Universitat Oberta de Catalunya (UOC)

Máster Universitario en Ciencia de Datos (*Data Science*)

TRABAJO FINAL DE MÁSTER

Área: 2

Navegación autónoma de vehículos aéreos no tripulados mediante técnicas de aprendizaje por refuerzo profundo

---

Autor: Francisco José Núñez Sánchez-Agustino

Tutor: Luis Esteve Elfau

Profesor: Jordi Casas Roma

Barcelona, 25 de julio de 2023

Créditos/Copyright

FICHA DEL TRABAJO FINAL

Título del trabajo:	Navegación autónoma de vehículos aéreos no tripulados mediante técnicas de aprendizaje por refuerzo profundo
Nombre del autor:	Francisco José Núñez Sánchez-Agustino
Nombre del colaborador/a docente:	Luis Esteve Elfau
Nombre del PRA:	Jordi Casas Roma
Fecha de entrega (mm/aaaa):	06/2023
Titulación o programa:	Máster Universitario en Ciencia de Datos
Área del Trabajo Final:	Área 2
Idioma del trabajo:	Español
Palabras clave	Aprendizaje por refuerzo profundo, visión artificial, navegación autónoma

Dedicatoria/Cita

A mi familia, por su apoyo y paciencia durante estos meses.

Agradecimientos

A Luis Esteve Elfau, cuya inestimable ayuda y orientación ha sido fundamental para desarrollar este trabajo.

Resumen

El presente trabajo es un estudio sobre la aplicación práctica de técnicas de aprendizaje por refuerzo profundo (DRL, *Deep Reinforcement Learning*) en el desarrollo de un sistema de navegación autónoma para vehículos aéreos no tripulados.

Este tipo de aeronaves - también conocido como UAV (*Unmaned Aerial Vehicle*) o simplemente dron - se utilizan en una amplia variedad de tareas como la vigilancia, búsqueda, rescate, topografía o la investigación científica, entre otras. En todas ellas la navegación autónoma es una característica más que deseable por la seguridad y eficiencia que puede aportar a la operatividad de estos aparatos.

Sin embargo, prescindir de la intervención humana presenta grandes desafíos como detectar y evitar obstáculos para garantizar la integridad de la aeronave y la seguridad de personas, animales, plantas u objetos presentes en el entorno.

En este sentido, el aprendizaje por refuerzo profundo se ha convertido en una alternativa muy prometedora para llevar a cabo esta compleja tarea, permitiendo al dron aprender a través de su experiencia e interacciones con distintos entornos, tanto simulados como reales.

Este documento pretende demostrar la viabilidad de un sistema de navegación que combine algoritmos de aprendizaje por refuerzo profundo con técnicas de visión artificial, para permitir a un dron sencillo de bajo coste, equipado únicamente con una cámara de vídeo frontal, navegar de manera autónoma el mayor tiempo posible.

Palabras clave: aprendizaje por refuerzo profundo, visión artificial, navegación autónoma.

Abstract

This work is a study on the practical application of Deep Reinforcement Learning (DRL) techniques in the development of an autonomous navigation system for Unmanned Aerial Vehicles (UAVs), also known as "drones".

These types of aircraft are used for a wide variety of tasks such as surveillance, search and rescue, topography, or scientific research, among others. In all of these tasks, autonomous navigation is a desirable feature due to the safety and efficiency it can provide to the operation of these devices.

However, eliminating human intervention presents significant challenges, such as detecting and avoiding obstacles to ensure the integrity of the aircraft and the safety of people, animals, plants or objects in the environment.

In this sense, deep reinforcement learning has become a very promising alternative to carry out this complex task, allowing the drone to learn through its experience and interactions with different environments, both simulated and real.

This document aims to demonstrate the viability of a navigation system that combines deep reinforcement learning algorithms with computer vision techniques, to enable a low-cost, simple drone equipped only with a front-facing video camera to navigate autonomously for as long as possible.

Keywords: deep reinforcement learning, computer vision, autonomous navigation.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Descripción de la propuesta

El tema elegido para este trabajo es la aplicación de un algoritmo de aprendizaje por refuerzo profundo (DRL, *Deep Reinforcement Learning*) combinado con técnicas de visión artificial, con el objetivo de desarrollar un sistema de navegación que permita a un vehículo aéreo no tripulado equipado con una cámara, volar de manera autónoma sin colisionar con ningún objeto u obstáculo de su entorno.

1.2. Justificación de su interés y relevancia

En los últimos años el uso de vehículos aéreos no tripulados ha experimentado un gran auge en sectores como la agricultura, industria, energía o servicios de emergencia. Esto se debe a las numerosas ventajas que ofrecen, como la capacidad para acceder a áreas inaccesibles o potencialmente peligrosas para las personas. Además, estos aparatos son capaces de operar con gran precisión y a bajo costo, en comparación con los métodos tradicionales.

Actualmente, su uso es habitual en tareas como la vigilancia, búsqueda y rescate, inspección de infraestructuras o investigación científica. En todas estas aplicaciones, es fundamental que estos vehículos sean capaces de operar con mínima o nula intervención humana para lograr sus objetivos de manera segura y eficiente.

El aprendizaje por refuerzo profundo es un campo relativamente nuevo de la inteligencia artificial que ha despertado un gran interés en el ámbito de los vehículos autónomos [1] [2] [3]. La creciente actividad investigadora evidencia el potencial de esta técnica para desarrollar sistemas más eficientes, que aprenden de su propia interacción con el entorno. Una capacidad que permite dotar a los drones de la autonomía necesaria para operar de manera segura en diferentes situaciones.

1.3. Motivación personal

La elección de la temática de este TFM está motivada principalmente por la oportunidad que representa para aplicar los conocimientos adquiridos en la asignatura de Aprendizaje por Refuerzo del Máster en Ciencia de Datos de la UOC.

Además, el desarrollo de este trabajo es un desafío personal, ya que el tema elegido forma parte de una área de investigación en pleno crecimiento, en la que las técnicas de aprendizaje por refuerzo profundo todavía están comenzando a demostrar su verdadero potencial. Como resultado, gran parte del trabajo necesario para llevar a cabo este TFM estará relacionado con el estudio y análisis de publicaciones técnicas para poder desarrollar un modelo de navegación autónoma que sea razonablemente efectivo.

1.4. Objetivos

Como se ha mencionado previamente, el objetivo de este trabajo es el estudio y aplicación de técnicas de aprendizaje por refuerzo para desarrollar un modelo que permita a un dron navegar de manera autónoma. Específicamente, se busca lograr que pueda volar en espacios interiores evitando colisiones, utilizando la información capturada por su cámara frontal.

Para lograr este objetivo el modelo implementará un sistema de aprendizaje por refuerzo profundo basado en una red neuronal convolucional (*Convolutional Neural Network*, CNN), que será entrenada utilizando las imágenes obtenidas de la cámara del dron. El sistema utilizará esta información para decidir la siguiente acción a tomar, evitando obstáculos y buscando permanecer en vuelo el mayor tiempo posible.

El proceso de entrenamiento se realizará en un entorno virtual para asegurar la integridad del dispositivo y evitar daños innecesarios. Una vez que se logre un comportamiento aceptable, el modelo será sometido a diferentes pruebas en un entorno real. Además, en este trabajo se persiguen también los siguientes objetivos:

- Evaluar la eficacia del modelo mediante la realización de pruebas en diferentes espacios interiores y situaciones.
- Analizar los resultados e identificar posibles mejoras y ajustes en el modelo para hacerlo más preciso y efectivo.
- Comparar diferentes algoritmos de aprendizaje por refuerzo profundo y determinar cuál es el más adecuado para lograr el objetivo principal propuesto.

1.5. Metodología de investigación y trabajo

Para lograr los objetivos propuestos es necesario revisar la literatura existente en el campo del aprendizaje por refuerzo profundo y la navegación autónoma de drones. Para llevar a cabo este proceso con rigurosidad y objetividad, se seguirá la estrategia de investigación descrita por Briony J. Oates [4] en su libro *"Researching Information Systems and Computing"*, específicamente en el capítulo 6 dedicado a la revisión de la literatura.

La metodología propuesta por Oates consta de varias etapas, incluyendo la identificación de la necesidad de revisión, la búsqueda y obtención de los documentos relevantes, la evaluación crítica y la síntesis de los resultados. En este trabajo, se utilizarán principalmente artículos científicos accesibles a través de bases de datos como IEEE Xplore [5], ArXiv.org [6] o Google Scholar [7] para identificar las publicaciones más relevantes aplicables al alcance propuesto.

En este sentido, dado que el desarrollo de un sistema de navegación autónomo puede ser muy amplio y complejo, este trabajo se centrará en un caso de uso de un entorno controlado: la navegación autónoma en espacios interiores mediante la información capturada por la cámara frontal del dron.

Para garantizar un proceso ordenado y sistemático, se aplicará una metodología de desarrollo en cascada. Además, se usará Python como lenguaje de programación y se empleará la librería Pytorch [8] para diseñar y entrenar la red neuronal. Las técnicas de generación de datos se basarán inicialmente en simulaciones de vuelo con la librería AirSim [9] y el motor gráfico Unreal Engine [10], en una etapa posterior se recopilarán datos de vuelo en un entorno real para validar el modelo entrenado.

1.6. Competencia de compromiso ético y global (CCEG) y Objetivos de Desarrollo Sostenible (ODS)

El desarrollo de un sistema de navegación autónomo como el planteado en este trabajo debe tener en cuenta los posibles impactos negativos en la dimensión de comportamiento ético y responsabilidad social de la CCEG, como la violación de la privacidad de las personas o los daños a seres vivos u objetos durante la operación autónoma del dron.

En relación con los ODS, un sistema como el propuesto puede contribuir al logro de varios objetivos. Por ejemplo, la navegación autónoma puede permitir la automatización de tareas aumentando la eficiencia y productividad en diversas aplicaciones, lo que sería relevante para el ODS 9 (Industria, innovación e infraestructura) . Además, la optimización de la ruta y tiempo de vuelo puede reducir su huella de carbono, ayudando al logro del ODS 13 (Acción por el clima).

1.7. Planificación del trabajo

A continuación, se muestra la planificación del TFM mediante una tabla de tareas a realizar, la lista de entregables que se generarán durante el desarrollo de este trabajo, así como un diagrama de Gantt que ilustra la planificación temporal de dichas tareas y entregables:

Tarea	Duración	Inicio	Fin
Plan de trabajo	12 días	01-03-2023	12-03-2023
Selección del tema	5 días	01-03-2023	05-03-2023
Elaboración de la propuesta	4 días	06-03-2023	09-03-2023
Elaboración del plan de trabajo	3 días	10-03-2023	12-03-2023
Preparación	15 días	13-03-2023	27-03-2023
Análisis e investigación del estado del arte	3 días	13-03-2023	15-03-2023
Revisión de la bibliografía seleccionada	3 días	16-03-2023	18-03-2023
Análisis y selección de algoritmos y herramientas	3 días	19-03-2023	21-03-2023
Identificación de metodología y técnicas de generación de datos	3 días	22-03-2023	24-03-2023
Ampliación de conocimientos sobre características operativas del dron	2 días	25-03-2023	26-03-2023
Ejecución	80 días	27-03-2023	14-06-2023
Selección, análisis y configuración del entorno virtual	10 días	27-03-2023	05-04-2023
Diseño y codificación del algoritmo para el entorno virtual	15 días	06-04-2023	20-04-2023
Entrenamiento y optimización del modelo en el entorno virtual	20 días	21-04-2023	10-05-2023
Selección y análisis del entorno real	15 días	11-05-2023	25-05-2023
Adaptación del modelo al entorno real	5 días	26-05-2023	30-05-2023
Preparación y desarrollo de pruebas en el entorno real	15 días	31-05-2023	14-06-2023
Análisis de resultados y conclusiones	10 días	15-06-2023	24-06-2023
Preparación de la presentación y defensa	7 días	26-06-2023	02-07-2023
Redacción de la memoria	117 días	01-03-2023	25-06-2023

Cuadro 1.1: *Lista de las tareas a realizar en el TFM.*

Entregable	Descripción	Fecha
PEC 1	Descripción de la temática del trabajo, justificación de su relevancia, definición de objetivos y plan de trabajo	12-03-2023
PEC 2	Descripción del estado del arte, selección de bibliografía e identificación de la metodología y técnicas de generación de datos a utilizar	26-03-2023
PEC 3	Informes sobre el desarrollo del diseño e implementación del trabajo y resultados obtenidos en los experimentos	28-05-2023
PEC 4.1	Borrador de la memoria del TFM para su revisión por parte del tutor	11-06-2023
PEC 4.2	Entrega final de la memoria completa con sus resultados y conclusiones	25-06-2023

Cuadro 1.2: *Lista de entregables del TFM.*

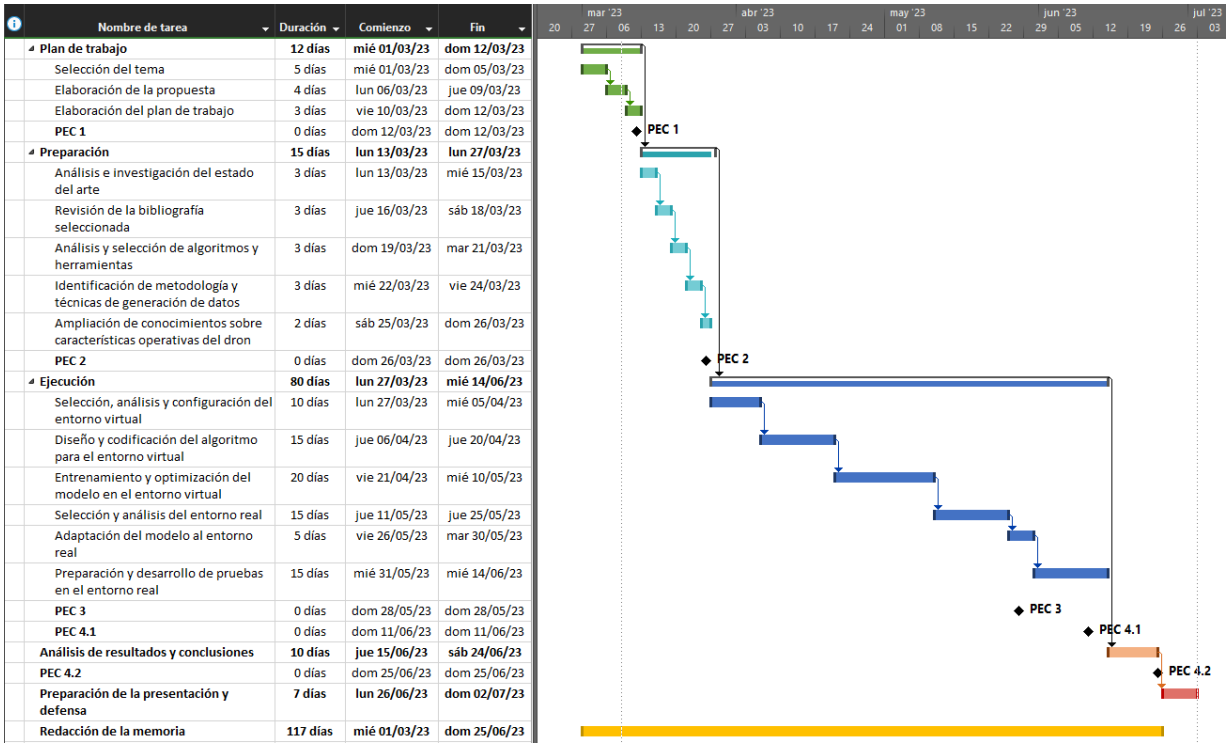

Figura 1.1: *Diagrama de Gantt del TFM*

El siguiente capítulo examina la aplicación del aprendizaje por refuerzo en la navegación autónoma, teniendo en cuenta el estado del arte en este campo y las particularidades de esta rama del aprendizaje automático. A continuación, se presentará el desarrollo del trabajo en capítulos posteriores, los cuales conducirán a sus conclusiones y reflexiones finales.

Capítulo 2

Estudio del estado del arte

2.1. Aprendizaje por refuerzo

El aprendizaje por refuerzo (*Reinforcement Learning*, RL) es una rama del aprendizaje automático que emula el proceso de aprendizaje por interacción. Este mecanismo, observable en la naturaleza, emplea una estrategia de prueba y error que permite tanto a animales como a humanos alcanzar objetivos y obtener recompensas (como alimento o calor) del entorno mediante la realización de secuencias específicas de acciones.

En esta forma de aprendizaje activo, el sujeto o "agente" interactúa con el entorno, que abarca todo lo externo a él. Éste recibe las acciones del agente y responde proporcionando información sobre su estado, que puede experimentar cambios o no en respuesta a dichas interacciones.

En RL, el problema a abordar se define mediante la función de recompensa, el entorno y los estados. Resolver dicho problema requiere diseñar un agente capaz de tomar decisiones acertadas con el objetivo de maximizar la recompensa acumulada a lo largo del tiempo, aprendiendo mediante la interacción con el entorno.

Este aprendizaje se basa en un ciclo interactivo e iterativo que se desarrolla de manera continua: el agente decide ejecutar una acción determinada y, en función de ésta, el entorno proporciona una observación sobre su estado. A medida que esta dinámica se desarrolla, se genera una secuencia de acciones, estados y recompensas que conforma la experiencia del agente. A partir de esta información acumulada, el agente puede lograr un aprendizaje progresivo a lo largo del tiempo.

El esquema de la Figura 2.1 muestra cómo, en cada instante t, el agente recibe del entorno el estado S_t junto con la recompensa R_t. Tras recibir ambas, toma la decisión de ejecutar la acción A_t. Por su parte, el entorno recibe esta acción A_t y responde emitiendo el siguiente estado S_{t+1} y la recompensa R_{t+1}.

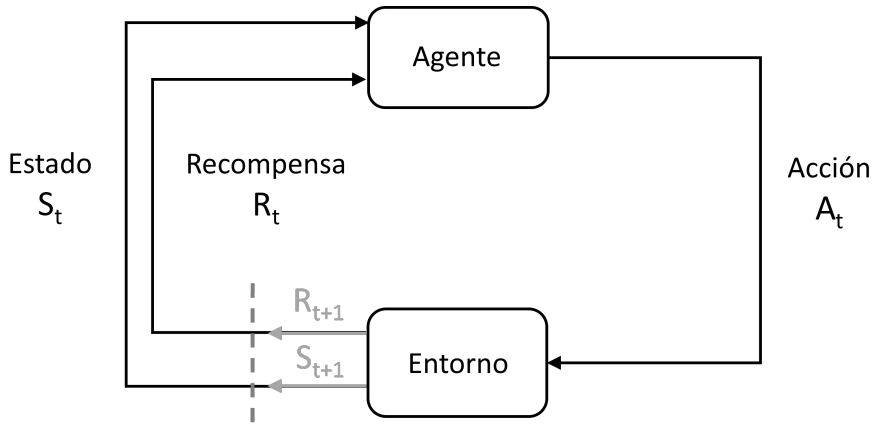

Figura 2.1: *Interacción agente-entorno*

Fuente: *Çetin et al. [11]*

2.1.1. Procesos de decisión de Markov

Los procesos de decisión de Markov (*Markov Decision Process*, MDP) proporcionan el marco matemático adecuado para abordar los problemas de RL, ya que permiten modelar la toma de decisiones en entornos inciertos con dinámicas aleatorias. En estos entornos, el agente no dispone de información completa sobre el estado actual ni su evolución futura y tanto las transiciones entre estados como las recompensas asociadas pueden estar sujetas a probabilidades en lugar de ser deterministas.

En este marco, hay dos conceptos esenciales para el aprendizaje por refuerzo:

- La política, que representa la estrategia que determina cómo un agente debe actuar en cada estado, es decir, qué acciones debe tomar.
- La función de valor, que estima el valor esperado de un estado - función $v(s)$ - o de llevar a cabo una acción en un estado concreto - función $q(s, a)$ - teniendo en cuenta las recompensas futuras y siguiendo una política determinada. Éstas son fundamentales para evaluar las políticas, permitiendo al agente actualizar y mejorar su estrategia a medida que interactúa con el entorno.

Resolver un problema de RL en el contexto de un MDP implica encontrar una política óptima que maximice la recompensa acumulada. En este sentido, en RL existen dos tipos de métodos: *on-policy* y *off-policy*. Los primeros evalúan y mejoran la política que está siendo utilizada para tomar decisiones, mientras que los *off-policy* hacen esto con una política diferente a la que se utiliza para obtener información del entorno.

Independientemente del enfoque seguido, para encontrar la política óptima se utilizan las ecuaciones de optimización de Bellman [12], las cuales permiten actualizar iterativamente las estimaciones de las funciones de valor hasta que convergen en una solución óptima.

Las ecuaciones para las funciones de valor de estado y acción óptimas son las siguientes:

$$v_*(s) = max_a q_*(s,a) = \max_a \sum_{\substack{\forall r \in R \\ \forall s' \in S}} p(r, s'|s, a)[r + \gamma v_*(s')]$$

$$q_*(s,a) = \sum_{\substack{\forall r \in R \\ \forall s' \in S}} p(r, s'|s, a) + \gamma \max_{a'} q_*(s', a')]$$

En estas fórmulas, $p(r, s'|s, a)$ representa la probabilidad de transición del estado s a s' con recompensa r al tomar la acción a. Por otro lado, $v_*(s')$ y $q_*(s', a')$ son las funciones de valor de estado y de acción óptimas, ponderadas por el factor de descuento $\gamma \in [0, 1)$. Éste determina la importancia relativa de las recompensas futuras en la estimación del valor de un estado o una acción: valores altos priorizan las obtenidas a largo plazo, mientras que valores bajos dan mayor peso a las que se consiguen a corto plazo.

Las ecuaciones de Bellman - que son recursivas y no lineales debido a las operaciones de maximización involucradas - se aplican tanto a algoritmos que utilizan un modelo explícito del entorno, como a algoritmos sin modelo, que aprenden directamente a través de la interacción con el entorno. Estas ecuaciones permiten al agente aproximar los valores óptimos de las funciones de valor, lo cual facilita la toma de decisiones informadas para maximizar la recompensa acumulada.

La manera en la que se aproxima la solución de estas ecuaciones caracteriza a los distintos algoritmos RL, tanto aquellos que utilizan un modelo como los que no. Por ejemplo, algoritmos "*model-free*" como Montecarlo [13], Q-Learning [14], SARSA [15], REINFORCE [16] o los métodos Actor-crítico [13], emplean distintas estrategias, algunas basadas en funciones de valor otras en optimización directa de la política, e incluso en una combinación de ambos enfoques. Esta diversidad les confiere características y propiedades únicas lo que permite seleccionar el más adecuado según el problema de RL a resolver.

2.1.2. Aprendizaje por refuerzo profundo

En problemas con espacios de acciones o estados muy grandes y desconocidos de antemano, como el control de un robot, tareas de planificación y optimización o ciertos videojuegos, las técnicas tradicionales de RL pueden resultar inadecuadas, especialmente aquellas que dependen de tablas para representar de manera discreta el espacio de estados y acciones, pues éstos se

pueden volver inmanejables en términos de almacenamiento y capacidad de cómputo.

En estos casos, se requieren soluciones que permitan aproximar la función de valor o la política generalizando a partir de ejemplos. Cuando esta función de aproximación es una red neuronal, nos referimos a este tipo de algoritmos como aprendizaje por refuerzo profundo o DRL (*Deep Reinforcement Learning*)

La figura 2.2 muestra un agente empleando una red neuronal profunda o DNN (*Deep Neural Network*) para aprender la política que seguirá en el entorno. El proceso de aprendizaje de la red implica ajustar sus parámetros o pesos de forma iterativa, buscando que la salida se acerque al valor deseado. Este ajuste se realiza mediante algoritmos de optimización de gradiente iterativos, cuyo objetivo es minimizar la función de pérdida. En el contexto de RL, esta función mide el error entre la salida de la red neuronal y el valor deseado, lo que se traduce en el error en la predicción del valor de estado o acción, o la diferencia entre la política actual y la óptima.

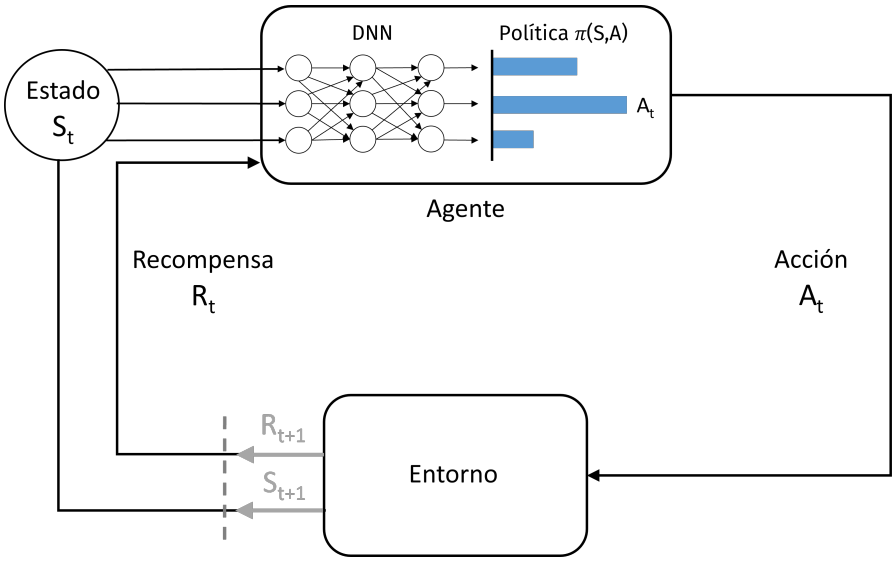

Figura 2.2: *Interacción agente-entorno en DRL*

Fuente: *Pozza et al. [17]*

El uso de DNN como estimadores universales no lineales, ha sido fundamental en los avances recientes en RL. Entre los algoritmos DRL más destacados se encuentran DQN (*Deep Q-Network*) [18], PPO (*Proximal Policy Optimization*) [19], A2C (*Advantage Actor-Critic*) [20], DDPG (*Deep Deterministic Policy Gradient*) [21], entre otros. Éstos han demostrado un rendimiento sólido en diferentes tareas y han sido fundamentales en el desarrollo y el estado del arte en este campo.

2.2. Navegación autónoma

La capacidad de los drones para navegar de forma autónoma, es decir, sin requerir intervención humana, todavía presenta ciertas limitaciones en cuanto a gestionar situaciones imprevistas o extremadamente complejas. A pesar de esto, posee un gran potencial para mejorar notablemente su eficiencia, seguridad y desempeño en la realización de tareas complicadas o repetitivas.

Esta habilidad se basa en un software impulsado por Inteligencia Artificial (IA) instalado en la placa controladora de vuelo de dichas aeronaves. Este software permite a los drones despegar, aterrizar y realizar inspecciones aéreas, entre otras tareas, de forma autónoma. Además, les proporciona la capacidad de tomar decisiones que aseguren no solo su propia integridad y seguridad, sino también la de personas, animales, plantas y objetos en su entorno.

Para implementar un sistema de navegación autónoma en un dron, es necesario contar con diversos sensores a bordo, como cámaras, acelerómetros y giroscopios, entre otros. Éstos generan datos que son analizados y procesados en tiempo real mediante algoritmos de IA, para permitir al dron navegar y realizar tareas específicas de manera autónoma. La mejora y optimización del tratamiento y análisis de los datos de estos sensores es una área de investigación en constante desarrollo.

En cuanto a las funciones necesarias para lograr el vuelo autónomo, las más relevantes se pueden agrupar en cuatro tipos de tareas principales: control de la aeronave, detectar y evitar obstáculos, planificación de rutas y vuelo coordinado de múltiples drones.

2.2.1. Control de la aeronave

El control del dron implica varias tareas, como el despegue, aterrizaje, estabilización y control de la altitud para garantizar un funcionamiento seguro mientras se encuentra en el aire. Para lograr esto, es necesario ajustar los componentes de su mecánica de vuelo, incluyendo los rotores, hélices o sistemas de estabilización, en caso de que disponga de ellos. La eficiencia en el control del dron desempeña un papel crucial en la optimización del consumo de energía y en la prolongación de la autonomía de vuelo.

El cuadricóptero, el dron más común, cuenta con cuatro rotores dispuestos en dos diagonales, girando en la misma dirección pero en sentido contrario a los de la diagonal opuesta, tal y como se puede apreciar en la Figura 2.3.

El desplazamiento del dron se controla modificando el empuje de cada rotor, lo que produce cambios en su orientación y posición. Esto da lugar a cuatro movimientos principales:

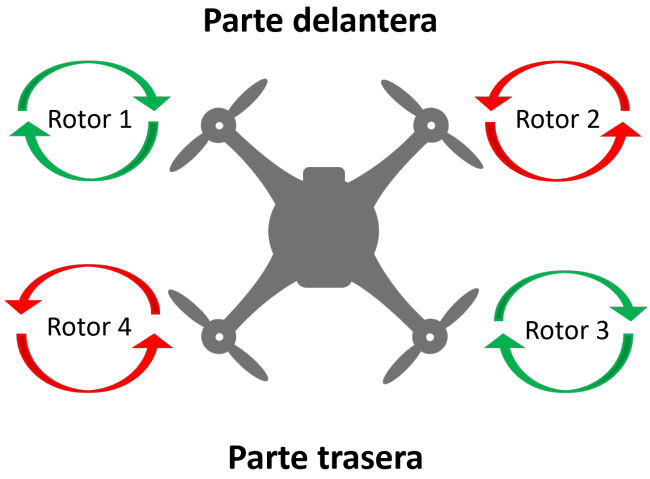

Figura 2.3: *Disposición de los rotores de un cuadricóptero*

Fuente: *elaboración propia*

- Cabeceo (*pitch*): movimiento hacia adelante y hacia atrás.
- Balanceo (*roll*): movimiento inclinándose hacia los lados.
- Guiñada (*yaw)*: rotación alrededor de su eje central vertical.
- Aceleración (*throttle*): cambio de velocidad en cualquier dirección.

La Figura 2.4 muestra las maniobras que se pueden realizar con los movimientos mencionados anteriormente. El ascenso se logra aumentando simultáneamente la velocidad de los cuatro rotores para generar un empuje mayor que su peso. Cuando el empuje es igual al peso se puede mantener una posición estacionaria. Para descender, se deben disminuir la velocidad de los rotores.

Para moverse en diferentes direcciones, el cuadricóptero debe inclinarse en la dirección deseada. Para lograr esto, se aumenta la velocidad de rotación de dos rotores "vecinos", tal y como muestra la ilustración, lo que hace que la aeronave se incline y se desplace en la dirección opuesta. De esta manera, se puede mover hacia adelante, atrás o a los lados.

Para girar en su propio eje vertical, se debe incrementar la velocidad de rotación de dos rotores diagonales opuestos, lo que provoca que el cuadricóptero gire en la dirección opuesta a la de estos rotores. De esta manera, se puede cambiar la orientación de la aeronave sin cambiar su posición en el espacio.

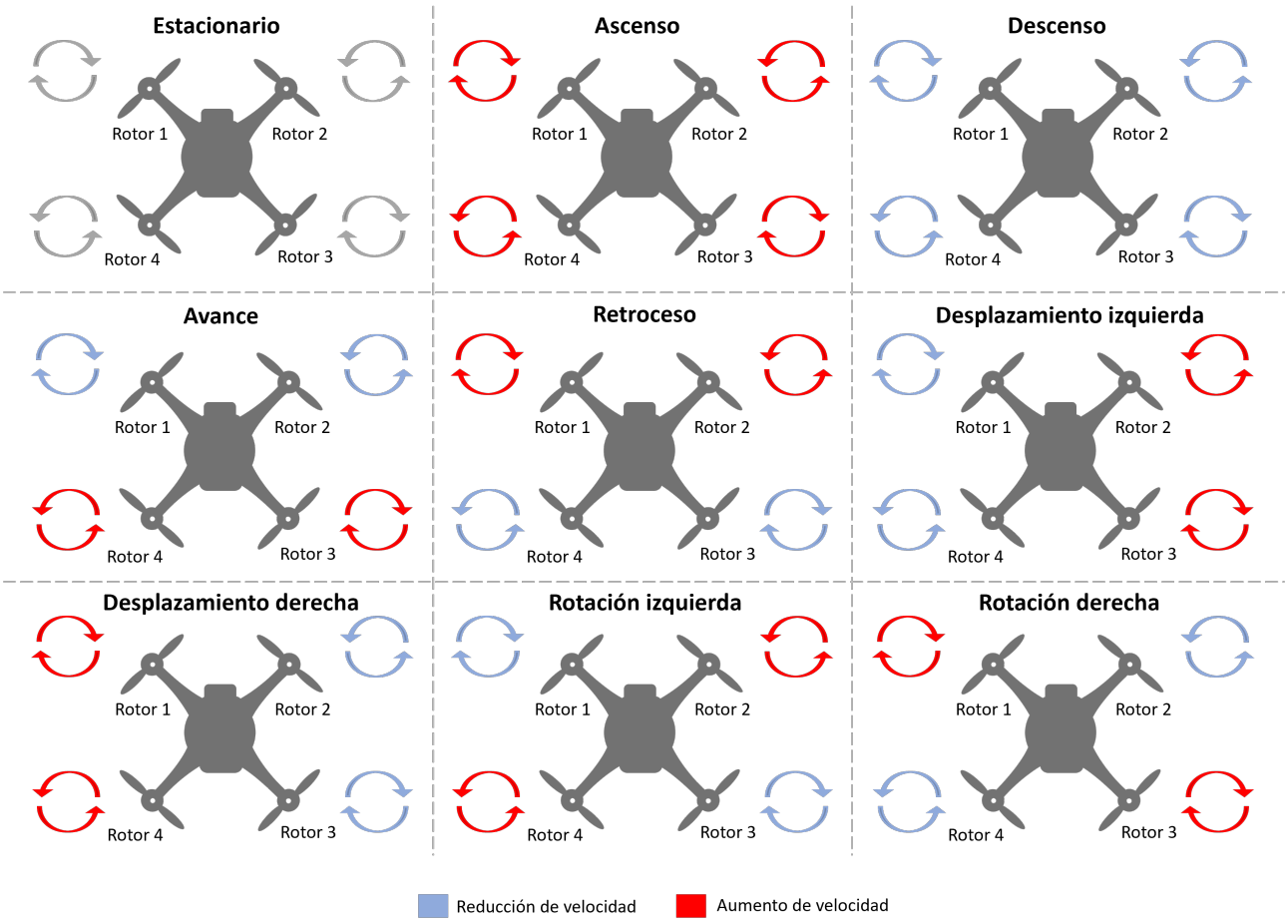

Figura 2.4: *Maniobras de un cuadricóptero*

Fuente: *elaboración propia*

2.2.2. Detectar y evitar obstáculos

La detección y evasión de obstáculos es una tarea crítica en la navegación autónoma de drones, particularmente en entornos complejos con obstáculos fijos y móviles. Para garantizar un vuelo seguro, se utilizan diferentes dispositivos y técnicas que permiten determinar la distancia a la que se encuentran los objetos en el entorno para evitar colisiones.

Entre los dispositivos más comunes se incluyen cámaras y sensores de distancia, que proporcionan información visual y de proximidad sobre el entorno. Un dispositivo cada vez más habitual es el escáner LiDAR (*Light Detection and Ranging*), que emite ráfagas de luz láser y mide el tiempo que tardan en reflejarse. Este proceso genera modelos tridimensionales de los objetos y determina su posición en el entorno, ofreciendo una mayor precisión y detalle en la detección de obstáculos.

La información obtenida a través de estos dispositivos se procesa empleando algoritmos como SLAM (*Simultaneous Localization and Mapping*) [22], que permite construir mapas del

entorno y localizar al dron en el mismo, o distintas variantes de algoritmos DRL. En ambos casos, suelen combinarse con técnicas de visión por computadora.

Independientemente del enfoque utilizado, para detectar, localizar y evitar obstáculos de manera efectiva, es fundamental que el sistema tenga la capacidad de identificar rápidamente los obstáculos y acciones evasivas apropiadas. Esto requiere contar con un hardware adecuado que incluya sensores y componentes mecánicos de calidad, y una placa controladora de vuelo con potencia de procesamiento suficiente.

2.2.3. Planificación de rutas

Para que los drones puedan realizar tareas autónomas, es esencial establecer objetivos concretos, como desplazarse desde un punto de partida hasta un destino específico o mantener una posición sobre una área determinada. Alcanzar estos objetivos requiere la planificación de trayectorias que consideren factores clave, como la distancia, la seguridad de la navegación y la autonomía limitada de estas aeronaves.

Esta planificación puede abordarse mediante diversas técnicas y algoritmos. Algunos enfoques incluyen el empleo de mapas o representaciones del entorno con información sobre el terreno y obstáculos [23], posibilitando incluso la aplicación de algoritmos clásicos como Dijkstra para calcular rutas óptimas en aquellos casos en los que se cuenta con información completa [24]. Por otro lado, hay enfoques que no hacen uso de mapas y se basan principalmente en técnicas de visión por computadora [25] para extraer características del entorno y aprender patrones que permitan llegar al destino de manera eficiente.

En cualquier caso, es importante diferenciar entre rutas óptimas globales y locales. Las globales consideran el entorno completo y todos los obstáculos conocidos para hallar la trayectoria más eficiente desde el inicio hasta el destino. Las locales, en cambio, abordan solo el entorno inmediato y obstáculos cercanos al dron, siendo útiles en situaciones con información incompleta u obstáculos dinámicos e imprevistos.

Al planificar rutas óptimas, se suelen aprovechar dispositivos como el sistema de posicionamiento global (GPS) o la unidad de medición inercial - IMU (*Inertial Measurement Unit*) - que proporciona datos sobre la orientación, aceleración y velocidad angular del dron, lo que permite al sistema de control mantener la estabilidad y ajustar la trayectoria según sea necesario. Ambos dispositivos, GPS e IMU, ofrecen información relevante sobre la ubicación actual del dron y permiten calcular la distancia y el ángulo entre dicha posición y el objetivo.

2.2.4. Vuelo coordinado de múltiples drones

Aunque todavía se encuentra en una etapa temprana de desarrollo, el vuelo coordinado de múltiples drones, conocido también como "enjambre", ofrece un enorme potencial para distribuir tareas y coordinar la operación de diferentes aeronaves de forma rápida y eficiente en áreas como la búsqueda y rescate en regiones extensas, o el apoyo a equipos de emergencia durante desastres naturales, como la monitorización de incendios forestales, tal y como muestra el esquema de la Figura 2.5.

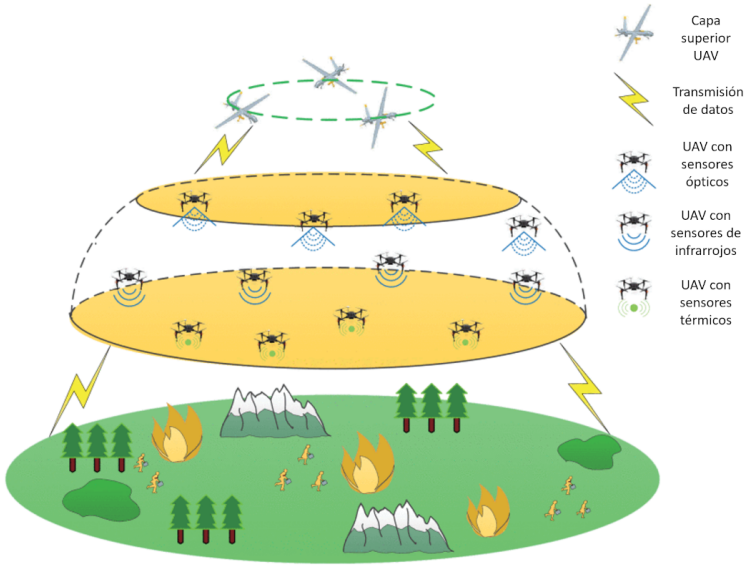

Figura 2.5: *Configuración en enjambre para prevención e inspección de incendios forestales*

Fuente: *Zhang et al.[26]*

Inspirados en el comportamiento de ciertos grupos de aves e insectos y partiendo de trabajos pioneros como el de Craig Reynolds [27], que exploró cómo los comportamientos simples de cada individuo de una bandada de pájaros o banco de peces, podían combinarse para producir comportamientos complejos, los algoritmos para operar enjambres de drones proporcionan un enfoque novedoso para coordinar varias aeronaves sin necesidad de contar con un operador humano por cada unidad. Estos sistemas pueden tomar decisiones de manera autónoma al compartir información entre sí, mejorando su eficacia y adaptabilidad a diversas situaciones.

Sin embargo, todavía son muchos los desafíos que se deben superar, ya que a medida que aumenta el número de drones, también lo hace la complejidad de su coordinación, como el cálculo de la trayectoria óptima que debe seguir cada individuo, lo que incrementa el riesgo de colisiones entre ellos. Esto es debido a que el vuelo en enjambre es más sofisticado que el individual, ya que los drones se deben coordinar mientras mantienen distintas formaciones de vuelo.

2.3. Aprendizaje por refuerzo y navegación autónoma

Durante la última década, ha habido un crecimiento considerable en el interés por aplicar el aprendizaje por refuerzo en la navegación autónoma de vehículos aéreos no tripulados, tal y como reflejan los trabajos de Azar et al. [2] o AlMahamid y Grolinger [3] que analizan el estado del arte en esta materia y servirán como referencias en este apartado. Sin duda, este aumento en interés por los algoritmos RL se debe a los notables avances logrados en áreas como el control de estas aeronaves y la planificación de rutas.

2.3.1. Ventajas

La principal ventaja del aprendizaje por refuerzo en la navegación autónoma de drones, especialmente con algoritmos DRL, es su capacidad para desarrollar agentes que aprenden y se adaptan a entornos desconocidos y variables. En contraste, soluciones más convencionales como SLAM [22] dependen del mapeo y conocimiento previo del entorno, limitando su capacidad de respuesta a situaciones cambiantes, como variaciones en las condiciones atmosféricas o la presencia de obstáculos dinámicos. Esta habilidad para tomar decisiones "inteligentes" brinda mayor flexibilidad y robustez frente a errores y fallos, permitiendo abordar soluciones más complejas, como la colaboración entre drones, necesaria para el funcionamiento de los enjambres mencionados anteriormente. Esto amplía significativamente los casos de uso en comparación con otras soluciones más tradicionales.

Los sistemas basados en algoritmos RL ofrecen otra ventaja importante al requerir sensores menos sofisticados y costosos. Esto es especialmente relevante para drones de bajo coste, ya que algunos dispositivos como los escáneres LiDAR suelen ser costosos y pueden consumir bastante energía. Al aprovechar el aprendizaje y la adaptabilidad de los algoritmos de DRL, los drones pueden utilizar sensores más simples y aún así navegar de manera efectiva. Esto no solo mejora la eficiencia y la autonomía del dron, sino que también reduce significativamente su coste.

Por otro lado, en determinadas ocasiones algunos dispositivos utilizados habitualmente en las tareas de la navegación autónoma pueden no estar operativos, como los sistemas de navegación por satélite al sobrevolar áreas restringidas. En estas situaciones, los algoritmos DRL pueden resultar muy útiles, ya que permiten desarrollar sistemas sin esa dependencia, utilizando exclusivamente sensores visuales.

2.3.2. Limitaciones y consideraciones a tener en cuenta

Como se mencionó anteriormente, la navegación autónoma mediante técnicas de aprendizaje por refuerzo es un campo en pleno desarrollo, lo que implica que aún existen numerosos

problemas por resolver. No obstante, es previsible que en los próximos años se logren avances significativos y se encuentren soluciones a algunos de estos retos:

- En cuanto a los algoritmos DRL, uno de los desafíos es la gran cantidad de datos y la complejidad de la arquitectura de red necesaria para obtener resultados óptimos. Las DNN requieren altas capacidades de cálculo, lo que se puede traducir en un mayor consumo de energía y aumento del tiempo necesario para procesar y responder a la información recibida del entorno. Esto dificulta su implementación en dispositivos con recursos limitados como algunos drones, especialmente los más asequibles.

- El entrenamiento de los sistemas basados en RL generalmente se lleva a cabo en entornos virtuales, debido a las limitaciones de procesamiento y consumo de energía de los drones, así como al alto coste que puede suponer dañarlos accidentalmente durante el proceso. Sin embargo, la realidad suele ser más compleja que los entornos simulados, lo que plantea desafíos en la transferencia del conocimiento adquirido por el agente en el entorno virtual a escenarios reales, puesto que la efectividad de esta depende en gran medida de la similitud entre los entornos de entrenamiento y prueba.

  Algunos entornos de simulación 3D - como los basados en los motores de videojuegos Unreal Engine [10] o Unity [28] - pueden representar condiciones ambientales de manera bastante realista, pero aún queda bastante por investigar en métodos de entrenamiento más generales que aseguren un rendimiento consistente en diferentes entornos y bajo diversas condiciones.

- Como se indicó previamente, la coordinación de la navegación autónoma de múltiples drones es una tarea muy compleja que aún se encuentra en una etapa de investigación temprana. En la navegación en enjambre, los sistemas RL deben sincronizar el vuelo de los drones para mantener su formación, al mismo tiempo que llevan a cabo otras tareas, como el control de cada aparato para evitar que colisionen entre sí o con obstáculos. Aunque se han logrado avances notables [29], todavía hay trabajo por hacer en este campo para que los enjambres puedan llevar a cabo sus tareas de manera eficiente.

- Según señalan AlMahamid y Grolinger [3], problemas como la navegación autónoma requieren desarrollar técnicas de evaluación y comparación de algoritmos de aprendizaje por refuerzo propias. El motivo es que es necesario considerar aspectos diferentes a los utilizados en las habituales evaluaciones con videojuegos clásicos, como los de la consola Atari 2600 [30], empleadas en numerosas publicaciones. En particular, la evaluación de algoritmos para la navegación autónoma debe tener en cuenta aspectos clave como

el conocimiento parcial del estado del entorno y la importancia de la profundidad en la información visual.

- Dada la gran variedad de algoritmos RL existentes, escoger el más apropiado para resolver una tarea de navegación autónoma puede resultar difícil. Indudablemente, el primer paso es adquirir una comprensión profunda del problema para diseñar una representación de los estados y función de recompensa que facilite la obtención de resultados óptimos. El diagrama de la Figura 2.6 basado en el esquema propuesto en [3], muestra otras cuestiones que se deben tener en cuenta a la hora de elegir el algoritmo adecuado.

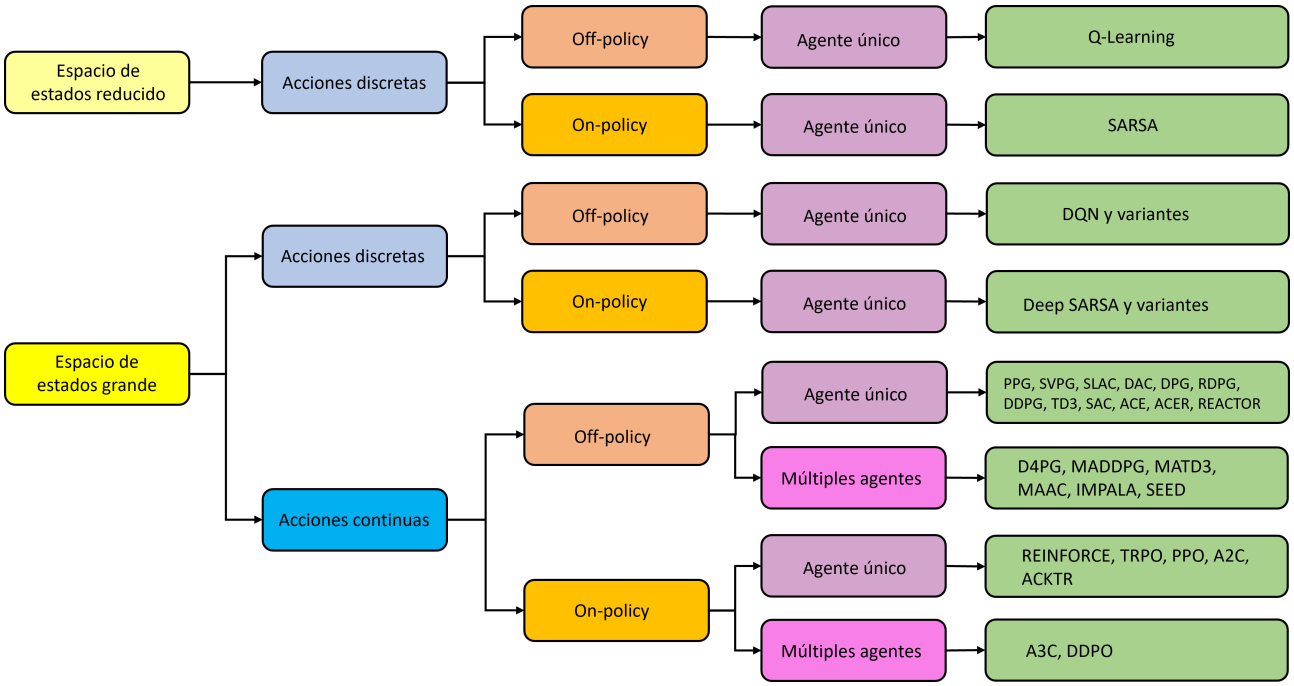

Figura 2.6: *Esquema simplificado para elegir un algoritmo RL adecuado*

Fuente: *Adaptación propia de [3]*

Siguiendo este esquema y salvo casos de uso muy particulares, la navegación autónoma se suele presentar como un problema con un espacio de estados muy grande. Esto es especialmente cierto si en los estados se consideran características como la posición, la velocidad y la orientación del dron, o la información visual capturada por una cámara instalada a bordo.

La siguiente cuestión a resolver sería si se requiere un espacio de acciones discreto o continuo. El primero implica que el dron puede ejecutar un conjunto finito y bien definido de acciones como subir, bajar o girar a la derecha e izquierda. Aunque esto puede simplificar bastante el problema, podría no ser apropiado en situaciones que exijan mayor precisión

en los movimientos. En tales casos, optar por acciones en un rango continuo ofrecería mayor flexibilidad y control sobre el desplazamiento de la aeronave, lo cual puede resultar preferible en entornos muy complejos.

Otro aspecto a determinar es si el algoritmo debe ser de tipo *on-policy* u *off-policy*. Los algoritmos *on-policy* proporcionan un entrenamiento estable, ya que tratan de mejorar la misma política que utilizan para tomar decisiones, y tienden a converger más rápido que los *off-policy*, sin embargo, pueden ser más propensos a hacerlo hacia una política subóptima.

El siguiente paso en este esquema simplificado sería determinar si el problema requiere de un único agente o de varios. La utilización de múltiples agentes puede acelerar el proceso de aprendizaje pero también aumentar la complejidad del sistema y requerir más recursos computacionales. Por otro lado, un agente único podría no ser suficiente para abordar problemas complejos que requieran una solución colaborativa.

Finalmente, quedaría seleccionar uno de los algoritmos que se adapten a las respuestas dadas a las cuestiones planteadas. Es importante tener en cuenta que su rendimiento no solo dependerá de su adecuación al problema, sino también de otros factores como la representación de los estados y el diseño de la función de recompensa indicados anteriormente, o la estrategia de exploración del entorno, la configuración de hiperparámetros del algoritmo o la arquitectura de red del agente, si se trata de un algoritmo DRL.

Capítulo 3

Modelo del entorno y diseño del agente

Como se mencionó en el Capítulo 2, la detección y evasión de obstáculos es esencial para lograr la navegación autónoma de drones. En este trabajo, se propone una solución basada en algoritmos de aprendizaje por refuerzo profundo, combinados con técnicas de visión artificial, con el fin de garantizar una navegación segura para el dron.

Para facilitar la comprensión de cómo se ha abordado el problema, en las siguientes secciones se detallan las características del entorno, el espacio de estados y acciones, la función de recompensa, el algoritmo DRL y la arquitectura de la red neuronal del agente. Además, se presentan las herramientas utilizadas en el desarrollo de la solución propuesta.

3.1. Características del entorno

Para llevar a cabo la implementación de la solución propuesta, se ha empleado tanto un entorno simulado como uno real. En ambos casos, se han seleccionado estancias amplias dentro de una vivienda, garantizando así suficiente espacio para que el dron maniobre y evite obstáculos. Los escenarios incluyen elementos comunes en el interior de una casa, como puertas, muebles y objetos decorativos, que representan los obstáculos típicos que el dron podría encontrar al navegar en dicho entorno.

El uso del entorno virtual tiene como objetivo evitar dañar accidentalmente al dron durante el entrenamiento, así como superar sus limitaciones de autonomía de vuelo. Para facilitar el proceso de transferencia del conocimiento adquirido por el agente en el entorno virtual al escenario real, se ha elegido un entorno 3D que cuenta con varias habitaciones representadas con gran realismo, gracias al uso de texturas de alta resolución y al elevado nivel de detalle con el que están representados los diferentes elementos y objetos. Las figuras 3.1 y 3.2 muestran, respectivamente, el mapa de distribución de las estancias de la vivienda 'virtual' y el contenido de una de ellas.

Figura 3.1: *Plano de distribución de las estancias del entorno virtual*

Fuente: *Gabro Media y Unreal Engine Marketplace [31]*

Figura 3.2: *Estancia principal del escenario virtual*

Fuente: *Gabro Media y Unreal Engine Marketplace [31]*

3.1.1. Espacio de observaciones

El espacio de observaciones en un problema de aprendizaje por refuerzo debe ofrecer al agente información detallada sobre el entorno para que pueda tomar decisiones adecuadas. La siguiente tupla muestra los elementos incluidos en el espacio de observaciones que se ha diseñado para ayudar al agente a evitar colisiones y maximizar la distancia recorrida durante el vuelo:

(*imagen RGB*, *imagen mapa profundidad*, *profundidad escena*, *distancia recorrida*)

La descripción de estos elementos es la siguiente:

- Imagen RGB: imagen de 300x300 píxeles capturada por la cámara frontal del dron. Esta representa su perspectiva en primera persona y es fundamental para que el agente pueda obtener información visual sobre su ubicación y orientación en el espacio, así como para detectar obstáculos y otros elementos relevantes para la toma de decisiones.

- Imagen del mapa de profundidad: se trata de una representación gráfica derivada de la imagen capturada por la cámara del dron. En ella, cada píxel representa la distancia entre el plano de la cámara y los distintos objetos de la escena. Estas distancias se visualizan a través de una escala de grises, tal y como se puede observar en la Figura 3.3, donde los objetos más cercanos se presentan en tonos claros, y los más distantes, en tonos oscuros. Su objetivo es permitir al agente una comprensión más efectiva de la estructura tridimensional del entorno que le rodea.

  En este trabajo, los mapas de profundidad son generados por la red neuronal "MiDaS" [32], previamente entrenada para esta tarea, cuyos detalles se describen en el apartado 3.3.5. Los mapas creados tienen unas dimensiones de 300x300 píxeles y se caracterizan por ser "inversos", lo cual implica que los píxeles correspondientes a objetos cercanos tienen valores numéricos altos en comparación con los más alejados.

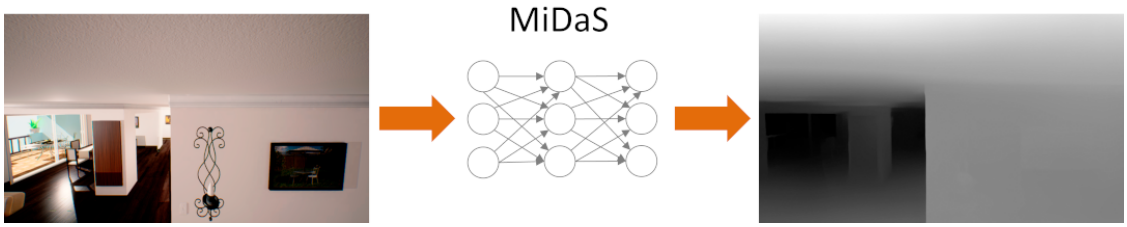

Figura 3.3: *Imagen capturada en el entorno virtual y mapa de profundidad generado*

Fuente: *Elaboración propia*

- Profundidad de la escena: es un número decimal obtenido a partir de los valores de los píxeles de dos regiones de interés (ROI, por las siglas de *Region Of Interest*) rectangulares definidas en el mapa de profundidad de la escena captada por el dron.

  Como se puede ver en la Figura 3.4 , la ROI más estrecha proporciona una visión limitada, centrándose en los valores de píxeles correspondientes a los objetos directamente en la trayectoria del dron. La ROI más grande, por otro lado, emula un campo de visión más amplio, detectando obstáculos cuando el dron está próximo a los límites de la habitación en la que se encuentra.

Para el cálculo del valor decimal de profundidad, se tiene en cuenta un umbral preestablecido, que representa un valor crítico de píxel a partir del cual se asume que los obstáculos están demasiado cerca del dron. Si ningún píxel en el mapa de profundidad excede este umbral, la "profundidad de la escena" corresponde al valor medio de los píxeles en la ROI más estrecha. Sin embargo, si se supera este umbral se considera que el dron está muy cerca de un obstáculo y se devuelve el máximo de los píxeles en la ROI más grande.

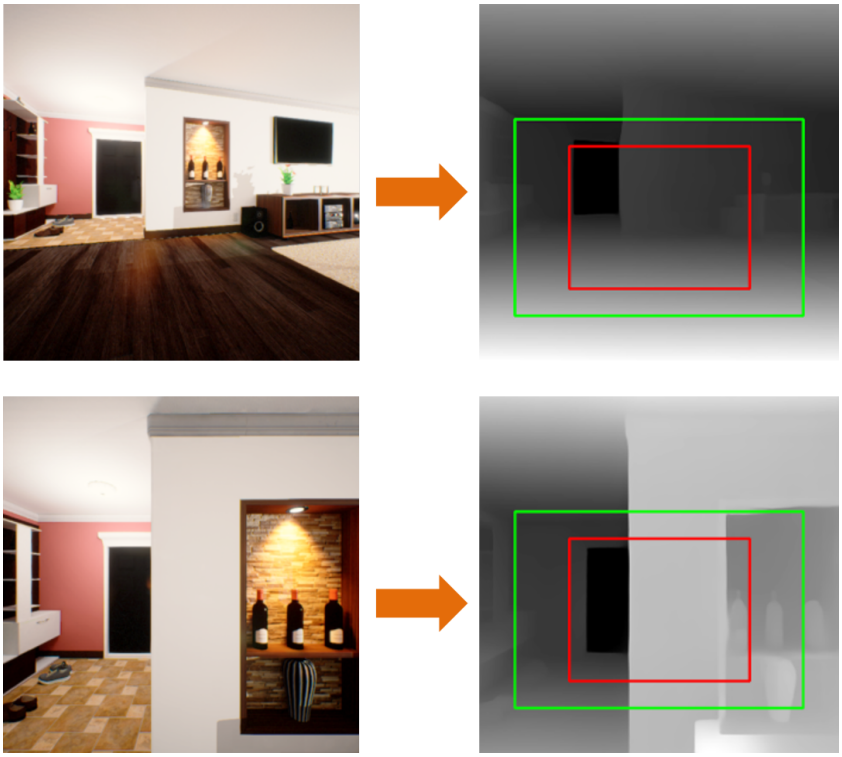

Figura 3.4: *Regiones de interés para calcular la profundidad de la escena*

Fuente: *Elaboración propia*

Este método de cálculo se ha diseñado considerando los efectos de la perspectiva en la escena captada por la cámara del dron: cuando está alejado de una pared, los obstáculos en la región central de la imagen son los más relevantes para su trayectoria, por lo que el valor medio de esta región puede considerarse una estimación adecuada del riesgo de colisión. Pero cuando el dron está cerca de una pared, la perspectiva cambia, y los obstáculos que aparecen cerca de los bordes de la imagen, aunque estén ligeramente desviados de la trayectoria del dron, pueden representar un riesgo significativo de colisión. En estas circunstancias, proporcionar el valor máximo de la región más grande puede ayudar a prevenir una colisión inminente.

La "profundidad de la escena"tiene como objetivo facilitar la toma de decisiones por parte

del agente al proporcionar una métrica simplificada de los riesgos potenciales de colisión en diferentes contextos. Como se detallará en el apartado 3.1.3, juega un papel esencial en la generación de recompensas, impactando de manera directa en cómo el agente aprende a evitar colisiones.

- Distancia recorrida: esta medida, expresada en metros, representa la distancia acumulada que el dron ha recorrido desde el comienzo de cada episodio. Su cálculo se basa en las velocidades lineales a lo largo de los ejes X, Y y Z, cada una multiplicada por el tiempo transcurrido desde la última acción, sumando las distancias obtenidas en cada eje. Esta valoración se actualiza tras cada acción ejecutada por el agente, ofreciendo así una estimación continua de la distancia total que el dron ha recorrido.

  Como se explicará en el apartado 3.1.3, este valor desempeña un papel fundamental en el proceso de aprendizaje del agente: cuanto mayor sea la distancia recorrida, más grande será la recompensa recibida, incentivándolo así a seguir avanzando. Además, esta medida también sirve para evaluar si el agente está explorando activamente el entorno o si, por el contrario, se encuentra estancado en una posición estacionaria.

3.1.2. Espacio de acciones

Aunque un dron tiene la capacidad de moverse en cualquier dirección en el espacio tridimensional, en este trabajo se ha optado por emplear un espacio de acciones discreto para simplificar tanto computacional como conceptualmente el proceso de entrenamiento del agente. De esta manera, se ha definido un conjunto limitado de acciones predefinidas que hacen que el problema sea más manejable y fácil de abordar en comparación con un espacio de acciones continuo. Como resultado, se han definido las siguientes tres acciones que el dron puede tomar:

- Avanzar: el dron avanza hacia adelante a una velocidad fija de 0,5 m/s durante 1 segundo.
- Girar a la izquierda: el dron gira sobre su eje vertical 45 grados en sentido antihorario.
- Girar a la derecha: el dron gira sobre su eje vertical 45 grados en sentido horario.

Estas acciones se implementan a través de un conjunto de funciones de código Python que envían la señal de control correspondiente al dron, ya sea a través de la API del simulador en el entorno virtual o directamente al dron en el entorno real.

3.1.3. Función de recompensa

La función de recompensa desempeña un papel esencial al proporcionar retroalimentación al agente acerca de la eficacia de sus acciones en relación con el objetivo propuesto. En el caso

planteado en este trabajo, la función ha sido diseñada para tener en cuenta tanto la capacidad del dron para evitar colisiones, como su progresión a través del entorno, incentivándolo a mantenerse en movimiento en lugar de quedarse estático, como se puede ver en el Algoritmo 1.

1 Algoritmo de la función de recompensa (Fuente: *elaboración propia)*

```
function CALCULAR RECOMPENSA(acción, profundidad_media, distancia_recorrida)
    recompensa ← 0
    if colisión then
        recompensa ← recompensa − penalización_colisión
        finalizar_episodio ← verdadero
    else
        if acción == "avanzar" then
            recompensa ← (1 + distancia_recorrida ∗ F_r)/(profundidad_media + 1)
        else
            recompensa ← 1/(profundidad_media + 1)
        end if
    end if
    return recompensa
end function
```

La función recibe la acción ejecutada, la profundidad media de la imagen tomada por la cámara del dron y la distancia recorrida durante el episodio. Si la acción ejecutada es "avanzar", la recompensa se incrementa proporcionalmente a la distancia recorrida. Para ajustar la importancia relativa de esta distancia, se utiliza el factor de ponderación F_{r}, cuyo valor se determina empíricamente, tal y como se indica en el apartado 4.1 del próximo capítulo.

Tanto si el dron avanza como si gira, la recompensa se divide por el valor de la profundidad media. Esto significa que, si existe espacio libre (valor bajo de profundidad media, ya que se trabaja con mapas de profundidad inversos), la recompensa será mayor. Por el contrario, si el espacio libre es limitado, la recompensa será menor, dado que un valor alto de profundidad media indica mayor proximidad a los obstáculos. De esta manera, se alienta al dron a desplazarse hacia áreas despejadas, reduciendo el riesgo de que se produzcan colisiones.

En el caso de que se produzca una colisión, el agente recibe una penalización que se sustrae de la recompensa total. Este valor de penalización se determina empíricamente y es lo suficientemente significativo como para desincentivar al dron de encontrarse con obstáculos. Además, si se produce este hecho, el episodio actual finaliza y se debe comenzar uno nuevo.

3.2. Diseño del agente

3.2.1. Algoritmo seleccionado

El algoritmo elegido para abordar el problema planteado en este trabajo es DQN (*Deep Q-Network*) [18], una extensión del método *Q-Learning* [14], que reemplaza la tabla de valores Q por una red neuronal que aproxima la función de valor de acción óptima para diferentes pares estado-acción.

Considerando el esquema propuesto en la Figura 2.6 y las características del problema, la elección de este algoritmo se fundamenta en las siguientes razones:

- El espacio de observaciones es de alta dimensionalidad, ya que incluye matrices de datos como la imagen RGB y el mapa de profundidad. Esto hace que los métodos de aprendizaje por refuerzo basados en tablas sean inadecuados, debido a la cantidad de información única que contienen las observaciones. Sin embargo, este escenario se ajusta perfectamente a la estrategia de aproximar la función de valor mediante una red neuronal de DQN.
- El espacio de acciones es limitado (avanzar, girar a la izquierda o derecha), lo que permite que la red neuronal se enfoque en aprender a mapear las complejas observaciones del estado a estas acciones discretas.
- DQN es un algoritmo *off-policy*, lo cual es especialmente útil en este contexto, ya que permite que el agente explore y aprenda de sus experiencias de manera simultánea.

A continuación, se describen de manera concisa las dos variantes del algoritmo DQN empleadas en este trabajo.

3.2.1.1. DQN con política ϵ-*greedy*, *experience replay buffer* y *target network*

Se trata de una versión mejorada del algoritmo DQN original, desarrollada por sus propios autores y presentada en [30]. Esta añade tres componentes clave:

- Una política *épsilon-greedy*, que permite al agente explorar el entorno tomando acciones aleatorias con una probabilidad épsilon (ϵ) y tomar la acción óptima según la función de valor con una probabilidad 1-ϵ. Esta política equilibra la exploración y la explotación, permitiendo al agente descubrir nuevas acciones y, al mismo tiempo, aprovechar el conocimiento adquirido hasta el momento.
- Un búfer de repetición de experiencias (*experience replay buffer*), para abordar el problema de los datos secuenciales y altamente correlacionados suministrados a la red durante

el entrenamiento. Este mecanismo permite almacenar experiencias pasadas - tuplas que contienen el estado en un momento dado, la acción tomada, la recompensa recibida y el siguiente estado - y seleccionar un subconjunto aleatorio de las mismas para el entrenamiento, lo que reduce la correlación, obteniéndose datos más independientes.

- Una red "objetivo" (*target network*), que es una copia de la red principal cuyos pesos permanecen fijos durante cierto tiempo, evitando que las actualizaciones de la función de valor afecten negativamente a los estados futuros. Esta estabilidad se logra actualizando periódicamente los pesos de la red objetivo con los de la red principal.

El algoritmo 2 muestra el funcionamiento de DQN con las mejoras añadidas:

2 Algoritmo DQN con *epsilon-greedy*, *experience replay buffer* y *target network* (Fuente: [33])

Inicializar el replay buffer D
Inicializar Q_θ con pesos aleatorios θ
Inicializar Q^{θ^-}, idéntica a Q_θ, $\theta^- = \theta$
Inicializar ϵ
for *episodio* $= 1$ hasta M **do**
  for $t = 1$ hasta T **do**
    Según la probabilidad ϵ seleccionar una acción aleatoria a dado un estado s_t
    Ejecutar la acción a y observar la recompensa r y el siguiente estado s'
    Almacenar el vector (s, a, r, s') en el replay buffer D
    Extraer un subconjunto aleatorio (s_j, a_j, r_j, s'_j) de J transiciones del replay buffer D
    for cada transición en el buffer **do**
      if el episodio ha terminado **then**
        Calcular variable objetivo $y_j = r_j$
      else
        Calcular variable objetivo $y_j = r_j + \gamma \, \text{máx}_{a'} \, Q^{\theta^-}(s'_j, a'; \theta^-)$
      end if
    end for
    Calcular la pérdida $L = \frac{1}{J} \sum_{j=0}^{J-1} [Q(s_j, a_j; \theta) - y_j]^2$
    Actualizar Q con SGD minimizando la pérdida
    Cada N iteracciones clonar los pesos de Q a Q^{θ^-} (i.e $\theta^- = \theta$)
  end for
end for

3.2.1.2. *Quantile Regression* DQN (QR-DQN)

QR-DQN es una variante de DQN propuesta por Dabney et al. [34] que incorpora las mejoras mencionadas en el apartado anterior. Sin embargo, se diferencia en su enfoque para estimar los valores de las acciones: mientras que DQN estima un único valor óptimo para cada par estado-acción, QR-DQN estima una distribución de posibles valores.

Para lograr esto, este algoritmo utiliza una distribución de probabilidad de cuantiles, que son puntos tomados a intervalos regulares en un conjunto de datos y representan la proporción de los datos que se encuentran por debajo de cada uno de ellos. Por ejemplo, el cuantil del 50 %, conocido como la "mediana", es el valor por debajo del cual se encuentra el 50 % de los datos.

Esta distribución de probabilidad de cuantiles proporciona una visión más completa de las diferentes recompensas que pueden asociarse a una acción en un estado determinado, ya que permite considerar tanto el valor medio esperado como la variabilidad de los valores. Esto es especialmente útil cuando una misma acción puede generar recompensas muy variables debido a la complejidad del entorno.

En el contexto de este trabajo, un ejemplo de la utilidad de este enfoque puede ser la acción "avanzar". Ésta suele tener un valor medio alto, ya que generalmente permite cubrir más distancia, pero también tiene una variabilidad alta, lo que implica que en algunas ocasiones puede llevar al dron a chocar con obstáculos. La capacidad de QR-DQN para estimar una distribución de valores de recompensa asociados a esta acción puede resultar crucial para tomar decisiones acertadas en un entorno complejo como este.

Para convertir DQN en un método "distribucional", son necesarios ajustes en la capa de salida de la arquitectura de la red para que tenga un tamaño adecuado al número de cuantiles. También se deben reemplazar la función de pérdida y el algoritmo de optimización por versiones adecuadas para trabajar con cuantiles.

El código del Algoritmo 3 muestra de manera simplificada la propuesta de los autores de QR-DQN para calcular el objetivo de Bellman "distribucional" y la pérdida de regresión de cuantiles. El primero se obtiene ponderando la suma de los valores cuantiles estimados. En este caso, el vector de parámetros θ se utiliza para estimar la distribución de valores cuantiles. Por otro lado, la pérdida de regresión de cuantiles se utiliza para actualizar los parámetros de la red neuronal y mejorar la estimación de los valores cuantiles.

3 *Quantile Regression Deep Q-Learning* (Fuente: [34])

Requerido: N, κ

Entrada: $s, a, r, s', \gamma \in [0, 1)$

Calcular el objetivo de Bellman distribucional

$Q(s', a') := \sum_{q_j} \theta_j(s', a')$

$a^* \leftarrow \arg\max_{a'} Q(s, a')$

$T\theta_j \leftarrow r + \gamma\theta_j(s', a^*), \forall j$

Calcular la pérdida de regresión de cuantiles

Salida: $\sum_{i=1}^{N} E_j \rho_{\kappa\tau_i}(T\theta_j - \theta_i(s, a))$

3.2.2. Arquitectura de red del agente

En un algoritmo de Aprendizaje por Refuerzo Profundo, la arquitectura de red del agente tiene la misión esencial de generar una estimación precisa de la acción óptima en función del estado del entorno. Las dos variantes del algoritmo DQN empleadas en este trabajo comparten la misma arquitectura, la cual ha sido diseñada teniendo en cuenta la diferente naturaleza de los datos contenidos en las observaciones, que incluyen dos imágenes (imagen RGB y mapa de profundidad) y dos valores escalares (profundidad media y distancia recorrida).

Para procesar de manera eficiente estos datos, se ha optado por integrar varias redes neuronales. Esta es una estrategia habitual en la resolución de problemas que requieren la unión de distintos tipos de datos y redes, tal y como sucede en problemas de robótica que implican combinar imágenes con datos de diferentes sensores [35], así como otros campos de aplicación con problemáticas similares, como el caso presentado en [36].

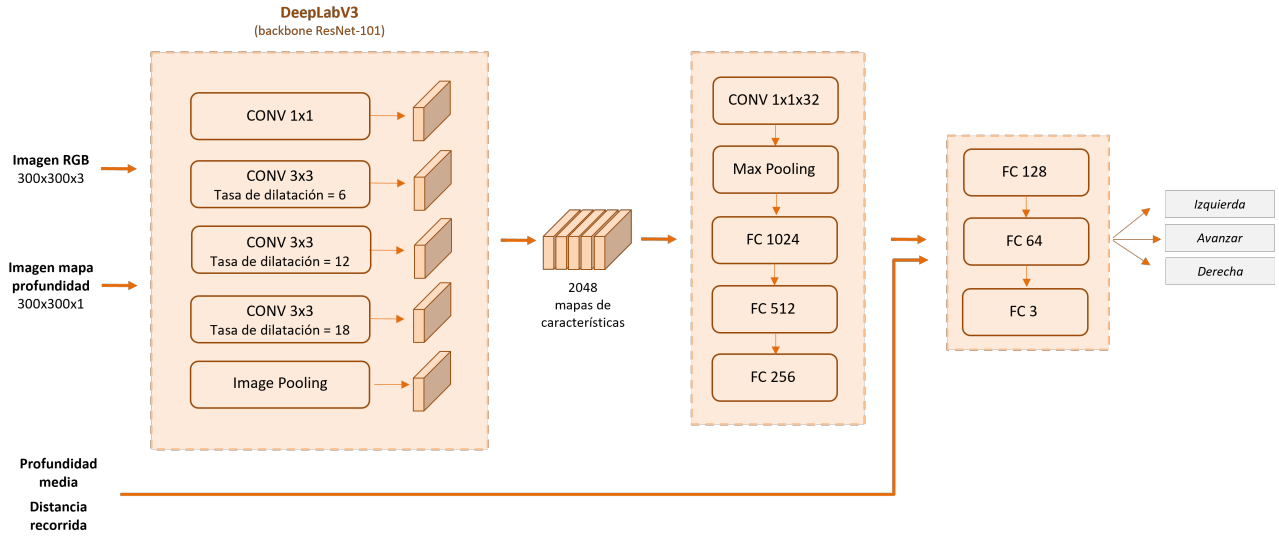

Figura 3.5: *Arquitectura de red propuesta para el agente*

Fuente: *Adaptación propia a partir de* [37]

Como se muestra en la Figura 3.5, la arquitectura diseñada emplea en primer lugar una red convolucional profunda previamente entrenada. Esta se utiliza para extraer características de alto nivel tanto de las imágenes RGB como de los mapas de profundidad. La red elegida para este propósito es DeepLabV3 [38]. En concreto, una implementación [39] pre-entrenada en un extenso conjunto de datos de imágenes del mundo real que emplea como base o *backbone* la arquitectura ResNet-101 [40], ampliamente reconocida por su habilidad para extraer características de alto nivel de las imágenes.

El motivo por el que se ha elegido DeepLabV3 es su capacidad para realizar de manera eficaz la "segmentación semántica" de imágenes, una tarea común en los sistemas de visión

artificial, que consiste en asignar a cada píxel una etiqueta correspondiente a la clase de objeto que representa. Esto es especialmente útil en el contexto de navegación autónoma, ya que puede permitirle distinguir diferentes tipos de objetos y estructuras como paredes, puertas o muebles.

La arquitectura, tal y como se muestra en la Figura 3.5, conserva las capas iniciales de la red DeepLabV3 con sus pesos, estas implementan una técnica denominada convolución "dilatada", que permite capturar mayor información contextual de las imágenes que las capas convolucionales convencionales. El modelo propuesto prescinde, sin embargo, del clasificador de DeepLabV3, permitiendo así que la salida de sus capas convolucionales sirva como extractor de características. Esta información alimenta las capas restantes de la arquitectura, que se encargan de aprender a detectar y evitar obstáculos.

La salida de esta primera red produce un total de 2048 mapas de características, que se procesan en la siguiente subred mediante una capa convolucional añadida para reducir el número de mapas a 32. A continuación, se aplica una capa de agrupación que reduce la resolución espacial de los mapas de 300x300 a 50x50 píxeles. Los mapas resultantes se "aplanan", es decir, se transforman de matrices a vectores, y se introducen en una serie de capas totalmente conectadas (*Fully-Connected*, FC), cada una seguida por una función de activación ReLU y una operación de *dropout* para prevenir el sobreajuste durante el entrenamiento.

Finalmente, las salidas de esta subred y los valores escalares normalizados se combinan dando lugar a un tensor de 256 + 2 elementos, que se introduce en la subred FC final del modelo, cuya última capa tiene una dimensión igual al número de acciones posibles. Esta produce una salida que será la estimación la mejor acción a tomar por el agente en un estado determinado, según el algoritmo DQN utilizado.

3.3. Herramientas utilizadas

En términos generales, el agente se ha programado con la librería Pytorch [8] y los algoritmos DRL han sido implementados mediante Stable-Baselines [41]. Durante el proceso de entrenamiento en el entorno virtual, se han llevado a cabo simulaciones de vuelo mediante AirSim [9] y Unreal Engine [10]. Para validar el modelo en un entorno real, se ha utilizado un conjunto de herramientas de software suministradas por el fabricante del dron físico [42] y bibliotecas de terceros que integran y extienden dichas herramientas [43]. Por último, para generar los mapas de profundidad, se ha empleado el modelo pre-entrenado MiDaS [44] disponible en Pytorch Hubs [45].

A continuación, se describen con mayor detalle estas herramientas:

3.3.1. Pytorch

PyTorch es una biblioteca de aprendizaje profundo desarrollada por Facebook AI que combina facilidad de uso y eficiencia. Es de código abierto, compatible con los sistemas operativos Linux, macOS y Windows, y está escrita en Python, C++ y CUDA.

Entre sus características principales se encuentra el tensor, una estructura de datos similar a un array multidimensional. Los tensores se emplean para representar datos de entrada, pesos o gradientes en redes neuronales y permiten realizar operaciones matemáticas en paralelo con los datos que almacenan, lo que mejora significativamente el proceso de cálculo y rendimiento de los modelos.

PyTorch también ofrece diferenciación automática, permitiendo el cálculo de gradientes para optimizar modelos de manera fácil y eficiente, y es capaz de aprovecha la aceleración en operaciones matemáticas mediante el uso de unidades de procesamiento gráfico (GPU), lo que permite una ejecución más rápida durante el entrenamiento y la inferencia.

3.3.2. Stable-Baselines3

Stable-Baselines3 (SB3) es una biblioteca de implementaciones de algoritmos de aprendizaje por refuerzo basada en PyTorch. Se caracteriza por tener una interfaz limpia y sencilla, permitiendo entrenar agentes RL con solo unas pocas líneas de código. Esto facilita centrarse en la experimentación y el ajuste del algoritmo, ahorrando dedicar tiempo a detalles de implementación complejos.

En general, la documentación de SB3 es muy completa y las implementaciones de los algoritmos han sido probadas rigurosamente para garantizar su estabilidad. La biblioteca ofrece una API consistente y limpia contando con algoritmos de última generación, como TD3, PPO, DDPG o DQN. Además, la biblioteca cuenta con una comunidad muy activa, lo que facilita la búsqueda de ayuda y ejemplos para resolver problemas específicos.

3.3.3. AirSim

AirSim es una plataforma de simulación de código abierto desarrollada por Microsoft Research en 2017, diseñada para la experimentación e investigación con drones y automóviles. Se utiliza ampliamente para probar algoritmos de aprendizaje profundo, visión por computadora o aprendizaje por refuerzo, enfocados en el desarrollo de sistemas de navegación autónoma.

AirSim es compatible con la versión 4 de Unreal Engine y se puede ejecutar en numerosos escenarios virtuales desarrollados con este motor gráfico, como paisajes naturales, entornos urbanos o espacios interiores, simulando efectos ambientales como condiciones de iluminación o diversas condiciones atmosféricas.

Un proyecto de aprendizaje por refuerzo basado en AirSim consta de tres componentes principales: un agente, el propio AirSim como simulador de vuelo o conducción, y un motor gráfico 3D, principalmente Unreal Engine, aunque existe una versión experimental para el motor gráfico Unity. El agente interactúa con AirSim para enviar y recibir acciones al vehículo y recibir estados, mientras que el motor gráfico proporciona un entorno realista con un motor de física que simula la dinámica del mundo real.

Cabe destacar que en 2022, el repositorio original de AirSim fue archivado por Microsoft para ser reemplazado por un nuevo producto comercial [46] basado en esta plataforma. Según la compañía, este nuevo producto ofrecerá herramientas y características más avanzadas, adaptadas específicamente a las necesidades de la industria aeroespacial.

3.3.4. Unreal Engine

Unreal Engine es un motor de videojuegos de código abierto y multiplataforma desarrollado por Epic Games. Ampliamente utilizado en la industria del videojuego y en aplicaciones de simulación y visualización arquitectónica, es reconocido por su capacidad para generar entornos y gráficos 3D de alta calidad y muy realistas. Además, cuenta con un motor de física que simula el comportamiento del mundo real con gran precisión y detalle.

En el ámbito del aprendizaje por refuerzo, Unreal Engine se integra con herramientas y bibliotecas de aprendizaje y se puede utilizar en combinación con simuladores como AirSim, para proporcionar entornos realistas en los que los agentes pueden aprender y perfeccionar sus habilidades en diferentes tareas. Estos entornos incluyen condiciones de iluminación variables, diferentes estructuras y objetos, y texturas variadas, lo que permite evaluar algoritmos de aprendizaje por refuerzo en situaciones diversas.

3.3.5. MiDaS

MiDaS [32] es un modelo de estimación de profundidad monocular que utiliza redes neuronales para predecir la profundidad en imágenes 2D. Para lograr una información lo más precisa posible, combina diversos conjuntos de datos, incluyendo algunos basados en películas 3D. Además, utiliza codificadores potentes y preentrenados para mejorar el rendimiento y la generalización en diferentes entornos y situaciones, lo que permite generar mapas de profundidad más detallados y fiables.

Este modelo está disponible en el repositorio PyTorch Hub, que proporciona modelos preentrenados como MiDaS, permitiendo además aprovechar la compatibilidad con GPU de Pytorch.

3.3.6. DJITelloPy

Se trata de una biblioteca Python diseñada para simplificar la interacción con el dron DJI Tello. Está desarrollada utilizando el SDK oficial del fabricante de este dron [42] y permite acceder de manera sencilla a los comandos de control de vuelo, captura y transmisión de vídeo en tiempo real, información del estado del dron y la configuración de diferentes parámetros de vuelo. Además, la biblioteca también proporciona acceso a funciones de bajo nivel para un control preciso de los motores y acceso a los datos de los sensores del dron.

3.4. Acceso al código fuente

El código fuente utilizado para implementar el sistema propuesto en este trabajo está disponible para su consulta en el repositorio: https://github.com/fnunezsanchez/tfm-uoc-data-science

Capítulo 4

Entrenamiento y resultados

En este capítulo, se presentan y analizan los resultados del entrenamiento del agente para las dos variantes del algoritmo DQN introducidas en el apartado 3.2.1 del capítulo anterior. En ambos casos, tanto la implementación como el entrenamiento se ha realizado utilizando las funciones proporcionadas por la librería Stable Baselines [41], la cual proporciona una plataforma sólida para ajustar los diferentes hiperparámetros de los algoritmos y del entorno.

El análisis comparativo de los resultados que se presenta en las siguientes páginas, evalúa el rendimiento de las dos variantes del algoritmo DQN propuestas en términos de su capacidad de aprendizaje, tratando de identificar las fortalezas y debilidades de cada implementación y determinar si alguna de ellas ofrece mejoras significativas en comparación con la otra.

4.1. Selección de los valores de los hiperparámetros

Elegir valores adecuados para los hiperparámetros, tanto del algoritmo como del entorno, es crucial en el entrenamiento de un agente de aprendizaje por refuerzo, ya que puede tener un impacto significativo en los resultados obtenidos. A continuación se detalla la selección que se ha realizado de estos valores con el objetivo de garantizar un entrenamiento y rendimiento razonablemente óptimo del agente.

4.1.1. Hiperparámetros del algoritmo

La configuración de los hiperparámetros del algoritmo se ha basado en gran medida en los valores comúnmente utilizados en trabajos previos, incluyendo los de los autores de DQN [18][30] y QR-DQN [34]. Además, se ha considerado de manera especial la investigación realizada por A. Anwar y A. Raychowdhury [47], dado que abordan una tarea similar a la de este trabajo. También se han realizado ajustes empíricos en algunos de estos hiperparámetros con

el objetivo de lograr un aprendizaje efectivo, teniendo en cuenta la naturaleza de la tarea y las características específicas del problema planteado en este trabajo.

Los siguientes son los valores de los hiperparámetros elegidos para las implementaciones tanto de DQN como de QR-DQN en Stable Baselines:

- *Buffer size = 10000*. Es el tamaño del búfer de repetición de experiencias utilizado en el algoritmo. En este caso, se ha seleccionado el valor más alto permitido por el hardware utilizado para el entrenamiento. Este tamaño se ha elegido con el objetivo de disponer de un conjunto amplio de experiencias pasadas para evitar una dependencia excesiva de las más recientes y problemas de sobreajuste.

- *Learning starts = 5000*. Establece el número de pasos de tiempo necesarios antes de que comience el entrenamiento. Durante este período inicial, el agente recopila datos de experiencia en el búfer antes de comenzar a aprender de ellos. Se ha elegido un valor similar al utilizado en [47], con la intención de almacenar una cantidad diversa de experiencias para intentar estabilizar y mejorar este proceso.

- *Train frequency = 4*. Especifica cada cuántos pasos de tiempo se actualizan los parámetros de la red neuronal. Los valores bajos permiten una adaptación más rápida a cambios en el entorno, pero pueden aumentar la correlación entre las experiencias y dificultar la convergencia del algoritmo. El valor elegido coincide con el utilizado en [30], y fue seleccionado tras varios entrenamientos de prueba con diferentes valores.

- *Target update interval = 1000*. Determina el número de pasos de tiempo entre las actualizaciones de la red neuronal objetivo (*target network*) con los parámetros de la red principal. La primera se utiliza para calcular las estimaciones de los valores esperados de las acciones, siendo estos fundamentales para guiar el proceso de aprendizaje. La elección del valor de este hiperparámetro fue determinada empíricamente, tratando de evitar valores muy bajos que generen inestabilidad al producirse demasiadas actualizaciones, y valores excesivamente altos que puedan ralentizar mucho el proceso.

- *Exploration fraction = 0.45*. Representa la fracción del período de entrenamiento en el cual se reduce gradualmente la tasa de exploración (valor de épsilon). Esto significa que durante ese intervalo, la probabilidad de seleccionar una acción aleatoria en lugar de la indicada por la política que se está aprendiendo, disminuye a medida que avanza el entrenamiento. El valor elegido también se ha determinado de manera empírica, tratando de favorecer la exploración de diferentes combinaciones de acciones que permitan al agente responder de manera eficiente a una mayor variedad de situaciones.

- *Exploration final eps. = 0.01.* Establece el valor final de épsilon, que determina la probabilidad mínima de exploración del agente en las últimas etapas del entrenamiento. Se ha elegido un valor relativamente bajo para priorizar la explotación de la política en el momento que el agente adquiere un mayor conocimiento del entorno.

- *Batch size = 32.* Este parámetro indica el número de ejemplos que se seleccionan aleatoriamente del búfer de experiencias para actualizar los pesos de la red principal en cada paso de entrenamiento. El valor seleccionado, que coincide con lo propuesto en [18] y [47], ofrece, en principio, un equilibrio adecuado entre la carga computacional y la precisión en la estimación de los gradientes.

- *Gamma = 0.99.* Es el "factor de descuento" que determina la importancia relativa de las recompensas futuras en comparación con las recompensas inmediatas. El valor seleccionado es muy común en problemas de aprendizaje por refuerzo y permite que el agente considere las recompensas futuras de manera significativa, promoviendo un aprendizaje a largo plazo.

- *Learning rate = 0.00008.* Es la tasa de aprendizaje que determina la magnitud de las actualizaciones de los pesos de la red en cada iteración del proceso de entrenamiento. El valor escogido es el resultado de experimentar con diversas opciones, seleccionándose la que proporcionaba el mejor equilibrio entre estabilidad y velocidad de aprendizaje. Se puede considerar un valor bajo, lo que permite ajustar con mayor precisión los pesos de la red y evita actualizaciones excesivamente bruscas que podrían afectar negativamente al aprendizaje.

Además de los hiperparámetros anteriores, la implementación de QR-DQN requiere también el siguiente hiperparámetro:

- *N_quantiles = 70.* Determina el número de cuantiles utilizados en la aproximación de la función de valor de acción. Al aumentar este valor, la función de Stable Baselines incrementa automáticamente el número de unidades de salida de la red neuronal del agente en la cantidad indicada. Esto permite una aproximación más precisa de la función de valor de acción, pero también implica un mayor consumo de recursos computacionales. El valor escogido ha sido determinado empíricamente, considerando el amplio rango de recompensas y la alta variabilidad asociada a la acción "avanzar" en el entorno propuesto.

4.1.2. Hiperparámetros del entorno

Estos parámetros corresponden a configuraciones específicas del entorno implementado en este trabajo. Sus valores han sido establecidos de manera empírica, teniendo en cuenta las

características particulares de la tarea de navegación autónoma del dron. Son igualmente relevantes a los parámetros mencionados previamente, ya que desempeñan un papel fundamental en el ajuste del comportamiento del agente y en el mantenimiento de un equilibrio adecuado entre la exploración y la explotación de la información del entorno.

A continuación se presentan los parámetros definidos y sus respectivos valores obtenidos todos ellos de manera empírica:

- *Duración máxima del episodio = 300.* Es el máximo de pasos de tiempo que puede durar cada episodio. El valor establecido equivale aproximadamente a 5 minutos de vuelo. El objetivo de este parámetro es agilizar el proceso de entrenamiento evitando que el agente se quede "atascado" en un episodio determinado.

- *Umbral de profundidad máxima = 220.* Se utiliza para facilitar la detección de obstáculos próximos al dron. Como se mencionó en el apartado 3.1.1, si el valor máximo de los píxeles en la región de interés más amplia del mapa de profundidad supera este umbral, se considera que el dron se encuentra en riesgo de colisión con un obstáculo. El propósito de este hiperparámetro es definir qué se considera como cercano al dron, y su ajuste tiene un impacto directo en el comportamiento del agente durante el entrenamiento.

- *Penalización por colisión = 100.* Representa la recompensa negativa que recibe el agente en caso de colisión del dron con un obstáculo del entorno. La elección de un valor alto para esta penalización tiene como propósito incentivar al agente a evitar colisiones y fomentar la búsqueda de estrategias que maximicen la seguridad del vuelo.

- *Factor de ponderación de la distancia recorrida = 1.5.* Este parámetro permite ajustar la importancia relativa de la distancia recorrida en la recompensa recibida por el agente, como se muestra en el Algoritmo 1. Equilibrar el impacto de la distancia en el cálculo de la recompensa mediante este factor es de gran importancia, ya que un valor demasiado bajo podría no incentivar al agente a avanzar, mientras que uno excesivamente alto podría llevarlo a intentar avanzar a toda costa, sin evitar las colisiones.

4.2. Algoritmo de optimización

Además de los hiperparámetros anteriores, es importante señalar que las dos variantes del algoritmo DQN elegidas, emplean *Adam* [48] durante el entrenamiento para ajustar los pesos de las dos subredes finales de la arquitectura descrita en el apartado 3.2.2. Se trata de un algoritmo de optimización basado en el descenso de gradiente estocástico ampliamente utilizado

en el aprendizaje profundo. Éste se encarga de actualizar los pesos de la red de acuerdo con la función de pérdida calculada en cada iteración del entrenamiento, que en este caso es el error cuadrático medio o MSE (*Mean Squared Error*).

4.3. Entrenamiento

Debido a las restricciones de calendario y recursos computacionales disponibles, el entrenamiento de las versiones del algoritmo DQN elegidas se ha limitado a un total de 50.000 pasos de tiempo. Los siguientes apartados muestran los aspectos más destacados de los resultados obtenidos con la configuración descrita anteriormente.

4.3.1. Recompensas obtenidas por episodio

En la gráfica de la Figura 4.1, se muestra la evolución de las recompensas obtenidas durante los episodios de entrenamiento de los agentes DQN y QR-DQN.

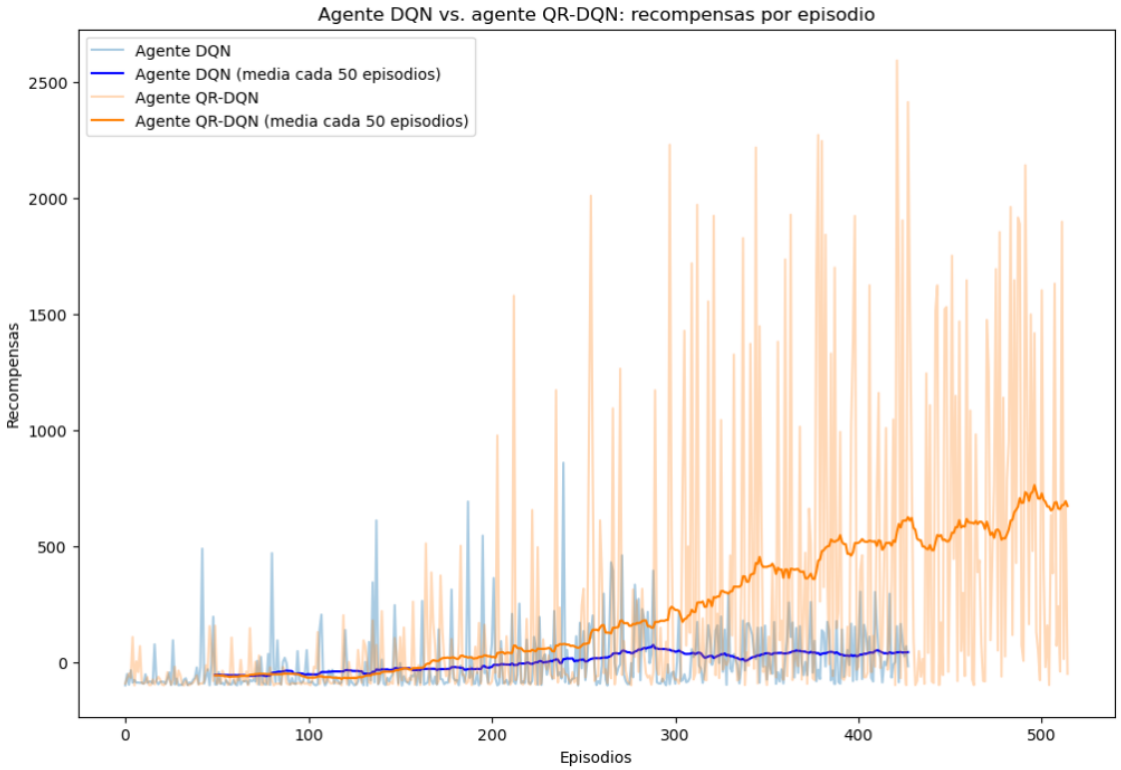

Figura 4.1: *Recompensas por episodio (agentes DQN y QR-DQN)*

Fuente: *Elaboración propia*

Se observa una variabilidad considerable en las recompensas obtenidas en ambos modelos.

Sin embargo, se puede apreciar una tendencia al alza claramente definida en el caso del agente QR-DQN, mientras que las recompensas del agente DQN parecen alcanzar un punto de estabilización después de cierto número de episodios. Este fenómeno se percibe más claramente en las líneas que representan las recompensas medias obtenidas cada 50 episodios en ambos agentes.

Ante estos resultados, es razonable preguntarse acerca del impacto real del extractor de características de la arquitectura de red presentada en la Figura 3.5. Para abordar esta cuestión y validar dicha arquitectura, se ha llevado a cabo un entrenamiento adicional desactivando este componente para el agente que obtiene las recompensas más altas, limitándolo a los datos escalares de distancia recorrida y profundidad de escena.

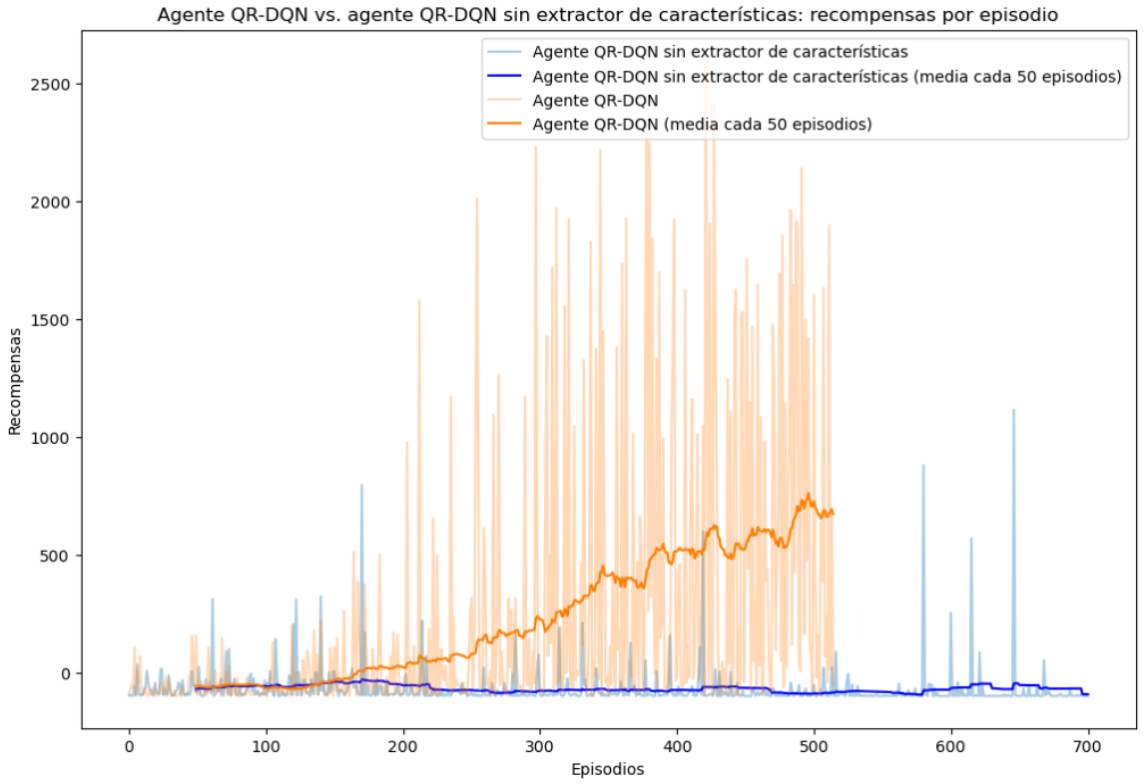

Figura 4.2: *Recompensas por episodio (agente QR-DQN con y sin extractor de características)*

Fuente: *Elaboración propia*

Los resultados, presentados en la Figura 4.2, son claros. Sin el extractor, se genera una cantidad muy elevada de episodios con recompensas predominantemente negativas, apuntando a un rendimiento deficiente en la tarea de navegación autónoma. Por el contrario, con el extractor de características, el agente parece obtener una comprensión más detallada del entorno. Esto corrobora la importancia de este componente para la captura y procesamiento efectivo de la información del entorno a partir de las imágenes RGB y mapas de profundidad.

4.3.2. Duración de los episodios

La duración promedio de los episodios es un indicador que puede ayudar a evaluar el proceso de aprendizaje del agente. No obstante, es importante tener presente que un tiempo de vuelo prolongado no siempre es positivo. A veces, este aumento puede deberse a acciones de giro repetitivas, motivo por el que se ha fijado un límite máximo de pasos de tiempo por episodio, tal y como se indicaba en el apartado 4.1.2.

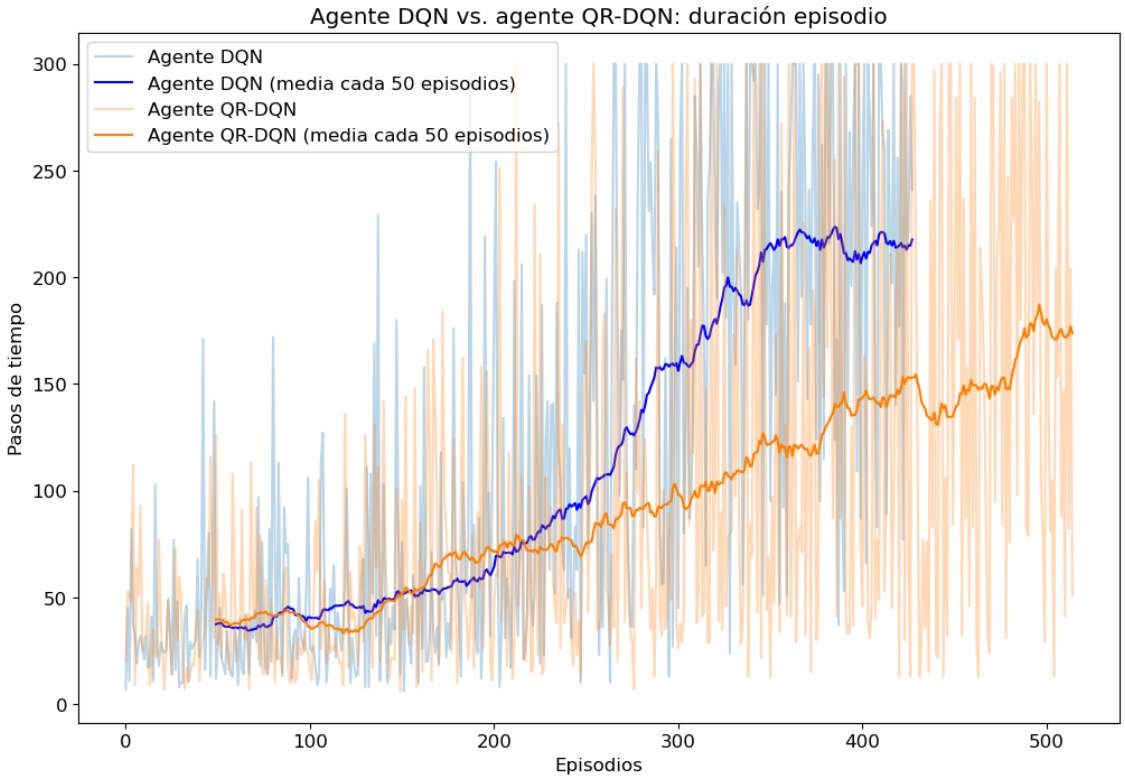

Figura 4.3: *Duración de los episodios en pasos de tiempo (agentes DQN y QR-DQN)*

Fuente: *Elaboración propia*

Observando las dos series de la gráfica de la Figura 4.3, se puede apreciar una tendencia ascendente en el promedio de pasos de tiempo por episodio durante el entrenamiento de ambos agentes, siendo más acentuada en el caso del agente DQN. También se observa una considerable variabilidad en los datos presentados en ambas gráficas. Sin embargo, esta parece ligeramente menos marcada para el agente DQN.

4.3.3. Distancia recorrida por episodio

Otra métrica útil para determinar si el agente está progresando en la resolución del problema planteado es examinar la distancia recorrida durante cada episodio, es decir, si el dron está

efectivamente en movimiento.

Una forma de aproximarse a esta evaluación es analizar el porcentaje de acciones de avance que se realiza en cada episodio. Como muestra la gráfica de la Figura 4.4, en el caso del agente DQN, este porcentaje tiende a disminuir a medida que avanza el proceso de entrenamiento, llegando a valores promedio inferiores al 10 % de las acciones ejecutadas por episodio. Este patrón sugiere que el agente muestra una preferencia por ejecutar acciones de giro, a pesar de que estas no contribuyen a aumentar la distancia recorrida. Por el contrario, en el caso del agente QR-DQN, el porcentaje de acciones de avance se mantiene relativamente estable, oscilando en un promedio de entre 30 y 40 % de las acciones.

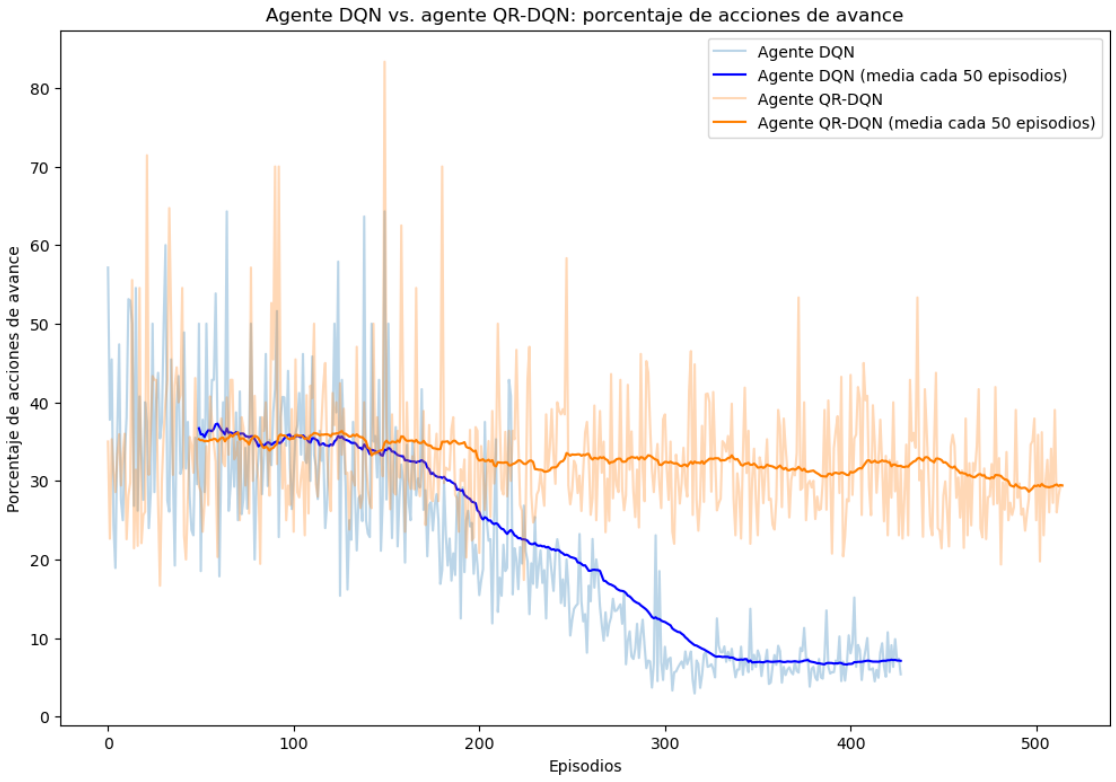

Figura 4.4: *Porcentaje de acciones de avance por episodio (agentes DQN y QR-DQN)*

Fuente: *Elaboración propia*

La gráfica de la Figura 4.5 proporciona quizás una perspectiva más precisa de esta métrica, al representar la distancia recorrida en cada episodio para ambos agentes. Si bien las dos series muestran una tendencia ascendente en los valores promedio, esta tendencia es claramente más pronunciada para el agente QR-DQN, tal como lo evidencia la media cada 50 episodios de la distancia recorrida.

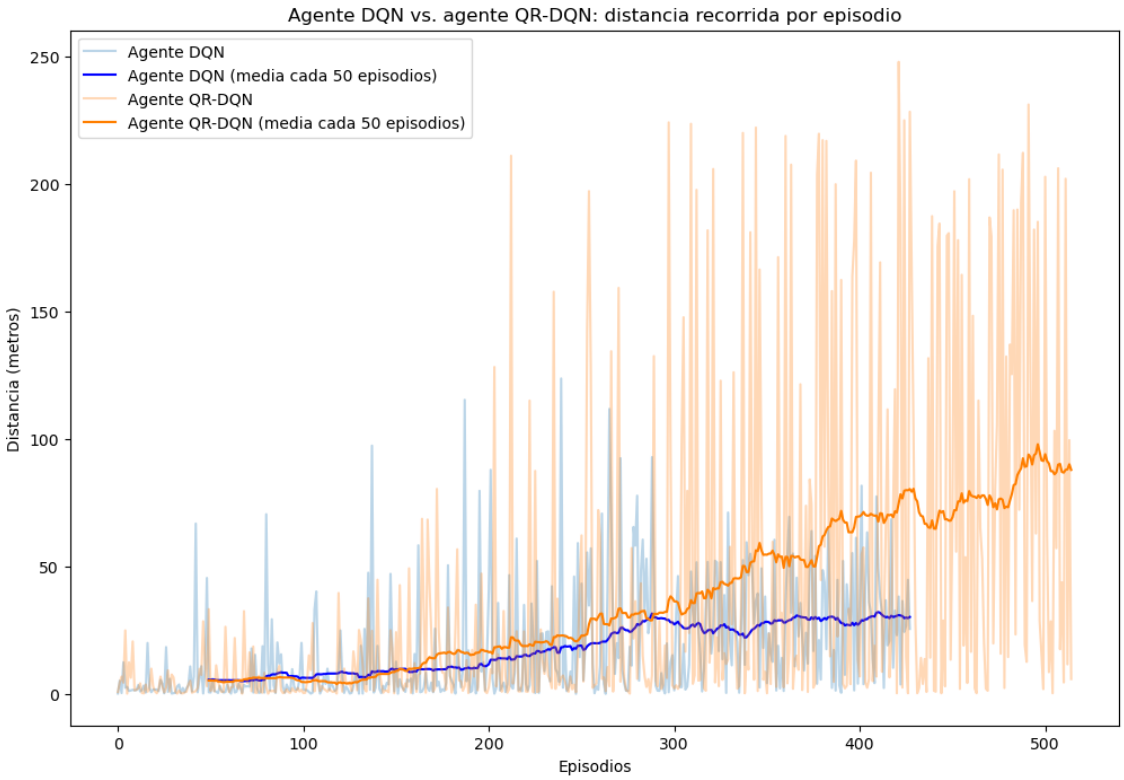

Figura 4.5: *Distancia recorrida por episodio (agentes DQN y QR-DQN).*

Fuente: *Elaboración propia*

4.3.4. Motivo de finalización de los episodios

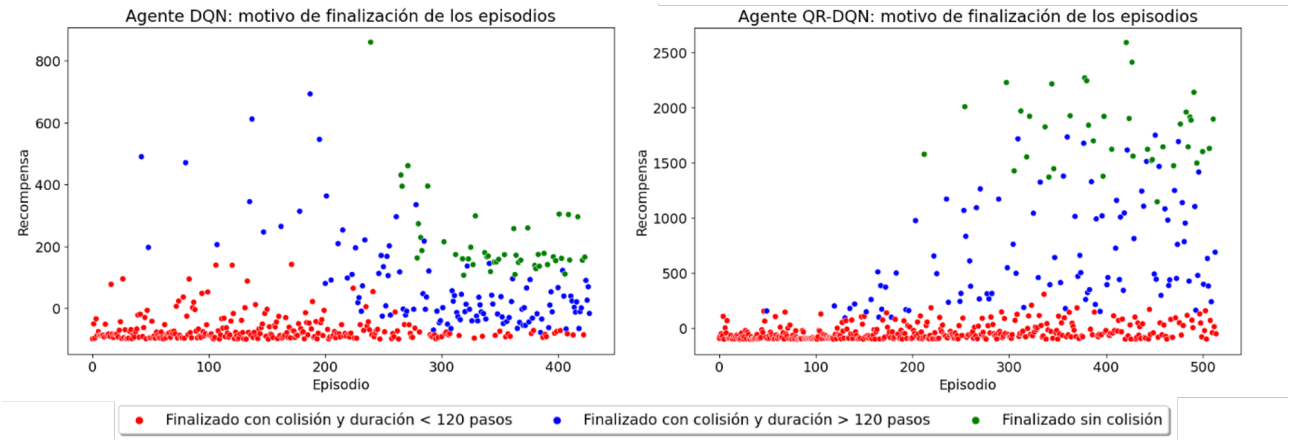

Figura 4.6: *Motivos de finalización de episodios de entrenamiento (agentes DQN y QR-DQN)*

Fuente: *Elaboración propia*

Como se mencionó en el Capítulo 3, existen dos razones por las que pueden finalizar los episodios: la colisión con un obstáculo en el entorno o alcanzar el límite máximo de pasos de tiempo permitido. Los gráficos de dispersión de la Figura 4.6 ofrecen una visión detallada de estos factores de terminación. En dichas gráficas, los episodios se diferencian mediante colores de acuerdo con su motivo de finalización, y se posicionan a diferentes alturas en función de la recompensa final obtenida.

Las gráficas revelan que en ambos casos, los episodios que concluyen sin ninguna colisión aparecen en general en la segunda mitad del proceso de entrenamiento. Se puede notar también que los valores de recompensa son notablemente más altos para los episodios del agente QR-DQN. Esto queda reflejado en la mayor escala vertical de su gráfica, cuyo rango es más de tres veces superior al de la gráfica del agente DQN.

4.4. Selección del agente y entrenamiento final

A partir de los indicadores anteriores, se puede concluir que el agente QR-DQN demuestra una capacidad de aprendizaje considerablemente más efectiva que la del agente DQN. Esto justifica su selección para la etapa final de entrenamiento, que consiste en llevar a cabo este proceso desde distintas ubicaciones en el escenario virtual.

Figura 4.7: *Ubicaciones para el despegue del dron*
Fuente: *Gabro Media, Unreal Engine Marketplace [31] y elaboración propia*

La intención de esta estrategia es fomentar el aprendizaje de políticas óptimas ante una mayor variedad de situaciones. Para ello, se han identificado tres puntos de despegue para el entrenamiento del agente en el entorno virtual, como se muestra en la Figura 4.7. Cada ubicación presenta una distribución de obstáculos y objetos diferente. De estos, el punto identificado como

"comedor" representa el despegue más complejo, dado que el dron se encuentra a una distancia mínima de varios obstáculos de la estancia, como paredes y muebles. Los otros dos puntos, son áreas relativamente despejadas que permiten un despegue y movimientos iniciales más seguros.

Los tres entrenamientos se han realizado de manera secuencial, comenzando en la ubicación denominada "salón", que es la misma que se utilizó en la comparativa de las dos variantes del algoritmo DQN del apartado anterior. Posteriormente, se ha llevado a cabo el segundo entrenamiento desde la ubicación "cocina", utilizando como base los pesos adquiridos al finalizar el primer entrenamiento. Finalmente, el último entrenamiento se ha realizado en el punto de despegue "comedor", partiendo de los pesos obtenidos en el entrenamiento inmediatamente anterior. Es importante destacar que para el segundo y tercer entrenamiento se ha utilizado el mismo valor inicial de épsilon y el mismo porcentaje de periodo de exploración que en el entrenamiento inicial. El motivo de esta decisión es el bajo rendimiento observado al utilizar valores más bajos de épsilon y periodos de exploración más cortos, que pretendían lograr una mayor preservación de los pesos aprendidos en el entrenamiento anterior.

El proceso consiste, por tanto, en iniciar el entrenamiento del agente en una ubicación de menor complejidad en términos de obstáculos, para luego usar los pesos aprendidos como punto de partida en un escenario más complicado. La finalidad de esta estrategia de aprendizaje es doble. Por un lado, se busca reducir el tiempo de entrenamiento al aprovechar los conocimientos adquiridos en ubicaciones anteriores del entorno. Por otro lado, se espera mejorar el rendimiento de los agentes entrenados, ya que se está proporcionando una base de conocimiento más sólida desde el inicio de cada fase de entrenamiento.

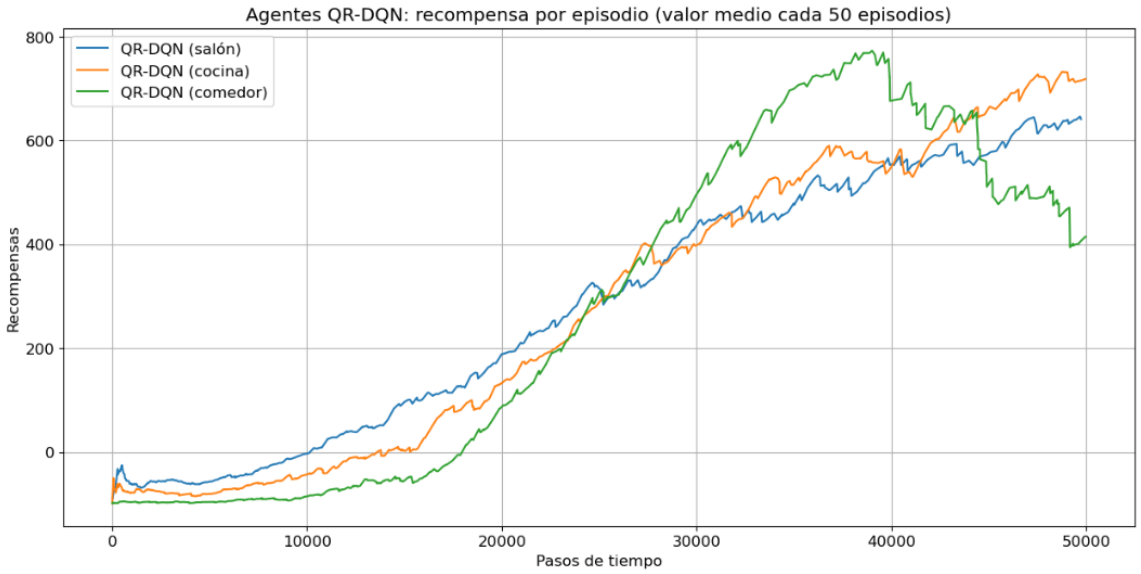

Figura 4.8: *Recompensas por episodio (agentes QR-DQN)*

Fuente: *Elaboración propia*

La Figura 4.8 muestra la evolución de las recompensas obtenidas durante los entrenamientos en las tres ubicaciones. En esta ocasión, se ha optado por utilizar los pasos de tiempo del entrenamiento como referencia, y no los episodios, ya que el número de estos varía mucho entre los tres entrenamientos, complicando la comparación visual de los datos.

La gráfica muestra una evolución similar en las recompensas promedio por episodio en el caso de los entrenamientos realizados desde las ubicaciones "salón" y "cocina". No obstante, en el punto de despegue "comedor", se registran valores significativamente superiores entre los pasos de tiempo 30.000 y 40.000 del proceso de entrenamiento. Más allá de esta cifra, las recompensas empiezan a descender de forma significativa, situándose por debajo de las logradas en los otros dos entrenamientos. Este comportamiento sugiere la posibilidad de un sobreajuste en el entrenamiento del agente, que podría estar afectando a su capacidad de aprendizaje.

La Figura 4.9 muestra la progresión de pasos de tiempo por episodio durante el entrenamiento. La tendencia para las ubicaciones "salón" y "cocina" es comparable a la observada en la gráfica de recompensas. Sin embargo, en la ubicación "comedor", se registran valores inferiores durante todo el proceso, manifestándose una disminución notable en la duración de los episodios a partir del mismo punto de inflexión identificado en la gráfica de recompensas.

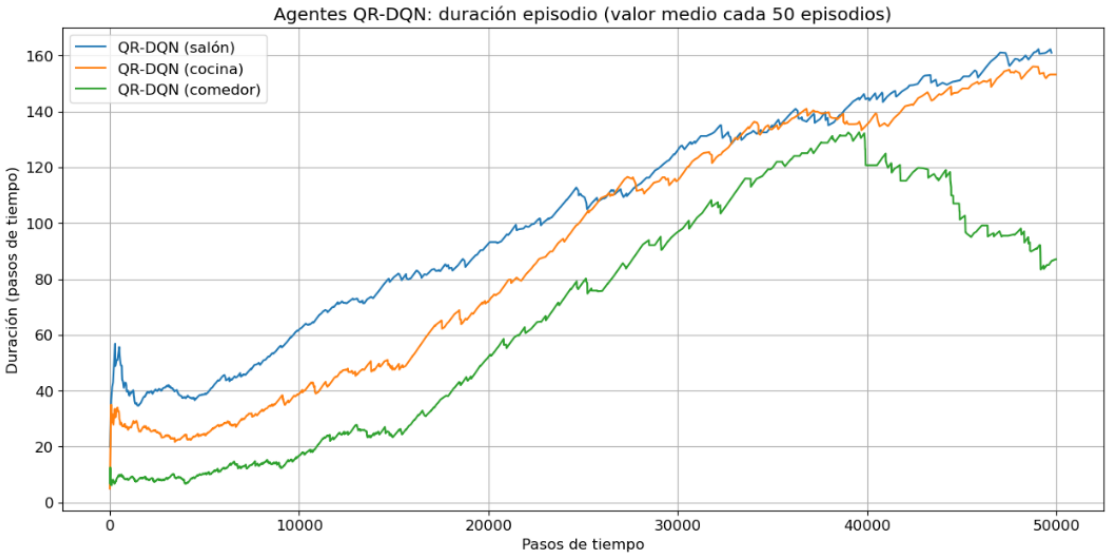

Figura 4.9: *Duración de los episodios en pasos de tiempo (agentes QR-DQN)*

Fuente: *Elaboración propia*

Para ilustrar las diferencias en el número de episodios ejecutados, la Figura 4.10 muestra la evolución de la distancia recorrida en los tres entrenamientos. En la gráfica es evidente que la cantidad de episodios ejecutados desde la ubicación "comedor" supera a las demás, probablemente debido a la mayor complejidad de este escenario. Además, el patrón en la distancia

recorrida es similar al observado en las gráficas previas, mostrando un notable descenso final.

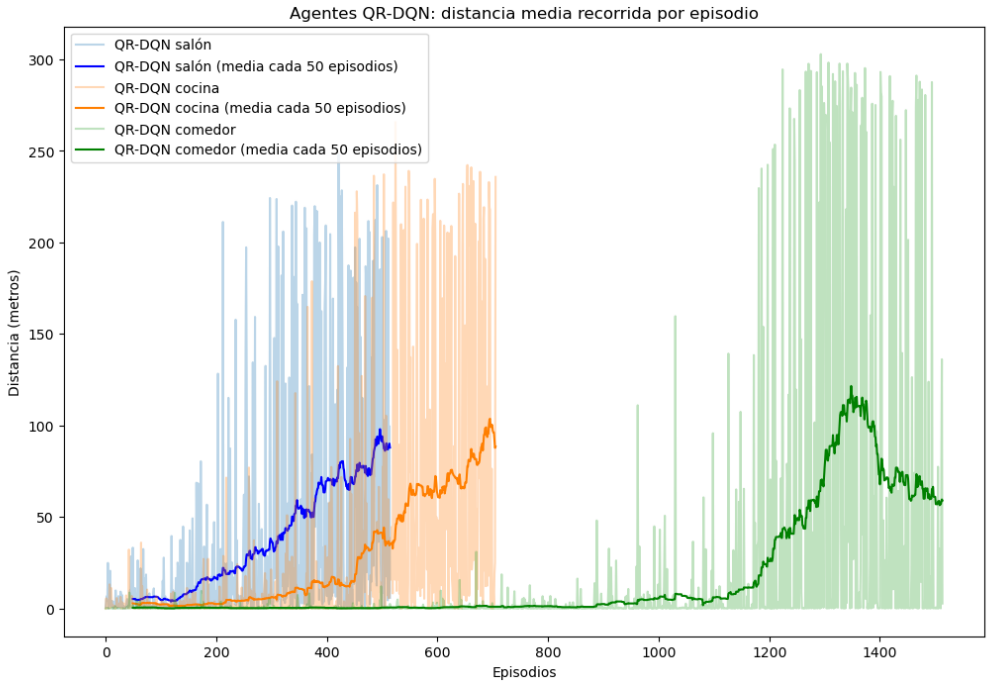

Figura 4.10: *Distancia recorrida por episodio (agentes QR-DQN)*

Fuente: *Elaboración propia*

La Figura 4.11 muestra la distribución de los episodios de entrenamiento en relación con el motivo de finalización y la recompensa obtenida. En las gráficas se aprecian similitudes considerables en la evolución de las proporciones de los distintos tipos de episodios a lo largo del tiempo en estas tres gráficas, lo que parece el indicio de un patrón de aprendizaje similar.

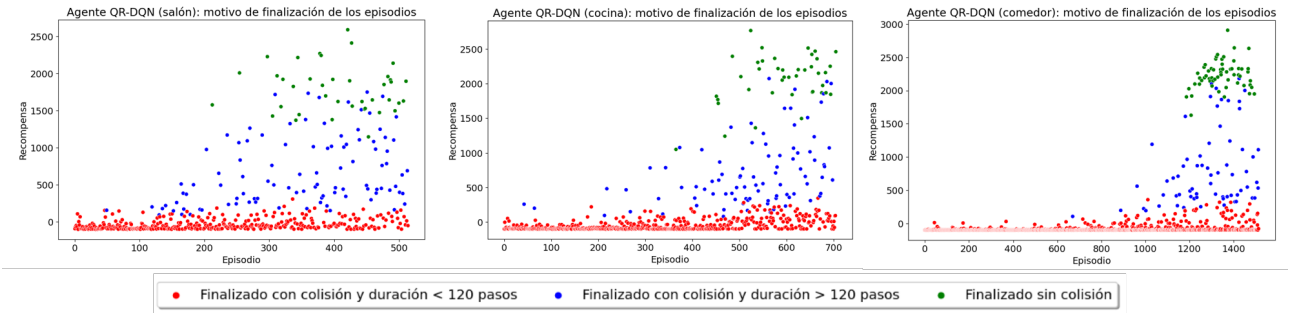

Figura 4.11: *Motivos de finalización de episodios de entrenamiento (agentes QR-DQN)*

Fuente: *Elaboración propia*

4.5. Resultados obtenidos

Para determinar la eficacia y el rendimiento de los agentes entrenados, se ha desarrollado un proceso de evaluación que abarca la ejecución de múltiples episodios de prueba en las tres localizaciones del entorno virtual mencionadas anteriormente, además de una prueba con un dron real. El objetivo de estos experimentos es descubrir qué modelo proporciona los resultados más satisfactorios bajo diversas condiciones de prueba.

4.5.1. Resultados en el entorno virtual

En las siguientes secciones, se muestran los resultados obtenidos por los agentes QR-DQN durante los episodios de prueba en el entorno virtual. Estos resultados abarcan tres conjuntos de 50 episodios, cada uno ejecutado en las diferentes áreas de despegue identificadas anteriormente: "salón", "cocina" y "comedor". Para cada agente, se evalúan diversas métricas, incluyendo la recompensa obtenida, la duración de los vuelos, la distancia recorrida, y el motivo de finalización de los episodios.

4.5.1.1. Recompensas obtenidas

La Figura 4.12 muestra una tabla de histogramas, los cuales representan la distribución de las recompensas que cada agente obtuvo al llevar a cabo los 50 episodios de prueba desde cada una de las áreas de despegue. En esta tabla, cada columna se corresponde con un agente específico, entrenado desde una ubicación concreta tal como se describió en el apartado 4.4. Por otro lado, cada fila muestra los gráficos de distribución de recompensas obtenidas desde una área de despegue en particular.

Al analizar detenidamente estos datos, el agente "QR-DQN (cocina)" muestra el mejor rendimiento general, a pesar de tener una proporción considerable de episodios con recompensas bajas, especialmente en el escenario más complejo. Esto sugiere que este modelo tiene una mayor capacidad de aprendizaje de políticas efectivas en comparación con los otros agentes. Por otro lado, el agente "QR-DQN (comedor)" ofrece el rendimiento menos satisfactorio, lo que indica que puede tener más dificultades que los otros para adaptarse a entornos nuevos o desconocidos. Además, el agente "QR-DQN (salón)", que fue el primero que se obtuvo en el proceso de entrenamiento, muestra un rendimiento relativamente equilibrado, pero generalmente inferior en comparación con el agente entrenado en el escenario "cocina", obteniendo una menor proporción de recompensas altas.

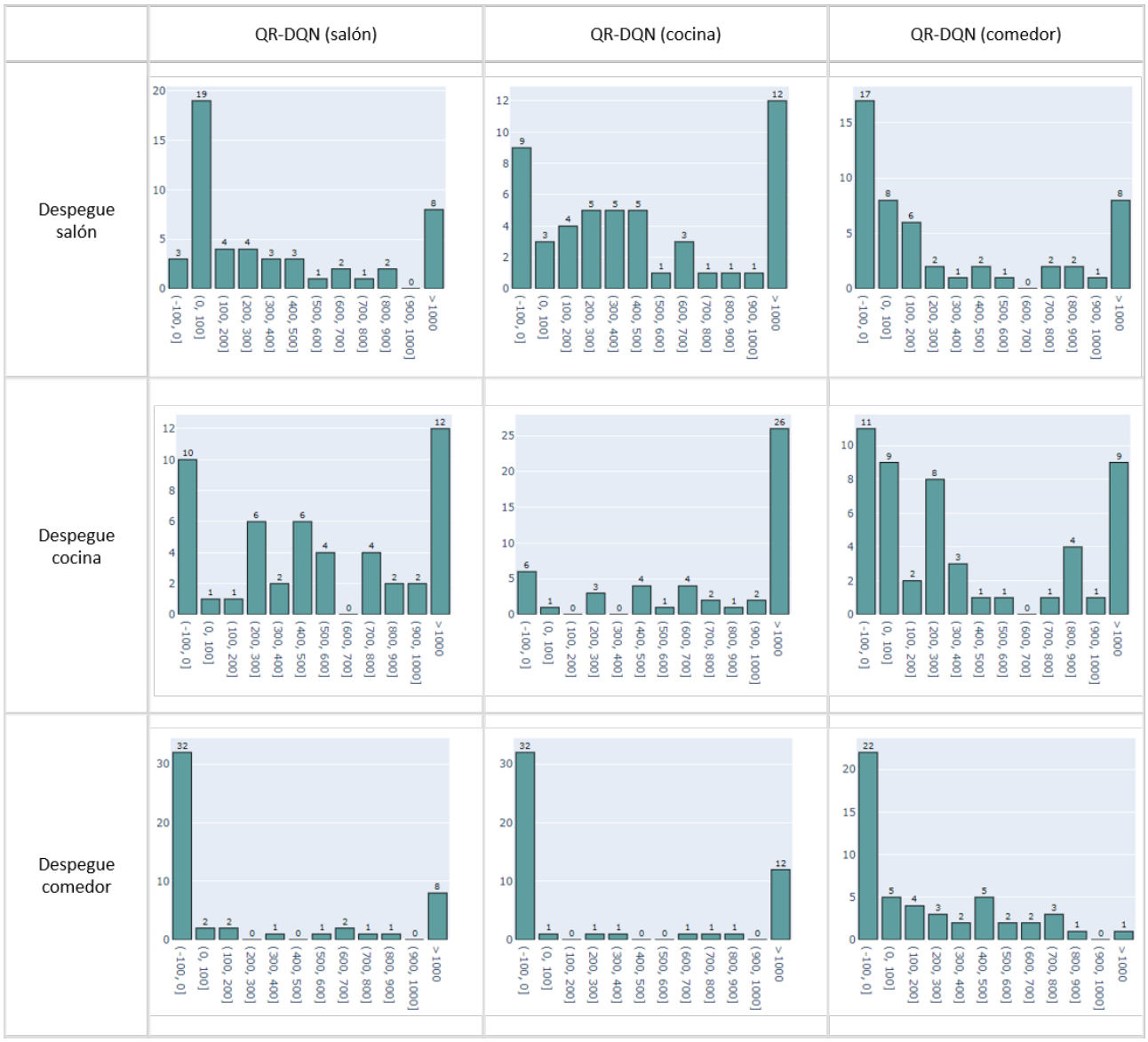

Figura 4.12: *Recompensas de los episodios de prueba (entorno virtual)*

Fuente: *Elaboración propia*

4.5.1.2. Duración de los vuelos

A continuación, se presentan los histogramas que muestran la distribución de la duración de los vuelos de los episodios ejecutados por los agentes en las distintas zonas de despegue, la organización de estos gráficos es similar a la del apartado anterior.

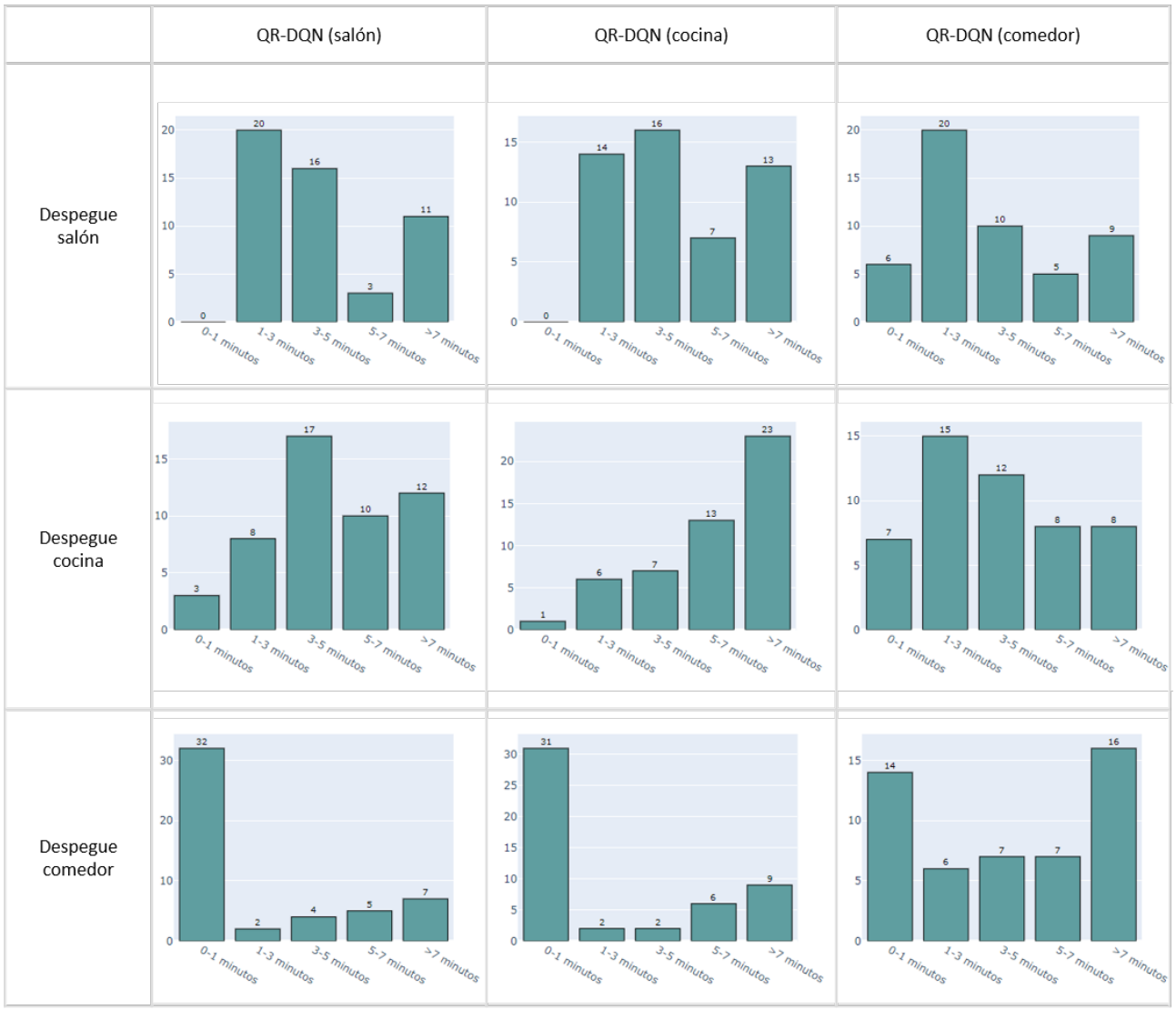

Figura 4.13: *Tiempo de vuelo de los episodios de prueba (entorno virtual)*

Fuente: *Elaboración propia*

Al igual que ocurría con las recompensas, el agente "QR-DQN (cocina)" muestra en general los mejores resultados al ofrecer una mayor proporción de vuelos con duración superior a 7 minutos en comparación con los otros agentes. Esta tendencia sugiere que este agente tiene una mayor capacidad para evitar colisiones en comparación con los otros dos. Por otro lado, el agente "QR-DQN (comedor)" presenta la mejor proporción de vuelos prolongados en el escenario en el que fue entrenado, superando a los otros dos agentes, lo que podría indicar cierto grado de sobreajuste en su aprendizaje, como sugería la gráfica de la Figura 4.8.

En general, los datos de duración de vuelo son coherentes con los histogramas de recompensas obtenidas por los agentes. No se observa una tendencia excesiva hacia vuelos extremadamente cortos o largos. Es importante tener en cuenta que la duración del vuelo se puede prolongar

por acciones repetitivas de giro o movimientos de avance hacia zonas poco despejadas, lo que provoca recompensas bajas según la lógica de la función de recompensa del Algoritmo 1. Esto puede explicar por qué el agente "QR-DQN (comedor)" registra algunos vuelos relativamente largos en el escenario más complejo, a pesar de obtener recompensas generalmente bajas según el histograma de la Figura 4.12.

4.5.1.3. Distancia recorrida

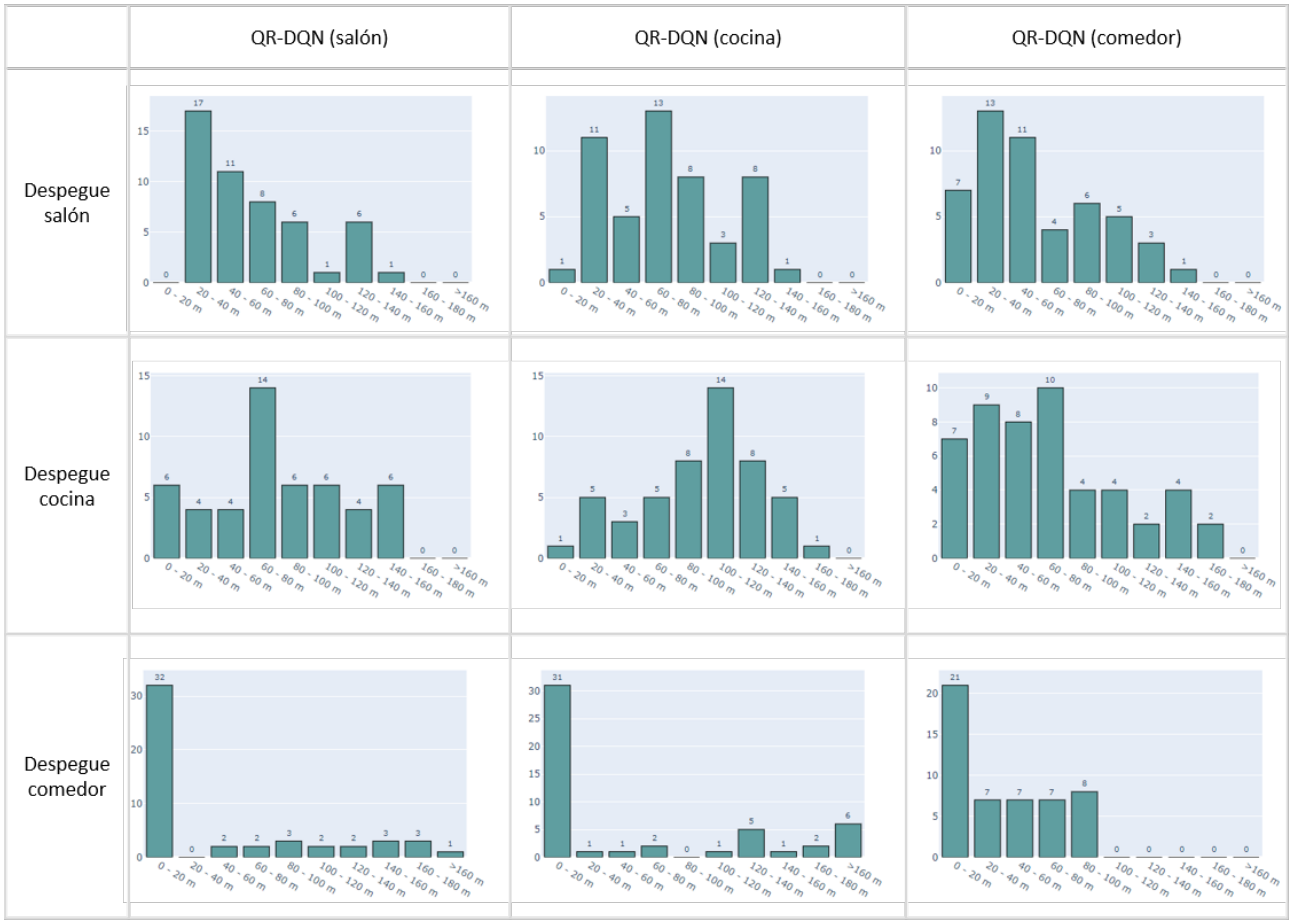

Figura 4.14: *Distancia recorrida en los episodios de prueba (entorno virtual)*

Fuente: *Elaboración propia*

La serie de histogramas de la Figura 4.14 muestra la distancia recorrida por cada agente durante los episodios de prueba desde cada una de las áreas de despegue. Nuevamente, es el agente "QR-DQN (cocina)" el que muestra, en general, un mejor desempeño en términos de lograr las mayores distancias recorridas, lo que sugiere una mayor capacidad para explorar el entorno. El agente "QR-DQN (comedor)" también obtiene valores razonablemente satisfactorios, pero su rendimiento es inferior al del agente "QR-DQN (cocina)", incluso en el escenario en

el que fue entrenado. El agente "QR-DQN (salón)"tiene una distribución de distancia recorrida más equilibrada, pero con una proporción menor de vuelos de larga distancia que los otros dos agentes.

4.5.1.4. Motivo de finalización del vuelo

Finalmente, la Figura 4.15 muestra una serie de gráficos de sectores que muestran los motivos de finalización de los episodios para cada agente y área de despegue. Estos gráficos confirman de nuevo el mejor desempeño del agente "QR-DQN (cocina)", mostrando en todos los escenarios el mayor porcentaje de episodios finalizados sin que se produzcan colisiones con obstáculos del entorno.

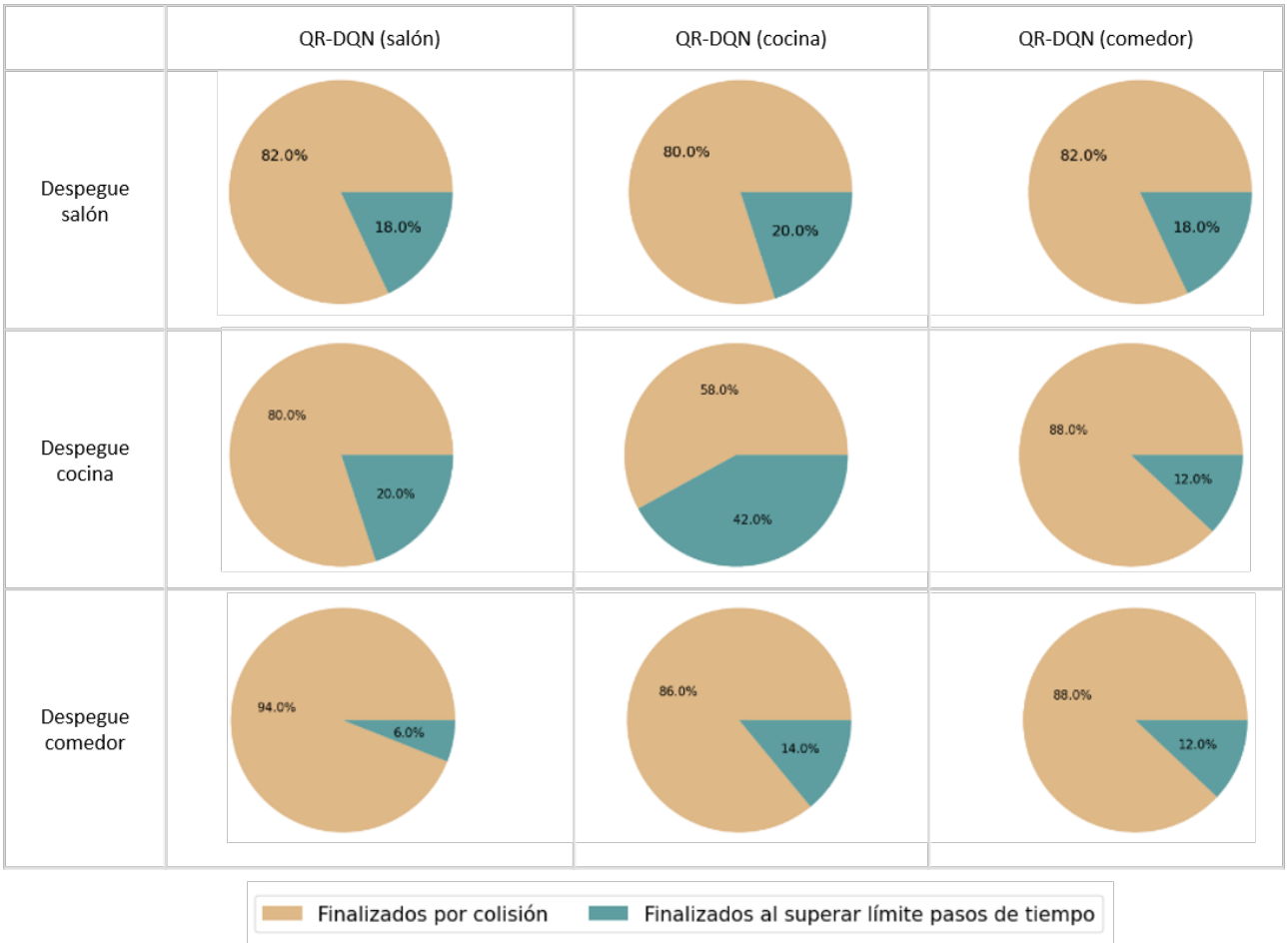

Figura 4.15: *Motivo de finalización de los episodios de prueba (entorno virtual)*

Fuente: *Elaboración propia*

4.5.2. Prueba con el dron físico

Los histogramas y gráficos anteriores revelan que el lugar de entrenamiento del modelo tiene un impacto significativo en su rendimiento. En la mayoría de los casos, se observa un mejor desempeño en el entorno donde los agentes han sido entrenados, lo que sugiere que podrían tener dificultades para aplicar su aprendizaje a nuevos escenarios.

En este apartado, el objetivo es determinar si, a pesar de todo, el conocimiento adquirido permite realizar vuelos razonablemente satisfactorios en un entorno real. Para llevar a cabo esta evaluación, se ha utilizado un dron doméstico de tamaño reducido: el modelo Ryzer Tello, fabricado por DJI, como se muestra en la Figura 4.16. Este aparato ofrece la ventaja de ser compatible con el lenguaje de programación Python, lo que permite enviar comandos a través de red WiFi y recibir datos, como las imágenes capturadas por su cámara frontal, a partir de las cuales se generarán los mapas de profundidad. Aunque su autonomía de vuelo es limitada (aproximadamente 10 minutos), es suficiente para realizar experimentos sencillos y evaluar la capacidad del modelo entrenado para adaptarse a un entorno físico.

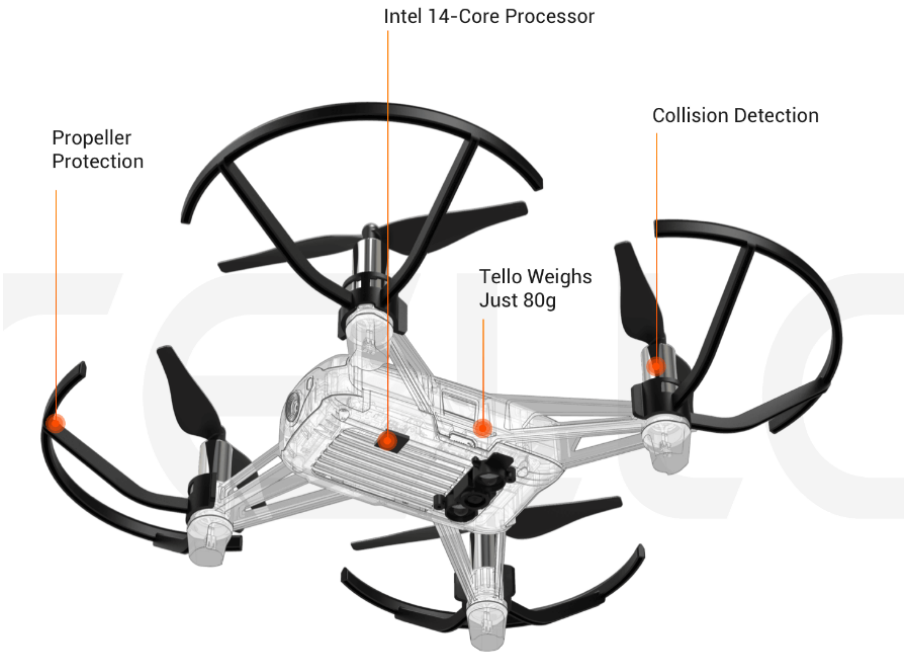

Figura 4.16: *Dron DJI Ryzer Tello*

Fuente: *DJI - Ryzer Robotics [42]*

Para realizar los vuelos de prueba se ha elegido una estancia de una vivienda con una cantidad limitada de obstáculos, que ofrece las condiciones de iluminación adecuadas para

permitir que el dron opere con cierta estabilidad. Esto se debe a que la IMU del dron utilizado es bastante sensible a los cambios en la luz ambiental, pudiendo afectar estos a la precisión de sus sensores y, en última instancia, a la estabilidad del propio dron. La Figura 4.17 muestra una vista aérea de la estancia en la que se han desarrollado los vuelos.

Figura 4.17: *Vista aérea de la estancia en la que se han realizado los vuelos del dron real*

Fuente: *Elaboración propia*

Con el objetivo de evitar daños significativos al dron, solo se han realizado tres vuelos, todos ellos utilizando el agente "QR-DQN (cocina)", que demostró el mejor rendimiento en los episodios de prueba en el entorno virtual. La Figura 4.18 muestra la trayectoria seguida por el dron durante dichos vuelos. Cada vuelo está representado en la imagen mediante colores diferentes, utilizando flechas simples para representar una acción de avance y flechas cruzadas para indicar la ejecución de una o varias acciones de giro.

Figura 4.18: *Trayectoria de los vuelos de prueba en el entorno real*

Fuente: *Elaboración propia*

Llama la atención cómo la trayectoria inicial es muy similar en los tres casos, dirigiendo al dron hacia una zona con varios obstáculos. De los tres vuelos, el único que no logra evitarlos es el representado en color naranja, mientras que los otros dos logran sortearlos al modificar su trayectoria y desplazarse a una zona más despejada de la estancia. El vuelo de color azul logra aterrizar de manera segura después de aproximadamente 4 minutos de vuelo, mientras que el de color verde, aunque tiene una duración y desarrollo parecido, experimenta problemas de estabilidad provocados por la IMU del dron. Esto resulta en una deriva significativa en lugar de permanecer estacionario mientras espera el siguiente comando, lo que causa una leve colisión que supone la finalización prematura del vuelo al activarse el aterrizaje automático.

A continuación se muestra una tabla que resume los datos más relevantes de cada vuelo realizado por el dron:

Vuelo	Acciones avance	Acciones giro	Distancia (m)	Duración (min)
Vuelo 1 (azul)	6	22	3	4:26
Vuelo 2 (naranja)	6	17	2.5	3:02
Vuelo 3 (verde)	8	20	3.5	3:44

Cuadro 4.1: *Detalle de los vuelos del dron real*

Fuente: *Elaboración propia*

Considerando los resultados obtenidos en el entorno real, se puede concluir que existe cierta transferencia de conocimiento, como evidencian los vuelos que logran evitar los obstáculos. Se observa que, al acercarse a ellos, el dron comienza a girar en busca de un área despejada hacia la cual desplazarse. Sin embargo, es importante destacar que el dron seleccionado es altamente sensible a factores externos y no ofrece la misma estabilidad que en el entorno virtual. Además, se ha notado una notable demora entre la emisión de comandos y su ejecución efectiva en el aparato, posiblemente debido a las limitaciones de procesamiento del equipo portátil utilizado en estos experimentos. Esto ocasiona un consumo rápido de la batería, reduciendo drásticamente la duración de los vuelos y por tanto la posibilidad de ejecutar un mayor número de acciones.

Capítulo 5

Conclusiones

En este último capítulo, se resumen los hallazgos y aprendizajes obtenidos en este trabajo sobre la navegación autónoma de drones utilizando técnicas de aprendizaje por refuerzo profundo. El objetivo de los siguientes apartados es proporcionar una reflexión sobre los desafíos enfrentados, limitaciones del enfoque adoptado y posibles áreas de mejora e investigación.

5.1. Resumen del modelo desarrollado

Este trabajo ha presentado un modelo de aprendizaje por refuerzo profundo para la navegación autónoma de drones. El modelo combina técnicas de visión artificial con algoritmos DRL ampliamente reconocidos y utilizados, y ha sido diseñado para ser aplicado tanto en entornos simulados como reales.

El modelo propuesto utiliza observaciones como la imagen RGB capturada por la cámara del dron, un mapa de profundidad calculado a partir de esta imagen y la distancia total recorrida por el dron. Además, se implementa una estrategia basada en regiones de interés (ROI) en el mapa de profundidad para estimar la profundidad de la escena, lo cual resulta fundamental para generar recompensas y tomar decisiones por parte del agente.

En el entorno definido, se ha simplificado el espacio de acciones del agente para que consista en avanzar a una velocidad constante, o girar a la izquierda o a la derecha sobre su eje sin cambiar su posición. La función de recompensa diseñada tiene como objetivo priorizar la evasión de obstáculos y el avance del dron, fomentando así que el agente maximice la distancia recorrida.

Para gestionar los espacios de observación de alta dimensionalidad de este entorno, se ha implementado el algoritmo DQN, en concreto se han empleado dos variantes que hacen uso de políticas *épsilon-greedy*, búfer de repetición de experiencias y red objetivo. Una de estas variantes es el algoritmo *Quantile Regression* DQN (QR-DQN), el cual estima una distribución de posibles valores en lugar de un único valor óptimo para cada par estado-acción. Esta

característica es especialmente útil cuando una misma acción puede generar recompensas muy variables, como ocurre con la acción de avance del dron.

El agente diseñado utiliza una red neuronal convolucional llamada DeepLabV3, que ha sido previamente entrenada para extraer características de alto nivel de las imágenes RGB y el mapa de características. Estas características se combinan con los valores de la distancia recorrida y la profundidad estimada de la escena para el procesamiento del agente.

5.2. Desafíos y limitaciones

- El diseño del entorno ha planteado varios desafíos para capturar la información necesaria y lograr un aprendizaje efectivo del agente. Uno de los más significativos ha sido el desarrollo de una función de recompensa que permita al dron evitar colisiones y priorizar su avance. En particular, el ajuste del factor de ponderación utilizado para definir la importancia de la distancia recorrida y la penalización por colisión ha sido especialmente complicado, y es probable que requiera ajustes adicionales para lograr un equilibrio óptimo y mejorar el rendimiento del agente.

- La implementación de una arquitectura adecuada para el agente y la determinación de la necesidad y configuración de sus componentes han sido otro desafío importante. Si bien el extractor de características utilizado ha demostrado ser fundamental para la detección de obstáculos, también plantea un desafío significativo debido a su complejidad computacional. Esto limita las posibilidades de implementación en aplicaciones en tiempo real que requieren sistemas con capacidad de procesamiento reducida, como los drones.

- Las variantes del algoritmo DQN seleccionadas son especialmente adecuadas para manejar grandes espacios de estados. Sin embargo, requieren de un gran número de muestras de entrenamiento y presentan una alta sensibilidad a los hiperparámetros. La optimización de estos es un proceso laborioso que no ha podido llevarse a cabo exhaustivamente debido a las limitaciones de tiempo disponible, lo cual también limitó la posibilidad de realizar entrenamientos más extensos. Además, ajustar los hiperparámetros asociados al entorno en sí también ha sido un desafío, ya que tienen un impacto decisivo en el comportamiento del agente, afectando el equilibrio de recompensas y los valores de información de las observaciones del entorno.

- Aunque se ha observado cierta transferibilidad de conocimientos entre los entornos virtual y real en las pruebas finales, durante el proceso de entrenamiento se ha constatado que la ubicación del agente ha tenido un impacto significativo en el rendimiento del agente, lo que limita su capacidad para generalizar y adaptarse a nuevos entornos.

- En cuanto a las pruebas con el dron físico, se identificaron problemas como demoras en la ejecución de comandos y un rápido agotamiento de la batería, lo cual indica limitaciones en la adaptabilidad a las diferencias existentes entre los entornos virtuales y reales.

5.3. Áreas de mejora e investigación

A pesar de las limitaciones señaladas, el trabajo desarrollado lleva a la conclusión de que es realmente viable el desarrollo de un sistema de navegación autónoma que integra algoritmos de aprendizaje por refuerzo profundo y técnicas de visión artificial en drones de bajo coste. Además, los desafíos identificados pueden servir como guía para futuras investigaciones que busquen abordar de manera más efectiva los problemas encontrados y mejorar el rendimiento y la capacidad de adaptación del agente en diferentes escenarios. Algunas líneas de trabajo sugeridas son las siguientes:

- Revisar y analizar varios aspectos del entorno es fundamental para avanzar hacia una solución más satisfactoria. Por ejemplo, se puede considerar la posibilidad de ajustar el espacio de acciones, evaluando si es conveniente modificar las acciones de giro o avance para lograr movimientos más precisos y permitir que el dron aprenda combinaciones de acciones más refinadas. Además, se debe realizar pruebas detalladas para evaluar la efectividad de la función de recompensa y lograr un equilibrio óptimo que mejore el rendimiento del agente.

- En cuanto a los ajustes de la arquitectura del agente, se sugiere explorar diferentes extractores de características además de DeepLabV3, estudiar el impacto de la resolución de las imágenes procesadas y el número de mapas de características extraídas. También se deben revisar la disposición y número de neuronas de las capas de las subredes y los mecanismos para mejorar la capacidad de evitar el sobreajuste.

- Es necesario realizar una revisión sistemática y exhaustiva de los hiperparámetros tanto del algoritmo como del entorno, explorando diversas combinaciones y aprovechando herramientas automatizadas de ajuste fino. Dado el carácter complejo del entorno y la tarea propuesta, se recomienda aumentar significativamente el número de pasos de tiempo de entrenamiento. Además, es fundamental estudiar los valores iniciales de épsilon para lograr un equilibrio adecuado y evitar el sobreajuste, manteniendo el conocimiento previo si se emplea un enfoque de entrenamiento progresivo como el propuesto en este trabajo.

- Otra dirección sugerida es explorar otras variantes del algoritmo DQN, e incluso considerar la posibilidad de experimentar con opciones como TD3 o PPO, aunque esto requiera

replantear el problema para trabajar con espacios de acciones continuas.

- En relación al entorno virtual empleado, se propone entrenar en escenarios virtuales con diferentes disposiciones de objetos y obstáculos, colores y condiciones de iluminación, con el objetivo de mejorar la capacidad de generalización del agente.

- Para avanzar hacia la aplicación en el entorno real, se plantea la posibilidad de entrenar en un entorno físico utilizando un dron con mayor autonomía, capacidad de procesamiento y protección contra impactos. Esto supondría además el procesamiento de las observaciones directamente a bordo, logrando así una notable reducción en el tiempo de respuesta del modelo entrenado en comparación con el método de procesamiento externo que se ha empleado en este trabajo.

Estas líneas de investigación pueden contribuir a superar las limitaciones actuales y mejorar el rendimiento y la capacidad de adaptación del agente en la tarea planteada en este trabajo.

Bibliografía

[1] Elallid et al. "*A comprehensive survey on the application of deep and reinforcement learning approaches in autonomous driving*". King Saud University (2022). https://doi.org/10.1016/j.jksuci.2022.03.013, consultado en marzo de 2023.

[2] A.T. Azar et al. "*Drone Deep Reinforcement Learning: A Review*". Electronics MDPI (2021). https://doi.org/10.3390/electronics10090999, consultado en marzo de 2023.

[3] F. AlMahamid y K. Grolinger. "*Autonomous unmanned aerial vehicle navigation using reinforcement learning: a systematic review*". Western University (2022). https://doi.org/10.48550/arXiv.2208.12328, consultado en marzo de 2023.

[4] Briony J. Oates. "*Researching information systems and computing*". SAGE Publications (2006). ISBN: 978-1446235447.

[5] IEEE Xplore. https://ieeexplore.ieee.org/Xplore/home.jsp. IEEE (2023), consultado en marzo de 2023.

[6] arXiv. https://arxiv.org. Cornell University (2023), consultado en marzo de 2023.

[7] Google Scholar. https://scholar.google.com. Google (2023), consultado en marzo de 2023.

[8] Pytorch. https://pytorch.org/. The Linux Foundation (2023), consultado en marzo de 2023.

[9] AirSim. https://microsoft.github.io/AirSim. Microsoft (2022), consultado en marzo de 2023.

[10] Unreal Engine. https://www.unrealengine.com. Epic Games (2023), consultado en marzo de 2023.

[11] Çetin et al. "*Drone navigation and avoidance of obstacles through deep reinforcement learning*". Universitat Politecnica de Catalunya (2019). https:

//upcommons.upc.edu/bitstream/handle/2117/186572/postprint_DOI_10_1109_DASC43569_2019_9081749.pdf, consultado en marzo de 2023.

[12] R. Bellman. "*Dynamic programming*". Princeton University Press (1957). ISBN: 978-0691079516.

[13] R. Sutton y A. Barto. "*Reinforcement learning: an introduction*". MIT Press (1998). ISBN: 978-0262193986.

[14] C. Watkins. "*Learning from delayed rewards*". King's College (1989). https://www.cs.rhul.ac.uk/~chrisw/new_thesis.pdf, consultado en marzo de 2023.

[15] G.A. Rummery y M. Niranjan. "*On-Line Q-Learning using connectionist systems*". University of Cambridge (1994). http://mi.eng.cam.ac.uk/reports/svr-ftp/auto-pdf/rummery_tr166.pdf, consultado en marzo de 2023.

[16] R. Williams. "*Simple statistical gradient-following algorithms for connectionist reinforcement learning*". Kluwer Academic Publishers (1992). https://link.springer.com/article/10.1007/BF00992696, consultado en marzo de 2023.

[17] Pozza et al. "*Quantum reinforcement learning: the maze problem*". Scuola Normale Superiore, Pisa (2021). https://www.researchgate.net/publication/360910430_Quantum_reinforcement_learning_the_maze_problem/link/62918a9c55273755ebbbefa1/download, consultado en marzo de 2023.

[18] Mnih et al. "*Playing Atari with Deep Reinforcement Learning*". DeepMind Technologies (2013). https://arxiv.org/pdf/1312.5602.pdf, consultado en marzo de 2023.

[19] Schulman et al. "*Proximal policy optimization algorithms*". OpenAI (2017). https://arxiv.org/abs/1707.06347, consultado en marzo de 2023.

[20] R. Sutton y A. Barto. "*Reinorcement learning, second edition: an introduction*". MIT Press (2018). ISBN: 978-0262039246.

[21] Lillicrap et al. "*Continuous control with deep reinforcement learning*". Google Deepmind (2019). https://arxiv.org/abs/1509.02971, consultado en marzo de 2023.

[22] H. Durrant-Whyte y T. Bailey. "*Simultaneous localisation and mapping (SLAM): part I The essential algorithms*". IEEE Robotics and Automation Magazine (2006). https://people.eecs.berkeley.edu/~pabbeel/cs287-fa09/readings/Durrant-Whyte_Bailey_SLAM-tutorial-I.pdf, consultado en marzo de 2023.

[23] W. Andrew et al. "*Deep learning for exploration and recovery of uncharted and dynamic targets from UAV-like vision*". International Conference on Intelligent Robots and Systems (2018). https://ieeexplore.ieee.org/document/8593751/, consultado en marzo de 2023.

[24] E. Dhulkefl et al. "*Dijkstra algorithm using UAV pathn planning*". Konya Technical University (2020). https://www.researchgate.net/publication/348035882_DIJKSTRA_ALGORITHM_USING_UAV_PATH_PLANNING/link/5fed9e5f45851553a00a21c0/download, consultado en marzo de 2023.

[25] R. Polvara et al. "*Sim-to-real quadrotor landing via sequential Deep Q-Networks and domain randomization*". MDPI Robotics (2020). https://mdpi-res.com/d_attachment/robotics/robotics-09-00008/article_deploy/robotics-09-00008-v2.pdf?version=1585134356, consultado en marzo de 2023.

[26] Q. Zhang et al. "*IoT Enabled UAV: Network Architecture and Routing Algorithm*". IEEE (2019). https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8599015, consultado en marzo de 2023.

[27] Craig Reynolds. "*Boids, background and update*". https://www.red3d.com/cwr/boids/. Craig Reynolds, consultado en marzo de 2023.

[28] Unity. https://unity.com/es. Unity Technologies (2023), consultado en marzo de 2023.

[29] Tožička et al. "*Application of Deep Reinforcement Learning to UAV Fleet Control*". AISC (2019). https://link.springer.com/chapter/10.1007/978-3-030-01057-7_85, consultado en marzo de 2023.

[30] Mnih et al. "*Human-level control through deep reinforcement learning*". Nature (2015). https://www.nature.com/articles/nature14236, consultado en marzo de 2023.

[31] Gabro Media Environments - Unreal Engine Marketplace. https://www.unrealengine.com/marketplace/en-US/product/apartment-interior. Gabro Media (2017), consultado en abril de 2023.

[32] R. Ranftl et al. "*FTowards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer*". IEEE Transactions on pattern analysis and machine intelligence (2020). https://arxiv.org/pdf/1907.01341.pdf, consultado en abril de 2023.

[33] Laura Ruiz Dern. "*Deep Q-networks*". Universitat Oberta de Catalunya (2021). https://www.uoc.edu, consultado en mayo de 2023.

[34] Dabney et al. "*Distributional Reinforcement Learning with Quantile Regression*". Google DeepMind (2017). https://arxiv.org/abs/1710.10044, consultado en mayo de 2023.

[35] S. Levine et al. "*Learning Hand-Eye Coordination for Robotic Grasping with Deep Learning and Large-Scale Data Collection*". Google (2016). https://arxiv.org/pdf/1603.02199.pdf, consultado en abril de 2023.

[36] A. Marcato et al. "*From Computational Fluid Dynamics to Structure Interpretation via Neural Networks: An Application to Flow and Transport in Porous Media*". American Chemical Society (2022). https://pubs.acs.org/doi/10.1021/acs.iecr.1c04760, consultado en abril de 2023.

[37] The ultimate guide to deeplabv3 with pytorch inference. https://learnopencv.com/deeplabv3-ultimate-guide/. LearnOpenCV (2023), consultado en mayo de 2023.

[38] L. Chen et al. "*Rethinking Atrous Convolution for Semantic Image Segmentation*". Google Inc. (2017). https://arxiv.org/pdf/1706.05587.pdf, consultado en mayo de 2023.

[39] Deeplabv3 model with resnet-101 backbone. https://pytorch.org/hub/pytorch_vision_deeplabv3_resnet101/. The Linux Foundation (2023), consultado en mayo de 2023.

[40] K. He et al. "*Deep Residual Learning for Image Recognition*". Microsoft Research (2015). https://arxiv.org/pdf/1512.03385.pdf, consultado en mayo de 2023.

[41] A. Raffin et al. "*Stable-Baselines3: Reliable Reinforcement Learning Implementations*". Journal of Machine Learning Research (2021). https://jmlr.org/papers/volume22/20-1364/20-1364.pdf, consultado en abril de 2023.

[42] Tello sdk. https://dl-cdn.ryzerobotics.com/downloads/tello/20180910/Tello%20SDK%20Documentation%20EN_1.3.pdf. DJI - Ryze Robotics (2023), consultado en mayo de 2023.

[43] DJITelloPy. https://github.com/damiafuentes/DJITelloPy/. Damian Fuentes - (2023), consultado en mayo de 2023.

[44] R. Ranftl et al. "*Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer*". IEEE (2020). https://arxiv.org/pdf/1907.01341.pdf, consultado en abril de 2023.

[45] Pytorch hub. https://pytorch.org/hub/. The Linux Foundation (2023), consultado en marzo de 2023.

[46] Project airsim. https://www.microsoft.com/en-us/ai/autonomous-systems-project-airsim?activetab=pivot1:primaryr3. Microsoft (2023), consultado en abril de 2023.

[47] A. Anwar y A. Raychowdhury. "*Autonomous Navigation via Deep Reinforcement Learning for Resource Constraint Edge Nodes Using Transfer Learning*". Georgia Institute of Technology (2020). https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=8978577, consultado en mayo de 2023.

[48] D. Kingma y J. Ley. "*Adam: a method ofr stochastic optimization*". ICLR (2015). https://arxiv.org/pdf/1412.6980.pdf, consultado en mayo de 2023.