Estos parámetros de la red se pueden dividir en dos tipos: los pesos asociados a cada par de neuronas y el sesgo (o "bias") de cada neurona.

Los pesos de una red neuronal son términos que dictan en qué proporción se suman los datos de entrada (los cuales se corresponden a los valores de las neuronas de la capa anterior) a cada neurona, mientras que el sesgo es un término constante que se suma a los valores de entrada. La suma ponderada de valores de neuronas de la capa anterior más el sesgo dan como resultado el valor de la neurona en cuestión. De esta forma, si tenemos cierta neurona j, podemos calcular el valor n_j que toma a través de la siguiente suma ponderada:

$$n_j := n_1 * w_{j1} + n_2 * w_{j2} + n_3 * w_{j3} + ... + n_i * w_{ji} + b_j \tag{3.4}$$

Los términos n_x son los valores de las neuronas de la capa anterior, mientras que los términos w_{jx} son sus pesos para la neurona j. Por último, b_j es el sesgo de la neurona j. Sabiendo la estructura de la red, es sencillo calcular el total de parámetros que guardar en memoria:

$$\begin{aligned} parmetros := pesos + sesgos &= (3*256 + 3*256^2 + 256) + (3 + 4*256 + 1) \\ &= 197.632 + 1.028 = 198.660 \end{aligned} \tag{3.5}$$

Debido a que la evaluación del sistema se realizará enteramente en simulación, la cantidad de parámetros no resulta relevante, si bien es algo a tener en cuenta en futuras pruebas en entornos reales.

3.4.2 Entrenamiento de la red

Una vez descrita la estructura de la red neuronal empleada, es necesario detallar la forma en la que se va a entrenar. El entrenamiento de la red tiene como objetivo el ajuste de los parámetros de esta para que el valor que tome la neurona de salida, ante cierta entrada, sea lo más parecido al esperado. En este caso, el objetivo que se persigue con el entrenamiento de la red es que ofrezca en su neurona de salida (denotada con una d en la Figura 3.5) un valor lo más cercano posible al valor real del ESDF en el punto descrito por los valores de las neuronas de entrada.

Existen numerosos métodos de entrenamiento para redes neuronales, siendo este un campo en constante expansión en la actualidad. Sin embargo, el método más común y, además, el que se utiliza en [34], es el llamado *backpropagation* (o, en español, propagación hacia atrás de los errores). Para poder llevar a cabo el entrenamiento, se necesitan una serie de muestras en forma de pares de valores, siendo uno de ellos una entrada a la red y el otro lo que se consideraría la salida "correcta" de la red. Estas muestras son las que proporcionan el SDF Local y el SDF Global que se explicaron anteriormente. Este algoritmo es secuencial:

1. Se eligen los puntos de salida y entrada para el algoritmo, los cuales dependen de las muestras dispobibles para el entrenamiento. En este caso, lo que el algoritmo considera entrada y salida coincide con la entrada y la salida de la propia red.
2. Secuencialmente, se introducen las entradas de las muestras en las neuronas de entrada y se deja que la red ofrezca una salida estimada en su neurona de salida.
3. Para cada entrada, la red determina el error total en la salida (es decir, la diferencia entre la salida correcta, proporcionada por la muestra, y la salida de la red).
4. Basándose en una determinada función de pérdida (o *loss function* en inglés), el algoritmo ajusta los parámetros internos de la red para minimizar dicha función para el conjunto total de muestras contemplado.

La función de pérdida a utilizar debe penalizar los comportamientos no deseados de la red. Si bien existen funciones muy simples (que se basan, por ejemplo, únicamente en la diferencia entre la salida deseada y la salida actual), implementar una mayor complejidad en ellas pueden ofrecer un resultado más satisfactorio. Tanto en [34] como en [24] se utilizan fórmulas idénticas dadas por

$$\mathscr{L}_{\text{total}}(\theta) := \lambda_{\text{SDF}}\mathscr{L}_{\text{SDF}}(\theta) + \lambda_{\text{Eikonal}}\mathscr{L}_{\text{Eikonal}}(\theta) \tag{3.6}$$

para unos pesos de la red θ. El primer término se encarga de penalizar el error en la salida, siendo el primer término una constante de ponderación, en este caso $\lambda_{\text{SDF}} = 5.0$, y el segundo término la diferencia entre la salida actual de la red y la deseada:

$$\mathscr{L}_{\text{SDF}}(\theta) := \mathscr{L}_{\text{SDF}}(f_\theta(x), s) = |f_\theta(x) - s| \tag{3.7}$$

Esta función está evaluada para una entrada x y siendo s el valor real (o, más bien, el ofrecido por la muestra, que es el valor más próximo al real al que se tiene acceso). Por otro lado, el segundo término se encarga de que la salida de la red cumpla la ecuación de la eikonal, una ecuación que debe cumplir el ESDF en casi todos sus puntos para ser correcto, como se detalló en el Capítulo 2. Como en el otro término, este está compuesto por una constante de ponderación $\lambda_{\text{Eikonal}} = 2.0$ y un segundo término de la forma

$$\mathscr{L}_{\text{Eikonal}}(\theta) := \mathscr{L}_{\text{Eikonal}}(\nabla f_\theta(x)) = ||\nabla f_\theta(x)| - 1| \tag{3.8}$$

Una vez repasada la función de pérdida, aún quedan algunos otros parámetros por determinar. Para realizar el proceso de entrenamiento, el conjunto completo de muestras obtenidas del SDF Global y el SDF Local se utiliza varias veces. Las veces que un conjunto de datos es pasado a través del algoritmo para realizar el proceso completo de backpropagation se denomina *epoch* (o época), siendo este un parámetro de suma importancia que debe ser ajustado con precisión. Una baja cantidad de épocas podría resultar en un mal ajuste de la red, mientras que una gran cantidad de épocas podría conllevar la aparición de un efecto llamado *overfitting*, donde la red se ajustaría muy bien a las entradas presentes en las muestras, pero muy mal a entradas no presentes en ellas.

Teniendo en cuenta que la estructura de la red es similar a la presente en [34], parece plausible aceptar, al menos en primera instancia, los parámetros de entrenamiento de dicho artículo. En este caso, lo que propuesto por los autores es separar el entrenamiento en dos fases, según el tiempo que lleve el sistema funcionando:

- En las primeras 5 iteraciones tras la puesta en marcha del sistema, se entiende que la red (cuyos parámetros iniciales son aleatorios) necesitará una mayor cantidad de épocas de entrenamiento para que su salida sea suficientemente buena. Es por ello que para estas primeras iteraciones se contemplan 50 *epochs*.
- Una vez superado el periodo de puesta en marcha del sistema, el valor para las demás iteraciones es de 10 *epochs*.

Continuando con más parámetros, la red neuronal utiliza las muestras disponibles (un total de 40.000 si se mantienen los parámetros originales de [34]) para realizar el ajuste de los parámetros de la red con el método de *backpropagation*. Sin embargo, es una práctica común no utilizar todas las muestras presentes de manera simultánea, separándolas en grupos llamados *batches*. Esto se realiza, principalmente, debido a que ajustar los parámetros de la red respecto a una gran cantidad de muestras (es decir, llevar a cabo un problema de optimización de tantos grados de libertad) conlleva un coste computacional elevado. Sin embargo, parece que en el caso de este sistema no es necesaria la separación de las muestras, por lo que se utilizarán todas a la vez en un único *batch*.

Para terminar con los detalles del entrenamiento de la red, se ha escogido el optimizador Adam, con 0.012 de decaimiento de pesos (*weight decay*) y 0.0004 de ratio de aprendizaje (*learning rate*), todo esto propuesto en [34].

3.5 Planificador

Una vez se termina con el entrenamiento de la red en cada iteración, lo que se obtiene es una red neuronal en un estado determinado por el valor de sus parámetros. El objetivo del bloque planificador es, como su propio nombre indica, realizar la planificación de la trayectoria. Para ello, este bloque debe importar el estado más actualizado de la red para poder utilizarlo en conjunto con algún algoritmo de planificación. En el caso de este trabajo, todas las simulaciones van a realizarse utilizando un A*.

El algoritmo A* ya ha sido descrito en su forma general en el Capítulo 2. Sin embargo, el algoritmo descrito en dicho capítulo no tendría en cuenta el ESDF a la hora de realizar la planificación, por lo que es necesario realizar cambios. Como se explicó anteriormente, el algoritmo A* utiliza dos funciones de coste diferentes para determinar el coste total de un determinado nodo: una función $g(n)$, que representa el coste que supone viajar desde el inicio de la trayectoria al nodo en cuestión, y una función $h'(n)$, que representa una estimación del coste de viajar del nodo actual al nodo final.

La principal utilidad de tener un ESDF para planificación radica en que se puede modificar la función $g(n)$ para que el planificador penalice pasar cerca de obstáculos. En el Algoritmo 1, presente en el Capítulo 2, se puede ver que esta función g de un nodo se calcula en función de la de su nodo padre, a la cual se le suma un cierto término de coste. De esta manera se llega a la definición de la función g que se utilizará en el sistema, que tiene la forma

$$g(n_{i+1}) = g(n_i) + c(n_i, n_{i+1}) \tag{3.9}$$

siendo $g(n_{i+1})$ la función del nodo hijo, $g(n_i)$ la función del nodo padre y $c(n_i, n_{i+1})$ una cierta función que determina la diferencia de coste entre ambos. Esta última tiene la siguiente forma:

$$c(n_i, n_{i+1}) = \|n_{i+1} - n_i\| + \frac{c_w}{O(n_i, n_{i+1})} \tag{3.10}$$

El primer término presente en esta ecuación representa la distancia entre ambos nodos, y es el término que habitualmente está presente en el algoritmo A*. Sin embargo, el segundo término modela un coste asociado a la distancia a los obstáculos, siendo c_w cierto factor ponderante y $O(n_i, n_{i+1})$ el coste asociado a la distancia. La forma más directa de definir este último término (y la utilizada en este trabajo) es igualarlo a la distancia al obstáculo más cercano, es decir, igualarlo al valor del ESDF en el nodo hijo. De esta manera, a más cerca de un obstáculo tenga que pasar el dron, mayor será el segundo término de $c(n_i, n_{i+1})$, aumentando el coste total de viajar a ese nodo y, como consecuencia, favoreciendo nodos que pasen más lejos de los obstáculos.

Con esto, se ha detallado el funcionamiento del sistema al completo en el ámbito teórico. En el siguiente capítulo se explicará cómo se ha realizado la implementación práctica.

4 Implementación del sistema

Una vez detallada toda la teoría necesaria para comprender el funcionamiento del sistema que se quiere implementar, el siguiente paso es la propia implementación. En primer lugar, se va a introducir brevemente el soporte sobre el cual se va a implementar el sistema, para posteriormente describir en detalle cómo se han implementado las funcionalidades de cada bloque del sistema.

4.1 ROS: el estándar actual en robótica

En el mundo de la robótica, la colaboración eficiente entre hardware y software es algo completamente necesario y esencial para el desarrollo y la implementación de sistemas robóticos avanzados. ROS (siglas de *Robot Operating System*) es una herramienta ampliamente utilizada en el mundo de la robótica que ha posibilitado transformar la forma en la que se diseña, desarrolla y opera la robótica moderna [3].

Se trata de un entorno de código abierto que fue diseñado de forma específica para responder a las necesidades de programación que conllevan las aplicaciones robóticas complejas. No es un sistema operativo al uso, sino que se trata de una serie de herramientas, librerías y convenciones que facilitan enormemente el desarrollo de proyectos del calibre de este trabajo.

La historia de ROS se remonta a 2007, donde nació a través de una colaboración entre el Stanford AI Lab y el Willow Garage Artificial Intelligence Lab. Su primera versión se lanzó de forma pública en 2010, y desde entonces su desarrollo ha continuado de forma ininterrumpida, convirtiéndose en un estándar de casi obligatorio uso en el ámbito de la investigación en robótica [36].

La principal ventaja de ROS es su modularidad. Lo que provee es una base flexible que permite a los desarrolladores construir sistemas complejos a través de nodos interconectados entre sí. Cada nodo representa un determinado proceso, y diferentes nodos pueden comunicarse entre sí a través de una estructura estándar de mensajes. Al ser la estructura tan descentralizada, es enormemente sencillo reutilizar código, emplear bloques en diferentes proyectos y potenciar la escalabilidad de los sistemas.

Además de su estructura muy amigable para proyectos de alta complejidad, ROS también ofrece una gran cantidad de herramientas y librerías, en permanente expansión, relacionadas con campos tan dispares como la planificación de trayectorias, la navegación, la manipulación, la percepción y el control automático, entre otros.

4.2 Implementación del sistema en ROS

La modularidad que caracteriza a ROS es algo que lo convierte en la plataforma ideal para sistemas como los desarrollados en este trabajo. En esta sección se va a decribir cómo se ha realizado la implementación de todo lo explicado anteriormente en la plataforma de ROS, comenzando por su estructura a grandes rasgos.

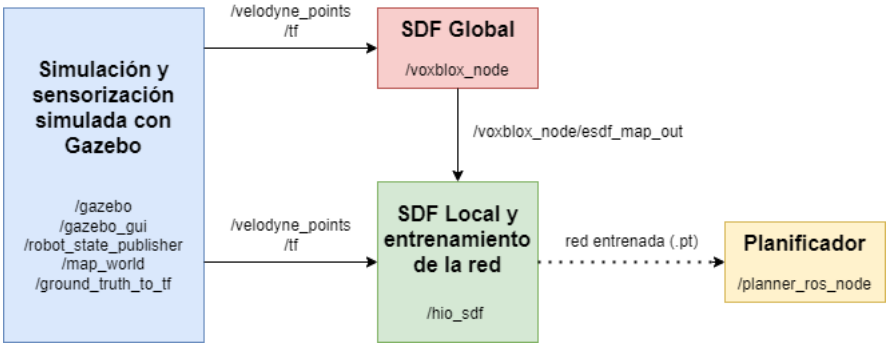


Figura 4.1 Esquema de la arquitectura del sistema en ROS.

Como se explicaba en la sección 4.1, ROS permite la modularidad a través de la puesta en marcha de varios nodos, cada uno implementando una funcionalidad específica, que luego pueden ser conectados entre sí a través de *topics*. En la Figura 4.1 se presenta un esquema general simplificado del sistema de ROS, donde cada bloque presente contiene uno o varios nodos que, en conjunto, implementan una cierta funcionalidad. Los bloques presentes son los siguientes:

- Un bloque de varios nodos que implementa la herramienta principal de simulación (Gazebo, de la que se hablará más adelante) y los sensores simulados.
- Un nodo que implementa el SDF Global a través de Voxfield [25].
- Un bloque que implementa tanto el SDF Local como el sistema de entrenamiento de la red, basado en [34].
- Un bloque que implementa el algoritmo de planificación.

Las líneas entre los diferentes bloques en la figura incluyen los nombres de los *topics* utilizados para compartir la información necesaria entre cada bloque de nodos. La conexión entre el nodo de entrenamiento de la red y el planificador es una excepción, pues su mecanismo de intercambio de información no incluye un *topic*. En esto se ahondará más adelante.

A lo largo de las siguientes subsecciones se va a explorar cada nodo o conjunto de nodos, explicando sus principales características y detallando cómo se interconectan entre sí.

4.2.1 Sensorización simulada con Gazebo

Como ya se ha descrito anteriormente, las entradas del sistema presentado son una serie de datos recogidos por sensores, los cuales serán procesados para alimentar a la red. Para poder probar el sistema y obtener resultados sobre su viabilidad, es necesario la implementación de un sistema completo de simulación, el cual se ha realizado con una de las herramientas más comunes para ello que están presentes en ROS: Gazebo.

Gazebo [1] es un simulador de código abierto orientado a la robótica que se integra de forma estrecha con ROS para proporcionar un entorno de simulación física 3D de gran realismo. En este entorno se pueden realizar simulaciones con robots para probarlos y desarrollarlos sin necesidad de tener hardware físico.

Una de las principales ventajas de Gazebo es que no solo permite simular la mecánica de un robot, sino también sus sensores y actuadores. En este trabajo, se ha utilizado Gazebo para simular un dron con un LiDAR incorporado. El entorno gráficopuede verse a modo de ejemplo en la Figura 4.2.

Los diversos nodos de este bloque de sensorización construyen el entorno de Gazebo con el dron situado sobre un mapa (del cual se hablará más adelante, en el Capítulo 5), permitiendo total libertad para realizar simulaciones con el sistema. Las principales salidas que son necesarias para el funcionamiento del sistema son dos:

- El topic */velodyne_points*, a través del cual se envían las nubes de puntos detectadas por el LiDAR.

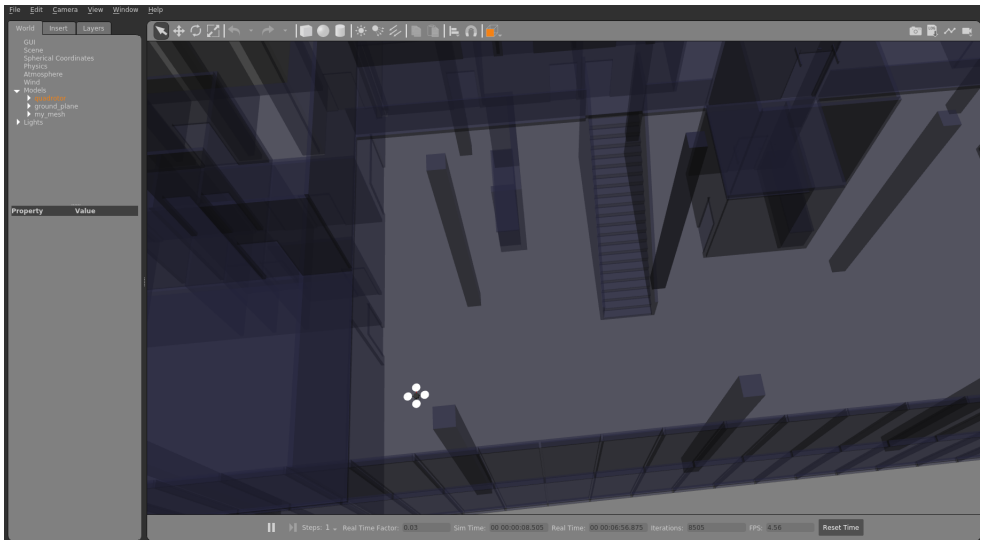


Figura 4.2 Entorno gráfico de Gazebo.

- El topic */tf*, a través del cual se envía la información sobre las transformaciones espaciales entre el sistema de coordenadas global y el sistema de coordenadas asociado al LiDAR.

Conociendo la información sobre la transformación de coordenadas entre el sistema global y el sistema local asociado al LiDAR, se puede convertir la nube de puntos generada por este sensor a coordenadas globales. Esta nube de puntos en coordenadas globales es la que necesitan recibir los nodos que se explicarán a continuación.

4.2.2 SDF Global: Voxfield

Uno de los bloques que necesitan los datos de los sensores (en este caso, simulados) es el que implementa el SDF Global, explicado con detalle en el Capítulo 3 y cuyo objetivo es obtener una lista de muestras para el entrenamiento basándose en una versión de muy baja resolución de un ESDF.

Como se explicó también en capítulos anteriores, este SDF Global está implementado con Voxfield, un paquete de ROS ya asentado, cuya documentación puede encontrarse en [25]. El paquete implementa una representación 3D del entorno utilizando *voxels*, pequeños volúmenes cúbicos, parecidos a píxeles en 3D. A partir de agrupaciones de estos *voxels*, cada uno almacenando cierta información (en este caso, el ESDF del punto central del espacio que representan), Voxfield crea un campo tridimensional que puede ser consultado a voluntad.

El sistema completo de Voxfield se implementa en un único nodo, escrito completamente en C++, cuyo código es abierto y se ha utilizado de forma íntegra, cambiando algunos parámetros para realizar las conexiones pertinentes a los otros bloques del sistema (por ejemplo, el nombre de los topics de entrada). Como ya se comentó al principio de este capítulo, una de las principales ventajas de ROS es su modularidad y la facilidad con la que se puede reciclar código, siendo este un ejemplo perfecto de esto último.

La principal razón por la cual se ha usado el sistema de Voxfield frente a otros paquetes con funcionalidades similares es su gran eficiencia en cuanto a consumo de recursos [34] [24], algo especialmente importante en sistemas que deben estar embarcados en un dron de capacidad computacional limitada.

Voxfield cuenta con una gran cantidad de *topics* de salida, entre los que se incluyen varios tipos de representaciones del espacio diferentes. Sin embargo, el nodo que se explicará a continuación solo necesita la información del mapa de voxels, el cual se envía a través del topic *voxblox_node/esdf_map_out*. De esta manera, el muestreo de puntos de dicho mapa para el entrenamiento de la red no se realiza en el propio nodo de Voxfield, sino en el nodo que se encarga del entrenamiento.

4.2.3 SDF Local y entrenamiento de la red: HIO-SDF

El siguiente bloque en la cadena de información recibe tantos los datos de los sensores como los datos del mapa generado por Voxfield y tiene, como salida, la versión más actualizada de la red hasta el momento. Este nodo, por tanto, tiene tres tareas fundamentales:

- Muestrear el mapa generado por Voxfield (SDF Global)
- Calcular los puntos asociados al SDF Local
- Realizar la implementación y el entrenamiento de la red neuronal

Para la implementación de este nodo, se ha utilizado el código público asociado al paper HIO-SDF[34], donde se implementa un sistema casi idéntico al que se quiere implementar en este trabajo, capaz de generar un ESDF a través de una red neuronal. En ese artículo, sin embargo, no se utiliza la red para ningún propósito específico más allá de para representar el espacio de forma fidedigna. El código de nodo ha sido editado para adaptarse a las nuevas entradas.

La implementación del HIO-SDF se realiza con un único nodo, escrito completamente en Python, donde se realiza de forma secuencial la obtención de las muestras para el entrenamiento (primero las muestras globales y después las locales) y el posterior entrenamiento de una red neuronal. Todo el código asociado a la red neuronal utiliza el paquete "pytorch", cuya documentación puede encontrarse en [2], y que tiene la especial ventaja de tener una contraparte en C++ (llamada "libtorch") que será empleada en el nodo posterior.

A la hora de obtener la red entrenada para que la use el planificador, existen diversas formas de comunicar ambos nodos. En un principio, lo más indicado parecía transmitir los parámetros de la red a través de un *topic* que sería recibido por el planificador, el cual copiaría los parámetros en su versión de la red. Esta implementación, sin embargo, da lugar a errores prácticos cuando un nodo utiliza pytorch y, el otro, libtorch, como es el caso.

En consecuencia, la solución final adoptada ha consistido en que el nodo de entrenamiento guarde en cada iteración la versión actualizada de la red como un archivo .pt, el cual guarda la estructura y los parámetros de la red. Este archivo será importado por el nodo planificador cada vez que realice una planificación.

4.2.4 Planificador

El último nodo del sistema es el que implementa los planificadores. El código implementado es una versión modificada del utilizado en [9]. Este paquete implementa una gran cantidad de planificadores diferentes, incluyendo varias versiones de A*, Theta* y Lazy-Theta*, además de ofrecer una interfaz muy amigable a la hora de incluir otro tipo de planificadores.

Para este trabajo, se ha modificado el código para que importe la red neuronal generada por el nodo anterior. De esta forma, y habiendo implementado un planificador que utiliza la versión modificada del algoritmo A* explicada en el Capítulo 3, el nodo realizará una consulta a la red cada vez que necesite el valor ESDF de un punto en concreto del espacio.

Con estos cuatro bloques, se obtiene un sistema funcional que se basa en una simulación y sus correspondientes sensores simulados para implementar de forma completa el sistema deseado. En el siguiente capítulo se explicarán los diferentes experimentos que se han realizado para poder validar y verificar el correcto funcionamiento del sistema.

5 Simulaciones y validación experimental

El objetivo de este capítulo es simular el sistema desarrollado en el Capítulo 4 y, además, validar experimentalmente el planificador con el ESDF creado por red neuronal integrado. Para realizar esta validación, la metodología utilizada va a ser comparar el planificador objeto del trabajo (que a partir de ahora pasará a llamarse A*-SIREN para una mayor claridad) con un planificador casi idéntico, pero que utiliza una versión del ESDF generada de forma *offline* y cargada en una matriz en memoria (que pasará a llamarse A*-M1 [9]). De esta manera, se estará comparando el planificador basado en la red neuronal con su versión "perfecta" (debido a que el SDF generado de manera offline es exacto). Así, el método para validar el planificador en estudio es comprobar cómo de similar es su trayectoria generada respecto a la generada por el A*-M1. También se compararán ambos algoritmos con un A* convencional para comprobar cómo cambian las trayectorias calculadas al tener en cuenta la distancia a los obstáculos.

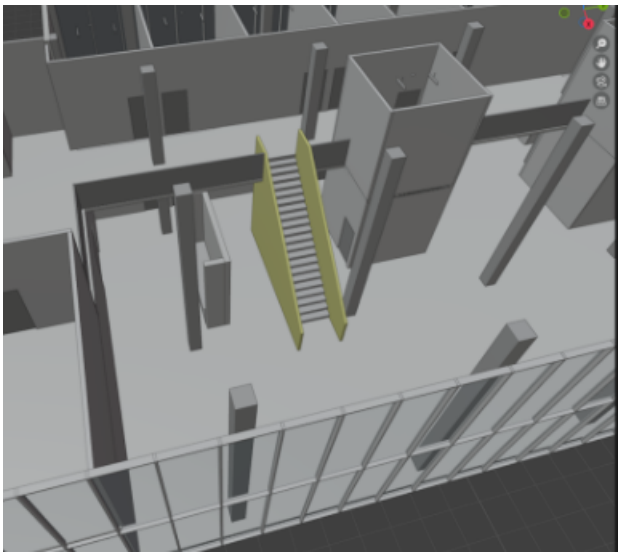


Figura 5.1 Escenario de simulación.

El escenario sobre el que se va a mover el dron durante las simulaciones se puede ver en la Figura 5.1. Se trata de un modelo de una de las plantas de un edificio de la Eindhoven University of Technology [26], del cual se utilizarán solo ciertas zonas. En este entorno se realizarán dos experimentos con trayectorias diferentes que pondrán a prueba diferentes aspectos de la red neuronal y el planificador, para así validar el funcionamiento del algoritmo de la forma más exhaustiva posible.

Antes de comenzar con la descripción de los diferentes experimentos, y como se dejó entrever en el Capítulo 3, es necesario aclarar que el sistema cuenta con una gran cantidad de grados de libertad en forma de constantes, que afectan en mayor o menor medida al resultado de la planificación. De todos estos parámetros, se ha encontrado que el parámetro M del SDF Global (véase el apartado 3.3), es decir, el número de puntos

de muestra asociados al SDF Global que se toman lejos de la superficie, tiene una influencia enorme en la calidad de los resultados durante las simulaciones.

El motivo de esto tiene su origen en que para planificar es necesario que el ruido a la salida de la red sea bastante reducido. Si el error fuera muy significativo, el planificador daría lugar a rutas erráticas y nada parecidas a la solución óptima, como se verá en los experimentos. El dron únicamente va a navegar por el espacio abierto y lejos de los obstáculos, por lo que un mayor M permitirá un mejor modelado de este espacio y, por tanto, conseguirá mejores resultados.

En los dos experimentos que se detallan a continuación, se realizan pruebas con diferentes valores del parámetro M desde los 30.000 puntos (siendo este el valor propuesto en [34]) hasta los 70.000 puntos.

5.1 Escenario 1 - Esquivar una columna

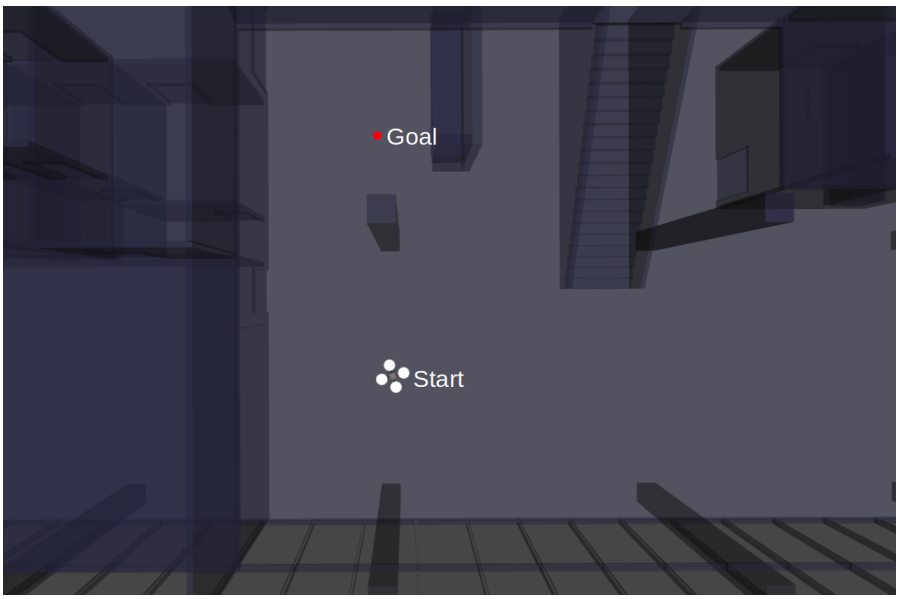


Figura 5.2 Punto inicial y punto final - Escenario 1.

El primer escenario, que se puede ver en la Figura 5.2, tiene como objetivo que el dron esquive una columna. Tanto en este escenario como en el siguiente, se va a comprobar la influencia de dos factores en la calidad de la planificación:

- Por un lado, se va a variar el parámetro M, tomando valores entre 30.000 y 70.000 puntos, con saltos de 10.000 puntos.
- Por otro lado, se van a realizar comparaciones entre el resultado de planificar con diferentes estados de entrenamiento de la red. Para ello, se ha movido el dron por el escenario, guardando cuatro estados de la red distintos. Las posiciones donde se han tomado estos estados de entrenamiento de la red pueden verse en la Figura 5.3, siendo $w1$ el menos entrenado y $w4$ el más entrenado.

Considerando estos dos grados de libertad, se obtienen un total de 20 posibles combinaciones (al tener cinco valores de M y cuatro estados de la red distintos), que se van a comparar entre sí y con los algoritmos A* y A*-M1.

La Tabla 5.1 recoge una medida del error de las 20 trayectorias calculadas. En este caso, la medida del error se obtiene calculando la distancia de cada *waypoint* de la trayectoria generada por el planificador A*-SIREN al *waypoint* más cercano de la trayectoria del planificador A*-M1, y normalizando después al dividir este

valor entre el número total de *waypoints* de la trayectoria generada por el algoritmo A*-SIREN. Esta medida permite cuantificar cómo de lejos está cada trayectoria generada por el A*-SIREN de la trayectoria óptima.

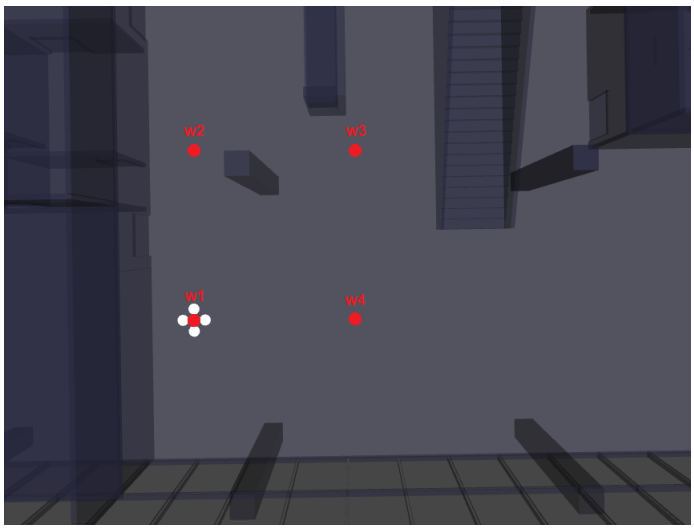


Figura 5.3 Estados de la red considerados - Escenario 1.

Se puede observar que el error cometido disminuye tanto al aumentar el parámetro M como al tomar estados de entrenamiento de la red más tardíos. También es notorio que, si bien existe una gran reducción del error entre los estados de entrenamiento $w1$ y $w2$, esta reducción del error ya no se observa entre estados consecutivos de entrenamiento posteriores (por ejemplo, entre los estados de entrenamiento $w3$ y $w4$, donde incluso hay trayectorias en las que el error aumenta). Esto se debe a que este primer escenario no tiene muchos obstáculos intermedios, lo que permite que la red esté suficientemente bien entrenada desde etapas muy tempranas de entrenamiento.

Esto, además, se puede comprobar gráficamente en la Figura 5.4, donde están representadas cuatro de las trayectorias más significativas. Lo más notorio en esta figura es que, si bien el algoritmo A* decide pasar la columna por la derecha, los otros algoritmos la pasan por la izquierda. Esto se explica debido a que, aunque el camino de la derecha es objetivamente más corto, también pasa muy pegado a varios obstáculos, algo que tanto el A*-SIREN como el A*-M1 tienden a evitar.

Otra parte importante es que incluso la trayectoria generada con el algoritmo A*-SIREN que tiene un mayor error (es decir, la generada con $M = 30.000$ y en la etapa de entrenamiento $w1$, representada en verde claro) es muy similar a la óptima (la generada por el A*-M1, representada en azul oscuro). La mejor trayectoria del A*-SIREN, de hecho, es casi imperceptible al estar solapada casi en su totalidad con la trayectoria calculada por el algoritmo A*-M1.

Con estos resultados, se puede concluir que en este escenario, el algoritmo A*-SIREN da muy buenos resultados, encontrando satisfactoriamente una trayectoria muy similar a la óptima incluso con pocas iteraciones de entrenamiento y pocas muestras en el espacio abierto.

Por último, y antes de pasar al segundo escenario, es interesante hacer notar que, aunque las trayectorias generadas por el algoritmo A*-SIREN no parecen ser muy diferentes entre sí, la red neuronal sí sufre una evolución durante el entrenamiento. Para ejemplificar esto se presenta la Figura **??**, donde se ha representado la salida de la red neuronal al principio del entrenamiento (es decir, en el estado $w1$) y al final del entrenamiento (es decir, en el estado $w4$) para tres alturas distintas, con $M = 30.000$. Cuantas más iteraciones de entrenamiento se llevan a cabo, mejor es la representación del entorno que guarda la red, reduciendo el ruido a la salida y proporcionando más fiabilidad en su predicción del valor ESDF de cada punto.

Tabla 5.1 Distancia media entre los waypoints de cada trayectoria calculada por el algoritmo A*-SIREN al waypoint más cercano de la trayectoria calculada por el algoritmo A*-M1, entre el número total de waypoints - Escenario 1.

num_voxfield _samples_freespace	Weights of the network			
	w1	*w2*	*w3*	*w4*
30000	1.002	0.795	**0.551**	0.669
40000	0.718	0.531	0.744	**0.513**
50000	1.118	0.638	**0.450**	0.669
60000	0.948	0.632	0.726	**0.620**
70000	0.671	0.494	0.450	**0.437**

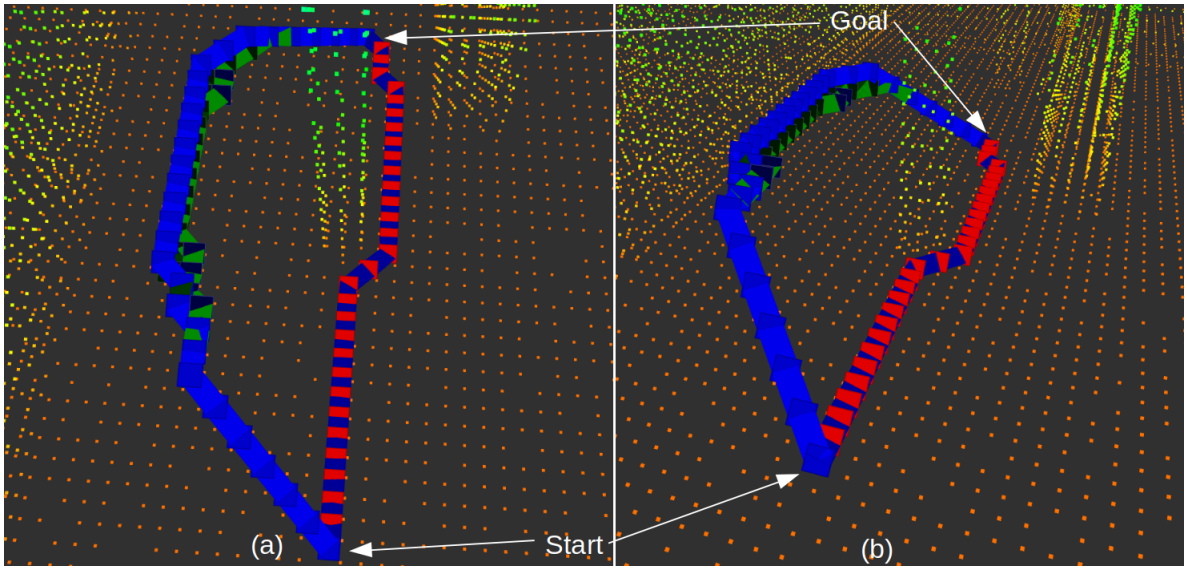


Figura 5.4 Ejemplos de trayectorias generadas - Escenario 1. En rojo, trayectoria generada por el algoritmo A*. En azul, trayectoria generada por el algoritmo A*-M1. En verde claro, trayectoria generada por el algoritmo A*-SIREN (M=30.000, w1). En verde oscuro, trayectoria generada por el algoritmo A*-SIREN (M=70.000, w4)..

5.2 Escenario 2 - Doblar una esquina

El segundo escenario que se va a contemplar consiste en que el dron doble una esquina. La posición inicial y la posición final de la trayectoria deseada se representan en la Figura 5.5. Este escenario tiene una gran diferencia respecto al primero: el dron no empieza con visibilidad directa sobre gran parte del entorno de la trayectoria. Este hecho permitirá comprobar qué errores comete el planificador cuando trata de calcular una trayectoria con estados de la red donde la propia red desconoce una gran parte del entorno.

Para entrenar el dron, se ha realizado un vuelo manual a través del entorno, tomando cuatro estados diferentes de entrenamiento, al igual que se hizo en el experimento anterior. En la Figura 5.6 se pueden ver las cuatro posiciones en las que se han guardadolos estados de entrenamiento de la red, siendo $w1$ el correspondiente a la etapa más temprana de entrenamiento y el $w4$ a la más tardía.

Como en el anterior caso, se han dispuesto en la Tabla 5.2 los errores en las trayectorias calculadas por el algoritmo A*-SIREN respecto a la trayectoria dada por el algoritmo A*-M1 (que, recordemos, es la óptima al considerar este último algoritmo una versión perfecta del ESDF, generada de forma *offline* con anterioridad). En este caso se pueden observar errores mucho mayores que en el primer escenario, con diferencias muy acusadas entre los estados $w1$ y $w2$ para la gran mayoría de valores de M.

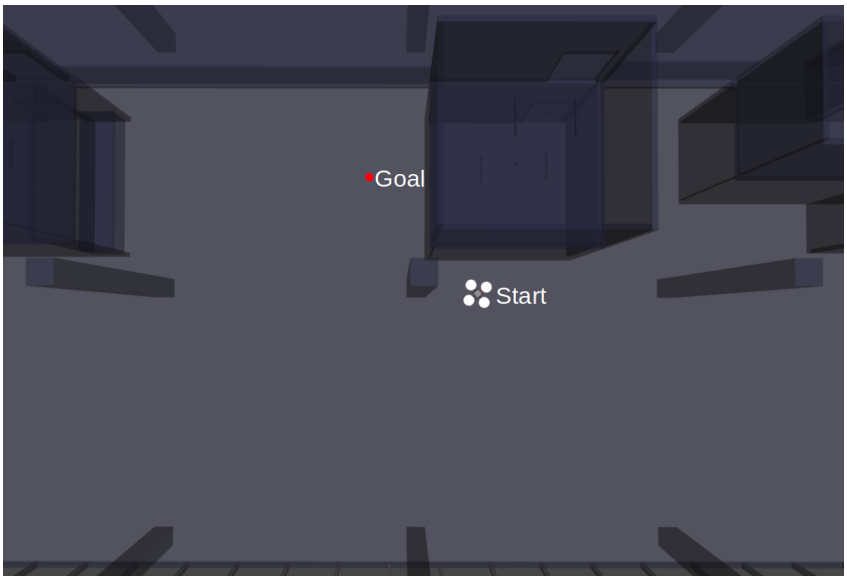


Figura 5.5 Punto inicial y punto final - Escenario 2.

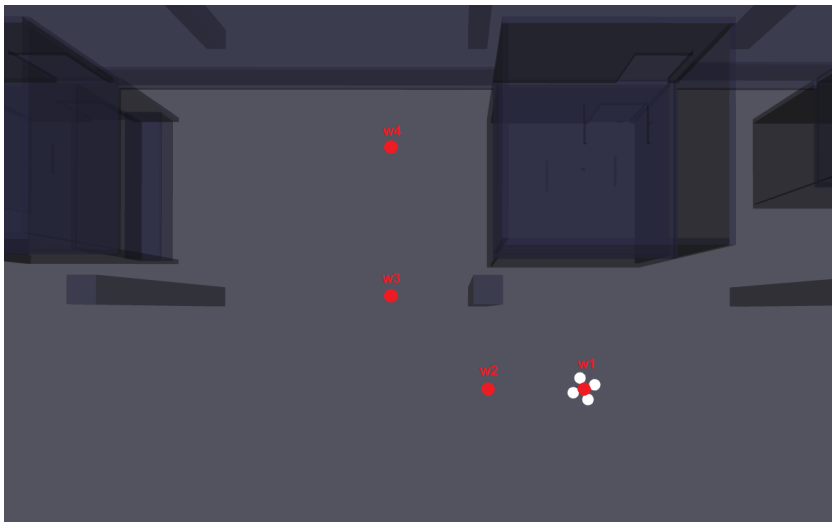


Figura 5.6 Estados de la red considerados - Escenario 2.

Tabla 5.2 Distancia media entre los waypoints de cada trayectoria calculada por el algoritmo A*-SIREN al waypoint más cercano de la trayectoria calculada por el algoritmo A*-M1, entre el número total de waypoints - Escenario 2.

num_voxfield _samples_freespace	**Weights of the network**			
	w1	***w2***	***w3***	***w4***
30000	2.731	1.291	1.245	**1.206**
40000	1.432	1.496	1.055	**0.843**
50000	2.751	1.335	**0.875**	1.079
60000	2.701	1.182	0.966	**0.783**
70000	1.409	1.385	1.170	**0.945**

Si se comparan las trayectorias de forma gráfica, en este escenario sí que se puede ver una gran variación entre los diferentes algoritmos. En la Figura 5.7 se pueden ver tres trayectorias comparadas. En este caso, no se ha considerado de interés incluir en esta gráfica la trayectoria proporcionada por el algoritmo A*, ya que

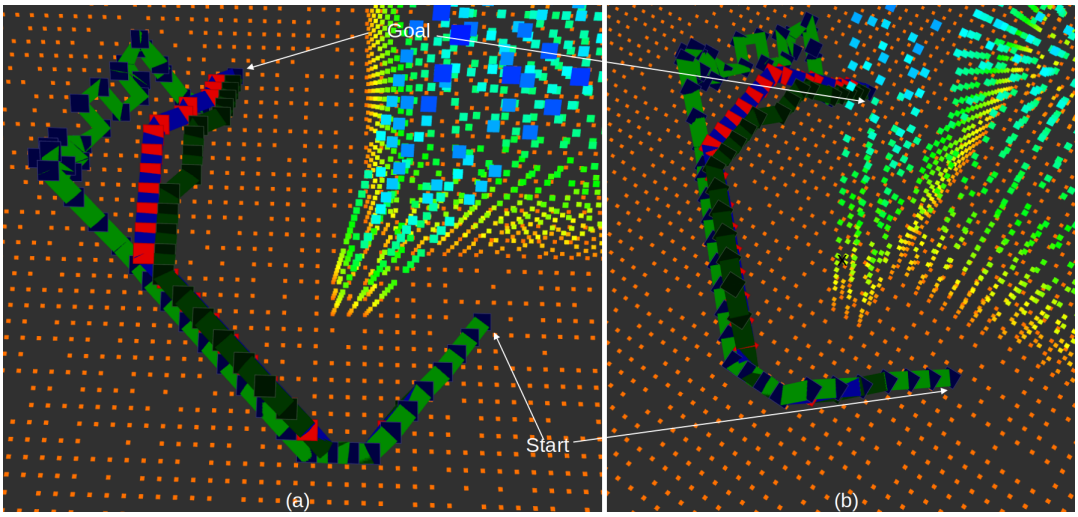


Figura 5.7 Ejemplos de trayectorias generadas - Escenario 2. En verde oscuro, trayectoria generada por el algoritmo A*-M1. En verde claro, trayectoria generada por el algoritmo A*-SIREN (M=30.000, w1). En rojo, trayectoria generada por el algoritmo A*-SIREN (M=70.000, w4)..

su única diferencia es que pasa mucho más cerca de la esquina (al no considerar obstáculos).

Comenzando por la trayectoria en verde claro, asociada al algoritmo A*-SIREN con $M = 30.000$ y en el estado de entrenamiento de la red $w1$, se puede observar no solo que difiere mucho respecto a la dada por el algoritmo A*-M1, sino que además tiene en su parte final un tramo bastante poco coherente, donde el dron debería ir en zig-zag, lo cual se debe a una red mal entrenada y con mucho ruido en la salida.

Algo interesante de esta trayectoria es que el movimiento errático solo se da al final de la trayectoria, es decir, solo se da en el entorno que el dron no puede ver. Esto se debe a que, al no conocer los obstáculos presentes tras doblar la esquina, las muestras del SDF Global tendran como valor considerado "real" una medida errónea, que inevitablemente dará lugar a un entrenamiento pobre y, en última instancia, a una mala planificación.

La mejor trayectoria generada por el algoritmo A*-SIREN, sin embargo, sí se parece mucho a la trayectoria óptima. Esto ocurre debido a que, una vez el dron ha podido ver todos los obstáculos que componen el terreno, las muestras obtenidas a través de los dos métodos de muestreo comentados en el Capítulo 3 son precisas y, por tanto, provocan un entrenamiento correcto de la red neuronal y una predicción bastante bueno por parte de esta.

En conclusión, en este segundo experimento se ha demostrado que el sistema tiene un comportamiento muy bueno siempre y cuando conozca los obstáculos presentes en el entorno inmediato por el cual debe transcurrir la trayectoria.

6 Conclusiones

En este proyecto se ha repasado completamente el proceso de integración de un ESDF generado a través de una red neuronal en un planificador de trayectorias para drones aéreos y su posterior validación.

Como se ha podido comprobar a lo largo de los capítulos, especialmente en el referido a simulaciones, la representación del espacio con un ESDF generado con redes neuronales ha ofrecido unos resultados prometedores, lo que demuestra que merece la pena una investigación más exhaustiva al respecto. La habilidad que las redes neuronales han demostrado tener para la representación de espacios tridimensionales abre la puerta no solo a aplicaciones en el ámbito de la robótica, sino también a muchos otros campos.

En lo correspondiente al diseño del sistema para el entrenamiento de la red neuronal, es destacable remarcar que la estructura empleada no es única, si bien muchos de los artículos empleados como referencia bibliográfica en este proyecto utilizan una implementación similar. La aplicación de los ESDF generados por redes neuronales al ámbito de la planificación de trayectorias no ha sido explorada exhaustivamente, y se abre la puerta a nuevas estructuras y formas de implementación que puedan mejorar la actuación del sistema.

Por último, el hecho de utilizar un entorno ampliamente empleado en la investigación como es ROS para montar el sistema al completo facilita enormemente la posibilidad de mejoras futuras y la utilización de lo investigado en proyectos de diversa índole.

6.1 Líneas futuras de desarrollo

A lo largo del trabajo se han desarrollado algunos conceptos que, si bien resultaría interesante ahondar en ellos, quedan fuera del alcance de este trabajo. En esta sección se recogen las líneas principales de trabajo a futuro.

En primer lugar, la gran cantidad de parámetros presentes dentro de cada bloque (tanto en los de generación de muestras para el entrenamiento como en la propia red) hace posible que la optimización completa del sistema presentado se vuelva una tarea extremadamente desafiante. Si bien durante las simulaciones se ha explorado con diferentes valores para algunos parámetros de interés, queda como trabajo a futuro investigar la influencia de variar otros valores en el comportamiento del sistema al completo.

Por otro lado, el sistema presentado es, a todos los efectos, un sistema de planificación global, donde la planificación se realiza de forma completa entre un punto de inicio y un punto de destino. La naturaleza de la representación espacial con redes neuronales, sin embargo, parecen apuntar más a sus posibles aplicaciones en planificación local (u *on-line*). Queda por tanto como trabajo a futuro la implementación del sistema para planificación local.

Otra de las posibles líneas de trabajo a futuro es la implementación de más tipos de planificadores en el sistema. Si bien las simulaciones se han realizado con un A* para que resaltaran más las propiedades de la representación del ESDF por parte de la red neuronal, existen otras múltiples estrategias, muchas de las

cuales explicadas en el Capítulo 1, que pueden ser implementadas como parte del sistema.

Por último, queda pendiente la investigación de posibles aplicaciones del sistema para replanificación con objetos móviles. Dada la naturaleza de la representación del ESDF con redes neuronales, sus propiedades para adaptarse a un entorno cambiante (con obstáculos que cambien con el tiempo) parecen hacer que sea muy útil en este ámbito. Es por ello que se deja como trabajo futuro investigar el comportamiento de este tipo de sistemas en entornos no estáticos.

Índice de Figuras

Bibliografía

[1] *Gazebo*, gazebosim.org.

[2] *Pytorch*, pytorch.org.

[3] *Ros: Robot operating system*, ros.org.

[4] *Sick sensor intelligence, https://www.sick.com/br/es/productos-y-soluciones/productos/sensores-lidar-y-de-radar/sensores-lidar/c/g575802*.

[5] Jose M. Peña Ana de Castro Megías, Yeyin Shi and Joe Maja, *Uavs for vegetation monitoring*, Basel, Switzerland MDPI - Multidisciplinary Digital Publishing Institute, 2021.

[6] John Buckley, *Air power in the age of total war*, Taylor and Francis, 2006.

[7] T. Chan and Wei Zhu, *Level set based shape prior segmentation*, 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 2, 2005, pp. 1164–1170 vol. 2.

[8] Chriscummingshrg, *Signed distance fields part 2: Solid geometry*, 07 2018.

[9] Jose A. Cobano, Rafael Rey, L. Merino, and F. Caballero, *Fast cost-aware lazy-theta over euclidean distance functions for 3d planning of aerial robots in building-like environments*, 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2022, pp. 13486–13493.

[10] Mihir Dharmadhikari, Tung Dang, Lukas Solanka, Johannes Loje, Huan Nguyen, Nikhil Khedekar, and Kostas Alexis, *Motion primitives-based path planning for fast and agile exploration using aerial robots*, 2020 IEEE International Conference on Robotics and Automation (ICRA), 2020, pp. 179–185.

[11] Luxin Han, Fei Gao, Boyu Zhou, and Shaojie Shen, *FIESTA: fast incremental euclidean distance fields for online motion planning of aerial robots*, CoRR **abs/1903.02144** (2019).

[12] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael, *A formal basis for the heuristic determination of minimum cost paths*, IEEE Transactions on Systems Science and Cybernetics **4** (1968), no. 2, 100–107.

[13] Chanyoungju Jeongeun Kim, Seungwon Kim and Hyoungil Son, *Unmanned aerial vehicles in agriculture: A review of perspective of platform, control and applications*, IEEE Access PP(99):1-1 (2019).

[14] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars, *Probabilistic roadmaps for path planning in high-dimensional configuration spaces*, IEEE Transactions on Robotics and Automation **12** (1996), no. 4, 566–580.

[15] O. Khatib, *Real-time obstacle avoidance for manipulators and mobile robots*, Proceedings. 1985 IEEE International Conference on Robotics and Automation, vol. 2, 1985, pp. 500–505.

[16] S.M. LaValle and J.J. Kuffner, *Randomized kinodynamic planning*, Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C), vol. 1, 1999, pp. 473–479 vol.1.

[17] Phuc Tran Huu Le, Nguyen Tam Nguyen Truong, Minsu Kim, Wonshoup So, and Jae Hak Jung, *Applying theta* in modern game*, J. Comput. **13** (2018), 527–536.

[18] Bharat Lohani and Suddhasheel Ghosh, *Airborne lidar technology: A review of data collection and processing systems*, Proceedings of the National Academy of Sciences, India Section A: Physical Sciences **87** (2017).

[19] Chunbo Luo, Wang Miao, Hanif Ullah, Sally McClean, Gerard Parr, and Geyong Min, *Unmanned aerial vehicles for disaster management*, pp. 83–107, 08 2019.

[20] Sina Sharif Mansouri, Christoforos Kanellakis, Dariusz Kominiak, and George Nikolakopoulos, *Deploying mavs for autonomous navigation in dark underground mine environments*, Robotics and Autonomous Systems **126** (2020), no. 18:6223.

[21] Simon Martinez-Rozas, Rafael Rey, David Alejo, Domingo Acedo, Jose Antonio Cobano, Alejandro Rodriguez-Ramos, Pascual Campoy, Luis Merino, and Fernando Caballero, *Skyeye team at mbzirc 2020: A team of aerial and ground robots for gps-denied autonomous fire extinguishing in an urban building scenario*, Field Robotics (2021), In Press.

[22] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng, *Nerf: representing scenes as neural radiance fields for view synthesis*, Commun. ACM **65** (2021), no. 1, 99–106.

[23] H. Oleynikova, M. Burri, Z. Taylor, J. Nieto, R. Siegwart, and E. Galceran, *Continuous-time trajectory optimization for online uav replanning.*, IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2016, p. 5332–5339.

[24] Joseph Ortiz, Alexander Clegg, Jing Dong, Edgar Sucar, David Novotny, Michael Zollhoefer, and Mustafa Mukadam, *isdf: Real-time neural signed distance fields for robot perception*, Robotics: Science and Systems, 2022.

[25] Yue Pan, Yves Kompis, Luca Bartolomei, Ruben Mascaro, Cyrill Stachniss, and Margarita Chli, *Voxfield: Non-projective signed distance fields for online planning and 3d reconstruction*, 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2022, pp. 5331–5338.

[26] Pieter Pauwels, Rens Koning, Bob Hendrikx, and Elena Torta, *Live semantic data from building digital twins for robot navigation: Overview of data transfer methods*, Advanced Engineering Informatics **56** (2023), 101959.

[27] Firas Raheem and Mustafa Badr, *Development of modified path planning algorithm using artificial potential field (apf) based on pso for factors optimization*, American Scientific Research Journal for Engineering, Technology, and Sciences **37** (2017), 316–328.

[28] Nathan Ranno and Dong Si, *Neural representations of cryo-em maps and a graph-based interpretation*, 04 2021.

[29] Norvig P. Russell S., *Artificial intelligence a modern approach*, Boston: Pearson, 2018.

[30] Yasmina Bestaoui Sebbane, *Smart autonomous aircraft: Flight control and planning for uav*, Boca Raton ; London : CRC Press, 2016.

[31] R. Sheh, S. Schwertfeger, and A. Visser, *16 years of robocup rescue*, KI-Künstliche Intelligenz **30** (2016), 267–277.

[32] Vincent Sitzmann, Julien N. P. Martel, Alexander W. Bergman, David B. Lindell, and Gordon Wetzstein, *Implicit neural representations with periodic activation functions*, 2020.

[33] A. Stentz, *Optimal and efficient path planning for partially-known environments*, Proceedings of the 1994 IEEE International Conference on Robotics and Automation, 1994, pp. 3310–3317 vol.4.

[34] Vasileios Vasilopoulos, Suveer Garg, Jinwook Huh, Bhoram Lee, and Volkan Isler, *HIO-SDF: Hierarchical Incremental Online Signed Distance Fields*, IEEE International Conference on Robotics and Automation, 2024.

[35] Jiankun Wang, Wenzheng Chi, Chenming Li, Chaoqun Wang, and Max Q.-H. Meng, *Neural rrt*: Learning-based optimal path planning*, IEEE Transactions on Automation Science and Engineering **17** (2020), no. 4, 1748–1758.

[36] Keenan Wyrobek, *The origin story of ros, the linux of robotics*, *https://spectrum.ieee.org/the-origin-story-of-ros-the-linux-of-robotics*, 10 2017.

[37] Junho Yeom, Youkyung Han, Anjin Chang, and Jinha Jung, *Hurricane building damage assessment using post-disaster uav data*, 07 2019, pp. 9867–9870.

[38] Ekram Khan Mohammed A. Qadeer Zainab Zaheer, Atiya Usmani, *Aerial surveillance system using uav*, 2016 Thirteenth International Conference on Wireless and Optical Communications Networks (WOCN) (2016).