UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

FACULTAD DE INGENIERÍA

MAESTRÍA EN CIENCIAS EN INTELIGENCIA ARTIFICIAL

Detección de Obstáculos y Planeación de Ruta para un Vehículo Autónomo

Tesis que como parte de los requisitos para obtener el grado de la Maestría en Ciencias en Inteligencia Artificial

Presenta:

Roberto de Jesús Alfaro López

Dirigido por:

Dr. Juan Manuel Ramos Arreguín

Querétaro, Qro. Diciembre 2023

REPOSITORIO INSTITUCIONAL DGBSDI-UAQ

Dirección General de Bibliotecas y Servicios Digitales
de Información

Detección de Obstáculos y Planeación de Ruta para
un Vehículo Autónomo

por

Roberto de Jesús Alfaro López

se distribuye bajo una Licencia Creative Commons
Atribución-NoComercial-SinDerivadas 4.0 Internacional.

Clave RI: IGMAC-309294

UNIVERSIDAD AUTÓNOMA DE QUERÉTARO

FACULTAD DE INGENIERÍA

MAESTRÍA EN CIENCIAS EN INTELIGENCIA ARTIFICIAL

Detección de Obstáculos y Planeación de Ruta para un Vehículo Autónomo

Tesis que como parte de los requisitos para obtener el grado de la Maestría en Ciencias en Inteligencia Artificial

Presenta:

Roberto de Jesús Alfaro López

Dirigido por:

Dr. Juan Manuel Ramos Arreguín

Dr. Juan Manuel Ramos Arreguín
Presidente

Dr. Marco Antonio Aceves Fernández
Secretario

Dr. Efrén Gorrostieta Hurtado
Vocal

Dr. Saúl Tovar Arriaga
Suplente

Dr. Jesús Carlos Pedraza Ortega
Suplente

Agradecimientos

Alcanzar nuestros sueños requiere de un esfuerzo sostenido y una constancia inquebrantable; valores fundamentales sin los cuales sería imposible alcanzar nuestras metas. Sin embargo, estos no son más que eslabones dentro de una extensa cadena de dedicación y perseverancia, forjada tanto por nosotros como por quienes nos han precedido, quienes tienen también un gran mérito en nuestros logros y son por tanto también, dignos de mención dentro de ellos.

Por esto y más, agradezco de todo corazón a mis padres por el regalo de la vida y de la educación, las cuales me fueron otorgadas con grandes sacrificios. Gracias igual a toda mi familia en general, por estar siempre allí, regalándome momentos de alegría y diversión sin los cuales la vida no sería digna de llamarse así. A los Legionarios de Cristo, por brindarme toda la formación humana durante mi adolescencia. A mis profesores y asesores de la maestría, quienes me adentraron en el mundo de la inteligencia artificial. Y finalmente y sobre todo, a Dios, por permitirme existir en este tiempo y espacio, y poder reír, llorar y, sobre todo, aprender.

Resumen

El presente documento detalla la creación de un sistema de evasión de obstáculos para un robot móvil tipo Ackermann, utilizando el aprendizaje por refuerzo con el algoritmo DQN. Detalla los principios teóricos que fundamentan este enfoque y describe la metodología aplicada para configurar el entorno de entrenamiento y realizar simulaciones. El sistema se implementó utilizando un sensor LiDAR 2D y una cámara RGB-D, que proporcionaron al robot una percepción ambiental efectiva. Se hace énfasis en la configuración de los hiperparámetros de entrenamiento, la estructura de la red neuronal, la función de recompensas, y los espacios de observación y acción, elementos claves para el rendimiento exitoso del agente. Finalmente, se presentan ejemplos ilustrativos del desempeño del robot, tanto en entornos simulados como en condiciones reales, demostrando la efectividad y aplicabilidad del sistema desarrollado en situaciones prácticas.

Palabras clave: Aprendizaje por refuerzo, red neuronal, DQN, robot móvil, LiDAR.

Abstract

This document details the creation of an obstacle avoidance system for an Ackermann-type mobile robot, using reinforcement learning with the DQN algorithm. It outlines the theoretical principles underlying this approach and describes the methodology applied to set up the training environment and conduct simulations. The system was implemented using a 2D LiDAR sensor and an RGB-D camera, which provided the robot with effective environmental perception. Emphasis is placed on the configuration of the training hyperparameters, the structure of the neural network, the reward function, and the observation and action spaces, key elements for the successful performance of the agent. Finally, illustrative examples of the robot's performance are presented, both in simulated environments and in real conditions, demonstrating the effectiveness and applicability of the developed system in practical situations.

Keywords: Reinforcement Learning, neural network, DQN, mobile robot, LIDAR

Índice

Índice de FIGURAS

Índice de tablas

Tabla de abreviaturas

Sigla/Abreviatura	Significado
PF	*Potential Fields Algorithm*
A*	*A* Search Algorithm*
LLM	*Large Language Model*
IMU	*Inertial Measurement Unit*
DQN	*Deep Q Network*
ASV	*Autonomous Surface Vehicle*
UAV	*Unmanned Aerial Vehicle*
ReLU	*Rectified Linear Unit*
CNN	*Convolutional Neural Network*
FCN	*Fully Connected Network*
ROS	*Robot Operating System*

Capítulo 1. Introducción

El área de la movilidad autónoma ha adquirido cada vez más relevancia en los últimos años, a tal grado de que importantes empresas como Google, General Motors, Toyota, Nvidia, Tesla, entre otras, han invertido grandes cantidades de recursos para la investigación y desarrollo de estas tecnologías, tanto para su aplicación en automóviles como en robótica. Sin embargo, a pesar de que, en efecto, se han logrado grandes avances en esta dirección, también es cierto que no se ha alcanzado la madurez suficiente para la implementación a gran escala de estos sistemas. Y es que, realmente, la facilidad con la que el ser humano desempeña esta tarea puede otorgar una falsa percepción de simplicidad para este problema; siendo, al contrario, bastante complejo. Esta complejidad radica especialmente, en la gran diversidad de sistemas y algoritmos distintos que deben colaborar en conjunto, para resolver facetas específicas contenidas dentro del gran problema principal. De esta manera se tendrían, por ejemplo, sistemas dedicados exclusivamente a la percepción y reconocimiento de objetos, otros centrados a la planeación de ruta, e incluso, otros más, diseñados para tomar decisiones de navegación en tiempo real, como la predicción de trayectorias para obstáculos dinámicos o la segmentación del camino/carretera; todo esto dentro de un sistema más grande. Se puede observar entonces, que la resolución de cada uno de estos subproblemas representa un gran desafío en sí mismo, y a ello se suman otros factores adicionales que inciden igualmente de manera significativa en el rendimiento, siendo uno de los más determinantes, el entorno operativo del vehículo o robot.

Dicho lo anterior, queda claro que los sistemas de movilidad autónoma lidian especialmente, con la dificultad de crear un algoritmo distinto para cada tarea específica; con los desafíos de comunicación entre cada uno de estos subsistemas; con la carga computacional [1] que requiere mantener a un sistema aún más grande compuesto de estos últimos; y finalmente, en ocasiones, con la poca flexibilidad y capacidad de generalización ante distintos entornos de operación que algoritmos

clásicos de evasión y planeación de ruta como PF y A*, pueden ofrecer [2]. Y es precisamente en este punto, en donde los métodos de *machine learning* se perfilan como grandes aliados para abordar estos desafíos, particularmente el área dentro de este, conocida como *reinforcement learning* o aprendizaje por refuerzo.

Si bien es importante notar, que ciertos métodos de *machine learning* poseen una trayectoria considerable siendo utilizados en sistemas de navegación autónoma; al menos en lo que respecta al aprendizaje por refuerzo, su aplicación ha sido ligeramente más reciente, especialmente en el *deep reinforcement learning*, en el que se utilizan redes neuronales para predecir las mejores acciones a realizar para el agente/robot. Pero ¿cuál es la razón por la que estos métodos se ven como una manera más poderosa de abordar la movilidad autónoma? Pues fundamentalmente, por sus habilidades de integración, planificación y adaptabilidad.

En términos muy generales, esta forma de aprendizaje automático permite que un agente (la entidad que realiza la tarea) aprenda a tomar las mejores decisiones para lograr un objetivo a través de prueba y error [3]. Esto provoca que el modelo actúe en el entorno, no por programación explícita de instrucciones o por haber aprendido de datos etiquetados, sino porque logra calcular las consecuencias que cada una de sus acciones puede tener a largo plazo, de esta manera, con la suficiente cantidad de interacciones, se puede conseguir un modelo capaz de adaptarse incluso, a entornos con los que no ha interactuado antes, o sea, un modelo con mayor capacidad de generalización o adaptabilidad.

Por otra parte, al combinar ciertos métodos de *reinforcement learning* con otros algoritmos del machine learning, particularmente, las redes neuronales, se pueden obtener nuevas ventajas. Y es que se debe recordar que las redes neuronales, con el debido procesamiento, permiten manejar una gran cantidad y tipos de datos, por lo que surge la posibilidad de integrar las múltiples fuentes de información del robot/vehículo en un solo modelo, prescindiendo así, de la necesidad de codificar y entrenar un solo algoritmo para cada subtarea, e incluso, abriendo la posibilidad de que los requerimientos computacionales sean menores o, al menos, se puedan

repartir de mejor manera. En otras palabras, con el uso de redes neuronales se puede integrar toda la información de los sensores en un solo modelo y permitir que este mismo aprenda a interpretarla a través de prueba y error, logrando una mejor integración e interpretación de la información sin necesidad de mantener varios algoritmos trabajando por su cuenta.

Es importante resaltar aquí, que muchos de los grandes avances que se han presenciado recientemente en el área de la inteligencia artificial, han sido gracias precisamente a la aplicación del aprendizaje por refuerzo. Ejemplos de esto son LLM's como Bard y GPT-4, o modelos como AlphaGo y AlphaStar; los primeros logrando imitar la capacidad humana de la comunicación al entablar con los humanos conversaciones complejas, y los segundos capaces de jugar al Go y a Starcraft respectivamente. Estos ejemplos permiten formarse una idea más clara del potencial del que es capaz el *reinforcement learning*.

Hasta ahora, se han expuesto algunos de los desafíos más grandes con los que cuenta la creación de sistemas de navegación autónoma; e igualmente, se ha descrito el por qué algunos de los métodos de inteligencia artificial, particularmente el aprendizaje por refuerzo, se perfilan como una herramienta poderosa para abordar el problema de manera simple y efectiva. Sin embargo, es crucial destacar que el impacto de la investigación y el desarrollo de estos sistemas no se confina exclusivamente a los dominios académico y tecnológico. En realidad, sus efectos se extienden a dimensiones sociales, económicas y ambientales, ofreciendo potencialmente mejoras significativas en la calidad de vida y abriendo nuevas áreas de oportunidad. Por ello, su estudio y avance son indispensables.

De esta manera, en el presente documento se presentan varios ejes cruciales por los cuales resulta relevante e incluso conveniente y provechoso el avance de las tecnologías de movilidad autónoma.

El primer eje es la seguridad vial. Según datos de la Organización Mundial de la Salud (OMS), aproximadamente 1.3 millones de personas mueren cada año como

resultado de accidentes de tráfico. Muchos de estos accidentes son resultado de errores humanos, como distracciones, consumo de alcohol o falta de experiencia. Los vehículos impulsados por tecnologías de movilidad autónoma podrían reducir significativamente el número de accidentes al eliminar estos factores de riesgo.

En segundo lugar, es bien sabido que la eficiencia en el transporte es un problema creciente en muchas ciudades del mundo. El tráfico, la congestión y la ineficiencia en las avenidas no son solamente una fuente de estrés para los individuos, sino que también contribuyen al deterioro ambiental por medio de la emisión de gases contaminantes. La implementación de estos sistemas de navegación puede optimizar el uso de las vías mediante una conducción más eficiente, colaborativa y, potencialmente, más respetuosa con el medio ambiente.

Otro aspecto crítico es la accesibilidad. Actualmente un gran número de personas poseen discapacidades o limitaciones físicas, para ellos el transporte convencional suele presentar desafíos considerables, incluso la movilidad en un mismo espacio. Los sistemas autónomos podrían brindarles un alto grado de independencia, tanto para transportarse dentro de una ciudad, como en espacios públicos o en su mismo hogar, recurriendo para esto último quizá, a robots o sistemas integrados en sillas de ruedas, como se puede ver en [4].

Por último, como se mencionó arriba, la implementación de estas tecnologías no se limita a vehículos de transporte de pasajeros. Existen un sinfín de aplicaciones especializadas como la exploración espacial, la atención de desastres naturales y la logística industrial que podrían beneficiarse de avances en esta área. De esta manera, robots autónomos podrían llevar a cabo tareas de entrega o manejo de productos, misiones de rescate en entornos peligrosos, o incluso, la exploración de terrenos inhóspitos, haciendo avanzar así los límites del conocimiento humano.

Dicho todo lo anterior, se puede observar, que el desarrollo de sistemas de movilidad autónoma eficientes y confiables, tiene implicaciones que van más allá de una simple demostración de avance tecnológico; de hecho, más bien, se trata de

una revolución que tiene el potencial de salvar vidas, mejorar la calidad de vida, optimizar el uso de recursos y abrir nuevos horizontes en la ciencia y exploración. Por todo ello, se decidió llevar a cabo la presente investigación.

El proyecto que se presenta en este documento tiene como objetivo profundizar en la aplicación de métodos de aprendizaje por refuerzo en sistemas de movilidad autónoma. Esto se plantea como una alternativa sofisticada a las estrategias tradicionales, que generalmente dependen de una variedad de algoritmos especializados para la detección de obstáculos y la planificación de rutas.

Delimitando con mayor detalle, en este trabajo se entrena a un robot móvil el cual tiene la misión de llegar a un objetivo mientras detecta los obstáculos del entorno y realiza la evasión de estos. El entorno objetivo son especialmente áreas al aire libre o interiores con la suficiente cantidad de espacio para las maniobras del vehículo debido a su estructura. El robot móvil es el modelo R2 de la marca Yahboom, el cual posee una configuración tipo Ackermann.

Además de esto, para dotar al agente autónomo de capacidades de percepción, se emplean diferentes configuraciones de sensores según el escenario de experimentación:

- En el entorno de simulación, el robot está equipado con un sensor IMU (Unidad de Medición Inercial) y un sensor LiDAR 2D para capturar información detallada del ambiente circundante.
- En las pruebas de campo, en entorno físico real, se añade una cámara RGB-D junto con un set de encoder para cada motor, a la configuración anterior de sensores para proporcionar una percepción más rica y detallada del entorno, complementando los datos obtenidos del IMU y del LiDAR 2D.

Los sensores antes mencionados, en el caso de la cámara RGB-D y del LiDAR 2D, se escogieron por el gran complemento que resulta cuando trabajan en

conjunto, el primero otorgando datos 3D detallados, y el segundo distancias precisas hacia cada obstáculo en 360°. Por su parte, el IMU y los encoder se usan para otorgarle al agente sus coordenadas y orientación en el espacio.

Para el entrenamiento del agente se recurrió al uso de DQN (*Deep Q Network*). Esto por ser un método que permite manejar una gran cantidad de observaciones del entorno y calcular la mejor acción a realizar, de manera eficiente. Además, la capacidad de generalización de este método ante estados no observados durante el entrenamiento es muy buena.

Dicho todo lo anterior, se dejan fuera de la investigación aspectos como, entornos con alta densidad, debido principalmente a la estructura del robot; condiciones extremas, como entornos lluviosos o con niebla; y la interacción con obstáculos dinámicos, como son por ejemplo la navegación en entornos con peatones u otros vehículos.

Por último, para finalizar la sección, se explica a continuación de manera general, el contenido de cada capítulo del documento:

- Antecedentes: Se explican los fundamentos teóricos necesarios para comprender el proyecto, así como también se da un repaso por algunos trabajos destacados en esta área.
- Hipótesis y Objetivos: Se detallan las metas del proyecto y los supuestos de la investigación.
- Materiales y métodos: Se exponen todos los pasos (metodología) seguidos para culminar el trabajo, así como todas las herramientas usadas.
- Resultados y discusión: Se analiza el comportamiento resultante del agente y se realizan comparaciones con otros trabajos.
- Conclusiones: Comentarios del autor y trabajo futuro.

Capítulo 2. Antecedentes

En el capítulo previo, se presentó la complejidad y desafíos que conlleva la implementación de sistemas de movilidad autónoma. Se señaló cómo esta complejidad se amplifica si se adopta un enfoque "clásico" para abordar el problema. En este método tradicional, se solucionan las diversas subtareas (como la detección de obstáculos y la planificación de rutas) de manera independiente para luego integrar todas las soluciones en un sistema unificado. Este enfoque se diferencia significativamente del aprendizaje por refuerzo. En este último, los parámetros del modelo se optimizan de manera autónoma hasta que el sistema es capaz de integrar toda la información que recibe en un modelo único y seleccionar las acciones más adecuadas por sí mismo. Esto elimina la necesidad de un ajuste meticuloso entre diferentes algoritmos, una tarea que normalmente recaería sobre el programador.

Dada la diversidad de estrategias que se pueden emplear para implementar ambos enfoques, ha emergido una gran cantidad de investigaciones y proyectos destinados a perfeccionar distintos aspectos de los sistemas de movilidad autónoma. Estos proyectos varían desde técnicas más tradicionales, hasta la aplicación de métodos más avanzados, como el aprendizaje por refuerzo. Por esta razón, resulta indispensable realizar un recorrido por algunos de los trabajos más significativos que se han desarrollado en este campo.

A continuación, se presentan primero, proyectos que fueron realizados siguiendo una perspectiva más “clásica”, los cuales abordaron el problema usando un algoritmo para cada subtarea. Y, por último, al final de la sección, se mencionarán otros proyectos que resolvieron el problema a través del aprendizaje por refuerzo.

Empezando por los primeros, destaca el trabajo desarrollado por Woo y compañía [5]. En él se trabajó en un ASV capaz de esquivar obstáculos y realizar búsquedas. El robot se ayudaba de un LiDAR 2D y un sensor de visión monocular para la detección de objetos que flotaran en el agua. Destacan, además, los

métodos usados para la planeación de trayectoria, usando el algoritmo A* y un mapa generado con ayuda del filtro Kalman.

Otro proyecto importante fue presentado por Wang y su equipo [6]. Este consistió en el desarrollo de un sistema de percepción y evasión de obstáculos para un UAV, pensado específicamente para su uso en áreas de cultivo. Este estaba conformado solamente por una cámara de profundidad la cual, junto con técnicas de aprendizaje profundo (específicamente CNN's), fusión de información RGB-D y el uso de TCS (Sistema de Control de Tareas, por sus siglas en inglés) podía detectar obstáculos con una precisión del 75.4% y determinar la estrategia óptima para evitarlos. Resalta además que el UAV también era capaz de percibir atributos de los objetos a su alrededor, como categoría, perfil y posición espacial 3D.

Otro sistema pensado para vehículos aéreos lo implementó Lin y colaboradores [7]. El sistema se montó en un cuadricóptero pequeño, equipado con una cámara estéreo. El enfoque se sustentaba, en un algoritmo para la detección de obstáculos basado en pares de imágenes de profundidad el cual proporcionaba la posición estimada del objeto, su velocidad y su tamaño. Además, formularon un CC-MPC (Chance-Constrained Model Predictive Controller) que mantenía la probabilidad de colisión con un obstáculo en movimiento por debajo de un umbral específico.

Resulta interesante también, el trabajo desarrollado por Back [8]. Aquí se integraron varios modelos en un UAV que le permitían reconocer y seguir senderos de bicicletas a la vez que evadía obstáculos, esto ayudándose de la visión otorgada al vehículo por una cámara. El UAV era capaz de mantenerse en el centro del sendero y, en caso de que por algún motivo este se desviará, algunos comandos de control se almacenaban por cierto tiempo, con el fin de que el vehículo pudiera retornar a su posición. El principal algoritmo utilizado fueron redes neuronales convolucionales.

Siguiendo por esta misma línea "clásica" de abordar la movilidad autónoma, pero enfocándose ante todo en mejorar la percepción de los vehículos, destacan

dos trabajos importantes. El primero desarrollado por Eppenberger y colaboradores [9], propone la implementación de un sistema de detección de obstáculos que permite el reconocimiento de su movimiento, clasificándolos de esta manera en estáticos y dinámicos, prediciendo su trayectoria y permitiendo una mejor evasión de ellos. Este sistema se implementó usando algoritmos de clustering y redes neuronales en conjunto, de esta manera consiguió una precisión en la detección de objetos estáticos de 96.9% y de 85.3% para objetos dinámicos.

El segundo trabajo importante enfocado en la percepción fue realizado por Zhou y su equipo [10]. En este proponen un esquema de evasión de obstáculos distinguida especialmente por el trabajo conjunto de dos modelos distintos una red neuronal convolucional (CNN) y un método propio de procesamiento de imágenes de LiDAR. Lograron un reconocimiento del 100% para los obstáculos gracias a la combinación de algoritmos.

En otro punto de este mismo enfoque tradicional se ubican algunos trabajos cuyo principal objetivo fue mejorar la planeación de ruta o la tarea de evasión, sin preocuparse demasiado por la percepción. Se destacan tres proyectos, el primero realizado por Soumic Sarkar, el segundo por Song y uno último por Cerbaro y su equipo.

En el primero, Sarkar y colegas [11] proponen algunas modificaciones al algoritmo PFF. Este algoritmo calcula el camino a seguir según el tamaño y la posición de los obstáculos en el entorno, sin embargo, en ocasiones se llega a mínimos locales. De esta manera en el proyecto se desarrolla una técnica que ayuda a evitar estas situaciones.

Los siguientes dos sistemas de evasión hechos por Song [12] y Cerbaro [13] respectivamente, se abordaron usando lógica difusa. Con respecto al primero se instaló en un USV (bote no tripulado) utilizando un sensor LiDAR para la percepción. En el segundo igualmente se usó un LiDAR, pero en un vehículo terrestre. El

principal diferenciador entre ambos fueron las funciones de membresía y las variables lingüísticas utilizadas.

Como se observa hasta este punto, los trabajos antes mencionados suelen recurrir a una gran diversidad de algoritmos y modelos de machine learning que trabajan en conjunto para abordar el problema de la navegación autónoma. A continuación, se presentan algunos otros trabajos que, en su lugar, utilizan el aprendizaje por refuerzo para esta tarea, permitiendo una integración más natural y flexible entre los sistemas de percepción y las decisiones del robot.

Como primer trabajo importante en este rubro se encuentra el realizado por Duguleana y Mogan [1]. El sistema implementado aquí, fue una combinación de Q learning con redes neuronales para llevar a cabo tareas de evasión y planeación de ruta. El entrenamiento se llevó a cabo en una simulación y la evaluación del modelo resultante en un entorno real.

Sobresale también el proyecto de Xie y compañía [14], en el que con sólo una cámara RGB monocular y una red neuronal, se logra la navegación y alcance de objetivos para un robot. Todo sin necesidad de información 3D adicional ni algoritmos de planeación de ruta. La técnica de refuerzo que utilizaron para entrenar la red fue D3QN (*Deep Double Q Network*).

Igual de interesante resulta la labor de Song [15]. En este documento se propone un método diferente de entrenamiento por refuerzo llamado MDRLAT. El objetivo principal era entrenar a un agente capaz de navegar en interiores. La red neuronal propuesta posee dos ramas de entrada, una para procesar imágenes y otra para extraer información de un sensor LiDAR. El algoritmo de refuerzo utilizado fue *Dueling Deep Q Network*.

Por último, para terminar esta sección, se desea destacar en gran medida dos trabajos cuyas propuestas de refuerzo pueden resultar de gran utilidad para el lector. La primera, desarrollada por Surmann y colaboradores [16], implementa aprendizaje por refuerzo en un robot equipado con un LiDAR 2D y una cámara de profundidad.

El algoritmo implementado para el entrenamiento fue GA3C (*Asynchronous Advantage Actor Critic*) lo que junto con un simulador 2D desarrollado por ellos mismos, permitió una considerable aceleración del entrenamiento. Además, resalta el hecho de que se combinan las observaciones del LiDAR y la cámara en un solo vector, simplificando así el problema de la integración de diferentes datos con los que se alimenta a un solo modelo (en este caso una red neuronal).

El segundo y último trabajo fue realizado por Stachowicz y su equipo [17], que implementaron un método de aprendizaje por refuerzo al que llamaron FastRLAP. Esta técnica consiste en entrenar a un agente de manera offline (o sea con "recuerdos" grabados de expertos) y posteriormente llevar a cabo un último entrenamiento en el entorno real. Los resultados son sumamente interesantes, pues gracias al preentrenamiento offline de la red neuronal, el vehículo (equipado con una cámara) logra aprender a navegar por el entorno en un periodo de 20 minutos.

A continuación, a manera de resumen, se muestra en la tabla 1 una compilación de los trabajos expuestos hasta ahora.

Año	Autor(es)	Descripción
2017	Woo, J.; Lee, J.; Kim, N.	ASV capaz de esquivar obstáculos y realizar búsquedas. Se usa el algoritmo A* para planeación de trayectoria y un mapa de cuadrícula generado por el filtro Kalman.
2020	Wang, D.; Li, W.; Liu, X.; Li, N.; Zhang, C.	UAV con cámara RGB-D, que usando redes neuronales convolucionales integraba la detección y evasión de obstáculos.
2020	Lin, J.; Zhu, H.; Alonso-Mora, J.	UAV con cámara estéreo para detección y evasión de obstáculos.

2020	Back, S.; Cho, G.; Oh, J.; Tran, X. T.; Oh, H.	Sistema para reconocer y seguir senderos de bicicleta y evadir obstáculos, basado en redes neuronales convolucionales.
2020	Eppenberger, T.; Cesari, G.; Dymczyk, M.; Siegwart, R.; Dubé, R.	Sistema para detección de objetos estáticos y dinámicos usando cámara estéreo y redes neuronales y clustering para identificar objetos.
2018	Zhou, C.; Li, F.; Cao, W.; Wang, C.; Wu, Y.	Sistema de evasión de obstáculos usando LiDAR 2D y webcam junto con una CNN de 10 capas.
2010	S. Sarkar, S. N. Shome, and S. Nandy	Se realiza una modificación al algoritmo PFF de evasión de obstáculos para evitar el error de los mínimos cuadrados.
2018	H. Song, K. Lee, and D. H. Kim	Implementación de evasión de obstáculos para un USV utilizando lógica difusa.
2020	J. Cerbaro, D. Martinelli, A. S. de Oliveira, and J. A. Fabro	Implementación de evasión de obstáculos para un robot terrestre de servicio utilizando lógica difusa.
2016	M. Duguleana and G. Mogan	Uso de Q learning en conjunto con redes neuronales para un vehículo.
2017	L. Xie, S. Wang, A. Markham, and N. Trigoni	Uso de D3QN para movilidad autónoma en un robot equipado con una cámara RGB.

2021	H. Song, A. Li, T. Wang	Implementación de MDRLAT como técnica de aprendizaje por refuerzo para un robot equipado con cámara y LiDAR.
2020	H. Surmann, C. Jestel, R. Marchel, F. Musberg, H. Elhadj, and M. Ardani	Creación de un entorno de simulación 2D para aprendizaje por refuerzo, así como el entrenamiento de un robot autónomo usando el algoritmo GA3C. Se utilizan un sensor LiDAR y una cámara de profundidad.
2023	K. Stachowicz, D. Shah, A. Bhorkar, I. Kostrikov, S. Levine	Implementación de la técnica FastRLAP para aprendizaje por refuerzo. Se realiza un preentrenamiento con experiencias recolectadas (*offline learning*) y por último un entrenamiento en el entorno de operación real, logrando un tiempo de convergencia en el aprendizaje online de 20 minutos.

Tabla 1. Trabajos relacionados.

Capítulo 3. Fundamento Teórico

En el presente apartado se dará un breve repaso por toda la teoría necesaria, para poder comprender cómo funcionan algunos de los sensores usados en los sistemas de detección de objetos, así como también, en qué consisten algunos de los algoritmos de inteligencia artificial más usados para este fin. Con ello se pretende que el lector pueda comprender de manera muy general el proyecto y las decisiones que se tomaron en su desarrollo.

A. LiDAR

Una de las tecnologías que más se han 'puesto de moda' en las aplicaciones relativas a la navegación autónoma (ya sean robots o vehículos), es precisamente el LiDAR. Este dispositivo nos permite, en términos generales, calcular la distancia que hay entre él y cualquier otro objeto que haya a su alrededor. Su nombre es un acrónimo del inglés *Light Detection and Ranging,* y está compuesto de, entre otras cosas, un láser y un sensor de luz. La figura 1 muestra el principio básico de operación de un LIDAR.

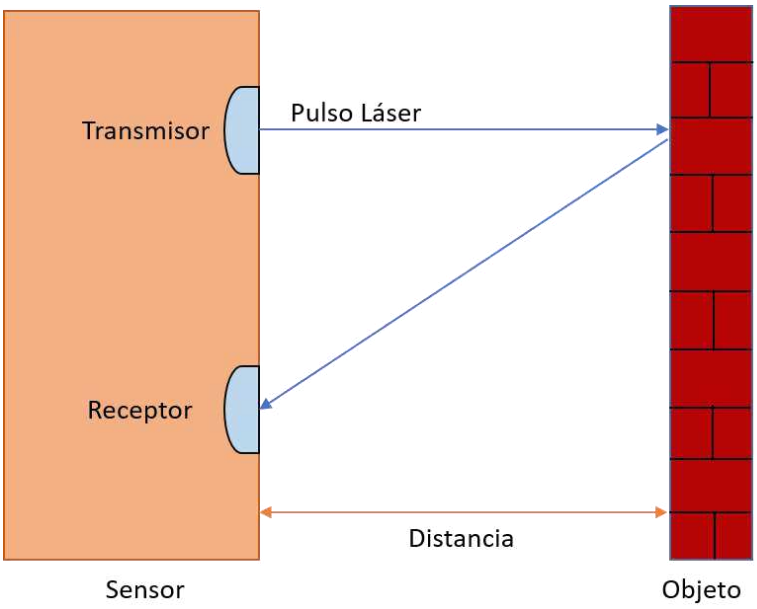

Figura 1. Esquema de funcionamiento de un LiDAR [18].

Su funcionamiento es relativamente sencillo: el láser emite un haz de luz que choca con los objetos a su alrededor, este haz rebota, es percibido por el sensor y se procede a calcular la distancia que hay entre el objetivo y él, dependiendo del tiempo que hubo entre la emisión del rayo y su detección; esta operación se realiza de acuerdo con (1) [18].

$$d = \frac{V * t}{2} \tag{1}$$

Donde:

- *d* es la distancia
- *V* la velocidad de la luz
- y *t* el tiempo total de emisión y retorno de la luz

Todas estas acciones se llevan a cabo muy rápidamente, y gracias a ello se pueden hacer mapeos del entorno donde el robot esté operando, permitiendo que pueda moverse por los espacios evitando los obstáculos que se le presenten [18].

Otro aspecto importante sobre estos dispositivos es que los hay de dos tipos, 2D y 3D, siendo la principal diferencia entre ellos que los LiDAR 2D mapean el entorno en X e Y, mientras que los LiDAR 3D lo hacen en el plano tridimensional. Esto se aprecia en la figura 2.

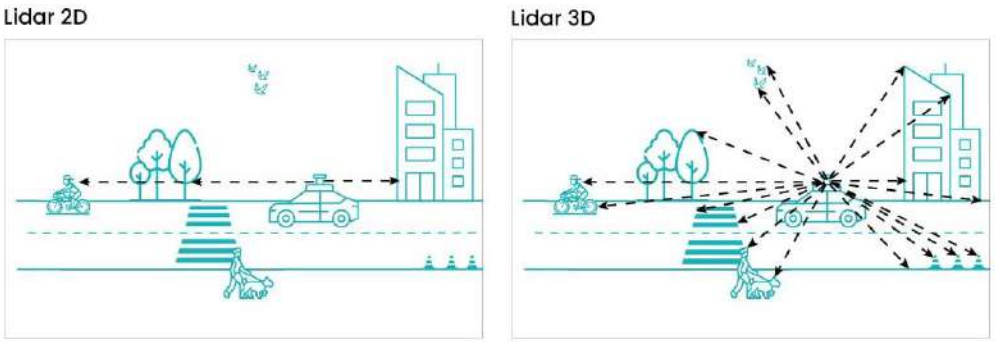

Figura 2. Diferencias entre LiDAR 2D y LiDAR 3D [19].

Se puede ver, por tanto, que estos sensores tienen un propósito parecido al de los radares o sonares, sin embargo, la forma de recolectar los datos de lo que lo rodea es lo que lo diferencia de ellos. Igualmente cabe mencionar, que el LiDAR que se usará en el proyecto será 2D, debido a que es un equipo con el que ya se cuenta, y los LIDAR 3D tienen precios muy elevados.

B. Cámaras de profundidad

Para el desarrollo de vehículos/robots autónomos, es imprescindible dotar al sistema, de sensores que le permitan percibir su entorno con la mayor precisión posible. Los LiDAR son ejemplo de ello; sin embargo, si se tuviera que elegir un sensor por sobre los demás, muy probablemente sería una cámara. Estos dispositivos son comunes para la mayoría de las personas, pues se encuentran integrados en smartphones, tablets, laptops, automóviles, entre otros, y básicamente, permiten capturar imágenes del lugar donde se encuentren. Así pues, las cámaras podrían ser para los robots, lo que los ojos son para los seres humanos, al menos teóricamente, pues en realidad no es tan sencillo como podría pensarse, ya que, para las computadoras, lo que están recibiendo son sólo un conjunto de píxeles con diferentes tonalidades que difícilmente pueden darles una idea del espacio a su alrededor (aunque sí existen algunos algoritmos que con mayor o menor éxito permiten esto). Por eso, para este tipo de tareas suelen emplearse cámaras ‘especiales’ que además de otorgarle al robot, las imágenes como las vemos, también les transmiten una sensación de profundidad que les ayuda a calcular la distancia a la que están las cosas a su alrededor. Esto se puede lograr de muchas maneras, y dependiendo de la tecnología usada, se dice que la cámara posee un sistema activo o pasivo [20], [21].

- Sistemas Pasivos: Se dice que las cámaras 3D tienen sistemas pasivos, cuando no se ayudan de alguna iluminación artificial que les permita hacer mapeos de profundidad. Quizá el sistema más conocido en este rubro, sean las cámaras estereoscópicas. Estas cámaras suelen tener dos o más lentes, que, bajo el principio de la triangulación de los rayos de luz desde

diferentes puntos, transmiten la percepción de tridimensionalidad; están inspirados en la visión binocular humana y funcionan muy bien en entornos bien iluminados, tanto en interiores como en exteriores, por esto son ampliamente utilizados en sistemas de navegación autónoma [20].

- Sistemas Activos: Por su parte, las cámaras con sistemas activos incorporan métodos para medir las distancias entre la cámara y los cuerpos circundantes, a través del uso de diversas tecnologías. Las técnicas más destacables son el ToF (*Time of Flight*) y la luz estructurada. La primera consiste en la medición del 'tiempo de vuelo' de un haz de luz láser, desde que se emite hasta que es recuperada por el sensor; un ejemplo de este tipo de cámara se puede encontrar en la segunda generación del Kinect. Por otro lado, los sistemas de luz estructurada proyectan sobre todo su rango visual, patrones (ya sean puntos o líneas) que les ayudarán a calcular la profundidad dependiendo del rango de distorsión que tengan; la primera generación del Kinect trabajaba de esta forma [20], [22].

Es relevante mencionar, que estos tipos de sistemas pueden unirse para trabajar en conjunto y aprovechar las ventajas que cada método aporta; por lo que, afortunadamente, no es necesario elegir una en específico, teniendo que renunciar a las propiedades de otra.

C. Nubes de puntos

Independientemente del tipo de cámara de profundidad con la que se cuente, es importante saber cómo aprovechar los datos que estas arrojan; para esto se utilizan las llamadas "nubes de puntos".

Las nubes de puntos son conjuntos de datos graficados como pequeñas esferas (puntos) en un espacio 3D. Cada dato individual representa una minúscula parte de la superficie del lugar o los objetos con los que la cámara esté interactuando (como dato rápido, estos puntos igual pueden ser obtenidos con ayuda de un LiDAR 3D);

de esta manera el vehículo/robot podrá percibir su entorno de una manera más detallada y ser capaz de detectar obstáculos, evadirlos y planear la mejor ruta para llegar a un punto específico. Por esto es importante el uso de las cámaras de profundidad, pues dotarán de un sentido tridimensional más detallado al sistema que se esté desarrollando [23].

Hay diferentes métodos para generar nubes de puntos a partir de cámaras de profundidad, esto dependerá de si la cámara posee un sistema activo o pasivo. Generalmente para una cámara estéreo, el proceso consiste en rectificar los cuadros de video y posteriormente calcular un mapa disparidad entre la imagen tomada con un lente y el otro, de esta manera se podrá hacer el cálculo para construir la escena 3D. Por otro lado, las cámaras con sistemas activos (también llamadas RGB-D) ya nos generan un mapa de profundidad, por lo que, con tan sólo algunos cálculos, es posible generar la nube de puntos del ambiente [24].

Una nube de puntos luce como en la figura 3.

Figura 3. Reconstrucción de un espacio usando nubes de puntos [25].

Como ya se mencionó, a partir de este tipo de visualizaciones el robot podrá tener un mejor sentido del lugar en el que opera; para lograr esto existen muchas técnicas de *Machine Learning* supervisado y no supervisado, que ayudarán a que

el autómata aprenda a detectar y reconocer obstáculos, aunque estos se examinarán en el transcurso del proyecto.

D. Redes Neuronales

Las redes neuronales artificiales son un modelo computacional inspirado en el cerebro humano, cuyo principal objetivo, es imitar su forma de aprendizaje para poder resolver problemas que resulten demasiado complejos si se usa la programación clásica. A pesar de que no ha sido sino hasta años recientes, que las redes neuronales han experimentado su mayor auge y evolución, en realidad hace ya varias décadas que esta tecnología fue pensada por primera vez en 1943 por Warren McCulloch y Walter Pitts, y perfeccionada posteriormente por una gran cantidad de científicos [26].

Para entender cómo funcionan, debemos saber que toda red neuronal está compuesta de pequeños nodos llamados “neuronas”, cada una de estas neuronas se conecta con otras intercambiando señales; así, cuando la red es estimulada (o sea que recibe ciertos valores de entrada) la red será capaz de generar una salida. Las conexiones que hay entre cada una de estas neuronas, forman lo que se les llama ‘capas’, y puede haber redes de una sola o con cientos de ellas; además, si una red tiene tres o más capas, se dice que tiene una capa de entrada, una o varias capas ocultas y una capa de salida. Anteriormente, se mencionó que estas redes son capaces de aprender por sí mismas, esto se logra, básicamente, modificando la fuerza que hay entre los enlaces de cada neurona (a estos valores se les llama “pesos”) y estableciendo ciertas “reglas” para la salida de cada una de ellas (llamadas función de activación y valor umbral); de esta forma, los distintos pesos que existen entre una misma entrada X1 y diferentes neuronas, hará que el valor cambie para cada una de ellas, del mismo modo, la función de activación y el valor umbral, calcularán una salida adecuada que será el valor de entrada para las neuronas que se encuentren más adelante en la red. Al proceso que consiste en modificar los pesos de las conexiones entre las neuronas, se le llama ‘entrenamiento’, y hay varias maneras de hacerlo, aunque el método más usado es

el algoritmo ‘backpropagation’ o ‘propagación hacia atrás’, que permite entrenar fácilmente a redes de grandes dimensiones [27].

En la figura 4 se puede ver el esquema básico de una sola neurona:

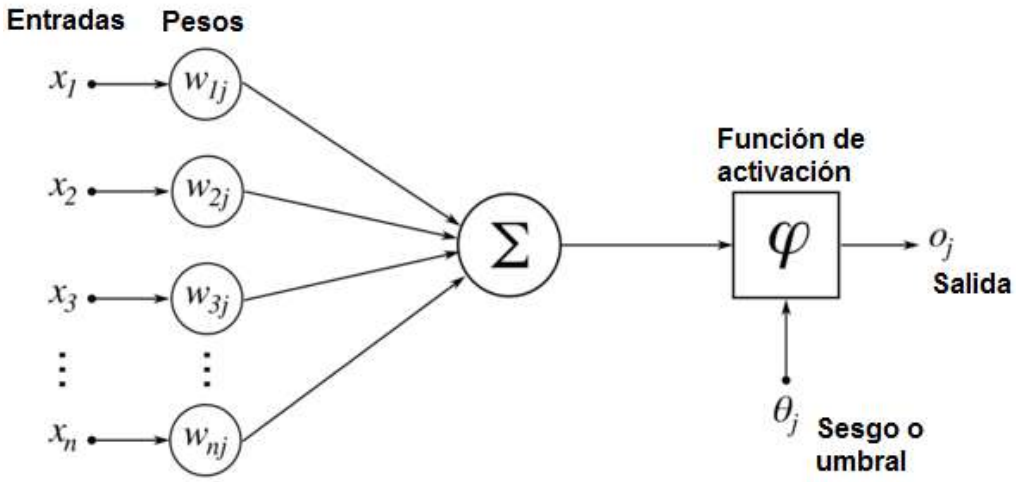

Figura 4. Esquema básico de una neurona artificial [27].

Así mismo, la arquitectura clásica de una red neuronal (la unión de muchas neuronas) se muestra en la figura 5.

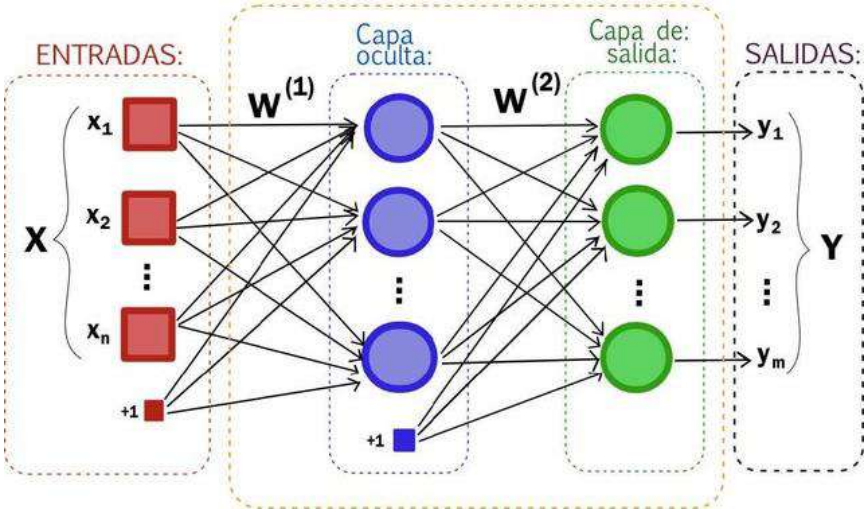

Figura 5. Arquitectura de una red neuronal [28].

Por último, es importante mencionar que hay muchos tipos de redes neuronales, cada una especializada en realizar ciertas tareas y con diversos tipos de conexiones entre los nodos. Las más usadas son las siguientes:

- Perceptrones Multicapa (*Redes Feedforward*): Estas redes transmiten la información de una neurona a otra sin ningún tipo de retroalimentación, ante cualquier entrada genera una salida con todas las señales dispersándose hacia adelante en la red (de ahí su nombre 'feedforward'). Son la base de todas las demás redes.

- Redes Neuronales Recurrentes: A diferencia del perceptrón multicapa, estas redes sí cuentan con ciclos de retroalimentación (*'feedback'*). Son capaces de procesar datos secuenciales y de generar una memoria, por lo cual son usadas para analizar texto, video o hacer predicciones de mercado [29].

- Redes Neuronales Convolucionales: Están optimizadas para trabajar con reconocimiento de objetos en imágenes, para lograr esto utilizan lo que se les llama convoluciones, lo que les ayuda, por ejemplo, a detectar bordes.

Las redes neuronales convolucionales, son las que tienen mayor relevancia en detección de obstáculos, es por esto que nos enfocaremos en desarrollar su teoría y repasar su funcionamiento en el presente documento.

E. Función de Activación

Ahora que se sabe qué es una red neuronal, se procederá a analizar a más detalle lo que son las redes neuronales convolucionales, aunque antes es importante repasar dos conceptos que resultan indispensables para entender el funcionamiento de estas, que son la función de activación y las operaciones de convolución.

Como se vio en el apartado anterior, las neuronas combinan sus entradas usando pesos, que pueden entenderse como la fuerza del enlace entre la salida de

una neurona y la entrada de otra. Una vez realizada la combinación lineal entre el valor de entrada y el peso, se procede a calcular una salida para la neurona, y es aquí donde entra el concepto de función de activación. La función de activación no es más que una función que recoge el nivel de estímulo obtenido de las entradas para modificarlas y generar una salida, de esta manera, se podrá transmitir información a las demás neuronas de la red [26].

Existen varios tipos de función de activación:

- Función Lineal
- Función Escalón
- Función Lineal con Saturación
- Función Sigmoidal
- Tangente Hiperbólica
- Función de activación Lineal Rectificada (ReLU)

F. Convoluciones

En el procesamiento de imágenes y datos secuenciales, existe una operación muy útil llamada ‘convolución’, esta resulta de especial interés al usar redes neuronales para clasificar fotografías, pues permiten resaltar algunas características de ellas, tales como los bordes o la nitidez de los objetos [26].

Como se sabe, una imagen no es más que un conjunto de píxeles de diferentes colores, que en conjunto forman las figuras de los objetos que se están viendo; este conjunto de píxeles puede representarse en forma de matriz. Partiendo de esto, se puede definir a la convolución como la transformación de una matriz (la imagen) ayudándose de otra a la que se le llama ‘Kernel’. Los Kernel pueden ser de diferentes tamaños (5x5, 3x3, etc.) y poseer distintos valores en cada uno de sus elementos, dependiendo del efecto que se quiera obtener en la imagen [26].

Por su parte la manera en que se realiza la operación es relativamente sencilla. Se pasa el Kernel por cada pixel de la matriz original (la imagen), al hacer esto lo primero que se hace es multiplicar cada valor del Kernel por los valores de la imagen que estén en la misma posición, se suma el total y se suple el valor del pixel central por ese total, esto se puede observar en la figura 6.

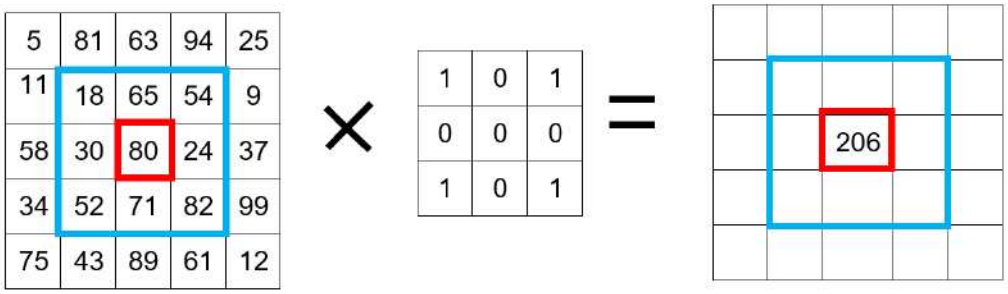

Figura 6. Operación de convolución sobre el pixel de una imagen.

Existen diferentes tipos de Kernel ya establecidos, que permitirán resaltar características específicas de una imagen, y de esta manera sea más sencillo para una red neuronal poder clasificar objetos dentro de una fotografía. Además, estas operaciones se usan en las redes neuronales convolucionales (de ahí su nombre) para optimizar el reconocimiento de la red.

G. Redes Neuronales Convolucionales (CNN)

No cabe duda, que las redes neuronales son uno de los algoritmos de la inteligencia artificial más interesantes y que quizá, incluso, llaman más la atención. Sin embargo, como todo, también tienen sus limitaciones, especialmente aquellas que confían en un solo tipo de arquitectura para hacer sus predicciones. Y es que, por ejemplo, si se desea que una red neuronal 'clásica' (de tipo feedforward por ejemplo) clasifique correctamente algún objeto en particular dentro de una imagen, muy probablemente esta red no será capaz de distinguirlo con mucha eficacia, no importa cuántos datos de entrenamiento haya recibido, ni cuántas iteraciones para el ajuste de los pesos se le den, con cualquier mínimo cambio que haya en las imágenes que se le dieron para entrenar, con respecto a las que se use para evaluar

la red, podría ser suficiente para que nuestro modelo se equivoque. Fue por este motivo, que surgieron, en 1998, gracias al trabajo de Yann Le Cun, las redes neuronales convolucionales (CNN), como un método que buscaría suprimir estas deficiencias [26].

El mecanismo de operación de las CNN's está inspirado en la forma en la que operan las neuronas de la corteza visual de los seres vivos; basta con saber, que en los ojos se encuentran dos tipos de neuronas, neuronas simples y neuronas complejas, las cuales se encargan de detectar líneas y bordes, agrupando progresivamente estos datos hasta formar la imagen que llega al cerebro. De una manera parecida opera una CNN, primero aplicando repetidas veces a la imagen una operación llamada convolución, lo cual permite extraer características diversas de los objetos en una imagen y posteriormente reduciendo la imagen, al tiempo que resalta dichas características, de tal manera que los objetos puedan distinguirse según sus formas, colores, y demás cualidades, por último, el resultado de estas operaciones será el input de una red neuronal feedforward, la cual clasificará la imagen según las clases que se hayan definido [26].

Las diversas capas de las CNN's y las operaciones que llevan a cabo se verán a continuación:

- Capa de Convolución: Es la que recibe los datos de entrada, en esta capa se realizan operaciones de 'convolución' (vistas anteriormente). De esta forma, se logran extraer las características más relevantes de una imagen y obtener diferentes 'efectos' que pueden hacer resaltar detalles específicos. Se suelen usar varios Kernel en una sola capa de convolución para que la red pueda reconocer las principales propiedades de los objetos. Por último, posterior a este proceso, suele usarse una función de activación 'ReLU, para normalizar las imágenes entre valores 0 y 1. [30]
- Capa de Agrupación (*Pooling*): En esta sección de la red, se reduce el tamaño de la imagen con el fin de reducir el coste computacional, además

se procuran mantener o incrementar aquellas características que nos habían resaltado las convoluciones. El método más usado para esto es el *'Max-Pooling'*, el cual reduce un área específica asignada por el usuario (por ejemplo de 2x2 píxeles), seleccionando el valor numérico más grande que haya allí (en nuestro ejemplo, si a una fotografía le aplicamos un *Max-Pooling* de 2x2 la imagen sería reducida a la mitad) [30].

- Perceptrón Multicapa: El último paso a realizar, una vez hechas las operaciones anteriores, es 'aplanar' la matriz de las imágenes resultantes; es decir, hacerla unidimensional, de tal forma que la podamos pasar como entrada para una red feedforward (perceptrón multicapa) la cual se encargará de hacer la clasificación final [30].

Cabe destacar, que puede haber más de una capa de convolución y agrupación, pero siempre siguiendo el mismo esquema (convolución primero y agrupación después). Muchas veces podrá ser más efectivo el modelo si hay más de estas capas, pero esto ya dependerá de la complejidad que el programador crea conveniente integrar a la red. Una arquitectura típica de una CNN se puede observar en la figura 7.

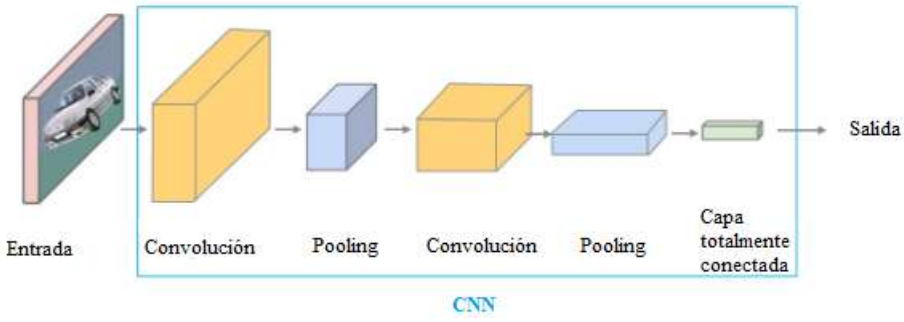

Figura 7. Arquitectura de una Red Neuronal Convolucional [31].

En esta sección se presentó el funcionamiento de las CNN, sin embargo, cabe aclarar que en el proyecto que se desarrolla en las siguientes páginas, se utiliza una versión un poco más simple de estas, realizando la operación de convolución en

una sola dimensión, esto debido al uso de un LiDAR 2D el cual otorga un solo vector de distancias.

H. Aprendizaje por Refuerzo

Una cualidad destacada de los seres vivos es su capacidad de aprender a resolver tareas a partir de experiencias positivas o negativas, también llamados a veces recompensas y castigos. Esta habilidad, permite a un ente prescindir de la respuesta correcta directamente, y hallar por sí mismo (o con ayuda de un guía) la solución. Gracias a la flexibilidad y sobre todo a la practicidad que otorga este tipo de aprendizaje, ha surgido en el área de la inteligencia artificial un campo que pretende imitar este comportamiento, este es el llamado aprendizaje por refuerzo.

Así pues, el aprendizaje por refuerzo surge de la necesidad de resolver tareas para las cuales poseer una base de datos etiquetada o conocer la respuesta correcta a priori no es viable o resulta extremadamente complicado. En este paradigma, un agente interactúa con su entorno y aprende de las consecuencias de sus acciones mediante un sistema de recompensas y castigos, muy similar al mecanismo de aprendizaje de los seres vivos. El objetivo es que el agente, a través de la experiencia y sin una guía explícita, sea capaz de descubrir estrategias que maximicen la suma de recompensas futuras, ajustando así sus decisiones a los desafíos que enfrenta. Cabe destacar que este tipo de aprendizaje ha cobrado cada vez más relevancia y ha sido uno de los factores clave que han permitido el desarrollo de los modelos más avanzados de IA hasta ahora, como GPT-4, AlphaGo, AlphaFold, etc.

De esta manera, a partir de esta sección se introducirá al lector al mundo del aprendizaje por refuerzo. Sin embargo, resulta conveniente familiarizarse antes con algunos términos importantes relacionados a este, de modo tal que se pueda entender mejor la idea detrás del funcionamiento de los algoritmos usados en esta área. Entre estos conceptos se encuentran los siguientes:

- Agente: Se mencionó en los párrafos anteriores. Se refiere al programa que se encarga de interactuar con el entorno y tomar decisiones dentro de él.
- Entorno: Es el escenario con el que el agente está en contacto y en donde se reflejan las consecuencias de sus decisiones. Puede ser determinista o estocástico [32].
- Espacio de estados: Se refiere a todas las situaciones posibles en que el entorno se puede encontrar. La instanciación de este es un simple estado.
- Espacio de observación: Su principal diferencia con el espacio de estados es que este está constituido por todas las partes de los estados que el agente puede percibir. Dicho esto, una observación podría estar compuesta por el estado completo del entorno o sólo un fragmento de este estado.
- Espacio de acciones: Está formado por el conjunto de todas las acciones que el agente tiene disponibles para elegir.
- Función de transición: Se encarga de cambiar el estado del entorno según la acción tomada por el agente; en otras palabras, define la dinámica de este ambiente.
- Función de recompensa: Es la responsable de dar las recompensas y castigos (recompensas negativas) correspondientes al agente según las decisiones que ha tomado.
- Episodio: Está compuesto por todas las pequeñas interacciones del agente desde que se despliega en el entorno hasta que gana o pierde.
- Retorno: Se refiere a la suma acumulada de recompensas del agente en un episodio.

Para una mejor compresión de estos conceptos considerar el siguiente ejemplo de un entorno llamado "*Frozen-Lake-v1*" de la librería Gymnasium en Python. En este entorno se presenta el problema de que un personaje debe alcanzar un recuadro específico para alcanzar una meta (representado con la figura de un regalo). Además de esto, el ambiente posee recuadros en los que hay trampas, por lo que la dificultad de la tarea no radica sólo en alcanzar la meta, sino también en que el agente debe aprender a evadir dichos obstáculos. Por último, aunado a esto, las decisiones del agente tienen cierto grado de aleatoriedad, por lo que una acción no te conducirá forzosamente al mismo estado en todas las ocasiones. Una captura de este entorno se presenta en la figura 8.

Figura 8. Captura del entorno *Frozen-Lake-v1*.

Entonces, realizando un análisis, se encuentra que los conceptos que se explicaron anteriormente poseen en este entorno, las siguientes particularidades:

- El espacio de estados está compuesto por 16 estados diferentes, uno por cada recuadro distinto en que el agente puede estar. Además, es un entorno estocástico debido a que el personaje puede llegar a diferentes posiciones aún habiendo tomado la misma decisión en las mismas coordenadas.

- El espacio de observaciones se compone de 16 valores distintos (uno por cada estado). En este caso (algo que no puede saber el lector a partir de la imagen) una observación es un simple número entero que distingue a cada recuadro y que el agente debe aprender a distinguir. Debido a esto se dice que son observaciones discretas.
- El espacio de acciones está formado por 4 decisiones distintas que corresponden a ir hacia la derecha, izquierda, arriba y abajo.
- La función de transición indica la probabilidad de llegar a un estado tomada una acción. En este caso para este ambiente es de 0.333 para los tres recuadros correspondientes a la dirección a la que el agente decidió ir. De esta manera si decide ir hacia la derecha existe el 0.333 de probabilidad de llegar justo al recuadro de la derecha, 0.333 de llegar al de la derecha-arriba y 0.333 a la derecha-abajo.
- La función de recompensa es simple. Se recibe +1 si se llega al objetivo y -1 si se cae en algún agujero.
- El episodio estaría compuesto por cada una de las acciones, recompensas y estados por los que se pasó desde el recuadro de origen hasta la meta o hasta caer en algún hueco.
- Por último, el retorno sería la recompensa acumulada en el episodio. Para este entorno sólo habría hasta el final un +1 o un -1.

Como se puede ver, los conceptos mencionados son sencillos de comprender. Y realmente resulta indispensable dominarlos a la perfección para poder definir correctamente las características del entorno que se debe crear al abordar un problema con aprendizaje por refuerzo.

I. Proceso de decisión de Markov

Hasta el momento, se han definido y explicado para el lector algunos de los elementos fundamentales del aprendizaje por refuerzo. Se observó que, en general,

un agente interactúa con un entorno, dicho entorno cambia según el comportamiento del agente y este último a su vez recibe una recompensa (negativa o positiva) por sus acciones. Sin embargo, para que el agente aprenda a maximizar estas recompensas a lo largo del tiempo (es decir, para que aprenda a tomar las mejores decisiones basadas en sus interacciones pasadas y actuales) es imprescindible contar con un modelo matemático que describa el entorno y la secuencia de decisiones de una manera formal y estructurada. Este modelo es conocido como Proceso de Decisión de Markov (MDP). El MDP es crucial porque proporciona el marco necesario para que el agente evalúe las consecuencias a largo plazo de sus acciones y elabore una estrategia que optimice sus recompensas acumuladas, lo que constituye la esencia del aprendizaje por refuerzo.

Un MDP se puede describir a través de una tupla con los elementos en (2) [32]:

$$< S, A, R(s,a), P(s'|s,a), \gamma > \quad (2)$$

Donde:

- S: un conjunto de estados
- A: un conjunto de acciones
- R(s,a): función de recompensa
- P(s'|s,a): función de transición
- γ: factor de descuento (constante utilizada para ajustar la importancia de las recompensas a lo largo del tiempo, siendo cercano a 0 para dar más peso a las recompensas inmediatas y cercano a 1 para priorizar las recompensas a largo plazo)

A la instanciación de estos conjuntos se le llama "experiencia" y se denota como (s_t, a_t, r_t, s_{t+1}) donde t es el instante de tiempo específico en el que se toma la decisión. De esta manera, utilizando esta notación, las interacciones agente-entorno descritas al principio se pueden visualizar siguiendo la figura 9:

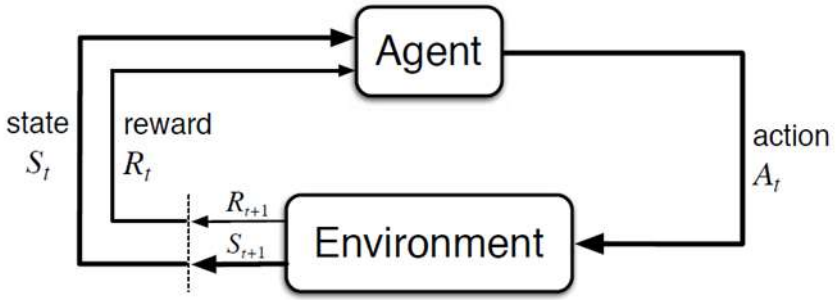

Figura 9. Interacción agente-entorno [33].

Así pues, se visualiza que el agente se enfrenta a una serie de decisiones secuenciales, en donde cada una de ellas lleva consigo una recompensa inmediata y consecuencias futuras. El programa debe maximizar el retorno obtenido por estas decisiones. Por desgracia, este no es un cálculo directo, pues maximizar las recompensas a corto plazo no siempre conduce a un resultado óptimo a largo plazo. (de hecho, en el entorno *Frozen-Lake-v1* ni siquiera existen recompensas intermedias). Es aquí donde entra en juego la noción de "función de valor".

J. Funciones de valor y políticas

En aprendizaje por refuerzo una política corresponde a una función usada por el agente para determinar las acciones a realizar [32]. Desde esta perspectiva, se puede definir como la estrategia que se sigue para resolver el problema. El principal objetivo del agente, entonces, es encontrar la política óptima que maximice la suma total de recompensas. La política se denota como $\pi(a|s)$ en donde se expresa la probabilidad de tomar la acción *a* cuando se está en el estado *s*. Además, al igual que varios de los conceptos anteriores, la política puede ser determinista o estocástica.

A partir de aquí, se deduce otro concepto muy importante, las funciones de valor. Una función de valor se puede entender como la suma esperada de recompensas futuras que el agente espera obtener. En otras palabras, una función de valor proporciona una estimación de lo que es "bueno" para el agente a largo plazo [32].

En aprendizaje por refuerzo se distinguen dos tipos, la función de valor de estado y la función de valor de acción:

- Función de valor de estado: Otorga una estimación de la suma de recompensas esperadas al seguir una política π a partir de un estado s. Se escribe como $V_{\pi}(s)$.
- Función de valor de acción: Realiza una estimación de la suma de recompensas esperadas al tomar una acción a desde un estado s y siguiendo una política π. Se denota $Q_{\pi}(s,a)$.

Es con ayuda de la estimación de estos valores, que un agente puede aprender a realizar una tarea, por ello son un aspecto clave de todos los algoritmos de aprendizaje por refuerzo.

Con el fin de dejar más claro el concepto y retomando el entorno *Frozen-Lake-v1* considerar el siguiente ejemplo:

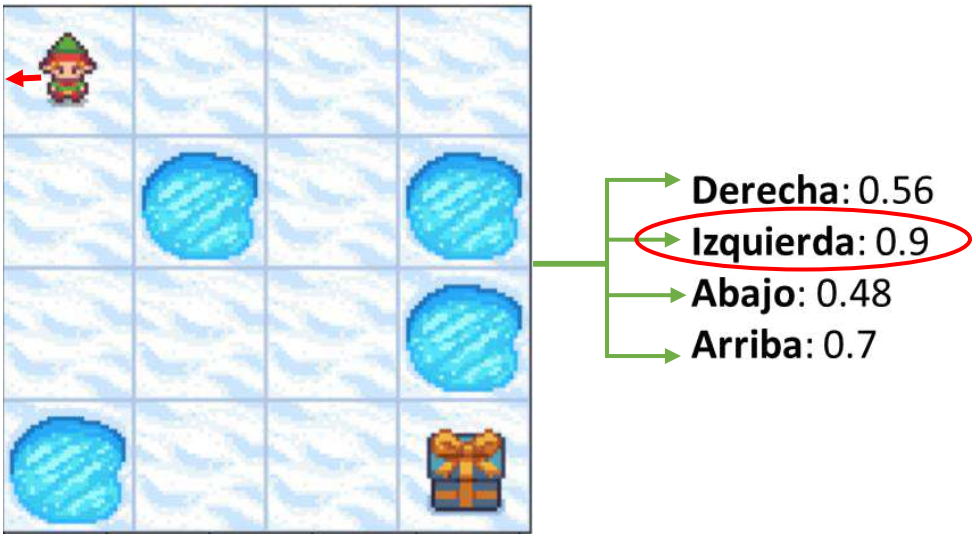

Figura 10. Valores de acción para el agente a partir del primer estado.

En la figura 10, se muestran los posibles valores de acción que tendría el agente en el entorno *Frozen-Lake*. Se puede observar que, si el valor de acción que se ha calculado es correcto, la mejor decisión que se puede tomar para el primer estado

del entorno sería ir a la izquierda. Esto tiene sentido si se toma en cuenta que es un entorno estocástico por lo que el ir hacia la derecha da una probabilidad de caer en el lago, al igual que ir hacia abajo, e intentar ir hacia arriba si bien podría conducir a un estado seguro próximo, a la larga expondría al agente a una situación con altas probabilidades de caer (se situaría cerca de dos agujeros). Así pues, la decisión menos arriesgada a largo plazo es ir hacia la izquierda, aun tomando en cuenta que podría demorarse la salida del agente de la primera columna de recuadros.

Por otra parte, la función de valor de estado resulta de manera similar en un número de qué tan bueno es estar en una situación específica. En el caso de *Frozen-Lake*, cada uno de los recuadros tendría un valor distinto y una correcta función de valor asignaría bajos valores a aquellos estados con altas probabilidades de caer y un alto valor a los estados cercanos a la meta.

K. Q-Learning

Una vez expuestas las bases y elementos fundamentales del aprendizaje por refuerzo, resultará de gran provecho estudiar, a manera de introducción, uno de los algoritmos más populares de este, el Q-learning.

En términos generales, el Q learning es un algoritmo que permite a un agente estimar la función de valor de acción óptima $Q_*(s,a)$. Esto se realiza de forma iterativa a través de los episodios usando la ecuación de Bellman para el Q-learning (3):

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[R(s_t, a_t) + \gamma \, max_{a'} Q\,(s_{t+1}, a) - Q(s_t, a_t)] \qquad (3)$$

Donde:

- α: Es la tasa de aprendizaje y permite controlar la magnitud de las actualizaciones.
- $R(s_t,a_t)$: Es la recompensa recibida al tomar la acción a_t en el estado s_t.

Una vez estimada la función *Q(s,a)*, los resultados se almacenan en una tabla llamada tabla Q, esto permite al agente tener una 'bitácora' a la que accederá para poder tomar decisiones según las estimaciones que se haya hecho para cada par estado-acción.

Otro aspecto importante del Q-learning es el manejo de la exploración y la explotación. La exploración se refiere al hecho de que un agente tome una decisión diferente a la que hasta el momento se ha calculado que es la mejor, la explotación es justamente lo contrario. El correcto balance de esto resulta indispensable, pues evita que el agente caiga en políticas subóptimas. La política que se sigue durante el entrenamiento para nivelar la exploración-explotación se le conoce como ε-greedy, en donde, con probabilidad ε el agente elige una acción al azar y con probabilidad $1-\varepsilon$ elige la acción que maximiza el valor actual *Q(s,a)*.

El Q-learning ha demostrado ser eficaz para converger a una política óptima. Sin embargo, a pesar de esto, también posee sus limitaciones, siendo la principal la dificultad que tiene para manejar espacios de estados muy grandes, ya que esto ocasionaría la generación de una tabla Q con dimensiones excesivas lo cual resultaría muy difícil o incluso imposible de manejar.

L. Deep Q-Network

Como se ha destacado, el enfoque tradicional del Q-learning involucra la creación de una tabla Q, que almacena y actualiza los valores asociados con cada par estado-acción. Este método es efectivo en entornos con un número manejable de estados, como el ya mencionado entorno *Frozen-Lake*. No obstante, en contextos donde el espacio de estados es extenso o infinito, como en la conducción autónoma, la gestión de una tabla Q extensa puede volverse inviable, tanto en términos de memoria como de eficiencia computacional. Además, la tabla Q enfrenta un desafío intrínseco: la incapacidad para generalizar a estados no visitados previamente, lo que limita severamente la versatilidad del agente en entornos dinámicos o desconocidos.

Para superar estas limitaciones, se desarrolló el DQN (Deep Q-Network), que incorpora el poder de las redes neuronales para la aproximación de la función Q. Este cambio permite al DQN manejar espacios de estado de gran escala y generalizar a partir de experiencias previas para inferir políticas adecuadas en estados no observados. La red neuronal, al funcionar como un aproximador de funciones, elimina la necesidad de almacenar una tabla Q, reduciendo la complejidad del espacio de almacenamiento y permitiendo una búsqueda más eficiente de las acciones óptimas.

Además, el DQN mejora significativamente la eficiencia del proceso de entrenamiento. A través de la propagación hacia adelante, la red neuronal evalúa todas las acciones posibles para un estado dado en una sola pasada, lo cual es sustancialmente más rápido que la actualización iterativa de los valores en una tabla. Este enfoque no solo acelera el aprendizaje, sino que también facilita la capacidad del agente para adaptarse y optimizar su comportamiento en entornos complejos y cambiantes. En la figura 11 se contrasta esta diferencia con el Q-learning.

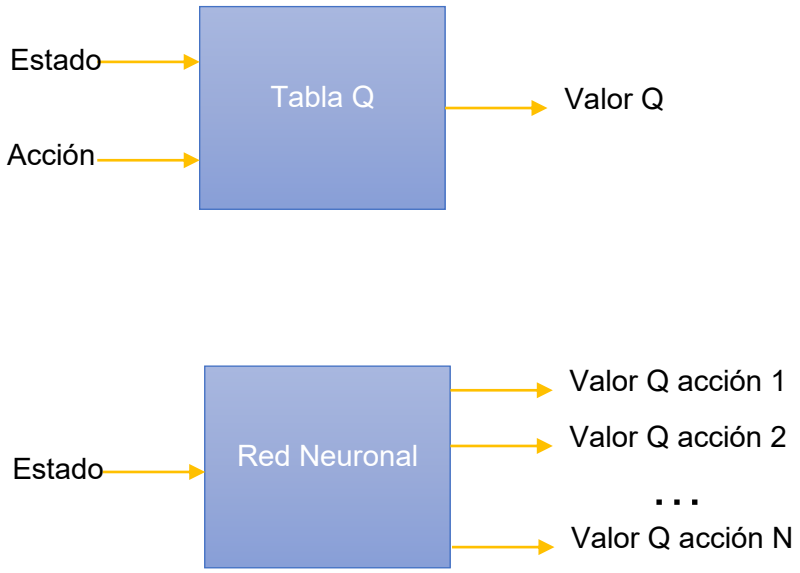

Figura 11. Diferencia en la forma de calcular los valores de acción entre el Q-learning y el DQN.

Un par de elementos más que resultan de gran importancia para este algoritmo, y que refuerzan su desempeño en el entrenamiento son el *experience replay* y la *target network*:

- *Experience Replay*: Es una técnica que rompe la correlación de las secuencias de observaciones al almacenar las experiencias del agente, es decir, las transiciones *(s, a, r, s')*, en un búfer de memoria conocido como replay buffer. Durante el proceso de entrenamiento, en lugar de actualizar la red con las transiciones en el orden en que ocurren, se toman muestras aleatorias de este búfer. Este muestreo aleatorio mitiga el riesgo de caer en mínimos locales durante el entrenamiento y mejora la estabilidad del aprendizaje.

- *Target Network*: Se utiliza para calcular el valor objetivo (target) en la actualización de la función Q. Esta target network es idéntica en arquitectura a la red principal que está siendo entrenada, pero sus pesos se actualizan con menos frecuencia. Al hacerlo, se proporcionan valores objetivo más consistentes y estables para las actualizaciones de la red principal, lo que ayuda a evitar oscilaciones y posibles divergencias en el aprendizaje. Por tanto, mientras la red principal, o red de evaluación, se utiliza para seleccionar la acción a ejecutar, la target network se emplea para generar los valores Q de referencia durante el proceso de actualización.

Se observa pues, que DQN resulta un algoritmo muy robusto y flexible, razón por la cual se eligió para el proyecto. Su eficacia al tratar con extensos espacios de estados y su competencia para generalizar a partir de experiencias previas lo convierten en una herramienta valiosa para enfrentar los retos inherentes a la movilidad autónoma, que es el foco de este estudio. En los capítulos subsiguientes, se explorará con precisión la implementación del DQN en el contexto de un robot móvil, detallando las herramientas empleadas y los resultados adquiridos a lo largo de las fases experimentales.

Capítulo 4. Hipótesis

El uso de una cámara RGB-D y un LiDAR 2D, en conjunto con métodos de aprendizaje máquina, proporcionan a un vehículo autónomo la percepción necesaria de su entorno que lo hace capaz de detectar obstáculos alrededor del vehículo y determinar una ruta para evadirlos.

Capítulo 5. Objetivos

A. Objetivo General

Desarrollar un sistema de detección de obstáculos y planeación de ruta usando una cámara RGB-D y un LiDAR 2D como sensores, y algoritmos de machine learning para su aplicación en un vehículo autónomo.

B. Objetivos Específicos

- Generar nubes de puntos del entorno de operación del autómata, a partir de los datos de profundidad arrojados por la cámara RGB-D y el LiDAR.
- Crear un entorno de simulación para el despliegue de los algoritmos antes de su aplicación en el robot real.
- Segmentar los objetos de las nubes de puntos con ayuda de algoritmos de machine learning, de tal forma que el vehículo tenga la capacidad de distinguir los obstáculos en el entorno.
- Implementar algoritmos de machine learning que permitan determinar la ruta a seguir para evadir los obstáculos usando la información de la cámara RGB-D y del sensor LiDAR 2D.

Capítulo 6. Materiales y Métodos

A. Hardware y software

El desarrollo de este proyecto implicó la integración de una serie de componentes de hardware y software seleccionados con el objetivo de crear un sistema robusto y eficaz para la movilidad autónoma. Estas herramientas no solo permitieron la operación efectiva del modelo/robot, sino que también facilitaron el proceso de desarrollo, entrenamiento y pruebas. Entre los componentes de hardware que se utilizaron se encuentran los siguientes:

- Yahboom Rosmaster R2: Constituye la base mecánica y de movilidad del proyecto. Posee una estructura Ackermann.
- Raspberry Pi 4: Sirve como el centro de procesamiento principal del robot en las pruebas en donde se utiliza exclusivamente el LiDAR.
- Jetson Nano 4G SUB: En las pruebas donde se incorpora la cámara RGB-D, este dispositivo actúa como el cerebro del sistema, proporcionando una potencia extra de cómputo.
- RPLIDAR A1M8: Sensor LiDAR 2D. Desempeña un papel crucial en la detección de obstáculos y la medición distancias.
- Intel Realsense D435: Cámara RGB-D usada como complemento de la observación dada por el LiDAR.

Por su parte, las herramientas de software sobre las que se construyó y entrenó el sistema de movilidad autónoma fueron:

- ROS Noetic: Opera como el sistema operativo del robot, así como de complemento para el entrenamiento del modelo.
- Gazebo: Como simulador, permitió el entrenamiento y la validación del modelo generado.

- *Packages* OpenAI-ROS y Ackermann_Vehicle: Estos *packages* desarrollados por [34], [35] permiten el primero, la implementación y creación de un *environment* de aprendizaje por refuerzo, y el segundo, el control de robots con estructura Ackermann en simulación.

B. Simulación

Como se explicó en capítulos anteriores, el aprendizaje por refuerzo (RL) implica un ciclo continuo de prueba y error, donde el agente interactúa con su entorno y ajusta sus parámetros en consecuencia. Esta metodología se adapta perfectamente a entornos virtuales, donde las interacciones ocurren completamente dentro de un espacio computacional. Sin embargo, la aplicación de RL en robótica presenta retos adicionales, ya que el entorno objetivo es el mundo real, lo que conlleva consideraciones significativas.

En primer lugar, el entrenamiento directamente en el entorno real puede resultar extremadamente lento y tedioso, pues cada episodio de entrenamiento requeriría ajustes manuales por parte del programador para preparar el lugar antes de la siguiente iteración. Por otra parte, resulta más crítico aún, el riesgo de daño físico al robot o a terceros debido al proceso inherente de prueba y error del RL, lo cual no implicaría solamente costos de reparación o reemplazo, sino que también podría tener implicaciones legales. Es por todo esto que, al entrenar agentes robóticos con RL, resulta de gran importancia la creación de una simulación que permita un entrenamiento más rápido y, además, evite daños físicos al robot o al entorno.

En este proyecto, la creación de una simulación efectiva para entrenar y evaluar el modelo implicó la elaboración meticulosa de dos componentes clave. El primero de estos fue el desarrollo de un archivo de descripción detallado para el robot. Este archivo no solo delineaba la estructura mecánica del robot, sino que también integraba virtualmente los sensores que serían utilizados en el mundo real. El segundo componente fundamental fue la creación de una serie de entornos simulados, o "mundos", que servirían como espacios de prueba para el robot. Estos

mundos fueron diseñados para simular una variedad de escenarios y condiciones físicas, permitiendo al robot interactuar y navegar en entornos que van desde configuraciones simples y controladas hasta contextos más complejos y desafiantes. A continuación, se presenta con más detalle cada uno de estos componentes.

Para el archivo de descripción del robot se decidió utilizar el formato URDF, esto debido a su flexibilidad y simplicidad para definir cada elemento mecánico del autómata. Cabe aclarar que, si bien Yahboom otorga un archivo URDF de descripción, este se tuvo que modificar mucho (integración de controladores, corrección de matrices de inercia y modificación de los archivos STL) para poder obtener un robot virtual que funcionara correctamente en Gazebo. En la figura 12 se observa el modelo obtenido:

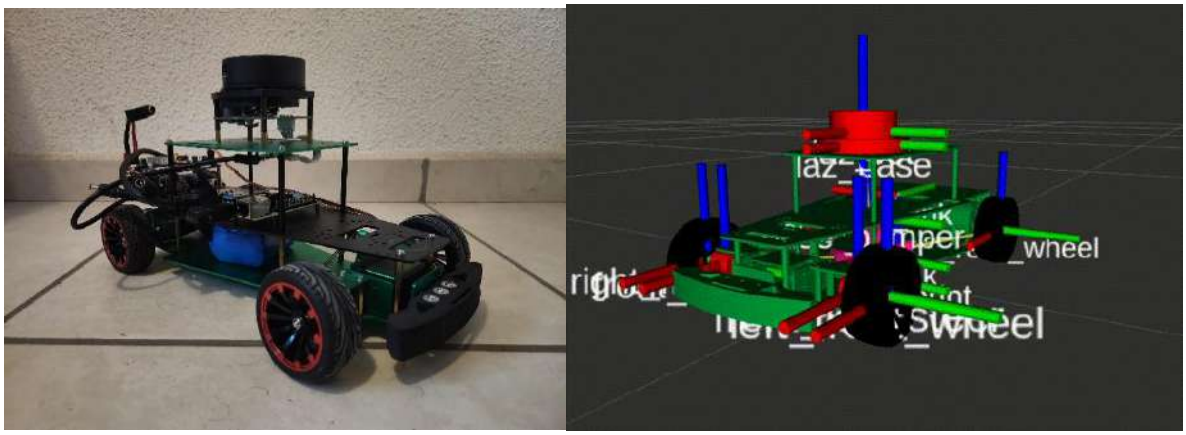

Figura 12. Rosmaster R2 y su descripción en URDF.

En esta misma línea, con el fin de que el desempeño del robot simulado correspondiera lo más posible con el físico, se configuraron de manera exhaustiva los sensores virtuales, los cuales correspondían al LiDAR, IMU y un sensor llamado *bumper*.

Con respecto al sensor LiDAR, la tabla 2 muestra los parámetros definidos para el sensor; se puede observar que son muy similares a los del RPLIDAR A1M8 usado en el robot físico.

Característica	Unidad	Valor/Rango Real	Valor/Rango Virtual
Tasa de escaneo	Hz	5.5	6
Rango de Distancia	Metro (m)	0.15-12	0.15-12
Ángulo de medición	Grados	0-360	0-360
Resolución	m	<1% de la distancia	0.015 m

Tabla 2. Especificaciones LiDAR virtual y físico [36].

Por su parte, la configuración del sensor IMU y del *bumper* resultó menos compleja. El sensor IMU (encargado de otorgar datos de velocidad y orientación) se configuró a una velocidad de operación de 50 Hz. El sensor *bumper* por otro lado, es el nombre dado a un dispositivo virtual en Gazebo que permite detectar colisiones; aparte de la correcta elección de dimensiones, este no requiere configuraciones adicionales.

Finalmente, con respecto a los *worlds* creados para el entrenamiento y evaluación del robot, se utilizaron especialmente tres. El primero de ellos, utilizado para el entrenamiento de orientación (etapa que se detallará más adelante), fue un mundo vacío sin obstáculos que Gazebo usa por defecto. El segundo, creado para un entrenamiento sencillo de evasión de obstáculos, consistía en una habitación con objetos comunes como cubos, paredes y sillas. Por último, se creó un segundo *world* específicamente para la evaluación del modelo, en el que se intentó recrear uno de los salones de la institución. La figura 13 muestra los dos mundos creados.

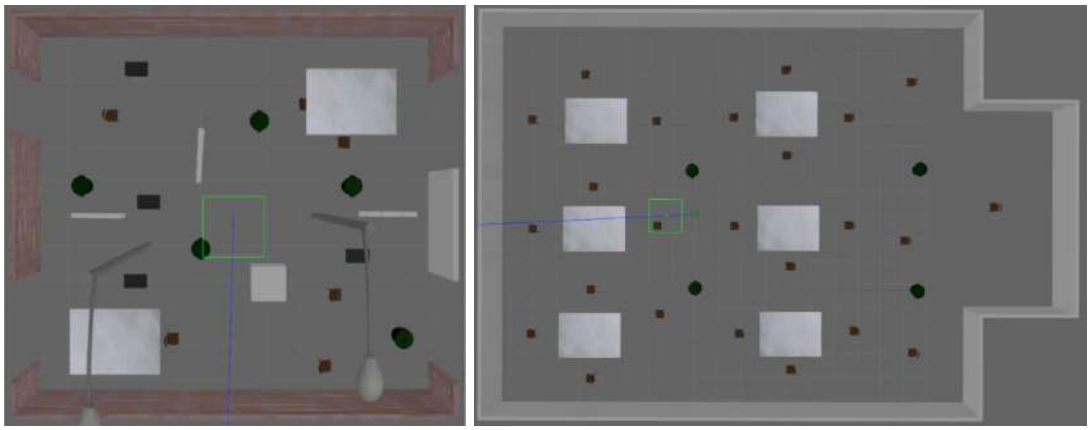

Figura 13. *Worlds* de entrenamiento (izquierda) y de evaluación (derecha).

C. Environment de entrenamiento

La implementación de aprendizaje por refuerzo para un robot requiere, no solamente de la creación de espacios y modelos virtuales, sino de una etapa aún más crucial para el correcto aprendizaje del modelo, esto es la definición de los espacios de observación, espacios de acción y la función de recompensas.

Como se vio anteriormente, el espacio de observación representa aquello que el agente puede 'ver' del entorno ya sea imágenes, escaneos, etc. El espacio de acciones son el conjunto de decisiones entre las que el modelo puede escoger. Y, finalmente, la función de recompensas define la manera en la que se castiga o recompensa al programa. A continuación, se detalla cómo se definió cada una de estas.

El espacio de observaciones resultó el más simple de definir para el proyecto. Se decidió que este estuviera compuesto por las tres últimas mediciones del LiDAR y los tres últimos ángulos de orientación que el robot había tenido hacia la meta (x, y). Esto se decidió así para poder otorgarle al agente cierto sentido de temporalidad [16].

Un esquema del espacio de observación definido se puede visualizar en la figura 14.

Figura 14. Espacio de observación del robot.

Por su parte, el espacio de acciones se visualiza en la tabla 3. Observar que, en primer lugar, se definió como un espacio discreto, es decir, que los valores de velocidad son valores fijos y no existen valores intermedios; esto debido a que DQN sólo permite trabajar con este tipo de espacios de acción. Además, es importante visualizar que, con el motivo de acelerar el proceso de aprendizaje, se definieron velocidades que permitieran ir hacia adelante o doblar, pero no ir en reversa o detenerse. Para la evaluación con el robot físico, se definió una política aparte que detuviera al vehículo a momento de llegar a la meta.

Velocidad (m/s)	1	2	3	4	5	6	7
Lineal	0.8	0.8	0.8	1.0	0.5	0.5	0.5
Angular	0.0	0.6	-0.6	0.0	0.0	0.4	-0.4

Tabla 3. Espacio de acciones del robot.

Por último, la función de recompensas fue por mucho la más difícil de establecer, algo bastante común al utilizar RL, pues requiere analizar el comportamiento del agente durante el entrenamiento y tener muy claros los

objetivos que se desean alcanzar y cómo se quieren alcanzar. La función obtenida, permitió que el agente aprendiera a llegar al objetivo siguiendo, en la mayoría de las ocasiones, una línea recta. Además, gracias a esta se logró igual que el agente pudiera evadir obstáculos pequeños y medianos. La función es la sumatoria (4) de las expresiones (5), (6), (7) y (8):

$$R = r_1 + r_2 + r_3 + r_4 \tag{4}$$

Donde:

$$r_1 = \begin{cases} 0.01 & si\ v_l \neq 0\ y\ v_a = 0 \\ -0.01 & si\ v_a \neq 0 \end{cases} \tag{5}$$

$$r_2 = -(1 - d_{min}) \quad si\ d_{min} < 1\ metro \tag{6}$$

$$r_3 = (|\theta_{ultimo}| - |\theta_{actual}|) \times 10 \tag{7}$$

$$r_4 = \begin{cases} -20 & si\ hubo\ colisión \\ +20 & si\ se\ alcanza\ el\ objetivo \end{cases} \tag{8}$$

Donde:

- v_l = velocidad lineal
- v_a = velocidad angular
- d_{min} = distancia más pequeña en la medición láser

Si se analizan las ecuaciones, se observa que la ecuación r_1 tiene como objetivo motivar al robot a seguir una línea recta, pues lo castiga ligeramente si dobla. La función r_2, se encarga de que el robot evite acercarse a los obstáculos. La ecuación

r_3 incentiva al robot a mantener su orientación hacia la meta en el ángulo correcto. Finalmente, r_4 recompensa al agente al final del episodio, ya sea porque se haya alcanzado la meta o se haya colisionado con algún objeto.

Una vez definido el environment, sólo queda definir los detalles del algoritmo que se utilizará, desde la arquitectura de la red neuronal, pasando por el procesamiento de la observación y concluyendo con el entrenamiento y evaluación del modelo resultante. Los siguientes apartados detallan las decisiones tomadas para esto.

D. Arquitectura de la red neuronal y preprocesamiento de la observación

Como se explicó anteriormente, el espacio de observaciones está compuesto de las tres últimas mediciones del LiDAR y de los tres últimos ángulos que el agente tuvo hacia el objetivo. Para que el modelo pudiera extraer información de estos datos, se construyó una red neuronal que recibiera dichos datos en dos ramas distintas; y al final se unieran en una última rama encargada de predecir las acciones. La figura 15 da una muestra de la red construida.

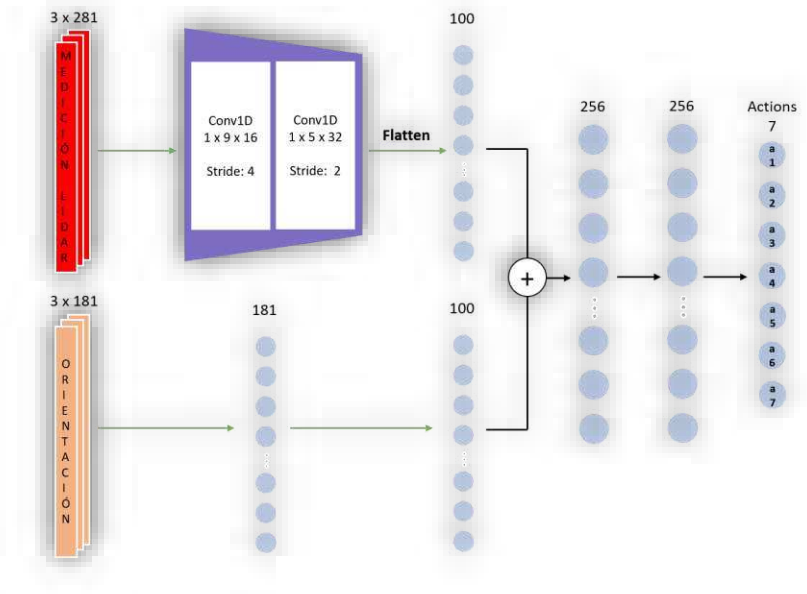

Figura 15. Arquitectura de la red neuronal. Basado en [16].

Entrando más a detalle en la red, la primera rama es la encargada de procesar las mediciones del LiDAR y se compone de dos capas convolucionales 1D con 16 y 32 canales de salida respectivamente y una capa densa al final. La segunda rama recibe los ángulos de orientación hacia el objetivo y consiste en una red *Fully Connected* (FCN). En la sección final, se unen todas las características extraídas y son procesadas por otra FCN encargada de predecir la siguiente mejor acción a realizar. Toda la red utiliza funciones de activación ReLU.

Un aspecto relevante al trabajar con modelos de *Machine Learning*, es definir el método que se utilizará para preprocesar los datos antes de que sean analizados por el algoritmo principal. Esto se hace con el fin de que el modelo pueda extraer las características relevantes de manera más sencilla, permitiendo incrementar su velocidad de aprendizaje e, igualmente, su eficacia en el desempeño de las tareas correspondientes.

El preprocesamiento de los datos se realizó de manera distinta para los escaneos láser y para los ángulos de orientación. Los métodos se describen a continuación:

- Preprocesamiento de mediciones láser: Para preparar los datos del sensor láser para su análisis, se efectuaron dos pasos esenciales. Inicialmente, se abordó el tratamiento de los valores de 'infinito' o '*inf*', los cuales aparecen cuando el láser no registra ningún obstáculo en su zona de operación; estos se reasignaron a un valor numérico de -1 para mantener la coherencia en el conjunto de datos y permitir que el modelo diferenciará con mayor claridad las diferencias entre zonas con objetos y zonas vacías. Posteriormente, se procedió a la normalización, dividiendo todas las mediciones por el umbral máximo del sensor, que es de 12 metros, con el fin de estandarizar el rango de los datos y facilitar su posterior análisis computacional.

- Preprocesamiento de orientación: La medición de la orientación del robot se realiza en un rango que va desde -180° a la izquierda hasta 180° a la derecha. Para la integración de estos datos en la red neuronal, se implementó un esquema de codificación binaria. Se utilizó un vector con 180 posiciones, donde un solo bit se establece en '1' para representar el ángulo de giro necesario en grados. Para denotar la dirección del giro, se incorporó un bit adicional al final del vector: un '1' indica un giro hacia la derecha y un '0' hacia la izquierda. Así, después de la codificación, cada vector de orientación comprende 181 bits. Un ejemplo de un vector final sería [0, 1, 0, 0, …, 0]; en este caso, se interpreta que el robot debe efectuar un ajuste mínimo hacia la izquierda, correspondiente a un ángulo de -1°.

E. Metodología de entrenamiento

Usualmente, al usar RL en robótica, una vez que se ha definido todo lo necesario, se lanza al agente a resolver el problema de forma directa. Sin embargo, en el transcurso de las pruebas este enfoque impidió conseguir buenos resultados en el aprendizaje. Es por esto que, una de las principales ideas propuestas en el presente trabajo, es una metodología de entrenamiento para DQN dividida en dos etapas, un entrenamiento de orientación y un entrenamiento de evasión (figura 16).

Figura 16. Metodología de entrenamiento.

La idea detrás de esta metodología de entrenamiento es bastante intuitiva: se basa en desarrollar habilidades específicas de manera independiente y secuencial dentro de la red neuronal. El proceso comienza con el entrenamiento de la rama encargada de la orientación, asegurando que el robot primero aprenda a navegar hacia su objetivo de manera efectiva; y culmina con un enfoque en el entrenamiento de la rama convolucional, cuyo papel es identificar y esquivar los obstáculos presentes en el medio.

Detallando más el proceso de entrenamiento, inicialmente, el robot se sitúa en un entorno virtual simplificado, sin obstáculos (el *empty world* mencionado antes), donde se enfoca en alcanzar dos metas diferentes. Este escenario está diseñado para que el robot experimente y aprenda de una amplia gama de ángulos de navegación, lo que fomenta una mejor generalización de su capacidad de orientación. Una vez que esta fase de entrenamiento concluye y se considera que el robot ha adquirido un entendimiento suficiente de los patrones de orientación, se procede a "congelar" o fijar los parámetros de la rama de orientación de la red. Este paso se basa en la idea de que ya se han identificado los patrones esenciales para una orientación precisa y que cualquier ajuste adicional podría introducir ruido no deseado, afectando negativamente la eficiencia de la red.

Con los parámetros de orientación establecidos, el entrenamiento avanza hacia la fase de evasión. En esta etapa, el robot no solo tiene el objetivo de llegar a un punto determinado, sino que también debe aprender a identificar y evitar obstáculos en su camino. Esta fase es crucial para desarrollar habilidades de navegación más complejas y adaptativas, permitiendo al robot operar de manera efectiva en entornos más realistas y desafiantes.

En las siguientes secciones, se profundizará en el análisis de los resultados obtenidos en cada una de las etapas de entrenamiento. Se detallará cómo la fase de orientación permitió desarrollar la capacidad del robot para navegar hacia los objetivos y cómo la congelación de parámetros y el entrenamiento de evasión fomentaron en el agente la habilidad de percibir y esquivar objetos.

Capítulo 7. Resultados y discusión

A. Entrenamiento de orientación

Como se dijo anteriormente, el entrenamiento de orientación del robot se llevó a cabo en un entorno sin obstáculos. Los hiperparámetros utilizados para este entrenamiento se muestran en la tabla 4:

Configuración/Hiperparámetro	Valor
Umbral al objetivo	0.2 m
Límite de timesteps para un episodio	2,000
Recompensa promedio objetivo (últimos 10 episodios)	16
Learning rate	$1x10^{-6}$
Factor de descuento γ	0.99
ε inicial	1.0
ε decay	0.999987
ε mínimo	0.02
Tamaño de experience replay	40,000
Batch size	64
Timesteps para sincronización del target network	2,000

Tabla 4. Hiperparámetros del entrenamiento de orientación.

Posterior a este se obtuvo una gráfica como la de la figura 17, en la que se observan las recompensas por episodio junto con las recompensas promedio de los últimos 10 episodios.

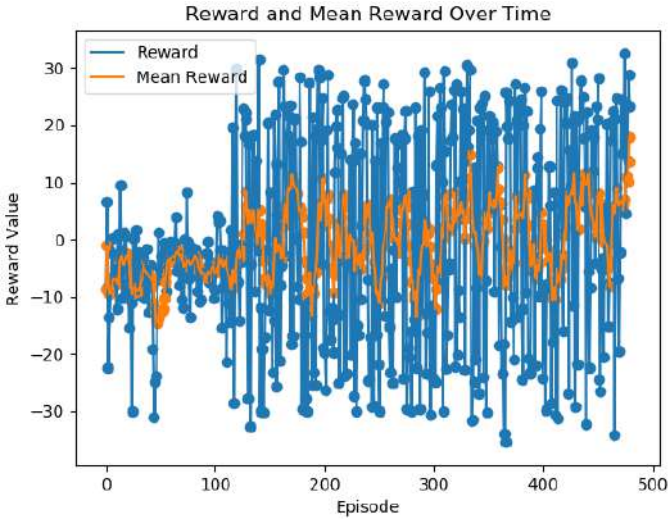

Figura 17. Gráfica de recompensas por episodio (azul) y recompensas promedio por episodio (naranja) del entrenamiento de orientación.

Una vez terminado el entrenamiento se procedió a evaluar el modelo en una prueba en la que tenía que recorrer una trayectoria de muchos objetivos en el entorno virtual. Cabe recordar que la meta final es que el robot logré llegar a un punto en específico y no a varios, por lo que este recorrido no fue visto durante el entrenamiento y supone, por tanto, un reto interesante para el agente. En la figura 18 se presenta una gráfica con el porcentaje de éxitos alcanzado y una de las trayectorias generadas.

Figura 18. Porcentaje de éxitos para 10 episodios (izquierda) y ejemplo de trayectoria (derecha).

Por último, se realizaron algunas pruebas de este primer modelo en un entorno real, mostrando, entre otros, el resultado de la figura 19. Se debe aclarar que el eje ‘x’ positivo se encuentra a lo largo del frente del robot y el eje ‘y’ positivo hacia la izquierda en un ángulo de 90 grados con respecto al primero:

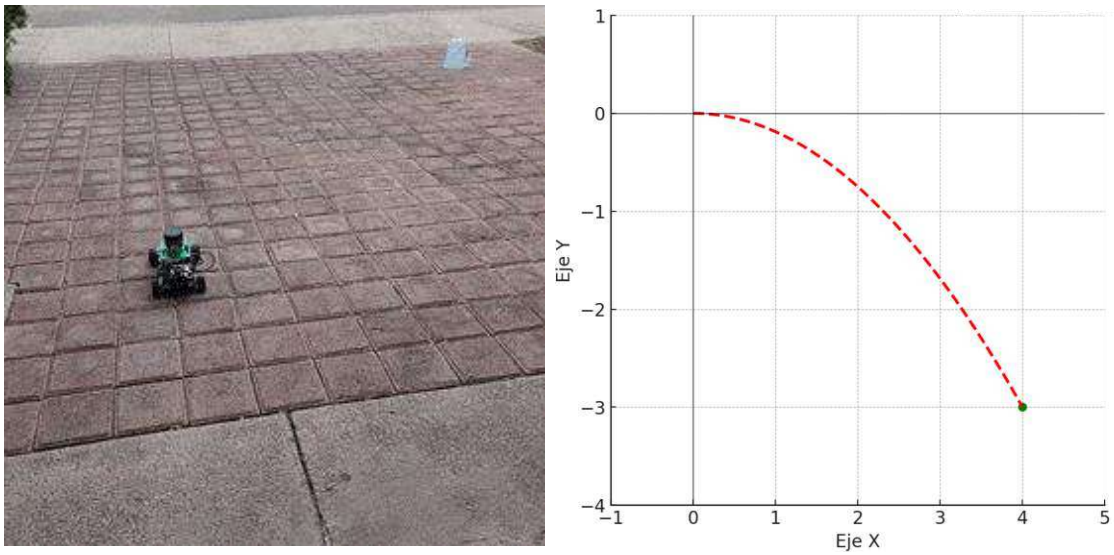

Figura 19. Trayectoria generada en un entorno real para un objetivo marcado por el objeto azul.

B. Entrenamiento de evasión

Por su parte, para el entrenamiento de evasión se realizaron dos entornos, uno exclusivo de entrenamiento y otro más para evaluar el desempeño del agente, esto último en particular para medir la capacidad de generalización del vehículo ante lugares no vistos anteriormente. Ambos entornos se muestran en la figura 13.

La configuración de hiperparámetros se muestra en la tabla 5.

Configuración/Hiperparámetro	Valor
Umbral al objetivo	0.3 m
Límite de timesteps para un episodio	3,000
Recompensa promedio objetivo (últimos 10 episodios)	23
Learning rate	$1x10^{-5}$
Factor de descuento γ	0.99
ε inicial	1.0
ε decay	0.999987
ε mínimo	0.02
Tamaño de experience replay	40,000
Batch size	64
Timesteps para sincronización del target network	2,000

Tabla 5. Hiperparámetros del entrenamiento de evasión.

La gráfica de recompensas por episodio y recompensas promedio resultantes se muestra en la figura 20.

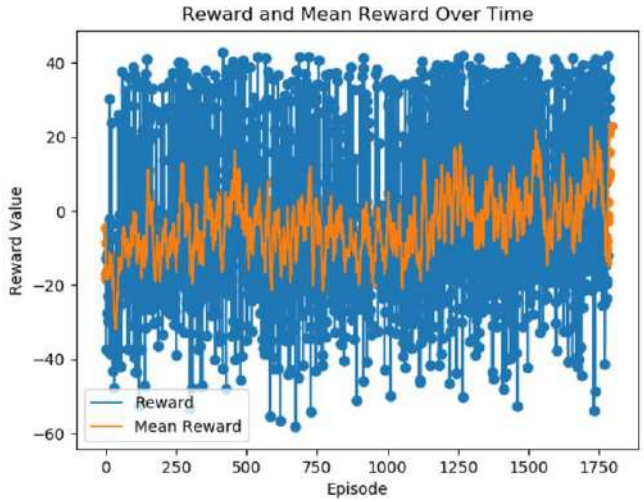

Figura 20. Gráfica de recompensas por episodio (azul) y recompensas promedio por episodio (naranja) del entrenamiento de evasión.

Al igual que en el entrenamiento de orientación, se llevó a cabo la evaluación del modelo, pero utilizando el segundo entorno virtual de la figura 13. Los resultados se muestran en las figuras 21, 22 y 23.

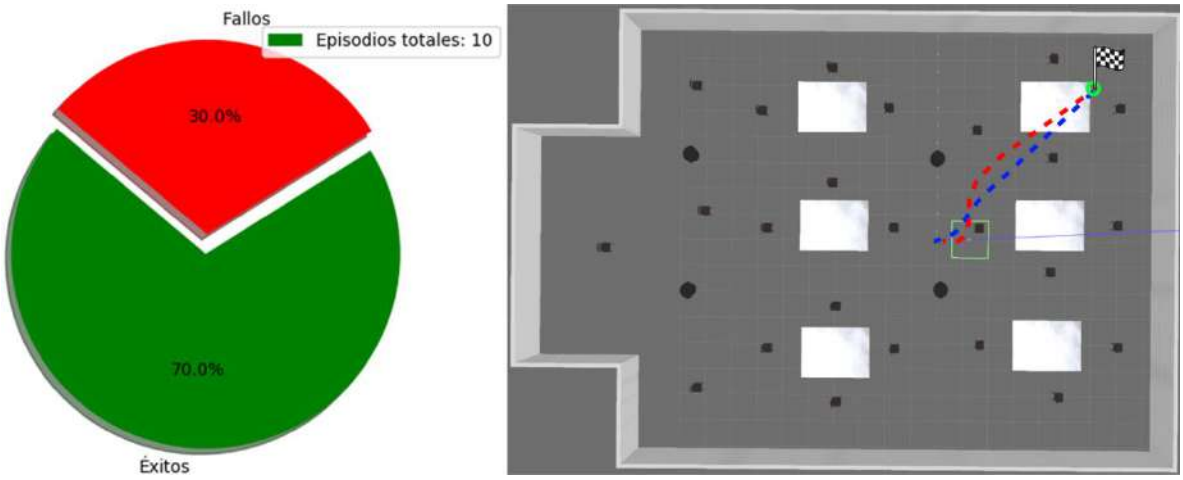

Figura 21. Porcentaje de éxitos para 10 episodios (izquierda) y comparación de trayectoria generada para el objetivo x=6 y=6 (imagen derecha en color rojo) vs trayectoria ideal más corta (imagen derecha en color azul).

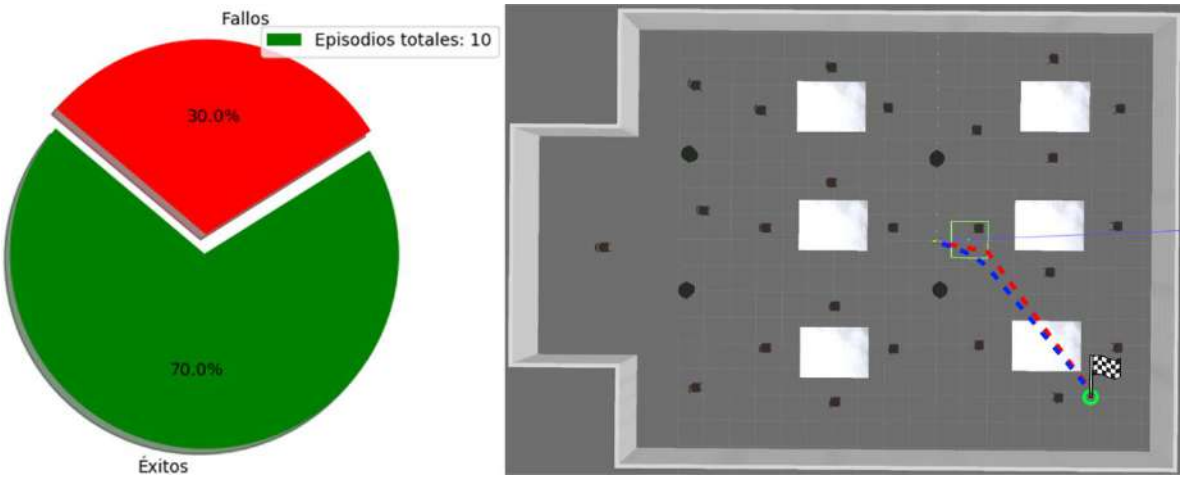

Figura 22. Porcentaje de éxitos para 10 episodios (izquierda) y comparación de trayectoria generada para el objetivo x=-6 y=-6 (imagen derecha en color rojo) vs trayectoria ideal más corta (imagen derecha en color azul).

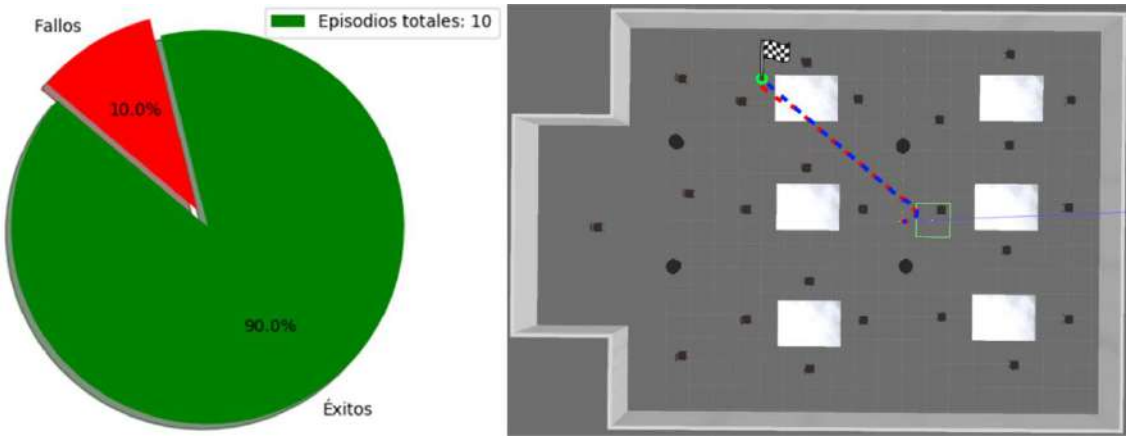

Figura 23. Porcentaje de éxitos para 10 episodios (izquierda) y comparación de trayectoria generada para el objetivo x=-6 y=6 (imagen derecha en color rojo) vs trayectoria ideal más corta (imagen derecha en color azul).

Además de estas pruebas, se procuró llevar al agente al extremo asignándole la tarea de tener que llegar a diversos puntos en el mapa. Recordar que el robot no está entrenado explícitamente para el alcance de diversos objetivos. El resultado se muestra en la figura 24.

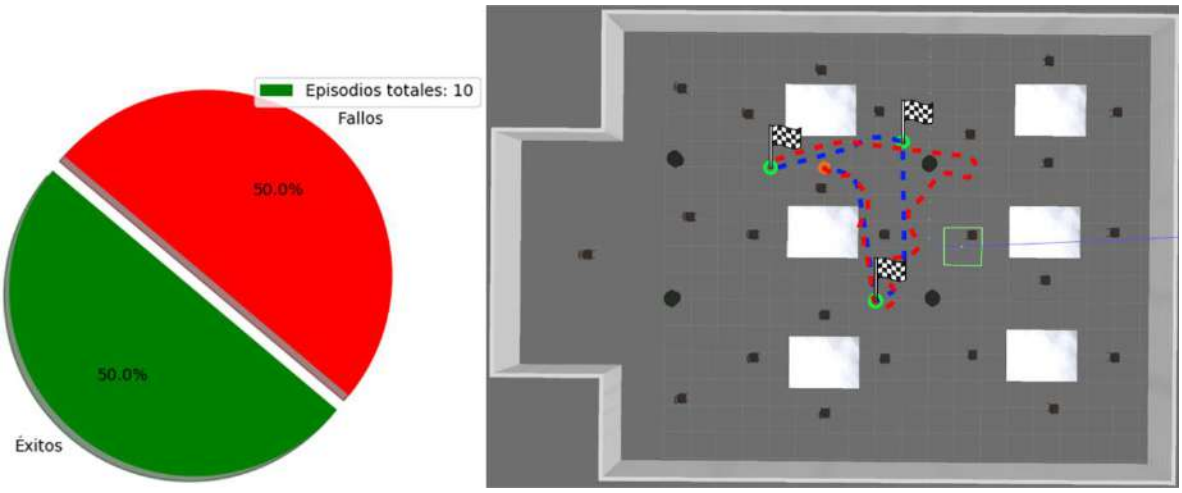

Figura 24. Porcentaje de éxitos para 10 episodios (izquierda) y comparación de trayectoria generada (imagen derecha en color rojo) vs trayectoria ideal más corta (imagen derecha en color azul).

Finalmente se realizaron también pruebas con el robot físico, consiguiendo los resultados de las figuras 25 y 26:

Figura 25. Trayectoria generada hacia el objetivo x=2 y=-1.

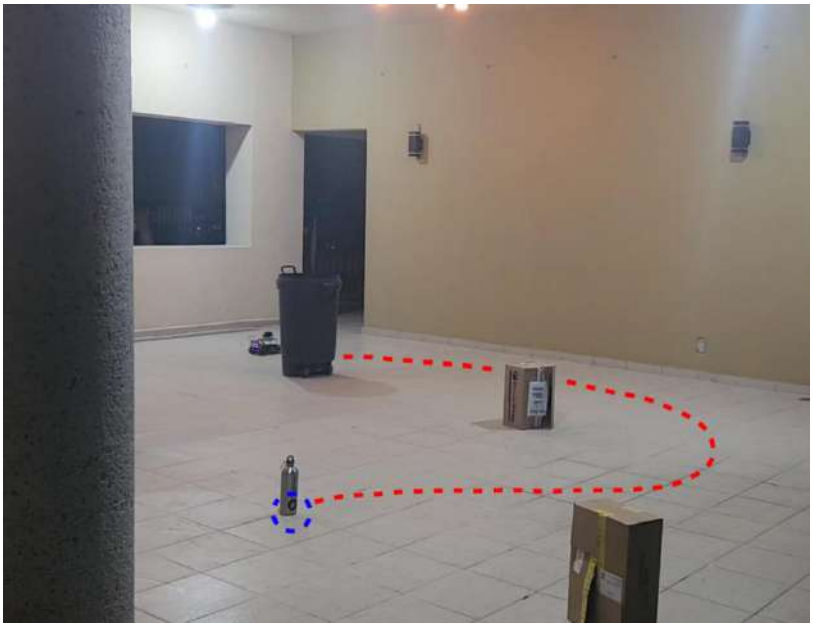

Figura 26. Trayectoria generada hacia el objetivo x=-1.5 y=-4.

C. Integración de la cámara

Finalmente, la última etapa del proyecto consistió en integrar la cámara RGB-D, de tal manera que el robot no estuviera limitado a las detecciones en un solo plano dadas por el LiDAR 2D. Para lograr esto se tenía como condición, el poder utilizar la misma red neuronal que se había entrenado antes, por lo que se debía convertir el canal de profundidad a un escaneo láser y posteriormente agruparlo con las observaciones del LiDAR (basado en [16]).

La herramienta de software que se utilizó para realizar este procesamiento fue el *package* ***depth_image_to_laserscan*** [37]. En términos generales esta herramienta funciona de la siguiente manera:

1) se recibe el último canal de profundidad generado por la cámara,
2) se toma una o más filas de pixeles para generar un escaneo láser a partir de las distancias que cada uno de estos pixeles calculó,
3) en caso de haber utilizado muchas filas se elige la distancia más pequeña de cada columna para colocarla en el vector de medición esto evita colisiones contra objetos con formas más irregulares.

El resultado de este proceso se puede visualizar en la figura 27.

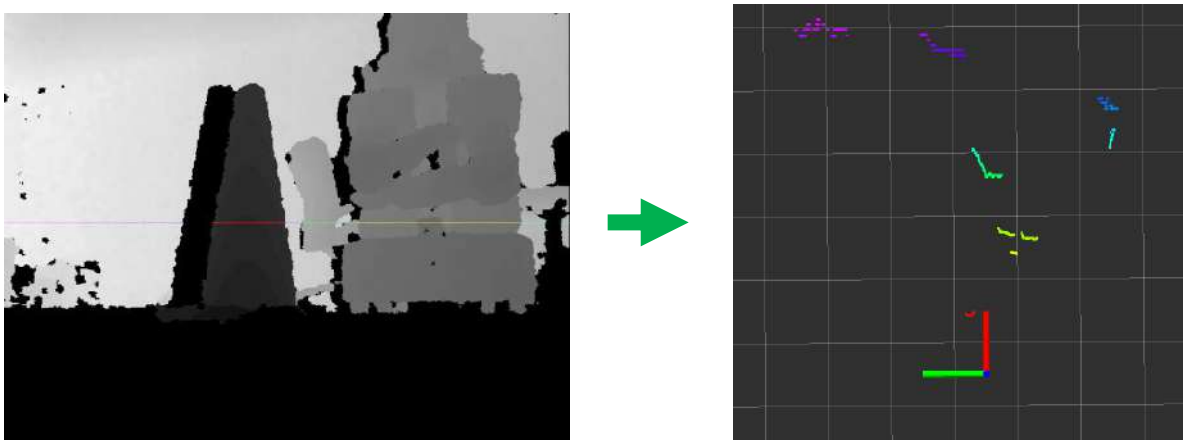

Figura 27. Proceso de conversión del *package depthimage_to_laserscan*.

Finalmente, para combinar las mediciones de ambos sensores se escogen las distancias mínimas. Este método resultó lo suficientemente efectivo como para evadir obstáculos que estaban fuera de la vista del LiDAR debido a su altura, como se ve en las figuras 28 y 29.

Figura 28. Trayectoria generada hacia el objetivo x=2.5 y=-0.5.

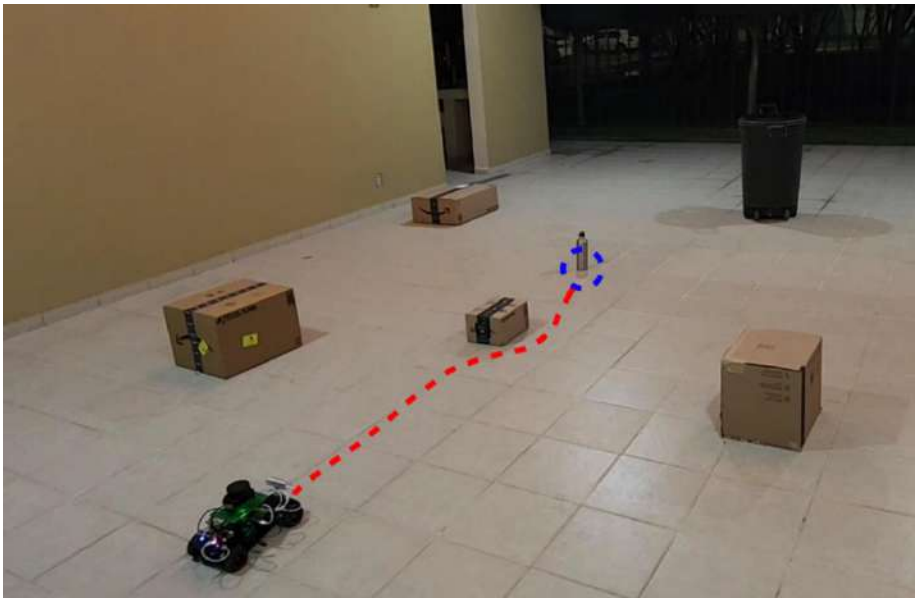

Figura 29. Trayectoria generada hacia el objetivo x=2.5 y=0.

Se puede ver, por tanto, que los resultados del presente proyecto resultaron satisfactorios y que, además, la metodología de entrenamiento propuesta resultó efectiva para el entrenamiento de un robot con configuración Ackermann a través del algoritmo DQN.

Capítulo 8. Conclusiones

En el presente documento, se ha presentado el fundamento teórico y la metodología seguida para el desarrollo de un sistema de evasión de obstáculos y alcance de objetivos para un robot móvil. Se dio un breve repaso por los conceptos fundamentales y posteriormente se detallaron las configuraciones llevadas a cabo para el entrenamiento de dicho modelo.

Además de esto, se propuso un esquema de entrenamiento para un robot tipo Ackermann usando el algoritmo DQN consistente en una etapa de entrenamiento de orientación y otra de evasión de obstáculos, lo que permite al robot aprender de forma adecuada la tarea final.

Igualmente se detalló una variación en la arquitectura de la red neuronal presentada por [16], que permite leer las observaciones de orientación y mediciones láser en dos ramas distintas. E igualmente una función de recompensas capaz de otorgar al agente un comportamiento adecuado para la tarea de navegación autónoma.

Finalmente, aunque los resultados fueron aceptables, se proponen como trabajo futuro, el aumento en el espacio de acciones del robot que le permita tener una mayor flexibilidad en su movimiento (quizá incluyendo la reversa) y/o la implementación de algoritmos que permitan manejar espacios de acción continuos con el objetivo de que el robot pueda especificar una velocidad más precisa y variable según la situación del entorno.

Capítulo 9. Bibliografía

[1] M. Duguleana and G. Mogan, “Neural networks based reinforcement learning for mobile robots obstacle avoidance,” *Expert Syst Appl*, vol. 62, pp. 104–115, 2016, doi: https://doi.org/10.1016/j.eswa.2016.06.021.

[2] R. Cimurs, J. H. Lee, and I. H. Suh, “Goal-Oriented Obstacle Avoidance with Deep Reinforcement Learning in Continuous Action Space,” *Electronics (Basel)*, vol. 9, no. 3, p. 411, Feb. 2020, doi: 10.3390/electronics9030411.

[3] Bing-Qiang Huang, Guang-Yi Cao, and Min Guo, “Reinforcement Learning Neural Network to the Problem of Autonomous Mobile Robot Obstacle Avoidance,” in *2005 International Conference on Machine Learning and Cybernetics*, IEEE, 2005, pp. 85–89. doi: 10.1109/ICMLC.2005.1526924.

[4] J. Pu, Y. Jiang, X. Xie, X. Chen, M. Liu, and S. Xu, “Low cost sensor network for obstacle avoidance in share-controlled smart wheelchairs under daily scenarios,” *Microelectronics Reliability*, vol. 83, pp. 180–186, Apr. 2018, doi: 10.1016/j.microrel.2018.03.003.

[5] J. Woo, J. Lee, and N. Kim, “Obstacle avoidance and target search of an Autonomous Surface Vehicle for 2016 Maritime RobotX challenge,” in *2017 IEEE Underwater Technology (UT)*, IEEE, 2017, pp. 1–5. doi: 10.1109/UT.2017.7890308.

[6] D. Wang, W. Li, X. Liu, N. Li, and C. Zhang, “UAV environmental perception and autonomous obstacle avoidance: A deep learning and depth camera combined solution,” *Comput Electron Agric*, vol. 175, p. 105523, Aug. 2020, doi: 10.1016/j.compag.2020.105523.

[7] J. Lin, H. Zhu, and J. Alonso-Mora, “Robust Vision-based Obstacle Avoidance for Micro Aerial Vehicles in Dynamic Environments,” in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2020, pp. 2682–2688. doi: 10.1109/ICRA40945.2020.9197481.

[8] S. Back, G. Cho, J. Oh, X.-T. Tran, and H. Oh, “Autonomous UAV Trail Navigation with Obstacle Avoidance Using Deep Neural Networks,” *J Intell Robot Syst*, vol. 100, no. 3–4, pp. 1195–1211, Dec. 2020, doi: 10.1007/s10846-020-01254-5.

[9] T. Eppenberger, G. Cesari, M. Dymczyk, R. Siegwart, and R. Dube, “Leveraging Stereo-Camera Data for Real-Time Dynamic Obstacle Detection and Tracking,” in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, IEEE, Oct. 2020, pp. 10528–10535. doi: 10.1109/IROS45743.2020.9340699.

[10] C. Zhou, F. Li, W. Cao, C. Wang, and Y. Wu, “Design and implementation of a novel obstacle avoidance scheme based on combination of CNN-based deep learning method and liDAR-based image processing approach,” *Journal of Intelligent & Fuzzy Systems*, vol. 35, no. 2, pp. 1695–1705, Aug. 2018, doi: 10.3233/JIFS-169706.

[11] S. Sarkar, S. N. Shome, and S. Nandy, “An Intelligent Algorithm for the Path Planning of Autonomous Mobile Robot for Dynamic Environment,” 2010, pp. 202–209. doi: 10.1007/978-3-642-15810-0_26.

[12] H. Song, K. Lee, and D. H. Kim, “Obstacle Avoidance System with LiDAR Sensor Based Fuzzy Control for an Autonomous Unmanned Ship,” in *2018 Joint 10th International Conference on Soft Computing and Intelligent Systems (SCIS) and 19th International Symposium on Advanced Intelligent Systems (ISIS)*, IEEE, Dec. 2018, pp. 718–722. doi: 10.1109/SCIS-ISIS.2018.00119.

[13] J. Cerbaro, D. Martinelli, A. S. de Oliveira, and J. A. Fabro, “WaiterBot: Comparison of Fuzzy Logic Approaches for Obstacle Avoidance in Dynamic Unmapped Environments Using a Laser Scanning System (LiDAR),” in *2020 Latin American Robotics Symposium (LARS), 2020 Brazilian Symposium on Robotics (SBR) and 2020 Workshop on Robotics in Education (WRE)*, IEEE, Nov. 2020, pp. 1–6. doi: 10.1109/LARS/SBR/WRE51543.2020.9307098.

[14] L. Xie, S. Wang, A. Markham, and N. Trigoni, “Towards Monocular Vision based Obstacle Avoidance through Deep Reinforcement Learning,” Jun. 2017.

[15] H. Song, A. Li, T. Wang, and M. Wang, “Multimodal Deep Reinforcement Learning with Auxiliary Task for Obstacle Avoidance of Indoor Mobile Robot,” *Sensors*, vol. 21, no. 4, p. 1363, Feb. 2021, doi: 10.3390/s21041363.

[16] H. Surmann, C. Jestel, R. Marchel, F. Musberg, H. Elhadj, and M. Ardani, “Deep reinforcement learning for real autonomous mobile robot navigation in indoor environments,” *arXiv preprint arXiv:2005.13857*, 2020, doi: https://doi.org/10.48550/arXiv.2005.13857.

[17] K. Stachowicz, D. Shah, A. Bhorkar, I. Kostrikov, and S. Levine, “FastRLAP: A System for Learning High-Speed Driving via Deep RL and Autonomous Practicing,” Apr. 2023.

[18] A. L. Montealegre-Gracia, “ Aplicaciones forestales de los datos LiDAR-PNOA en ambiente mediterráneo: su filtrado e interpolación y el modelado de parámetros estructurales con apoyo en trabajo de campo,” Universidad de Zaragoza, 2017.

[19] CADDEN, “Lidar Technology,” https://www.cadden.fr/en/lidar-technology/.

[20] Z. Xiong, Y. Zhang, F. Wu, and W. Zeng, “Computational Depth Sensing : Toward high-performance commodity depth cameras,” *IEEE Signal Process Mag*, vol. 34, no. 3, pp. 55–68, May 2017, doi: 10.1109/MSP.2017.2669347.

[21] L. W. J. I. G.-J. A. B. A. Keselman, “Intel (R) Realsense (TM) Stereoscopic Depth Cameras,” *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1267–1276, 2017.

[22] A. Bhandari and R. Raskar, “Signal Processing for Time-of-Flight Imaging Sensors: An introduction to inverse problems in computational 3-D imaging,” *IEEE Signal Process Mag*, vol. 33, no. 5, pp. 45–58, Sep. 2016, doi: 10.1109/MSP.2016.2582218.

[23] A. G. Vladimir, “Point clouds registration and generation from stereo images. Inf. Content Process,” *Information Content and Processing*, vol. 3, pp. 193–199, 2016.

[24] G. A. Kordelas, P. Daras, P. Klavdianos, E. Izquierdo, and Q. Zhang, “Accurate stereo 3D point cloud generation suitable for multi-view stereo reconstruction,” in *2014 IEEE Visual Communications and Image Processing Conference*, IEEE, Dec. 2014, pp. 307–310. doi: 10.1109/VCIP.2014.7051565.

[25] STEREO LABS, “Mapeo Espacial,” https://www.stereolabs.com/docs/spatial-mapping.

[26] F. Berzal, *Redes Neuronales & Deep Learning*, vol. 1. 2018.

[27] M. Bravo, A. Molero, J. Rojas, K. Prieto, and A. Gomez, “Desarrollo de un sensor virtual de flujo usando Redes Neuronales Artificiales para Bombas de Cavidades Progresivas en el campo San Diego de Cabrutica,” *3er congreso suramericano de petróleo y gas 2015*. Nov. 2015. doi: 10.13140/RG.2.2.20329.49764.

[28] J. L. Reyes-Cruz, M. G. Sánchez-Trujillo, and R. Mejía-Ramírez, “Influencia de la formación universitaria en las actitudes emprendedoras,” *Revista CEA*, vol. 5, no. 10, pp. 117–133, Jul. 2019, doi: 10.22430/24223182.1240.

[29] ATRIA INNOVATION, “¿Qué son las redes neuronales y sus funciones?,” https://www.atriainnovation.com/que-son-las-redes-neuronales-y-sus-funciones/.

[30] J. I. Bagnato, “¿Cómo funcionan las Convolutional Neural Networks? Visión por Ordenador,” https://www.aprendemachinelearning.com/como-funcionan-las-convolutional-neural-networks-vision-por-ordenador/.

[31] H. Andrade Carrera, S. Sinche Maita, and P. Hidalgo Lascano, “Modelo para detectar el uso correcto de mascarillas en tiempo real utilizando redes neuronales convolucionales,” *Revista de Investigación en Tecnologías de la Información*, vol. 9, no. 17, pp. 111–120, Jan. 2021, doi: 10.36825/RITI.09.17.011.

[32] J. Torres, *Introducción al aprendizaje por refuerzo profundo: teoría y práctica en Python*. Kindle Direct Publishing, 2021.

[33] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: A Bradford Book, 2018.

[34] “ackermann_vehicle,” https://github.com/hdh7485/ackermann_vehicle.

[35] The Construct, “OpenAI ROS,” https://wiki.ros.org/openai_ros.

[36] SLAMTEC, “RPlidar A1,” https://www.slamtec.ai/home/rplidar_a1/.

[37] “depthimage_to_laserscan,” http://www.ros.org/wiki/depthimage_to_laserscan.