GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

Desarrollo de un sistema de navegación para un robot móvil utilizando aprendizaje por refuerzo

Autor: Emmanuel Luque Bayón

Director: Jaime Boal Martín-Larrauri

Director: Álvaro Jesús López López

Madrid

Agosto de 2020

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

Desarrollo de un sistema de navegación para un robot móvil utilizando aprendizaje por refuerzo

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el curso académico 2019/20 es de mi autoría, original e inédito y no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada de otros documentos está debidamente referenciada.

Fdo.: Emmanuel Luque Bayón Fecha: 22/ 08/ 2020

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Jaime Boal Martín-Larrauri Fecha: 22/ 08/ 2020

Firmado digitalmente por BOAL MARTIN LARRAURI JAIME - 05304600H
Fecha: 2020.08.22 13:02:14 +02'00'

Fdo.: Álvaro Jesús López López Fecha: 22/ 08/ 2020

GRADO EN INGENIERÍA EN TECNOLOGÍAS INDUSTRIALES

TRABAJO FIN DE GRADO

Desarrollo de un sistema de navegación para un robot móvil utilizando aprendizaje por refuerzo

Autor: Emmanuel Luque Bayón

Director: Jaime Boal Martín-Larrauri

Director: Álvaro Jesús López López

Madrid

Agosto de 2020

Agradecimientos

Quisiera expresar mi agradecimiento a mis directores de proyecto Jaime Boal Martín-Larrauri y Álvaro Jesús López López, quienes con su inestimable ayuda y dedicación lideraron el camino en el desarrollo de este proyecto.

Agosto 2020

Emmanuel Luque Bayón

DESARROLLO DE UN SISTEMA DE NAVEGACIÓN PARA UN ROBOT MÓVIL UTILIZANDO APRENDIZAJE POR REFUERZO

Autor: Luque Bayón, Emmanuel.
Director: Boal Martín-Larrauri, Jaime.
Director: López López, Álvaro Jesús.
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas.

RESUMEN DEL PROYECTO

En este proyecto se han realizado numerosas pruebas de simulación para que un robot móvil aprenda a recorrer un circuito cuadrangular sin chocarse mediante aprendizaje por refuerzo. Como resultado el robot ha aprendido a recorrer el circuito exitosamente y sin errores.

Palabras clave: Robot móvil, Navegación, Inteligencia artificial, Aprendizaje por refuerzo, Simulación, CoppeliaSim, Python

1. Introducción

El aprendizaje por refuerzo es, junto al aprendizaje supervisado y al no supervisado, una de las principales técnicas de 'Machine Learning' o 'Aprendizaje automático'. Actualmente es empleado en distintas áreas como ingeniería, matemáticas, economía, informática, psicología o neurociencia para multitud de tareas: optimización de controles, investigación operativa, estudio del comportamiento humano, aprendizaje automático, etc.

El objetivo del aprendizaje por refuerzo es lograr que un agente aprenda a realizar una tarea de manera óptima y en el menor tiempo posible en un entorno determinado. Para ello es necesario asignar unos objetivos, de manera que el agente recibirá una recompensa al alcanzarlos y una penalización en caso de no lograrlo. De esta manera el agente aprenderá a valorar qué acciones es conveniente realizar en cada momento para lograr recibir la máxima recompensa posible a largo plazo.

2. Definición del Proyecto

El motivo principal que ha dado lugar al desarrollo de este proyecto ha sido el hecho de que la inteligencia artificial, y en concreto el aprendizaje por refuerzo, es el futuro en cuanto a la optimización de procesos y, ante la necesidad de lograr optimizar el control de nuestro robot virtual para las clases de laboratorio de ciertas asignaturas de ICAI, se ha decidido probar su eficacia en este proyecto.

De esta manera se logra averiguar si con el control PD convencional que se diseñó en el pasado para el mismo circuito nos hallamos cerca o lejos de la solución óptima que es capaz de aportarnos el algoritmo de aprendizaje por refuerzo.

3. Descripción del sistema

El proyecto se ha llevado a cabo inicialmente en el entorno de una máquina virtual de Linux sobre Windows mediante la plataforma de simulación de CoppeliaSim y programando el código de manera externa en Python. Posteriormente se realizó el cambio de entorno a Windows para mayor simplicidad.

El esquema general del proyecto se representa en la *Ilustración 1*, donde aparece la transferencia de datos que tiene lugar entre el entorno (virtual o de simulación V-Rep) y el agente (implementado con código Python).

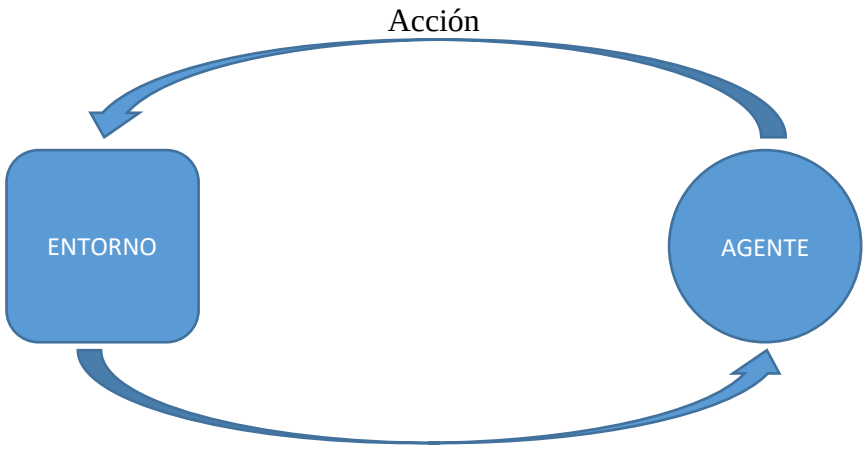

Ilustración 1 Arquitectura General del Sistema

Para la modelización de la política del robot se ha utilizado una red neuronal de 4 capas diseñada mediante Keras con 12 neuronas en la capa de entrada cuyas neuronas de las capas intermedias se encuentran dispuestas según la *Ilustración 2*.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 48)                624
_________________________________________________________________
dense_1 (Dense)              (None, 24)                1176
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 125
=================================================================
Total params: 1,925
Trainable params: 1,925
Non-trainable params: 0
```

Ilustración 2 Estructura de la Red Neuronal

El circuito cuadrangular que se ha usado para realizar las pruebas aparece representado en le *Ilustración 3*.

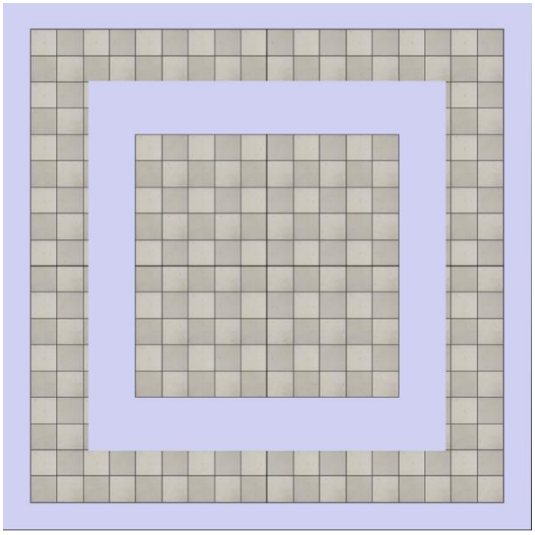

Ilustración 3 Circuito Cuadrangular

El robot móvil virtual que se ha usado en este proyecto es el Pioneer P3DX [1].

Las acciones que puede realizar el robot son: girar mucho hacia la izquierda, girar poco hacia la izquierda, no girar, girar poco hacia la derecha o girar mucho hacia la derecha. En todo momento la velocidad lineal permanece constante.

4. Resultados

En un primer momento el robot únicamente fue capaz de seguir el primer tramo recto del circuito de manera oscilatoria. Más tarde logró recorrerlo de manera lineal y recta, pero sin control. Tras modificar el tamaño de la memoria máxima el robot logró recorrer ¾ partes del circuito, tal y como se observa en la *Ilustración 4.*

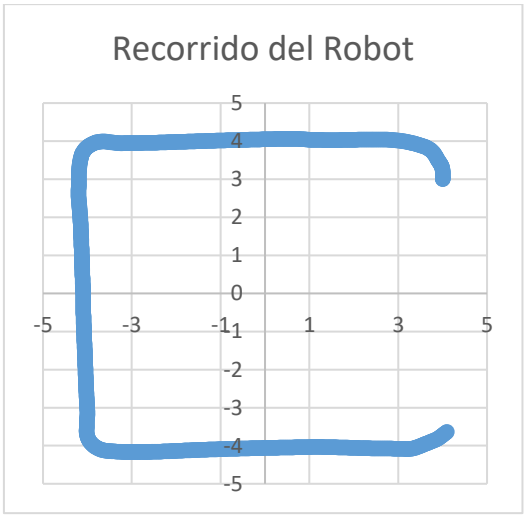

Ilustración 4 Recorrido del Robot en el Trial 296

Finalmente, tal y como se puede observar en la *Ilustración 5*, el robot ha aprendido a recorrer con éxito el circuito adaptándose lo mejor posible a los giros y recorriendo de manera totalmente lineal los tramos rectos.

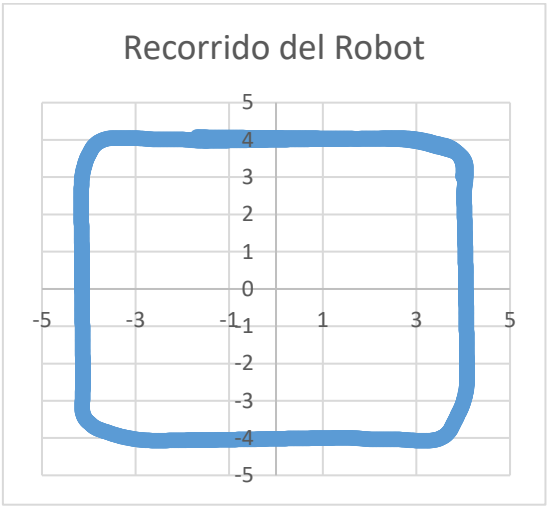

Ilustración 5 Recorrido del Robot en el Trial 472

5. Conclusiones

Los resultados han sido satisfactorios y han estado a la altura de las expectativas, demostrando el control desarrollado mediante aprendizaje por refuerzo ser más efectivo y hábil que el control PD convencional, mejorando la rapidez del robot en recorrer el circuito, reduciendo el tiempo empleado para recorrerlo por completo (en un 25% aproximadamente), y su destreza para hacer los giros pertinentes, tal y como se observa en las gráficas presentadas.

De cara a futuros estudios relacionados con el control de robots móviles mediante algoritmos de aprendizaje por refuerzo se recomienda tener muy en cuenta el tamaño de la memoria máxima del robot para lograr obtener resultados aceptables.

6. Referencias

[1] Robot Pioneer P3DX https://www.generationrobots.com/en/402395-robot-mobile-pioneer-3-dx.html Última visita: 30/08/2020

[2] Curso UCL sobre RL http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html Última visita: 18/10/2019

[3] Aprendizaje por refuerzo con Keras y OpenAI: DQN https://towardsdatascience.com/reinforcement-learning-w-keras-openai-dqns-1eed3a5338c Última visita: 23/06/2019

DEVELOPMENT OF A NAVIGATION SYSTEM FOR A MOBILE ROBOT USING REINFORCEMENT LEARNING

Author: Luque Bayón, Emmanuel.
Supervisor: Boal Martín-Larrauri, Jaime.
Supervisor: López López, Álvaro Jesús.
Collaborating Entity: ICAI – Pontifical University of Comillas

ABSTRACT

In this project, numerous simulation tests have been carried out so that a mobile robot learns to travel a quadrangular circuit without colliding through reinforcement learning. As a result, the robot has learned to run the circuit successfully and without errors.

Keywords: Mobile Robot, Navigation, Artificial Intelligence, Reinforcement Learning, Simulation, CoppeliaSim, Python

1. Introduction

Reinforcement learning is, along with supervised and unsupervised learning, one of the main 'Machine Learning' or 'Machine Learning' techniques. He is currently employed in different areas such as engineering, mathematics, economics, computing, psychology, or neuroscience for a multitude of tasks: optimization of controls, operations research, study of human behavior, machine learning, etc.

The goal of reinforcement learning is to get an agent to learn to perform a task optimally and in the shortest possible time in each environment. To do this, it is necessary to assign objectives, so that the agent will receive a reward for reaching them and a penalty for not achieving it. In this way, the agent will learn to assess what actions should always be taken in order to receive the maximum possible reward in the long term.

2. Project definition

The main reason that has led to the development of this project has been the fact that artificial intelligence, and specifically reinforcement learning, is the future in terms of process optimization and, given the need to optimize control of our virtual robot for the laboratory classes of certain ICAI subjects, it has been decided to test its effectiveness in this project.

In this way, it is possible to find out if with the conventional PD control that was designed in the past for the same circuit we are near or far from the optimal solution that the reinforcement learning algorithm is capable of providing us.

3. Description of the system

The project was initially carried out in the environment of a Linux virtual machine on Windows using the CoppeliaSim simulation platform and programming the code externally in Python. Subsequently, the environment change to Windows was made for greater simplicity.

The general scheme of the project is represented in *Illustration 1*, where the data transfer that takes place between the environment (virtual or V-Rep simulation) and the agent (implemented with Python code).

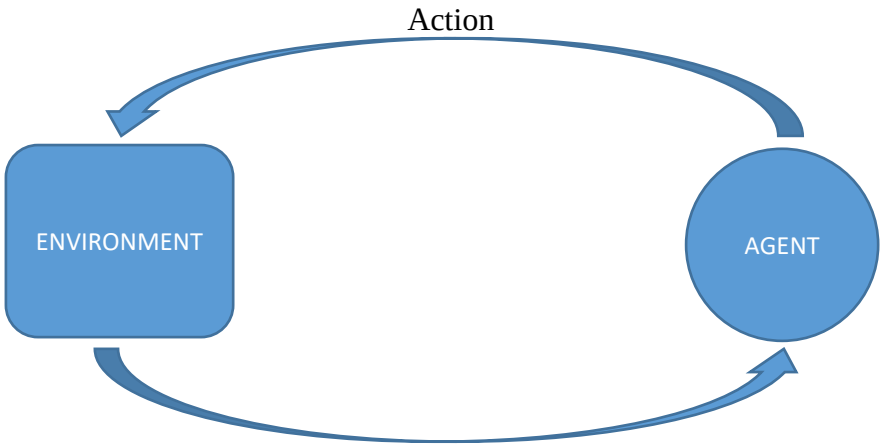

Illustration 1 General System Architecture

For the modelling of the robot policy, a 4-layer neural network designed by Keras with 12 neurons in the input layer has been used, whose neurons in the intermediate layers are arranged according to *Illustration 2*.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 48)                624
_________________________________________________________________
dense_1 (Dense)              (None, 24)                1176
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 125
=================================================================
Total params: 1,925
Trainable params: 1,925
Non-trainable params: 0
```

Illustration 2 Neural Network Structure

The quadrangular circuit that has been used to carry out the tests is represented in *Illustration 3*.

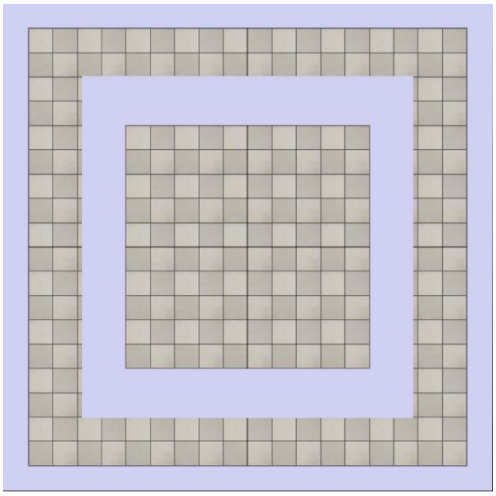

Illustration 3 Quadrangular Circuit

The virtual mobile robot that has been used in this project is the Pioneer P3DX [1].

The actions that the robot can perform are the following: turn a lot to the left, turn a little to the left, do not turn, turn a little to the right or turn a lot to the right. At all times the linear velocity remains constant

4. Results

Firstly, the robot was only able to follow the first straight section of the circuit in an oscillatory way. Later, he managed to follow it in a linear and straight way but without control. After modifying the maximum memory size, the robot managed to follow ¾ parts of the circuit, as shown in *Illustration 4*.

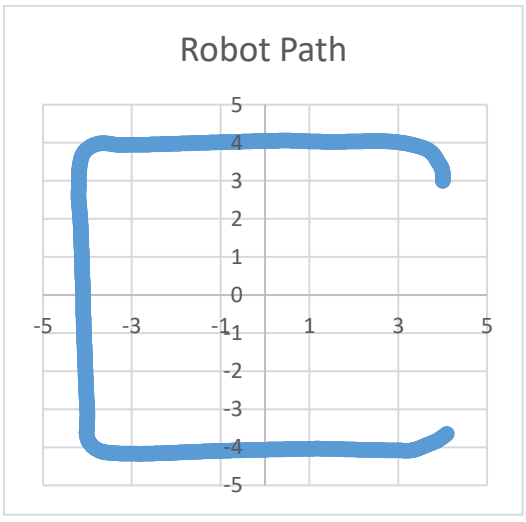

Illustration 4 Robot Path in Trial 296

Finally, as can be seen in *Illustration 5*, the robot has learned to successfully follow the circuit, adapting itself as best as possible to the turns and traveling the straight sections in a completely linear way.

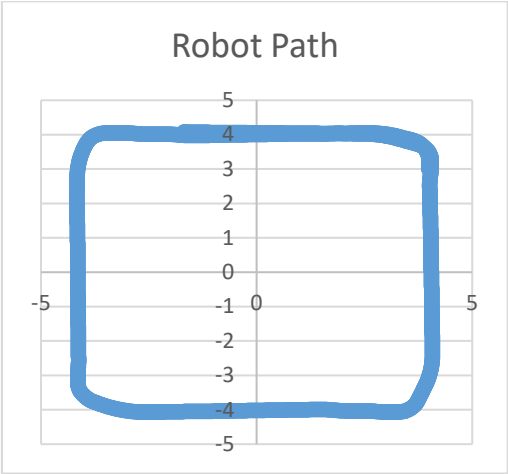

Illustration 5 Robot Path in Trial 472

5. Conclusions

The results have been satisfactory and have lived up to expectations, showing the control developed through reinforcement learning to be more effective and skillful than the conventional PD control, improving the speed of the robot in traveling the circuit, reducing the time used to travel it completely (by about 25%), and their ability to make the appropriate turns, as shown in the graphs presented.

For future studies related to the control of mobile robots using reinforcement learning algorithms, it is strongly recommended to take into account the maximum memory size of the robot to achieve acceptable results.

6. References

[1] Pioneer P3DX Robot https://www.generationrobots.com/en/402395-robot-mobile-pioneer-3-dx.html Last visit: 30/08/2020

[1] UCL Course on RL http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html Last visit: 10/18/2019

[2] Reinforcement learning with Keras and OpenAI: DQN https://towardsdatascience.com/reinforcement-learning-w-keras-openai-dqns-1eed3a5338c Last visit: 06/23/2019

ÍNDICE

Índice de figuras

Índice de tablas

Índice de acrónimos

MDP:	Procesos de Decisión de Markov
POMDP:	Procesos de Decisión de Markov Parcialmente Observables
MC:	Montecarlo
TD:	Diferencia Temporal
GLIE	Codicioso en el Límite con Exploración Infinita
GTSOM:	Mapas Teóricos del Juego Autoorganizados
PD:	Controlador Proporcional-Diferencial
API:	Interfaz de Programación de Aplicaciones
VM:	Máquina Virtual
AI:	Inteligencia Artificial
2D:	Dos Dimensiones
UCL:	*University College* de Londres
RL:	Aprendizaje por Refuerzo
DQN:	Red Q Profunda

CAPÍTULO 1. INTRODUCCIÓN

1.1 Descripción de los términos principales.

Bajo este encabezado se brindará información básica que debe conocer sobre el aprendizaje por refuerzo para comprender mejor el proyecto.

- **Agente**

Sujeto en el proceso de aprendizaje. El objetivo de un proyecto de aprendizaje por refuerzo es que el agente exhiba un comportamiento optimizado.

- **Entorno**

Conjunto de entidades y variables sobre las que el agente no tiene control. Se puede interpretar como el medio en el que el agente se desenvuelve, y con el que interactúa.

- **Estado**

Percepción del agente de la situación en la que se encuentra el entorno, que en procesos de decisión markovianos cambia cada vez que realiza una acción.

Al hablar de estado, hemos de diferenciar dos estados: el estado del entorno y el estado del agente. Por un lado, el estado del entorno es la descripción total y absoluta del entorno, la cual se usa para proporcionar una recompensa al agente y en ocasiones no es conocido en su totalidad por el agente. Mientras que, por otro lado, el estado del agente es la información del entorno que este usa para escoger la acción a realizar, que puede o no coincidir con el estado del entorno.

- **Recompensa**

Evaluación (escalar) de la acción realizada por el agente en el instante anterior.

1.2 Aprendizaje por refuerzo

El aprendizaje por refuerzo es, junto al aprendizaje supervisado y al no supervisado, una de las principales técnicas de *'Machine Learning'* o 'Aprendizaje automático'. Actualmente es empleado en distintas áreas como ingeniería, matemáticas, economía, informática, psicología o neurociencia para multitud de tareas: optimización de controles, investigación operativa, estudio del comportamiento humano, aprendizaje automático, etc [1][2][3][4].

Lo que diferencia el aprendizaje por refuerzo de otras formas de *'Machine Learning'* es la forma en la que aprende a desenvolverse en el entorno el "agente" o sujeto en proceso de aprendizaje: por lo general no se le dan instrucciones previas al agente sino que aprende por sí solo, la valoración del trabajo del agente no es instantánea, sino que se aporta al final de cada iteración, el tiempo de aprendizaje es realmente importante y los datos que recibe el agente del entorno después de realizar cada acción se ven influidos por estas acciones. Gracias al aprendizaje por refuerzo se han logrado diversos objetivos como enseñar a un robot a caminar [5], optimizar el desempeño de los brazos robóticos en las fábricas [6] o vencer con solo 24 horas de aprendizaje a los campeones mundiales de ajedrez y su equivalente asiático, *"shogi"* [7].

El objetivo del aprendizaje por refuerzo es lograr que un agente aprenda a realizar una tarea de manera optimizada y en el menor tiempo posible en un entorno determinado. Para ello es necesario asignar unos objetivos, de manera que el agente recibirá una recompensa al alcanzarlos y una penalización en caso de no lograrlo. De esta manera el agente aprenderá a valorar qué acciones es conveniente realizar en cada momento para lograr recibir la máxima recompensa posible a largo plazo. Es por esto por lo que en ocasiones será necesario sacrificar recompensas inmediatas para obtener después mayores recompensas a largo plazo.

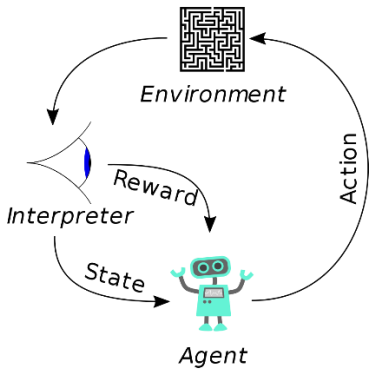

Figura 1.1 Estructura general del Aprendizaje por Refuerzo
[Aprendizaje por refuerzo https://www.wikiwand.com/en/Reinforcement_learning 10/05/2020]

La información sobre el aprendizaje por refuerzo se obtuvo de una serie de conferencias grabadas en video en 2015 en las que David Silver, profesor de ciencias de la computación en el *University College* de Londres, explica qué es el aprendizaje por refuerzo. Junto con

las conferencias grabadas en video, los PDF explicativos que utiliza como soporte para sus clases también están disponibles en la web.

El enlace a la página de Internet donde se encuentra el material descrito a continuación es [8].

En todo aprendizaje por refuerzo existen dos factores básicos que interactúan: el agente y su entorno, lo cual recuerda a la famosa frase "yo soy yo y mi circunstancia" de Ortega y Gasset [9]. En cada paso del proceso de aprendizaje o instante 't', el agente comienza realizando una acción en base a su conocimiento del entorno y sus objetivos; a continuación, recibe una recompensa por ello y, debido a la acción realizada, su percepción del entorno o estado cambia. Entonces, al modificarse su estado debido a la acción realizada, se verá obligado a escoger de nuevo la acción que considere que podrá aportarle un mayor beneficio en el futuro, modificando de nuevo su estado y recibiendo de nuevo una recompensa. El proceso se repetirá hasta alcanzar el objetivo principal para el que fue diseñado o realizar una acción que lo penalice de manera que deba volver a empezar desde el estado inicial. En caso de que el agente haya alcanzado su principal objetivo volverá mismamente al estado inicial para seguir perfeccionando su actuación a lo largo de todo el proceso.

Para poder aplicar el aprendizaje por refuerzo debemos contar con un entorno en el que las acciones del agente tengan como consecuencia el cambio de estado del mismo. Si las acciones del agente no son capaces de modificar su estado en el entorno no tiene sentido emplear aprendizaje por refuerzo.

La historia es la secuencia de observaciones, acciones y recompensas. El estado es función de la historia del agente y determina qué sucederá en el futuro. Cuando el estado futuro no depende del pasado, sino que es solo función del presente se le llama 'Estado de Markov'. Al hablar de estado, hemos de diferenciar dos estados: el estado del entorno y el estado del agente. Por un lado, el estado del entorno es la descripción total y absoluta del entorno, la cual se usa para proporcionar una recompensa al agente y en ocasiones no es conocido en su totalidad por el agente. Mientras que, por otro lado, el estado del agente es la información del entorno que este usa para escoger la acción a realizar, que puede o no coincidir con el estado del entorno.

Por tanto, siempre podremos tener dos tipos distintos de problemas de aprendizaje por refuerzo: un problema en el que el estado del agente coincida con el del entorno, conociendo por tanto el agente todo el entorno al completo, el cual es conocido como "Problema de entornos totalmente observables" o "Procesos de decisión de Markov" (MDP), y otro tipo de problema en el que el estado del agente no coincide con el del entorno, no conociendo el agente por tanto el estado del entorno al completo, y se conocen como "Procesos de decisión de Markov parcialmente observables" o POMDP.

Es preciso observar que, a grandes rasgos, en todo problema de aprendizaje por refuerzo el agente puede incluir uno o varios de estos tres componentes: la política, la función valor y el modelo.

La política es la aplicación que transforma estados en probabilidades de seleccionar las distintas acciones disponibles. En los MDPs, se define como problema de control a la búsqueda de la política óptima, que permita al agente alcanzar sus objetivos y evitar las penalizaciones. Existen dos tipos de políticas: las políticas deterministas, según las cuales en cada estado el agente realiza únicamente una acción determinada, y las políticas estocásticas, según las cuales cada estado del agente lleva asociado una probabilidad diferente para cada acción posible a realizar.

La función valor se define como la esperanza de la suma ponderada de las recompensas futuras desde cada estado, siguiendo a partir del mismo una determinada política y por tanto indica la conveniencia o no del estado en cuestión. De esta forma el agente puede tomar decisiones, elegir el estado futuro que le sea más conveniente y actuar en consecuencia. Más adelante se describe esta función, también conocida como "Valor del estado" o V(s).

El modelo se forma a partir de la información sobre el entorno con la que cuenta el agente y por tanto le sirve al agente para anticiparse a los cambios en el entorno. El modelo suele irse perfeccionando a medida que el agente va obteniendo más información sobre el entorno. De esta manera, el agente obtiene dos funciones que caracterizan al modelo: la función 'P', que predice el siguiente estado, y la función 'R', que predice cuál será la próxima recompensa inmediata; las dos funciones condicionadas a la acción que decida realizar el agente en el estado actual en que se encuentra.

En los problemas en los que el agente no cuenta con un modelo de partida puede construirse uno para sí mismo al comienzo del aprendizaje interactuando con el entorno para después desarrollar su política, aunque no es necesario. Si el agente cuenta de partida con un modelo del entorno todo será más sencillo.

De esta manera, podemos realizar varias clasificaciones de nuestro agente. A la hora de tomar decisiones nuestro agente puede basar su criterio en la política, en la función valor o en ambas. En este último caso el agente se conoce como 'Actor crítico'. Por otro lado, también podemos clasificar al agente según si usa un modelo del entorno o no.

El aprendizaje por refuerzo es básicamente un aprendizaje a base de prueba y error. El problema más común en aprendizaje por refuerzo es el problema exploración-explotación, debido a que el agente debe encontrar un balance óptimo entre recorrer de nuevo los caminos que ya le han dado un resultado positivo en el pasado (explotación) y probar nuevas técnicas inexploradas que puedan llegar a ser más eficientes que las ya conocidas (exploración). El objetivo fundamental del agente es encontrar la mejor política en base a su experiencia teniendo cuidado de no perder demasiadas recompensas y mantener niveles de eficiencia aceptables. Tanto la exploración como la explotación son beneficiosas y deben estar presentes ambas en todo proceso de aprendizaje por refuerzo. Es común que al comienzo del aprendizaje tenga lugar la exploración en mayor medida y a medida que evoluciona el agente adquiera la explotación cada vez más importancia.

En todo aprendizaje por refuerzo contaremos con dos funciones básicas que nos ayudarán a tomar decisiones mediante la asignación de un valor determinado a los distintos estados y a las posibles acciones a realizar en cada estado. Estas dos funciones son la función V(s) o función "Valor del estado", la cual asigna un valor a cada estado, y la función Q(s,a) o función "Valor de la acción", la cual asigna un valor a cada acción posible a realizar en cada estado. La función V(s) es la recompensa que cabe esperar al comenzar en el estado 's' y seguir la política 'π' a partir de ahí. La función Q(s,a) es la recompensa que cabe esperar al comenzar en el estado 's', realizar la acción 'a' en ese estado y luego seguir la política 'π' a partir de ahí. Por tanto, podemos afirmar que los estados adquieren mayor valor en la medida en que nos permiten tomar mejores decisiones y las decisiones estarán más valoradas en la medida en que nos permitan alcanzar mejores estados. La función "Valor del estado" V(s) es la siguiente:

$$V^{\pi}(s) = E[R|s, \pi]$$

La función "Valor de la acción" Q(s,a) es la siguiente:

$$Q^{\pi}(s, a) = E[R|s, a, \pi]$$

El objetivo del aprendizaje por refuerzo es encontrar la política que maximice estas dos funciones en todos los estados y para todas las acciones, es decir, que lleve al agente por el camino óptimo para conseguir la máxima recompensa posible. Estas funciones óptimas siempre existirán. A veces es posible que existan varias soluciones óptimas para el mismo problema.

La ecuación de Bellman es la ecuación que relaciona el valor de los estados con el valor de las acciones posibles en cada estado (es decir, V con Q). La solución óptima de los MDP se obtiene maximizando esta ecuación, pero al no ser una ecuación lineal no hay una forma directa de hacerlo, por lo que se suele resolver por iteración. Para resolverla contamos con varios métodos iterativos posibles: Iteración de valores, Iteración de políticas, *Q-learning* y Sarsa. Estos métodos son útiles para resolver MDPs totalmente observables, sin embargo, para resolver MDPs infinitas y continuas, MDPs parcialmente observables, o MDPs sin descuento con recompensa media necesitaremos otras técnicas distintas.

Comúnmente podemos resolver dos tipos de problemas distintos de aprendizaje por refuerzo: predicción y control. Una manera básica de resolverlos es hacerlo mediante algoritmos de programación dinámica síncrona, en los cuales todos los estados se actualizan en paralelo. En los problemas de predicción se busca evaluar una política aleatoria de partida con el objetivo de hacernos una idea de cómo será la política óptima. Por ello en ocasiones en los problemas de predicción lograremos encontrar la política óptima pero no es lo común. La ecuación que se usa en este tipo de problemas es la Ecuación de Expectativas de Bellman y el algoritmo consiste en la evaluación iterativa de la política escogida. En cambio, en los problemas de control lo que se busca es hallar la política óptima.

La primera opción que tenemos para hallar esta política óptima en los problemas de control es mediante el algoritmo de Iteración de la Política y la Ecuación de Expectativas de Bellman junto a la ecuación de la Mejora de la Política Codiciosa. Este método se puede entender como la resolución iterativa de un problema de predicción, de manera que primero evaluaremos una política aleatoria determinada y una vez evaluada, los resultados de la evaluación nos permitirán escoger una política de partida mejor que la que usamos inicialmente para de nuevo volver a evaluar esta política y obtener así de nuevo unos resultados distintos y optimizados. Este proceso deberá repetirse hasta que encontremos la política óptima. Por este método siempre seremos capaces de encontrar la política óptima y cada paso ya nos brindará información útil sobre la solución final del mismo.

El segundo método que suele usarse para resolver problemas de control es el algoritmo de la Iteración del Valor y la Ecuación de Optimización de Bellman. Este método consiste en la iteración de la Ecuación de Optimización de Bellman y se usa cuando tenemos situado exactamente el objetivo final de nuestro problema. Es un método que nos permitirá hallar el camino más corto desde la situación inicial a la situación objetivo y por lo general nos permitirá encontrar más rápidamente la solución de nuestro problema que con el primer método. El único inconveniente de usar este método es que por lo general podremos extraer mucha menos información de los pasos intermedios con este segundo método que con el primero.

Estos algoritmos mencionados anteriormente son algoritmos de programación dinámica síncrona, pero también se podrían usar algoritmos de programación dinámica asíncrona para resolver estos problemas. Estos últimos algoritmos son muy útiles para resolver un problema en el que tengamos una cantidad desmesurada de estados a actualizar y nos demos cuenta de que podríamos resolver el problema más eficazmente y en menos tiempo si en vez de actualizar todos los estados a la vez, actualizamos solo aquellos estados que sepamos que van a sufrir más cambios en su función valor a lo largo del proceso de optimización y por tanto vayan a ser más relevantes en nuestro problema. Mediante este tipo de algoritmos de programación dinámica asíncrona los estados son actualizados individualmente, en cualquier orden, en vez de en paralelo.

Los métodos mencionados hasta el momento eran útiles únicamente cuando poseemos un modelo del entorno, es decir, cuando conocemos su dinámica y gracias a ella somos capaces de establecer un sistema de transiciones y recompensas, obteniendo finalmente la función valor. Los casos en los que no conocemos la dinámica del entorno se conocen como “Aprendizaje por refuerzo sin modelo” y en este caso se nos pueden presentar dos tipos de problemas de nuevo: predicción y control. Para resolver los problemas de predicción podemos emplear dos algoritmos: el método de Monte-Carlo (MC) y el método de la Diferencia Temporal (TD).

El método de Monte-Carlo (MC) es un método empleado para resolver problemas de predicción episódicos, es decir, problemas en situaciones que siempre terminan. Se basa en una idea muy simple: la estimación de los valores de los estados se realiza mediante prueba y error. Como estamos en un problema de predicción, el objetivo será estimar cuál será la función valor $V\pi$ resultante de aplicar una política determinada que se seguirá en todo momento. Al final de cada episodio aplicando la misma política, el agente recibirá una recompensa total. El promedio de todas estas recompensas totales recibidas al finalizar cada episodio será el valor que tomará $V\pi$ en nuestro caso. Cuantos más intentos episódicos se realicen, más estaremos seguros de la fiabilidad de nuestro valor $V\pi$. Hay dos variantes de este método: el primer método o método de la primera visita de MC consiste en incrementar el contador para el promedio únicamente la primera vez que el agente pase por el estado 't' en un episodio; sin embargo, el segundo método o método de todas las visitas de MC incrementa el contador siempre que el agente pase por el mismo estado 't' aunque ya haya pasado por allí en ese mismo episodio.

El método de la Diferencia Temporal (TD) es un método empleado para resolver problemas de predicción en los cuales el algoritmo aprende a medida que se desarrolla el episodio y no únicamente al final, como pasaba en el método de MC. Por tanto, puede emplearse también en entornos continuos y no solo episódicos. Esta característica hace que este nuevo método presente una ventaja competitiva importante respecto al método de MC. En este nuevo método el valor de $V\pi$ se actualiza no a través de la experiencia del agente sino a través de estimaciones que realiza el agente en cada estado acerca del siguiente. Estas estimaciones del siguiente estado se conocen como 'Objetivo TD'.

Existen ventajas y desventajas de usar cada uno de estos dos métodos: el método TD puede aprender sin conocer el resultado final de cada episodio mientras que el MC no; el método MC tiene una alta varianza, lo cual es perjudicial, pero sin embargo es insesgado lo cual siempre es positivo, mientras que el método TD tiene menos varianza, pero sin embargo tiene algo de sesgo. Además, el método MC es fácil de entender y usar y tiene buenas propiedades de convergencia, además de no ser muy sensible al valor inicial, mientras que el método TD suele ser más eficiente que el método MC y es más sensible al valor inicial. El método MC converge hacia la solución con menor error cuadrático medio y el método TD converge hacia la solución que tenga un mayor parecido con el Modelo de Markov. Por todo esto, podemos concluir que el método TD es más eficiente en entornos de Markov pero el método MC es más eficiente en entornos que no son de Markov.

Existe un tercer método de resolución de problemas de aprendizaje a medio camino del método MC y el TD llamado método TD(λ), el cual es similar al método TD pero en lugar de estimar únicamente el valor del siguiente estado lo que hace es estimar el valor de dos, tres, cuatro, etc estados según prefiera el usuario y emplea el valor de λ (entre 0 y 1) para dar más importancia a las estimaciones de estados cercanos que a las de los estados más alejados.

Por otro lado, podemos emplear el método TD para realizar lo que se llama una 'Vista hacia delante' o una 'Vista hacia detrás'. Al hacer la 'Vista hacia delante' lo que hacemos es realizar la estimación de todos los estados hasta el final del episodio, y esto sería similar a lo que obtendríamos con el método MC para $\lambda = 1$. Para $\lambda=0$ lo que obtendríamos será similar al algoritmo TD. En la 'Vista hacia detrás' lo que haremos será emplear la función de las trazas de elegilibilidad o E(s) para dar mayor importancia a las modificaciones de $V\pi$ provocadas por las estimaciones de los estados más recientes o que más se repitan en nuestro episodio.

Por otro lado, para resolver los problemas de control cuando no disponemos de un modelo del entorno o cuando disponemos de un modelo pero es demasiado grande para usarlo podemos hacerlo de dos maneras: mediante aprendizaje en política o aprendizaje fuera de política. El aprendizaje en política consiste en aprender del mismo trabajo que se realiza, aprendiendo acerca de la política a partir de la experiencia muestreada a partir de la misma. El aprendizaje fuera de política o *off-policy* consiste en aprender de un trabajo ya realizado previamente, aprendiendo acerca de la política a partir de la experiencia muestreada a partir de otra política distinta.

Dentro del aprendizaje en política volvemos a encontrarnos el método de Monte-Carlo (MC) y el método de la Diferencia Temporal (TD). Una primera forma de resolver estos problemas sería mediante el método de la Iteración de la Política Generalizada empleando la Función Acción-Valor con el método de Monte-Carlo, sin embargo, este método de la Política Generalizada no nos garantiza que alcancemos la política óptima ya que no es apto para explorar debido a que hay acciones cuya probabilidad de ser realizadas por el agente es igual a cero. Es por ello que se suele usar comúnmente un método de exploración llamado *ε-Greedy* según el cual con una probabilidad de 1- ε se elegirá la acción de mayor valor en cada estado y con una probabilidad de ε se elegirá otra acción totalmente aleatoria. Para asegurarnos de que mediante este método hemos hallado la política óptima debemos asegurarnos de que las condiciones *'Greedy in the Limit with Infinite Exploration'* (GLIE) se cumplan, las cuales son en primer lugar que todos los pares estado-acción hayan sido explorados un número infinito de veces y que la política converja hacia una política determinista.

De nuevo, el método de la Diferencia Temporal (TD) resulta más efectivo para la resolución de este tipo de problemas ya que cuenta con numerosas ventajas: tiene menor varianza, se actualiza en línea y en secuencias incompletas. Al algoritmo de diferencia temporal en este tipo de problemas de entorno desconocido se le conoce como "Sarsa". Este algoritmo consiste en lo siguiente: se comienza inicializando Q(s,a) para todos los estados y todas las acciones arbitrariamente y teniendo en cuenta que la Q del estado terminal para todas las acciones vale 0. En cada episodio se inicializará el estado 'S', se escogerá una acción 'A' del estado 'S' usando una política derivada de Q (*ε-Greedy* por ejemplo) y entonces se repetirá en cada paso del episodio lo siguiente: primero elegimos la acción 'A', observamos la recompensa y el nuevo estado (S′) en el que nos

encontramos; entonces elegimos la acción (A´) del estado (S´) en que nos encontramos usando la política derivada de Q (*ε-Greedy* por ejemplo); a continuación,

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R + \gamma Q(S', A') - Q(S, A)]$$

$$S \leftarrow S'; A \leftarrow A'$$

hasta que S sea un estado terminal. Igual que en los problemas de predicción, Sarsa también se puede usar con un enfoque de Vista hacia delante o de Vista hacia atrás.

En cuanto a las técnicas de aprendizaje *off-policy*, si nos centramos en enfoques que infieren la política a partir de la evaluación de funciones de valor, quizás el método de referencia en la actualidad es el de *Q-Learning*. En este algoritmo la política objetivo es determinista hacia el máximo valor de la acción que se puede realizar desde cada estado, pero la política de comportamiento es *ε-Greedy*, la cual permite la exploración. En este caso, los resultados de la nueva política tienen mucha relación con la información obtenida hasta el momento en los episodios anteriores de interacción con el entorno. Los pasos del algoritmo son los siguientes: en primer lugar, se inicializa Q(s,a) para todos los estados y todas las acciones arbitrariamente y teniendo en cuenta que la Q del estado terminal para todas las acciones vale 0. En cada episodio se inicializará el estado 'S' y entonces se repetirá en cada paso del episodio lo siguiente: se escogerá una acción 'A' del estado 'S' usando una política derivada de Q (*ε-Greedy* por ejemplo); entonces elegimos la acción 'A', observamos la recompensa y el nuevo estado (S´) en el que nos encontramos; a continuación y hasta que S sea un estado terminal,

$$Q(S, A) \leftarrow Q(S, A) + \alpha [R\,\gamma + \gamma \max_a Q(S', A) - Q(S, A)]$$

$$S \leftarrow S''$$

1.3 Estado de la cuestión

Tras realizarse un estudio a fondo del estado de la cuestión, se consideraron los siguientes artículos como los más representativos.

Tabla 1. *Artículos revisados*

Referencia	**Aporte**	**Algoritmo toma de decisión**	**Método**
[10] Zuo, et al. (2014)	Objetivo: salir de un laberinto en espiral Acciones: Acción F: Seguir hacia delante 100 mm Acción L: Girar a la izquierda 15º Acción R: Girar a la derecha 15º Estados: (1 2 3) // 1 - Cerca del objeto // 2 - Intermedio // 3 - Lejos del objeto Recompensa: (0 1 5) – (L R F) para el estado anterior Sensores: (izquierda derecha recto)	[10]	*Q-learning*
[11] Altuntas, et al. (2016)	Se compara algoritmo Sarsa y *Q-learning*.	*ε-greedy*	Sarsa y *Q-learning*

[12] Khriji, et al. (2011)	- Navegación basada en **lógica difusa**. Recompensa: Distancia == Near, entonces r=60000 Si puede elegir la mejor acción y la elige entonces r=1500 Si no la elige, entonces r=-1500 Si no puede elegir la acción entonces r=0 Parámetros: alfa=0.2 gamma=0.8 Sensores: *(Ver figura 1)*	*ε-greedy*	*Q-learning*
[13] Yen, Hickey (2003)	Dos importantes avances: **1)Aprendizaje Jerárquico** **2) Aprendizaje con caracterización básica del entorno**	*ε-greedy*	*Q-learning*
[14] Zhuang (2005)	Ajuste adaptativo del factor de aprendizaje. Solo para entornos discretos, no continuos.	[14]	Sarsa
[15] Gökçe, Akın	Mejorar el aprendizaje del agente transfiriéndole entre etapas únicamente la parte que le interesa.	*ε-greedy*	*Q-learning*
[16] Truong, Trung (2018)	Navegación del robot con mayor precaución en entornos habitados.	*ε-greedy*	Aprendizaje por refuerzo profundo (basado en *Q-learning*)

[17] Menegaz, Engel (2009)	Se emplea la red GTSOM para la navegación del robot.	*ε-greedy*	*Q-learning*
[18] Tamiselvi, et al. (2011)	*Ver figura 2.*	*ε-greedy*	*Q-learning*
[19] Xiaoyun, et al. (2018)	*Q-learning* modificado en el cual se combina el algoritmo de toma de decisiones *ε-greedy* y el algoritmo de exploración de Boltzmann.	*ε-greedy* y exploración de Boltzmann (combinados)	*Q-learning*
[20] Jawad, Ihsan (2013)	Asignación retroactiva de las recompensas.	Exploración de Boltzmann	*Q-learning*

Principales algoritmos empleados en los artículos descritos en el estado del arte:

- Algoritmo de exploración de Boltzmann: consiste en que el agente a la hora de escoger qué acción realizar en un estado determinado, tomará una decisión aleatoria y cada acción verá ponderada su probabilidad de ser escogida en función del su valor Q(s, a), de manera que será más probable elegir las acciones más beneficiosas hasta el momento [20].
- *ε-greedy*: este algoritmo de toma de decisión es un caso particular de la familia de políticas conocidas como *"ε-soft"* y en él el agente realizará de manera aleatoria una de estas dos acciones: la primera consistirá en escoger aquella acción que haya dado mejor resultado hasta el momento, lo cual hará con probabilidad 1-ε, y la segunda en escoger una acción al azar entre todas las posibles [18].
- *Q-learning*: es el método más conocido en cuanto a las técnicas de aprendizaje fuera de política. En este algoritmo la política objetivo es determinista hacia el máximo valor de la acción que se puede realizar desde cada estado, pero la política de comportamiento es *ε-greedy*, la cual posibilita un adecuado equilibrio entre exploración y explotación. En este caso, los resultados de la nueva política tienen mucha relación con la información obtenida hasta el momento en los episodios anteriores de interacción con el entorno. Los pasos del algoritmo son los siguientes: en primer lugar, se inicializa Q(s,a) para todos los estados y todas las acciones arbitrariamente y teniendo en cuenta que la Q del estado terminal para todas las acciones vale 0. En cada episodio se inicializará el estado 'S' y entonces se repetirá en cada paso del episodio lo siguiente: se escogerá una acción 'A' del estado 'S' usando una política derivada de Q (*ε-greedy* por ejemplo); entonces elegimos la acción 'A', observamos la recompensa 'R' y el nuevo estado (S´) en el que nos encontramos; a continuación y hasta que S sea un estado terminal,

$$Q(S,A) \leftarrow (1- \alpha)\, Q(S,A) + \alpha\, [R + \gamma \max_a Q(S',A) - Q(S,A)]$$
$$S \leftarrow S'$$

  Siendo:
  - '$\max_a$' la máxima recompensa que es posible obtener al realizar cualquier acción en el nuevo estado
  - 'α' el índice de aprendizaje el cual se encuentra comprendido entre 0 y 1 e indica el ritmo de aprendizaje del agente, es decir, la relevancia que tendrán los nuevos descubrimientos frente a las conclusiones obtenidas hasta el momento. Es por ello por lo que se entiende que en las primeras fases de aprendizaje del agente este factor es cercano a 0.9 y poco a poco se va reduciendo.
  - 'γ' el factor de descuento, el cual siempre es menor que la unidad y cuanto más pequeño sea indicará que el agente dará cada vez menos importancia a las recompensas que es posible obtener en el futuro para valorar la acción a realizar en el estado en que se encuentra. Es una medida de la 'confianza' del agente respecto al futuro [10].

- Sarsa: este algoritmo consiste en lo siguiente: se comienza inicializando Q(s,a) para todos los estados y todas las acciones arbitrariamente y teniendo en cuenta

que la Q del estado terminal para todas las acciones vale 0. En cada episodio se inicializará el estado 'S', se escogerá una acción 'A' del estado 'S' usando una política derivada de Q (*ε-Greedy* por ejemplo)) y entonces se repetirá en cada paso del episodio lo siguiente: primero elegimos la acción 'A', observamos la recompensa y el nuevo estado (S´) en el que nos encontramos; entonces elegimos la acción (A´) del estado (S´) en que nos encontramos usando la política derivada de Q (*ε-Greedy* por ejemplo); a continuación,

$$Q(S,A) \leftarrow Q(S,A)+\alpha[R+\gamma Q(S',A')-Q(S,A)]$$
$$S \leftarrow S'; A \leftarrow A'$$

Siendo los parámetros los mismos que en el algoritmo de *Q-learning* [14].

Como conclusiones del estado del arte podemos observar que la mayoría de proyectos de navegación inteligente de robots utilizan como método el *Q-learning* [10][11][12][13][15][16][17][18][19][20], usando alguno también Sarsa [11][14], y como algoritmo de toma de decisión el *ε-greedy* [11][12][13][15][16][17][18], aunque estudios más recientes confirman que la mejor forma de implementar el algoritmo es mediante una combinación del algoritmo de exploración de Boltzmann y el *ε-greedy* [19].

Por otro lado, hay artículos que aportan visiones mucho más innovadoras de lo normal. En el artículo [12] la navegación se basa en lógica difusa cuyo objetivo es el siguiente: se quiere que el robot se comporte de manera diferente en cada situación distinta. Por tanto, habrá una función que asigne a cada estado del robot un comportamiento distinto (*goal reaching, obstacle avoidance, wall following, emergency situation)*. Los distintos estados se configuran a partir de 3 variables:

1) Distancia entre el robot y el objetivo.
2) Distancia entre el robot y el obstáculo.
3) Ángulo error entre la dirección del objetivo y la del robot.

En [13] se hace mención a dos importantes avances: aprendizaje jerárquico, en el cual hay dos agentes: primario y secundario. El agente primario calcula la ruta al objetivo y busca seguirla. Además, el agente secundario esquiva los obstáculos por el camino que marca el primario y la información que obtenga podrá usarse para evitar obstáculos en otros entornos también. Por otro lado, se habla de aprendizaje con caracterización básica del entorno, lo cual consiste en que del entorno se obtienen solo ciertas características en grupos de manera que en función de la característica obtenida de cada grupo se le asigna una recompensa al agente por cada característica de cada grupo obtenida del entorno. Esto convierte al algoritmo en mucho más adaptable a nuevos entornos y moldeable a nuevas situaciones.

En [14] se realiza un ajuste adaptativo del factor de aprendizaje. En la etapa inicial, se construye una estrategia inicial. En la segunda, la estrategia está optimizada y su rendimiento mejora rápidamente. En la fase final, la estrategia de control se aproxima a la estrategia óptima y su rendimiento se estabiliza. Para cada etapa de aprendizaje, el

aprendizaje es diferente y puede conducir a un rendimiento de aprendizaje diferente. Se requieren, por tanto, diferentes ritmos de aprendizaje en diferentes etapas de aprendizaje. Por lo tanto, es razonable ajustar el ritmo de aprendizaje de acuerdo con la entropía de la estrategia.

El objetivo principal de [15] es mejorar el ritmo de aprendizaje del agente mediante la transferencia de las partes relevantes de los conocimientos adquiridos como resultado de experiencias anteriores. La principal contribución de este estudio es fusionar estos dos enfoques para transferir sólo el conocimiento relevante en un entorno.

Finalmente, en [20] solo se asigna una recompensa positiva cuando el agente alcanza el estado objetivo, por tanto, todas las transiciones de estado que no sean las de la transición al estado objetivo o la transición a un estado fallido reciben una recompensa nula, almacenando los valores de estado, acción y recompensa de cada transición del estado inicial al estado objetivo y luego repitiendo o retro propagando esta trayectoria de estado-acción en el orden inverso al final del episodio. Toda la sucesión de estado-acción de esa trayectoria se actualizará automáticamente.

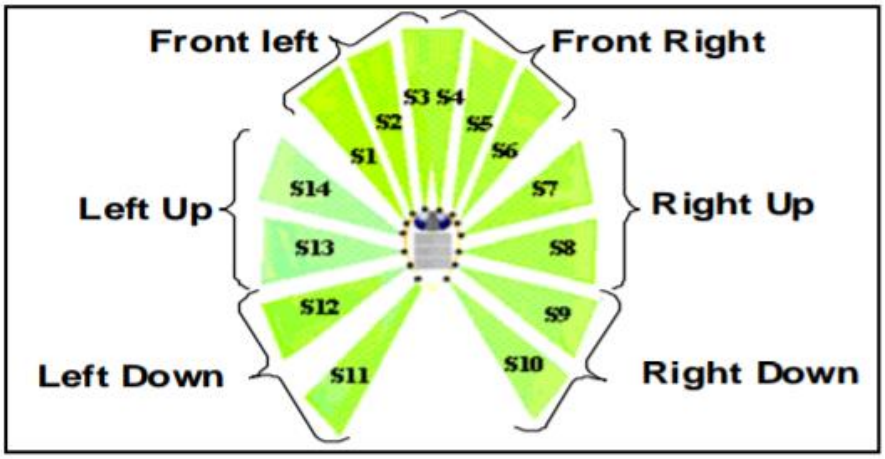

Figura 1.2 Configuración de los sensores del robot [12]

Debe considerarse que la configuración de sensores del robot que aparece en la *Figura 1.2* es muy similar a la configuración de sensores del robot que se utilizará en el proyecto. La diferencia entre la configuración de estos sensores es que en el proyecto el robot tiene 16 sensores y en la *Figura 1.2* solo hay 14 sensores. Sin embargo, es muy fácil hacerse una idea de la configuración real utilizada en el proyecto solo observando la *Figura 1.2*.

También es relevante notar que 10 de estas 16 mediciones de sensores ultrasónicos serán muy importantes en el proyecto porque serán algunas de las 12 entradas de la red neuronal que funcionarán como el "cerebro" del robot, ayudándolo a elegir la mejor acción dependiendo de su estado. Las otras dos entradas restantes de la red neuronal serán las dos últimas velocidades aplicadas a las ruedas izquierda y derecha.

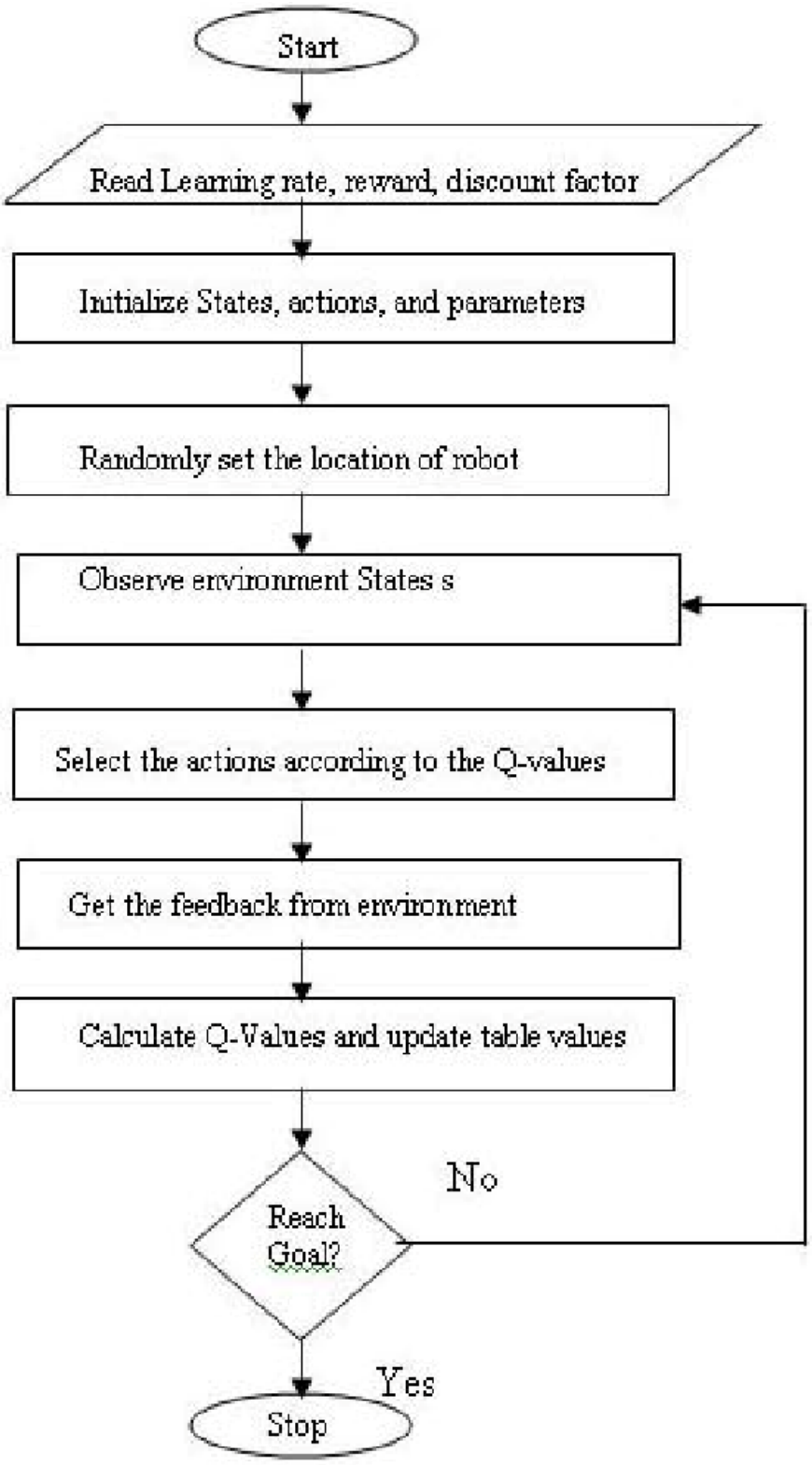

Figura 1.3 Algoritmo Q-learning [18]

1.4 Motivación

El motivo principal que ha dado lugar al desarrollo de este proyecto ha sido el hecho de que la inteligencia artificial, y en concreto el aprendizaje por refuerzo, se ha convertido en los últimos tiempos en una tecnología innovadora y revolucionaria en multitud de campos. Hoy en día observamos la creciente proliferación de robots en las empresas y debido a la multitud de entornos cambiantes que han surgido, entre otros motivos, por la exigente personalización de los productos que cada vez más clientes solicitan, nos damos cuenta de que ya no va a ser posible programarlos uno a uno para realizar tareas concretas, sino que será necesario que ellos mismos aprendan a adaptarse a las distintas situaciones que puedan tener lugar. Ya comenzamos a ver los primeros coches autónomos por las carreteras y los primeros robots que aprenden a caminar por sí mismos y a coger objetos por sí solos en las fábricas de montaje. No pasará mucho tiempo hasta que veamos la implementación de esta tecnología en numerosos campos hoy en día inimaginables. Consideramos que esta tecnología es el futuro en cuanto a la optimización de procesos y, ante la necesidad de lograr optimizar el control de nuestro robot virtual para las clases de laboratorio de ciertas asignaturas de ICAI, se ha decidido probar su eficacia en este proyecto.

1.5 Objetivos

El objetivo de este trabajo ha sido, en primer lugar, la optimización del control de un robot virtual en un entorno simulado de manera que hemos sido capaces de averiguar si el aprendizaje por refuerzo nos conduce, en nuestro caso de estudio, a una solución más optimizada que la que fuimos capaces de aportar mediante el control que diseñamos a mano.

El primer paso que se persigue a la hora de encontrar el control óptimo es el hecho de lograr obtener un control aceptable, es decir, que en nuestro caso de estudio el robot aprenda a recorrer el circuito completo de manera eficaz sin chocarse. Por tanto, este es el primer y básico objetivo que se persigue en este proyecto.

Por otro lado, conocer cuál es la solución óptima para el robot a la hora de recorrer el circuito nos ha ayudado a valorar la solución que aportaba nuestro algoritmo de control convencional y poder así conocer si estamos cerca o lejos del control óptimo del robot virtual.

CAPÍTULO 2. ARQUITECTURA DEL SISTEMA

Como se describió anteriormente en la introducción del proyecto, el objetivo que se persigue con este trabajo es el desarrollo de un algoritmo de control de un robot móvil mediante aprendizaje por refuerzo para lograr un control óptimo del mismo de manera que el robot sea capaz de recorrer el circuito sin chocarse en el menor tiempo posible y así poder compararlo con el control del robot que se podría llevar a cabo con un control convencional como el PD. Esto se realiza así teniendo en consideración que un control convencional como el PD es capaz de aportar una solución al problema, pero en teoría el algoritmo de control desarrollado mediante aprendizaje por refuerzo es capaz de aportarnos la solución óptima para este problema, y eso es lo que queremos comprobar.

En primer lugar, la primera decisión a tomar fue la del entorno de simulación a utilizar y se escogió CoppeliaSim [21] por ser este un entorno con físicas realistas, ampliamente utilizado en proyectos de robótica de todo tipo y suficientemente asequible para una primera toma de contacto con un entorno de simulación de estas características. Una vez escogido el entorno de simulación se procedió al diseño de los circuitos en los cuales debería moverse el robot y estos fueron diseñados a mano. Se construyeron 3 circuitos: un primer circuito con forma cuadrangular de nivel básico, un segundo circuito de nivel avanzado en forma de espiral, y un tercer circuito de nivel intermedio en forma ovalada el cual fue diseñado a imagen del circuito presente en el laboratorio de control de ICAI para probar la eficacia del algoritmo en un circuito real en un futuro. El robot que se decidió utilizar fue el robot Pioneer P3DX debido a que es un robot que se ajusta a las necesidades de nuestro proyecto pues se trata de un robot móvil con 3 ruedas: una derecha, una izquierda y una centrada en su parte posterior, y 2 motores: derecho e izquierdo, el cual dispone de 16 sensores de proximidad que constituyen un elemento esencial para el desarrollo de un algoritmo de control como el que se desarrolla en este proyecto.

Tras concretar estos aspectos previos, se pensó en utilizar las herramientas de entrenamiento de algoritmos de aprendizaje por refuerzo de Open-AI Gym [22], desarrolladas por Google y de código abierto. Con estas mismas herramientas se logró entrenar en el pasado algoritmos de aprendizaje por refuerzo que fueron capaces de dominar todo tipo de juegos de Atari y demás entornos [22]. El lenguaje de programación escogido fue Python, instalado con Miniconda [23], debido a que es el único lenguaje compatible con Open-AI Gym por el momento. Para lograr enlazar estas herramientas de Open-AI Gym con nuestro entorno de simulación de CoppeliaSim fue necesario emplear la biblioteca de GitHub llamada Pyrep [24]. Aunque CoppeliaSim es un simulador apto para ejecutarse en una amplia variedad de sistemas operativos, Pyrep solo es compatible con Linux por el momento y es por ello por lo que se decidió emplear este sistema

operativo para el desarrollo del proyecto. Al no disponer de un ordenador con el sistema operativo Linux se decidió ejecutar la simulación en VM VirtualBox [25], una máquina virtual en Windows con el sistema operativo Linux Ubuntu 18 instalado en la misma. Finalmente se optó por prescindir de Open-AI Gym y PyRep para poder trabajar con el sistema operativo nativo (Windows) y mejorar la capacidad de cálculo. De este modo se pudieron reducir los tiempos de ejecución.

Para el desarrollo de la red neuronal, se utilizó la biblioteca Keras [26], derivada de TensorFlow [27], desarrollada por Google y de código abierto.

No se han empleado datos externos para el desarrollo de este proyecto.

El esquema general del proyecto se representa en la *Figura 2.1* donde aparece la transferencia de datos que tiene lugar entre el entorno (virtual o de simulación V-Rep) y el agente (implementado con código Python).

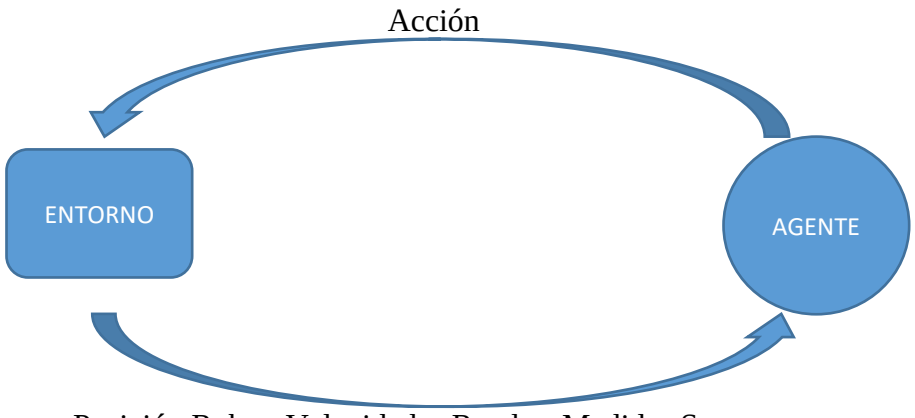

Figura 2.1 Arquitectura General del Sistema

Para la modelización de la política del robot se ha utilizado una red neuronal de 4 capas diseñada mediante Keras con 12 neuronas en la capa de entrada (las 10 medidas de los sensores del robot y las 2 últimas velocidades linear y angular aplicadas al robot) cuyas neuronas de las capas intermedias se encuentran dispuestas según la *Figura 2.2* y representadas en la *Figura 2.3*. La función de error a minimizar es la del error cuadrático medio y se ha usado el optimizador Adam. Adam es una combinación del *Stochastic Gradient Descent with momentum* y *RMSprop*. Este algoritmo de optimización promete una mayor eficiencia que otros algoritmos, haciendo que el error de la red llegue al mínimo en el menor tiempo posible. [28]

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 48)                624
_________________________________________________________________
dense_1 (Dense)              (None, 24)                1176
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 125
=================================================================
Total params: 1,925
Trainable params: 1,925
Non-trainable params: 0
```

Figura 2.2 Estructura Red Neuronal

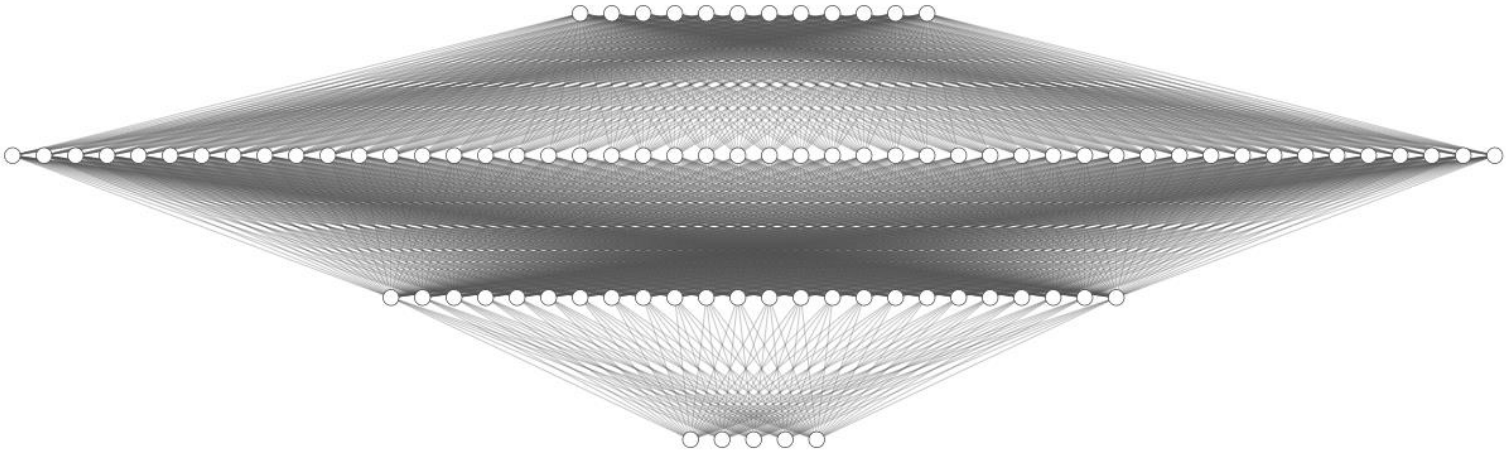

Figura 2.3 Representación Red Neuronal

El circuito cuadrangular que se ha usado para realizar las pruebas aparece representado en la *Figura 2.4.*

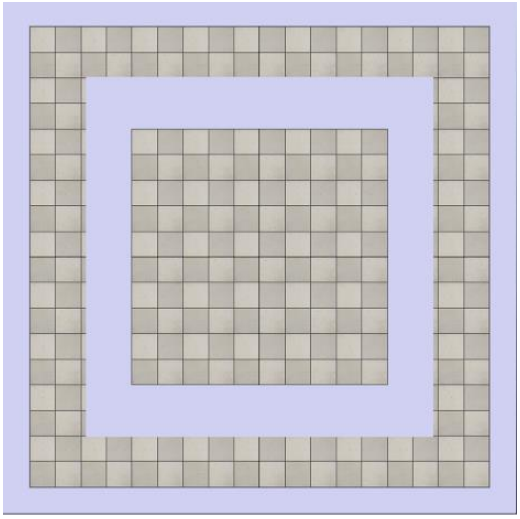

Figura 2.4 Circuito Cuadrangular

Las acciones que puede realizar el robot son: girar hacia la izquierda (w = -1 rad/s), girar levemente hacia la izquierda (w = -0.2 rad/s), no girar (w = 0 rad/s), girar levemente hacia la derecha (w = 0.2 rad/s) o girar hacia la derecha (w = 1 rad/s). En todo momento la velocidad lineal permanece constante (v = 0.5 m/s).

Estas 5 acciones que puede realizar el robot corresponden a las 5 neuronas presentes en la capa de salida de la red neuronal.

CAPÍTULO 3. ALGORITMOS PROPUESTOS

En cuanto al código y los algoritmos empleados en el desarrollo de este proyecto, el código base que se ha utilizado y adaptado para el diseño de los algoritmos internos del robot (diseño de funciones adaptadas al entorno utilizado como *step* o *reset*, cálculo de la posición del robot, establecimiento de un objetivo, medición de la distancia del robot al objetivo y su ángulo de desfase respecto a la dirección del objetivo, asignación de recompensas en función de todo lo anterior, etc) es [29] Se ha usado la navegación de tipo *"Odometry"* descrita en ese código, la cual emplea sensores de proximidad para la navegación del robot y el cálculo de su posición en el circuito.

Algunas funciones del código anterior han sido reemplazadas por otras que ya se habían implementado en el robot con anterioridad para el diseño del control PD en CoppeliaSim como la de leer sensores, aplicar velocidades a las ruedas, etc y también algunas otras referidas a funciones de la API remota de CoppeliaSim. Otras funciones se han obtenido directamente de la API remota de CoppeliaSim. La descripción de las funciones de la API remota de CoppeliaSim se encuentra en el siguiente enlace: [30]

En cuanto al algoritmo de aprendizaje por refuerzo, se ha empleado el algoritmo de Aprendizaje Profundo Q con redes neuronales o *"Deep Q-Learning"* debido a que al disponer de un espacio de observación continuo (10 medidas de los 16 sensores del robot a las paredes y las dos últimas velocidades aplicadas a las ruedas derecha e izquierda) se necesitaba un algoritmo que utilizara redes neuronales para relacionar estas 12 entradas descritas con las correspondientes salidas. En un principio y para la versión más simplificada del algoritmo estas salidas fueron: girar ligeramente a la izquierda, girar ligeramente a la derecha o no girar. Tal como se observa, en esta primera fase la velocidad lineal del robot y el ángulo de incremento de giro en cada iteración se mantuvieron fijos. El código base del algoritmo de aprendizaje profundo o *"Deep Q-Learning"* se obtuvo del siguiente enlace: [31]

Poniendo el foco en la funcionalidad del código en el *main* y el algoritmo de aprendizaje profundo o *"Deep Q-Learning"*, es necesario explicar su funcionamiento así como algunas de sus particularidades. En primer lugar, como es sabido, el propio nombre de *"Deep Q-Learning"* se debe al uso de redes neuronales para el desarrollo del algoritmo de aprendizaje por refuerzo. En el caso que concierne a este proyecto, no se está usando una única red neuronal, como es habitual, sino que se hace uso de dos redes neuronales. Este método fue propuesto y desarrollado por Deepmind para optimizar la convergencia del algoritmo de *"Deep Q-Learning"* tradicional [31] y su eficacia se basa en lo siguiente: se dispone de dos redes neuronales, una de ellas llamada *"model"*, la cual determinará qué acción se debe tomar en función del estado en cada momento, y otra llamada *"target_model"* la cual se encargará de localizar un objetivo óptimo de cara a futuras acciones. Es preferible utilizar dos redes neuronales para el desarrollo de estas dos funciones por separado debido a que de esta manera se logra obtener una red neuronal

que establece objetivos hacia el futuro más parsimoniosos, y por tanto con mayor estabilidad, y otra red neuronal que tome las decisiones en cada instante más cambiante, la cual se adapte en mayor medida a las respuestas del entorno. Esto permite disponer de un objetivo hacia el futuro mayormente estable a la vez que una adaptación al entorno presente más rigurosa, obteniendo por tanto mayor estabilidad y convergencia en el proceso. Para el desempeño de estas tareas se introduce un nuevo hiperparámetro llamado "tau", el cual permitirá un balance ajustable en el entrenamiento de la red neuronal *"target_model"* donde los pesos de la red más adaptable *"model"* con los que se actualiza *"target_model"* poseerán únicamente una pequeña importancia relativamente a la configuración previa de la red *"target_model"*, permaneciendo *"target_model"* más estable.

Se inicializa gamma, épsilon, épsilon_min, épsilon_decay, learning_rate y tau
Se inicializa "step" a 0 y "trial" a 0
Se repite (para cada "trial"):
    Se resetea el entorno (vuelve el robot a su posición inicial)
    Se repite (para cada "step")
        épsilon = max(épsilon_min, épsilon)
        Si "random" < épsilon
            Se realiza una acción aleatoria
        Si no,
            Se realiza la acción con mayor valor que prediga "model" según el estado actual
        Se recibe el nuevo estado al que conduce la acción, la recompensa, si se ha finalizado el "trial" o no (si se debe salir del bucle "step"), y si se ha tenido éxito o no
        Se cuenta un "step" más
        Se guarda en la memoria lo aprendido en un bloque (el estado inicial, la acción realizada, el estado al que se llega, la recompensa recibida, si se ha finalizado el "trial" o no, y si se ha tenido éxito o no)
        Se toma una muestra aleatoria de un conjunto de bloques de información guardados en la memoria
        Se repite (para cada bloque de información)
            Se obtiene la predicción de "target_model" a partir del estado actual
            Si se ha finalizado con éxito el "trial"
                En la predicción de "target_model" se modifica lo predicho para la acción de ese bloque y se sustituye por la recompensa recibida en ese bloque.

Si no se ha finalizado con éxito el "trial"
Q_future = máxima predicción de "target_model" en el estado del bloque
En la predicción de "target_model" se modifica lo predicho para la acción de ese bloque y se sustituye por: reward + Q_future*gamma
Se entrena "model" con el estado del bloque y la nueva predicción de "target_model" modificada.
Se obtienen los pesos de la red "model" → weights
Se obtienen los pesos de la red "target_model" → target_weights
Se repite (para cada peso de la red "target_model")
target_weights[i] = weights[i]*tau + target_weights[i]*(1-tau)
Se actualizan los pesos de la red "target_model" con los nuevos pesos modificados
El estado actual pasa a ser el nuevo estado obtenido después de la acción
Si se ha finalizado el "trial"
Se cuenta un "trial" más
Se inicializa "step" a 0
Si no se ha finalizado con éxito el "trial":
épsilon = épsilon*épsilon_decay
Si "step" > 40:
Se guarda el modelo de la red
Si se ha finalizado con éxito el "trial":
Se guarda el modelo de la red

Este es el pseudocódigo del algoritmo de *Deep Q-learning* y lo que ocurre en el *main*. A continuación, se explica más en detalle este pseudocódigo:

La funcionalidad del código en el *main* es la siguiente: en primer lugar se inicializan las variables necesarias para almacenar los datos y así poder representarlos posteriormente, después se crea el agente y se inicializa el bucle *"trial"* el cual medirá una iteración por cada intento del robot en recorrer el circuito (hasta el momento en que choque contra una pared, momento en el cual volverá al punto de partida), y dentro del bucle *"trial"* se encontrará el bucle *"step"*, el cual medirá una iteración por cada acción que realice el robot en función de su estado. Esta acción se basará en la decisión tomada por la red neuronal *"model"*. En el bucle *"trial"* se comenzará siempre reiniciando el entorno, lo cual llevará al robot al punto de partida, y se termina almacenando la información relevante para representarla en gráficas posteriormente. En el bucle *"step"* se decide en primer lugar la acción a realizar en función del estado almacenado en la iteración previa, tras ello se realiza la acción y se almacena el nuevo estado al que se ha llegado *("new_state"),* la recompensa *("reward"),* si se ha chocado con una pared *("done")* y si se ha dado una vuelta completa *("info")*. A continuación, se guarda en la memoria con la función *"remember"* el estado de partida *("cur_state")*, la acción realizada *("action")*, la recompensa recibida *("reward")*, el nuevo estado al que se ha llegado *("new_state")*, si se ha chocado con una pared *("done")* y si se ha dado una vuelta completa *("info").*

Después se procede a entrenar la red neuronal modelo *("model")* a partir de una serie de muestras aleatorias almacenadas en la memoria previamente para evitar cualquier tipo de sesgo, teniendo en cuenta para ello la recompensa recibida y la máxima recompensa que cabe esperar en el futuro desde ese estado (la cual aporta la red *"target_model"*), siendo esta máxima recompensa ponderada por el clásico factor *"gamma"*. Tras ello se entrena la red neuronal *"target_model"* mediante el método mencionado anteriormente haciendo uso de la "tau". Si el nuevo estado es un estado de choque con la pared, se sale del bucle *"step",* se vuelve al bucle *"trial"* y se reinicia el entorno, y si no es un estado de colisión se itera de nuevo el bucle *"step".*

CAPÍTULO 4. ANÁLISIS DE RESULTADOS

4.1 Resultados del caso base

En un primer momento se decidió únicamente penalizar al robot con una penalización muy alta cada vez que se chocase con una pared (-1000) y se obtuvo como resultado un algoritmo de control a partir del cual el robot aprendió a recorrer aproximadamente la mitad del circuito cuadrangular sin chocarse, aunque presentaba numerosas oscilaciones en su trayectoria.

Es por eso por lo que se decidió mantener la penalización anterior y además recompensar al robot por alejarse de las paredes de manera que recibía en todo momento como recompensa la menor distancia a las paredes que sus sensores de proximidad fueran capaces de observar y el robot aprendió a recorrer el circuito sin oscilaciones, alejándose todo lo posible de las paredes del circuito en todo momento, realizando giros perfectos centradamente entre las dos paredes delimitantes del recorrido, pero sin embargo sin control en las zonas de movimiento recto lineal a lo largo de los pasillos rectos del circuito. Debido a este comportamiento inusual sin control resultó imposible para el robot llegar a recorrer completamente el circuito ya que era muy probable que a lo largo de algún pasillo recto terminara chocando con la pared debido a un ángulo de salida de la curva no del todo perfecto y centrado en el pasillo recto.

Una vez obtenidos estos resultados se procedió a realizar un estudio de los hiperparámetros "Tau" y *"Learning Rate"* para comprobar si fuera posible optimizar el aprendizaje del robot en el circuito y lograr que lo llegara a recorrer completamente. Se realizaron 30 pruebas en total (con 3 valores de "Tau" y 10 valores de *"Learning Rate"*) y a pesar de no lograr que el robot llegara a recorrer el circuito con éxito se observó que los valores óptimos para estos hiperparámetros eran de 0.05 para "Tau" y de 0.0002 para *"Learning Rate"*.

Tras realizar numerosas pruebas en el entorno virtual de Linux se comprobó que se estaban obteniendo multitud de errores en cuanto a la medición de los *"steps"* o "pasos" que daba el robot en cada intento de recorrer el circuito, los cuales se reflejaban en que a medida que avanzaba la simulación se medían cada vez menos *"steps"* del robot para recorrer la misma distancia. Esto impedía obtener una medida realista acerca de si el robot estaba realmente aprendiendo a recorrer o no el circuito y daba una sensación de que el robot en lugar de aprender experimentaba el efecto contrario. Se trató de solucionar el error sin éxito. Este error puede deberse a un fallo existente en la versión de CoppeliaSim para Linux o a un error que aparece a la hora de tratar de sincronizar el código utilizado con el simulador. A este error se le sumaba un error de ralentización de la simulación a medida que esta avanzaba el cual también se trató de solucionar sin éxito.

Debido a todos estos problemas encontrados en Linux se decidió cambiar de plataforma y adaptar los archivos que se estaban usando en Linux para poder usarlos en Windows y se empleó para ello el editor y ejecutor de código profesional Pycharm [32]. Los archivos utilizados y modificados adaptados al nuevo entorno fueron "main.py", "dqn.py", "robot_p3dx.py", "robot.py", "sim.py", "simConst.py" y "timing.py" [33], lo cual contribuyó a simplificar en gran medida el código empleado para llevar a cabo la simulación y se continuó utilizando como simulador para las pruebas CoppeliaSim aunque en su versión para Windows.

Una vez que se logró trasladar con éxito todo el entorno a Windows se encontró el problema de que al ejecutar la simulación el simulador comenzaba a consumir de manera exponencial más y más recursos de la memoria RAM del ordenador lo que provocaba que la simulación se ralentizase y se terminase parando siempre al poco tiempo y bloqueándose al realizar el robot un número no muy alto de intentos para tratar de recorrer el circuito. Este error se logró solucionar actualizando la versión de TensorFlow utilizada (se utilizaba la versión 1.0 y se actualizó a la versión 2.3). Tras actualizar TensorFlow y a continuación instalar el paquete Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019 x64 [34] se solucionó el error de memoria RAM y se pudo proceder a realizar las pruebas de la simulación correctamente y alcanzando un número considerable de intentos sin llegar a colapsar nunca más la memoria ram y sin encontrar más errores que cierta ralentización de la simulación con el paso del tiempo pero en perfecta sincronización las mediciones de los *"steps"* en las gráficas con la visualización del robot en el simulador. Es posible que esta fuera también la solución a los problemas que se experimentaron previamente en Linux. A continuación, se describen los resultados obtenidos tras estas pruebas.

Las dos primeras pruebas que se realizaron se llevaron a cabo asignando al robot una memoria de tamaño máximo 2000 y se utilizaron para comprobar qué influencia podía tener el usar en el simulador 10 ppf (*"simulation passes per frame"*, lo cual se utiliza para acelerar la visualización en el simulador pues solo se muestran al usuario en la pantalla 1 de cada 10 acciones ejecutadas por el robot en el simulador) o 20 ppf, y se concluyó que esta diferencia no era relevante por lo que decidió usarse 10 ppf en el resto de las pruebas. En ambas pruebas el robot no superó los 500 steps (1/3 del circuito).

A continuación, se probó a modificar el tamaño de la memoria máxima del robot y en vez de 2000 se le asignó un tamaño máximo de 200. Gracias a esto por primera vez el robot logró alcanzar el máximo número de *steps* posible asignado (1000 *steps*, lo que equivale a recorrer ¾ partes del circuito) en varias ocasiones consecutivas durante la misma prueba. A continuación, se presentan los resultados obtenidos:

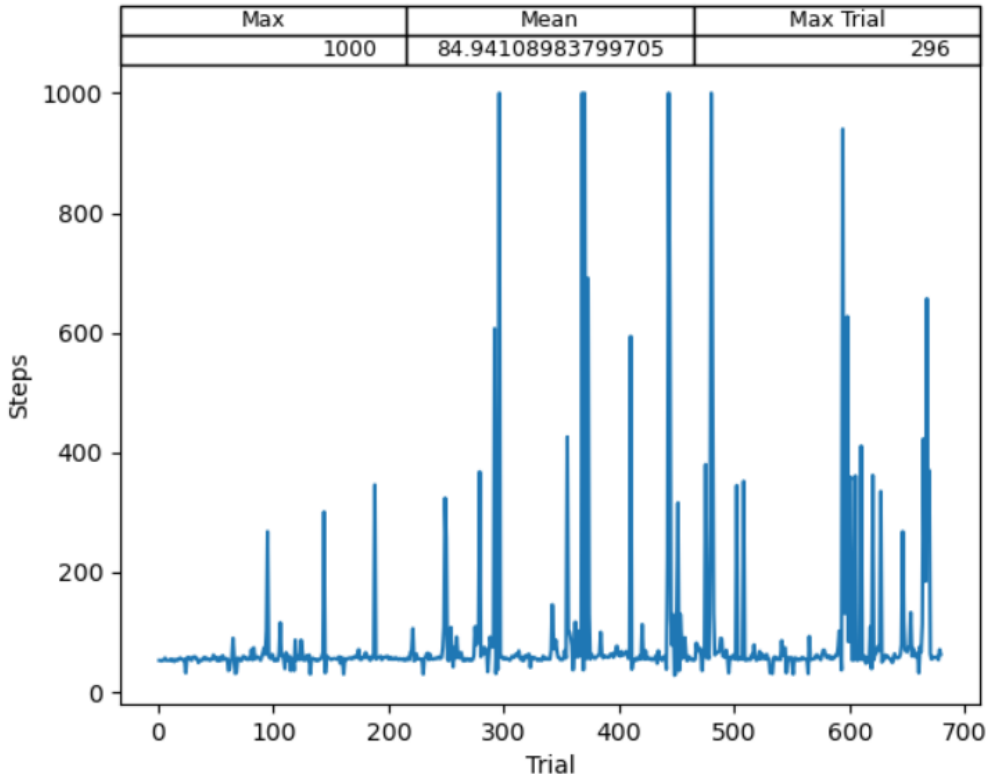

Figura 4.1 Steps por Trial

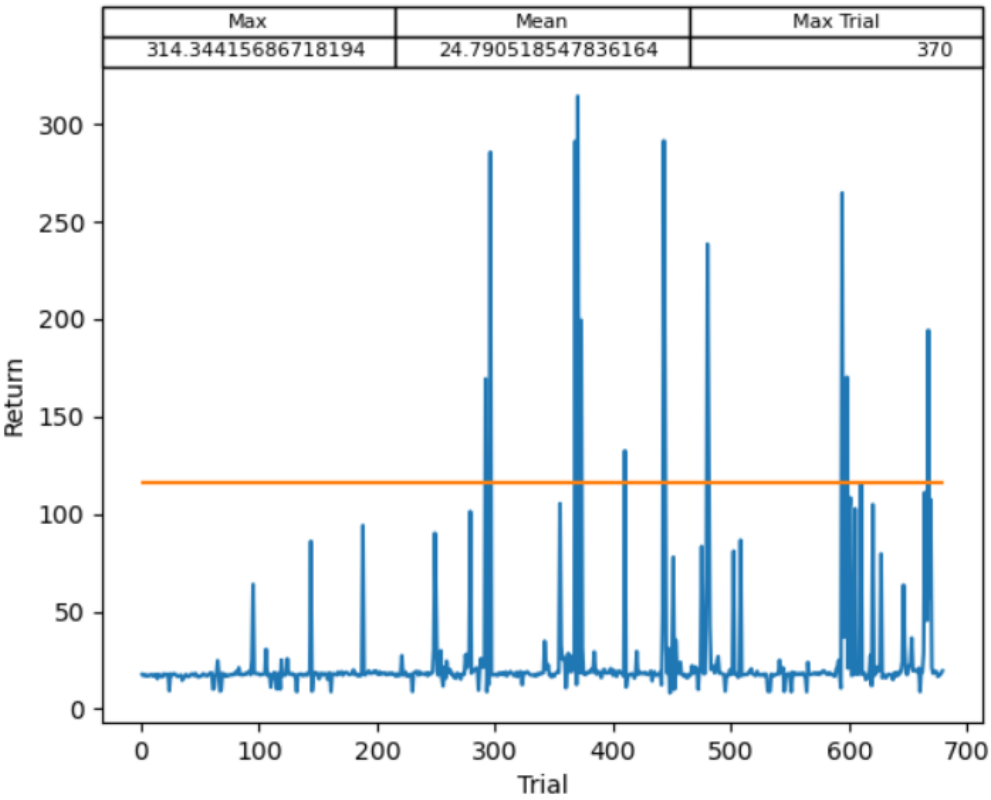

Figura 4.2 Recompensa por Trial

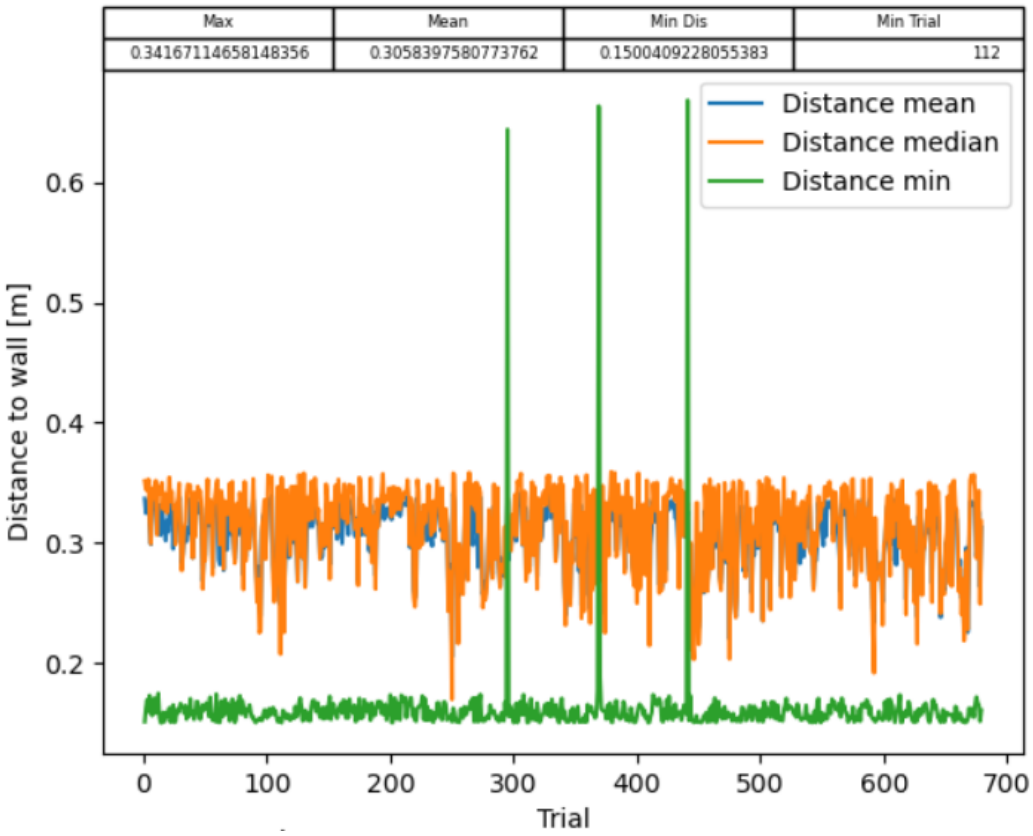

Figura 4.3 Distancia Media, Mediana y Mínima a la pared por Trial

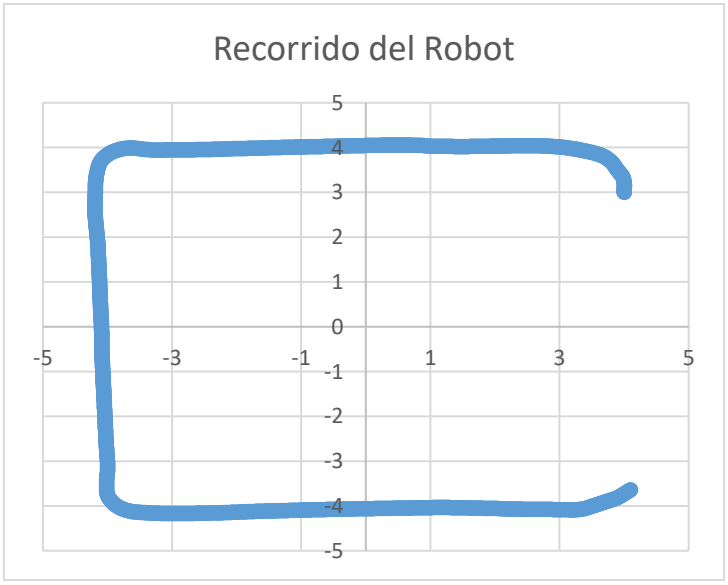

Figura 4.4 Recorrido del Robot en el Trial 296

Para certificar la validez del método y la eficacia del aprendizaje se planeó realizar de nuevo la prueba anterior con exactamente los mismos parámetros y únicamente modificando el parámetro del máximo número de *steps* por recorrer (2500) para que el robot tuviera libertad de recorrer el circuito por completo. Estos fueron los resultados:

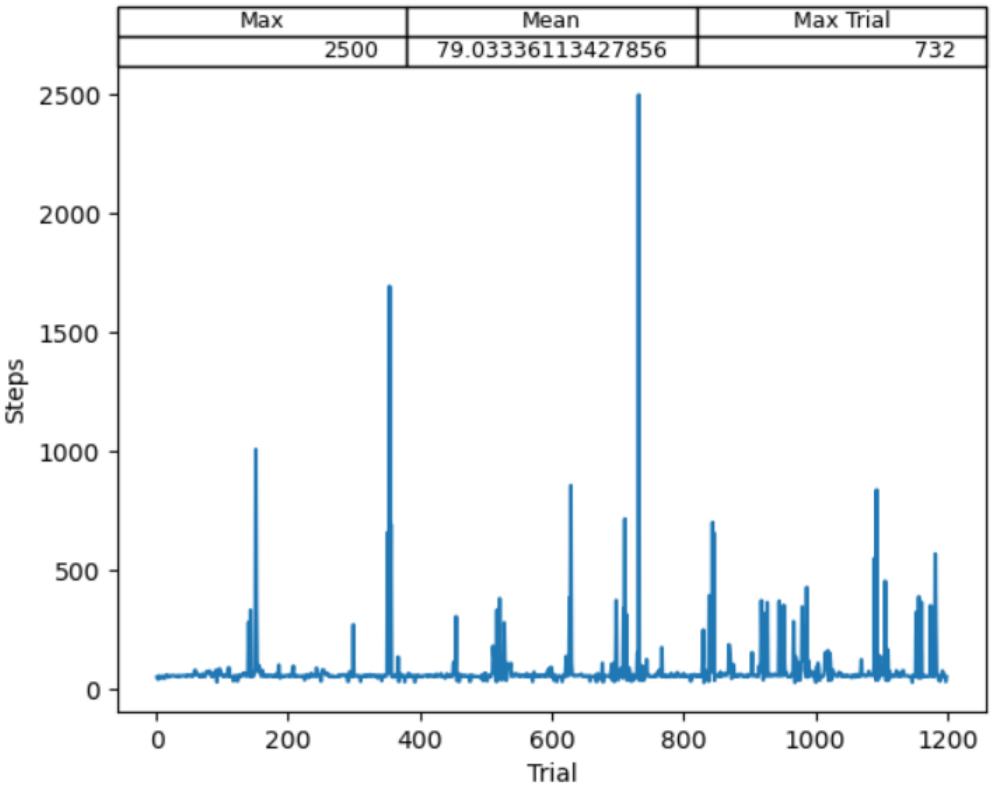

Figura 4.5 Steps por Trial

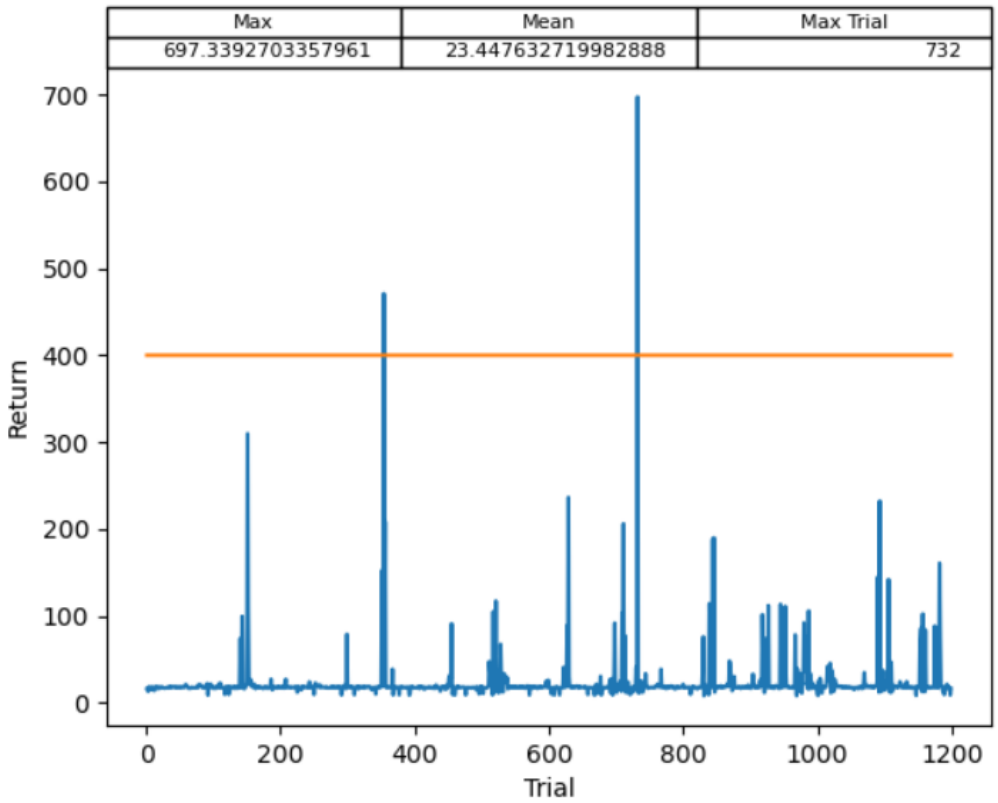

Figura 4.6 Recompensa por Trial

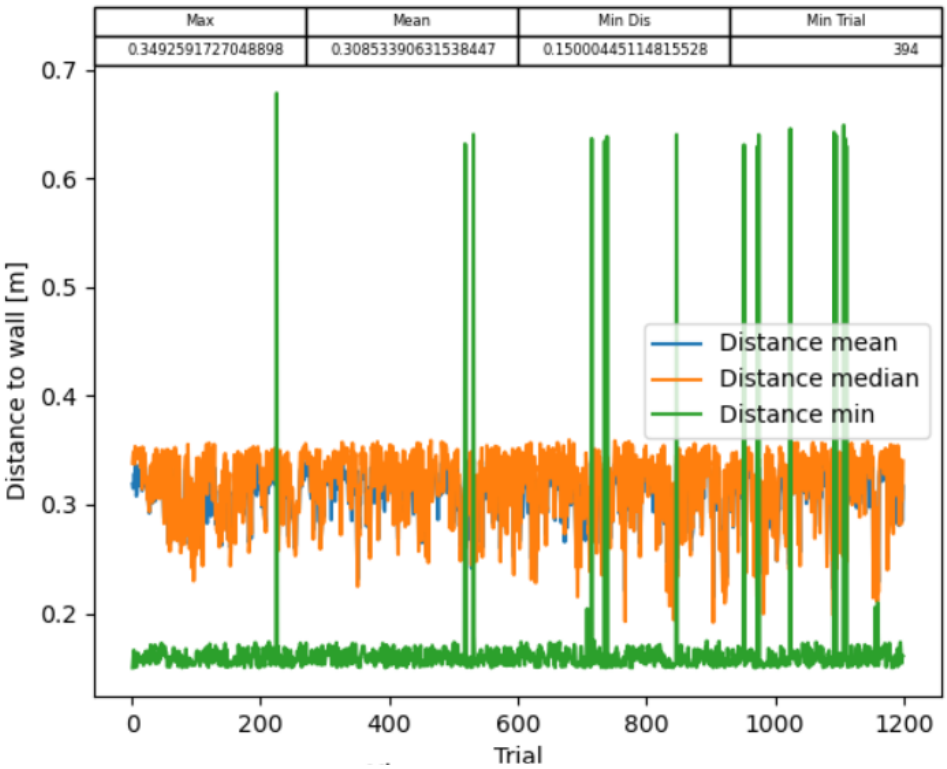

Figura 4.7 Distancia Media, Mediana y Mínima a la pared por Trial

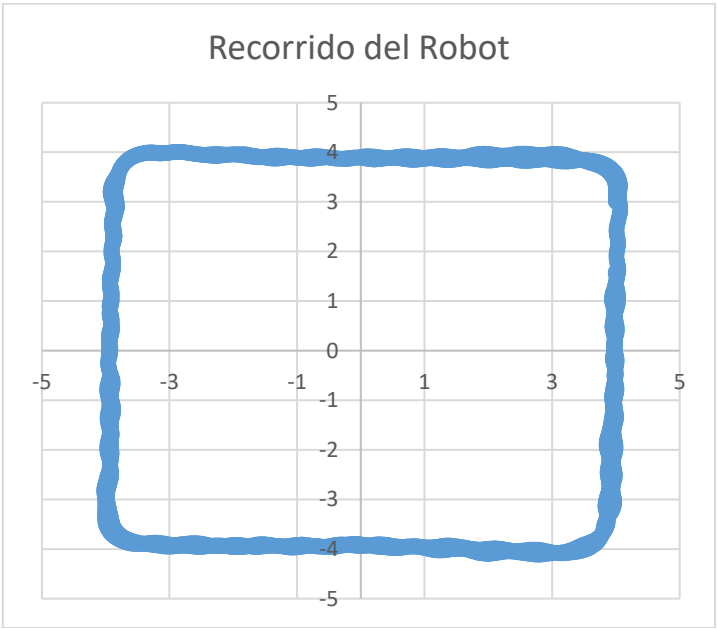

Figura 4.8 Recorrido del Robot en el Trial 732

Esta prueba verifica la validez del método de aprendizaje empleado ya que en ella el robot logra recorrer el circuito por completo y no únicamente una vez sino incluso dos veces en la misma prueba y logrando llegar al máximo de *steps* asignado (2500) dando así dos vueltas completas al circuito (*Figura 4.8)*. Más adelante se quiso realizar otra prueba poniendo un límite de *steps* más reducido (1500) para que se diese únicamente una vuelta completa como máximo al circuito y estos fueron los resultados:

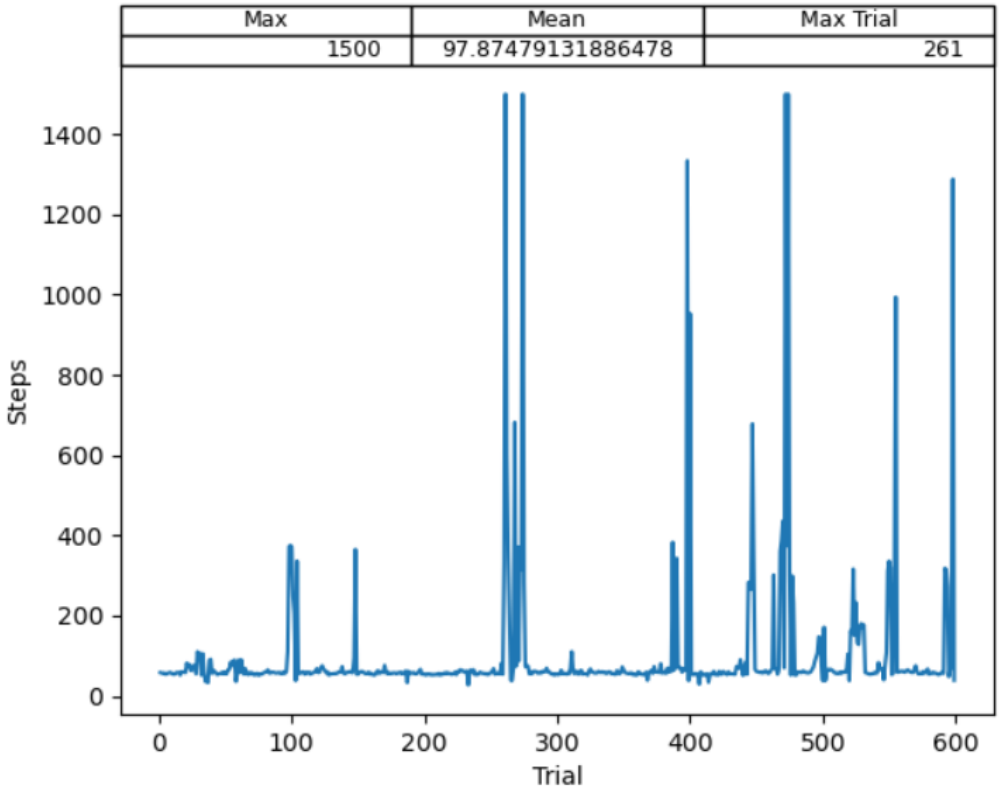

Figura 4.9 Steps por Trial

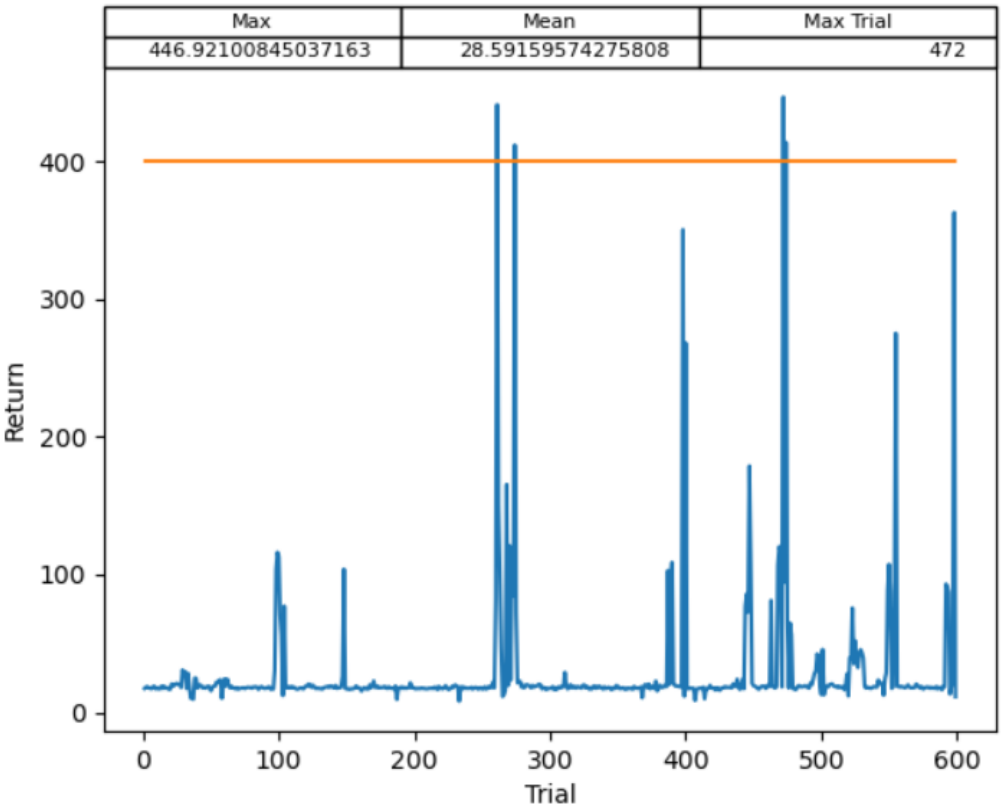

Figura 4.10 Recompensa por Trial

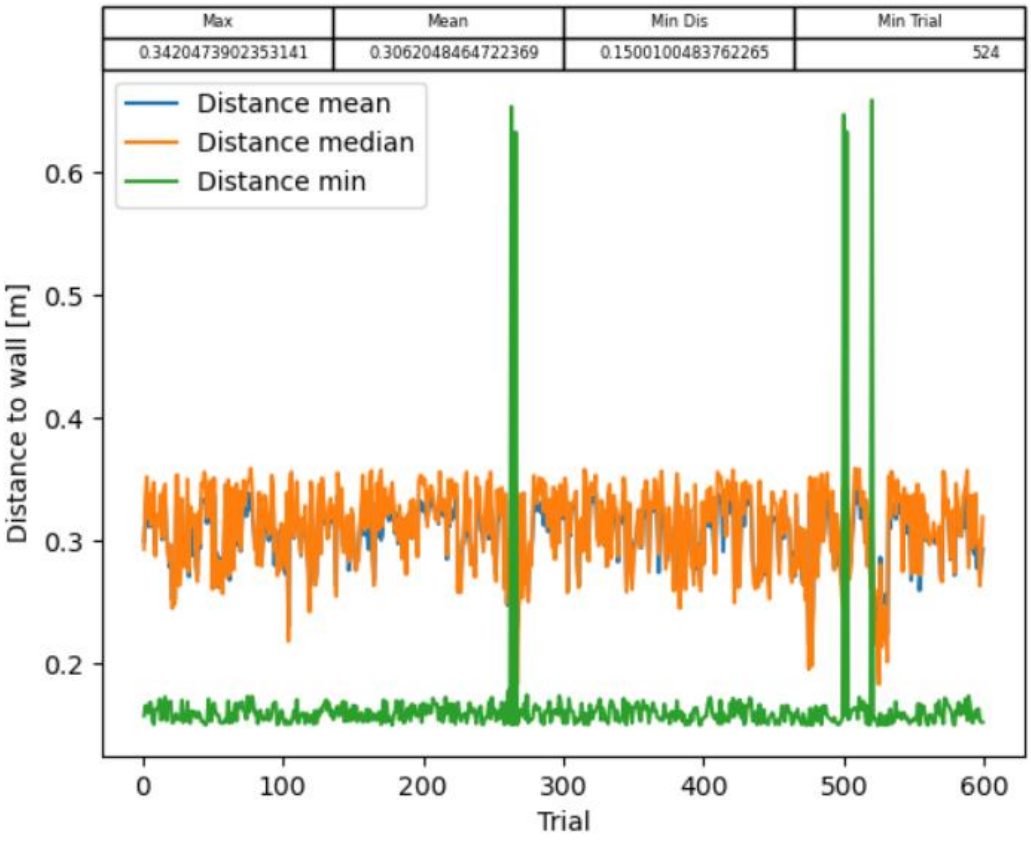

Figura 4.11 Distancia Media, Mediana y Mínima a la pared por Trial

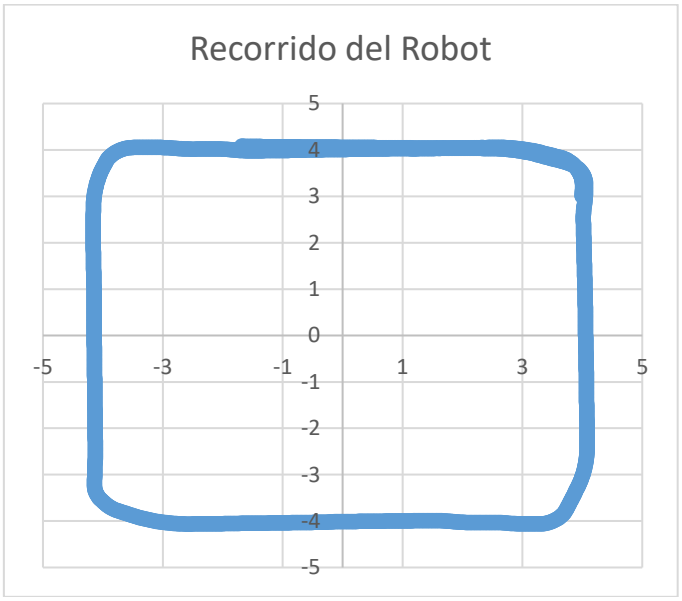

Figura 4.12 Recorrido del Robot en el Trial 472

En esta prueba el robot aprende perfectamente a recorrer el circuito por completo sin chocarse una vez más. De modo similar se realizaron muchas más pruebas y todas ellas resultaron exitosas. Para comprobar cómo afecta al resultado final modificar el tamaño máximo de la memoria se realizó una prueba en la que se asignaba el tamaño máximo de 400 a la memoria y se obtuvo el siguiente resultado:

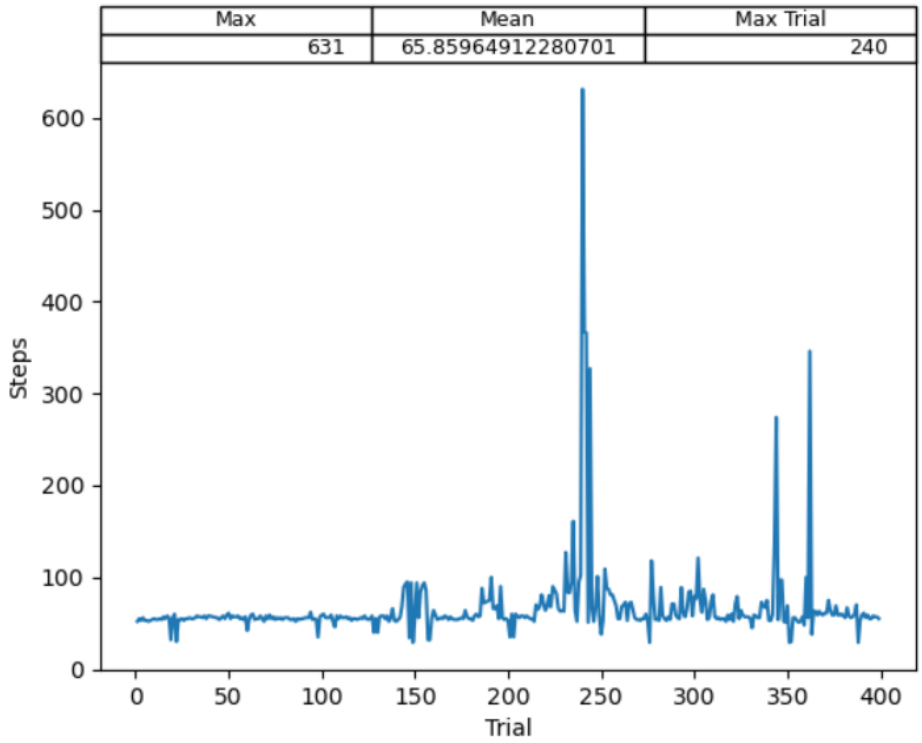

Figura 4.13 Steps por Trial

En esta prueba se puede comprobar el efecto adverso de aumentar demasiado el tamaño máximo de la memoria pues tal y como se observa en la gráfica el robot no es capaz de recorrer ni siquiera la mitad del recorrido total. A continuación se decidió realizar la misma prueba pero en este caso con una tamaño máximo de la memoria de 300 y se obtuvo el siguiente resultado:

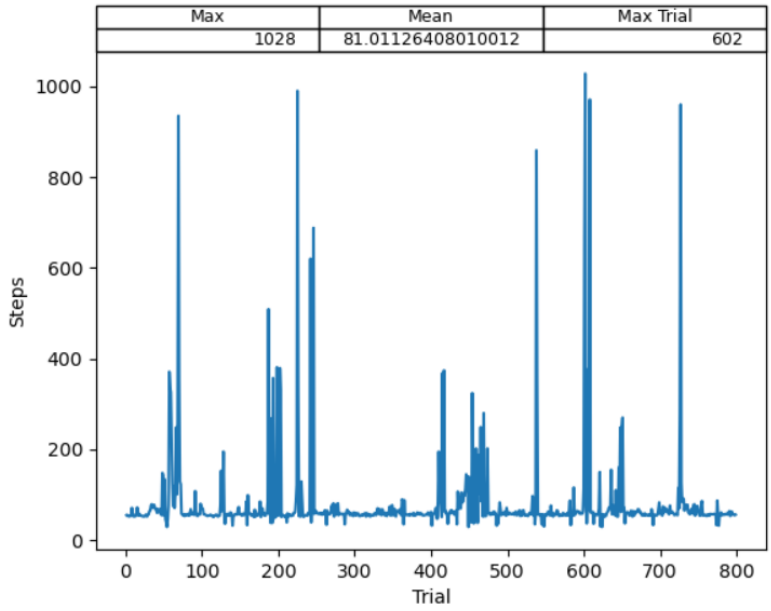

Figura 4.14 Steps por Trial

En esta prueba se puede apreciar que el resultado ha mejorado notablemente respecto a la prueba anterior pero sin embargo está todavía lejos de alcanzar la eficacia que se obtiene al usar 200 como el tamaño de la memoria máximo.

Tras esta prueba se trató de comprobar cuál sería el efecto de reducir el tamaño máximo de la memoria por debajo de 200, asignándole un valor de 100 para comprobar si 200 es realmente el valor óptimo o puede existir todavía alguno más beneficioso reduciendo todavía más el tamaño de la memoria. Los resultados obtenidos se presentan a continuación:

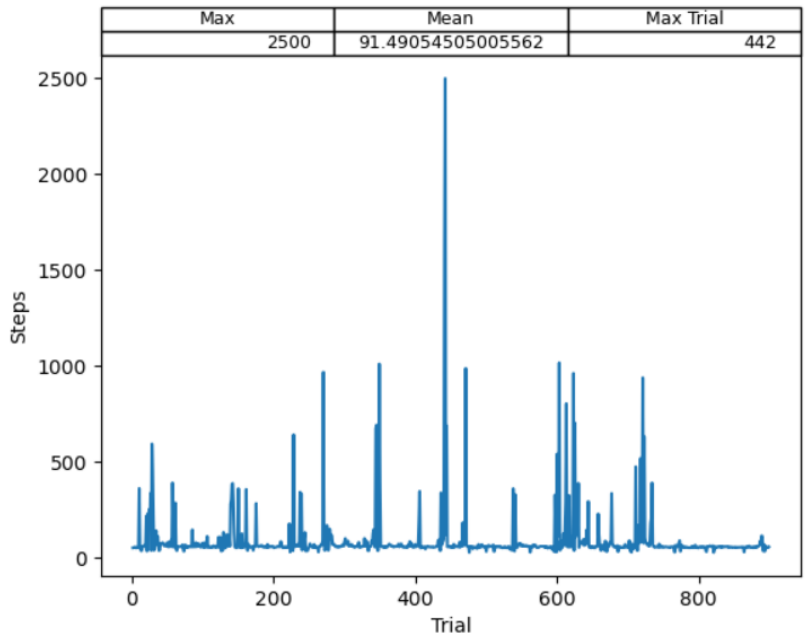

Figura 4.15 Steps por Trial

De esta prueba se puede concluir que 100 también sería un tamaño máximo de memoria aceptable ya que el robot ha llegado a dar 2 vueltas seguidas completas al circuito alcanzando el límite máximo de steps, aunque 200 como tamaño máximo sigue siendo algo más beneficioso ya que, tal como lo presentamos en una gráfica anterior, el robot logra recorrer el circuito por completo varias veces durante la misma prueba mientras que en este caso únicamente lo logra recorrer una única vez durante la prueba. Por tanto, podemos concluir que 200 es el tamaño óptimo de memoria del robot.

Finalmente, una vez obtenidos los resultados exitosos, se procedió a comparar el control PD del que se disponía originalmente con el control desarrollado mediante aprendizaje por refuerzo y lograr determinar así el nivel de optimización proporcionado mediante este último frente al original.

En primer lugar, se realizó una comparación cualitativa entre estos dos controles para determinar con cuál de los dos el robot lograba adaptarse en mayor medida al circuito.

La trayectoria que describe el robot mediante el control PD original aparece representada en la *Figura 4.16.* El objetivo de este control fue el de mantener continuamente una distancia con las paredes del circuito de 0.36175 m (avanzar por el centro del pasillo) de manera que únicamente giraba cuando fuera imposible continuar siguiendo la pared debido a que una nueva pared se interponía en su camino. En este caso el robot se pararía y giraría hasta que observase el camino nuevamente despejado (describiendo en nuestro caso un giro de 90º) para continuar avanzando por el circuito. El resultado se observa en la *Figura 4.16.*

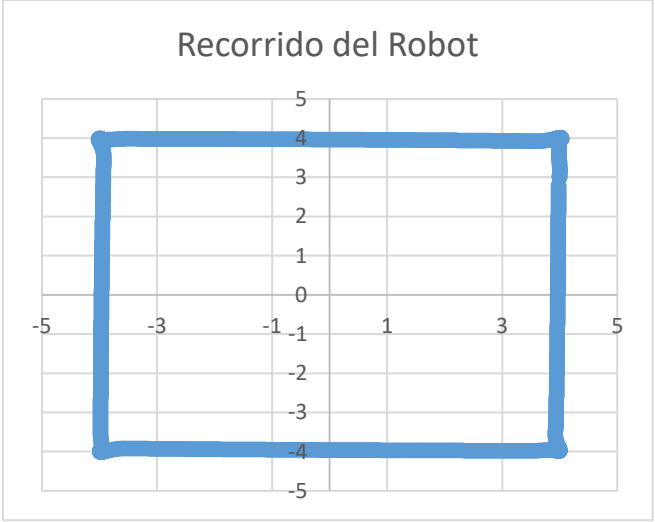

Figura 4.16 Recorrido del Robot con el control PD

Por otro lado, la trayectoria que describe el robot mediante el control desarrollado utilizando aprendizaje por refuerzo aparece representado en la *Figura 4.17.*

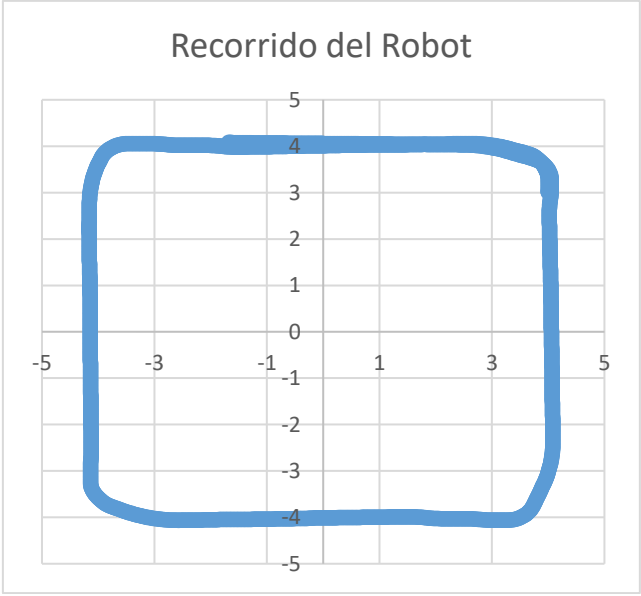

Figura 4.17 Recorrido del Robot en el Trial 472

De la comparación de estas dos trayectorias se puede observar la gran ventaja competitiva que presenta el control desarrollado mediante aprendizaje por refuerzo. En este control no se le dan instrucciones previas al robot acerca de cómo debe moverse por el circuito, sino que se deja que el mismo robot, tras explorar el entorno e interactuar con él, sea capaz de encontrar la trayectoria óptima para recorrer el circuito evitando las penalizaciones y tratando de maximizar en todo momento la recompensa obtenida. Aprende que no debe chocarse con la pared bajo ningún concepto y que debe tratar de mantener la mayor distancia posible a las paredes del circuito en todo momento. De esta manera se logra una adaptación al circuito mucho mayor, la cual queda patente en la enorme adaptación a las curvas del circuito que ha desarrollado el robot utilizando el control obtenido a través de aprendizaje por refuerzo, además de recorrer el circuito tratando siempre de situarse en el centro del pasillo. Esta adaptación a las curvas resulta más difícil de conseguir mediante un control convencional como el PD.

Por otro lado, se llevó a cabo de igual manera una comparación cuantitativa entre estos dos controles para determinar con exactitud el grado de optimización que se pudo alcanzar empleando el control obtenido mediante aprendizaje por refuerzo frente al control PD convencional.

Se utilizó como medidor el tiempo que tarda el robot en dar una vuelta completa al circuito con cada uno de estos dos controles manteniendo constante la velocidad de avance en todo momento y se obtuvo el siguiente resultado: haciendo uso del control PD original el robot empleó 40 segundos en recorrer el circuito por completo, mientras que con el control desarrollado a partir de aprendizaje por refuerzo el robot únicamente necesitó 30.2 segundos para recorrerlo. En ambas pruebas se mantuvo fija y constante en todo momento la velocidad lineal del robot en 0.5 m/s. De esta prueba se concluye que el algoritmo de aprendizaje por refuerzo ha logrado optimizar el desempeño del robot en el circuito aproximadamente un 25% respecto al resultado obtenido a partir del control PD original.

4.2 Análisis de sensibilidad

Se ha llevado a cabo un análisis exhaustivo del efecto de los distintos parámetros en los resultados el cual se presenta en la *Tabla 2.*

Tabla 2. *Parámetros ajustables*

Parámetro	Función / Efecto	Valor asignado
Gamma	Gamma mide la repercusión que tiene el valor asignado a los posibles estados futuros a la hora de asignar un valor al estado actual. Cuanto más alto sea este parámetro más importancia tendrán las recompensas o penalizaciones obtenidas en estados futuros respecto al estado actual. Entre 0 y 1.	1
Épsilon mínimo	Este parámetro indica cual es la probabilidad que existirá siempre como mínimo de realizar una acción aleatoria en cada instante. No es conveniente que este parámetro sea demasiado alto debido a que podría suceder que resulte imposible alcanzar una política aproximadamente determinista debido a ello. Entre 0 y 1. Cuanto más alto mayor porcentaje de aleatoriedad existirá como mínimo siempre.	0.1
Épsilon decay	Este parámetro define la rapidez con la que decrece épsilon a lo largo de cada intento del robot (“trial”) de recorrer el circuito dentro de la misma prueba. Cuanto más alto sea más lento decrecerá épsilon en cada “trial” ya que este valor se multiplica por épsilon en cada “step”. Entre 0 y 1.	0.95
Learning rate	Este parámetro mide la influencia que tienen las recompensas y las penalizaciones en general a la hora de modelar la política. Influye a la hora de entrenar la red neuronal. Es importante que este valor no sea muy alto pues podría suponer una situación de falta de convergencia de la simulación. Entre 0 y 1.	0.0002
Tau	Este parámetro mide la influencia que tienen las nuevas recompensas y penalizaciones recibidas a la hora de entrenar la red neuronal “target model” la cual se encarga de definir un objetivo estable en el futuro al cual debe tratar siempre de acercarse la red neuronal “model”. Es importante que este parámetro no sea demasiado alto debido a que podría dar lugar a una situación de inestabilidad en la cual la red neuronal “model” no tenga claro hacia dónde debe tender y el método termine por no converger. Entre 0 y 1.	0.05

El término “Valor asignado” se refiere al valor asignado a ese parámetro el cual se ha demostrado ser el óptimo para este proyecto.

CAPÍTULO 5. CONCLUSIONES Y TRABAJOS FUTUROS

5.1 Conclusiones sobre la metodología

Es preciso considerar que las herramientas que se plantearon inicialmente para llevar a cabo este proyecto no resultaron del todo fructuosas debido a ciertos problemas que se lograron solucionar posteriormente al cambiar de plataforma. Es por ello por lo que la distribución del tiempo para la realización del trabajo se ha visto afectada. Sin embargo, es importante destacar que una vez que se resolvieron los problemas que aparecieron inicialmente los resultados correctos no tardaron más de una semana en aparecer. Esto se debe con seguridad a la enorme cantidad de tiempo que se invirtió en el correcto planteamiento del problema y diseño adecuado del código el cual no era capaz de ofrecer sus frutos debido únicamente a multitud de problemas técnicos que se tuvieron que enfrentar. Lograr superarlos todos ha sido un enorme motivo de satisfacción.

Por tanto, se concluye que una vez que se ha logrado la puesta a punto de las herramientas básicas para comenzar la exploración y el entrenamiento de la red neuronal para poder empezar a desarrollar el algoritmo, los primeros resultados han aparecido con rapidez y al poco tiempo, lo que nos confirma la validez, idealidad y eficacia de las herramientas utilizadas.

5.2 Conclusiones sobre los resultados

Los resultados finalmente han sido satisfactorios y han estado a la altura de las expectativas, demostrando el control desarrollado mediante aprendizaje por refuerzo ser más efectivo y hábil que el control PD convencional, mejorando la rapidez del robot en recorrer el circuito, reduciendo el tiempo empleado para recorrerlo por completo, y su destreza para hacer los giros pertinentes, tal y como se observa en las gráficas presentadas.

Sobre todo ha resultado sorprendente la enorme importancia que finalmente ha resultado tener el hecho de asignar correctamente el tamaño máximo de la memoria del robot para lograr un aprendizaje exitoso y la rapidez con la que, una vez se asigna correctamente este valor, el robot logra aprender a recorrer el circuito por completo de manera exitosa.

5.3 Recomendaciones para futuros estudios

- De cara a futuros estudios relacionados con el control de robots móviles mediante algoritmos de aprendizaje por refuerzo se recomienda encarecidamente tener muy en cuenta el tamaño de la memoria máxima del robot para lograr obtener resultados aceptables.
- Algo que ha llamado la atención es el hecho de que nuestras gráficas presentan resultados exitosos pero sin embargo no encontramos curvas de crecimiento más o menos constantes a medida que pasa el tiempo de aprendizaje sino que obtenemos resultados exitosos esporádicos. Pensamos que esto se debe a que el factor de aleatoriedad nunca ha comenzado siendo lo suficientemente pequeño al comienzo de cada uno de los trials y por tanto no hemos logrado adquirir del todo una política determinista en el momento crucial en el que se toma la primera curva lo que puede haber provocado esa gran aleatoriedad que se observa en las gráficas. Sería recomendable tener esto en cuenta para futuros estudios.
- Por otro lado, resulta muy recomendable no reducir nunca el factor de descuento 'γ' por debajo de 0.5 para que el robot sea capaz de tener en cuenta con la suficiente antelación las penalizaciones o recompensas derivadas de las decisiones tomadas en estados futuros que le indiquen al robot la conveniencia o no de mantenerse en la dirección del camino elegido.
- También se recomienda mantener el factor "ε" de aleatoriedad en todo momento como mínimo entre 0 y 0.1 (sin ser este un valor demasiado alto) para evitar posibles estancamientos del robot en la repetición de rutas recorridas anteriormente y facilitar la rectificación en todo momento del aprendizaje, especialmente en la fase de explotación, así como para comprobar la eficacia del robot ante posibles perturbaciones que puedan aparecer en el sistema de control a la hora de implementar el algoritmo en un robot físico.
- En cuanto a la asignación de recompensas y penalizaciones, es de vital importancia remarcar el rechazo a los posibles choques con las paredes del circuito determinando una penalización considerablemente más alta que el resto de recompensas o penalizaciones que puedan existir por mantenerse recorriendo el circuito el mayor tiempo posible o por acercarse demasiado a las paredes del circuito, consecutivamente, para que el robot pueda comprender con claridad el objetivo primordial del control que se está realizando y evitar posibles desajustes desafortunados.

CAPÍTULO 6. BIBLIOGRAFÍA

[1] Aprendizaje por refuerzo profundo https://www.iic.uam.es/aprendizaje-profundo-por-refuerzo/ Última visita: 12/03/2020

[2] Aprendizaje por refuerzo y el tratamiento del cáncer. https://www.iic.uam.es/lasalud/aprendizaje-por-refuerzo-y-tratamiento-cancer/ Última visita: 12/03/2020

[3] Conceptos de inteligencia artificial: ¿qué es el aprendizaje por refuerzo? https://www.xataka.com/inteligencia-artificial/conceptos-inteligencia-artificial-que-aprendizaje-refuerzo Última visita: 12/03/2020

[4] La IA ayuda a descubrir que tenemos neuronas optimistas y pesimistas https://www.technologyreview.es/s/11824/la-ia-ayuda-descubrir-que-tenemos-neuronas-optimistas-y-pesimistas Última visita: 12/03/2020

[5] Presentamos la IA que enseña a los robots a caminar https://scienews.com/es/la-tecnolog-a/8362-presentado-a-la-ia-que-por-s-mismo-nos- ense-a-robots-caminar.html Última visita: 12/03/2020

[6] Control de un brazo robótico 2D con aprendizaje por refuerzo profundo https://blog.floydhub.com/robotic-arm-control-deep-reinforcement-learning/ Última visita: 12/03/2020

[7] Introducción al aprendizaje por refuerzo https://planetachatbot.com/introducci%C3%B3n-al-aprendizaje-por-refuerzo-f910d669d077 Última visita: 12/03/2020

[8] Curso UCL sobre RL http://www0.cs.ucl.ac.uk/staff/d.silver/web/Teaching.html Última visita: 18/10/2019

[9] Yo soy yo y mi circunstancia https://autismodiario.org/2017/07/19/yo-soy-yo-y-mi-circunstancia/ Última visita: 12/03/2020

[10] B. Zuo, J. Chen, L. Wang and Y. Wang, "A reinforcement learning based robotic navigation system," *2014 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, San Diego, CA, 2014, pp. 3452-3457.

[11] N. Altuntas, E. Imal, N. Emanet and C. Nur, " Reinforcement learning-based mobile robot navigation, " *2016 Turk J Elec Eng & Comp Sci*, 24: pp. 1747 – 1767.

[12] Khriji, L, Touati, F, Benhmed, K & Al-Yahmedi, "Mobile robot navigation based on Q-learning technique",*2011 International Journal of Advanced Robotic Systems*, vol. 8, no. 1, pp. 45-51.

[13] Gary G. Yen, Travis W. Hickey, "Reinforcement learning algorithms for robotic navigation in dynamic environments",*2003 ISA Transactions*, vol 43, issue 2, pp 217-230, ISSN 0019-0578

[14] X. Zhuang, "The Strategy Entropy of Reinforcement Learning for Mobile Robot Navigation in Complex Environments," *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, Barcelona, Spain, 2005, pp. 1742-1747.

[15] B. Gökçe and H. L. Akın, "Implementation of Reinforcement Learning by transfering sub-goal policies in robot navigation," *2013 21st Signal Processing and Communications Applications Conference (SIU)*, Haspolat, 2013, pp. 1-4.

[16] T. Xuan Tung and T. Dung Ngo, "Socially Aware Robot Navigation Using Deep Reinforcement Learning," *2018 IEEE Canadian Conference on Electrical & Computer Engineering (CCECE)*, Quebec City, QC, 2018, pp. 1-5. `

[17] M. Menegaz and P. M. Engel, "Using the GTSOM network for mobile robot navigation with reinforcement learning," *2009 International Joint Conference on Neural Networks*, Atlanta, GA, 2009, pp. 2073-2077.

[18] D. Tamilselvi, S. M. Shalinie and G. Nirmala, "Q learning for mobile robot navigation in indoor environment," *2011 International Conference on Recent Trends in Information Technology (ICRTIT)*, Chennai, Tamil Nadu, 2011, pp. 324-329.

[19] X. Liu, Q. Zhou, H. Ren and C. Sun, "Reinforcement Learning for Robot Navigation in Nondeterministic Environments," *2018 5th IEEE International Conference on Cloud Computing and Intelligence Systems (CCIS)*, Nanjing, China, 2018, pp. 615-619.

[20] J. Muhammad and I. Ö. Bucak, "An improved Q-learning algorithm for an autonomous mobile robot navigation problem," *2013 The International Conference on Technological Advances in Electrical, Electronics and Computer Engineering (TAEECE)*, Konya, 2013, pp. 239-243.

[21] CoppeliaSim https://www.coppeliarobotics.com/ Última visita: 05/04/2020

[22] Gym https://gym.openai.com/ Última visita: 05/04/2020

[23] Miniconda https://docs.conda.io/en/latest/miniconda.html Última visita: 23/06/2019

[24] Pyrep https://github.com/stepjam/PyRep Última visita: 05/04/2020

[25] VirtualBox https://www.virtualbox.org/ Última visita: 23/06/2019

[26] Keras https://keras.io/ Última visita: 23/06/2019

[27] TensorFlow https://www.tensorflow.org/ Última visita: 20/08/2020

[28] Adam https://medium.com/@eddydecena/entendiendo-las-redes-neuronales-part-1-fca3adf78c5b#:~:text=Adam,en%20el%20menor%20tiempo%20posible. Última visita: 31/08/2020

[29] Gym-Vrep https://github.com/Souphis/gym-vrep Última visita: 23/06/2019

[30] Funciones de API remota de CoppeliaSim (Python) https://www.coppeliarobotics.com/helpFiles/en/remoteApiFunctionsPython.htm Última visita: 23/06/2019

[31] Aprendizaje por refuerzo con Keras y OpenAI: DQN https://towardsdatascience.com/reinforcement-learning-w-keras-openai-dqns-1eed3a5338c Última visita: 23/06/2019

[32] Pycharm https://www.jetbrains.com/es-es/pycharm/ Última visita: 20/08/2020

[33] Código TFG https://upcomillas-my.sharepoint.com/:f:/g/personal/201601537_alu_comillas_edu/EpBCtH7bKhhKrTjB2zTVlCsBtiWBQlZJj1aJlObs4ZniIQ?e=q37oir Última visita: 31/08/2020

[34] Microsoft Visual C++ Redistributable for Visual Studio 2015, 2017 and 2019 x64 https://support.microsoft.com/es-es/help/2977003/the-latest-supported-visual-c-downloads Última visita: 31/08/2020

ANEXO I

Los pasos a seguir para la instalación de las herramientas necesarias y la configuración del entorno para llevar a cabo este proyecto son los siguientes:

WINDOWS

1) Descarga e instalación de la máquina virtual VirtualBox.

2) Creación de disco duro virtual Linux versión Ubuntu 16.04/18.04

3) Descarga e instalación de Conda o Miniconda (Python 3.6+).

4) Descargar e instalar CoppeliaSim 4.0.0+.

5) Descargar e instalar la biblioteca Pyrep realizando las modificaciones pertinentes del archivo “bashrc”.

6) Descargar e instalar OpenAI Gym.

7) Descargar e instalar la biblioteca gym-vrep (se usa el entorno “Odometry”).

8) Descargar e instalar Keras.

9) Descargar e instalar Matplotlib.

10) Descargar e instalar Pandas

11) Cambiar los archivos sim.py y simconst.py que vienen de serie en Pyrep por los archivos con el mismo nombre que contienen las funciones y constantes de la API remota de CoppeliaSim.

12) Modificar los archivos de la carpeta gym-vrep ubicada en la dirección /home/USER/miniconda3/envs/USER/lib/python3.7/site-packages/gym_vrep-1.1.0-py3.7.egg/gym_vrep según sea necesario para implementar las funciones y algoritmos que se deseen. En este proyecto fue de vital importancia modificar los archivos mobile_robot_navigation.py, navigation_algos.py y smartbot.py para implementar las funciones de la API remota de CoppeliaSim necesarias y el diseño de nuevas funcionalidades.

13) Ejecutar el archivo simple_run.py desde el terminal modificando el main según se desee para la implementación del algoritmo de aprendizaje por refuerzo escogido.

LINUX

1) Descarga e instalación de conda o miniconda (Python 3.6+).

2) Descargar e instalar CoppeliaSim 4.0.0+.

3) Descargar e instalar Pyrep realizando las modificaciones pertinentes del archivo "bashrc".

4) Descargar e instalar OpenAI Gym.

5) Descargar e instalar gym-vrep (se usa el entorno "Odometry").

6) Descargar e instalar Keras.

7) Descargar e instalar Matplotlib.

8) Descargar e instalar Pandas.

9) Cambiar los archivos sim.py y simconst.py que vienen de serie en Pyrep por los archivos con el mismo nombre que contienen las funciones y constantes de la API remota de CoppeliaSim.

10) Modificar los archivos de la carpeta gym-vrep ubicada en la dirección /home/USER/miniconda3/envs/USER/lib/python3.7/site-packages/gym_vrep-1.1.0-py3.7.egg/gym_vrep según sea necesario para implementar las funciones y algoritmos que se deseen. En este proyecto fue de vital importancia modificar los archivos mobile_robot_navigation.py, navigation_algos.py y smartbot.py para implementar las funciones de la API remota de CoppeliaSim necesarias y el diseño de nuevas funcionalidades.

11) Ejecutar el archivo simple_run.py desde el terminal modificando el main según se desee para la implementación del algoritmo de aprendizaje por refuerzo escogido.

ANEXO II: INTEGRACIÓN CON OBJETIVOS DE DESARROLLO SOSTENIBLE (ODS)

Hoy en día son mundialmente conocidas las situaciones de desigualdad, pobreza y condiciones medioambientales desfavorables que por desgracia se sufren en numerosos lugares del planeta. Ante estas situaciones tan desfavorables la ONU (Organización de las Naciones Unidas) puso en marcha en el año 2020 un primer plan para combatir la desigualdad y la exclusión social conocido como "Objetivos de Desarrollo del Milenio" al que solicitaron unirse a todas las naciones del mundo dentro del cual se incluían 8 objetivos: 1) Erradicar la pobreza y la exclusión social 2) Lograr como mínimo una educación primaria universal 3) Lograr la igualdad de género y empoderar a la mujer 4) Reducir la mortalidad infantil 5) Mejorar las condiciones sanitarias de las mujeres embarazadas 6) Combatir el VIH/sida y la Tuberculosis 7) Asegurar la sostenibilidad medioambiental y 8) Lograr una colaboración e implicación de todas las naciones para el desarrollo.

En 2015 se revisó el desarrollo de estos objetivos y, tras realizar una encuesta a nivel mundial para conocer cuál era la importancia que las personas asignaban a cada objetivo, se puso en marcha un segundo plan conocido como "Objetivos de Desarrollo Sostenible" dentro del cual se incluían 17 objetivos: 1) Erradicar la pobreza 2) Erradicar el hambre 3) Mejorar la salud y las condiciones de vida saludable 4) Lograr una educación de calidad 5) Alcanzar la igualdad de género 6) Obtener agua potable 7) Energía limpia y al alcance de todos 8) Mejorar las condiciones de trabajo y asegurar el crecimiento económico 9) Asegurar el avance de la industria, la innovación y las infraestructuras 10) Reducir las condiciones de desigualdad 11) Lograr ciudades y comunidades sostenibles 12) Asegurar la consumición y la producción responsable 13) Tener en cuenta siempre las repercusiones de nuestras acciones para proteger el medioambiente 14) Proteger la vida acuática 15) Proteger la vida terrestre 16) Paz, justicia e instituciones fuertes 17) Colaboración entre los distintos países para alcanzar los objetivos. Estos objetivos se clasifican en objetivos económicos (8, 9, 10, 12), sociales (1, 2, 3, 4, 5, 7, 11, 16) y de la biosfera (6, 13, 14, 15). El objetivo 17 es un objetivo transversal a las tres categorías.

El objetivo con el que se encuentra más estrechamente ligado este proyecto es el objetivo número 9, el cual consiste en "Construir infraestructura resiliente, promover la industrialización inclusiva y sostenible y fomentar la innovación". Se trata de un objetivo esencial para lograr mejorar las condiciones tanto económicas como sociales, así como medioambientales de un país. Un país con industria y sobre todo una industria inclusiva y sostenible que apuesta por la innovación es

un país con futuro. Un país de este tipo contará siempre con amplias garantías de crecimiento económico y tendrá más posibilidades de salir adelante tras cualquier gran desgracia que pueda sufrir.

Una industria inclusiva es aquella en la que se dan las condiciones de trabajo, sanitarias y de seguridad necesarias para que se respete en todo momento la dignidad y el enorme valor humano de sus trabajadores, así como de los candidatos a trabajar en la misma. Situaciones desfavorables contrarias a la industria inclusiva serían jornadas de trabajo demasiado extensas sin posibilidades de conciliar la vida familiar con la laboral, carencia de días de descanso y recuperación semanales y anuales, prohibición de trabajo a mujeres u hombres únicamente por su condición sexual, tendencias sexuales, religión, raza o condición económico-social, explotación de menores, trabajos en los que se ponga en peligro la seguridad e integridad de los trabajadores y en los cuales no se garantice su protección ni cuidados en caso de accidentes, trabajos sin remuneración, etc.

Una industria sostenible es aquella en la que se apuesta por garantizar la protección del medio ambiente y por la utilización de energías limpias, se reduce al máximo el material nocivo para el medioambiente que se necesita para fabricar sus productos y se buscan alternativas más amistosas con el medioambiente, se recicla materia prima y se reutiliza, etc.

Una industria que apuesta por la innovación es una industria cambiante, abierta a la novedad, competitiva y actualizada. Una industria que busca siempre mejorar y abrirse paso en nuevos mercados. Una industria con futuro.

Dentro del objetivo número 9 de desarrollo sostenible presentado anteriormente cabe destacar con mucha relevancia para este proyecto el subobjetivo número 9.5, el cual consiste en “mejorar la investigación científica, mejorar las capacidades tecnológicas de los sectores industriales en todos los países, en particular los países en desarrollo, incluso, para 2030, alentar la innovación y aumentar sustancialmente el número de trabajadores de investigación y desarrollo por cada millón de personas y el gasto público y privado en investigación y desarrollo”.

Este proyecto se encuentra totalmente alineado con el objetivo 9 y en concreto con el subobjetivo 9.5 debido a que el empleo de la técnica de aprendizaje por refuerzo aquí empleada para el desarrollo de un sistema de navegación para un robot móvil es una apuesta clara y concisa por la innovación. Podría incluso afirmarse con seguridad que el algoritmo de aprendizaje por refuerzo es una técnica en sí misma innovadora ya no solo por la época y el contexto en el que se implementa sino también por el desarrollo mismo del algoritmo. Emplear un algoritmo como el que aquí se emplea de aprendizaje por refuerzo significa estar abierto a que el agente descubra por sí mismo la política o comportamiento que le

llevará a realizar una tarea de manera optimizada sin necesitar de injerencias externas de ningún tipo. El agente en este proceso de aprendizaje interactuará con el entorno y lo explorará con el fin de aprender a comportarse en el mismo de la manera más optimizada posible, para lo cual probará de manera aleatoria diferentes combinaciones de acciones en función de los estados y observará en qué situaciones resultará más conveniente para él realizar dichas acciones, de manera que el agente aprenderá que en unas situaciones es más conveniente realizar unas determinadas acciones y en otras situaciones será más conveniente realizar otras distintas.

Al contrario que en los algoritmos de control convencionales en los que se busca que el agente responda a las órdenes directamente marcadas, en este caso no se trata de transmitir ninguna orden de comportamiento al robot sino de darle cierto grado de libertad en sus acciones para que sea él mismo el que, tras interactuar con el entorno, averigüe cuál es en realidad el comportamiento más beneficioso para él en ese entorno.

La responsabilidad de adaptarse al entorno siempre recae sobre el agente y esto puede ser muy beneficioso a la hora de abrir la puerta a la innovación debido a que muchas veces el agente termina exhibiendo comportamientos que no se encontraban inicialmente planeados por el diseñador del algoritmo, sino que son descubiertos directamente por el agente. Esto se debe a que muchas veces este agente llega a poseer un conocimiento más profundo acerca del entorno tras interactuar con él que el conocimiento acerca el entorno que pueda poseer el diseñador de dicho algoritmo cuando lo diseña. La gran ventaja de este algoritmo es que no es necesario que el diseñador del algoritmo posea un conocimiento profundo acerca del entorno, lo que en entornos complejos puede ser difícil, sino que solo se necesita que sea capaz de marcar unas preferencias claras y el robot hará lo posible para orientar sus acciones y adaptarse así a estas preferencias de la manera que resulte más optimizada para él en función de sus posibles acciones y del entorno. Un caso muy ejemplar e ilustrativo de este concepto es el hecho de que un brazo robótico puede aprender una técnica para coger objetos muy distinta a la humana que resulte más optimizada aún para él.

Es por las razones anteriormente mencionadas que esta técnica constituye una apuesta clara por la innovación. Además, el desarrollo de esta técnica solo precisa de un ordenador capaz de llevar a cabo una simulación y esto hace de la técnica una opción muy económica y ajustada al presupuesto de un país en vías de desarrollo.

Gracias al empleo de esta técnica se puede lograr optimizar procesos hasta en un 25%, como se ha demostrado en este proyecto, lo cual constituye una mejora muy relevante, demostrando así ser una técnica económica y eficaz.