Diseño de una estrategia para la planeación de rutas de navegación autónoma de un robot móvil en entornos interiores usando un algoritmo de aprendizaje automático

Design of a strategy for the planning of autonomous navigation routes of a mobile robot in indoor environments using a machine learning algorithm

Diego León Ramírez Bedoya

Universidad Nacional de Colombia
Facultad de Minas, Departamento de Ciencias de la Computación y de la Decisión
Medellín, Colombia
2020

Diseño de una estrategia para la planeación de rutas de navegación autónoma de un robot móvil en entornos interiores usando un algoritmo de aprendizaje automático

Design of a strategy for the planning of autonomous navigation routes of a mobile robot in indoor environments using a machine learning algorithm

Diego León Ramírez Bedoya

Tesis presentada como requisito parcial para optar al título de:
Maestría en Ingeniería - Ingeniería de Sistemas

Director:
Jovani Alberto Jimenez Builes Ph. D.
Co-Director:
Gustavo Acosta Amaya Ph. D.

Línea de Investigación:
Robótica
Grupo de Investigación:
Grupo de Investigación y Desarrollo en Inteligencia Artificial GIDIA

Universidad Nacional de Colombia
Facultad de Minas, Departamento de Ciencias de la Computación y de la Decisión
Medellín, Colombia
2020

(Dedicatoria)

A mi madre que ha sido el pilar de mi vida, a mi pareja Sandra que con su compañía y consejos me ha ayudado a mantener la calma y encontrar las respuestas en los momentos de incertidumbre y ha mi hijo con el cual he podido compartir buenos momentos en mi vida.

Agradecimientos

Agradezco al Doctor Jovani Alberto Jimenez Builes que gracias a su acompañamiento, asesoría y gestión esta tesis y los artículos generados a partir de ella son posibles. También, agradezco al Doctor Gustavo Alonso Acosta Amaya que durante el transcurso de toda mi vida académica ha sido un referente profesional y moral, el cual fue clave para el logro de esta investigación.

Resumen

El problema de la navegación autónoma de los robots en entornos internos debe superar varias dificultades como la dimensionalidad de los datos, el costo computacional y la posible presencia de objetos móviles. Esta tesis se orienta al diseño de una estrategia de planeación de rutas para la navegación autónoma de robots en entornos interiores con base en el aprendizaje automático. Para lo cual se caracteriza algunas estrategias que reporta la literatura, se especifica el algoritmo de aprendizaje automático DQN, para luego ser implementado en la plataforma robótica Turtlebot del simulador Gazebo. Además, se realizó una serie de experimentos cambiando los parámetros del algoritmo para hacer la validación de la estrategia que muestra como la plataforma robótica por medio de la exploración del ambiente y la posterior explotación de conocimiento hace una planeación de la ruta eficaz. Vídeo del experimento puede ser encontrado en https://youtu.be/5ehdh-BvY7E.
Palabras clave: (Navegación autonoma, robotica, Aprendizaje por refuerzo, inteligencia artificial).

Abstract

The problem of autonomous robot navigation in internal environments must overcome various difficulties such as the dimensionality of the data, the computational cost and the possible presence of mobile objects. This thesis is oriented to the design of a planning strategy of routes for autonomous navigation of robots in interior environments based on automatic learning, for which it characterizes some strategies that the literature reports, The DQN machine learning algorithm is specified, to be implemented on the Turtlebot robotic platform of the Gazebo simulator. In addition, a series of Experiments changing the parameters of the algorithm to validate the strategy that shows how the robotic platform through the exploration of the environment and the subsequent exploitation of knowledge makes effective route planning. Video of Experiment can be found at https://youtu.be/5ehdh-BvY7E.
Keywords: (Navigation, robotics, reinforcement learning, artificial intelligence).

Lista de Figuras

Lista de Tablas

Contenido

1 Introducción

En la actualidad se puede apreciar una fuerte incorporación de la inteligencia artificial y, en particular, el aprendizaje automático en los métodos de la industria y la ingeniería. Por ejemplo, la integración de estos algoritmos en la realización de videojuegos demuestran que el aprendizaje por refuerzo implementado con redes neuronales permite proponer nuevas estrategias de competición, con igual o mejor precisión que un ser humano [1][2].

Recientes investigaciones prueban que la navegación autónoma con algoritmos de aprendizaje por refuerzo profundo con redes neuronales convolucionales (CNN), establecen un gran potencial en las políticas de desplazamiento de los robots [1][2].

Por otra parte la diversidad de entornos en que tienen que operar los robots, hace necesario que las plataformas robóticas realicen actividades necesarias para el ser humano, incluyendo las peligrosas o complejas [3]. Además, los robots domésticos deben enfrentar una amplia variedad de estructuras y obstáculos dinámicos que impide el buen desempeño de los algoritmos tradicionales [4] [5], por lo cual se debe realizar un reconocimiento de rutas del entorno para hacer una navegación eficaz.

Es sabido que la implementación de instrucciones para la navegación de plataformas robóticas con algoritmos tradicionales es una labor especializada, en la cual, el usuario debe realizar una configuración inicial que requiere cierto nivel de destreza. Por ende, resulta práctico y conveniente implementar métodos de aprendizaje automático que permitan al robot aprender de su ambiente [6].

Esta tesis aborda el problema de la implementación de un algoritmo de aprendizaje automático para la planeación de rutas en la navegación autónoma de robots en ambientes interiores. La tesis está dividida en cuatro secciones: en la primera sección, se define el planteamiento del problema, impactos, aportes, hipótesis, objetivos y marco teórico. La segunda sección muestra un estado del arte exhaustivo de las ultimas investigaciones realizadas en el mundo de la navegación autónoma de robots con algoritmos de aprendizaje por refuerzo. La tercera sección se refiere a una estrategia para la selección de un algoritmo por refuerzo con sus características principales. La cuarta sección describe el experimento que se realizó para evaluar el algoritmo. En sexta sección se realiza una evaluación del experimento donde se muestra las diferentes métricas con el análisis de resultados y las gráficas obtenidas. En la séptima sección se expone las conclusiones con las recomendaciones para trabajos futuros.

1.1. Planteamiento del problema

El problema de la navegación autónoma de robots incluye el desplazamiento en ambientes internos con evasión de obstáculos [7] [4] y este paradigma conlleva varias dificultades como la dimensionalidad de los datos, el costo computacional, y la posible presencia de barreras móviles [8][7][9]. Además, es común que las plataformas robóticas interpretan el entorno por medio de su sistema sensorial y, en consecuencia, se deben analizar un sinnúmero de situaciones para un recorrido eficaz. Es de resaltar que descubrir la ruta óptima constituye un problema tipo P-Hard [10] y la literatura propone soluciones con algoritmos clásicos como el Dijkstra, genético y A * [1]. Sin embargo, la dificultad de disponer de la información global del ambiente los hace inviables cuando se maneja altos niveles de incertidumbre. Pues, en el mundo real existe la posibilidad de que las plataformas robóticas no dispongan de un modelo del entorno y, por lo tanto, deben aprender por medio de la exploración. Por otra parte, los robots deben enfrentar situaciones inesperadas no consideradas en su programación inicial, motivo por el cual se plantean algoritmos de machine learning en tiempo real y sin intervención humana [10]. En este orden de ideas se propone abordar el problema de la planeación de rutas para la navegación autónoma de robots en ambiente interiores por medio de métodos del aprendizaje automático.

1.2. Impactos

- Contribuir en el ámbito académico con algoritmos validados de aprendizaje automático para la navegación autónoma de robots.
- Propiciar en el sector productivo colombiano la apropiación de conocimiento en la navegación autónoma de robots para la posible implementación en procesos industriales.
- Identificar futuras líneas de investigación en el área de la robótica y aprendizaje automático.

1.3. Aportes

En el ámbito de la planeación de rutas en la navegación autónoma de robots con algoritmos de aprendizaje automático se resaltan los siguientes aportes de esta tesis:

- Levantamiento de un estado del arte exhaustivo de las recientes investigaciones de navegación autónoma de robots que utilizan el aprendizaje con refuerzo.
- Formulación de una estrategia para la selección de un algoritmo de aprendizaje automático aplicado a la planeación de trayectorias.

- Caracterización de los métodos de aprendizaje por refuerzo y como implementarlos en ambientes simulados.
- Establecimiento de criterios para la implementación de un algoritmo de aprendizaje por refuerzo en un robot.
- Definición de una métrica para evaluar el aprendizaje de una plataforma robótica en etapas tempranas de la exploración.

1.4. Hipótesis

Es posible que la aplicación de una estrategia de planeación de rutas para una plataforma robótica móvil con un algoritmo de aprendizaje automático, facilite la navegación autónoma de un robot en un ambiente interior desconocido.

1.5. Objetivos

1.5.1. Objetivo General

Diseñar una estrategia de planeación de rutas para la navegación autónoma de robots en entornos interiores con base en aprendizaje automático.

1.5.2. Objetivos Específicos

- Caracterizar algunas estrategias que reporta la literatura en el tema de la planeación de rutas para la navegación autónoma de robots móviles.
- Especificar un algoritmo de aprendizaje automático para la planeación de rutas en la navegación autónoma de robots móviles basados en la previa caracterización de estrategias.
- Implementar un algoritmo de aprendizaje con refuerzo para la planeación de rutas en la navegación autónoma de robots móviles.
- Validar la estrategia de la planeación de rutas de navegación en ambientes internos mediante pruebas experimentales.

1.6. Marco teórico

La inteligencia artificial es un conjunto de métodos computacionales bioinspirados que trata de emular el pensamiento humano en áreas como el reconocimiento de patrones, procesamiento de lenguaje natural, resolución de problemas, robótica, aprendizaje autónomo [1][7].

El resultado es una herramienta que aporta soluciones a la ingeniería y a la ciencia para buscar respuestas donde las soluciones convencionales tiene un costo elevado o no es práctico de realizar.
El aprendizaje automático es una área de la inteligencia artificial que diseña algoritmos con la capacidad de aprender por ellos mismos, sin ser explícitamente programados. Tal es el caso de la predicción del comportamiento crediticio de un cliente bancario analizando la conducta de otros usuarios con las mismas características o la clasificación de imágenes para definir detalles como colores, tipos de objetos, reconocimiento facial, por mencionar algunas [10].

1.6.1. Aprendizaje supervisado

Se dice que el aprendizaje es supervisado cuando los datos que se disponen para el entrenamiento incluye los resultados de experimentos o simulaciones previas. Por ejemplo, por medio de una regresión lineal y con el historial de facturación de energía se puede predecir el consumo eléctrico de una ciudad. Algunos de los algoritmos más representativos son la regresión lineal, regresión logística, árboles de decisión, Naïve Bayes, máquinas de soporte vectorial [6][11][12].

1.6.2. Aprendizaje No supervisado

Es cuando el algoritmo se encarga de clasificar la información sin poseer resultados a priori o etiquetas con valores. Se pueden mencionar algunos algoritmos que pertenecen a esta categoría como los de clustering, análisis de componentes principales, descomposición en valores singulares [6][11][12].

1.6.3. Aprendizaje automático por refuerzo

Es cuando un agente aprende por sí mismo a través de prueba y error, con un sistema de penalización y recompensas Figura **1-1**. Los algoritmos más conocidos son $Q - learning$, criterio de optimalidad, acercamiento al valor de la función, método de Montecarlo.

1.6.4. Programación dinámica

En el aprendizaje por refuerzo se puede presentar que el robot tiene un conocimiento total de las acciones y estados del problema, con su función de transición. No obstante, es posible que el agente desconoce el ambiente, pero por medio de la exploración obtiene el modelo, para luego utilizar métodos matemáticos y técnicas de programación dinámica para planear la ruta de navegación. Estos tipos de problemas son conocidos como métodos basados en modelos, y es utilizado para solucionar problemas de optimización [6][11][12].
Para valorar los modelos mencionados en el párrafo anterior existen dos tipos de funciones 1) valor: función $V(s)$, que estima la bondad de estar en un estado (s) ecuación 1-1, 2) función

$Q(s, a)$, que estima la bondad de realizar una acción (a) en un estado(s) ver ecuación 1-2.

$$V^{\pi}(s) = \sum_{k=0}^{\infty} \gamma^{\pi} \rho(s_k, \pi(s_k)) \tag{1-1}$$

$$Q^{\pi}(s, a) = \sum_{k=0}^{\infty} \gamma^{\pi} \rho(s_k, a_k) \tag{1-2}$$

La función (V) y (Q) están relacionadas cuando tiene la misma política (π) de la siguiente manera ver ecuación 1-3.

$$V^{\pi}(s) = Q^{\pi}(s, \pi(s_k)) \tag{1-3}$$

1.6.5. Procesos de decisión de Markov

Se puede representar un sistema de aprendizaje por refuerzo como una máquina de decisión de Markov, el cual es es una máquina de estado finito que resuelve problemas de decisión secuencial [6][11][12], donde cualquier valor futuro depende del actual y el cual se representa con la tupla $< S, A, T, R >$ donde :
S: Es el conjunto de estados.
A: Es el conjunto de acciones a realizar.
T: Es la matriz de probabilidades para pasar de un estado (s) a un estado (s') dado una acción (a), ver ecuación 1-4.

$$T(s, a, s') \tag{1-4}$$

R: Es la recompensa que se da del pasar del estado
(s) al estado (s') dado la acción (a) ver ecuación 1-5

$$R_s = E[R_{t+1} | S_t = s] \tag{1-5}$$

1.6.6. Políticas

Las políticas en el aprendizaje por refuerzo se representan con el símbolo (π) y en un caso determinista representa la acción a realizar basado en el estado en que se encuentre [6][11][12] ver ecuación 1-6.

$$a_k = (s_k) \tag{1-6}$$

Dado la función de transición (f) se cambiará el estado $(s1$)ver ecuación 1-7 :

$$s_1 = f(s_0, a_0) \tag{1-7}$$

y el agente recibe una recompensa dado la probabilidad que de una acción (a) pase a un estado (s) ecuación 1-8 :

$$r_0 = \pi(s, a) \tag{1-8}$$

En el caso estocástico se selecciona una acción (a) estocástica dado un estado (s) ver ecuación 1-9:

$$\pi_0(a|s) = P[a|s : \theta] \tag{1-9}$$

1.6.7. Aprendizaje ON-Line vs OFF-line

El aprendizaje on-line es útil para problemas del mundo real, en especial cuando se aprende directamente del ambiente y no se tiene la información general del entorno. El aprendizaje off-line es cuando la información general está disponible e incluso se posee un data para el entrenamiento de los algoritmos [13].

1.6.8. Aprendizaje por diferencia temporal

El aprendizaje por diferencia temporal TD es un subgrupo del aprendizaje por refuerzo y muestra la manera de predecir un valor que depende de la cantidad futura. El TD lleva a cabo el aprendizaje mediante la implementación de un proceso de muestreo utilizando alguna política conocida y métodos de programación dinámica desde su presente [13].

1.6.9. Algoritmos de aprendizaje automático por refuerzo

Estos algoritmos consisten en la interacción de un agente con el entorno y según las acciones que ejecute el robot recibe una recompensa o una penalización generando así un aprendizaje. La mayoría de estos algoritmos se basan en la ecuación de Bellman y toman como entradas los estado (s) y las acciones (a)[1][14] y las acciones futura son tomadas por políticas (π) que pueden devolver acciones con un mejor valor a corto o largo plazo según en el estado (s) que se encuentre [8][10].

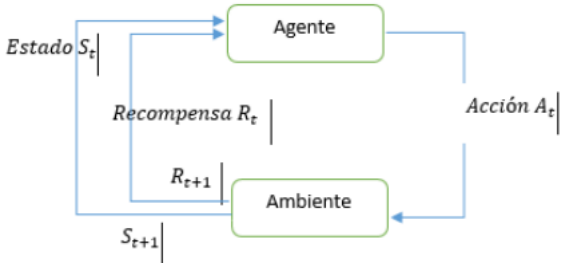

Figura 1-1: Diagrama clásico del aprendizaje automático.

1.6.10. Enfoque clásico de la planificación de rutas para robots

Los métodos clásicos se caracterizan en que se debe disponer de un mapa del entorno para luego buscar el camino más corto por medio de funciones objetivos. Por ejemplo, tenemos los de descomposición por celdas, campos potenciales, mapas de rutas, algoritmo A*.

1.6.11. Métodos basados en descomposición de celdas

Este método consiste en subdividir en espacios más pequeños (celdas) las áreas libres de obstáculos para definir un recorrido desde el punto inicial hasta el destino. La ruta se planea por medio de un grafo en el que las celdas son representadas por nodos y aristas cuando estos son adyacentes [15][6].

1.7. Conclusión del capítulo

Aunque el aprendizaje de máquina se divide en supervisado, no supervisado y por refuerzo, se observa rápidamente que el RL es la mejor opción para solucionar el problema de la planeación de rutas, debido a que presenta la posibilidad de que una plataforma robótica aprenda por sí misma la ruta a seguir en ambientes desconocidos.

2 Análisis de las investigaciones

2.1. Algoritmos clásicos

Los métodos de campos potenciales están basados en el fenómeno físico de las cargas eléctricas, donde se habla de una atracción hacia la meta y una repulsión hacia los obstáculos, es un método meramente reactivo que no contempla el aprendizaje [16]. Pero este algoritmo es propuesto con nuevas estrategias para los problemas de la navegación local y evasión de obstáculos. Por ende, se separa los algoritmos de control con dos mapas de campos potenciales, el primero permite construir la ruta y, el segundo, representa la geometría del obstáculo. El método por división de celdas consiste en dividir el espacio en diferentes nodos para identificar las zonas libres [17]. Los posibles criterios para evaluar esta estrategia es: 1) tener en cuenta el tiempo para descomponer el área en cuadrículas y encontrar la ruta; 2) obtener el número de celdas y relaciones ; 3) la circularidad (redondez) de las casilla para una menor probabilidad de superposición de los límites; 4) la optimización de los puntos de referencia; 5) la longitud de la trayectoria; 6) la forma de la celda que se sugiere sea rectangular o triangular, ya que muestran mejores resultados con los espacios referenciados y los flancos del grafo [18]. Una variación de estos algoritmos es con funciones armónicas que son calculadas sobre una cuadrícula no regular, donde el muestreo está sesgado hacia el tipo, tamaño de las celdas y las regiones más cercanas de la meta [19]. En el método de la ruta probabilística del mapa (roadmaps) el espacio también es dividido en cuadrículas para seleccionar una zona libre, marcada con nodos interconectados entre sí. Además, de construir una ruta que evalúe la probabilidad de que la región asociada esté ocupada. Actualmente, se realiza investigaciones con este tipo de algoritmo con herramientas tecnológica vigentes en la robótica como el simulador Gazebo en la navegación autónoma de robots [20].

2.2. Algoritmos Heurísticos

En esta sección se habla de diferentes algoritmos que utilizan pruebas y ensayos para encontrar la mejor ruta para la navegación de robots, específicamente los relacionados con el aprendizaje automático Figura **2-1**. También incluye las ventajas y desventajas de utilizar sensores como cámaras y dispositivos láser. Se hace un análisis de los últimos descubrimientos enfocados con este paradigma y se realiza una caracterización de varias estrategias utilizadas para la implementación de estos (Tabla 3-1 y 2-2). También se muestran distintas tácticas

Tabla 2-1: Significado de los acrónimos

Acrónimo	Significado
A2C	Advantage Actor-Critic
A3C	Asynchronous advantage actor-critic
APDyna-Q	Asynchronous phased Dyna-Q
AMCL	Adaptive Monte Carlo Localization
ARN	Autonomous robot navigation
DDQN	Double deep Q-networ
DFP	Direct Future Prediction
DQN	Deep Q-Learning Networks
DRL	Deep reinforcement learning
FACL	Fuzzy Actor Critic-Learning
GA3C	GPU implementation of the Asynchronous Advantage Actor-Critic
GNP-TSRL	Genetic network programing-wo stage reinforcement learning
hHRL	Hybrid hierarchical reinforcement learning
IDRL	Imitation deep reinforcement learning
IL	Imitation learning
IRL	Incremental reinforcement learning
LB-WayPtNav	Learning-based waypoint approach to navigation
LFRL	Lifelong federated reinforcement learning
LSTM	Long short-term memory
MPC	Model predictive control
MTDL	Modified Temporal Difference Learning
PPO	Proximal Policy Optimization
Q	Quality
RL	Reinforcement Learning
RMP	Riemannian motion policy
SFRL	Supervised fuzzy reinforcement learning
SLAM	Simultaneous localization and mappin
nsQ	n-step Q-learning
nsBQ	n-step bootstrapped Q-learning
RELM	Reinforcement extreme learning machine

para la planeación de rutas con o sin modelo (incluyendo vehículos aéreos) y se efectúa una introducción al uso del hardware en la nube.

2.2.1. Metodología de búsqueda

Fuentes de información

La investigación se limitó a las siguientes Bases de datos especializadas Science Direct, Springer Journals, IEEE Xplore, Scimago Journal, Scopus, EBSCO, Proquest Central, buscadores como Google Scholar con artículos publicados a partir del 2015. Se limitó la búsqueda a las área temáticas de la ciencia de la computación e ingeniería.
TITLE-ABS-KEY (reinforcement AND learning AND navigation AND robots)

Criterios de inclusión y exclusión

Se incluyen artículos publicados en revistas indexadas, capítulos de libros o libros encontrados en Science Direct, Springer Journals, IEEE Xplore, Scimago Journal, Scopus, EBSCO, Proquest Central, Google Scholar.

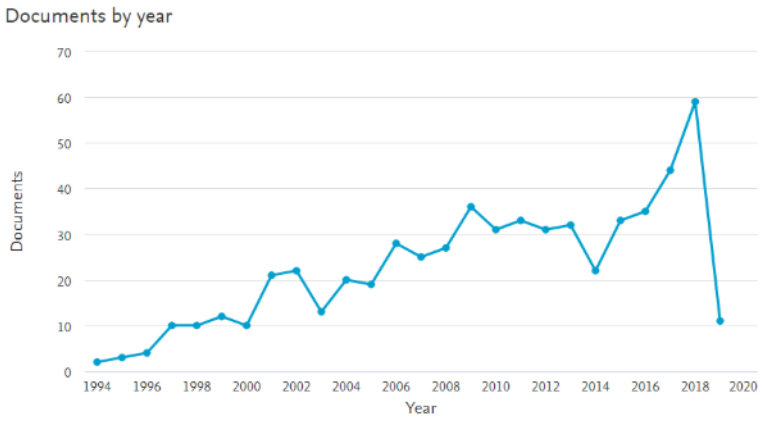

Figura 2-1: Publicaciones realizadas en los ultimo años referente a la navegación autónoma de robots por aprendizaje automático

2.2.2. Aprendizaje por imitación reforzada para la navegación autónoma de robots

Actualmente las plataformas robóticas aprenden tareas específicas por medio de la imitación, lo cual es implementado exitosamente en la resolución de problemas de diferentes dominios [21]. La navegación autónoma de robots también se ve influenciada por los algoritmos basados en el mimetismo, para lo cual se usa mapas de diversas complejidades en conjunto con herramientas tecnológicas como el Middleware ROS (que aporta un conjunto de librerías para la navegación autónoma de robots). En este sentido los robots requieren de un maestro, el cual puede ser un humano, un robot o un software y, por ejemplo, las librerías que maneja el planificador move-base muestran resultados más viables en el entrenamiento de los agentes

que los aportados por un humano en precisión y tiempo [21]. Sin embargo, no sólo es necesario imitar, sino comprender lo que se hace y, para esto, se proponen métodos de aprendizaje por refuerzo inverso con redes neuronales que permiten aprender una ruta por emulación sin definir manualmente la función de recompensa. En este caso se utiliza las personas a modo de experto y, aunque, no genera las soluciones óptimas, se puede mejorar los resultados combinándolos con algoritmos como el A* [22]. No obstante, la falta de coincidencia en la representación, la deficiente escalabilidad y la dificultad de implementar restricciones hace que sea tediosa la implementación de estos algoritmos en el mundo real [23].

2.2.3. El aprendizaje por diferencia temporal

La Diferencia Temporal (TD) es un subgrupo del aprendizaje por refuerzo, en el cual la información se obtiene del ambiente, mediante la implementación de un proceso de muestreo, políticas conocidas y métodos de programación dinámica que predice un valor dependiendo de los resultados conocidos. Además, existen modificaciones para mejorar el rendimiento del algoritmo como 1) definir el inicio de la trayectoria desde el punto de origen, 2) cuantificar el factor de utilidad para cada celda en una cuadrícula, 3) utilizar la ecuación Q-Learning, 4) asignar una función de recompensa para todos los posibles movimientos, 5) agregar el factor de utilidad y la recompensa correspondiente, 6) comparar el factor de utilidad modificado de todos movimientos que son inmediatamente posibles, 7) calcular la diferencia en el factor de utilidad para el estado actual y siguiente, 8) seguir el camino con el estado de mayor valor y repetir los pasos 1-5 hasta que la posición de destino sea alcanzada. La aplicación de las anteriores recomendaciones demuestran dar mejores resultados que otros métodos como el Dijkstra's [24].

2.2.4. Planificación de rutas con cuadrículas e inteligencia artificial sobre servidores

Cuando se tiene el conocimiento global del ambiente y del modelo, es posible dividirlo en celdas para clasificar las posiciones de las metas, los obstáculos y el propio robot. Por ejemplo, se diseña una ruta con redes neuronales utilizando una grilla con casillas habilitadas o bloqueadas. Las primeras, son marcadas con "1" y representa una celda de color blanco. Las segundas, se les asignan un "0" y simboliza una casilla de color negro. El costo del movimiento cardinal es igual a "1" y el diagonal cuesta "2". El valor total del camino corresponde a la suma de todos los movimientos realizados y la meta del algoritmo es encontrar las celdas adyacentes desbloqueadas que permitan definir una trayectoria [25]. Por otra parte, se contempla la posibilidad de que existan varios obstáculos con movimiento o robots en un mismo ambiente, para esto se propone una planificación de ruta basados en Deep Learning, donde las plataformas robóticas estén conectados a un servidor que procesa la información capturada con las peticiones de los usuarios. Asimismo, una cámara obtiene imágenes del

entorno en tiempo real, la cual sirve para clasificar los agentes robóticos, los objetivos y los obstáculos por medio de la señalización con diferentes colores [5]. La principal ventaja de una estrategia con software en un servidor o en la nube, es que se puede construir un algoritmo de aprendizaje automático que aproveche la experiencia de diferentes plataformas robóticas para fortalecer la toma de decisiones. También es posible usar con puntos de referencia (WayPtNav), con submódulos de percepción y planificación [26]. La percepción se utiliza para la observación RGB actual con una CNN que predice la siguiente posición de desplazamiento. La planificación usa el modelo dinámico para definir una trayectoria suave y desplazar al robot al objetivo a través de un controlador con retroalimentación lineal basado en modelos dinámicos que rastrea las trayectorias con señales semánticas para la navegación orientada a objetivos [27]. Por otra parte el aprendizaje por refuerzo se puede extender fácilmente a la navegación multi agente de robots, para solucionar problemas de coordinación, dimensionalidad y cambios en el espacio. Las redes neuronales y las técnicas de suavizado del kernel basadas en políticas de gradientes se aplican para aproximar la estimación de un entorno desconocido para generalizarse a nuevos escenarios [28], además puede que el método esté libre de ajustes de parámetros fuera de línea [29]. Adicionalmente, es posible modelar la política de recorrido como una combinación de objetivos dinámicos que prevenga colisiones acopladas de manera que aprendan conjuntamente [30].

2.2.5. Planificación jerárquica y aprendizaje automático

Un interesante propuesta de este algoritmo es el ADDPG, el cual consiste en una política asíncrona determinista gradiente profunda, que utiliza redes neuronales combinadas con el método del actor-crítico de manera jerárquica. Para lo anterior, se define la función de translación como:

$$v_t = f(x_t, p_t, v_{t-1}) \tag{2-1}$$

Donde (x_t) es la información sin procesar de los sensores, (p_t) es la ubicación del objetivo, (v_{t-1}) es la velocidad final del robot. En este sentido, cuando la plataforma robótica logra su propósito se gratifica con una recompensa positiva y posteriormente se detiene. Por otra parte, se asigna una penalización en caso de colisión y de no darse ninguno de los sucesos anteriores se resta la localización de la meta por la última posición del agente robótico $d_{t-1} - d_t$ multiplicada por un hiper parámetro (C_r) motivando así la búsqueda. En esta estrategia se utilizó el algoritmo de localización en un mapa para la georreferenciación del robot y del objetivo. Además, se usan coordenadas polares para la posición final del agente robótico en vez de la ubicación de la meta. Este método demuestra ser más robusto que un planificador de movimientos basado en mapas de baja dimensión [31]. Otra táctica es utilizar el diseño de un bosquejo global tradicional para el cálculo de rutas óptimas, y uno local profundo con un controlador de velocidad para evaluar los comandos de movimiento, para lo cual se recurre al aprendizaje por imitación que modela el comportamiento apropiado

y una generalización a fin de permitir alcanzar con frecuencia la meta en nuevos modelos [32].

2.2.6. Aprendizaje por refuerzo con lógica fuzzy

La lógica difusa es implementada en conjunto con el aprendizaje por refuerzo para mejorar el rendimiento de estos algoritmos. Por ejemplo, se utiliza el ángulo y la distancia del robot para asignar el valor a conjuntos difusos que permiten a un controlador, tipo sugeno, en conjunto con el algoritmo Fuzzy Actor Critic Learning (FACL), ubicar la meta por medio de coordenadas polares y converger rápidamente en ausencia de obstáculos [33]. También, es usado para la implementación de comportamientos y disminuir el número de acciones necesarias que obtienen la mejor recompensa del ambiente y evitar caer en mínimos locales [34]. De hecho, se puede combinar el paradigma de optimización evolutiva con un algoritmo de colonia de hormigas continuo multiobjetivo que mejore su eficiencia de aprendizaje. Asimismo, los datos de entrenamiento se recopilan aplicando el control Fuzzy durante cada prueba de mando y, además, las redes neuronales sustituta por refuerzo difuso (RNFS) se encargan de evaluar el rendimiento del algoritmo [35]. Cabe considerar que existen datos inconsistentes, ruidosos, erróneos y de difícil recopilación que se pueden usar para inicializar el valor del entrenamiento de cada acción candidata en las reglas difusas. Por ende, se diseñó el algoritmo de aprendizaje supervisado de Fuzzy Sarsa (SFSL), el cual consiste en un controlador Takagi – Sugeno de orden cero. En este sentido, cada regla es considerada en el módulo principal del controlador del robot y, además, el aprendizaje difuso ajusta los parámetros de conclusión del controlador fuzzy para disminuir el tiempo de aprendizaje, el número de fracasos y mejorar la calidad del movimiento del robot en los entornos de prueba [36].

2.2.7. SLAM con aprendizaje por refuerzo

Es deseable agregar la posibilidad a las plataformas robóticas de aprender y negociar con ambientes dinámicos por medio del SLAM. Para esto, el robot debe construir una ruta con algoritmos de aprendizaje por refuerzo y, luego, modelar el entorno que con la experiencia adquirida, lo cual permite definir las acciones a seguir [37]. En algunos casos la estructura no se puede capturar con facilidad y es posible evaluar la actividad realizada desde un estado con respecto a la posición estimada por el SLAM. Así, se genera las trayectorias a prueba de fallos en las que el margen de error sea aceptable en comparación de sus verdaderos valores [38]. Además, En la literatura científica acerca del aprendizaje por refuerzo podemos hablar del aprendizaje basado en valores o en políticas. En el primero se determina el mayor valor que se genera de una acción dado un estado y en el segundo el valor se optimiza directamente. Debido a que el método tiene sus deficiencias se combina el algoritmo asíncrono gradiente de política determinista profunda (ADDPG) con un algoritmo actor-crítico. Así, el actor ajusta

los parámetros de la política mediante un ascenso del gradiente, mientras que el crítico estima la función de valor de la acción utilizando un algoritmo de evaluación como el aprendizaje de diferencia temporal (TD) [39].

2.2.8. Aprendizaje de refuerzo incremental con barrido priorizado para entornos dinámicos

Este método propone un aprendizaje por refuerzo incremental que detecta las modificaciones del entorno y que da prioridad al rumbo definido. En este caso, el ambiente se actualiza de forma iterativa y, por ende, sus funciones de valor, acción y estado usan nuevas recompensas por medio de la programación dinámica. IRL fusiona la nueva información con la ya existente, disminuyendo el conflicto entre ellos [40]. Adicionalmente, es posible integrar los pasos requeridos en el RL a los algoritmos genéticos modificados mostrando mayor rendimiento a los procedimientos tradicionales [41]. Cabe aclarar que en estos métodos se imita la selección natural de la naturaleza en la mutación y el cruce de las especies, siempre buscando los genes más fuertes para producir un gen con característica de los padres mejoradas [42]. Es un algoritmo de optimización que aplicado a la navegación autónoma de robots busca un conjunto de posibles soluciones para luego combinarlas entre sí y, en teoría, encontrar una nueva ruta de navegación mejor que las anteriores [43].

2.2.9. Generalización del aprendizaje con refuerzo

En [44] se sugiere generar sobreajuste en las plataformas robóticas aunque se entrene en diferentes ambientes, por lo tanto proponen combinar el aprendizaje con refuerzo con técnicas de aprendizaje supervisado llamado Regularización De Invarianza.

2.2.10. Navegación neuronal autónoma con política de movimiento riemanniana

La navegación neuronal autónoma con política de movimiento Riemanniana introduce una técnica basada en imágenes que aprovecha la estructura de políticas para un aprendizaje profundo del manejo vehicular, la cual predice los puntos de control RMP del vehículo, desde los cuales los comandos se pueden calcular analíticamente [45].

2.3. Aplicaciones

2.3.1. Planificación de movimientos

Al momento de diseñar una estrategia para la navegación autónoma de robots se tiene en cuenta la planeación de movimientos para explorar el ambiente o recorrer una ruta predefi-

Tabla 2-2: Características de los algoritmos usados.

Autores	RL con mapas	Simulacion	Sistema real
[46]	Y	Y	N
[21]	N	Y	N
[24]	N	Y	N
[26]	N	Y	Y
[27]	Y	Y	Y
[29]	N	Y	N
[31]	N	Y	N
[32]	Y	Y	N
[33]	N	Y	Y
[34]	N	Y	N
[36]	N	N	Y
[40]	Y	Y	Y
[41]	N	Y	N
[43]	N	Y	N
[45]	N	Y	Y
[47]	Y	Y	Y
[48]	N	Y	N
[49]	N	Y	Y
[50]	N	Y	Y
[51]	Y	Y	N
[52]	N	Y	N
[53]	N	Y	Y
[54]	N	Y	N
[55]	N	Y	N

nida. Este reto puede ser descrito como un problema de Markov parcialmente observado y, por lo tanto, es aplicable un algoritmo basado en Q-learning. Por ende, es factible utilizar la tabla Q para predecir valores que determinen la siguiente acción. Los parámetros para modelar el entorno se capturan por medio de una cámara o láser y, así, obtener lo posición relativa que controle la velocidad lineal y angular de la plataforma robótica. Los experimentos con esta táctica demuestran que con una nueva barrera el agente va hacia el objetivo sin mayor contratiempo a diferencia del método tradicional move-base [56]. Otra maniobra efectiva, cuando la información general de la distribución del entorno está disponible, es usar una red neuronal Pos-Net, el cual es un perceptrón multicapa (MLP) que recibe un vector (p) con la ubicación actual del robot, la muestra de tiempo (t), la matriz de valores (q)

permitiendo el uso de una función de adaptación con las coordenadas cartesianas de la meta y obtener un arreglo con salida de 3 valores (x, y, t). La estrategia mencionada demostró con diferentes velocidades una buena relación de convergencia para navegar en entornos con múltiples obstáculos (estáticos o dinámicos) [57]. Otro reto al que se debe enfrentar la planificación de movimientos con aprendizaje por refuerzo son los fallos del sistema sensorial. Cuando un sensor falla se requiere modificar la política de cambios de posición para percibir correctamente el entorno. Debido a que la navegación en ocasiones es realizada sin el modelo es recomendable la función de recompensa:

$$r_t = rf_t \times \sum\sum \lambda(x, y) \tag{2-2}$$

Donde (rf_t) premia la combinación de velocidades que permite los avances del robot y es una matriz booleana usada para ofrecer recompensas con una métrica de cobertura de $\lambda(x, y) \in 0, 1$ que premia comportamientos que abarquen gran cobertura [58]. También se consigue un algoritmo de muestreo estable y eficaz para la capacitación de políticas de navegación de robots con métodos de aprendizaje por refuerzo basado en valores y,además , un grafo computacional generalizado con procedimientos basados en funciones de valor (con o sin modelos) que tiene en cuenta la eficiencia, estabilidad y rendimiento final como métricas para evaluar la propuesta. Para lo anterior, su usa un modelo que aprende de imágenes monoculares; es decir, se observa la representación actual y selecciona una acción cada 0.25 segundos, si se genera un choque se hace una copia de seguridad y comienza desde el último episodio permitiendo que la plataforma aprenda a desplazarse a una velocidad fija a través del entorno [14]. En un contexto más general se propone DQN como un sistema de navegación autónomo para robots móviles que integra una navegación local basada en el aprendizaje por refuerzo profundo y una navegación global basada en mapas topológicos. El sistema busca dar respuestas a ambientes saturados de obstáculos dinámicos como puede ser el tránsito continuo de peatones [47]. Por otra parte, la técnica DRL adaptada del proceso que propone el método GA3C tiene la ventaja asíncrona del algoritmo actor-crítico sobre una GPU, por lo tanto utiliza muchos predictores con información del ambiente y envía los datos a una red neuronal convolucional para calcular los gradientes de la rutina de optimización, así pues se considera la predicción de profundidad y la estimación de posición [59].

2.3.2. Planeación de ruta para vehículos aéreos

Existen experimentos con vehículos aéreos por la comunidad científica que buscan la autonomía en el vuelo, por ejemplo el aprendizaje por refuerzo geométrico para la planificación de rutas, que explota una matriz de recompensa específica, donde se seleccionan los puntos candidatos de la región en la ruta geométrica [63]. Por otra parte, se evalúa la combinación de el aprendizaje por refuerzo y el modelo de control predictivo (MPC), para realizar una implementación eficiente que proporciona la habilidad para la evasión de obstáculos. En este caso el algoritmo se usa para guiar en entornos complejos como corredores sin salida, en los que se requiere maniobras de reversa [48].

Tabla 2-3: Enfoque de los artículos

Autores	Enfoque	Año
[21]	IL+IR + IR-IL	2018
[24]	MTDL	2018
[26]	LFRL+ A2C	2019
[27]	LB-WayPtNav	2019
[29]	PPO	2018
[30]	IDRL	2019
[32]	Dijkstra's+ AMCL	2019
[33]	FACL	2015
[34]	LRI+ LRP	2017
[36]	SFSL	2016
[40]	IRL	2019
[41]	GNP-TSRL	2019
[43]	AG	2015
[45]	RMP	2019
[47]	DDQN	2017
[48]	RL +MPC	2017
[49]	A3C ICM-A3C	2019
[50]	A2C+nsBQ+ nsBQ	2017
[51]	IL+IR	2019
[52]	DQN	2018
[53]	DRL+LSTM	2019
[54]	DFP	2019
[55]	hHRL	2019
[60]	APDyna-Q	2018
[61]	RELM	2016
[62]	APDyna-Q	2018

2.3.3. Laser vs Visión

Los sensores basados en la visión son susceptibles a los fenómenos ópticos como los cambios de iluminación, contraste, enfoque entre otros. Por lo tanto, es factible para un robot con un planificador de navegación DRL utilizar un láser disperso para el adquisición de información, pues en este caso es menos susceptible al ruido ambiental, de modo que se define con más precisión la trayectoria y los órdenes de movimiento. Debido a lo anterior, es posible migrar de la simulación a la realidad sin ajustes fuertes en el modelo de entrenamiento o la imagen de entrada. Además, un agente basado en el aprendizaje automático no requiere una

representación rigurosa en colores y texturas, por ende no es necesario cámaras y el proceso se puede implementar de una manera simple y a bajo costo computacional comparado con la búsqueda de la ruta basada en SLAM [49].

Tabla 2-4: Tipos de obstáculos y sensores utilizados en publicaciones recientes

Autores	Sensores	Obstáculos Dinámicos
[46]	NA	N
[21]	Laser	N
[24]	NA	N
[26]	Imaging, Camera	Y
[27]	Camera	N
[29]	Laser	Y
[30]	N/A	N
[32]	Camera	Y
[33]	Sonar	N
[34]	Sonar	N
[36]	Camera	N
[40]	Sonar	Y
[41]	Infrarrojo	N
[43]	ligth	Y
[45]	Camera Laser	Y
[47]	Camera	Y
[48]	Camera/Lidar	N
[49]	Laser	N
[50]	360-degree camera, Laser	Y
[51]	Camera	Y
[52]	Laser	Y
[53]	Laser,ultrasonic,Visual	N
[54]	RGB frames	N
[55]	NA	N

2.4. Discusión del capítulo

El aprendizaje automático se incorpora a la navegación autónoma del robot a través de diversas estrategias, pero la gran mayoría propone entrenar los algoritmos en entornos virtuales que tienen la misma configuración del entorno real, ya sea para algoritmos con o sin modelo o con las características fuera de línea o en línea se muestran en la Tabla 2-2. Esto se debe

a que la velocidad de procesamiento en una simulación lleva menos tiempo que en el mundo real. La tendencia de investigación en la planificación de rutas en ARN con algoritmos de aprendizaje automático en la literatura se refiere a diferentes técnicas supervisadas y no supervisadas, rápidamente señala el aprendizaje por refuerzo y el aprendizaje de imitación como tendencia en la investigación global al aprendizaje automático aplicado a ARN. Los algoritmos más comunes en la planificación de rutas en ARN fueron Q-learning, actor-crítico y diferencia temporal (Tabla 2-3). Aunque varios autores utilizaron los formatos originales de estos algoritmos, la literatura informa que estos algoritmos por sí solos no son los más eficientes en el mundo real debido a la gran cantidad de datos que deben procesarse y al tiempo que lleva converger. Debido a esto, el uso de esos algoritmos se denota con otros paradigmas utilizados por la inteligencia artificial o los métodos de optimización matemática. Las arquitecturas más utilizadas para la planificación de rutas en ARN fueron los robots diferenciales utilizados con una amplia variedad de sensores como sonares, lásers, cámaras web o sensores RGB-D (Tabla 2-4). Por un lado, existe una tendencia a utilizar lenguajes de programación. como Python y MATLAB. Por otro lado, se ejecutaron experimentos y simulaciones con Open AIGym y Gazebo para analizar los algoritmos de aprendizaje automático.

2.5. Conclusión del capítulo

En este capítulo se caracteriza las últimas investigaciones sobre el aprendizaje por refuerzo aplicado al ARN y se puede observar cómo se incrementa el interés de este tema en el mundo. Lo anterior es debido a que los investigadores encuentran en la inteligencia artificial un nuevo campo de exploración para solucionar el problema de la navegación autónoma de robots.
Por otra parte, se observa que en la mayoría de las publicaciones simulan antes de implementar los algoritmos en una plataforma real y aunque se reconoce las limitaciones por la falta de variables como lo es el ruido ambiental o el error de los sensores se logran buenos resultados. En cuanto los algoritmos y sensores usados existen una gran diversidad de opciones, pero queda claro que el uso de redes neuronales para interpretar los datos de gran dimensionalidad obtenida por los sensores es lo común en el estado del arte.

3 Especificación de un algoritmo

Es de aclarar que en la actualidad existe una gran diversidad de métodos que fueron diseñados para diferentes objetivos y la selección de un RL depende de las necesidades específicas de cada proyecto. En este sentido se usa el simulador OpenAI Gym que facilita las experimentaciones de RL antes de ser llevados a un robot Figura **3-1** y para explicar algunos conceptos necesarios para la selección de un algoritmo.

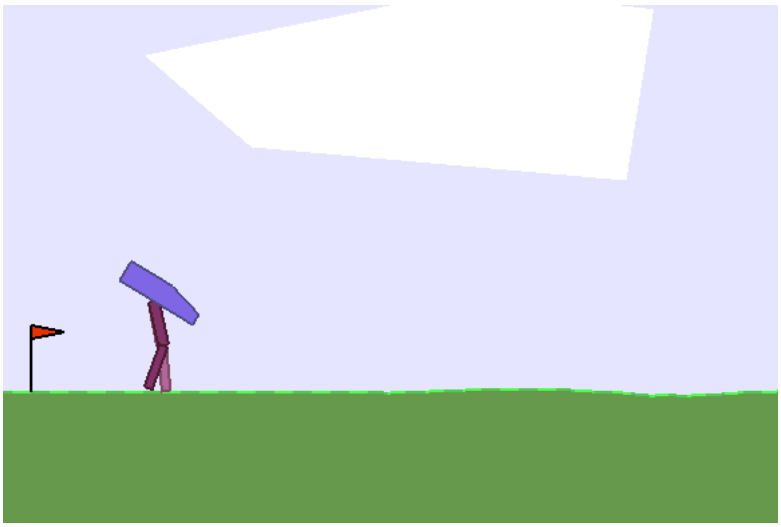

Figura 3-1: Bipedal, Robot con dos piernas que debe aprender a mantener el equilibrio y a caminar por si solo en el simulador OpenAI Gym.

3.1. Características de los algoritmos heurísticos para la navegación autónoma de robots

Los algoritmos heurísticos no obtienen resultados directos, sino que por medio de ensayos se obtiene la solución. En este sentido se debe tener en cuenta si son episódicos, su política, uso de un modelo y sus métodos de búsqueda.

3.1.1. Algoritmo episódico o continuo

Un algoritmo episódico tiene definido claramente el momento de finalización del experimento, por ejemplo, el simulador OpenAI Gym dispone del juego Mountain Car, que consiste en un vagón que se encuentra en un valle oscilando hasta alcanzar una bandera ubicada en la cima de una montaña Figura **3-2**. En el inicio de este reto se empieza con un gran número de oscilaciones para adquirir velocidad y subir la pendiente. En otras palabras, la oscilación está relacionada con la velocidad y la posición, lo que facilita llegar al objetivo para finalizar el episodio Figura **3-3**. Por otra parte, existen los procesos continuos donde no se tiene

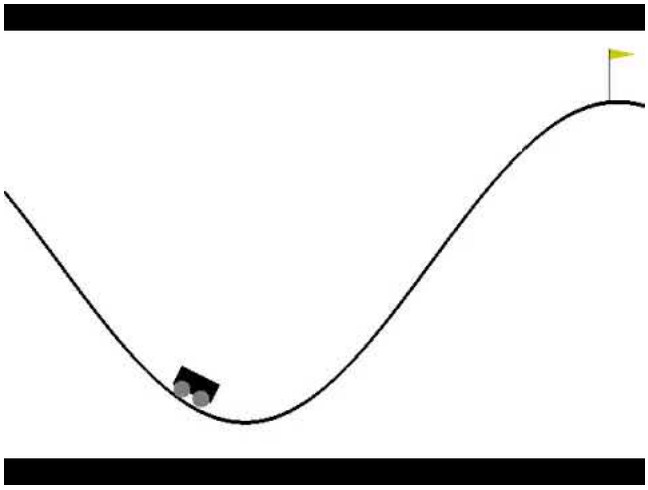

Figura 3-2: Simulación MountCar de OpenAIGym

claro cuándo termina el proceso. Por ejemplo, un sistema que se encarga de clasificar correos electrónicos y se desconoce el tiempo que el aplicativo estará operativo.

3.1.2. Basado en modelo

La existencia del modelo es un concepto importante que puede usar algoritmos como I2a, World Models, MBMF que buscan usar datos existentes por medio de la política (π) las acciones (a) y los estados (s) y aprender de estos sin una exploración del entorno por parte del robot.

3.1.3. Libre de modelo

Libre de modelo es cuando se carece de información previa y se debe obtener los datos del entorno por medio de los sensores del robot.

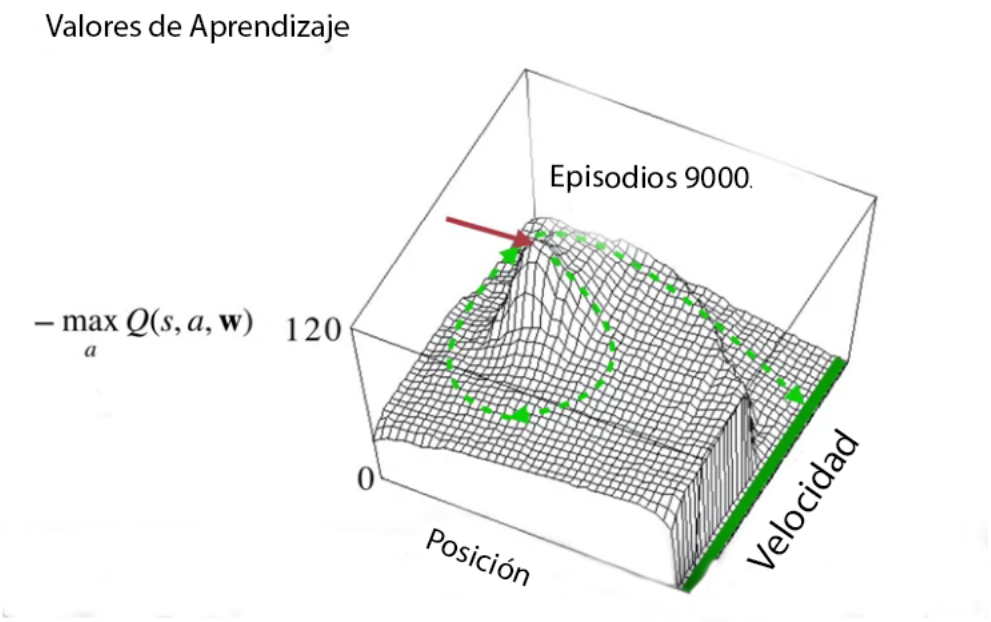

Figura 3-3: Relación entre la velocidad, la posición y el número de oscilaciones de Mountaincar de OpenAI Gym.

3.1.4. Políticas, Valores y Gradientes

Iteración de políticas

Algunos algoritmos como SARSA, conceptos de programación dinámica y los métodos de aproximación sucesivos se pueden clasificar como iteración de políticas, puesto que encuentran y actualizan la función de valor para hallar una política estabilizadora [64].

Iteración de valores

A partir de una suposición inicial la función de valor se modifica directamente al combinar las fases de evaluación y mejora en una sola actualización. Por ejemplo, algoritmos como $Q-learning$, $R-learning$ cumplen con este criterio. La iteración de políticas y la de valores son métodos críticos y pueden considerarse casos especiales de políticas generalizadas [64].

Gradiente de políticas

Los métodos de Gradiente de políticas -también conocidos como de solo actor- calcula el gradiente del costo funcional con respecto a los parámetros desconocidos en la aproximación de la política [64].

ON-Policy

Es cuando la recopilación de los datos y el cálculo de la política óptima van juntas en la misma estimación [64].

OFF-Policy

Es cuando la recopilación de los datos y el cálculo de la política óptima se pueden hacer con estimaciones diferentes [64].

3.1.5. Estrategia de selección de Acción

Estrategia de selección de Acción ϵ-greedy

Es cuando la política selecciona una acción (a) más que las demás y las acciones restantes son utilizadas en un porcentaje igual [65].

Estrategia de selección de Acción Boltzmann o soft-max

Cuando el algoritmo se encuentra explorando, la probabilidad de seleccionar una acción (a) es ponderada por su valor en relación con los otros valor-acción [66] utilizando la siguiente ecuación 3-1.

$$P(a) = \frac{e^{Q(a)/\tau}}{\sum_{b=1}^{n} e^{Q(b)/\tau}} \tag{3-1}$$

Métodos tabulares

Es el método más básico de reinforcement learning, consiste en almacenar los valores de un estado (s) en una tabla, con la limitante que sólo se evalúa los estados visitados. Incluso, si el entorno es grande se debe crear un arreglo con el tamaño del lugar y esto puede ser demandante para la memoria del computador, además este procedimiento no es un bueno para la generalización [65].

Métodos Lineales

Este método funciona cuando el entorno es un modelo continuo y es demasiado grande para que sea práctico almacenar la información en una tabla. Así mismo, ese procedimiento tiene la ventaja que puede generalizar y, por lo tanto, no es necesario visitar todos los estados. Generalmente es usado si el espacio es posible dividirlo en mosaicos y con la desventaja de que se requiere un conocimiento a priori del ambiente por parte el usuario [65].

3.1.6. Redes neuronales

Cabe anotar que debido a la dimensión de los datos no es factible realizar algoritmos de aprendizaje con refuerzo utilizando métodos tradicionales, por la tanto, se hace indispensable la incorporación de redes neuronales que puedan tomar decisiones respecto a los patrones obtenidos por los sensores del robot. Además, al igual que el método lineal, se puede generalizar valores a estados que no han sido visitados y que tienen la capacidad de aprender por sí mismas [67] ecuación 3-2.

$$net_i^{(l)} = \sum_{j=1}^{sl-1} W_{ij}^{(l)} h_j^{(l-1)} + b_i^{(l)} \tag{3-2}$$

Donde (sl) es el numero de neuronas, $W_{ij}^{(l)}$ es la coneccion de los pesos , $b_i^{(l)}$ es el umbral, y $net_i^{(l)}$ ies la entrada de la i-th neurona en la capa 1

Red neuronal profunda o DNN

Es una red neuronal artificial con varias capas ocultas entre la capa de entrada y de salida. Al igual que las redes neuronales artificiales poco profundas, las DNN pueden modelar relaciones no lineales complejas.

Redes neuronales recurrentes

No tienen una estructura de capas definida, sino que permiten conexiones arbitrarias entre neuronas, incluso pueden crear ciclos, con esto se consigue temporalidad permitiendo que la red tenga memoria. Las redes neuronales recurrentes son muy potentes para el análisis de secuencias, texto, sonido o vídeos. Una de las más conocidas es las de tipo LSTM.

Redes neuronales convolucionales CNN

Son un tipo de red neuronal artificial con aprendizaje supervisado que identifica distintas características en las entradas y que reconocen objetos. Para ello, la CNN poseen capas ocultas especializadas con una jerarquía, que permiten detectar líneas o curvas y que se especializan hasta llegar a las más profundas para identificar formas complejas. Estas redes realizan convoluciones, lo cual consiste en tomar grupos de píxeles cercanos de la imagen de entrada e ir operando matemáticamente mediante el producto escalar contra un kernel y, así, generar una matriz de salida.

3.1.7. Arrepentimiento

El arrepentimiento es una métrica que permite evaluar el rendimiento de algoritmos que solucionan problemas sencillos, para lo cual se requiere conocer la política óptima en cada

estado a través de la pérdida acumulada durante el proceso de aprendizaje, pero la navegación autónoma de robots presenta un reto más complejo donde es mejor utilizar el valor acumulado de sus recompensas o el promedio.

$$R_t = T\rho - \sum_{t=0}^{T-1} r_t \tag{3-3}$$

La ecuación 3-3 permite evaluar el arrepentimiento donde (T) es el número de pasos de tiempo ,(r_t) es el vector de recompensa recibido en el tiempo, (ρ) es el vector de recompensa promedio que sigue la política óptima [68].

3.1.8. Algoritmos de aprendizaje por refuerzo

Algoritmo Q-Learning

Este algoritmo cuenta con la ecuación 3-4 y puede ser probado con arreglos bidimensionales donde los espacios vacíos pueden ser representados con un valor diferente a los ocupados (1,0).

$$Q(S_t, At) \leftarrow Q(S_t, At) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}a, a) - Q(S_t, A_t)] \tag{3-4}$$

Estos valores pueden ser asignados manualmente o por medio de una función matemática que los asigne al azar y, así, evitar sesgar o sobre ajustar el experimento. Las simulaciones hacen uso de imágenes segmentadas con un cuadricula y cada celda puede ser representada en una matriz con uno, cero o otro valor dependiendo si se quiere manejar recompensas. Es de resaltar que este algoritmo es OFF-Line, por lo tanto, se deben de tener los datos a priori para el debido funcionamiento del algoritmo. Además, posee una función de acción valor Q.

Algorithm 1 Q-learning: Learn function $Q : \mathcal{X} \times \mathcal{A} \to R$

Require:

Acciones $\mathcal{A} = \{1, \ldots, n_a\}$, $A : \mathcal{X} \Rightarrow \mathcal{A}$
Función de recompensa $R : \mathcal{X} \times \mathcal{A} \to R$
Caja-negra (probabilistico) función de transición $T : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$
Rata de aprendizaje $\alpha \in [0, 1]$, typically $\alpha = 0{,}1$
factor de descuento $\gamma \in [0, 1]$
procedure QLearning($\mathcal{X}$, A, R, T, α, γ)
    Inicializa $Q : \mathcal{X} \times \mathcal{A} \to R$ arbitrariamente
    while Q si no converge **do**
        Iniciar estado $s \in \mathcal{X}$
        while s no es terminal **do**
            Calcular π según Q y la estrategia de exploración (e.g. $\pi(x) \leftarrow_a Q(x, a)$)
            $a \leftarrow \pi(s)$
            $r \leftarrow R(s, a)$ ▷ recibe la recompensa
            $s' \leftarrow T(s, a)$ ▷ recibe el nuevo estado
            $Q(s', a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot (r + \gamma \cdot \max_{a'} Q(s', a'))$
            $s \leftarrow s'$
        end while
    end while
return Q
end procedure

Algoritmo Sarsa

Algorithm 2 SARSA: Función de aprendizaje $Q : \mathcal{X} \times \mathcal{A} \to R$

Require:

Estados $\mathcal{X} = \{1, \ldots, n_x\}$
Acciones $\mathcal{A} = \{1, \ldots, n_a\}$, $A : \mathcal{X} \Rightarrow \mathcal{A}$
Función de recompensa $R : \mathcal{X} \times \mathcal{A} \to R$
Caja-negra (probabilística) función de transición $T : \mathcal{X} \times \mathcal{A} \to \mathcal{X}$
Rata de aprendizaje $\alpha \in [0, 1]$, típicamente $\alpha = 0{,}1$
Factor de Descuento $\gamma \in [0, 1]$
$\lambda \in [0, 1]$: Trade-off between TD and MC
procedure SARSA($\mathcal{X}$, A, R, T, α, γ, λ)
  Inicializa $Q : \mathcal{X} \times \mathcal{A} \to R$ arbitrarily
  while Q is not converged **do**
    Select $(s, a) \in \mathcal{X} \times \mathcal{A}$ Arbitrariamente
    while s is not terminal **do**
      $r \leftarrow R(s, a)$ ▷ Recibe la recompensa
      $s' \leftarrow T(s, a)$ ▷ Recibe el nuevo estado
      Calcula π basado en Q (e.g. epsilon-greedy)
      $a' \leftarrow \pi(s')$
      $Q(s, a) \leftarrow (1 - \alpha) \cdot Q(s, a) + \alpha \cdot (r + \gamma Q(s', a'))$
      $s \leftarrow s'$
      $a \leftarrow a'$
    end while
  end while
return Q
end procedure

Sarsa vs Q-learning

Un algoritmo como SARSA es preferible en situaciones en que es importante el desempeño durante el aprendizaje o la generación de experiencia. Por ejemplo, se desea proteger un robot de tomar malas decisiones en la etapa de exploración. Q-learning es mejor cuando no es trascendental el rendimiento del agente en el proceso de capacitación (simulaciones).

Algoritmo Deep Q-network DQN

El algoritmo DQN es construido con principios parecidos al $Q-learning$ pero sin la limitante de utilizar una tabla para el almacenamiento de datos, puesto que posee una red neuronal que le permite generalizar utilizando la ecuación $r_j + \gamma max_{a'} Q(\phi_{j+1}, a'; \theta)$ y con capas que

construyen representaciones más abstractas de los datos y, que en algunos casos, aprenden conceptos directamente de los sensores sin algún tipo de procesamiento.

Algorithm 3 DQN

Inicializar ambiente env $(robot, goal, obstacles)$
Inicializar datos de almacenamiento en (D)
Inicializar DQN primaria con $Q_{primary}$ *con random pesos* θ
Inicializar DQN primaria con Q_{target} *con random pesos* θ
for *episode* $\leftarrow t$ to EP_{MAX} **do**
  $s_t = reset\ env$
  while *paso actual* $<$ *max episodios pasos* **do**
    if $random\_floating\_value < \epsilon$ **then**
      $selecciona\_random\_accion\ (a_t)$
    else
      $a_t = argmax_a Q_{primary}(s_t, a'; \theta)$
    end if
    $s_{t+1} = move\ siguiente\ estado\ usando$
    el actualmente almacenado(a_t) *experiencia* s_t, a_t, r_t, s_{t+1} *in* D
    if $paso\_Actual < paso\_predefinido\_entrenamiento$ **then**
      $selecciona\_random\_accion\ (a_t)$
      $Entrena\ la\ red\ con\ mini-batches\ de\ D$
      $$\text{y} = \begin{cases} r_i \\ r_i + \gamma\ max_{a\prime} Q_{target} s_{i+1}, , a\prime|\theta_i;) \end{cases}$$
      $Calcula\ la\ perdida\ con\ (y^i - Q_{primary}(s_i, a_i|\theta)\bar{\theta})^2$
      $modificar\ pesos\ \theta\ de\ DQN\ primaria$
      $cada\ \tau\ pasos\ cambiar\ pesos\ \theta\ a\ \theta'$
    end if
  end while
end for

Algoritmo Deep Deterministic Policy Gradient (DDPG)

Utiliza el esquema actor crítico para espacios y acciones continuas, Este algoritmo asume que las políticas son deterministas y usa un buffer de reproducción para ordenar la información obtenida [69].

Algoritmo Twin Delayed DDPG (TD3)

TD3 es parecido al DDPG en el sentido que es para eventos continuos, almacena la información en un buffer de reproducción para para mejorar la eficiencia. Pero este algoritmo utiliza

dos críticos, dos actores y el Q valor-state. Además, para mejorar la predicción agrega ruido al buffer de reproducción [70].

Algoritmo Asynchronous Advantage Actor-Critic A3C

Este algoritmo proviene del clásico modelo actor critico y en donde ambos aprenden al mismo tiempo y posee varios agentes que buscan la solución y comparten el aprendizaje entre ellos. El actor cambia continuamente la política en la búsqueda de superar las expectativas de los críticos que a su vez actualizan la función de valor que evalúa a los actores.
[71][72].

Algorithm 4 Asynchronous advantage actor-critic (A3C)

```
Inicializar las redes del actor y el critico con pesos aleatorios
for cada episodio ∈ [1, n] do
  descargar los pesos principales de cada Actor Critico
  for cada − Actor − Critico ∈[1,n] do
      inicializar el S de inicial manera aleatoria
      for cada t ∈ [1, k] do
          seleccione acción a_t de la red actor
          ejecute la acción de la red actor
          observe la recompensa r_t de la red critica y el siguiente estado
          Modifique los parámetros de la red del actor
      end for
      Modifique los parámetros de la red del critico
  end for
  Cargue los nuevos pesos a las redes del actor -critico
end for
```

3.2. Exploración o explotación

En un algoritmo de aprendizaje por refuerzo existen dos fases: la primera, es de exploración que se encarga de sondear diferentes rutas para definir donde se obtienen mejores recompensas con diferentes acciones (a) dado un estado (s). La segunda, es la explotación que consiste en aprovechar lo aprendido en la exploración para buscar la ruta óptima. En otras palabras, la explotación toma la mejor decisión con la información existente y la exploración obtiene los valores de las rutas.

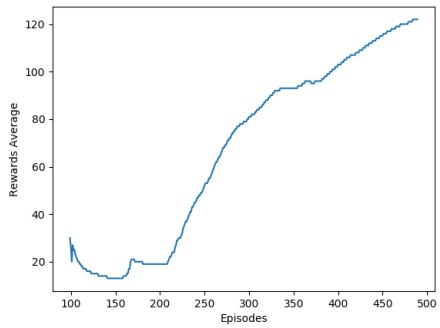

Figura 3-4: Recompensas algoritmo DQN con simulación CartPole-v1.

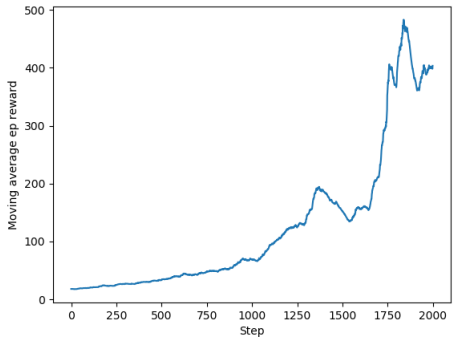

Figura 3-5: Recompensas algoritmo con simulación CartPole-v1.

Interpretación de la exploración o explotación en las gráficas

Profundizando un poco más en la exploración y la explotación se aprecia que en el algoritmo DQN la recompensa promedio es mayor al principio Figura **3-4** y comparándolo con el algoritmo A3C Figura **3-5** se puede llegar a la errónea conclusión de que el algoritmo converge más rápidamente y por lo tanto es mejor. Pero realmente, si se hace un análisis de las etapas del aprendizaje por refuerzo, se aprecia que el algoritmo A3C posee una etapa de exploración mayor que DQN y, por ende, la exploración puede generar recompensas con valores bajos o negativos como es el caso del algoritmo TD3 Figura **3-6** sin que esto sea un indicador de que al final del proceso se obtienen peores o mejores resultados.

3.2.1. Estrategia para la Selección del algoritmo

Para seleccionar un algoritmo para la implementación en plataformas robóticas se puede utilizar un método tabular Figura **3-7**, el cual es factible si el reconocimiento del entorno requiere pocos datos. Esta estrategia tiene un costo computacional alto, pues maneja toda la información en arreglos bidimensionales lo que no es recomendable si los datos provienen

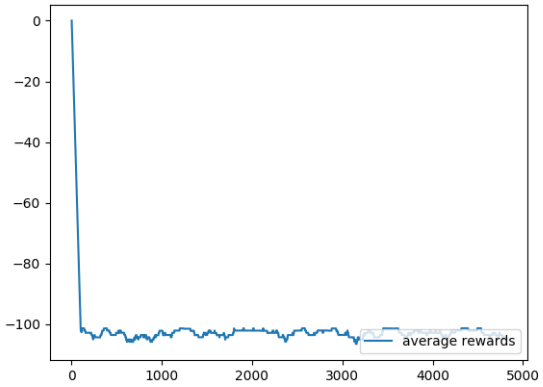

Figura 3-6: Recompensas algoritmo TD3 con simulación de robot bipedal.

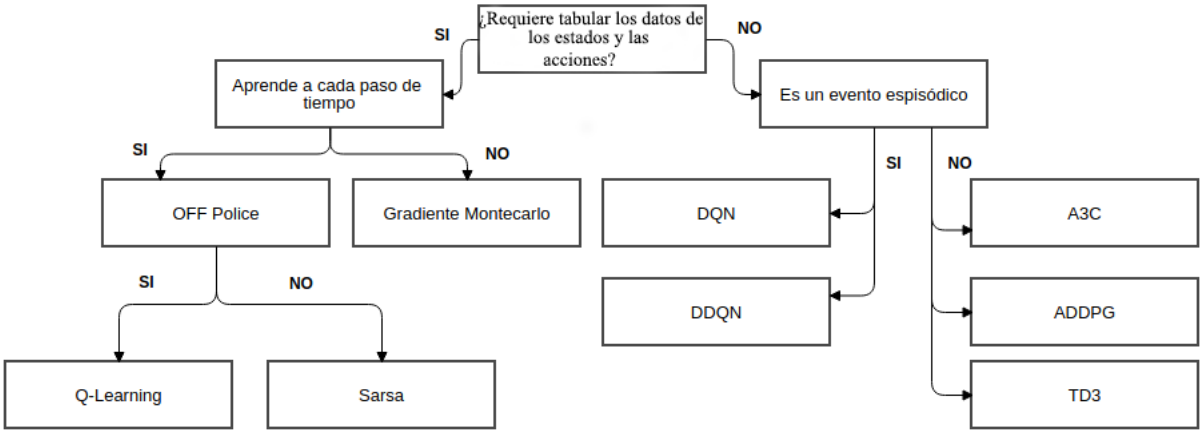

Figura 3-7: Taxonomía de algoritmos de aprendizaje por refuerzo.

de una cámara de vídeo o un láser de alta resolución, además que es una solución con baja generalización.

Una vez definido lo anterior se debe analizar si el evento es episódico o continuo ver Figura **3-7**. En conclusión, como parte del diseño del experimento para esta tesis la información del sistema se obtendrá de un láser. Además, se busca una buena generalización con un seguimiento por episodios debido a que el robot reinicia cada vez que se estrella con un obstáculo, por lo tanto, se opta por DQN que es reconocido como la base de los RL con redes neuronales con enfoque episódico.

3.3. Conclusión del capítulo

La estrategia de este capítulo puede ser utilizado con otros métodos diferentes a los aquí presentados, para lo cual se debe analizar las características y enfoques de diseño. El comportamiento de los algoritmos están influenciados por la configuración de sus parámetros, en especial, si se trabaja con redes neuronales y si va primar le exploración o la explotación del entorno, además, los criterios de selección de acción se deben definir ya sea Boltzmann o ϵ-greedy.

4 Implementación del algoritmo

4.1. Transferencia de OpenAIGym a una plataforma robótica

En la navegación autónoma de robots existe una gran variedad de fenómenos físicos que son complejos de modelar como la interacción de fluidos, cuerpos flexibles o el área de superficie. En este sentido se propone la adaptación de acciones en vez de políticas y, por lo tanto, se tiene en cuenta como parámetros el estado ($s \in S$), acciones ($a \in A$) y observaciones ($o \in O$) [73]. También en [73] se define la trayectoria como un conjunto de observaciones y acciones $\tau = (o, a, o_n, s_n, o_{n+1}, s_{n+1}, ...)$ donde el sistema pasa de un estado a otro $T = (s, a) = s'$ y donde τ_{-k} se refiere a las últimas k observaciones donde las acciones dependen de las últimas políticas π donde $a = \pi(\tau_{-k})$.

Los algoritmos de aprendizaje por refuerzo DQN y DDPG demuestran ser efectivos en la navegación autónoma de robots, pero DQN conlleva dificultades en la transferencia de aprendizaje, por lo tanto es necesario definir características para predecir las recompensas de manera confiable y estimar cómo evolucionan estas características con el tiempo. Permitiendo así, el diseño de redes neuronales que realicen soluciones de manera progresiva y facilitar la planeación de rutas en entornos interiores.

4.1.1. Clasificación o regresión en la implementación

Las actividades de este tipo de algoritmo puede ser analizadas desde dos perspectivas diferentes. La primera sería analizarla a manera de clasificación, para lo cual la red neuronal tiene varias salidas que representan la acción a ejecutar. En el segundo caso se maneja como una regresión donde el resultado final de la NN representa el ángulo de desplazamiento de la plataforma robótica y las acciones son tomadas a modo de un valor continuo que obtiene un rango de $(0, 2\pi)$ y, por lo tanto cuenta con una sola salida. En este proyecto se experimenta con DQN que es un algoritmo representativo del aprendizaje por refuerzo y que permite aplicar soluciones a problemas episódicos con redes neuronales.

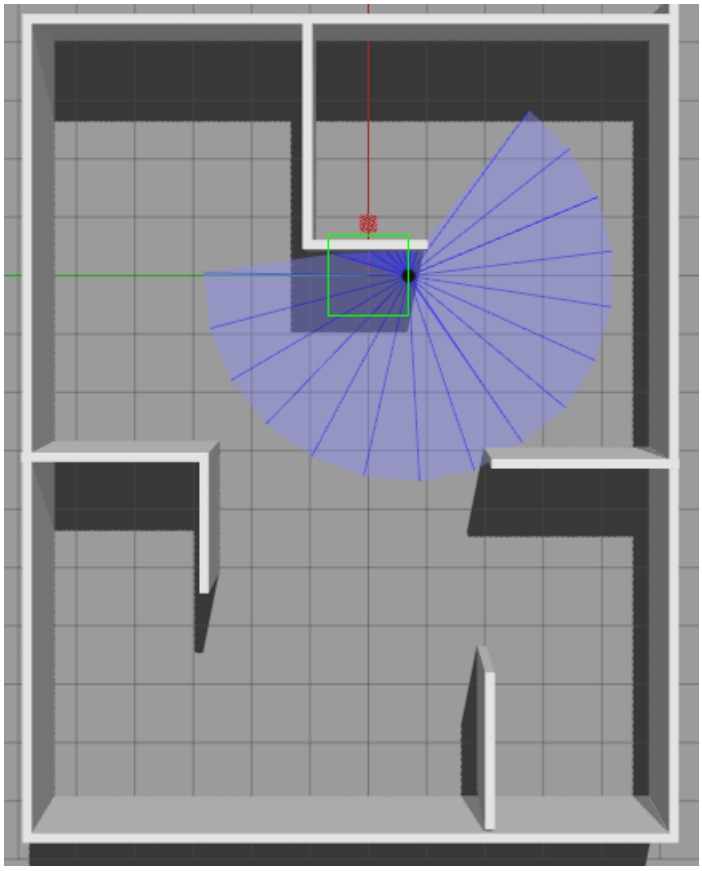

Figura 4-1: Plataforma Robotica Turtlebot con Gazebo.

4.2. Herramientas de simulación

La mayoría de autores poseen diferentes formas de evaluar los algoritmos pero coinciden en que lo mas conveniente es hacer una simulación antes de probarlos en el mundo real.

4.2.1. Gazebo

El simulador Gazebo está enfocado principalmente a la robótica donde se puede seleccionar diferentes tipos de plataformas como Robotino que pertenece al sector privado Figura **4-2** o la Turtlebot que es hardware open source Figura **4-3**, además, el simulador incluye ambientes con obstáculos dinámicos y estáticos donde se pueden realizar pruebas implementando algoritmos como Q-learning, DQN, Actor-Critic, ADDPG entre otros.

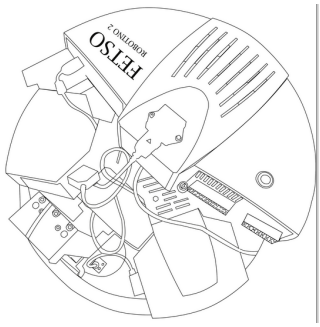

Figura 4-2: Robot Festo Robotino

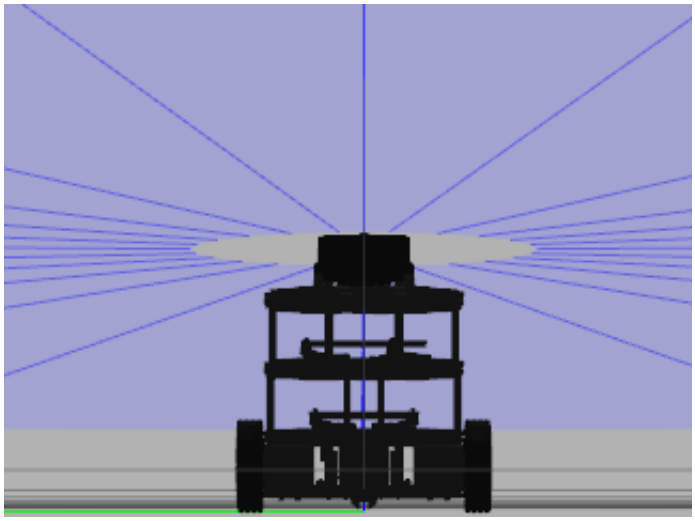

Figura 4-3: Plataforma Robotica Turtlebot.

Tabla 4-1: Velocidad angular según la acción seleccionada en DQN

Acción	Velocidad angular (π/s)
0	-1,5.
1	-0,75
2	0
3	0,75
4	1,5

4.2.2. Turtlebot3

El robot utilizado en este experimento es el Turtlebot3 Burguer Figura **4-3** del simulador Gazebo que es un robot diseñado con fines educativos con un tamaño de 138 mm x 178 mm x 192 mm (L x W x H) y, por lo tanto, es uno de los robots más pequeños de la línea Turtlebot. Esta plataforma está equipada con un sensor laser con un rango de 360 grados

con hardware y software open source, lo que significa que se puede compartir y construir los planos y el código. La configuración que se usa para la implementación se encuentra en la Tabla 4-1.

4.3. Clases y métodos utilizados en Python

A continuación se explican algunos métodos y clases comunes que se codifican para implementar los algoritmos en Python, antes de realizar dicha implementación se configuró el computador según lo indica los items de la Tabla 4-2.

Tabla 4-2: Recomendación para RL en Linux

Item	recomendaciòn
1	Versión de Python.
2	Compatibilidad de versiones librerías.
3	Dimensión de la observación (state).
4	Dimensión de las acciones.
5	Tamaño I/O de las redes neuronales.
6	CUDA correctamente instalado.
7	Deshabilitar la contraseña de supervisión

4.3.1. Enviroment

Obtiene la información del ambiente para las entradas de una red neuronal o un arreglo bidimensional que permite predecir decisiones sobre las acciones con mas probabilidades de alcanzar la meta.

4.3.2. Reset

Que reinicia la ubicación de la plataforma robótica en el ambiente virtual.

4.3.3. Distancy

Obtiene los valores (x, y) del destino y la posición actual del robot, además aplicando la ecuación de pitágoras se obtiene la distancia entre la plataforma robótica y la meta.

4.3.4. Yaw

Informa la dirección de la plataforma robótica que se puede obtener al convertir los cuaterniones en ángulos por medio del teorema de rotación de Euler.

4.3.5. GetState

Se segmenta el espacio en un numero finito de grados y luego por medio de un instrumento de detección se obtiene las diferentes distancias a las que se encuentra los objetos. Es de resaltar que es posible encontrar valores fuera de escala (infinito) por la ausencia de algún elemento que interrumpa el haz de luz, para esto es aconsejable convertirlo a un rango aceptable, por ejemplo 3.5 mts.

4.3.6. GetAction

Se obtiene la acción a realizar basada en las políticas definidas. En el caso del DQN, la acción se maneja como un problema de clasificación que propone cinco diferentes salidas en la red neuronal (Tabla 4-2).

4.4. Descripción del experimento

El algoritmo implementado en la plataforma robótica obtiene una recompensa inmediata en cada paso (r) y el valor de esta depende del estado (s) en que se encuentre. En caso de que el agente llegue a la meta se le asigna ($r = 200$) y el episodio continúa y en caso de que el robot colisione con un obstáculo se penaliza con ($r = -200$) reiniciando el robot, para volver a la posición $x = 0, y = 0$.
El objetivo de implementar una recompensa inmediata es obtener al final del episodio una acumulación de estas e incentivar la búsqueda de la meta en el algoritmo ver Ecuación 4-1 donde γ es un factor de descuento entre $(0, 1)$ que define la importancia de las actuales y futuras recompensas.

$$R_t = \sum \gamma r_\tau \tag{4-1}$$

En la Figura **4-1** se aprecia la plataforma robótica en el simulador Gazebo rodeado de un haz láser de color azul dividido en 24 secciones para abarcar su entorno en un ángulo de 360 grados que detectan los obstáculos subyacentes y la posición de la meta que está representada por un plano rojo rectangular casi transparente . El entrenamiento tiene 6000 episodios con las condiciones de aprendizaje que busca alcanzar el rectángulo de llegada, disminuir las veces que colisiona y llegar al número máximo de pasos por experimento.
En cuanto a la capa neuronal esta diseñada como una red neuronal profunda con tres capas completamente conectada, donde los valores de entrada son los 24 puntos medidos por el laser lidar, más el ángulo de dirección del modelo simulado.
La metodología para realizar la planeación de la ruta sigue los siguientes pasos en el algoritmo DQN: 1) se obtiene la información del estado (s) por medio del sistema sensorial del robot y es asignado como el estado actual, 2) Toma el entrenamiento para escoger la acción óptima con

el mayor valor Q correspondiente al valor actual $Q(s, a) = max_{a'}Q(s, a')$, 3) El rendimiento se obtiene del estado siguiente, 4) se modifica el estado actual s=s' y se determina la posición de objetos cercanos y repite los pasos 2 y 3.

4.5. Conclusión del capítulo

La simulación implementada en Gazebo tiene una gran similitud a lo que se hace en el mundo real, con la ventaja de que permite al investigador realizar los ajustes necesarios, obtener representaciones automáticas y generar funciones de recompensas a partir de los datos simulados sin arriesgar la plataforma robótica. Pero, a pesar de todas estas ventajas, cabe recordar que siempre existirán errores en los modelos resultantes y, por lo tanto, se debe ser precavido si se desconoce la verdadera dinámica del sistema y la precisión del modelo construido.

5 Validación de la estrategia

Como estrategia de validación se propone modificar los valores de algunos parámetros del algoritmo DQN y visualizar cual es la mejor configuración en la planeación de rutas para entornos interiores, para ello existen hiper-parámetros que modifican notablemente el comportamiento Figura **5-1**. En este sentido se aprecia la posibilidad de variar la profundidad de la red neuronal, el número de neuronas, el tipo de optimizador o el método de búsqueda.

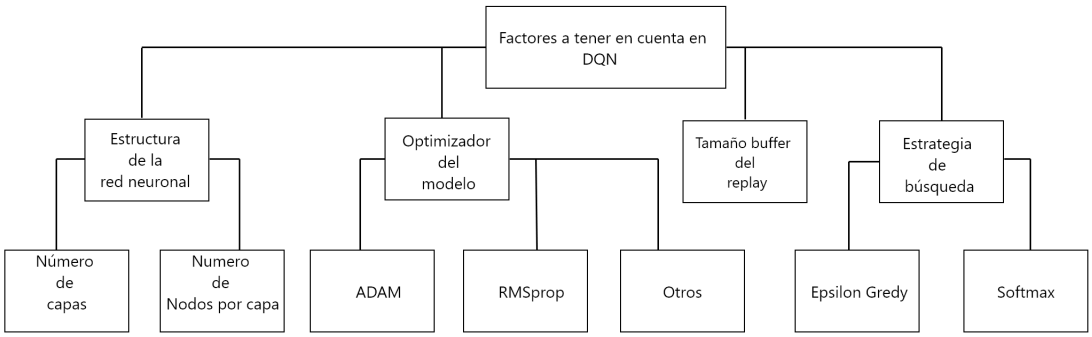

Figura 5-1: Trayectoria ideal de la plataforma robótica.

En la Figura **5-2** se visualiza la estructura de dos redes neuronales profundas con dos capas densas de 64 bits y una capa de salida de 5 bits, más una capa de activación Relu.
Train-start 64

En la Figura **5-3** se configura el experimento a 120 episodios con una variación de puntaje de -5000 a 3100, utilizando una rata de aprendizaje 0.05 y un optimizador RMSprop, que nos permite visualizar valores negativos que van disminuyendo hasta casi desaparecer en el episodio 50. Luego, en episodio 60 no se percibe un incremento significativo en las recompensas obtenidas. De lo anterior, se puede inferir que la plataforma robótica se estrella frecuentemente en los primeros 50 episodios con los muros del ambiente y después del episodio 60 no se visualiza un aprendizaje.

Tabla 5-1: Hiperparametros algoritmo DQN

Hyper parameteros	Valor
Episode-step	6000
Target-update	2000
Discount-factor	0.99
Epsilon	1.0
Epsilon-decay	0.99
Epsilon-min	0.05
Batch-size	64
mMemory	1000000

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                1856
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 5)                 325
_________________________________________________________________
activation_1 (Activation)    (None, 5)                 0
=================================================================
Total params: 6,341
Trainable params: 6,341
Non-trainable params: 0
_________________________________________________________________
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_4 (Dense)              (None, 64)                1856
_________________________________________________________________
dense_5 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_6 (Dense)              (None, 5)                 325
_________________________________________________________________
activation_2 (Activation)    (None, 5)                 0
=================================================================
Total params: 6,341
Trainable params: 6,341
Non-trainable params: 0
```

Figura 5-2: Recompensas en cada pasos del robot a través explora el ambiente

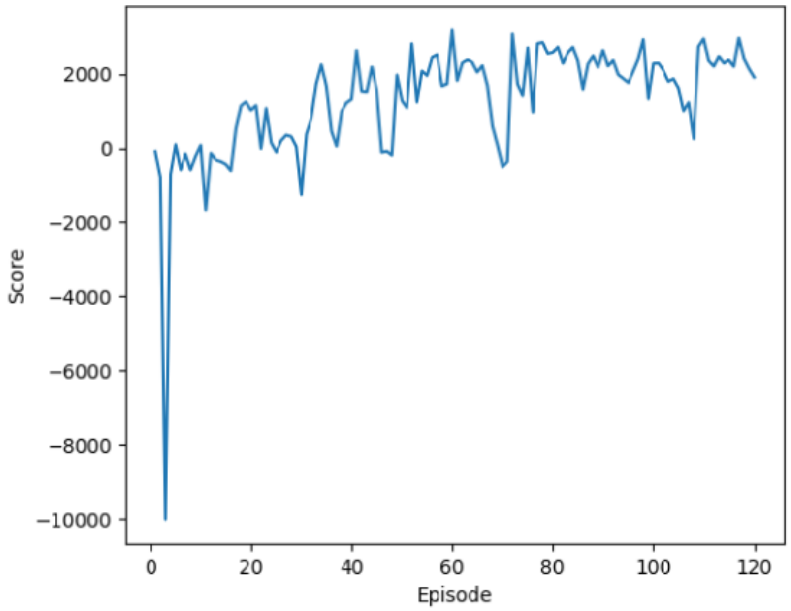

Figura 5-3: Recompensa generada en 120 episodios por el algoritmo por refuerzo DQN con una rata de aprendizaje de 0.05 y el optimizador RMSprop

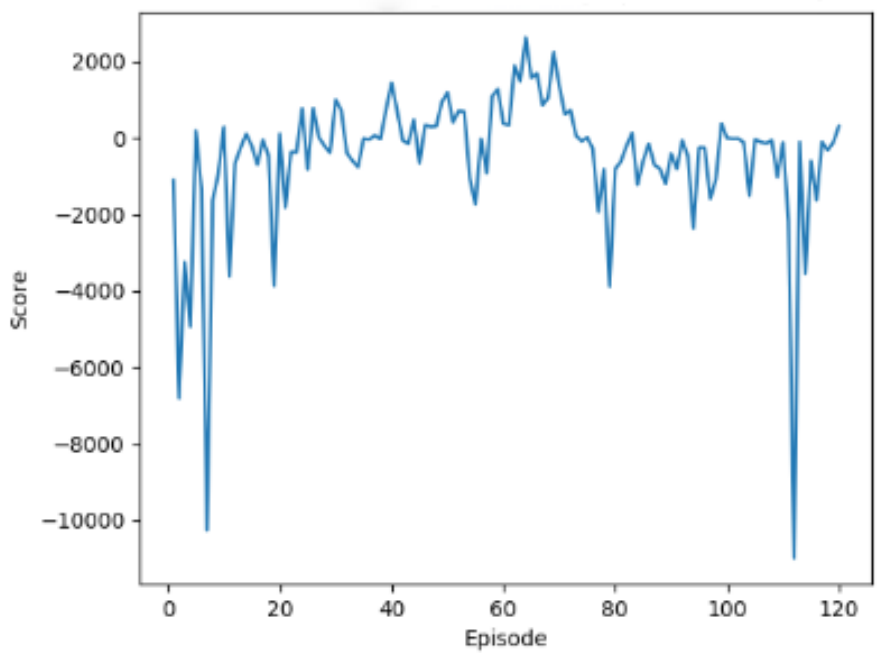

Figura 5-4: Recompensa por episodios con rata de aprendizaje 0.1 y optimizador Adam

En la Figura **5-4** se configura el experimento a 120 episodios y se observa una variación de puntaje de -11000 a 2100 utilizando una rata de aprendizaje 0.1 y un optimizador Adam que permite visualizar que el algoritmo en la plataforma robótica obtiene valores negativos que tienden a 0, pero sostiene una tendencia negativa a partir del episodio 65. La gráfica muestra que con esta configuración el resultado es pobre y no tiende a buscar la meta e incluso vuelve a chocar con objetos a pesar de que el modelo conserva el mismo número de episodios de entrenamiento de los otros experimentos.

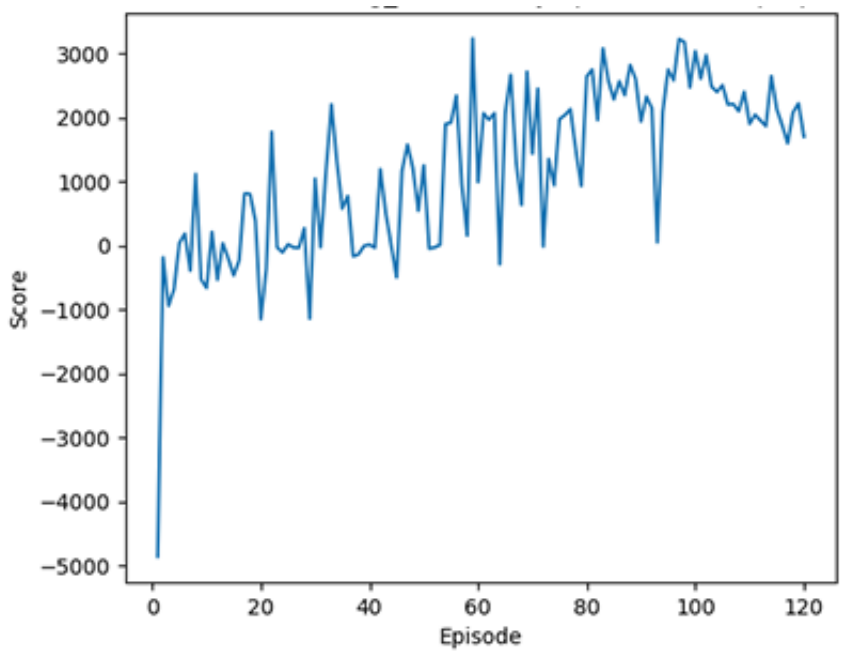

Figura 5-5: Optimizador RMSprop, rata de aprendizaje 0.01

En la figura **5-5** el experimento se configura a 120 episodios y se observa la variación en el puntaje de -11000 a 3100 utilizando una rata de aprendizaje 0.01 y un optimizador Adam que permite visualizar valores negativos en los primeros episodios de la gráfica, pero que obtiene una tendencia positiva rápidamente. Dicha gráfica muestra que el robot aprende a evitar obstáculos con un excelente rendimiento para encontrar la meta, es decir, en este modelo prima la explotación rápida el conocimiento en vez de la exploración.

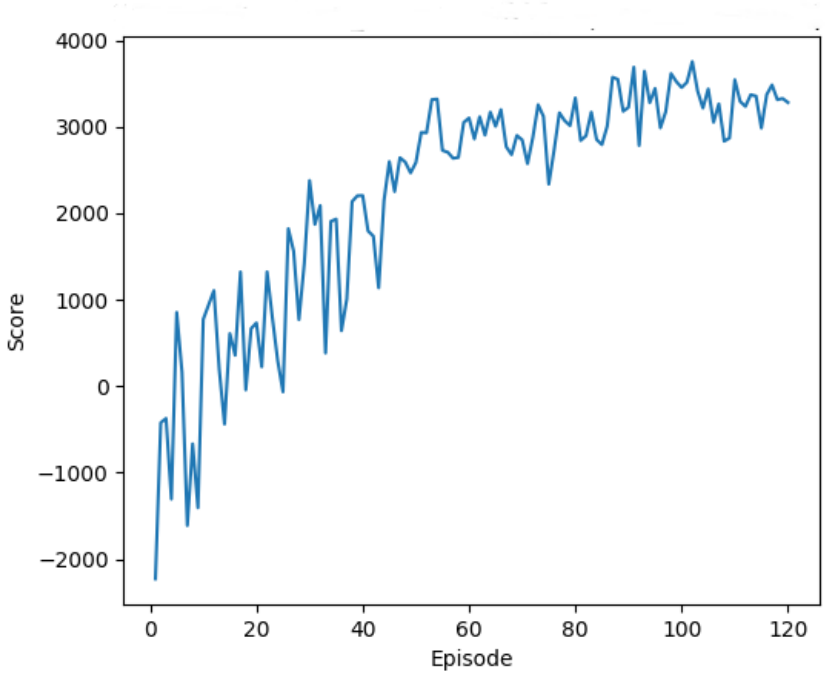

Figura 5-6: Rata de aprendizaje de 0.00025 y optimizadr RMSprop

En la figura **5-5** el experimento se configura a 120 episodios y se observa una variación de -2000 a 3900 utilizando una rata de aprendizaje 0.00025 y un optimizador RMSprop que permite visualizar valores negativos en los primeros episodios de la gráfica, pero que también logra una tendencia positiva rápidamente. Sin embargo, comparado con la prueba anterior **5-5** los episodios se incrementan negativamente lo que demuestra una preferencia a la exploración, para obtener un alto puntaje en la explotación, es decir, las decisiones aleatorias son mayores al principio, no obstante, es mejor al final y, por lo tanto, es la configuración que mejores resultados presenta hasta el momento.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_1 (Dense)              (None, 64)                1856
_________________________________________________________________
dense_2 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout_1 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_3 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_4 (Dense)              (None, 5)                 165
_________________________________________________________________
activation_1 (Activation)    (None, 5)                 0
=================================================================
Total params: 8,261
Trainable params: 8,261
Non-trainable params: 0
_________________________________________________________________
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_5 (Dense)              (None, 64)                1856
_________________________________________________________________
dense_6 (Dense)              (None, 64)                4160
_________________________________________________________________
dropout_2 (Dropout)          (None, 64)                0
_________________________________________________________________
dense_7 (Dense)              (None, 32)                2080
_________________________________________________________________
dense_8 (Dense)              (None, 5)                 165
_________________________________________________________________
activation_2 (Activation)    (None, 5)                 0
=================================================================
Total params: 8,261
Trainable params: 8,261
Non-trainable params: 0
```

Figura 5-7: Estructura de la red neuronal con una capa oculta adicional de 32 bits

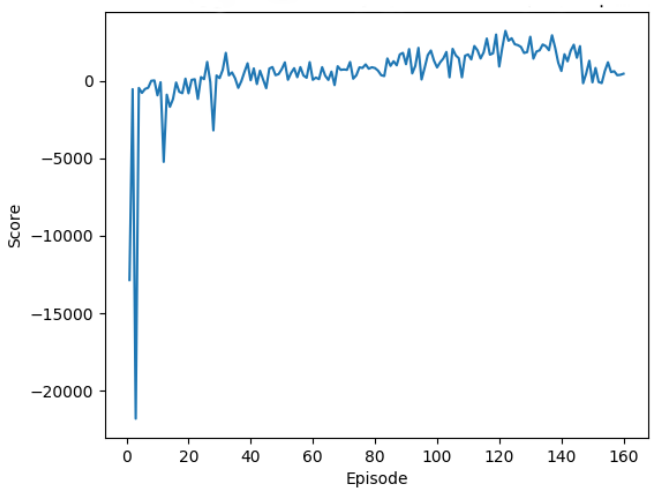

Figura 5-8: Capa escondida Adicional con rata de aprndizaje de 0.00025 y optimizador RMSprop

En la Figura **5-8** se configura el experimento a 120 episodios y se observa una variación de score de -20000 a 3900 utilizando una rata de aprendizaje 0.00025 y un optimizador RMSprop con una red densa de 32 bits oculta adicional que permite visualizar valores negativos en los primeros episodios de la gráfica y que obtiene una tendencia positiva rápidamente. Sin embargo, se puede apreciar que después del episodio 120 la pendiente empieza a decrecer, por la tanto, esta configuración en 120 episodios no es la mejor comparada con la red neuronal que tiene la misma estructura de hiper-parámetros sin la capa de 32 bits.

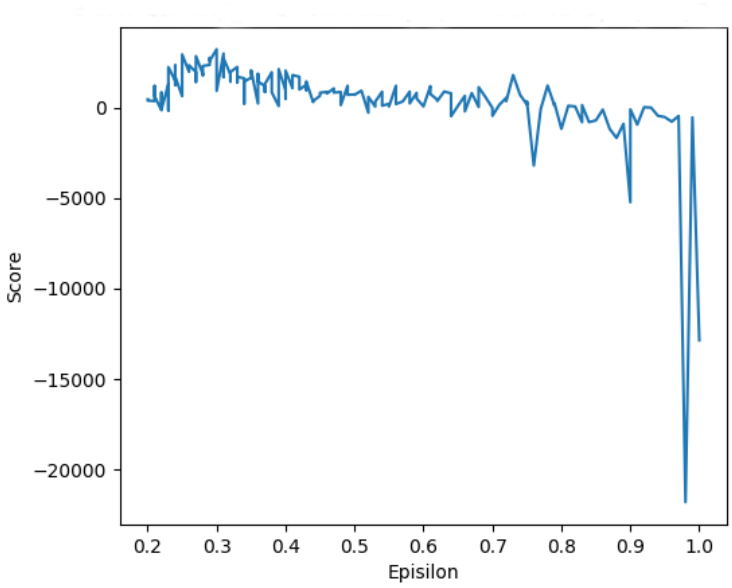

Figura 5-9: Epsilon rata de aprendizaje 0.00025, optimizador RMSprop con capa adicional

5.1. Resultados de la planeación de rutas

En la Figura **5-10** se visualiza cual sería la ruta ideal, pero los resultados del experimento fueron muy diferentes debido a que el algoritmo aprendió a planear las rutas con cierto nivel de tolerancia. En realidad, el recorrido es más parecido a la figura **5-11**, donde se aprecia una trayectoria asimétrico con una longitud, curvatura y tiempo muy diferente al ideal.
En este orden de ideas el robot llego a la meta el 93 por ciento de las veces.

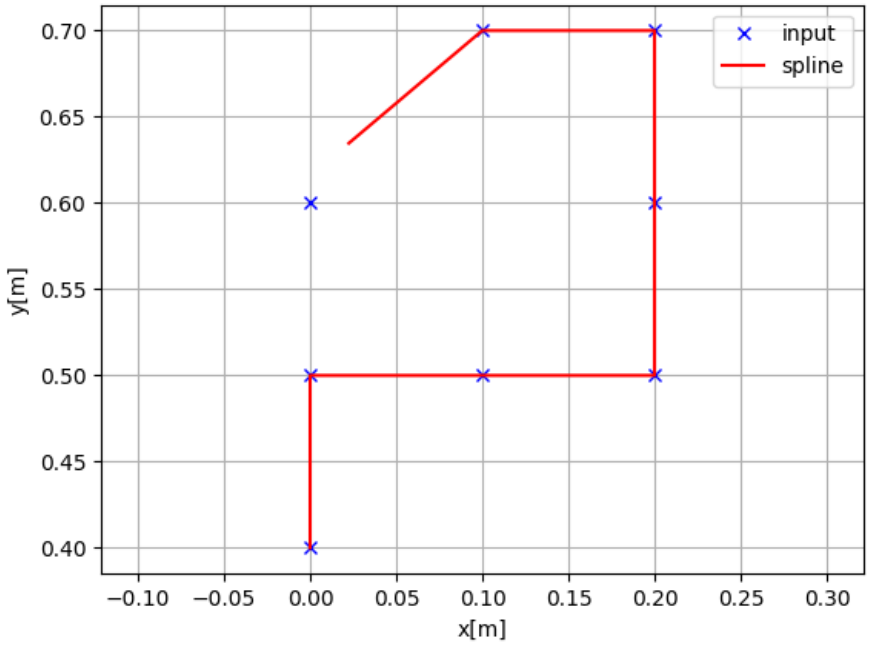

Figura 5-10: Trayectoria ideal de la plataforma robótica.

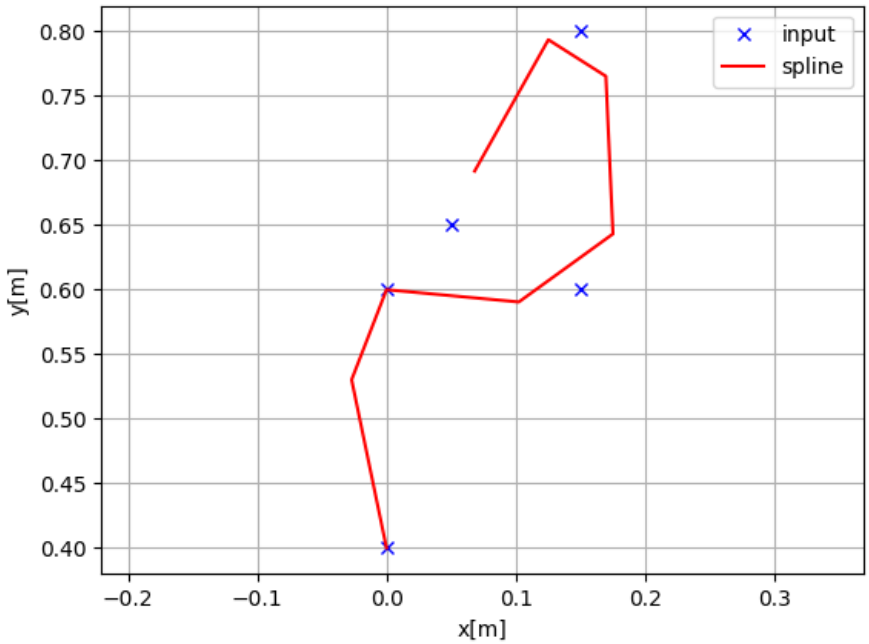

Figura 5-11: Trayectoria seleccionada por la plataforma robótica en uno de sus episodios para llegar a la meta.

5.1.1. Metricas propuesta para la exploración

En esta tesis se propone un métrica para evaluar el rendimiento de un algoritmo de aprendizaje por refuerzo para etapas tempranas de exploración que realiza múltiples recorridos antes de converger. Por lo tanto, la métrica propuesta tiene tres variables a saber: La primera es cuando alcanza la meta $s_{t=goal}$, la segunda es si el tiempo de colapso (c_t) aumenta a medida que crece los episodios y la tercera es en el momento que llega al numero máximos de pasos en el episodio s_{end} todas estas variables son ponderas por sus pesos según los requerimientos del programador ($W_{(0,1,2)}$):

$$\vartheta = \frac{\sum_{t=1}^{t=episodes} w_0 c_t + w_1 s_{t=goal} + w_2 s_{t=end}}{episodes} = \begin{cases} c_t < c_{t+1} = 1 \\ c_t > c_{t+1} = -1 \\ s_{t=goal} = 1 \\ s_{t=end} = 1 \end{cases} \tag{5-1}$$

Esta métrica se propone para futuras pruebas donde el número de episodios en la exploración sean demasiados.

5.2. Conclusión del capítulo

En esta etapa se realizan diferentes pruebas para definir algunos hiper-parámetros para configurar la estructura de la red neuronal de la plataforma robótica lo cual es una actividad empírica basada en la corrección de los experimentos. Es así, que la configuración que sobresalió fue con una rata de aprendizaje de 0.00025 que es la menor utilizada en los ensayos, la distribución que mejor se acopló para la solución del experimento fue con dos capas ocultas y el optimizador RMSprop fue el que proporcionó un rendimiento superior. Por último, al recorrido le falta mejorar la simetría para poder compararse con lo que haría un ser humano.

6 Conclusiones y recomendaciones

6.1. Conclusiones

En las últimas investigaciones sobre el aprendizaje por refuerzo aplicado al ARN se observa como se incrementa el interés de este tema en el mundo. Lo anterior es debido a que los investigadores encuentran en la inteligencia artificial un nuevo campo de exploración para solucionar el problema de la navegación autónoma de robots.
Por otra parte se observa que en la mayoría de las publicaciones simulan antes de implementar los algoritmos en una plataforma real, y aunque se reconoce las limitaciones por la falta de variables como el ruido ambiental o el error de los sensores, se logró buenos resultados. En cuanto los algoritmos y sensores usados existen una gran diversidad de opciones, pero está claro que el uso de redes neuronales para interpretar los datos de gran dimensionalidad obtenida por los sensores es lo común en el estado del arte.
Además en la actualidad la navegación autónoma esta dominada por algoritmos de aprendizaje supervisado o bio-inspirados, pero la investigación del aprendizaje por refuerzo está ganando espacio, puesto que tiende a generar nuevas alternativas a las soluciones actuales. En este sentido, el diseño de la estrategia propuesta permitió implementar el algoritmo DQN en la plataforma robótica Turtlebot que busca competentemente la meta y evitar obstáculos de manera eficaz.
Por otra parte los hiper-parámetros permiten configurar la estructura de la red neuronal de la plataforma robótica, lo cual es una actividad empírica basada en la corrección de las pruebas que se hacen. Es así que los experimentos arrojaron que la configuración con mejores resultados fue con una rata de aprendizaje de 0.00025 que es la menor utilizada en los experimentos, la estructura que mejor se acopló para la solución del experimento fue con dos capas densas ocultas y el optimizador RMSprop fue el que obtuvo mejor rendimiento. En cuanto a la ruta se evidencia una falta de simetría para poder compararlo con lo que haría un ser humano.
En términos generales el aprendizaje por refuerzo abre nuevas posibilidades de investigación en el dominio de la robótica y el uso de algoritmos cada vez más poderosos permitirá desarrollar robots más independientes de la intervención humana.
La estrategia de selección aquí presentada puede ser utilizado con otros algoritmos diferentes al DQN para lo cual se debe tener en cuenta sus características y enfoques con que fue diseñado. El comportamiento de los algoritmos también esta influenciados por la configuración de sus parámetros en especial si se trabaja con redes neuronales. Por último se debe tener

en cuenta si va primar le exploración o la explotación del entorno y los criterios de selección de acción que se escojan como Boltzmann o ϵ-greedy.

6.2. Trabajos futuros

En esta época del virus covid-19 la posibilidad de seguir la trayectoria recorrida en un ambiente interior por medio de un robot puede servir para mantener una adecuada limpieza. También la posibilidad de usar los algoritmos de aprendizaje por refuerzo para detectar peatones por medio de simulaciones puede ayudar a evitar accidentes en entornos inesperados donde no ha sido entrenado efectivamente o no se tiene previsto en el algoritmo preestablecido. Otro posible campo de investigación es en el área de la medicina, donde una prótesis humana puede aprender los diferentes esfuerzos que deben hacer dependiendo del contexto del recorrido.

Bibliografía

[1] Niels Justesen y col. "Deep learning for video game playing". En: *IEEE Transactions on Games* 12.1 (2019), págs. 1-20.

[2] Volodymyr Mnih y col. "Playing atari with deep reinforcement learning". En: *arXiv preprint arXiv:1312.5602* (2013).

[3] Joohyun Woo y Nakwan Kim. "Vision based obstacle detection and collision risk estimation of an unmanned surface vehicle". En: *2016 13th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. IEEE. 2016, págs. 461-465.

[4] Herke van Hoof. "Machine Learning through Exploration for Perception-Driven Robotics". Tesis doct. Technische Universität Darmstadt, 2016.

[5] Min Hyuk Woo, Soo-Hong Lee y Hye Min Cha. "A study on the optimal route design considering time of mobile robot using recurrent neural network and reinforcement learning". En: *Journal of Mechanical Science and Technology* 32.10 (2018), págs. 4933-4939.

[6] Urcera I Martín y col. "Generación de trayectorias robóticas mediante aprendizaje profundo por refuerzo". Tesis de mtría. Universitat Politècnica de Catalunya, 2018.

[7] Gary G Yen y Travis W Hickey. "Reinforcement learning algorithms for robotic navigation in dynamic environments". En: *ISA transactions* 43.2 (2004), págs. 217-230.

[8] Timothy P Lillicrap y col. "Continuous control with deep reinforcement learning". En: *arXiv preprint arXiv:1509.02971* (2015).

[9] Anish Pandey, S Pandey y DR Parhi. "Mobile robot navigation and obstacle avoidance techniques: A review". En: *Int Rob Auto J* 2.3 (2017), pág. 00022.

[10] Samuel Chenatti y col. "Deep Reinforcement Learning in Robotics Logistic Task Coordination". En: *2018 Latin American Robotic Symposium, 2018 Brazilian Symposium on Robotics (SBR) and 2018 Workshop on Robotics in Education (WRE)*. IEEE. 2018, págs. 326-332.

[11] Pablo Quintía Vidal. "Robots capaces de aprender y adaptarse al entorno a partir de sus propias experiencias". Tesis doct. Universidade de Santiago de Compostela, 2013.

[12] Pablo Escandell Montero. "Aprendizaje por refuerzo en espacios continuos: algoritmos y aplicación al tratamiento de la anemia renal". Tesis doct. Universitat de València, 2014.

[13] Marco Wiering y Martijn Van Otterlo. *Reinforcement learning*. Vol. 12. Springer, 2012.

[14] Gregory Kahn y col. "Self-supervised deep reinforcement learning with generalized computation graphs for robot navigation". En: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, págs. 1-8.

[15] Anis Koubâa y col. *Robot Path Planning and Cooperation*. Vol. 772. Springer, 2018.

[16] O Khatib y JF Le Maitre. "Dynamic controlof manipulators operating in a complex environment". En: *On Theory and Practice of Robots and Manipulators, 3rd CISM-IFToMM Symp*. Vol. 267. 1978.

[17] Nils J Nilsson. *The quest for artificial intelligence*. Cambridge University Press, 2009.

[18] Ramon Gonzalez, Marius Kloetzer y Cristian Mahulea. "Comparative study of trajectories resulted from cell decomposition path planning approaches". En: *2017 21st International Conference on System Theory, Control and Computing (ICSTCC)*. IEEE. 2017, págs. 49-54.

[19] Jan Rosell y Pedro Iniguez. "Path planning using harmonic functions and probabilistic cell decomposition". En: *Proceedings of the 2005 IEEE international conference on robotics and automation*. IEEE. 2005, págs. 1803-1808.

[20] Neerendra Kumar, Zoltán Vámossy y Zsolt Miklós Szabó-Resch. "Robot path pursuit using probabilistic roadmap". En: *2016 IEEE 17th International Symposium on Computational Intelligence and Informatics (CINTI)*. IEEE. 2016, págs. 000139-000144.

[21] Mark Pfeiffer y col. "Reinforced imitation: Sample efficient deep reinforcement learning for mapless navigation by leveraging prior demonstrations". En: *IEEE Robotics and Automation Letters* 3.4 (2018), págs. 4423-4430.

[22] Chen Xia y Abdelkader El Kamel. "Neural inverse reinforcement learning in autonomous navigation". En: *Robotics and Autonomous Systems* 84 (2016), págs. 1-14.

[23] Billy Okal y Kai O Arras. "Learning socially normative robot navigation behaviors with bayesian inverse reinforcement learning". En: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2016, págs. 2889-2895.

[24] Devika S Nair y P Supriya. "Comparison of Temporal Difference Learning Algorithm and Dijkstra's Algorithm for Robotic Path Planning". En: *2018 Second International Conference on Intelligent Computing and Control Systems (ICICCS)*. IEEE. 2018, págs. 1619-1624.

[25] Aleksandr I Panov, Konstantin S Yakovlev y Roman Suvorov. "Grid path planning with deep reinforcement learning: Preliminary results". En: *Procedia computer science* 123 (2018), págs. 347-353.

[26] Boyi Liu, Lujia Wang y Ming Liu. "Lifelong federated reinforcement learning: a learning architecture for navigation in cloud robotic systems". En: *IEEE Robotics and Automation Letters* 4.4 (2019), págs. 4555-4562.

[27] Somil Bansal y col. "Combining optimal control and learning for visual navigation in novel environments". En: *Conference on Robot Learning*. 2020, págs. 420-429.

[28] David Luviano Cruz y Wen Yu. "Path planning of multi-agent systems in unknown environment with neural kernel smoothing and reinforcement learning". En: *Neurocomputing* 233 (2017), págs. 34-42.

[29] Pinxin Long y col. "Towards optimally decentralized multi-robot collision avoidance via deep reinforcement learning". En: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, págs. 6252-6259.

[30] Yue Jin y col. "Efficient multi-agent cooperative navigation in unknown environments with interlaced deep reinforcement learning". En: *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE. 2019, págs. 2897-2901.

[31] Lei Tai, Giuseppe Paolo y Ming Liu. "Virtual-to-real deep reinforcement learning: Continuous control of mobile robots for mapless navigation". En: *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2017, págs. 31-36.

[32] Ashwini Pokle y col. "Deep local trajectory replanning and control for robot navigation". En: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, págs. 5815-5822.

[33] F Lachekhab y M Tadjine. "Goal seeking of mobile robot using fuzzy actor critic learning algorithm". En: *2015 7th International Conference on Modelling, Identification and Control (ICMIC)*. IEEE. 2015, págs. 1-6.

[34] Akram Adib y Behrooz Masoumi. "Mobile robots navigation in unknown environments by using fuzzy logic and learning automata". En: *2017 Artificial Intelligence and Robotics (IRANOPEN)*. IEEE. 2017, págs. 58-63.

[35] Chia-Feng Juang y Trong Bac Bui. "Reinforcement Neural Fuzzy Surrogate-Assisted Multiobjective Evolutionary Fuzzy Systems With Robot Learning Control Application". En: *IEEE Transactions on Fuzzy Systems* 28.3 (2019), págs. 434-446.

[36] Fatemeh Fathinezhad, Vali Derhami y Mehdi Rezaeian. "Supervised fuzzy reinforcement learning for robot navigation". En: *Applied Soft Computing* 40 (2016), págs. 33-41.

[37] Alma Y Alanis y col. "Integration of an Inverse Optimal Neural Controller with Reinforced-SLAM for Path Panning and Mapping in Dynamic Environments". En: *Computación y Sistemas* 19.3 (2015), págs. 445-456.

[38] Vignesh Prasad y col. "Learning to Prevent Monocular SLAM Failure using Reinforcement Learning". En: *Proceedings of the 11th Indian Conference on Computer Vision, Graphics and Image Processing*. 2018, págs. 1-9.

[39] Beril Sirmaçek y col. "Reinforcement learning and slam based approach for mobile robot navigation in unknown environments". En: *ISPRS Workshop Indoor 3D 2019*. 2019.

[40] Zhi Wang y col. "Incremental reinforcement learning with prioritized sweeping for dynamic environments". En: *IEEE/ASME Transactions on Mechatronics* 24.2 (2019), págs. 621-632.

[41] Siti Sendari y col. "Exploration of genetic network programming with two-stage reinforcement learning for mobile robot". En: *Telkomnika* 17.3 (2019), págs. 1447-1454.

[42] Alexander B Filimonov y Nikolay B Filimonov. "The peculiarities of application of the potential fields method for the problems of local navigation of mobile robots". En: *2018 XIV International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE)*. IEEE. 2018, págs. 208-211.

[43] Rajat Kumar Panda y BB Choudhury. "An effective path planning of mobile robot using genetic algorithm". En: *2015 IEEE International Conference on Computational Intelligence & Communication Technology*. IEEE. 2015, págs. 287-291.

[44] Michel Aractingi y col. "Improving the Generalization of Visual Navigation Policies using Invariance Regularization". En: (2019).

[45] Xiangyun Meng y col. "Neural autonomous navigation with riemannian motion policy". En: *2019 International Conference on Robotics and Automation (ICRA)*. IEEE. 2019, págs. 8860-8866.

[46] Ramon Gonzalez, Marius Kloetzer y Cristian Mahulea. "Comparative study of trajectories resulted from cell decomposition path planning approaches". En: *2017 21st International Conference on System Theory, Control and Computing (ICSTCC)*. IEEE. 2017, págs. 49-54.

[47] Yuki Kato, Koji Kamiyama y Kazuyuki Morioka. "Autonomous robot navigation system with learning based on deep Q-network and topological maps". En: *2017 IEEE/SICE International Symposium on System Integration (SII)*. IEEE. 2017, págs. 1040-1046.

[48] Colin Greatwood y Arthur G Richards. "Reinforcement learning and model predictive control for robust embedded quadrotor guidance and control". En: *Autonomous Robots* 43.7 (2019), págs. 1681-1693.

[49] Haobin Shi y col. "End-to-end navigation strategy with deep reinforcement learning for mobile robots". En: *IEEE Transactions on Industrial Informatics* 16.4 (2019), págs. 2393-2402.

[50] Jake Bruce y col. "One-shot reinforcement learning for robot navigation with interactive replay". En: *arXiv preprint arXiv:1711.10137* (2017).

[51] Carlos Celemin, Javier Ruiz-del Solar y Jens Kober. "A fast hybrid reinforcement learning framework with human corrective feedback". En: *Autonomous Robots* 43.5 (2019), págs. 1173-1186.

[52] Seung-Ho Han y col. "Sensor-Based Mobile Robot Navigation via Deep Reinforcement Learning". En: *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*. IEEE. 2018, págs. 147-154.

[53] Liulong Ma y col. "Learning to Navigate in Indoor Environments: from Memorizing to Reasoning". En: *arXiv preprint arXiv:1904.06933* (2019).

[54] Dmytro Mishkin, Alexey Dosovitskiy y Vladlen Koltun. "Benchmarking classic and learned navigation in complex 3D environments". En: *arXiv preprint arXiv:1901.10915* (2019).

[55] Ye Zhou, Erik-Jan van Kampen y Qiping Chu. "Hybrid Hierarchical Reinforcement Learning for online guidance and navigation with partial observability". En: *Neurocomputing* 331 (2019), págs. 443-457.

[56] Lv Qiang y col. "A model-free mapless navigation method for mobile robot using reinforcement learning". En: *2018 Chinese Control And Decision Conference (CCDC)*. IEEE. 2018, págs. 3410-3415.

[57] Mihai Duguleana y Gheorghe Mogan. "Neural networks based reinforcement learning for mobile robots obstacle avoidance". En: *Expert Systems with Applications* 62 (2016), págs. 104-115.

[58] Fidel Aznar Gregori, Mar Pujol, Ramón Rizo y col. "Obtaining fault tolerance avoidance behavior using deep reinforcement learning". En: (2019).

[59] T Tongloy y col. "Asynchronous deep reinforcement learning for the mobile robot navigation with supervised auxiliary tasks". En: *2017 2nd International Conference on Robotics and Automation Engineering (ICRAE)*. IEEE. 2017, págs. 68-72.

[60] Xingyu Zhao y col. "Asynchronous reinforcement learning algorithms for solving discrete space path planning problems". En: *Applied Intelligence* 48.12 (2018), págs. 4889-4904.

[61] Hongjie Geng y col. "Reinforcement extreme learning machine for mobile robot navigation". En: *Proceedings of ELM-2016*. Springer, 2018, págs. 61-73.

[62] Xingyu Zhao y col. "Asynchronous reinforcement learning algorithms for solving discrete space path planning problems". En: *Applied Intelligence* 48.12 (2018), págs. 4889-4904.

[63] Baochang Zhang y col. "Cooperative and geometric learning for path planning of UAVs". En: *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2013, págs. 69-78.

[64] Rushikesh Kamalapurkar y col. *Reinforcement learning for optimal feedback control*. Springer, 2018.

[65] Christopher Gatti. *Design of experiments for reinforcement learning*. Springer, 2014.

[66] Todd Hester. *TEXPLORE: Temporal Difference Reinforcement Learning for Robots and Time-Constrained Domains*. Springer, 2013.

[67] Jinglun Yu, Yuancheng Su y Yifan Liao. "The Path Planning of Mobile Robot by Neural Networks and Hierarchical Reinforcement Learning". En: *Frontiers in Neurorobotics* 14 (2020).

[68] Peter Vamplew y col. "Empirical evaluation methods for multiobjective reinforcement learning algorithms". En: *Machine learning* 84.1-2 (2011), págs. 51-80.

[69] Wen Wu y col. "Deep Deterministic Policy Gradient with Clustered Prioritized Sampling". En: *International Conference on Neural Information Processing*. Springer. 2018, págs. 645-654.

[70] Shauharda Khadka y col. "Collaborative evolutionary reinforcement learning". En: *arXiv preprint arXiv:1905.00976* (2019).

[71] Yoko Sasaki y col. "A3C Based Motion Learning for an Autonomous Mobile Robot in Crowds". En: *2019 IEEE International Conference on Systems, Man and Cybernetics (SMC)*. IEEE. 2019, págs. 1036-1042.

[72] Alfonso B Labao, Mygel Andrei M Martija y Prospero C Naval. "A3C-GS: Adaptive Moment Gradient Sharing With Locks for Asynchronous Actor-Critic Agents". En: *IEEE Transactions on Neural Networks and Learning Systems* (2020).

[73] Paul Christiano y col. "Transfer from simulation to real world through learning deep inverse dynamics model". En: *arXiv preprint arXiv:1610.03518* (2016).