UNIVERSIDAD SANTO TOMÁS

Arquitectura multi-agente descentralizada para detección de eventos en el ambiente

Realizado por

Daniel Fernando Castaño Ortiz

Proyecto de grado presentado en cumplimiento del requisito para
optar por el titulo de Ingeniero Electrónico

Grupo de Investigación GED (Grupo de Estudio y Desarrollo en Robótica)
Facultad de Ingeniería Electrónica
División de Ingenierías

Febrero de 2026

Arquitectura multi-agente descentralizada para detección de eventos en el ambiente

Realizado por

Daniel Fernando Castaño Ortiz

Proyecto de grado presentado en cumplimiento del requisito para optar por el título de Ingeniero Electrónico

Dirigido por

Ing. David Alejandro Martínez Vásquez PhD.

Grupo de Investigación GED (Grupo de Estudio y Desarrollo en Robótica) Facultad de Ingeniería Electrónica
División de Ingenierías

Febrero de 2026

Autoridades de la Universidad

RECTOR GENERAL
R.P. FRAY ALVARO JOSÉ ARANGO RESTREPO, O.P.

VICERRECTOR ADMINISTRATIVO Y FINANCIERO GENERAL
R.P. FRAY HENDER ALVEIRO RODRIGUEZ PÉREZ, O.P.

VICERRECTOR ACADÉMICO GENERAL
R.P. FRAY MAURICIO ANTONIO CORTES GALLEGO, O.P.

SECRETARIO GENERAL
Dra. INGRID LORENA CAMPOS VARGAS

DECANO DIVISIÓN DE INGENIERÍAS
R.P. FRAY JAVIER ANTONIO HINCAPIÉ ARDILA, O.P.

SECRETARIA DE DIVISIÓN
E.C. LUZ PATRICIA ROCHA CAICEDO

DECANO FACULTAD DE INGENIERÍA ELECTRÓNICA
ING. LEORNADO FABIO YEPES ARBELAEZ.

Nota de aceptación

_____
_____
_____
_____
_____
_____

_____

Firma del autor

_____

Firma del jurado

_____

Firma del jurado

BOGOTÁ D.C. ——— DE 2025

Advertencia

La Universidad Santo Tomás no se hace responsable de las opiniones y conceptos expresados en el trabajo de grado, solo velará por que no se publique nada contrario al dogma ni a la moral católica y porque el trabajo no tenga ataques personales y únicamente se vea el anhelo de buscar la verdad científica.

Capítulo III –Art. 46 del Reglamento de la Universidad Santo Tomás.

Dedicatoria

Dedico este trabajo a mi familia, amigos y director, por su constante apoyo, guía y confianza a lo largo de este proceso.

Agradecimientos

Este trabajo no habría sido posible sin el apoyo, la comprensión y la compañía de muchas personas que, de una u otra forma, hicieron parte de este largo camino.
En primer lugar, quiero mostrar mi más sincero agradecimiento a mi director, por sus continuas orientaciones, por la paciencia que tuvo conmigo y por la confianza que depositó en mi trabajo, su dirección fue la que me hizo superar los momentos de dudas y el que me ayudó a continuar el camino, ofreciendo siempre la motivación necesaria. Gracias por sus enseñanzas, por dedicarse y por exigirme siempre dar lo mejor de mí.
A mi familia, por ser mi base más firme. Gracias por su amor incondicional y por las palabras de ánimo en los momentos complicados, así como por celebrar con entusiasmo cada uno de los logros, el apoyo y la comprensión que me brindaron fueron lo que me hizo llegar hasta donde estoy.
A mis amigos, puesto que con sus compañías, sus consejos y su buen humor hicieron que este recorrido fuera más fácil. Gracias por estar, por escucharme, por motivarme a seguir adelante y por recordarme siempre lo importante que se ha de disfrutar del camino recorrido.
Por último, quiero agradecer a todas las personas que en un sentido u otro han aportado a la realización de este proyecto. A quienes me dedicaron tiempo, ya sea con el conocimiento o con simples palabras de ánimo, todos y cada uno de ellos dejaron su huella en este trabajo y en mi formación como persona y profesional.

Índice general

Resumen

La detección de eventos en el ambiente constituye un reto central en la robótica moderna, especialmente en sistemas que buscan autonomía, cooperación y toma de decisiones distribuidas. Los sistemas multi-agente descentralizados (MAS) surgen como una alternativa eficiente frente a los enfoques centralizados, ya que ofrecer mayor escalabilidad, robustez y tolerancia a fallos. Este trabajo propone el diseño e implementación de una arquitectura multi-agente descentralizada basada en ROS 2 para la detección de eventos en el ambiente, utilizando plataformas móviles TurtleBot3 Burger y aprendizaje por refuerzo profundo.

La investigación se desarrolló con un enfoque experimental y aplicado, estructurado en tres fases: (1) análisis y selección de variables físicas, algoritmos y plataforma robótica; (2) desarrollo de la arquitectura descentralizada, integrando un agente líder entrenado mediante el algoritmo Deep Q-Learning (DQN), junto con agentes seguidores controlados con el algoritmo *Follower* de ROBOTIS; y (3) validación del sistema en el simulador Gazebo, mediante el uso del middleware ROS 2 para la comunicación entre nodos y la gestión independiente de espacios de nombres para cada robot.

Durante 1000 episodios de entrenamiento, el agente líder presentó una mejora progresiva en la recompensa acumulada, demostrando un aprendizaje estable y una navegación autónoma libre de colisiones. En la simulación multi-agente, los robots seguidores reprodujeron con precisión la trayectoria del líder, manteniendo formaciones estables y comunicación efectiva. Finalmente, las pruebas físicas confirmaron la correcta transferencia del modelo entrenado, manteniendo la coherencia del comportamiento observado en simulación.

Los resultados evidencian que la combinación del aprendizaje por refuerzo profundo con una arquitectura descentralizada basada en ROS 2 constituye una estrategia viable para el desarrollo de comportamientos cooperativos en robótica móvil. El sistema diseñado reduce la dependencia de control central, mejora la escalabilidad del sistema y sienta las bases para futuras validaciones en entornos físicos más complejos.

Palabras clave: Sistemas multi-agente, aprendizaje por refuerzo, Deep Q-Learning, ROS 2, Gazebo, robótica descentralizada.

Abstract

Environmental event detection is a key challenge in modern robotics, particularly in systems that require autonomy, cooperation, and distributed decision-making. Decentralized multi-agent systems (MAS) emerge as an efficient alternative to centralized approaches, offering greater scalability, robustness, and fault tolerance. This work proposes the design and implementation of a decentralized multi-agent architecture based on ROS 2 for environmental event detection, using TurtleBot3 Burger mobile platforms and deep reinforcement learning.

The research followed an experimental and applied approach, structured into three phases: (1) analysis and selection of physical variables, algorithms, and robotic platform; (2) development of the decentralized architecture integrating a leader agent trained through the Deep Q-Learning (DQN) algorithm and follower agents controlled by the ROBOTIS *Follower* algorithm; and (3) system validation in the Gazebo simulator using the ROS 2 middleware for inter-node communication and independent namespace management for each robot.

Over 1000 training episodes, the leader agent showed a progressive improvement in cumulative rewards, achieving stable learning and autonomous navigation without collisions. In the multi-agent simulation, the follower robots accurately replicated the leader's trajectory, maintaining stable formations and effective communication. Finally, physical tests confirmed the successful transfer of the trained model, maintaining consistency with the simulated behaviors.

The results confirm that combining deep reinforcement learning with a decentralized ROS 2-based architecture is a viable strategy for developing cooperative behaviors in mobile robotics. The designed system reduces the dependence on a central controller, enhances scalability, and establishes a solid foundation for future validations in more complex physical environments.

Keywords: Multi-agent systems, reinforcement learning, Deep Q-Learning, ROS 2, Gazebo, decentralized robotics.

Índice de figuras

Índice de tablas

Glosario

Agente Entidad autónoma capaz de percibir su entorno, tomar decisiones y ejecutar acciones con base en un objetivo definido.

Agente líder Robot encargado de explorar el entorno y aprender una política de navegación mediante aprendizaje por refuerzo profundo.

Agentes seguidores Robots que replican el movimiento del agente líder utilizando un algoritmo de seguimiento cooperativo (*Follower*).

Aprendizaje por refuerzo (RL) Paradigma del aprendizaje automático donde un agente aprende a tomar decisiones mediante prueba y error, recibiendo recompensas o penalizaciones según sus acciones.

Deep Q-Learning (DQN) Algoritmo de aprendizaje por refuerzo profundo que emplea redes neuronales para aproximar la función de valor de acción $Q(s, a)$, permitiendo tomar decisiones en espacios de estado continuos o de alta dimensión.

ROS 2 (Robot Operating System 2) Middleware de código abierto que facilita la comunicación entre nodos, la gestión de tópicos, servicios y acciones en sistemas robóticos distribuidos.

Gazebo Simulador 3D utilizado para modelar y probar entornos robóticos, ofreciendo simulación física realista e integración directa con ROS 2.

TurtleBot3 Burger Plataforma robótica móvil diferencial desarrollada por ROBOTIS, equipada con sensor LIDAR, odometría e integración nativa con ROS 2 y Gazebo.

LIDAR (Light Detection and Ranging) Sensor óptico que utiliza pulsos láser para medir distancias con alta precisión, generando un mapa 2D del entorno.

Nodo Unidad básica de ejecución en ROS 2 que realiza una función específica (como procesar datos de sensores, controlar motores o publicar información).

Tópico Canal de comunicación en ROS 2 donde los nodos publican o suscriben mensajes para intercambiar información de manera asíncrona.

Servicio Mecanismo de comunicación en ROS 2 que permite la interacción sincrónica entre nodos mediante solicitudes (*request*) y respuestas (*response*).

Política de acción Conjunto de reglas o estrategias aprendidas que determinan qué acción debe ejecutar el agente en cada estado del entorno.

Recompensa Valor numérico que representa el resultado de una acción en el entorno; guía el proceso de aprendizaje del agente.

Episodio Secuencia completa de interacciones entre el agente y el entorno, desde un estado inicial hasta alcanzar una meta o una condición terminal.

Arquitectura descentralizada Estructura en la que cada agente opera de forma autónoma, tomando decisiones locales sin depender de un controlador central.

Follower Algoritmo provisto por ROBOTIS que permite a un robot seguir la trayectoria de otro mediante control de posición y orientación.

Red neuronal profunda Modelo computacional inspirado en el cerebro humano, compuesto por múltiples capas que transforman las entradas en salidas mediante pesos ajustables.

Experiencia de repetición (Replay Buffer) Memoria utilizada por el algoritmo DQN para almacenar transiciones pasadas (s, a, r, s') y reutilizarlas durante el entrenamiento para mejorar la estabilidad del aprendizaje.

Capítulo 1

Introducción

1.1. Planteamiento del Problema

En los sistemas multi-agentes (MAS) centralizados, cada agente requiere una conexión constante con un organismo central para establecer comunicación y control. Haciendo que éste evalúe el estado y determine las acciones necesarias para cada uno de los agentes, enviando luego estas instrucciones a través de la red a los agentes correspondientes. Asimismo, esta estructura centralizada simplifica la comunicación al dirigirla hacia una entidad central, lo que puede ser útil en situaciones donde se requiere un control preciso y coordinado de las acciones de los agentes [1].

Sin embargo, este enfoque no es adecuado para la mayoría de las aplicaciones en las que cada agente debe tomar decisiones de forma independiente, debido a que esto hace que el sistema dependa de la unidad central, y por tanto, su fallo podría resultar en la falla de todo el sistema. Además, la carga en la red de comunicación es alta en un sistema centralizado, ya que implica retransmisiones constantes desde la entidad central a los agentes. Estas limitaciones afectan la eficiencia del sistema, especialmente cuando se incrementa el número de agentes, lo que aumenta significativamente los datos intercambiados entre los agentes y el controlador central. Este problema se agrava debido a limitaciones prácticas, como la capacidad limitada de los equipos de comunicación o restricciones presupuestarias, que dificultan el manejo de un gran número de agentes; por ende, los sistemas de red controlados centralmente generalmente enfrentan desafíos significativos de escalabilidad, dado que todos los cálculos se realizan en la unidad central de procesamiento. Si se agrega una gran cantidad de agentes afecta considerablemente el tiempo necesario para calcular las medidas de control, por lo tanto, en situaciones problemáticas reales, el control cooperativo con un enfoque centralizado se vuelve más complicado [1].

Por otro lado, en el Laboratorio de la Facultad y específicamente en el grupo de investigación GED [2], se emplea ROS (Robot Operating System) de forma centralizada en las implementaciones de sistemas multi-agentes. Esta configuración centralizada presenta desafíos específicos en el ámbito de ROS y dentro del laboratorio. por ejemplo, la dependencia de un nodo central usualmente puede restringir la escalabilidad y la autonomía de los agentes, dificultando la adaptación a entornos dinámicos. por lo tanto, la configuración y gestión de un sistema centralizado en ROS pueden ser complejas y requerir un conocimiento profundo de la plataforma, lo que puede representar una barrera para estudiantes menos experimentados. En cuanto a la seguridad, también es una preocupación relevante, ya que un nodo centralizado puede constituir un punto de vulnerabilidad en

términos de ataques cibernéticos o fallos técnicos. Estos desafíos deben ser abordados con el fin de maximizar la eficiencia y la efectividad de los sistemas multi-agentes en el laboratorio [3].

Las aplicaciones actuales de sistemas multi-agentes implementadas en agricultura de precisión [4], búsqueda y rescate requieren una mayor autonomía por parte de los agentes. Esto se debe a la necesidad de capturar información sobre eventos en el ambiente relacionados mediante variables físicas, como imágenes que muestran variaciones en la salud de las plantas para la detección de plagas, o movimientos que sugieren la presencia de personas bajo escombros, entre otros ejemplos. Esta información debe ser transmitida a los agentes vecinos de manera eficiente para que las decisiones puedan tomarse de forma más rápida, sin depender de una unidad de cómputo centralizada que pueda encontrarse a una distancia considerable.

Es por eso que surge la pregunta problema ¿Cómo diseñar un sistema multi-agente descentralizado con ROS, capaz de identificar cambios en el ambiente determinados por una variable física?.

1.2. Objetivos

1.2.1. Objetivo general

Diseñar una arquitectura multi-agente descentralizada basada en ROS para detección de eventos en el ambiente.

1.2.2. Objetivos específicos

- Identificar una plataforma disponible dentro del laboratorio de robótica que cumpla con los requerimientos de software y hardware para la implementación del algoritmo MAS a través de un lenguaje de programación como Python y bajo la arquitectura del sistema operativo ROS.
- Establecer un algoritmo MAS que permita a los agentes comunicarse bajo una arquitectura ROS descentralizada, de tal manera que puedan tomar decisiones independientes que lleven al sistema al cumplimiento de una meta común.
- Implementar dentro de la plataforma robótica seleccionada, un sistema de navegación multi-agente orientado a los cambios de una variable física del ambiente.
- Validar el funcionamiento de la plataforma robótica seleccionada en un ambiente de pruebas que involucre la arquitectura ROS y la variable física que determina los cambios en el ambiente.

1.3. Justificación

La detección de eventos en el ambiente es un desafío fundamental en el campo de la robótica y la inteligencia computacional, con aplicaciones cruciales en áreas como la vigilancia ambiental y la gestión de desastres naturales. Actualmente, los enfoques centralizados dominan la investigación en este campo, sin embargo, presentan limitaciones en términos de escalabilidad y robustez [5].

El proyecto de grado se enfocará en los sistemas multi-agentes descentralizados como una alternativa prometedora; el uso de una plataforma mediante ROS descentralizado cambiará la forma en que los agentes se ubican

y toman decisiones en entornos complejos. Este enfoque no solo ofrecerá un entorno ideal para la investigación en sistemas multi-agentes descentralizados, sino que también abrirá nuevas posibilidades en el campo de la robótica y la inteligencia computacional.

Los sistemas multi-agentes descentralizados ofrecen varias ventajas significativas, en primer lugar, mejoran la escalabilidad al distribuir la carga computacional entre los agentes, permitiendo manejar entornos más grandes y complejos de manera más eficiente. Además, aumentan la robustez del sistema al reducir los puntos únicos de falla y mejorar la tolerancia a fallos [1] [5].

Esta investigación se alinea estrechamente con la línea de investigación en Robótica e Inteligencia Computacional [2], ofreciendo contribuciones teóricas y prácticas significativas; de hecho, se espera que los resultados de este trabajo no solo avancen el conocimiento en el campo, sino que también tengan aplicaciones prácticas en situaciones del mundo real. Agregando a lo anterior, la implementación de sistemas multi-agentes descentralizados tiene el potencial de atraer a más estudiantes al semillero. Al realizar este proyecto se podrán explorar más algoritmos, como Teoría de juegos [6], Reinforcement Learning [7] con sus variaciones [8] o algoritmo de Consensus [9] también con sus respectivas variaciones [10], esto enriquecerá la formación académica y profesional, haciendo que la oportunidad de trabajar en un proyecto innovador y relevante en un entorno de investigación estimulante sea un incentivo importante para los estudiantes interesados en entrar al grupo de investigación GED.

Para concluir, este proyecto de grado tiene un valor significativo en términos de avance científico, aplicación práctica y formación de estudiantes, ya que, la adopción de sistemas multi-agentes descentralizados para la detección de eventos en el ambiente tiene el potencial de transformar la forma en que se aborda este problema, abriendo nuevas oportunidades para la investigación y la innovación en el campo de la robótica y la inteligencia computacional.

1.4. Impacto Social

La implementación de sistemas multi-agentes descentralizados para la detección de eventos en el ambiente puede tener un impacto social significativo, alineándose con los Objetivos de Desarrollo Sostenible (ODS) 6, 9, 11 y 15. Agregado a esto, también tiene un impacto en la Universidad Santo Tomás, específicamente en el laboratorio de Robótica de la facultad de Ingeniería Electrónica.

Esta tecnología podría estimular la investigación en sistemas multi-agentes descentralizados, generando nuevos proyectos que fomenten la innovación en la detección de eventos en el ambiente. Esto podría inspirar a otros investigadores y estudiantes del laboratorio de Robótica a explorar nuevas aplicaciones y mejoras en esta área, contribuyendo al ODS 9 (Industria, innovación e infraestructura) [11].

En la industria agropecuaria, la implementación de estos sistemas podría mejorar la eficiencia y la productividad, debido a que al detectar eventos ambientales como plagas, enfermedades o condiciones climáticas adversas de manera más precisa y oportuna, los agricultores podrían tomar medidas preventivas o correctivas, reduciendo pérdidas y aumentando la producción, lo que también se alinea con el ODS 9 [11]. Además, estos sistemas podrían aplicarse en búsqueda y rescate, mejorando la coordinación y eficiencia de las operaciones en situaciones de emergencia. Esto contribuiría al ODS 11 (Ciudades y comunidades sostenibles), al fortalecer la capacidad de respuesta ante desastres y mejorar la seguridad de las comunidades.[12]

En el ámbito de la monitorización ambiental, esta tecnología podría utilizarse para detectar y controlar la contaminación ambiental, supervisar la calidad del aire y del agua, y gestionar de manera más eficiente los

recursos naturales. Los agentes descentralizados podrían recopilar datos ambientales en diferentes ubicaciones y momentos, proporcionando una visión más completa y detallada de la situación ambiental y facilitando la toma de decisiones para la protección y conservación del medio ambiente, con esto se apoya a los ODS 6 (Garantizar la disponibilidad de agua y su gestión sostenible y el saneamiento para todos) [13] y 15 (Gestionar sosteniblemente los bosques, luchar contra la desertificación, detener e invertir la degradación de las tierras, detener la pérdida de biodiversidad) [14].

En resumen, la implementación de sistemas multi-agentes descentralizados para la detección de eventos en el ambiente podría impulsar la investigación en este campo, ofreciendo herramientas innovadoras para mejorar la eficiencia y la productividad en diferentes sectores, alineándose así con los ODS 6, 9, 11 y 15.

Capítulo 2

Estado del arte

Tomando como punto de partida el planteamiento del problema y su justificación, se procede a realizar una revisión bibliográfica centrada en investigaciones recientes relacionadas con sistemas multi-agentes descentralizados para la detección de eventos en el ambiente. Se busca también abordar estudios sobre la coordinación y comunicación entre agentes, así como el uso de tecnologías como ROS (Robot Operating System) para la implementación de estos sistemas. A pesar de la relevancia y potencial impacto de estos sistemas, se observa una limitada cantidad de información específica en este campo, lo que recalca la necesidad de contribuciones adicionales en esta área.

2.1. Sistemas Multi-agentes (MAS) descentralizados

En [15], se presenta un control adaptativo para coordinar un enjambre de drones en tareas cooperativas, utilizando un algoritmo descentralizado que sigue una trayectoria evitando colisiones y mantiene a ciertos drones dentro de su área de percepción. La implementación se realiza en Python y ROS, con resultados de simulación para comparaciones entre diferentes escenarios y futuros algoritmos. Por otro lado, [16] propone un algoritmo descentralizado de aprendizaje por refuerzo para coordinar agentes en sistemas multiagente, ajustando dinámicamente la tasa de aprendizaje entre agentes para mejorar la coordinación de estrategias. Además, [17] considera configuraciones donde la comunicación entre agentes es posible, formulando un problema de toma de decisiones en un sistema multiagente cooperativo con retrasos de comunicación constantes, y propone un método de aprendizaje por refuerzo que produce altos rendimientos en este problema. Finalmente, en [18], se propone una estrategia de auto-curación integrada en varias etapas para sistemas de distribución inteligente utilizando un sistema multiagente (MAS). La estrategia descompone el problema en subproblemas abordados por un marco de control unificado con diferentes algoritmos de etapas. Los agentes de toma de decisiones varían con los puntos de falla y la transición entre etapas, siendo totalmente descentralizada. Se enfoca en la coordinación entre algoritmos de etapas, como adaptación de comunicación y restauración de servicio, ofreciendo un buen rendimiento en tiempo real y funciones de auto-curación completas.

Estos artículos son útiles, ya que presentan ejemplos concretos de implementaciones y algoritmos descentralizados aplicables en el contexto de sistemas multi-agente para la detección de eventos en el ambiente. Además, ofrecen perspectivas sobre cómo abordar problemas de coordinación, comunicación y toma de decisiones en sistemas distribuidos, aspectos clave en este tipo de investigación.

2.2. ROS (Robot Operating System) en MAS

En [3], se presenta un estudio empírico para evaluar el rendimiento en tiempo real de ROS 2.0 en las capas de software del sistema y de comunicación. Se encontró que ROS 2.0, con su mejora en la comunicación a través del protocolo DDS, ofrece un comportamiento determinista en la capa del sistema, permitiendo determinar si las tareas son planificables y pueden cumplir con los plazos temporales especificados. Se prestó especial atención a la tasa de pérdida de datos y a la latencia general de los mensajes entre nodos. Los experimentos, realizados bajo diversas condiciones de trabajo, demostraron que ROS 2.0 es más adecuado que ROS 1.0 en términos de rendimiento en tiempo real, siendo una opción viable para aplicaciones del mundo real, como sistemas de robot de servicio multi-agente. Por otro lado, en [19], se describe un sistema de búsqueda cooperativa de UAV distribuido basado en ROS1-ROS2 sin un nodo central, utilizando ROS1-Mavros para el control, ROS2 para la colaboración y un algoritmo de planificación de enjambre de partículas. La combinación de ROS1 y ROS2 permite un control estable y un intercambio eficiente de información entre drones, como se demuestra en simulaciones con PX4, Gazebo y QgroundControl. Finalmente, en [20], se presenta una investigación sobre grupos descentralizados de robots móviles, los cuales ofrecen flexibilidad en la configuración, fácil multiplicación de robots en un grupo y alta precisión de control. Se mencionan ejemplos de aplicaciones en transporte a gran escala, donde la sincronización temporal de todos los robots es crucial, destacando el uso de ROS (Sistema Operativo de Robots) para controlar grupos de robots, lo cual proporciona herramientas para el desarrollo de software y el control de hardware básico de robots, incluyendo la transmisión de datos de control a través de Wi-Fi, el protocolo de transmisión de datos más común en robótica móvil, aunque susceptible a interferencias externas.

Los artículos investigados ofrecen información sobre el rendimiento en tiempo real de sistemas de comunicación como ROS 2.0, así como ejemplos de sistemas de búsqueda cooperativa y grupos descentralizados de robots móviles. Esta información puede ser relevante para el diseño y la implementación de sistemas multiagente descentralizados para la detección de eventos en el ambiente.

2.3. Algoritmos de Detección de Eventos

En [21], se presenta un algoritmo de simulación para sistemas híbridos de múltiples agentes, que consisten en varios conjuntos de ecuaciones diferenciales no suaves. Este algoritmo evita el uso de un paso de tiempo global y permite que cada subsistema se simule de forma asincrónica. Se demuestra su eficiencia en un problema de vehículos autónomos en carretera, mostrando una reducción significativa en el tiempo de computación en comparación con técnicas tradicionales. Por otro lado, en [22], se aborda el problema de consenso en sistemas multiagente (MASs), ampliamente investigado en años recientes debido a su aplicación en diversas áreas como ingeniería, formación de redes, cooperación de robots y vehículos aéreos no tripulados. El consenso implica que todos los agentes logren un valor común mediante la interacción entre ellos. Se han desarrollado numerosos enfoques y protocolos, destacando el control de consenso basado en eventos, que actualiza el controlador solo cuando el error de medición supera cierto umbral, reduciendo así la carga en la red de comunicación y estimulando la investigación en este campo. Además, [23] propone estrategias de consenso adaptativo para sistemas de múltiples agentes cooperativos, enfrentados a topologías de red dinámicas y que operan en entornos desconocidos, no estructurados y dinámicos. Se motiva en tácticas de búsqueda y rescate durante respuestas a desastres, donde la detección y seguimiento de múltiples activos es crucial. Las estrategias de consenso en sistemas de múltiples agentes son la base para lograr formaciones autónomas, agrupaciones y sincronización

de posiciones, siendo esencial una comunicación efectiva. Sin embargo, los canales de comunicación inalámbrica poco confiables y las cargas computacionales pesadas pueden afectar negativamente el rendimiento del sistema, lo que hace necesarias soluciones novedosas en problemas de control y estimación distribuidos en aplicaciones de sistemas de múltiples agentes.

Estos artículos presentan algoritmos y técnicas que pueden ser aplicados en sistemas multi-agentes descentralizados para la detección de eventos en el ambiente. Por ejemplo, ofrecen formas eficientes de modelar agentes con comportamientos complejos, así como métodos para lograr que los agentes colaboren y tomen decisiones efectivas en entornos dinámicos y desconocidos, aspectos cruciales en este tipo de sistemas.

Capítulo 3

Diseño Metodológico

3.1. Fases y Metodología

El presente proyecto de grado se desarrollará siguiendo una metodología centrada en la implementación de un sistema multi-agente descentralizado para la detección de eventos en el ambiente. Esta se divide en tres etapas que podemos ver en la Figura 1, Investigación y Análisis Preliminar, Desarrollo de algoritmo Multi-agente y arquitectura descentralizada y Pruebas, Validación y Documento Final. Cada una de estas etapas se llevará a cabo siguiendo un enfoque riguroso y estructurado, con el objetivo de garantizar el correcto desarrollo del sistema propuesto.

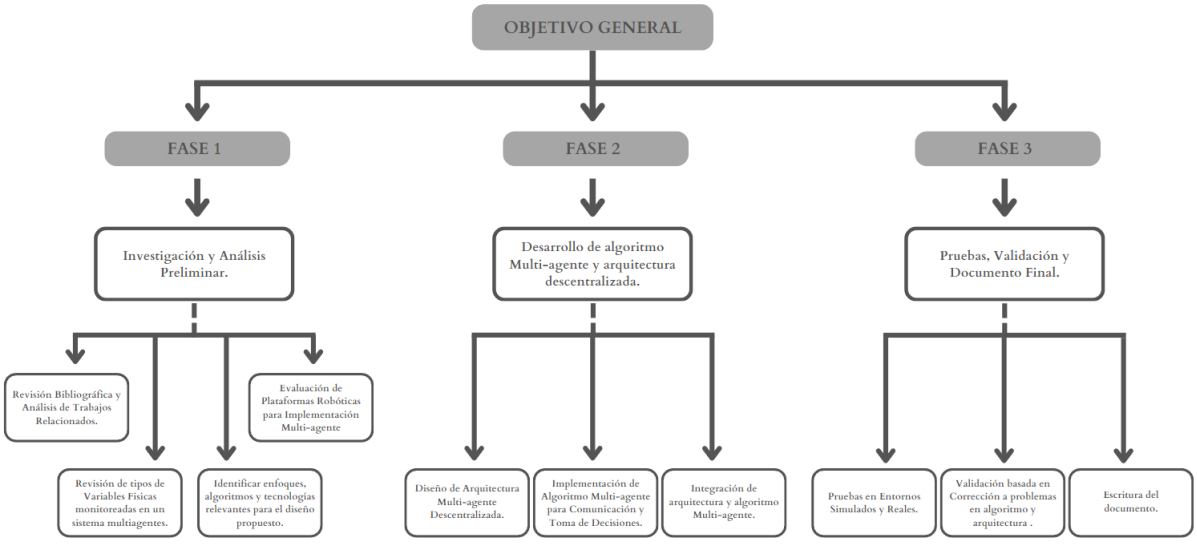

FIGURA 1: Mapa de metodología, fuente autor

Capítulo 4

Marco Teórico

Para el desarrollo del proyecto es necesario definir y aclarar los conceptos utilizados en la ejecución de este, con el fin de dar un entendimiento básico de la solución a realizar.

4.1. Sistemas multi-agente (MAS)

Un sistema multi-agente (MAS, por sus siglas en inglés) está formado por varios agentes que interactúan dentro de un mismo entorno. Cada agente es una entidad autónoma capaz de percibir, razonar y actuar con base en sus propios objetivos o en metas compartidas con otros agentes [24], [25]; a diferencia de un sistema centralizado, en un MAS no existe un único controlador que dicte las decisiones, sino que los agentes colaboran o compiten entre sí para resolver una tarea común.

De manera general, un MAS puede representarse como un conjunto de agentes $A = \{a_1, a_2, ..., a_n\}$ que interactúan con un entorno E. Cada agente observa una parte del estado del entorno, ejecuta una acción y recibe una recompensa o retroalimentación según el resultado. El comportamiento global del sistema emerge de la combinación de esas decisiones individuales sin que sea necesario un control central [26].

Los sistemas multi-agente se usan ampliamente en robótica, ya que permiten dividir tareas complejas entre varios robots. Por ejemplo, un grupo de robots puede explorar un área, transportar objetos o moverse en formación, comunicándose entre ellos para mantener la coordinación [27], [28]. Este enfoque también mejora la tolerancia a fallos: si un robot deja de funcionar, los demás pueden continuar la tarea sin que el sistema colapse.

En arquitecturas descentralizadas, cada agente toma decisiones con base en su información local, comunicándose solo con los vecinos cercanos o siguiendo reglas simples de cooperación. Entre los modelos más comunes de coordinación se encuentran el consenso (donde los agentes buscan un acuerdo común) y el esquema líder-seguidor, donde un agente guía el movimiento del resto [29], [30]; estas estrategias permiten lograr comportamientos colectivos estables, como desplazamiento en grupo, mantenimiento de formaciones o la prevención de colisiones.

4.2. Aprendizaje por refuerzo (Reinforcement Learning)

El aprendizaje por refuerzo (RL, por sus siglas en inglés) es una rama del aprendizaje automático que se basa en la idea de que un agente puede aprender a actuar en un entorno mediante ensayo y error. En lugar de recibir ejemplos etiquetados (como en el aprendizaje supervisado), el agente obtiene una recompensa o penalización en función de las consecuencias de sus acciones; utiliza esa retroalimentación para mejorar su comportamiento a lo largo del tiempo [31], [32].

En este paradigma, el agente interactúa de manera continua con un entorno (*environment*), observando su estado, ejecutando una acción y recibiendo una recompensa. El objetivo del agente es aprender una política $\pi(a|s)$, que indique cuál acción a debe realizar en cada estado s para maximizar la suma de recompensas futuras esperadas [26].

Matemáticamente, el proceso se modela como un Proceso de Decisión de Markov (MDP, por sus siglas en inglés), definido por el conjunto (S, A, P, R, γ), donde:

- S es el conjunto de estados posibles del entorno.
- A es el conjunto de acciones disponibles.
- $P(s'|s, a)$ es la probabilidad de transición al nuevo estado s' tras ejecutar la acción a desde el estado s.
- $R(s, a)$ es la recompensa inmediata obtenida.
- $\gamma \in [0, 1]$ es el **factor de descuento**, que determina la importancia de las recompensas futuras.

El valor esperado de una política se describe mediante la **ecuación de Bellman**, que relaciona el valor de un estado con las recompensas y los valores de los estados futuros:

$$V^{\pi}(s) = \sum_{a} \pi(a|s) \sum_{s'} P(s'|s, a)[R(s, a) + \gamma V^{\pi}(s')]$$

A partir de esta formulación, el agente busca la política óptima π^* que maximice el retorno esperado de recompensas. En la práctica, esto se logra actualizando de forma iterativa los valores asociados a cada acción mediante la siguiente regla de actualización, conocida como ecuación de Bellman para Q-Learning [33]:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)]$$

donde:

- α es la tasa de aprendizaje,
- r es la recompensa inmediata,
- γ es el factor de descuento,
- s' es el nuevo estado alcanzado después de aplicar la acción a.

Este proceso de aprendizaje permite que el agente ajuste gradualmente su comportamiento con base en la experiencia acumulada, favoreciendo las acciones que conducen a mayores recompensas a largo plazo. En

aplicaciones robóticas, como la navegación autónoma, el aprendizaje por refuerzo resulta especialmente útil, pues permite que los robots aprendan estrategias de movimiento óptimas mediante la interacción directa con su entorno simulado o real [34], [35].

4.3. Aprendizaje profundo por refuerzo: Deep Q-Learning (DQN)

El algoritmo Deep Q-Learning (DQN) es una extensión del método clásico de *Q-Learning*, en el se emplean redes neuronales profundas para aproximar la función de valor $Q(s, a)$. Este enfoque fue desarrollado para resolver el problema de escalar el aprendizaje por refuerzo a entornos con espacios de estados muy grandes o continuos, donde resulta inviable almacenar una tabla Q explícita [36].

En el aprendizaje tradicional, el agente mantiene una tabla con los valores $Q(s, a)$, actualizada mediante la ecuación de Bellman. Sin embargo, en entornos complejos como la navegación robótica el número de estados posibles es tan alto que resulta imposible representar todos los pares (s, a). Para resolver esta limitación, DQN sustituye la tabla por una red neuronal que toma como entrada el estado del entorno y produce los valores $Q(s, a; \theta)$ para todas las acciones posibles, donde θ representa los pesos de la red.

El objetivo del entrenamiento es ajustar los parámetros θ de manera que el valor estimado $Q(s, a; \theta)$ se acerque al valor óptimo definido por la ecuación de Bellman. Para ello, se minimiza la siguiente función de pérdida:

$$L(\theta) = \mathbb{E}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta^-) - Q(s, a; \theta)\right)^2\right]$$

donde:

- r es la recompensa recibida tras ejecutar la acción a,
- s' es el nuevo estado alcanzado,
- γ es el factor de descuento,
- θ^- son los parámetros de una red objetivo (*target network*) que se actualiza periódicamente para mejorar la estabilidad del entrenamiento.

Durante el proceso de aprendizaje, el agente almacena las transiciones (s, a, r, s') en una memoria de repetición de experiencias (*Replay Buffer*). De este modo, las muestras se reutilizan de manera aleatoria, rompiendo la correlación temporal entre los datos y permitiendo un entrenamiento más estable [37]. El uso de esta técnica fue uno de los principales avances introducidos por DQN, junto con la implementación de la red objetivo que evita oscilaciones en los valores de Q.

La política de selección de acciones generalmente equilibra la exploración y la explotación mediante una estrategia simple en la que el agente elige una acción aleatoria con probabilidad ϵ (exploración) y la mejor acción conocida con probabilidad $1-\epsilon$ (explotación). A medida que el entrenamiento progresa, el valor de ϵ disminuye gradualmente, favoreciendo decisiones más precisas basadas en la experiencia adquirida.

4.4. Arquitecturas descentralizadas en robótica multi-agente

En un sistema robótico multi-agente, la arquitectura de control define cómo se organizan los flujos de información y la toma de decisiones entre los distintos robots. Estas arquitecturas pueden clasificarse, de manera general, en tres tipos: centralizadas, distribuidas y descentralizadas [28], [29].

En una arquitectura centralizada, un nodo o servidor principal recopila la información de todos los agentes y calcula las acciones que cada uno debe ejecutar. Aunque este enfoque puede ofrecer una coordinación precisa, presenta limitaciones en términos de escalabilidad, robustez y tolerancia a fallos, ya que la pérdida del nodo central puede comprometer todo el sistema [27].

Por el contrario, en las arquitecturas descentralizadas, cada agente toma decisiones de forma autónoma, utilizando únicamente la información obtenida de su entorno local o de los agentes cercanos. Este tipo de arquitectura permite una mayor flexibilidad, reduce la dependencia de la comunicación global y mejora la resiliencia ante fallos individuales [29], [38]. En este enfoque, el comportamiento colectivo del grupo emerge de la interacción local entre los agentes, lo que favorece la adaptabilidad en entornos dinámicos.

Uno de los modelos más empleados en robótica cooperativa descentralizada es el esquema líder-seguidor (*leader-follower*). En este modelo, un agente actúa como líder y define la trayectoria o el objetivo principal, mientras que los demás agentes (seguidores) ajustan su movimiento con base en la información recibida del líder o de sus vecinos cercanos [28], [29]. Este paradigma, al combinar coordinación y simplicidad computacional, permitiendo mantener formaciones, realizar exploración cooperativa o ejecutar tareas de transporte conjunto [30].

4.5. Simulación y comunicación robótica: ROS 2 y Gazebo

El desarrollo y validación de sistemas multi-agente en robótica requieren entornos que permitan la comunicación entre múltiples nodos, la simulación física realista y la integración con algoritmos de control y aprendizaje. En este contexto, dos de las herramientas más utilizadas en investigación robótica son el Robot Operating System 2 (ROS 2) y el simulador Gazebo; ambos ofrecen una infraestructura modular, abierta y altamente interoperable [39], [40].

4.5.1. Robot Operating System 2 (ROS 2)

ROS 2 es un marco de software basado en componentes para el desarrollo de sistemas robóticos distribuidos. Su arquitectura se basa en una comunicación de tipo publicador-suscriptor, donde los nodos intercambian información a través de tópicos y servicios dentro de una red. Cada nodo puede ejecutar tareas independientes, como procesar sensores, controlar actuadores o ejecutar algoritmos de aprendizaje, permitiendo la integración modular de múltiples agentes [41].

A diferencia de su predecesor ROS 1, ROS 2 incorpora una capa de comunicación basada en el estándar DDS (Data Distribution Service), lo que permite soportar sistemas distribuidos y en tiempo real. Esto es especialmente importante en arquitecturas descentralizadas, donde los robots deben comunicarse de forma directa sin depender de un nodo central [42]. Además, ROS 2 ofrece herramientas de sincronización temporal (*use_sim_time*), remapeo de tópicos y espacios de nombres (*namespaces*), funcionalidades clave para la ejecución simultánea de múltiples robots.

El ecosistema ROS 2 cuenta con una amplia colección de paquetes, entre ellos `turtlebot3_bringup`, `turtlebot3_navigation2` y `turtlebot3_machine_learning`, los cuales facilitan la configuración de robots móviles, el control autónomo y la integración con algoritmos de aprendizaje por refuerzo. En proyectos multi-agente, ROS 2 permite lanzar instancias independientes de cada robot, garantizando la comunicación aislada mediante `namespaces` como `/TB3_1`, `/TB3_2`, etc., lo que evita interferencias y posibilita la cooperación descentralizada.

4.5.2. Gazebo

Gazebo es un simulador físico de código abierto diseñado para probar y validar algoritmos de control y aprendizaje en entornos tridimensionales realistas. Ofrece un motor de física que reproduce el comportamiento dinámico de los robots, como fuerzas, colisiones, fricción y gravedad, lo que permite replicar de forma precisa el comportamiento de los sistemas reales [40].

En combinación con ROS 2, Gazebo permite ejecutar simulaciones en tiempo real donde los robots interactúan con modelos de obstáculos, sensores y metas virtuales. Los sensores, como el *LIDAR*, cámaras o IMUs, generan datos que son publicados en tópicos ROS, lo que facilita su procesamiento por los nodos de control o aprendizaje. Asimismo, las acciones de movimiento se envían al simulador mediante comandos de velocidad (`/cmd_vel`), cerrando el ciclo de percepción–acción requerido para el aprendizaje por refuerzo.

Para entornos multi-agente, Gazebo permite instanciar varios robots simultáneamente mediante archivos de lanzamiento (`.launch.py`) que definen los modelos, posiciones iniciales y parámetros individuales de cada robot. Cada instancia del robot mantiene su propio conjunto de tópicos y transformaciones `/tf`, lo cual asegura independencia funcional y coherencia en la comunicación distribuida; esta capacidad fue fundamental para el desarrollo del presente proyecto, ya que permitió validar la interacción simultánea del agente líder (entrenado con DQN) y los agentes seguidores (controlados por el algoritmo *Follower*) dentro de un mismo entorno virtual.

La integración entre ROS 2 y Gazebo proporcionan una plataforma robusta de validación, que permite simular, monitorear y ajustar los parámetros de los agentes antes de su implementación física. Esta sinergia entre middleware y simulador reduce riesgos, optimiza tiempos de desarrollo y facilita la transferencia del conocimiento adquirido en simulación hacia el mundo real, en concordancia con las mejores prácticas en investigación robótica actual [40], [41].

Capítulo 5

Desarrollo Conceptual

Este capítulo explica el desarrollo conceptual del sistema multi-agente descentralizado propuesto. Primero se presentan los elementos fundamentales del diseño, como las variables físicas, el enfoque de aprendizaje y la plataforma robótica seleccionada (Fase 1). Luego se describe la arquitectura de control, donde el agente líder utiliza el algoritmo Deep Q-Learning (DQN) y los agentes seguidores emplean el algoritmo Follower de ROBOTIS (Fase 2). Finalmente, se aborda la implementación y validación en simulación con *Gazebo*, detallando los archivos `launch`, la generación de metas dinámicas y los nodos del entorno y del agente (Fase 3).

5.1. Fase 1: Investigación y Análisis Preliminar

Esta fase tiene como propósito establecer los fundamentos conceptuales y técnicos que orientan el desarrollo del sistema multi-agente descentralizado propuesto. En ella se realiza un proceso de análisis previo que permite identificar las condiciones experimentales, las herramientas de hardware y software, y los criterios de diseño que guiarán la implementación posterior del sistema.

El análisis se divide en tres componentes principales: en primer lugar, se lleva a cabo la **selección de las variables físicas** que permitirán la percepción del entorno por parte de los agentes; posteriormente, se define el **enfoque de aprendizaje** más adecuado para dotar de autonomía y capacidad de decisión al sistema, optando por el uso del algoritmo *Deep Q-Learning (DQN)*; y finalmente, se presenta la **selección de la plataforma robótica**, en la cual se justifica la elección del *TurtleBot3 Burger* como unidad base para la experimentación y validación del proyecto.

Estas tres etapas permiten sentar las bases que estructuran la primera fase, asegurando coherencia entre los objetivos planteados, las capacidades del entorno de laboratorio y la arquitectura multi-agente propuesta.

5.1.1. Selección de variables físicas

Durante la etapa inicial del proyecto, se realizó una revisión de diferentes tipos de variables físicas potencialmente útiles para la percepción del entorno por parte de los agentes robóticos. Entre las variables analizadas, se consideraron la distancia, la temperatura, la humedad, la iluminancia, la presión y el nivel de sonido. Cada una de ellas presenta potencial de aplicación en distintos contextos de control y aprendizaje autónomo, en función del tipo de entorno y de los objetivos específicos del sistema.

Sin embargo, tras analizar las condiciones del Laboratorio de la Facultad; específicamente del grupo de investigación GED y los requerimientos del proyecto, se optó por emplear variables de distancia como principales fuentes de información sensorial. Esta decisión se fundamenta en que los sensores de este tipo, como los *LIDAR*, *ultrasónicos* o *infrarrojos* son más accesibles, precisos y seguros para realizar pruebas en entornos controlados, además de estar integrados de forma nativa en las plataformas *TurtleBot3*.

Diversos estudios coinciden en que la medición de distancia mediante sensores LIDAR y ultrasónicos constituye una de las herramientas más confiables para la navegación autónoma en robótica móvil, por su precisión, bajo ruido y capacidad para operar en tiempo real [43], [44], [45]. Estos sensores permiten al robot percibir la geometría del entorno, detectar obstáculos y generar mapas locales, lo que los convierte en la opción más adecuada para tareas de evasión de colisiones y planificación de trayectorias.

El uso de la distancia como variable física principal permite al agente obtener una representación directa del entorno, identificando la proximidad a obstáculos y delimitando zonas seguras de navegación. Esto facilita el diseño del modelo de aprendizaje por refuerzo, ya que el espacio de estados puede construirse a partir de valores normalizados de distancia que reflejan la interacción entre el robot y su entorno inmediato. Por tanto, esta elección no solo se ajusta a las capacidades técnicas del laboratorio, sino que también optimiza el proceso de entrenamiento y validación de los agentes en simulación y, posteriormente, en plataforma física.

Para complementar la justificación anterior, en la Tabla 1 se presenta un resumen comparativo de las principales variables físicas consideradas. Se evaluaron según su relevancia para la navegación autónoma, facilidad de implementación y compatibilidad con el entorno del laboratorio. De acuerdo con este análisis, la variable distancia resultó ser la más adecuada para el desarrollo del sistema.

Variable física	Descripción general	Aplicación en robótica	Evaluación para el proyecto
Distancia	Determina la separación entre el robot y los objetos del entorno. Se mide mediante sensores LIDAR, ultrasónicos o infrarrojos.	Navegación autónoma, evitación de obstáculos y mapeo del entorno.	Alta precisión, fácil integración y disponibilidad en el TurtleBot3.
Temperatura	Representa el nivel térmico del ambiente o de componentes electrónicos.	Monitoreo de condiciones ambientales o sobrecalentamiento de sistemas.	Requiere sensores adicionales y no aporta información relevante para navegación.
Humedad	Mide la cantidad de vapor de agua en el aire.	Control ambiental y monitoreo de entornos interiores.	Baja relevancia para la toma de decisiones en navegación.
Iluminancia	Indica la intensidad de luz incidente en el entorno.	Ajuste de visión artificial o control de exposición en cámaras.	No es necesaria en entornos controlados de laboratorio.
Presión	Expresa la fuerza ejercida sobre una superficie.	Medición de altitud en drones o sistemas neumáticos.	No aplica al tipo de plataforma terrestre utilizada.
Nivel de sonido	Evalúa la intensidad acústica del entorno.	Reconocimiento de comandos de voz o respuesta a estímulos sonoros.	Requiere procesamiento adicional y no aporta información espacial directa.

TABLA 1: Comparación de variables físicas analizadas para la percepción del entorno

5.1.2. Selección del algoritmo

De acuerdo con el objetivo general del proyecto, que consiste en diseñar una arquitectura multi-agente descentralizada basada en ROS para la detección de eventos en el ambiente, se evaluaron diferentes estrategias de aprendizaje que permitieran dotar a los agentes de la capacidad de percibir, decidir y actuar de manera autónoma ante los cambios de una variable física del entorno.

Entre los enfoques considerados, se analizaron los métodos de **aprendizaje supervisado**, **aprendizaje no supervisado** y **aprendizaje por refuerzo**. Este último se identificó como el más apropiado, ya que permite que el agente aprenda mediante la interacción directa con el entorno, recibiendo recompensas o penalizaciones en función de las consecuencias de sus acciones. Esta característica es esencial en entornos dinámicos donde no existe un conjunto de datos previamente etiquetado y donde las condiciones cambian constantemente, como ocurre en la navegación robótica.

En particular, se seleccionó el algoritmo Deep Q-Learning (DQN) por su capacidad de combinar los principios del *Q-Learning* clásico con la potencia de las redes neuronales profundas para aproximar la función de valor de acción. Esta integración permite manejar espacios de estado de alta dimensión, como los generados por el sensor LIDAR del TurtleBot3, que proporciona una nube de datos de distancia alrededor del robot.

Diversos estudios respaldan la eficacia del Deep Q-Learning en aplicaciones de robótica móvil. En particular, se ha demostrado que este tipo de algoritmos basados en aprendizaje por refuerzo profundo permiten a los robots navegar de forma autónoma en entornos dinámicos, evitando obstáculos y optimizando trayectorias mediante el uso de sensores como el LIDAR [46], [47], [48]. Estos enfoques también destacan por su capacidad para generalizar comportamientos aprendidos en simulación hacia entornos reales, lo que los convierte en una herramienta sólida para proyectos basados en ROS2 y plataformas como *TurtleBot3*.

En la arquitectura propuesta, únicamente uno de los agentes es sometido al proceso de entrenamiento mediante el algoritmo DQN. Este agente actúa como referencia o agente líder, encargado de explorar el entorno y optimizar su política de acción a partir de la experiencia acumulada. Una vez que este agente alcanza un comportamiento estable, los demás agentes del sistema siguen sus trayectorias y decisiones en tiempo real, replicando su política de navegación y adaptándola a su propia percepción local.

Este enfoque permite reducir significativamente la carga computacional asociada al entrenamiento simultáneo de múltiples agentes y, al mismo tiempo, mantiene la coherencia del comportamiento colectivo. Cada robot conserva su autonomía en la ejecución bajo la arquitectura ROS descentralizada, pero las decisiones globales emergen de la coordinación implícita al seguir el comportamiento del agente líder entrenado.

5.1.3. Selección de la plataforma robótica

Dentro del laboratorio de robótica se dispone de diferentes plataformas que pueden emplearse para la implementación de algoritmos de navegación autónoma y aprendizaje multi-agente, entre las cuales se encuentran los Traxxas Racer Then, el AWS DeepRacer, los TurtleBot3 y los robots de la Middle Size League. Cada una de estas plataformas presenta características particulares en cuanto a tamaño, hardware disponible, capacidades sensoriales, nivel de integración con *ROS* y facilidad de implementación de algoritmos basados en aprendizaje por refuerzo.

Tras un análisis comparativo de sus especificaciones técnicas y de las condiciones del entorno experimental, se determinó que la plataforma TurtleBot3 Burger es la más adecuada para el desarrollo del presente proyecto.

Esta elección se debe principalmente a que los TurtleBot3 poseen una integración nativa con ROS 2 Humble, cuentan con un sensor LIDAR 360° que permite obtener mediciones precisas de distancia y disponen de una arquitectura modular y abierta que facilita la incorporación de algoritmos personalizados desarrollados en *Python* [49], [50].

Además, su tamaño compacto, bajo costo operativo y compatibilidad con los entornos de simulación *Gazebo* y *Rviz* hacen de esta plataforma una opción ideal para validar sistemas de navegación multi-agente en escenarios controlados de laboratorio. Investigaciones recientes respaldan esta elección, destacando que el TurtleBot3 Burger es ampliamente utilizado en proyectos de navegación autónoma basados en aprendizaje por refuerzo profundo y sensores LIDAR dentro del ecosistema ROS 2, demostrando su idoneidad para entornos experimentales y académicos [51].

Por otro lado, la plataforma AWS DeepRacer, aunque orientada al aprendizaje por refuerzo, está diseñada principalmente para competiciones de conducción autónoma a pequeña escala, con sensores limitados y soporte parcial para ROS [52]. Asimismo, los vehículos Traxxas Racer Then ofrecen gran velocidad y tracción diferencial, pero su enfoque recreativo y la falta de sensores avanzados los hace poco adecuados para entornos cooperativos multi-agente [53]. Finalmente, los robots de la Middle Size League (FIRA/Robocup) presentan una configuración robusta con múltiples sensores, aunque requieren un espacio físico amplio y una adaptación compleja a ROS 2, lo que limita su uso en entornos académicos controlados [54].

La Tabla 2 presenta un resumen comparativo de las plataformas disponibles y los criterios evaluados para la selección final.

Plataforma	**Tipo de robot**	**Sensores principales**	**Compatibilidad con ROS2**	**Evaluación para el proyecto**
Traxxas Racer Then	Vehículo terrestre a escala con tracción diferencial	Cámara frontal, sensor IMU	Parcial	Alta velocidad pero limitada precisión sensorial. No ideal para navegación autónoma multi-agente.
AWS DeepRacer	Vehículo autónomo de una sola unidad	Cámara RGB, sensor de inercia	Parcial (ROS1 con adaptaciones)	Orientado a competiciones individuales, no diseñado para cooperación entre agentes.
TurtleBot3 Burger	Robot móvil diferencial compacto	LIDAR, IMU, sensores de odometría	Completa (ROS2 Humble)	Integración total con ROS2, ideal para aprendizaje por refuerzo y sistemas multi-agente.
Middle Size League	Plataforma robótica de gran tamaño (FIRA/Robocup)	Cámaras, sensores ultrasónicos y de fuerza	Limitada (ROS1 con adaptación compleja)	Requiere amplio espacio físico y alto mantenimiento. No viable en laboratorio.

TABLA 2: Comparación de plataformas robóticas disponibles en el laboratorio

Con la Fase 1 se establecieron las bases necesarias para el desarrollo del sistema, comprendiendo la selección de la variable física de distancia, el uso del sensor LIDAR, la definición del algoritmo Deep Q-Learning (DQN) y la elección de la plataforma TurtleBot3 Burger. Estos elementos proporcionan el marco de trabajo sobre el cual se construirá la arquitectura multi-agente descentralizada que será abordada en la siguiente fase.

5.2. Fase 2: Desarrollo del algoritmo y arquitectura descentralizada

Esta fase describe el proceso de diseño e implementación del sistema multi-agente descentralizado propuesto. Su propósito es integrar los componentes de control y aprendizaje que permiten la cooperación entre los diferentes robots del entorno.

En primer lugar, se analiza el funcionamiento del algoritmo *Follower* provisto por ROBOTIS, empleado para el seguimiento cooperativo entre robots. Posteriormente, se detalla el algoritmo de *aprendizaje profundo por refuerzo* (DQN) incluido en el paquete `turtlebot3_machine_learning`, utilizado para el entrenamiento del agente líder. Finalmente, se aborda la integración entre ambos algoritmos, la cual permite combinar el aprendizaje autónomo del líder con el comportamiento de seguimiento de los demás agentes, consolidando así una arquitectura descentralizada capaz de coordinar acciones colectivas sin depender de un controlador central.

5.2.1. Algoritmo *Follower* (ROBOTIS)

El algoritmo *Follower* desarrollado por ROBOTIS forma parte de los ejemplos básicos de la plataforma *TurtleBot3* y se encuentra disponible en el paquete `turtlebot3_applications` [55]. Este ejemplo tiene como objetivo permitir que un robot móvil siga la trayectoria o el movimiento de otro robot, manteniendo una distancia constante y una orientación alineada. El principio de funcionamiento se basa en la lectura continua de la odometría del robot líder y en el control diferencial de las velocidades lineal y angular del seguidor, con el fin de minimizar el error relativo entre ambos.

En la documentación oficial de ROBOTIS [56], se describe que el *Follower* utiliza la información de odometría y las transformaciones publicadas en el tópico `/tf` para estimar la posición del robot líder con respecto al marco de referencia del seguidor. A partir de esta información, el algoritmo calcula la distancia euclidiana y el ángulo relativo, los cuales son empleados por un controlador proporcional que ajusta las velocidades del seguidor. El comando resultante se publica en el tópico `/cmd_vel`, lo que permite que el robot mantenga una separación segura y siga de forma fluida los movimientos del líder.

El proceso se puede resumir en tres etapas principales:

1. **Percepción:** el nodo recibe la información de odometría del robot líder mediante las transformaciones `/tf`.
2. **Cálculo de error:** se determina la distancia y el ángulo entre el robot seguidor y el líder.
3. **Control de movimiento:** se calculan las velocidades lineal y angular necesarias para corregir el error y mantener la trayectoria de seguimiento.

El *Follower* se implementa como un nodo independiente que suscribe los tópicos `/odom` y `/tf`, y publica las velocidades en `/cmd_vel`. En un entorno multi-robot, cada agente opera dentro de su propio espacio

de nombres (*namespace*), por ejemplo, `/TB3_1`, `/TB3_2`, etc, lo que permite ejecutar múltiples instancias del mismo nodo sin interferencias. Además, mediante la configuración de los remapeos en el archivo `launch`, es posible definir qué robot actuará como líder y cuáles serán sus seguidores.

En el entorno de pruebas del laboratorio de robótica, este algoritmo se utiliza para modelar el comportamiento cooperativo entre agentes. El primer robot actúa como *líder*, siendo teleoperado o ejecutando un comportamiento autónomo aprendido mediante el algoritmo DQN, mientras que los demás robots ejecutan el nodo *Follower*. De esta forma, cada seguidor replica las trayectorias del líder basándose en su posición y orientación, logrando una formación en cadena con comportamiento coordinado.

El seguimiento se realiza de manera descentralizada, ya que cada seguidor toma decisiones locales en función de la información que recibe del entorno y del líder. Esto facilita la integración con el sistema multi-agente propuesto, donde el agente principal (entrenado con aprendizaje por refuerzo) establece el comportamiento de referencia, y los demás robots aprenden indirectamente al reproducir su dinámica.

El correcto funcionamiento del *Follower* depende de la precisión de la odometría y de la sincronización temporal entre los robots. La deriva de odometría o los retrasos en las transformaciones pueden afectar la estabilidad del seguimiento. Por ello, se recomienda mantener frecuencias de publicación superiores a 10 Hz y calibrar periódicamente los sensores de encoders. Asimismo, el algoritmo está diseñado para funcionar en entornos controlados con trayectorias suaves y sin obstáculos dinámicos, lo que coincide con las condiciones del laboratorio empleadas en el presente trabajo.

El *Follower* constituye una base sólida para implementar comportamientos de cooperación entre robots móviles, permitiendo que múltiples unidades TurtleBot3 ejecuten tareas coordinadas. Su integración con el algoritmo de aprendizaje profundo DQN proporciona una arquitectura híbrida en la que el conocimiento adquirido por el agente líder se propaga al resto del sistema de manera descentralizada, fortaleciendo el concepto de colaboración multi-agente en el marco del proyecto.

5.2.2. Algoritmo de aprendizaje (DQN) – Paquete `turtlebot3_machine_learning` (ROBOTIS)

El paquete `turtlebot3_machine_learning` [57], desarrollado por ROBOTIS, implementa un esquema de aprendizaje por refuerzo profundo aplicado al robot *TurtleBot3*, basado en el algoritmo Deep Q-Learning (DQN). Este paquete permite que el robot aprenda una estrategia de decisión autónoma para desplazarse dentro del entorno, evitando obstáculos y seleccionando trayectorias seguras mediante retroalimentación por recompensas. Según la documentación oficial, el software está diseñado para ejecutarse en *Ubuntu 22.04* bajo *ROS 2 Humble*, utilizando *Python 3.10* y bibliotecas de aprendizaje profundo como `TensorFlow` y `Keras`.

Previo a la implementación del algoritmo, fue necesario preparar el entorno de desarrollo compatible con el paquete `turtlebot3_machine_learning`. Este paquete requiere la instalación de dependencias específicas como `numpy`, `scipy`, `tensorflow`, `keras` y `pyqtgraph`, utilizadas para el procesamiento de datos, el cálculo numérico y la construcción de redes neuronales profundas.

La instalación se realizó clonando el repositorio oficial de ROBOTIS:

```
git clone -b humble https://github.com/ROBOTIS-GIT/turtlebot3_machine_learning.git
```

Posteriormente, el paquete fue incorporado al *workspace* de ROS 2 y compilado mediante el comando:

```
colcon build
```

Este procedimiento genera los nodos y servicios necesarios para ejecutar tanto el entrenamiento como la inferencia del modelo DQN. La documentación de ROBOTIS recomienda ejecutar los ejemplos inicialmente en un entorno simulado (Gazebo) antes de su implementación en los robots físicos, con el fin de garantizar la estabilidad del aprendizaje y una correcta comunicación entre los nodos del sistema [58].

El paquete `turtlebot3_machine_learning` aplica el algoritmo **Deep Q-Learning (DQN)**, una técnica de aprendizaje por refuerzo que combina el aprendizaje de valores Q con una red neuronal profunda. El objetivo del agente es aprender una estrategia de control que maximice la recompensa acumulada durante la navegación en entornos con obstáculos.

El flujo general del algoritmo se puede resumir de la siguiente manera:

1. **Percepción del entorno:** el agente recibe como entrada un vector de estado compuesto principalmente por las mediciones del sensor LIDAR, la velocidad lineal y la orientación del robot.
2. **Selección de acción:** con base en el estado actual, la red neuronal predice los valores $Q(s, a)$ para cada acción posible (por ejemplo, avanzar, girar a la izquierda o girar a la derecha). El agente elige la acción que considera más adecuada según los valores estimados, incorporando ocasionalmente decisiones aleatorias para fomentar la exploración y evitar comportamientos repetitivos.
3. **Ejecución y retroalimentación:** la acción seleccionada se ejecuta en el entorno simulado. El agente observa el nuevo estado y recibe una recompensa proporcional a su desempeño (por ejemplo, penalización por colisión o recompensa por avanzar hacia áreas seguras).
4. **Actualización de la red:** se calcula el error de Bellman y se ajustan los pesos de la red neuronal mediante retropropagación, reduciendo la diferencia entre los valores predichos y los observados.
5. **Almacenamiento de experiencias:** cada transición (s, a, r, s') se almacena en un búfer de memoria de repetición, desde el cual se extraen muestras aleatorias para entrenar la red de manera más estable y evitar correlaciones entre episodios consecutivos.

Este proceso se repite durante múltiples episodios hasta que el agente alcanza un comportamiento estable y coherente con los objetivos de navegación. Una vez completado el entrenamiento, el modelo obtenido puede ejecutarse en modo *inferencia*, permitiendo que el robot tome decisiones en tiempo real basadas en el conocimiento adquirido, sin necesidad de continuar el proceso de aprendizaje durante la operación.

En el contexto del presente trabajo, el aprendizaje por refuerzo profundo se aplicó exclusivamente al agente líder del sistema multi-agente descentralizado. Este agente fue el único sometido al proceso de entrenamiento utilizando el paquete `turtlebot3_machine_learning`, desarrollando una estrategia de control autónoma mediante el algoritmo DQN, que le permite desplazarse de forma eficiente dentro del entorno evitando obstáculos.

Una vez finalizado el entrenamiento, el modelo obtenido (archivo con los pesos de la red neuronal) se implementó únicamente en el agente líder. Este robot utiliza lo aprendido para desplazarse de forma autónoma y tomar decisiones en tiempo real según la información que recibe de sus sensores, los demás robots, denominados agentes seguidores, no usan el modelo DQN ni requieren entrenamiento adicional. En su lugar, ejecutan el algoritmo *Follower* proporcionado por ROBOTIS, que les permite seguir al agente líder manteniendo una distancia constante y sincronizando su movimiento dentro del entorno.

De esta manera, el sistema combina el aprendizaje por refuerzo profundo aplicado al líder con un esquema de seguimiento cooperativo para los seguidores. Esto permite lograr un comportamiento grupal coherente y estable sin depender de un controlador central, manteniendo la naturaleza descentralizada del sistema.

La Fase 2 consolidó la estructura funcional del sistema multi-agente descentralizado mediante la integración del aprendizaje por refuerzo profundo en el agente líder y el control cooperativo de los seguidores. El resultado fue una arquitectura capaz de coordinar varios robots en tiempo real, demostrando la viabilidad del enfoque propuesto.

5.3. Fase 3: Implementación y validación en entorno de simulación

Esta fase describe el proceso de implementación y validación del sistema multi-agente descentralizado dentro del entorno de simulación *Gazebo*. El objetivo principal fue comprobar el funcionamiento coordinado entre el agente líder, entrenado mediante aprendizaje por refuerzo profundo, y los agentes seguidores controlados con el algoritmo *Follower*. Para ello, se desarrolló un entorno virtual que permitió replicar las condiciones reales de operación del sistema y verificar la interacción entre los distintos nodos del proyecto.

5.3.1. Configuración del entorno de simulación en Gazebo

El simulador *Gazebo* se utilizó como plataforma de validación, aprovechando su integración con *ROS 2 Humble*. En esta fase se trabajó con dos configuraciones principales:

- **Entrenamiento individual del agente líder:** a través del archivo `turtlebot3_dqn_stage2.launch.py`, que inicializa un único robot *TurtleBot3 Burger* en un entorno cerrado con obstáculos para ejecutar episodios de aprendizaje profundo.
- **Validación multi-robot:** mediante el archivo `multi_robot.launch.py`, que lanza simultáneamente tres robots con espacios de nombres independientes (`/TB3_1`, `/TB3_2` y `/TB3_3`) y posiciones iniciales definidas. Cada robot carga su propio *state publisher* y nodos de control, garantizando la independencia de sus tópicos y transformaciones.

La inicialización del entorno multi-robot en Gazebo se realiza a través del archivo `multi_robot.launch.py`, el cual lanza tres instancias del robot *TurtleBot3 Burger* con espacios de nombres independientes y configuraciones personalizadas. El siguiente comando ejecuta el proceso completo desde la terminal:

```
ros2 launch turtlebot3_gazebo multi_robot.launch.py
```

Este archivo de lanzamiento crea un entorno donde los nodos `gzserver` y `gzclient` inicializan el simulador, mientras que los nodos `spawn_turtlebot3` y `robot_state_publisher` se encargan de instanciar cada robot en su respectiva posición inicial. De esta forma, se obtiene un entorno de simulación con múltiples agentes funcionando de manera independiente bajo una arquitectura descentralizada.

5.3.2. Generación dinámica de metas en el entorno (`dqn_gazebo.py`)

El script `dqn_gazebo.py` es el encargado de generar y actualizar dinámicamente las metas (goal boxes) dentro del entorno de simulación. Este nodo actúa como una interfaz entre *ROS 2* y Gazebo, proporcionando servicios para inicializar, reiniciar y modificar las metas en tiempo de ejecución.

Cada vez que el agente alcanza un objetivo o incurre en una colisión, el nodo elimina el modelo anterior y crea una nueva meta en una posición aleatoria dentro de los límites del mapa, garantizando diversidad en las trayectorias de aprendizaje. En la siguiente figura se muestra un ejemplo visual del punto de destino generado en Gazebo, que sirve como referencia para que el agente líder complete su tarea de navegación durante el proceso de entrenamiento.

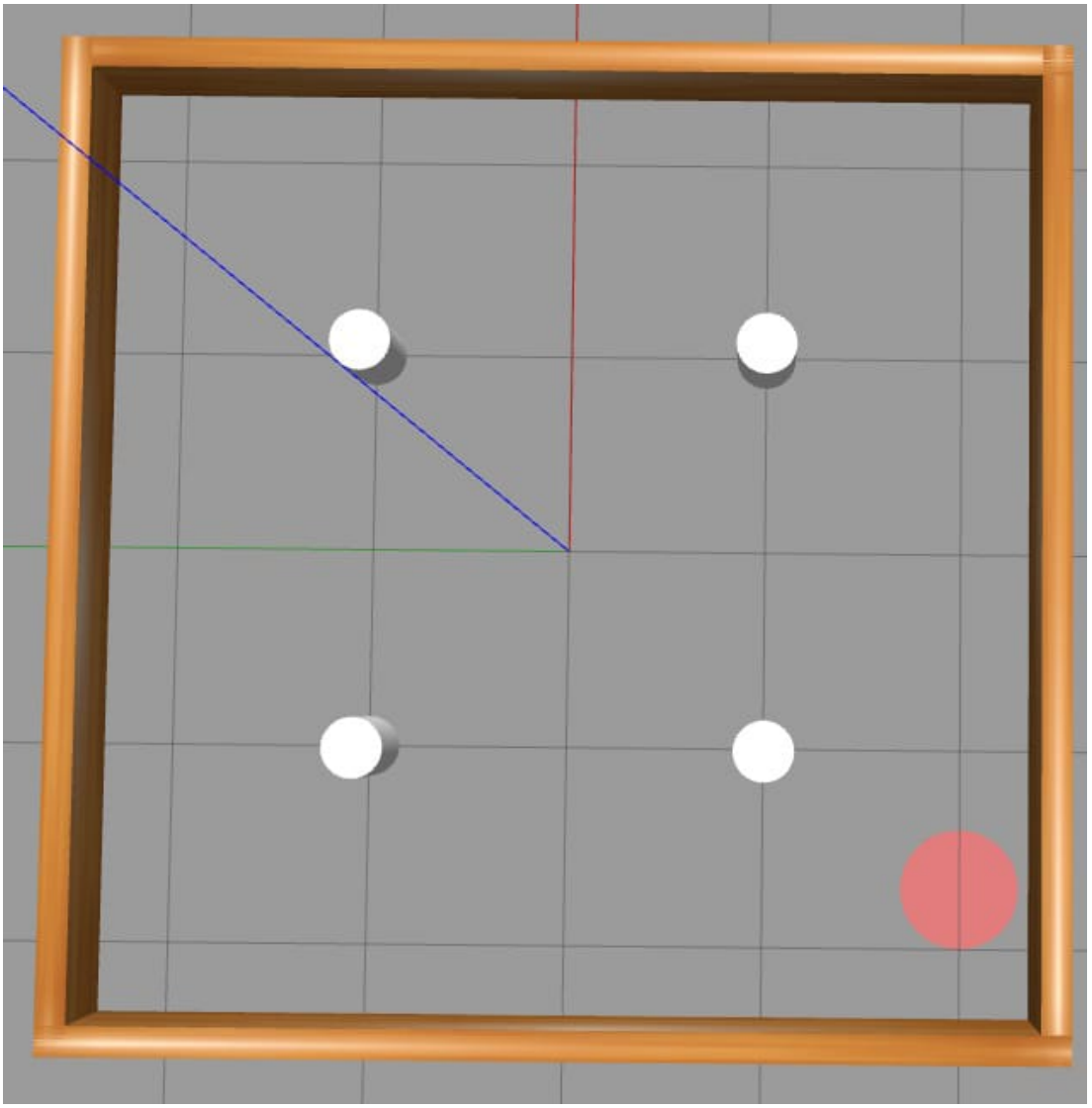

FIGURA 2: Ejemplo del punto de destino (goal) en Gazebo utilizado durante el entrenamiento.

El nodo define servicios como `initialize_env`, `task_succeed` y `task_failed`, los cuales son llamados por el entorno de entrenamiento para actualizar el escenario según los resultados de cada episodio. Esto permite automatizar el ciclo de aprendizaje sin intervención manual.

5.3.3. Control del entorno de entrenamiento (`dqn_environment.py`)

El archivo `dqn_environment.py` constituye el entorno de aprendizaje del agente. Su función es gestionar la interacción con Gazebo, calcular las recompensas, determinar los estados y evaluar el éxito o fracaso de cada episodio. El nodo se comunica con los tópicos `/scan` (LIDAR) y `/odom` (odometría), además de publicar comandos de velocidad lineal y angular en `/cmd_vel`; utiliza la información del sensor LIDAR para construir el vector de estado, compuesto por 26 mediciones que describen la distancia a los obstáculos.

Para visualizar la información captada por el sensor LIDAR y cómo esta define el estado del agente, en la Figura 3 se muestra la percepción del robot dentro del entorno simulado. Las zonas azules representan los rayos del escáner láser y las superficies detectadas, permitiendo identificar obstáculos cercanos y regiones libres para la navegación.

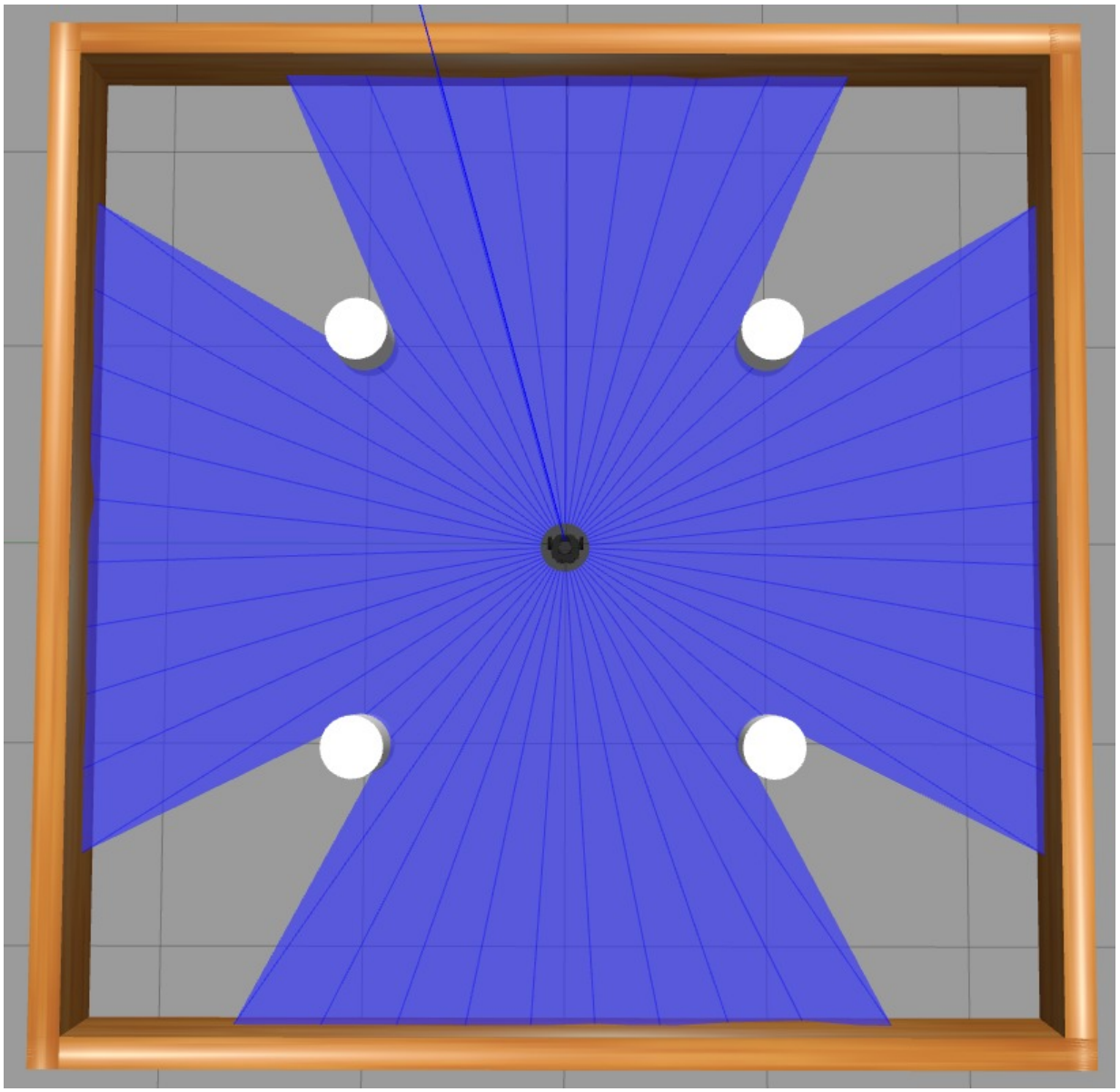

FIGURA 3: Percepción del entorno mediante el sensor LIDAR del agente líder en Gazebo.

Durante el entrenamiento, el entorno calcula la recompensa total combinando:

- una recompensa direccional (r_{yaw}) que premia al agente cuando se orienta hacia la meta, y

- una penalización por proximidad a obstáculos (r_{obs}), que disminuye el valor de recompensa cuando el robot se acerca demasiado a objetos del entorno.

Los estados de colisión o de éxito se determinan en función de los valores LIDAR y la distancia a la meta. Cuando el agente llega al objetivo, se ejecuta el servicio `task_succeed`; en caso contrario, si ocurre una colisión o el episodio supera un tiempo límite, se llama a `task_failed`.

La Figura 4 muestra diferentes instancias de la simulación donde el entorno evalúa el desempeño del agente, alternando entre episodios exitosos y colisiones.

```
daniel-tesis@danieltesis:~$ ros2 run turtlebot3_dqn dqn_environment
[INFO] [1762631359.910895438] [rl_environment]: Make environment called
[INFO] [1762631360.539653118] [rl_environment]: goal initialized at [0.500000, 0.000000]
[INFO] [1762631437.663259000] [rl_environment]: Make environment called
[INFO] [1762631438.563739802] [rl_environment]: goal initialized at [0.500000, 0.000000]
directional_reward: 0.999999, obstacle_reward: 0.000000
directional_reward: 1.000000, obstacle_reward: 0.000000
directional_reward: 1.000000, obstacle_reward: 0.000000
directional_reward: 0.994970, obstacle_reward: 0.000000
directional_reward: 0.956405, obstacle_reward: 0.000000
directional_reward: 0.970798, obstacle_reward: 0.000000
directional_reward: 0.990059, obstacle_reward: 0.000000
directional_reward: 0.950059, obstacle_reward: 0.000000
directional_reward: 0.911803, obstacle_reward: 0.000000
directional_reward: 0.839604, obstacle_reward: 0.000000
directional_reward: 0.820346, obstacle_reward: 0.000000
```

FIGURA 4: Ejecución del entorno de entrenamiento: detección de colisiones y finalización de episodios exitosos.

El registro en consola confirma el funcionamiento del cálculo de recompensas y los estados alcanzados durante la simulación, como se aprecia en los mensajes:

```
directional_reward: 0.999999, obstacle_reward: 0.000000
directional_reward: 0.294729, obstacle_reward: -4.423280
Goal Reached
Collision happened
```

5.3.4. Agente DQN y proceso de aprendizaje (`dqn_agent.py`)

El archivo `dqn_agent.py` define la estructura del agente que implementa el algoritmo Deep Q-Learning. Este nodo mantiene la comunicación con el entorno de entrenamiento a través de los servicios `make_environment`, `reset_environment` y `rl_agent_interface`.

La red neuronal utilizada por el agente está compuesta por cuatro capas densas con 512, 256, 128 y 5 neuronas. Cada capa emplea la función de activación *ReLU*, mientras que la capa de salida es lineal y representa los valores $Q(s, a)$ asociados a cada acción posible. El modelo se entrena utilizando el optimizador *Adam* y la función de error cuadrático medio (MSE). Durante el proceso de aprendizaje, el agente interactúa con el entorno y almacena sus experiencias como transiciones (s, a, r, s'). A partir de estas, selecciona muestras aleatorias que usa para ajustar los pesos de la red, logrando un aprendizaje más estable y eficiente.

La Figura 5 muestra el registro del entrenamiento en consola, donde se aprecia la reducción progresiva del parámetro de exploración ϵ y la mejora en el rendimiento del agente a lo largo de los episodios.

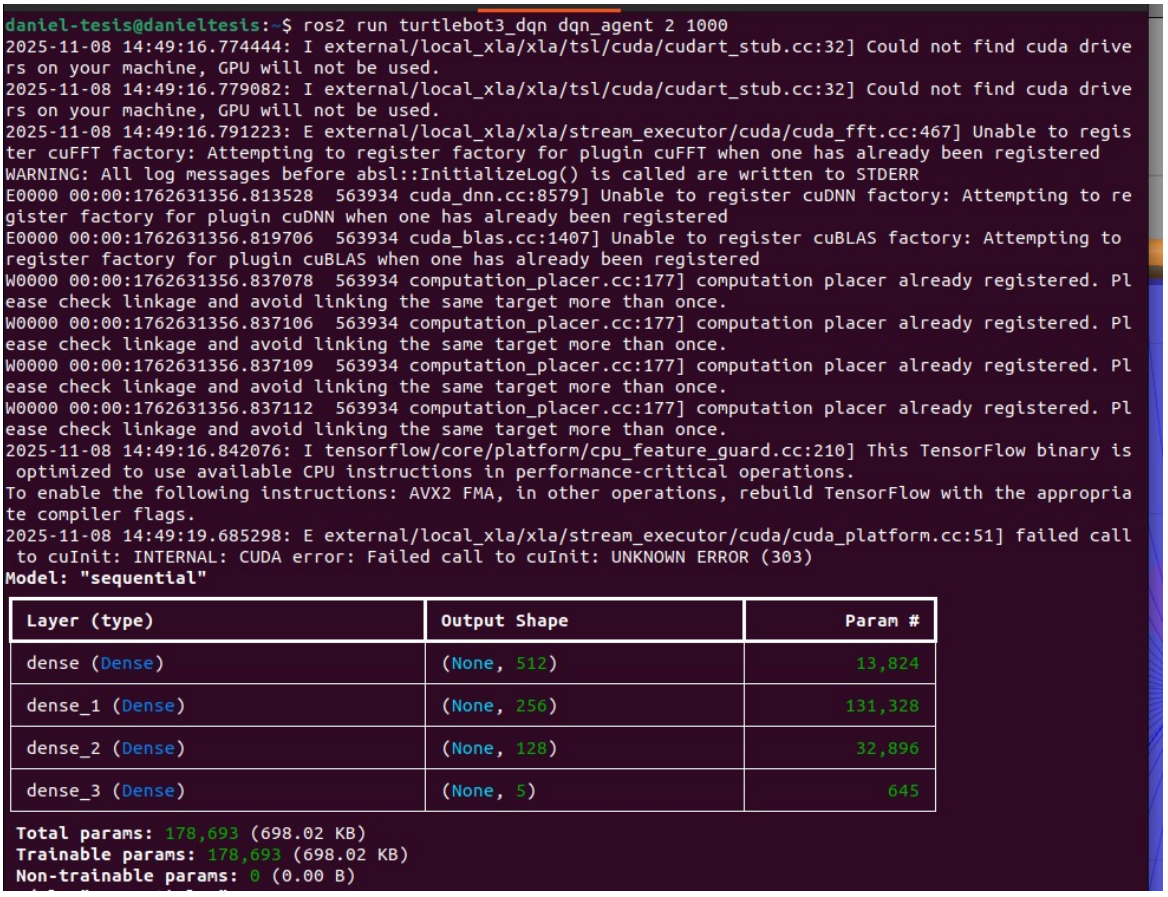

FIGURA 5: Evolución del entrenamiento del agente DQN en consola.

Una vez alcanzada la convergencia, el modelo entrenado se guarda en un archivo HDF5 (por ejemplo, `stage2_episode1000.h5`), que contiene los pesos finales de la red neuronal. Este modelo se carga posteriormente durante la fase de inferencia para controlar al agente líder en el entorno multi-robot.

La Figura 6 presenta la estructura del modelo DQN generada por Keras, donde se detalla la configuración de las capas, la cantidad de parámetros entrenables y la función de pérdida utilizada.

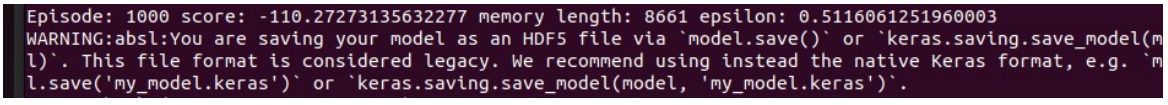

FIGURA 6: Resumen de la arquitectura de la red neuronal DQN.

5.3.5. Integración y funcionamiento del sistema

Una vez finalizado el entrenamiento, el modelo DQN se utilizó únicamente en el agente líder, mientras que los demás robots, denominados agentes seguidores, ejecutaron el algoritmo *Follower* desarrollado por ROBOTIS. De esta forma, el líder toma decisiones de manera autónoma usando lo aprendido durante el entrenamiento, y los seguidores reproducen su movimiento manteniendo una distancia constante.

5.3.5.1. Validación del modelo entrenado (`dqn_test.py`)

El archivo `dqn_test.py` se utilizó para comprobar el funcionamiento del modelo entrenado en el entorno de simulación. Este código carga los pesos del modelo (por ejemplo, `stage2_episode1000.h5`) y los aplica directamente en modo de *inferencia*, es decir, ejecutando el comportamiento aprendido sin continuar el proceso de entrenamiento.

Durante esta fase, el agente líder aplica lo aprendido para tomar decisiones en tiempo real según las lecturas de sus sensores. El nodo se comunica con el servicio `rl_agent_interface`, que permite enviar las acciones calculadas por el modelo y recibir la información actualizada del entorno, junto con la recompensa obtenida y el estado del episodio (si continúa, termina o se reinicia).

El proceso sigue un ciclo continuo que:

1. Inicializa el entorno y obtiene el estado inicial del robot.
2. Calcula la acción más adecuada mediante el método `get_action()`.
3. Envía la acción al entorno y recibe una nueva observación junto con la recompensa.
4. Repite estos pasos hasta que el robot llega al objetivo o el episodio finaliza.

El modelo conserva la misma estructura de red neuronal utilizada durante el entrenamiento, con cuatro capas densas de 512, 256, 128 y 5 neuronas. En esta etapa solo se usa para tomar decisiones, por lo que no se realizan ajustes ni actualizaciones en sus parámetros. Esto permite evaluar el comportamiento del agente en condiciones reales de simulación, comprobando su capacidad para desplazarse, evitar obstáculos y alcanzar el objetivo sin necesidad de reentrenamiento.

La Figura 7 muestra el resultado de esta validación, donde el agente líder aplica el modelo entrenado mientras los robots seguidores imitan su movimiento usando el algoritmo *Follower*, dentro de una arquitectura descentralizada.

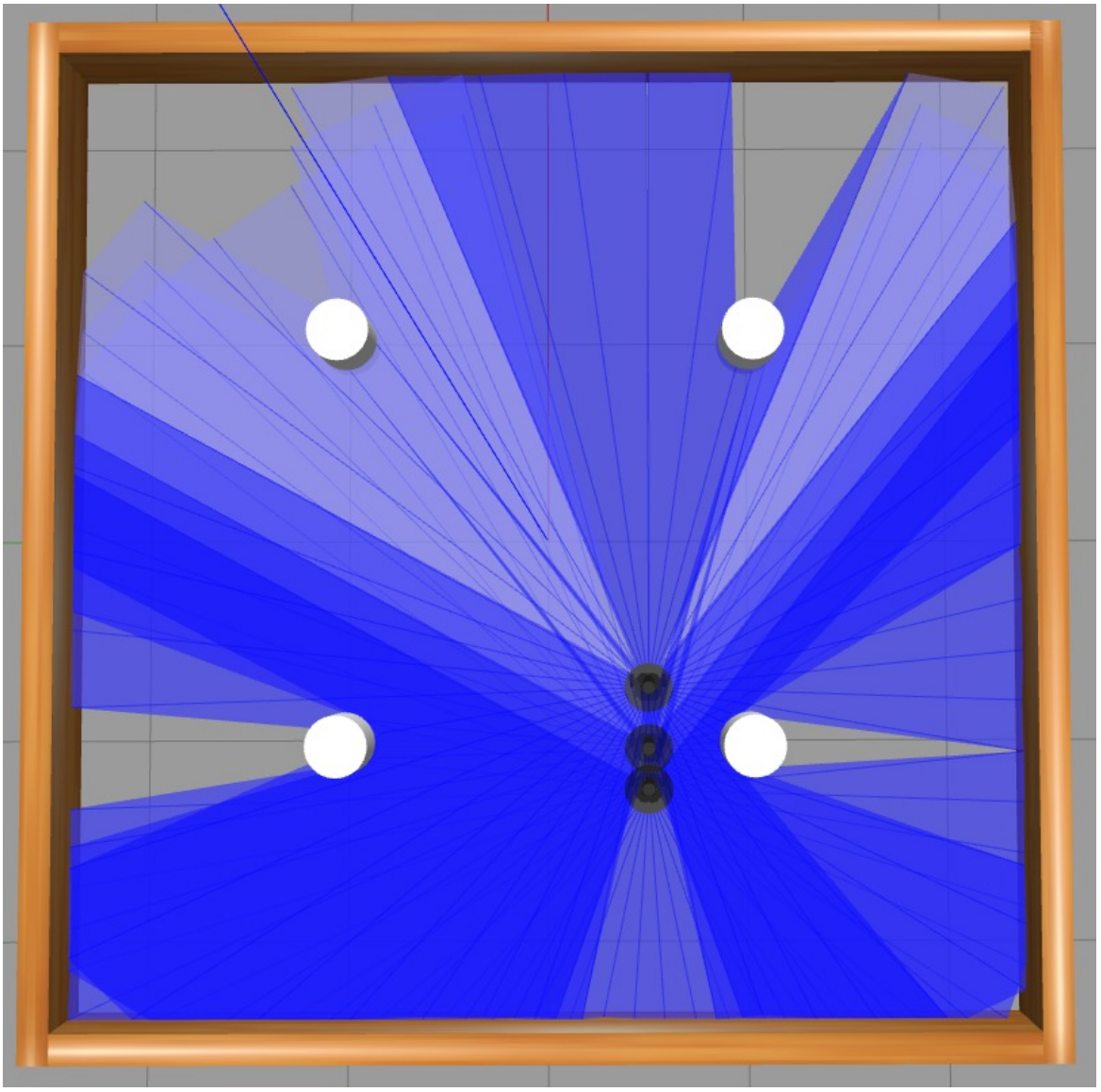

FIGURA 7: Simulación del sistema multi-agente descentralizado en Gazebo: líder entrenado con DQN y seguidores con algoritmo Follower.

Esta configuración permitió validar en simulación la cooperación entre los agentes y la correcta comunicación entre los nodos desarrollados, verificando la estabilidad del sistema antes de su aplicación en la plataforma física.

Esta fase permitió implementar y validar en simulación la arquitectura propuesta del sistema multi-agente descentralizado. A través de la integración del algoritmo DQN en el agente líder y del algoritmo *Follower* en los agentes secundarios, se logró un comportamiento cooperativo coherente y estable dentro del entorno virtual. Además, el uso del simulador Gazebo facilitó la verificación de la comunicación entre los nodos, la generación dinámica de metas y el control simultáneo de los robots en tiempo real; Estos resultados sentaron las bases para la siguiente sección del documento, dedicada al análisis y evaluación de los resultados obtenidos, donde se examina el desempeño del sistema.

Capítulo 6

Resultados y Discusión

En esta sección se presentan los resultados obtenidos durante la fase de entrenamiento del agente líder mediante el algoritmo *Deep Q-Learning (DQN)* en el entorno simulado de *Gazebo*. El objetivo principal de este análisis es evaluar la evolución del aprendizaje del agente en función de la recompensa acumulada a lo largo de los episodios de entrenamiento, verificando su capacidad para mejorar su decisión de navegación y evitar colisiones con los obstáculos del entorno.

Los resultados se registraron y visualizaron empleando la herramienta TensorBoard, permitiendo observar la tendencia de las recompensas en diferentes intervalos del proceso de entrenamiento (de 0 a 1000 episodios). A partir de estas gráficas se analizan la progresión del aprendizaje, la estabilidad del modelo, y la convergencia de la política entrenada.

En los apartados siguientes se muestran las recompensas promedio obtenidas en distintos rangos de episodios, seguidas por una gráfica suavizada que evidencia la tendencia general del aprendizaje del agente.

6.1. Evolución de las recompensas durante el entrenamiento

Durante el entrenamiento del agente líder, se registraron las recompensas obtenidas en cada episodio para analizar el comportamiento del algoritmo *Deep Q-Learning (DQN)* a lo largo de 1000 iteraciones. A continuación, se presentan los resultados segmentados por intervalos de 100 episodios, con el fin de observar la evolución del aprendizaje y la estabilidad del modelo en distintas etapas del proceso.

6.1.1. Episodios 0–100

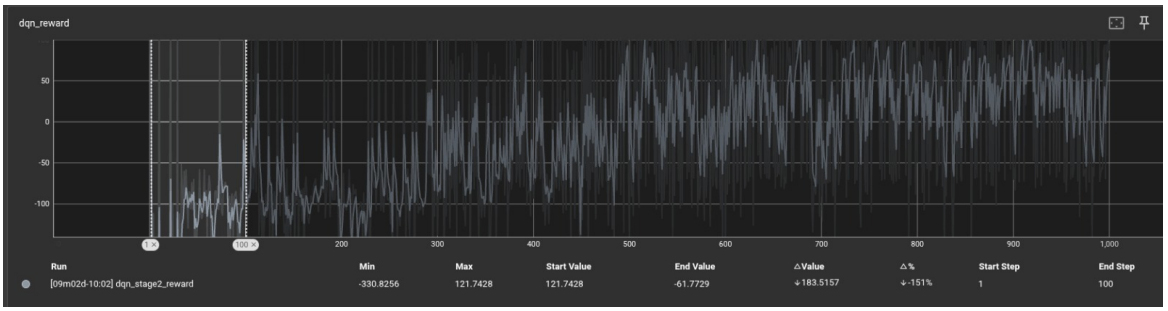

FIGURA 8: Evolución de la recompensa acumulada entre los episodios 0 y 100.

En la primera etapa del entrenamiento se observa una alta variabilidad en las recompensas, con valores predominantemente negativos que llegan hasta aproximadamente -330. Esto indica que el agente se encuentra en una fase inicial de exploración, en la cual todavía no ha aprendido a evitar colisiones ni a orientarse hacia la meta. Los picos negativos frecuentes reflejan penalizaciones derivadas de choques o acciones ineficientes, lo cual es esperado en las primeras iteraciones del aprendizaje por refuerzo.

6.1.2. Episodios 0–200

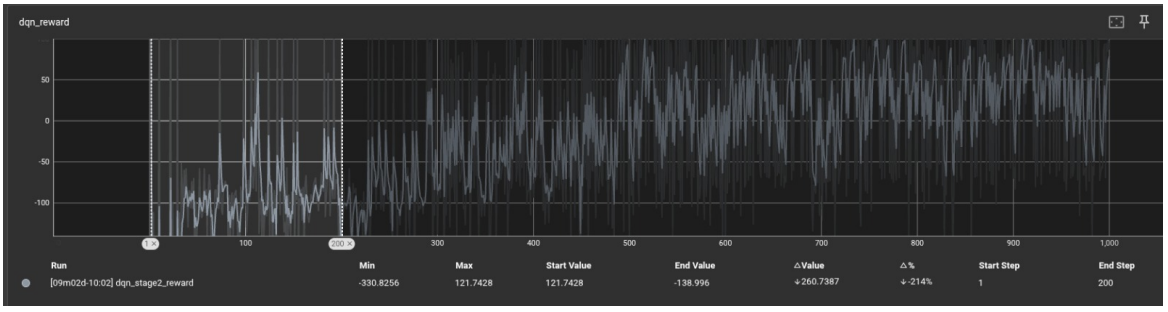

FIGURA 9: Evolución de la recompensa acumulada entre los episodios 0 y 200.

En el segundo tramo del entrenamiento, el agente comienza a mostrar una reducción gradual en la frecuencia de recompensas extremadamente negativas. Si bien la dispersión de valores sigue siendo alta, se evidencia una leve tendencia ascendente que sugiere una mejora en la toma de decisiones. Esto se debe a que el modelo va ajustando progresivamente sus pesos y disminuye la cantidad de acciones aleatorias, concentrándose cada vez más en las decisiones que le han dado mejores resultados en experiencias anteriores.

6.1.3. Episodios 0–300

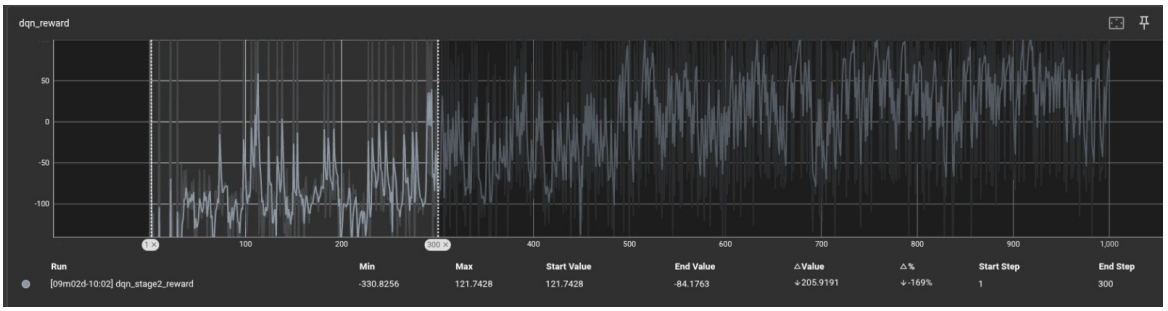

FIGURA 10: Evolución de la recompensa acumulada entre los episodios 0 y 300.

A medida que avanza el entrenamiento hacia los 300 episodios, las recompensas comienzan a estabilizarse alrededor de valores más cercanos a cero. Esto indica que el agente ha adquirido una mejor comprensión del entorno y logra completar trayectorias sin colisionar en algunos intentos. El incremento en la frecuencia de recompensas positivas demuestra que el modelo empieza a identificar patrones óptimos de navegación, aunque aún se presentan fluctuaciones asociadas al proceso de exploración.

6.1.4. Episodios 0–400

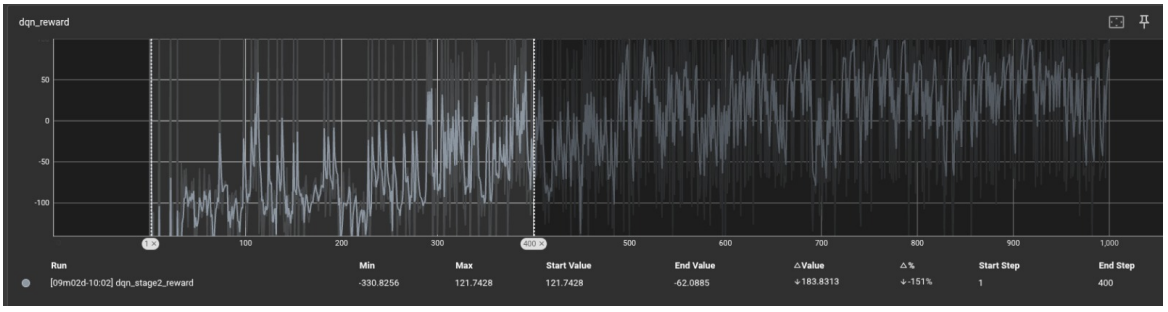

FIGURA 11: Evolución de la recompensa acumulada entre los episodios 0 y 400.

En este punto, la curva de recompensas presenta una tendencia más consistente hacia valores positivos, con una reducción significativa de los mínimos alcanzados. El agente muestra una mayor estabilidad en su comportamiento, evidenciando un progreso notable en la ejecución de trayectorias seguras y eficientes. Esta etapa marca el inicio de la convergencia parcial del modelo, donde las actualizaciones de los pesos comienzan a consolidar una política más efectiva.

6.1.5. Episodios 0–500

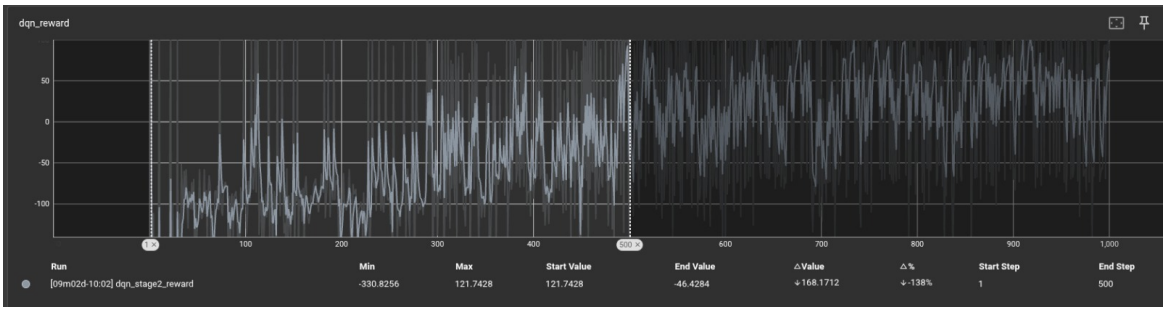

FIGURA 12: Evolución de la recompensa acumulada entre los episodios 0 y 500.

Al llegar a los 500 episodios, el agente evidencia una mejoría sostenida en su desempeño. Las recompensas negativas se presentan con menor frecuencia y magnitud, mientras que los valores positivos comienzan a ser más comunes. Esto indica que el modelo ha logrado una comprensión más estable del entorno, aprendiendo a evitar colisiones y a orientarse hacia la meta con mayor consistencia. En esta etapa, el agente muestra una transición clara desde la exploración inicial hacia un comportamiento más guiado por la experiencia acumulada, lo que refleja un proceso de aprendizaje efectivo.

6.1.6. Episodios 0–600

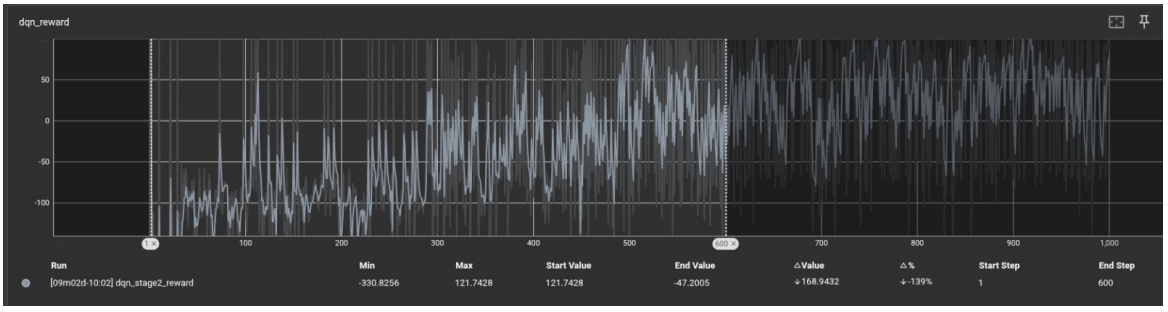

FIGURA 13: Evolución de la recompensa acumulada entre los episodios 0 y 600.

En el rango de 0 a 600 episodios, las recompensas se estabilizan dentro de un intervalo más estrecho, con una tendencia general ascendente. El agente demuestra una mayor capacidad de navegación autónoma, completando trayectorias sin colisiones con mayor frecuencia. Aunque aún se observan fluctuaciones puntuales, estas corresponden a intentos de exploración residuales que contribuyen a refinar la política aprendida. El comportamiento global del modelo sugiere que el aprendizaje ha alcanzado una fase de consolidación.

6.1.7. Episodios 0–700

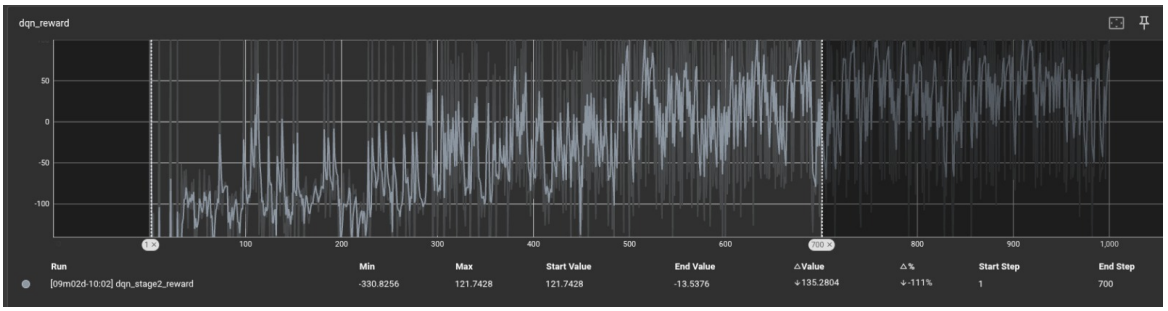

FIGURA 14: Evolución de la recompensa acumulada entre los episodios 0 y 700.

A partir de los 700 episodios, se aprecia una tendencia positiva más marcad en las recompensas promedio. El agente ha adquirido una estrategia más eficiente para avanzar hacia el objetivo evitando obstáculos, lo que se traduce en una mayor frecuencia de episodios exitosos. Las caídas ocasionales en la curva de recompensa reflejan situaciones de exploración o ajustes menores del modelo, pero no afectan la estabilidad general del aprendizaje.

6.1.8. Episodios 0–800

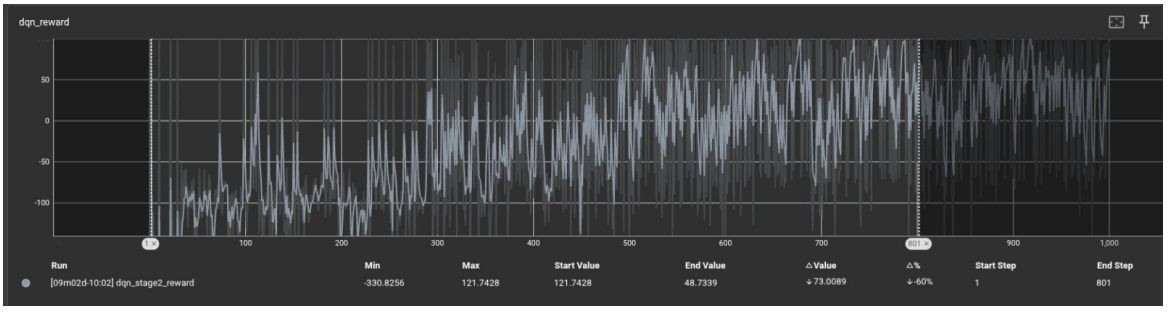

FIGURA 15: Evolución de la recompensa acumulada entre los episodios 0 y 800.

En esta etapa, el desempeño del agente alcanza una mayor consistencia, con recompensas promedio positivas sostenidas. El modelo demuestra haber internalizado patrones de comportamiento que le permiten desplazarse con eficacia hacia la meta sin incurrir en colisiones frecuentes. La disminución de la variabilidad en los valores obtenidos refuerza la idea de que el agente ha logrado un nivel de aprendizaje estable y repetible dentro del entorno simulado.

6.1.9. Episodios 0–900

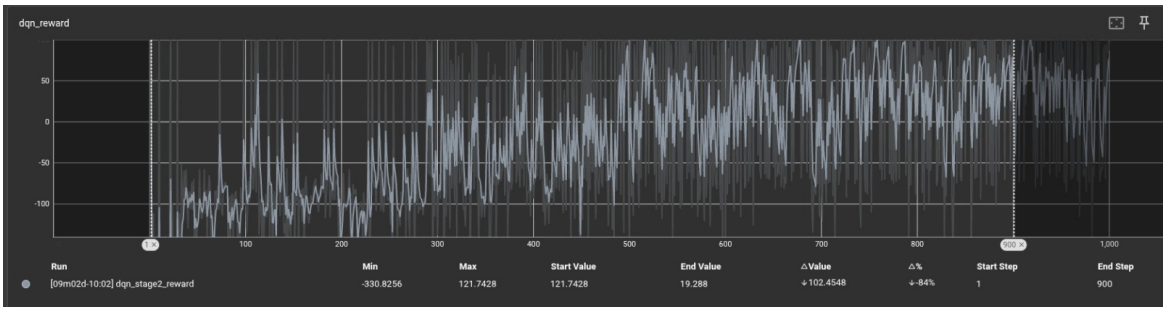

FIGURA 16: Evolución de la recompensa acumulada entre los episodios 0 y 900.

En los episodios cercanos a 900 se observa una convergencia más clara del modelo. Las recompensas tienden a estabilizarse en valores positivos, lo que indica que el agente ha aprendido a mantener trayectorias seguras y eficientes en la mayoría de los intentos. Las caídas abruptas, que eran comunes en etapas iniciales, prácticamente desaparecen, lo cual evidencia un control más fino del movimiento y una mejor reacción ante obstáculos.

6.1.10. Episodios 0–1000

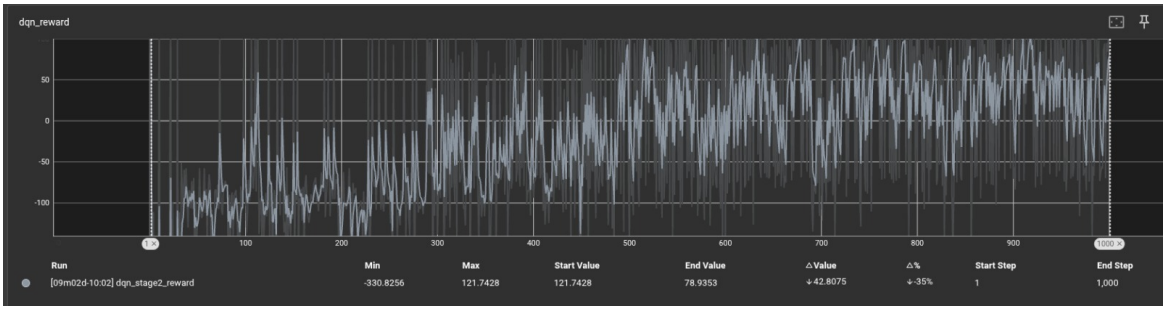

FIGURA 17: Evolución de la recompensa acumulada entre los episodios 0 y 1000.

Al completar los 1000 episodios de entrenamiento, el modelo alcanza una fase de estabilidad y madurez. Las recompensas promedio se mantienen consistentemente positivas, confirmando que el agente ha aprendido una política efectiva de navegación. El comportamiento del sistema demuestra que el aprendizaje por refuerzo ha sido exitoso: el robot es capaz de desplazarse hacia la meta evitando colisiones y optimizando su trayectoria en función de la experiencia adquirida.

6.1.11. Comparación global del entrenamiento

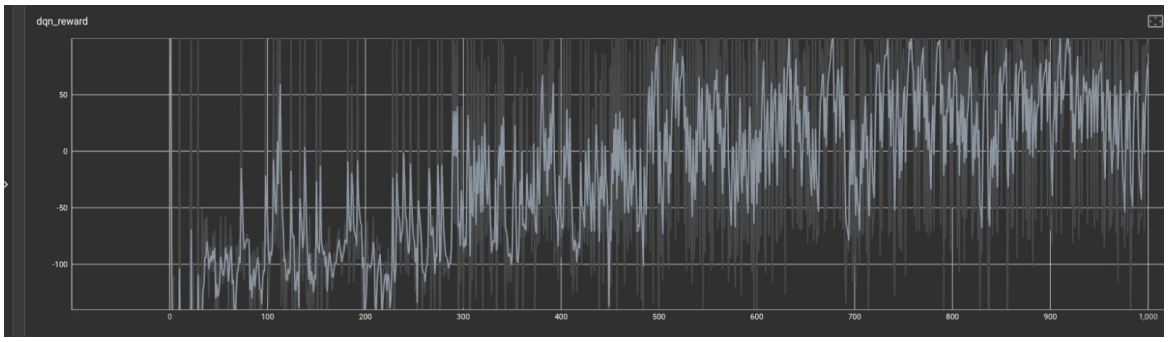

FIGURA 18: Evolución completa de las recompensas a lo largo de los 1000 episodios de entrenamiento.

La Figura 18 presenta una vista general del entrenamiento completo. Se aprecia una clara tendencia ascendente en las recompensas acumuladas, pasando de valores negativos pronunciados en las primeras fases a valores positivos estables en los últimos episodios. Esto confirma que el agente mejoró su capacidad de toma de decisiones de manera progresiva, ajustando su comportamiento en función de las recompensas obtenidas y reduciendo la frecuencia de colisiones. El aprendizaje muestra una convergencia gradual, con un punto de inflexión alrededor del episodio 600, donde el rendimiento comienza a estabilizarse.

6.1.12. Análisis del suavizado de recompensas

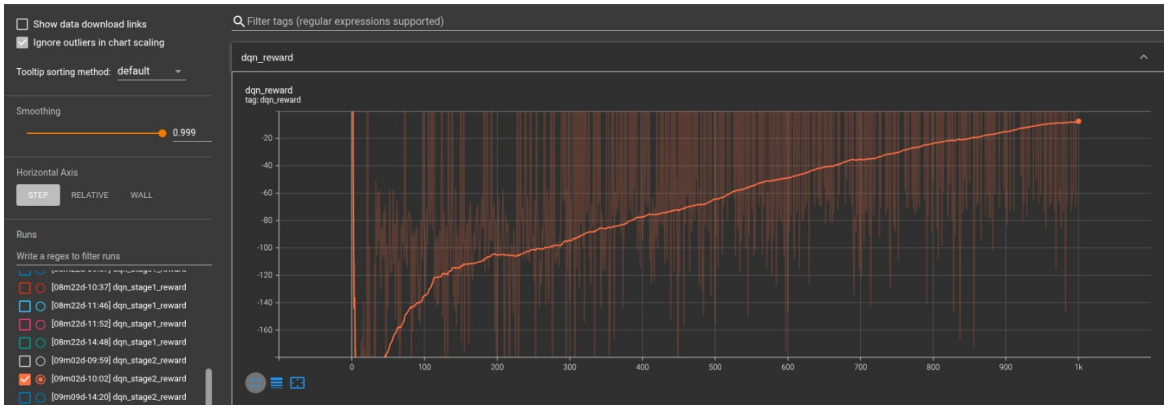

FIGURA 19: Curva suavizada de recompensas promedio durante el entrenamiento.

Para analizar el comportamiento general del aprendizaje, se aplicó un proceso de suavizado de recompensas promedio sobre los 1000 episodios. Esta representación permite observar de forma más clara la tendencia global de mejora del agente, eliminando las variaciones bruscas propias de la exploración inicial. Como se aprecia en la Figura 19, el promedio móvil de recompensas crece de manera sostenida hasta alcanzar un valor estable, lo que evidencia que el modelo logró aprender una política robusta y consistente.

En conjunto, los resultados demuestran que el agente DQN logró un aprendizaje sólido y progresivo, adquiriendo la capacidad de desplazarse de forma autónoma y segura dentro del entorno simulado. El modelo aprendió a identificar patrones efectivos de movimiento, evitando colisiones y optimizando su trayectoria hacia el objetivo.

6.2. Discusión de Resultados

El análisis de las recompensas obtenidas a lo largo de los 1000 episodios de entrenamiento permite evidenciar la eficacia del algoritmo DQN implementado en el agente líder. Durante las primeras etapas del proceso, el modelo mostró una gran variabilidad en las recompensas debido a la exploración del entorno y a la falta de experiencia previa. Sin embargo, conforme avanzó el entrenamiento, las recompensas se estabilizaron progresivamente, alcanzando valores positivos sostenidos que reflejan una mejora constante en la toma de decisiones del agente.

Esta evolución indica que el modelo logró aprender una política de navegación eficiente basada en la retroalimentación del entorno. El agente desarrolló la capacidad de desplazarse hacia la meta minimizando colisiones y manteniendo trayectorias más cortas y estables. La disminución de las oscilaciones en las recompensas demuestra que el comportamiento se volvió más predecible y seguro, lo que representa una convergencia satisfactoria del proceso de aprendizaje.

Por otro lado, el suavizado de la curva de recompensas evidencia un crecimiento continuo y sostenido, lo cual sugiere que el modelo no solo aprendió a evitar errores, sino que optimizó sus acciones a partir de la experiencia acumulada. Esto se traduce en un agente que comprende mejor su entorno y responde de forma más adecuada ante diferentes configuraciones de obstáculos y metas.

La validación posterior en el entorno multi-agente confirmó que el modelo entrenado mantuvo un comportamiento coherente al interactuar con los demás robots. El agente líder, utilizando el modelo DQN entrenado, logró navegar de forma autónoma dentro del entorno, mientras los agentes seguidores, controlados por el algoritmo *Follower*, replicaron su trayectoria con precisión y estabilidad. Esta interacción demostró la correcta comunicación entre los nodos del sistema y validó la efectividad de la arquitectura descentralizada implementada.

En términos generales, los resultados confirman que el uso de aprendizaje por refuerzo profundo en combinación con una estructura multi-agente descentralizada es una estrategia viable para desarrollar comportamientos cooperativos en robots móviles. El entrenamiento permitió que un único agente aprendiera la política de navegación, la cual fue posteriormente reproducida por los demás robots, reduciendo la carga computacional y manteniendo la coherencia del sistema. Con esto se establece una base sólida para la siguiente etapa de validación experimental en plataforma física.

Con esto se establece una base sólida para la siguiente etapa de validación experimental en plataforma física.

6.2.1. Validación en plataforma física

Finalmente, se realizó una prueba experimental en la plataforma física utilizando tres robots TurtleBot3 Burger en un entorno controlado que replicó las condiciones de la simulación (Figura 20 y Figura 21). El escenario incluyó obstáculos dispuestos estratégicamente para evaluar la capacidad de navegación autónoma del agente entrenado y la coordinación descentralizada entre los robots.

Durante la ejecución, se observó que el agente líder, previamente entrenado en simulación mediante el algoritmo DQN, mantuvo trayectorias estables evitando colisiones y reaccionando adecuadamente ante los obstáculos. Los agentes seguidores reprodujeron de forma sincronizada el comportamiento aprendido, ajustando sus movimientos en función de la información local percibida por sus propios sensores, lo que evidencia la correcta transferencia de la política de navegación en la arquitectura multi-agente descentralizada.

Aunque se detectaron pequeñas desviaciones debido al ruido sensorial y a la fricción del terreno, el sistema demostró un desempeño coherente con los resultados obtenidos en el entorno virtual, validando la viabilidad del enfoque propuesto para su aplicación en escenarios reales.

FIGURA 20: Disposición inicial de los robots TurtleBot3 en el entorno físico.

FIGURA 21: Posición final tras la navegación autónoma multi-agente.

Capítulo 7

Conclusiones y Trabajos Futuros

7.1. Conclusiones

El desarrollo del proyecto permitió diseñar e implementar una arquitectura multi-agente descentralizada basada en ROS 2, capaz de coordinar el comportamiento de varios robots móviles mediante aprendizaje por refuerzo profundo. Los resultados obtenidos evidencian que la combinación de este tipo de algoritmos con sistemas distribuidos constituye una alternativa viable y eficiente para el desarrollo de sistemas robóticos cooperativos.

En relación con el primer objetivo específico, orientado a identificar una plataforma adecuada para la implementación, se concluye que el TurtleBot3 Burger fue la opción más conveniente dentro de las condiciones del laboratorio. Su integración nativa con ROS 2, el uso de sensores LIDAR de 360°, su arquitectura modular y su facilidad de programación en *Python* permitieron desarrollar y validar el sistema con un alto grado de compatibilidad y estabilidad. Además, su bajo costo y tamaño compacto facilitaron la realización de múltiples pruebas en entornos simulados sin comprometer la precisión de los resultados.

Respecto al segundo objetivo, relacionado con el diseño del algoritmo multi-agente, se logró construir una arquitectura descentralizada funcional en la que cada robot opera bajo su propio espacio de nombres, sin depender de un nodo central. El agente líder, entrenado con el algoritmo *Deep Q-Learning (DQN)*, alcanzó un comportamiento autónomo y eficiente, mientras que los agentes seguidores, controlados mediante el algoritmo *Follower* de ROBOTIS, reprodujeron sus trayectorias con estabilidad y sincronización. Esta estructura confirmó la viabilidad del enfoque descentralizado, garantizando comunicación efectiva y coordinación dinámica entre los nodos del sistema.

En cuanto al tercer objetivo, centrado en la implementación del sistema de navegación multi-agente, se consiguió desarrollar un entorno de simulación en *Gazebo* que permitió validar la interacción entre los agentes. Durante el entrenamiento, el modelo DQN mostró una mejora continua en las recompensas acumuladas y una reducción significativa de colisiones, lo que demuestra que el agente fue capaz de aprender estrategias de navegación óptimas. El comportamiento observado reflejó una convergencia estable del modelo y una correcta adaptación al entorno simulado.

Finalmente, en relación con el cuarto objetivo, orientado a la validación funcional del sistema, se comprobó que el modelo entrenado mantuvo su desempeño al interactuar con los agentes seguidores. El líder ejecutó su política aprendida de manera consistente, mientras los seguidores lograron mantener la distancia adecuada y replicar sus movimientos con precisión. Estos resultados confirman la efectividad del diseño propuesto y

demuestran que es posible implementar un sistema multi-agente descentralizado capaz de cooperar sin la necesidad de un controlador central, garantizando escalabilidad y robustez frente a fallos individuales.

En conjunto, el proyecto contribuye al avance del estado del arte en sistemas robóticos cooperativos, al demostrar que la integración de aprendizaje por refuerzo profundo (DQN) y arquitecturas descentralizadas basadas en ROS 2 constituye un enfoque sólido para desarrollar comportamientos colectivos inteligentes en robots móviles. Este trabajo sienta las bases para futuras implementaciones físicas y aplicaciones en escenarios reales, como exploración, monitoreo ambiental y logística autónoma.

7.2. Trabajos Futuros

A partir de los resultados obtenidos, se identifican diversas líneas de trabajo que podrían fortalecer los aportes de este proyecto. En primer lugar, se propone mejora de la implementación física del sistema, optimizando el desempeño del modelo DQN previamente entrenado al aplicarlo sobre los robots TurtleBot3 reales. Esto incluye el ajuste de parámetros de control, la reducción del impacto del ruido sensorial y la evaluación del sistema bajo diferentes condiciones de iluminación, espacio y comunicación inalámbrica. El objetivo es fortalecer la robustez del sistema frente a las variaciones inherentes del entorno físico y garantizar un comportamiento estable y coherente con respecto al observado en simulación.

En segundo lugar, se plantea la extensión del sistema a escenarios dinámicos, incorporando obstáculos móviles y entornos cambiantes que obliguen a los agentes a readaptar sus estrategias de navegación, evaluando así la capacidad de generalización del modelo.

De igual forma, se sugiere explorar un enfoque de aprendizaje multi-agente cooperativo, en el cual varios agentes aprendan de manera simultánea políticas de navegación complementarias, permitiendo la toma de decisiones distribuida y la cooperación autónoma.

Por otro lado, la optimización de la función de recompensa y la arquitectura de la red neuronal podría mejorar la velocidad de convergencia y la estabilidad del entrenamiento, especialmente al escalar el sistema a un mayor número de robots.

Finalmente, una línea prometedora consiste en la integración de percepción avanzada, combinando sensores de visión como cámaras RGB-D con el aprendizaje por refuerzo para abordar tareas más complejas, como la detección de objetos, el reconocimiento de entornos o la navegación visual guiada.

Bibliografía

[1] S. Ajwad, «Distributed control of multi-agent systems under communication constraints : application to robotics», Tesis doct., sep. de 2020.

[2] «GED - Grupo de Estudio y Desarrollo en Robótica», dirección: https://scienti.minciencias.gov.co/gruplac/jsp/visualiza/visualizagr.jsp?nro=00000000002964.

[3] J. Park, R. Delgado y B. Choi, «Real-Time Characteristics of ROS 2.0 in Multiagent Robot Systems: An Empirical Study», *IEEE Access*, vol. PP, págs. 1-1, ago. de 2020. DOI: 10.1109/ACCESS.2020.3018122.

[4] M. Zaryouli, M. T. Fathi y M. Ezziyyani, «Data collection based on multi-agent modeling for intelligent and precision farming in lokoss region morocco», en *2020 1st International Conference on Innovative Research in Applied Science, Engineering and Technology (IRASET)*, 2020, págs. 1-6. DOI: 10.1109/IRASET48871.2020.9092214.

[5] P. Vanella, *Implementation of ROS-based multi-agent slam centralized and decentralized approaches*, dic. de 2023. dirección: https://webthesis.biblio.polito.it/29366/.

[6] J. R. Marden, G. Arslan y J. S. Shamma, «Cooperative Control and Potential Games», *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, n.º 6, págs. 1393-1407, 2009. DOI: 10.1109/TSMCB.2009.2017273.

[7] S. Hoet y N. Sabouret, «Reinforcement Learning of Communication in a Multi-agent Context», en *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, vol. 2, 2011, págs. 240-243. DOI: 10.1109/WI-IAT.2011.125.

[8] J. Wang, J. Wu y X. Kong, «Multi-agent Simulation for Strategic Bidding in Electricity Markets Using Reinforcement Learning», *CSEE Journal of Power and Energy Systems*, vol. 9, n.º 3, págs. 1051-1065, 2023. DOI: 10.17775/CSEEJPES.2020.02820.

[9] D. Martínez y E. Mojica-Nava, «Distortion based potential game for distributed coverage control», *Information Sciences*, vol. 600, págs. 209-225, 2022, ISSN: 0020-0255. DOI: https://doi.org/10.1016/j.ins.2022.03.090. dirección: https://www.sciencedirect.com/science/article/pii/S0020025522003176.

[10] G. Cardona y J. Calderon, «Robot Swarm Navigation and Victim Detection Using Rendezvous Consensus in Search and Rescue Operations», *Applied Sciences*, vol. 9, pág. 1702, abr. de 2019. DOI: 10.3390/app9081702.

[11] «Objetivo 9: Construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación», dirección: https://www.un.org/sustainabledevelopment/es/infrastructure/.

[12] «Objetivo 11: Lograr que las ciudades sean más inclusivas, seguras, resilientes y sostenibles», dirección: https://www.un.org/sustainabledevelopment/es/cities/.

[13] «Objetivo 6: Garantizar la disponibilidad de agua y su gestión sostenible y el saneamiento para todos», dirección: https://www.un.org/sustainabledevelopment/es/water-and-sanitation/.

[14] «Objetivo 15: Gestionar sosteniblemente los bosques, luchar contra la desertificación, detener e invertir la degradación de las tierras, detener la pérdida de biodiversidad», dirección: https://www.un.org/sustainabledevelopment/es/biodiversity/.

[15] C. Escribano García-Machín, «Leader-Follower Decentraliced Control of a Nanoquadrotor Swarm», 2019.

[16] M.-L. Li, S. Chen y J. Chen, «Adaptive Learning: A New Decentralized Reinforcement Learning Approach for Cooperative Multiagent Systems», *IEEE Access*, vol. 8, págs. 99 404-99 421, 2020. DOI: 10.1109/ACCESS.2020.2997899.

[17] T. Ikeda y T. Shibuya, «Centralized Training with Decentralized Execution Reinforcement Learning for Cooperative Multi-agent Systems with Communication Delay», en *2022 61st Annual Conference of the Society of Instrument and Control Engineers (SICE)*, 2022, págs. 135-140. DOI: 10.23919/SICE56594.2022.9905866.

[18] W. Li, Y. Deng, M. Zhang, J. Li, S. Chen y S. Zhang, «Integrated Multistage Self-Healing in Smart Distribution Grids Using Decentralized Multiagent», *IEEE Access*, vol. 9, págs. 159 081-159 092, 2021. DOI: 10.1109/ACCESS.2021.3131214.

[19] G. Xu, Z. Yang, W. Lu y L. Zhang, «Decentralized Multi-UAV Cooperative Search Based on ROS1 and ROS2», en *International Conference on Autonomous Unmanned Systems*, Springer, 2021, págs. 2427-2435.

[20] M. Siwek, K. Besseghieur y L. Baranowski, «The effects of the swarm configuration and the obstacles placement on control signals transmission delays in decentralized ROS-embedded group of mobile robots», en *AIP Conference Proceedings*, AIP Publishing, vol. 2029, 2018.

[21] J. M. Esposito y V. Kumar, «An asynchronous integration and event detection algorithm for simulating multi-agent hybrid systems», *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol. 14, n.º 4, págs. 363-388, 2004.

[22] J. Wang, X. Luo y J. Yan, «Event-Triggered Consensus Control for Second-Order Multi-Agent Systems With/Without Input Time Delay», *IEEE Access*, vol. PP, págs. 1-1, oct. de 2019. DOI: 10.1109/ACCESS.2019.2946263.

[23] F. Muñoz Palacios, E. S. Espinoza Quesada, H. La, S. Salazar, S. Commuri y L. R. Garcia Carrillo, «Adaptive consensus algorithms for real-time operation of multi-agent systems affected by switching network events», *International Journal of Robust and Nonlinear Control*, vol. 27, oct. de 2016. DOI: 10.1002/rnc.3687.

[24] M. Wooldridge, *An Introduction to MultiAgent Systems*, 2.ª ed. Wiley, 2009.

[25] Y. Shoham y K. Leyton-Brown, *Multiagent Systems: Algorithmic, Game-Theoretic, and Logical Foundations*. Cambridge University Press, 2009.

[26] S. Russell y P. Norvig, *Artificial Intelligence: A Modern Approach*, 4.ª ed. Pearson, 2021.

[27] G. Weiss, *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*. MIT Press, 1999.

[28] W. Ren y R. W. Beard, *Distributed Consensus in Multi-vehicle Cooperative Control*. Springer, 2008.

[29] R. Olfati-Saber, J. A. Fax y R. M. Murray, «Consensus and cooperation in networked multi-agent systems», *Proceedings of the IEEE*, vol. 95, n.º 1, págs. 215-233, 2007.

[30] M. Brambilla, E. Ferrante, M. Birattari y M. Dorigo, «Swarm robotics: a review from the swarm engineering perspective», *Swarm Intelligence*, vol. 7, n.º 1, págs. 1-41, 2013.

[31] R. S. Sutton y A. G. Barto, *Reinforcement Learning: An Introduction*, 2.ª ed. MIT Press, 2018.

[32] L. P. Kaelbling, M. L. Littman y A. W. Moore, «Reinforcement Learning: A Survey», *Journal of Artificial Intelligence Research*, vol. 4, págs. 237-285, 1996.

[33] C. J. Watkins y P. Dayan, «Q-Learning», *Machine Learning*, vol. 8, n.º 3-4, págs. 279-292, 1992.

[34] J. Kober, J. A. Bagnell y J. Peters, «Reinforcement Learning in Robotics: A Survey», *The International Journal of Robotics Research*, vol. 32, n.º 11, págs. 1238-1274, 2013.

[35] T. P. Lillicrap et al., «Continuous control with deep reinforcement learning», *arXiv preprint arXiv:1509.02971*, 2016.

[36] V. Mnih et al., «Human-level control through deep reinforcement learning», *Nature*, vol. 518, págs. 529-533, 2015.

[37] L.-J. Lin, «Self-improving reactive agents based on reinforcement learning, planning and teaching», *Machine Learning*, vol. 8, n.º 3-4, págs. 293-321, 1992.

[38] J. Cortés, S. Martínez y F. Bullo, «Robust rendezvous for mobile autonomous agents via proximity graphs in arbitrary dimensions», *IEEE Transactions on Automatic Control*, vol. 51, n.º 8, págs. 1289-1298, 2006.

[39] M. Quigley et al., «ROS: an open-source Robot Operating System», en *IEEE International Conference on Robotics and Automation (ICRA) Workshop on Open Source Software*, 2009.

[40] N. Koenig y A. Howard, «Design and use paradigms for Gazebo, an open-source multi-robot simulator», *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, págs. 2149-2154, 2004.

[41] S. Macenski, F. Martín, R. White y J. Ginés Clavero, «The ROS 2 Navigation System (Nav2): Design, architecture, and methods», *The International Journal of Robotics Research*, vol. 41, n.º 7, págs. 688-716, 2022.

[42] B. Gerkey, *ROS 2: Towards a fully distributed, real-time robotic middleware*, Open Robotics Blog, 2020. dirección: `https://www.openrobotics.org/blog`.

[43] Y. Chen, Y. Fan y M. Jin, «Research on Sensor Technology in Mobile Robot Navigation», *Applied and Computational Engineering*, vol. 93, n.º 1, págs. 50-55, 2024.

[44] StandardBots. «Types of sensors in robotics: Complete guide with examples». dirección: `https://standardbots.com/blog/every-type-of-sensors-in-robotics---explained`.

[45] F. Cañadas-Aránega et al., «Autonomous collaborative mobile robot for greenhouses», *ScienceDirect*, 2024, One of the most common sensors used for navigation is the LiDAR sensor.

[46] X. Qiu, K. Wan y F. Li, «Mobile Robot Navigation Using Deep Reinforcement Learning», *Processes*, vol. 10, n.º 12, pág. 2748, 2022.

[47] Y. Zhu, W. Z. W. Hasan, H. R. H. Ramli, N. M. H. Norsahperi, M. S. M. Kassim e Y. Yao, «Deep Reinforcement Learning of Mobile Robot Navigation in Dynamic Environment: A Review», *Sensors*, vol. 25, n.º 11, pág. 3394, 2025.

[48] L. Doko et al., «A Comprehensive Review of Mobile Robot Navigation Using Deep Reinforcement Learning Algorithms in Crowded Environments», *Journal of Intelligent & Robotic Systems*, vol. 110, 2024.

[49] ROBOTIS, *Robot Platform — TurtleBot3 Features*, Consulta: 10 de noviembre de 2025, 2025. dirección: `https://emanual.robotis.com/docs/en/platform/turtlebot3/features/`.

[50] ROBOTIS, *TurtleBot3 — Overview and ROS 2 Integration*, Consulta: 10 de noviembre de 2025, 2025. dirección: `https://emanual.robotis.com/docs/en/platform/turtlebot3/overview/`.

[51] A. Kumar y K. Konathalapalli, «Autonomous Navigation of ROS 2 based TurtleBot3 in Static and Dynamic Environments using Intelligent Approaches», *International Journal of Information Technology*, 2025, Uso de TurtleBot3 con LiDAR y aprendizaje por refuerzo profundo.

[52] A. W. Services. «Get to know your AWS DeepRacer vehicle». Consulta: 10 de noviembre de 2025. dirección: https://docs.aws.amazon.com/deepracer/latest/developerguide/inspect-your-vehicle.html.

[53] W. contributors. «Traxxas — RC Vehicle Manufacturer Overview». Consulta: 10 de noviembre de 2025. dirección: https://en.wikipedia.org/wiki/Traxxas.

[54] R. Federation. «Middle Size League — RoboCup Federation». Consulta: 10 de noviembre de 2025. dirección: https://msl.robocup.org/.

[55] L. ROBOTIS Co., *turtlebot3_applications*, https://github.com/ROBOTIS-GIT/turtlebot3_applications, Accessed: November 9, 2025, n.d.

[56] L. ROBOTIS Co. «TurtleBot3 Basic Examples – Follower». Accessed: November 9, 2025. dirección: https://emanual.robotis.com/docs/en/platform/turtlebot3/basic_examples.

[57] L. ROBOTIS Co., *turtlebot3_machine_learning*, https://github.com/ROBOTIS-GIT/turtlebot3_machine_learning/tree/humble, Último acceso: 9 de noviembre de 2025, n.d.

[58] L. ROBOTIS Co. «TurtleBot3 Machine Learning – Deep Q-Learning». Accessed: November 9, 2025. dirección: https://emanual.robotis.com/docs/en/platform/turtlebot3/machine_learning/.