Escuela
Politécnica
Superior

Control de robot móvil mediante reinforcement learning

Grado en Ingeniería Robótica

Trabajo Fin de Grado

Universitat d'Alacant
Universidad de Alicante

Autor:
Izan Viñes Castaño
Tutor/es:
Jorge Pomares Baeza
Álvaro Belmonte Baeza

Control de robot móvil mediante reinforcement learning

Autor

Izan Viñes Castaño

Tutor/es

Jorge Pomares Baeza

Departamento de Física, Ingeniería de Sistemas y Teoría de la Señal

Álvaro Belmonte Baeza

Departamento de Ciencia de la Computación e Inteligencia Artificial

Grado en Ingeniería Robótica

Escuela
Politécnica
Superior

Universitat d'Alacant
Universidad de Alicante

ALICANTE, Mayo 2025

Justificación y objetivos

El motivo principal que me ha llevado al desarrollo de este Trabajo de Fin de Grado es mi interés en las tecnologías más actuales y novedosas como la propia inteligencia artificial y sus aplicaciones, especialmente en la robótica y a todo lo que ella engloba. Más concretamente, me resultó muy interesante como mediante una técnica basada en IA, como lo es el aprendizaje por refuerzo, es posible replicar de manera muy fiel como los seres vivos tenemos esa capacidad de poder aprender de nuestro entorno para tomar las mejores decisiones en cada momento. Esta técnica como tal, no se imparte en la titulación de manera oficial, por lo que me pareció un buen ámbito en el que poder indagar e investigar pues todo apunta a que las tareas futuras más complejas serán implementadas por técnicas de control basadas en aprendizaje por refuerzo

El objetivo, por tanto, de este trabajo, se basa en aplicar esta técnica en un robot móvil terrestre. Actualmente este tipo de robots siguen trayectorias predefinidas, ya sea físicamente o a a nivel de software, y en los mejores casos, están dotados de algoritmos que calculan y planifican las trayectorias más optimas para alcanzar un punto en su espacio de trabajo mediante una navegación natural.

Agradecimientos

Me gustaría agradecer a mi familia y a mi pareja por el apoyo que me han brindado en todo momento, y durante toda esta etapa, en la que tomar decisiones complicadas ha sido mucho más sencillo gracias a ellos.

Sé consciente de las limitaciones de tu habilidad,
pero no permitas que afecte la ambición de tu mente.

Michael Jordan.

Índice general

Índice de figuras

Índice de cuadros

1 Introducción

La Inteligencia Artificial (IA) ha inundado las calles de la sociedad actual de una manera sorprendente, convirtiéndose así, en un término que se repite cada vez más. Esto es debido a su gran influencia en múltiples campos y disciplinas como la medicina o la industria entre otros. Sin embargo, la realidad es que los orígenes de la IA tienen lugar a mediados del siglo XX, junto con el nacimiento de los primeros ordenadores en 1943. Ese mismo año, Warren McCulloch y Walter Pitts, publicaron un artículo en el que introdujeron el primer modelo matemático basado en un grupo de neuronas capaces de realizar cálculos lógicos y computacionales, a través de lógica booleana, asentando todo un precedente en las bases teóricas de las ”redes neuronales”, y dando sentido al termino de ”artificial”. Alrededor del año 1950 aproximadamente, el matemático Alang Touring planteó una cuestión que a día de hoy perdura, como es el hecho de si las máquinas son capaces de pensar replicando así el comportamiento humano de tal forma que fueran indistinguibles. Para demostrar la ”inteligencia”, llevó a cabo una prueba conocida como Test de Touring (Turing (1950)), la cual se basaba en la hipótesis de que si una máquina se comportaba de manera inteligente tanto en sus acciones como en sus respuestas, podía ser considerada como inteligente (figura1.1).

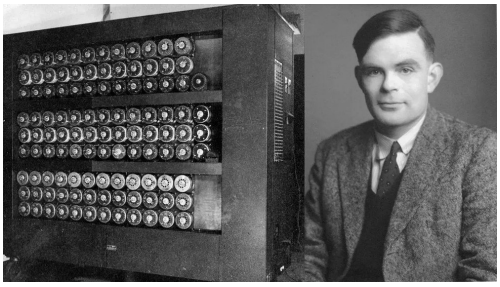


Figura 1.1: Alan Turing y la máquina de Turing

Fuente: `https://pivotal.digital/insights/1936-alan-turing-the-turing-machine`

Poco tiempo después, en 1956, durante la Conferencia de Dartmouth, en Hanover (Estados Unidos), en la que participaron autores como JhonCarthy, Marvin Minsky, Nathaniel Rochester y Claude E. Shannon, se formalizó y se establecieron las bases de este nuevo paradigma de la *inteligencia artificial*. Estos sistemas que se referían a este término eran algoritmos básicos que resolvían problemas determinados, los cuales se basaban en unas ciertas lógicas, pero nunca adaptándose a diferentes situaciones o entornos, ni tampoco aprendiendo de experiencias de manera autónoma. Ese mismo año, ters asistentes a esa misma conferencia, Allen Newell, Helbert Simon y Cliff Shaw, publicaron "Logic Theory Machine", el que se considera como el primer programa informático de inteligencia artificial. Este conseguía, a través de axiomas simples, construir operaciones cada vez más elaboradas y complejas, con el objetivo de demostrar un teorema dado, intentando emular un razonamiento lógico, sin necesidad de hacer una búsqueda exhaustiva de todas las posibles soluciones, llevando a cabo un proceso mucho más eficiente. En 1964, Joseph Weizenbaum, un investigador informático del Instituto de Tecnología de Massachusetts (MIT), creó *ELIZA*, el primer chatbot capaz de mantener conversaciones gracias al procesado del lenguaje natural. Este hito fue muy destacable ya que acercó de una manera realista por primera vez a la IA con los humanos y permitió la interacción entre ambas partes tal y como ya se había vislumbrado anteriormente. Como tal, es considerada la precursora de los chatbots que disponemos en la actualidad como *Siri*, *Alexa* o *ChatGPT*, pues a pesar de ser mucho más avanzados, comparten la funcionalidad de interpretar y simular aquellas respuestas que te podría dar un ser humano.

Esta época justamente coincide con la fecha en la que el primer robot industrial, el *Unimate*, desarrollado por George Devol, fue integrado en una planta de General Motors. Es sencillo ver como la robótica y la Inteligencia Artificial han ido de la mano desde sus respectivos inicios, aunque no fue hasta muchos años más tarde cuando se comenzó a fusionar ambas tecnologías. Esto es así porque dentro del campo de la IA se vivieron unos años, en la década de los 70, conocidos como el "invierno de la IA". En este periodo, se estancó el desarrollo y la investigación por la inteligencia artificial, debido a una falta de financiación y de interés tanto por los grupos de investigación, como por la propia industria y los gobiernos. Las expectativas y las promesas en los años anteriores fueron muy ambiciosas, sin embargo, en ese momento existían numerosas limitaciones técnicas que no permitían poder escalar todos los desarrollos

llevados a cabo hasta la fecha a problemas del mundo real. Este no fue el único momento de declive en el ámbito de la IA, ya que se volvió a repetir una historia similar años más tarde, conocido como el ”segundo invierno de la IA”. La incapacidad para materializar todos los gastos de grandes empresas en beneficios provocó de nuevo que las investigaciones dejaran de recibir financiación, y que la IA, volviera a sufrir un retroceso en su crecimiento.

Los grandes avances en el mundo de la IA comenzaron a florecer en la década de los 90, cuando se comenzó de nuevo a indagar en algoritmos basados en redes multicapa (MLP) que fusionan la lógica de numerosas neuronas para realizar tareas concretas y extrapolables a la práctica. En 1997, ve la luz, una computadora con el nombre de *Deep Blue* y desarrollada por *International Business Machines Corporation* (IBM), la cual fue capaz de derrotar al entonces campeón mundial de ajedrez, Garry Kasparov. Este hito tuvo gran importancia debido a su visibilidad mediática así como por su potencial mostrado a la hora de jugar un juego con tanta complejidad. A partir de 2010, y con la salida de ImageNet, una base de datos *open source* de aproximadamente 14 millones de imágenes, numerosos investigadores continuaron desarrollos tanto previos como nuevos dándose así otra nueva revolución en este campo. Este hito marcó el inició de una era de auge para el *deep learning*, ya que gracias a la gran cantidad de datos etiquetados, se entrenaron redes neuronales capaces de conseguir resultados impresionantes para tareas de visión por computador. El impacto fue tal, que el aprendizaje profundo se extendió a otros campos, como el del procesamiento del lenguaje natural o el reconocimiento de voz, convirtiéndose así en el paradigma sobre el que gira toda la inteligencia artificial actual y futura.

La robótica, como ya se ha mencionado anteriormente, nace de manera simultanea a la inteligencia artificial y crece de forma paralela a ella, sin embargo, nunca han estado tan unidas como en la actualidad. Esto se debe a que la robótica tradicional estaba centrada en la industria, donde se priorizaba los beneficios económicos y donde las aplicaciones se ven muy limitadas por las tareas repetitivas que se desempeñan en este sector. Con el nacimiento de nuevas vertientes como la robótica aplicada a la medicina, la robótica espacial o la robótica social, la IA ha ido introduciéndose cada vez más a diferentes usos. El *reinforcement learning* es una de estas aplicaciones concretas, donde el objetivo principal es realizar el control de una determinada plataforma robótica para desempeñar una tarea concreta. Sus aplicacio-

nes son muy amplias, pues puede ser aplicado a cualquier ámbito de la robótica, desde los brazos antropomórficos industriales tradicionales hasta los robots humanoides recientemente presentados por empresas como Tesla o Boston Dynamics.

Más concretamente, este Trabajo Final de Grado se orientará a la integración de esta tecnología al control de un robot móvil, logrando así poder enviar un robot con estas características a una posición del espacio deseada. Las aplicaciones actuales de este tipo de robots son muy variadas, siendo un muy buen ejemplo la propia robótica industrial, que gracias al reciente surgimiento de los conocidos como *Autonomous Mobile Robots* (AMR), esta sufriendo una revolución total, dónde estos mediante navegación natural han de tener la capacidad de adaptarse a los entornos industriales cambiantes, y colaborar tanto con otros robots como con los propios operarios. Otros ámbitos dónde también toman gran importancia son el espacial, dónde se realizan inversiones millonarias para el desarrollo de nuevos *Rovers* (figura1.2), más eficaces en tareas de exploración, o el propio sector automovilístico, dónde los grandes fabricantes desarrollan sus tecnologías para conseguir una conducción autónoma lo más fiable y segura posible.

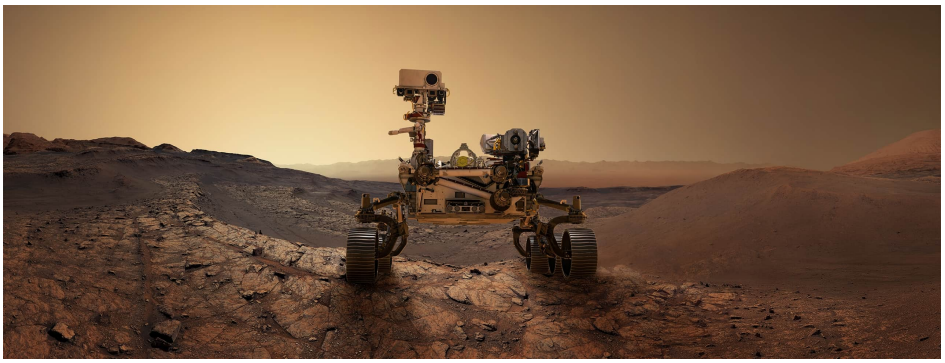


Figura 1.2: Rover Perseverance

Fuente: `https://technetics.com/mars-perseverence-rover-touch-down/`

1.1 Estructura del trabajo

A continuación se detallan las diferentes partes de las que consta el trabajo realizado:

- **Marco teórico** (Capítulo 2): Se detallan los términos y elementos conceptuales que conforman el aprendizaje por refuerzo, necesarios para su entendimiento. Posteriormen-

te se hará un repaso de los métodos actuales y algoritmos utilizados en el mundo de la robótica móvil, y tras ello, se presentará la aplicación concreta del aprendizaje por refuerzo en este ámbito.

- **Objetivos** (Capítulo 3): Se establecen los objetivos del trabajo, desde un punto de vista general como específico.

- **Metodología** (Capítulo 4): Se presentaran las herramientas empleadas para la construcción del entorno y el aprendizaje del robot en simulación centrado en la explicación de los métodos de aprendizaje utilizados para su entrenamiento.

- **Desarrollo** (Capítulo 5): Se presentan los resultados obtenidos de los diferentes entrenamientos bajo diferentes implementaciones y entornos, así como de una explicación detallada de cómo se ha realizado ese proceso de experimentación para encontrar un resultado lo más óptimo posible.

- **Resultados y conclusiones** (Capítulo 6): Finalmente, se cerrará el desarrollo del trabajo con las conclusiones del proyecto, analizando los resultados de una manera más concisa y presentando futuras posibles mejoras e incluso diferentes perspectivas posibles del trabajo abordado.

2 Marco Teórico

En este apartado del trabajo se detallará, en un primer lugar, aquellos conceptos que principalmente son necesarios para el entendimiento del aprendizaje por refuerzo, como lo son la propia inteligencia artificial y el aprendizaje profundo. De esta manera se mostrarán las ideas que sustentan las bases de la tecnología que se va a abordar y su relación con ellas.

Tras la contextualización, se definirá el paradigma del aprendizaje por refuerzo y se citarán las características principales del mismo. Se realizará una breve presentación del estado actual de la robótica móvil en los diferentes campos de aplicación, y cuáles han sido los algoritmos de control clásicos que se han utilizado hasta la fecha. Para concluir este apartado se expondrá la aplicación concreta de un robot móvil controlado mediante aprendizaje por refuerzo y cómo afecta esta integración al control del mismo.

2.1 Introducción y fundamentos

Para poder entender con claridad el aprendizaje por refuerzo, cómo se define, sus principales características y cuál es su funcionamiento, es necesario exponer y comprender las diferentes disciplinas que la engloban. La **inteligencia artificial**, es principalmente un concepto teórico que abarca cualquier desarrollo informático que tenga como objetivo la creación de aplicaciones que puedan imitar comportamientos o habilidades concretas, las cuales entendemos como inteligentes los seres humanos. La IA es mucho más fácil de definir como un sistema experto en una cierta materia específica, cuyo funcionamiento es tomar decisiones basadas en el conocimiento. Estas decisiones aplican reglas de lógica simple en las que si se da una condición concreta, entonces su ”salida” será una acción concreta preprogramada para esa condición, y no otra. Estas reglas vienen definidas por el propio desarrollador, cuyo control viene acotado en su totalidad, por el conjunto de hechos, conceptos, y pautas, que a

su vez definirán el completo comportamiento del sistema.

Este concepto de inteligencia artificial fue el que se adoptó en sus inicios y el cual da sentido al resto de paradigmas que han surgido dentro de el mismo. Sin embargo, es muy limitado ya que, además de depender del conocimiento predefinido, ni manejan incertidumbres en su entrada, por lo que todo es blanco o negro, y no son capaces de aprender de la experiencia, no pudiéndose adaptar a nuevas situaciones, teniendo que ser añadidas esas nuevas reglas manualmente. El ***Machine Learning*** o aprendizaje automático solventa todas esas barreras, dotando a los sistemas de la capacidad de aprendizaje. Este subcampo de la IA, puede desarrollar nuevos algoritmos matemáticos que permitan a la propia máquina aprender, sin ser explícitamente programadas por su desarrollador para dar esa respuesta concreta. La máquina posee la capacidad de extraer patrones de datos sin necesidad de ninguna regla, relacionando los datos de entrada, y utilizándolos para realizar predicciones, tomar decisiones y generalizar comportamientos.

Los datos de entrada son la base fundamental del aprendizaje automático, ya que estos ejemplos previos que conforman el conocido como ”conjunto de entrenamiento”, son los utilizados para el aprendizaje como tal. En función de la aplicación concreta que se requiera diseñar, existen diferentes algoritmos que se ajustan de una mejor forma:

- **Aprendizaje supervisado:** El modelo matemático generado aprende a partir de un conjunto de datos de entrenamiento, que están etiquetados previamente, es decir, los datos de entrada con los que se entrena el modelo son conocidos en su totalidad. El objetivo por tanto pasa por generalizar esas situaciones concretas, para dar respuestas a entradas desconocidas para el modelo. Existen numerosos algoritmos de este tipo de aprendizaje, que optimizan la salida mediante un modelo matemático concreto y representando los datos de entrada de una determinada manera. Algunos ejemplos son la regresión lineal, el *K-Nearest Neighbors* (K-NN) o los *Support-Vector Machines* (SVM). Una de las aplicaciones más conocidas del aprendizaje supervisado es YOLO, utilizado para la detección de objetos en imágenes. Este requiere que el conjunto de entrenamiento este etiquetado por completo, y no solo eso, si no que también requiere de las coordenadas de cada etiqueta, es decir, de cada objeto, para su posterior detección (figura2.1).

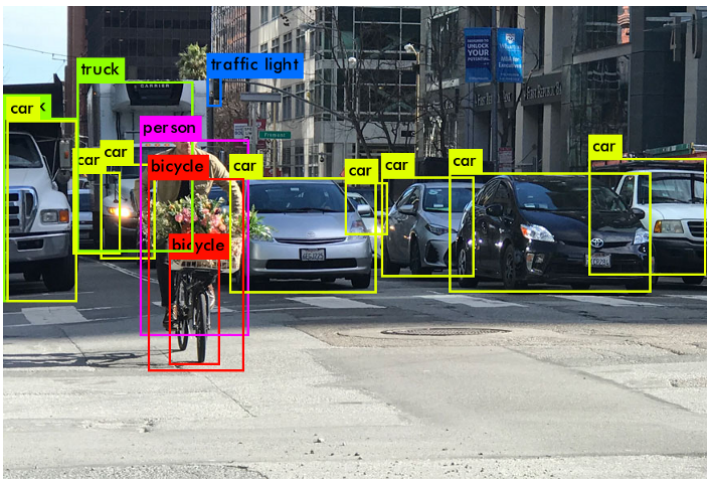


Figura 2.1: YOLO

Fuente: `https://medium.com/analytics-vidhya/yolo-explained-5b6f4564f31`

- **Aprendizaje no supervisado:** Por otro lado, este otro modelo trabaja directamente con conjuntos de datos sin etiquetar, es decir, sin un agente externo que clasifique los datos de entrada. El objetivo recae en encontrar patrones en los datos para clasificarlos en clases, que también son desconocidas para el modelo. Gracias a este patrón es posible encontrar similitudes en grandes cantidades de datos. Para aprender el modelo busca agrupar los datos con características similares. Los algoritmos más comunes son los de *clustering*, en los que se logra dividir el conjunto de entrada en clusters, donde los puntos poseen similitudes entre sí. También existen otros métodos más elaborados como *Principal Component Analysis*(PCA) capaces de reducir la dimensión de los datos y representarlos de manera mucho más sencilla para su posterior análisis de una manera mas sencilla.

- **Aprendizaje por refuerzo:** Este tipo de aprendizaje automático es el más reciente de los tres tipos y nació consagrándose como una de las fuentes de investigación con mayor potencial de aplicación. Debido a su naturaleza, no puede ser definido como supervisado, ya que no parte de datos etiquetados, y tampoco como no supervisado, pues tampoco agrupa conjuntos de datos según ciertas características. El aprendizaje por refuerzo se limita a entrenar a un agente para la toma de decisiones, mediante prueba y error, haciendo que este interactué con su entorno para que consiga maximizar la recompensa

acumulada en función de sus acciones. En la siguiente sección se entrará más en detalle del funcionamiento de este, pues es el paradigma entorno al que gira el desarrollo de este trabajo (Sección 2.2).

Los tres enfoques de aprendizaje automático han seguido avanzando en su desarrollo de manera independiente, sin embargo, todos ellos convergen en la introducción de **redes neuronales** para la mejora de sus integraciones en sus respectivas aplicaciones. Estas se han convertido en un componente indispensable de todo desarrollo del aprendizaje automático moderno. Las redes neuronales, son un paso más para asemejarse a los seres humanos, pues concretamente, simulan la estructura biológica de un cerebro humano. Las nodos, que simulan las neuronas del cerebro humano, son interconectadas y organizadas en capas; de entrada si reciben los datos a procesar, ocultas si llevan a cabo el procesamiento de la información mediante la activación de ellas mismas sgún una función, y de salida si ofrecen un resultado final o una predicción.(figura2.2)

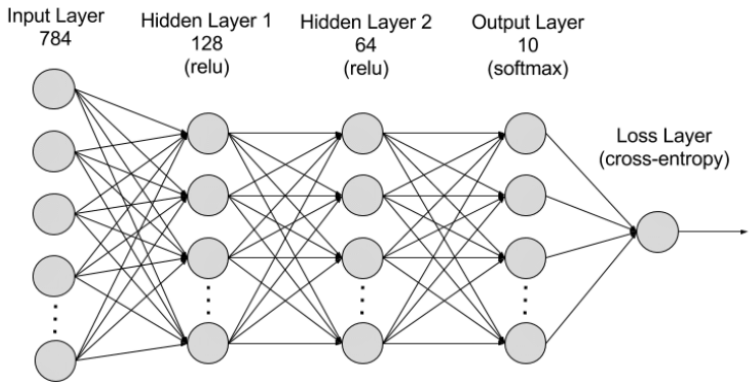


Figura 2.2: Red Neuronal

Fuente: `https://aws.amazon.com/es/what-is/neural-network/`

La estructura de las redes neuronales se asemeja mucho a los grafos ponderados, pues cada neurona de cada capa se conecta a otras con enlaces que tienen un determinado peso, los cuales se ajustan para mejorar los resultados esperados. El criterio de ajuste de estos valores pasa por el proceso de entrenamiento de la red, un mejor ajuste repercute en un funcionamiento más óptimo de la red, y consigo en una mejora en el aprendizaje de patrones. Al mismo tiempo cada neurona dispone de un valor adicional, llamado sesgo (b) que permite

un mejor ajuste. La función que define el *output* de cada neurona es por tanto:

$$z = w_1x_1 + w_2x_2 + \cdots + w_nx_n + b \tag{2.1}$$

Las redes neuronales han ido avanzando y mejorando con el paso del tiempo. Un avance clave de estas fue la integración del *backpropagation*, que le dotaba a la red con la posibilidad de ajustar los pesos automáticamente mediante la diferencia de la salida generada y la esperada, junto con algoritmos de optimización. De esta manera, el error calculado se propagaba hacia detrás con el objetivo de minimizar el error en nuevos **inputs**. Otro de los hitos que ha marcado el devenir reciente de esta ciencia, es el desarrollo del *deep learning* o aprendizaje profundo. Este término hace referencia al uso de redes neuronales con múltiples capas ocultas, las cuales aportan esa "profundidad" a la red. Su objetivo principal es el de abordar problemas mucho más complejos que los planteados con las redes neuronales tradicionales, ya que gracias a ellas, es posible conseguir jerarquizar la propia red, y disolver el problema en partes que son abordadas por la red en diferentes profundidades de la misma. El incremento desorbitado de las cantidades de datos con las que se trabajan en la actualidad, ha provocado que se conviertan en una herramienta imprescindible entre las empresas tecnológicas.

El aprendizaje por refuerzo ha ido beneficiándose e implementando todos estos avances a lo largo del tiempo, logrando evolucionar de un aprendizaje clásico a un aprendizaje profundo, permitiendo a su vez la implementación de entornos de un mayor grado de complejidad, como podrían ser los entornos robóticos, caracterizados por el control de muchos parámetros para un funcionamiento óptimo.

2.2 *Reinforcement Learning*

2.2.1 Introducción y evolución histórica

El aprendizaje por refuerzo, tal y como su propio nombre indica, consiste en lograr que un sistema, en este caso un agente, aprenda de su experiencia tal y como lo hacen los seres vivos. Este aprendizaje se hace de manera autónoma, y busca maximizar la recompensa obtenida en todo momento. El refuerzo viene dado por la recompensa que el agente recibe del entorno al encontrarse en diferentes estados según la cadena de acciones que ha ido ejecutando.

El refuerzo será positivo si el agente hace una toma de decisiones correcta para resolver el problema dado, y negativo en caso contrario. Este concepto es más fácil de entender mediante un símil. Por ejemplo, un perro al cual se quiere adiestrar, será recompensado con premios si recoge y trae la pelota que se le ha lanzado, o si da la pata, y en cambio, si no camina junto a la persona que la ha sacado a pasear, esta le rectificará mediante un tirón en la correa. Estos ejemplos son concebidos como entornos, cuyo objetivo y desarrollo del programador que quiere aplicar este nuevo paradigma al control de un robot, será el de desarrollar y ajustar un entorno a su conveniencia para poder maximizar los resultados esperados e intentar que el robot aprenda un comportamiento que le lleve a desempeñar la tarea en cuestión de la manera más generalizada posible, es decir, intentando conseguir que sea capaz de resolver la tarea en el mayor número de entornos distintos.

La disciplina en cuestión ha evolucionado considerablemente a lo largo de las últimas décadas, desde sus primeras ideas hasta convertirse en una de las áreas más destacadas dentro de la inteligencia artificial. Su desarrollo ha sido impulsado por una serie de hitos teóricos y prácticos marcados por importantes contribuciones de investigadores clave. A pesar de haber sufrido un crecimiento tan alto recientemente, sus inicios se remontan a hace más de medio siglo. Como tal la historia del aprendizaje por refuerzo se divide en tres ramas que, aunque inicialmente fueron independientes, convergieron para dar lugar a lo que conocemos como el aprendizaje por refuerzo moderno. Estas son el aprendizaje por prueba y error, el control óptimo, y la tercera, y un poco menos clara o con menor peso es la de métodos de diferencia temporal.

El control óptimo comenzó a formalizarse en los años 50 de la mano de Richard Bellman, quien extendió una teoría del siglo XIX de Hamilton y Jacobi, para definir problemas de control de sistemas dinámicos. Bellman introdujo la programación dinámica como un enfoque matemático para resolver problemas de toma de decisiones en sistemas dinámicos. La programación dinámica se basa en descomponer un problema complejo en subproblemas más simples que se resuelven de forma secuencial, lo que permite optimizar decisiones a lo largo del tiempo. Concretamente, la programación dinámica estocástica se aplica cuando hay incertidumbre o aleatoriedad en el sistema, y busca encontrar la política óptima (una secuencia de decisiones) para maximizar o minimizar un objetivo (por ejemplo, beneficios o costos).

Estos problemas estocásticos son formalizados matemáticamente como procesos de decisión de Markov (MDP), en los que un agente toma decisiones en un entorno donde el estado cambia de forma probabilística. El objetivo es encontrar una política (función que define qué acción tomar en cada estado) que maximice la recompensa esperada en el largo plazo, dado un conjunto de reglas y probabilidades. Esto se logra resolviendo las ecuaciones de Bellman, que permiten determinar el valor de cada estado y la mejor acción a seguir.

A pesar de que la programación dinámica requiere conocer completamente el sistema para controlarlo, sus métodos iterativos e incrementales se asemejan a los procesos de aprendizaje, lo que lleva a que, en la práctica, sean considerados también dentro del campo de RL.

Por otro lado, la segunda rama de aprendizaje por prueba y error tiene sus raíces en la psicología, particularmente en las teorías de ”refuerzo” que Edward Thorndike desarrolló a principios del siglo XX. Thorndike formuló la *Ley del Efecto*, que establece que las acciones seguidas de resultados satisfactorios se vuelven más probables de repetirse en el futuro, mientras que las acciones seguidas de resultados insatisfactorios tienden a disminuir. Esta idea es crucial en el aprendizaje por refuerzo, ya que implica una combinación de búsqueda (al probar diferentes alternativas) y memoria (al recordar qué acciones fueron más exitosas).

En la inteligencia artificial temprana, investigadores como Marvin Minsky, Farley y Clark (1954) comenzaron a explorar modelos computacionales de este tipo de aprendizaje. Minsky desarrolló máquinas analógicas denominadas SNARCs, que intentaban modelar el proceso de aprendizaje por prueba y error, mientras que Farley y Clark propusieron una máquina de red neuronal para aprender de este modo. Sin embargo, entre los años 60 y 70, el aprendizaje supervisado cobró mayor protagonismo y el aprendizaje por prueba y error perdió visibilidad, sufriendo un estancamiento durante esta etapa.

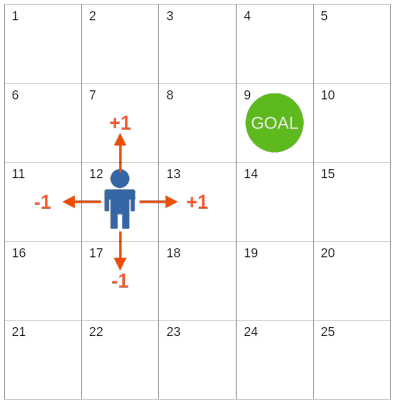
A partir de la década de los 70, el aprendizaje por refuerzo renació de la mano de Harry Klopf, pues entendió como los esfuerzos y avances de la IA se centraban en problemas como la clasificación supervisada o la resolución de problemas determinísticos. La capacidad de aprender de la interacción con un entorno dinámico y cambiante, a través de un proceso de prueba y error, no estaba siendo adecuadamente explorada, haciendo que profundizara en el concepto de comportamiento adaptativo. Klopf centró sus esfuerzos en estudiar como controlar el entorno para poder obtener resultados deseados. Su trabajo influyó fuertemente en

otros investigadores como Barto y Sutton, y ayudó a clarificar la distinción entre el aprendizaje supervisado y el aprendizaje por refuerzo.

Los métodos de diferencia temporal (TD) representan el tercer hilo que une los conceptos anteriores. Estos métodos se basan en la diferencia entre las predicciones sucesivas del mismo valor en diferentes momentos temporales (por ejemplo, la probabilidad de ganar en un juego, que aumenta y disminuye con el paso de la partida). Este método permite al agente ajustar en cierta manera sus decisiones de manera continua, basándose en esas diferencias entre predicciones actuales y futuras de los valores esperados de las acciones. El trabajo pionero de Arthur Samuel en 1959 con su programa de ajedrez ya incluía ideas de TD, pero el campo no despegó hasta que Barto y Sutton, mencionados anteriormente, refinaron las ideas y las aplicaron al aprendizaje por refuerzo en la década de 1980.

Poco tiempo después, en 1989, todas estos paradigmas fueron utilizados en conjunto por Chris Watkins, proponiendo el algoritmo *Q-learning*, una técnica que permite a los agentes aprender a maximizar su recompensa a largo plazo sin tener que conocer el modelo del entorno. Este algoritmo utiliza el concepto de ”valor Q”, que asocia un valor a las acciones en los estados, y ajusta esos valores a medida que el agente interactúa con el entorno. Esto impulsó significativamente la investigación en el campo y consolidó el aprendizaje por refuerzo tradicional como una técnica clave para resolver problemas de toma de decisiones. Sin embargo, este método presentaba una serie de limitaciones, pues por ejemplo la función Q, la cual asigna los valores de recompensas esperadas en pares de estado-acción, se almacena en una tabla, lo que limita su uso a espacios de estado pequeños o discretos debido a problemas de escalabilidad. Además, este enfoque carece de capacidad de generalización, pues la tabla utilizada es fija. Años más tarde surgieron nuevos algoritmos, que solucionaban todos estos problemas, y que introducían redes neuronales profundas para aproximar las funciones, sin necesidad de utilizar tablas fijas que modelaran dicha función. Este nuevo enfoque, el aprendizaje por refuerzo profundo, aportó la capacidad de manejar espacios de estado mucho más grandes, complejos y lo que es más interesante, continuos. La evolución del enfoque tradicional al enfoque profundo o automático, se produjo gracias a la introducción de las redes neuronales, permitiendo generalizar y aprender patrones complejos, haciendo que el abanico de posibles aplicaciones creciese considerablemente y su uso fuera más generalizado entre

grupos de investigación.



	↑	↓	←	→
1	-	+1	-	+1
2	-	+1	-1	+1
3	-	+1	-1	+1
4	-	+1	-1	-1
5	-	+1	+1	-
...				
23	+1	-	-1	+1
24	+1	-	-1	-1
25	+1	-	+1	-

Figura 2.3: Ejemplo de *Q-Table* descrita para cada par de estado-acción

Fuente: `https://blog.spiceai.org/posts/2021/12/15/understanding-q-learning-how-a-reward-is-all-you-need/`

En resumen, es posible definir el aprendizaje por refuerzo de una manera formal como una estructura diseñada para resolver aplicaciones de control, o problemas de toma de decisiones, construyendo agentes que aprenden del entorno, interactuando con él, a través de prueba y error, y obteniendo recompensas, positivas y negativas, por dichas acciones que toma, y siendo estas el único *feedback* del que dispone dicho agente para aprender. Su variante ”profunda” consigue evolucionar este paradigma introduciendo redes neuronales profundas a la hora de aproximar y optimizar las funciones y políticas, sin necesidad de tablas, expandiendo las posibles aplicaciones de una forma muy considerable abordando problemas continuos y mucho más grandes y escalables. A continuación se detallarán las partes por las que está constituido y definido el aprendizaje por refuerzo:

2.2.2 Elementos del Reinforcement Learning

En este apartado del trabajo se abordarán los diferentes elementos específicos que componen el aprendizaje por refuerzo en su totalidad y cómo afectan al conjunto de este.

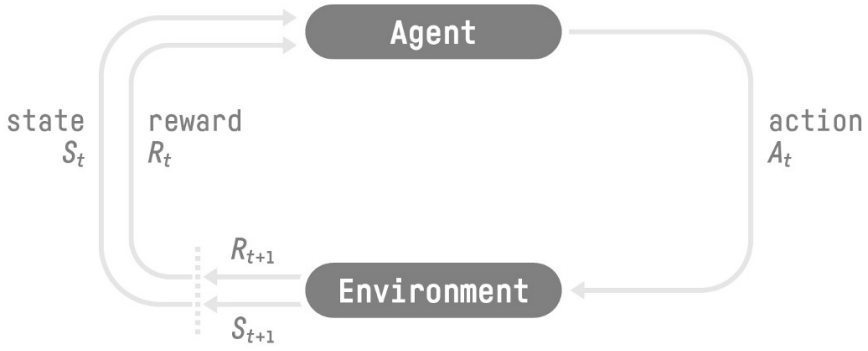


Figura 2.4: Esquema general de aprendizaje por refuerzo

Fuente: `https://huggingface.co/learn/deep-rl-course/unit1/rl-framework`

2.2.2.1 Agente

El agente se presenta como la parte más importante del aprendizaje por refuerzo, pues es el encargado de aprender de manera autónoma qué decisión tomar en cada momento, la cual maximice la recompensa obtenida, es decir, sea la mejor decisión y más eficiente para encontrar el camino a la resolución de la problemática dada. Todo ello, lo llevará a cabo en función de las recompensas o castigos que el entorno le da por interactuar con este tomando una determinada acción. El agente posee memoria, pues gracias a ella es capaz de aprender de acciones y estados pasados, para no cometer los mismo ”errores”. Como tal, el proceso interno que ejecuta para deducir qué acción tomar ante cada estado recibe el nombre de política. El objetivo de este más concretamente, será encontrar cuál es la política que maximice sus recompensas. La política es actualizada y evoluciona con el entrenamiento del agente, utilizando las acciones pasadas, sus respectivas recompensas obtenidas del entorno y las observaciones.

En definitiva, el agente es aquello que queremos que aprenda, es decir, que consiga descubrir

una función optimizada π*(estado)* que maximice el resultado esperado al actuar aplicándola a los diferentes estados en los que se va encontrado el propio agente.

2.2.2.2 Entorno e interacción

Todo agente ocupa un lugar en un espacio determinado. Este espacio en el que el agente desempeña sus acciones recibe el nombre de entorno. La principal función del entorno es la de definir qué recompensas son concedidas al agente en función de las acciones tomadas y al estado en el que se encuentra. Gracias al modelado del entorno, el agente puede interactuar de manera bidireccional con él ejecutando sus acciones y recibiendo la respuesta acorde.

En el desarrollo del trabajo el entorno será el simulador de *Mujoco*, en el cual encontraremos un punto objetivo rojo y el propio robot; y a su vez, la librería de *Gym* que proporciona las herramientas para poder modelarlo indicando las recompensas y otros parámetros.

A su vez, y muy relacionado con el entorno y las interacciones del agente con él, se debate una problemática la cual todo agente debe afrontar en su toma de decisiones. Este recibe el nombre del dilema entre explotación y exploración. La exploración del entorno se centra en que el agente lleve a cabo acciones aleatorias con el objetivo de encontrar más información sobre el propio entorno. Por otro lado, la explotación se basa en maximizar de manera recurrente esa información conocida del entorno que maximiza la recompensa obtenida. Si la observación que tiene el agente del entorno es limitada, este se centrará en aquellas fuentes de recompensa cercanas o fáciles de alcanzar a pesar de que la recompensa obtenida sea pequeña, en comparación a otras posibles recompensas que podría encontrar en el entorno. El objetivo de todo esto es encontrar ese balance óptimo entre ambos enfoques, ya que es igual de necesario obtener recompensas para maximizar la respuesta acumulativa, como explorar el entorno en busca de nuevas fuentes de recompensas desconocidas que quizá otorguen al agente unos valores mayores y que haría a su vez que saliera de ese mínimo local de recompensas en el que se hallaba bajo un comportamiento tan poco atrevido.

2.2.2.3 Estados y observaciones

Como la propia palabra indica, los estados son las representaciones en las que se encuentra el agente en cada instante de tiempo, dentro del entorno, con unos valores determinados. Estos

pueden ser desde la posición actual del robot, su orientación, los valores de los sensores... Esta descripción del entorno en un momento dado se representa como S_t, y contiene consigo toda la información sobre el entorno sin excepciones, necesaria para que el agente pueda seguir aprendiendo y tomando decisiones para maximizar la recompensa. Es imprescindible recalcar que el estado es una descripción completa del entorno, y que no hay nada de información oculta. En cambio, si nuestro entorno es parcialmente visible, hablamos de una observación en su lugar, es decir, es una descripción local del estado del mundo.

El estado es clave para llevar a cabo las transiciones de nuestro sistema, cada vez que el agente toma una decisión siguiendo las reglas que predefinen el entorno, cambiará de un estado a otro. Las recompensas asociadas a estas transiciones ayudará al agente a decidir qué estados son deseables y cuáles no.

2.2.2.4 Acciones

Una acción representa la decisión tomada, llevada a cabo por el agente, en un estado concreto para hacer la transición a un nuevo estado, siguiendo siempre el criterio establecido por la política que se está entrenando en su propio ”cerebro”. Estas acciones se definen como a_t, donde t es el instante de tiempo en el que se toma dicha acción. Todas las posibles acciones que existen en un entorno concreto reciben el nombre de espacio de acciones, y pueden ser de dos tipos en función del problema a resolver e inclusive las limitaciones dinámicas del agente:

- **Espacio discreto:** es aquel cuyo número de posibles acciones es finito. Este tipo de espacios se suelen dar en los videojuegos, como por ejemplo en Super Mario Bros, unicamente tenemos 4 posibles acciones: izquierda, derecha, arriba (saltar) y abajo (agacharse).
- **Espacio continuo:** en este caso, el número de posibles acciones que puede llevar a cabo el agente es infinito. Este tipo suele ser más típico de problemas del mundo real, como por ejemplo un coche autónomo, que ha de decidir cuantos grados ha de girar en cada instante de tiempo. Los grados que ha de girar está acotado en el espacio de los números reales, por lo que podrá tomar valores infinitos. El entorno desarrollado en este trabajo también pertenece a este tipo de espacio, pues el robot deberá decidir entre números reales para su giro, y su avance o retroceso.

2.2.2.5 Recompensa

Otro de los elementos fundamentales en aprendizaje por refuerzo es el de la recompensa, pues este, es el único *feedback* que el agente recibe como resultado del entorno por su toma de decisiones, y de qué tanto bien o mal lo está haciendo. Es uno de los parámetros que la política tiene en cuenta para poder tomar esa decisión e ir ajustándose a medida que trascurren instantes de tiempo. Es crucial diseñar una buena función de recompensa, que cuantifique lo bien que el agente desempeña su objetivo en el entorno. Esta recompensa será un valor escalar, que conforme mejor sea el resultado obtenido de las acciones del agente, el valor de esta será mayor, y cuanto peor sea, más bajo será este valor, pudiendo ser incluso hasta negativo en muchas ocasiones. La función de recompensa será acumulativa a lo largo del tiempo y a su vez, el valor en cada instante, dependerá de la acción efectuada en ese momento y el estado en el que se encuentra. La forma general de representarla es la siguiente:

$$R(\tau) = r_{t+1} + r_{t+2} + r_{t+3} + r_{t+4} + \cdots \tag{2.2}$$

dónde r_t son cada una de las recompensas devueltas en cada instante, τ es la trayectoria o secuencia de estados y acciones, y cuya función tiene como resultado global devolver la recompensa acumulada. Esta función también se puede reescribir como:

$$R(\tau) = \sum_{k=0}^{\infty} r_t + k + 1 \tag{2.3}$$

La función de recompensa por tanto se describe matemáticamente como el sumatorio de las recompensas en cada instante de tiempo, y donde t es la duración del episodio.

Sin embargo, en la práctica no es posible añadir la función de recompensa de esta manera, puesto que está incompleta. Aquellas recompensas que ocurren antes en nuestro entorno, son más propensas a que ocurran haciendo que sean mucho más predecibles que aquellas recompensas a largo plazo. La idea de esto es compensar este dilema, descontando los valores futuros a través de una componente γ. Aquellas recompensas lejanas o muy futuras, a pesar de ser mayores, serán más descontadas por este parámetro al no estar tan seguros si conseguiremos alcanzarlos debido a la incertidumbre del propio entorno. El valor para que sea efectivo tomará valores de entre 0 y 1, normalmente un 0.95. Si aumentamos el valor del

parámetro, el descuento será menor, y a su vez el agente se centrará más en las recompensas más lejanas. Si por consiguiente lo disminuimos, el agente se centrará en las recompensas más a corto plazo. Todo ello conlleva el modelado de la incertidumbre futura afecte en las recompensas que nuestro entorno da como resultado. La función por tanto se reescribiría de la siguiente forma:

$$R(\tau) = r_{t+1} + \gamma r_{t+2} + \gamma^2 r_{t+3} + \gamma^3 r_{t+4} + \cdots \tag{2.4}$$

$$R(\tau) = \sum_{k=0}^{\infty} \gamma^k r_t + k + 1 \tag{2.5}$$

2.2.2.6 Episodios y tareas

Otros elementos relevantes dentro del aprendizaje por refuerzo son los episodios y las tareas. Los episodios, son secuencias completas de interacciones entre el agente y el entorno, que parten de un estado inicial y que consiguen alcanzar un estado final o terminal. Esto no siempre es del todo así, puesto que existen tareas a resolver cuya naturaleza impiden que tenga un estado terminal y sean continuas de manera infinita. En este caso el agente debe aprender a interactuar de manera continua eligiendo siempre la mejor acción disponible en cada momento, sin un objetivo como tal.

Sin embargo, en la mayoría de ámbitos y ocasiones, si que existe ese estado terminal o final, dando como resultado un episodio constituido por una serie de estados, acciones, recompensas y nuevos estados. Este estado terminal ha de ser definido en el entorno, y corresponderá con el objetivo final de la tarea a realizar. También cabe destacar, que es posible generar estados que trunquen el entrenamiento en caso de que no se alcance el objetivo, consiguiendo así evitar bucles infinitos o la búsqueda del agente de una política que diste mucho de la óptima.

2.2.2.7 Política

La política formalmente es la función de la cual el agente depende para comportarse de una determinada manera en un estado concreto, es decir, es el cerebro del agente. Mediante el entrenamiento se espera que la política se asemeje lo máximo posible a la función objetivo que se desea aprender, siendo esta aquella política que maximice la recompensa esperada

cuando el agente actúa bajo sus directrices. Existen dos tipos de política en función de su resultado:

- **Determinista:** se le considera de esta manera a aquella política que dado un estado concreto devuelve siempre la misma acción a realizar.

$$a_t = \pi(s_t) \tag{2.6}$$

- **Estocástica:** aquella función que devuelve como resultado una función de probabilidad del conjunto de acciones posibles a realizar por el agente en ese estado. El resultado por tanto no es siempre el mismo ni exclusivo de ese estado. Esta manera modela mejor los problemas extrapolados al mundo real, pero es mucho más compleja de encontrar pues introduce la incertidumbre del propio entorno.

$$\pi(a_t|s_t) = P[A|s] \tag{2.7}$$

2.2.3 Procesos de decisión de Markov

El marco matemático en el qué se sustenta el aprendizaje por refuerzo recibe el nombre de procesos de decisión de Markov (MDP). Este modelado, es empleado para representar problemas donde un agente toma decisiones en un entorno dinámico con el objetivo de maximizar recompensas acumuladas a lo largo del tiempo, llevando a cabo transiciones entre diferentes estados, siguiendo unas reglas exactas (determinista), o bien una distribución de probabilidades (estocástico). Los MDPs son clave para formalizar esa interacción agente-entorno, junto con todos los elementos mencionados en el apartado anterior, para así poder representar el paradigma del aprendizaje por refuerzo. La principal característica de este modelado matemático es que cumple la propiedad de Markov, la cual indica que la transición al siguiente estado, depende únicamente del estado actual y la acción tomada, no de los acontecimientos pasados.

De manera general, los elementos que definen formalmente un MDP vienen definidos como

una tupla y son los siguientes:

$$MDP = (S, A, P, R, \gamma) \tag{2.8}$$

- S**: Conjunto de estados posibles.** Representa todos los posibles estados en los que se puede encontrar el sistema en un momento determinado. Un estado $s \in S$ contiene toda la información necesaria para tomar una acción que maximice la recompensa acumulada esperada.

- A**: Conjunto de acciones.** Todas las acciones disponibles para ejecutar por parte de nuestro agente, utilizadas para lograr el objetivo.

- $P(s'|s, a)$**: Modelo de transición.** Describe la distribución de probabilidad sobre los posibles estados futuros, es decir, la probabilidad de avanzar de un estado s a un estado s' tomando una acción cualquiera a.

- $R(s, a, s')$**: Función de recompensa.** Es la recompensa inmediata que recibe el agente del entorno al moverse entre estados. Puede depende unicamente del estado actual y de la acción $R(s, a)$.

- $\gamma \in [0, 1]$**: Factor de descuento.** Como ya se ha mencionado en el apartado 2.2.2.5, representa cómo se balancea las recompensas futuras en favor de las inmediatas para la toma de decisiones del agente.

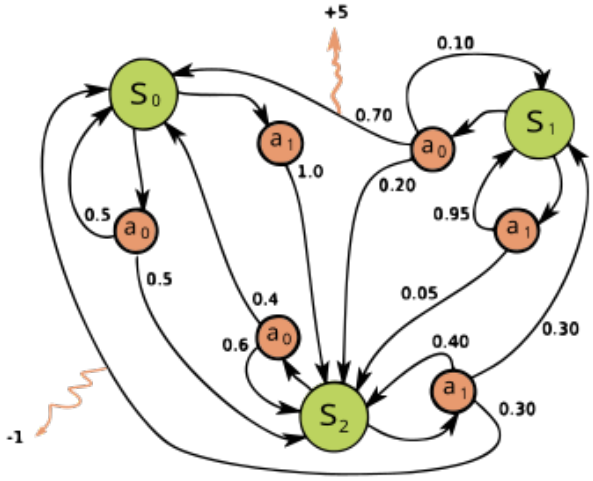


Figura 2.5: Representación de un MDP

Fuente: `https://www.cs.us.es/~fsancho/Cursos/SVRAI2122/MDP.md.html`

2.2.4 Política óptima, funciones de valor y ecuación de Bellman

Como se ha mencionado anteriormente en el apartado (2.2.2.7), el objetivo del aprendizaje por refuerzo pasa principalmente, por encontrar una política óptima π^* que maximice el retorno de la recompensa esperada, cuando el agente actúe según esta. Descrito formalmente se tiene la siguiente expresión:

$$\pi^* = \max_{\pi} \mathbb{E}(\pi) \tag{2.9}$$

donde $\mathbb{E}(\pi)$ es el retorno esperado en un episodio siguiendo esa política.

La gran parte de algoritmos empleados en aprendizaje por refuerzo se basan en estimar una función que materialice la calidad de un estado, o bien, de un par estado-acción. Esta función recibe el nombre de función de valor (2.10), y se define con respecto a las políticas del agente. Formalmente, la función de valor de un estado s dada una política π se denota como $v_\pi(s)$ y es igual al retorno esperado desde el estado s, si nuestro agente sigue la política

π hasta finalizar el episodio.

$$v_\pi(s) = \mathbb{E}_\pi[G_t \mid S_t = s] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right] \tag{2.10}$$

donde $\mathbb{E}_\pi$ es el valor o retorno esperado siguiendo dicha política π, o en otras palabras, la suma de todas las recompensas en un episodio. Esa política seguida puede ser o no la óptima, siendo una cualquiera.

También es posible aplicar esta misma formulación matemática, para el caso en el que se requiera definir el valor de una acción a tomar a, en el estado s, siguiendo una política cualquiera π como:

$$q_\pi(s, a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a] = \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a \right] \tag{2.11}$$

Más concretamente, encontramos 4 tipos distintos de funciones de valor, si bien se rigen por la política óptima o no, y si el valor viene dado por un estado o por un par estado-acción:

- ***On-Policy Value Function***: es la función más simple, la cual devuelve el retorno esperado si el entorno empieza en el estado s y se actúa según la política π cualquiera. Este tipo hace referencia a la ecuación (2.10)

- ***On-Policy Action-Value Function***: Hace referencia a la ampliación de la función de valor anterior (*On-Policy Value Function*) para un par estado-accion. Su definición matemática viene dada por una de las ecuaciones ya mostradas (2.11)

- ***Optimal Value Function***: Devuelve el retorno esperado si el entorno empieza en el estado s y se actúa según la política óptima.

$$v_{\pi^*}(s) = \max_\pi \mathbb{E}_\pi[G_t \mid S_t = s] = \max_\pi \mathbb{E}_\pi \left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s \right] \tag{2.12}$$

- ***Optimal Action-Value Function***: Devuelve el retorno esperado si el entorno empieza en el estado s, se realiza una acción arbitraria a y tras ello comienza a actuar conforme

a la política óptima.

$$q_{\pi^*}(s,a) = \max_{\pi} \mathbb{E}_{\pi}[G_t \mid S_t = s, A_t = a] = \max_{\pi} \mathbb{E}_{\pi}\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \mid S_t = s, A_t = a\right] \tag{2.13}$$

Para calcular esta serie de valores definidos por la función de valor de una política, se utiliza la conocida como ecuación de Bellman (2.14). Se emplea la propia ecuación en un estado futuro s', y a su vez, hace uso de la política utilizada, sin ser necesariamente la óptima, y de la probabilidad de transición de un estado s a uno s', que puede ser conocida o no.

$$v_{\pi}(s) = \sum_{a} \pi(a \mid s) \sum_{s',r} p(s', r \mid s, a) \left[r + \gamma v_{\pi}(s')\right] \tag{2.14}$$

2.2.5 Clasificaciones del Reinforcement Learning

El aprendizaje por refuerzo, al igual que otros paradigmas similares, ha ido desarrollándose en diferentes vertientes a lo largo del tiempo, implementando diversos enfoques y aplicando muchas técnicas en función del problema, el entorno y sus requisitos entre otros factores. En este apartado del trabajo, se expondrán las diferentes y más recurrentes técnicas utilizadas a la hora de entrenar un agente y se indicaran las principales clasificaciones dentro del aprendizaje por refuerzo.

2.2.5.1 Clasificación según aprendizaje y políticas

Como ya se ha definido en el apartado (2.2.2.7), la política de nuestro agente toma vital importancia al ser la encargada de definir el comportamiento de este dentro de un entorno en concreto. Según qué aprende el agente es posible diferenciar varios tipos:

- **Métodos Policy-Based:** en este tipo de métodos el agente aprende directamente una política, es decir, qué acción es óptima en cada uno de los estados posibles. Estas acciones no tienen porque ser exclusivas, si no que también es posible que lo que se aprenda es una distribución de probabilidad de posibles acciones en cada uno de estos mismos estados.

- **Métodos Value-Based:** este otro método busca aprender una función de valor, en lugar de una función de política, que por contra, indique cómo de buena es una acción o un estado para maximizar la recompensa acumulada. Concretamente esta función asigna un valor esperado determinado a un estado en caso de estar en él. Un ejemplo que ya se ha mencionado anteriormente de este tipo de aprendizaje es el *Q-Learning*, pues se aprende una función *Q(s,a)* que da como resultado el valor esperado de una acción *a* estando en un estado *s*.

- **Métodos Actor-Critic:** el método en cuestión está basado en los dos anteriores y combina ideas de los anteriores comentados. Este dispone de un actor que aprende la política, es decir, sobre él recae el peso de decidir que acción tomar en base de las observaciones recibidas. Además, de manera paralela hay un crítico que evalúa la calidad de la acción tomada por el actor, utilizando una función de valor clásica. El más famoso de este tipo de algoritmos es el *A2C*.

Por otro lado, según cómo aprende el agente y cómo utiliza las políticas para aprender, encontramos otros dos tipos de clasificación:

- **Métodos On-Policy:** el agente aprende y evalúa al mismo tiempo la misma política que está usando para tomar decisiones, siendo idénticas la utilizada para explorar ejecutando acciones y la optimizada, que es la que se está aprendiendo.

- **Métodos Off-Policy:** el agente en este caso utiliza políticas distintas para aprender y para tomar decisiones. Por ejemplo sería el caso de un robot que aprende a moverse mientras toma decisiones aleatorias a la hora de realizar una acción determinada para explorar.

2.2.5.2 Clasificación según el tipo de entorno y algoritmo utilizado

- **model-based:** En un entorno *model-based*, el agente tiene acceso al modelado del entorno. Esto significa, que el agente utiliza aplica un algoritmo en el que la probabilidad de transición entre estados es conocida, así como las recompensas obtenidas en cada uno de ellos. Gracias a esta información conocida, el agente puede planificar explícitamente

sus acciones resolviendo la ecuación de Bellman (2.14) para calcular las funciones de valor y encontrar la política óptima. Los algoritmos más destacados son:

- ***AlphaZero*** (Silver y cols. (2018))
- ***Model Predictive Control (MPC)*** Qin y Badgwell (1997)
- ***Model-Based Value Estimation (MBVE)*** (Feinberg y cols. (2018))
- ***I2A*** Racanière y cols. (2017)

- **model-free:** En este otro tipo de entornos, el agente no conoce el modelo del entorno. Por tanto, el agente no conoce la probabilidad de transición entre estados, ni la estructura del entorno, ni la función de recompensa que lo define. El agente, mediante un conocimiento muy limitado, es capaz de conseguir aproximar la política óptima buscada, pues no puede calcular de forma explicita la ecuación de Bellman. Buscan aprender las consecuencias de sus acciones a través de la experiencia y los resultados obtenidos a raíz de esta interacción.Este tipo de algoritmos son aplicables a entornos dinámicos, donde la incertidumbre es mayor, y el propio entorno está sujeto a constantes cambios. Estos se asemejan más al mundo real, y a las aplicaciones que pueden ser trasladadas a este, como podría ser la que ocupa este trabajo. Los algoritmos *model-free* más recurrentes son:

  Basados en *Q-Learning*

  - ***DQN*** (Mnih y cols. (2013))
  - ***HER*** (Andrychowicz y cols. (2017))

  Basados en *Policy Optimization*

  - ***A2C*** (Mnih y cols. (2016))
  - ***PPO*** Schulman y cols. (2017)

  Enfoque mixto

  - ***DDPG*** (Lillicrap (2015))
  - ***SAC*** (Haarnoja y cols. (2018))
  - ***TD3*** (Fujimoto y cols. (2018))

2.2.6 *Deep Reinforcement Learning (DRL)*

El hito más revolucionario del aprendizaje por refuerzo en los últimos años ha sido la introducción de redes neuronales profundas para estimar de una manera más eficiente la función de la política óptima buscada (*policy-based*),la función de valor (*value-based*) que devuelve un valor para cada par de estado-acción como ya se ha visto. Este momento se produjo exactamente en 2013 con la publicación del algoritmo *DQN* (Mnih y cols. (2013)). Este nuevo planteamiento dio paso a nuevas lineas de investigación relacionadas con la materia, y se convirtió en la base del aprendizaje por refuerzo moderno. Tanto es así, que la gran parte de nuevos algoritmos surgidos con fecha posterior a esta, se basan principalmente en este nuevo planteamiento.

La combinación de redes neuronales permite manejar espacios de estado y acciones mucho mayores, resolviendo así el problema de dimensionalidad que impedía aplicar métodos de aprendizaje por refuerzo en entornos con espacios de estado muy grandes o complejos, como lo pueden ser problemas de robótica moderna. Además, gracias a las propias características del aprendizaje profundo, los agentes son capaces de ser más flexibles, y adaptarse a un número más amplio de situaciones, así como de manejar entornos dinámicos y con mayor incertidumbre como propiamente son las aplicaciones del mundo real.

2.3 Desarrollo y control clásico de la robótica móvil

La robótica móvil es una rama concreta de la robótica enfocada al diseño, desarrollo y el control de robots, que tienen como característica principal e inequívoca, el hecho de que son capaces de moverse de manera autónoma en los entornos que les rodean. La gran parte de estos robots operan en entornos dinámicos, siendo imprescindible que dispongan de herramientas para adaptarse a cambios constantes y la aparición de nuevos obstáculos. La capacidad de interactuar con entornos cambiantes provoca que su aplicación al mundo real se haga de manera mucho más efectiva, sin necesidad de modificar su entorno si no que este se adapte a él, a diferencia de la robótica tradicional. Esta tipología de robots hacen uso de actuadores, sensores y diferentes algoritmos para navegar por el entorno en el que se implementan y llevar a cabo tareas específicas dentro de él. Sus aplicaciones son muy variadas, desde entornos

industriales como la logística, donde se utilizan los conocidos AGVs (*Autonomous Guided Vehicles*) y AMRs (*Autonomous Mobile Robots*) donde desempeñan funciones de transporte de mercancías en almacenes y fábricas; hasta otros campos como el de la exploración espacial, con Rovers para la recolección de datos; o la agricultura, para monitoreo, recolección y fumigación de cultivos.

En este apartado del trabajo se van a definir las bases de la robótica móvil, sus características, los diferentes aspectos que abarca, y se expondrán cuales son los algoritmos utilizados de manera más recurrente en este ámbito.

2.3.1 Modelado y cinemática de un robot móvil

Cuando se menciona la robótica móvil, es necesario hablar sobre la interacción física del propio vehículo con su entorno, la cual lleva a cabo con el objetivo de conseguir un desplazamiento. Esta interacción es complementaria a la manipulación y recibe el nombre de locomoción, donde el entorno está fijo, que no significa que no pueda estar sometido a cambios, y es el robot el que se mueve respecto de él.

Para la realización de este trabajo se centrará el estudio de la robótica móvil donde los actuadores, que otorgan la posibilidad de movimiento al robot, sean exclusivamente ruedas. Estas ruedas deben estar configuradas bajo una configuración determinada, o en caso contrario, el vehículo no podrá desplazarse. Debe existir por tanto un punto concreto sobre el que se realice el giro del robot, llamado centro de curvatura instantáneo (CCI). No todas las configuraciones de las rueda permiten un movimiento correcto, pero si que existen varios modelos de configuración para el CCI, como por ejemplo uno en el que coincida la intersección de todos los ejes de las ruedas.

Existen diferentes combinaciones según la configuración y disposición de los actuadores en el diseño de los vehículos. De manera general, las configuraciones en robótica móvil no permiten al robot alcanzar todas las posiciones de su espacio de trabajo de manera directa, por lo que se hablará de restricciones no holonómicas. El ejemplo más claro es el de un coche, el cual puede alcanzar posiciones en un plano XY, sin embargo para hacerlo, no se puede mover lateralmente por lo que tiene que maniobrar.

2.3.1.1 Modelado cinemático

El modelado cinemático describe el movimiento del robot sin considerar las fuerzas o torques aplicados, centrándose en su geometría y restricciones de movilidad. Existen gran variedad de tipos de configuraciones de robots móviles, cada una de ellas con su propia ecuación de movimiento. Las principales son las siguientes:

- **Modelo diferencial**: se basa en dos ruedas motrices controladas de forma independiente, generalmente acompañadas por una o más ruedas pasivas (también llamadas castor) para mantener el equilibrio. La dirección del movimiento se determina por la diferencia de velocidad entre las dos ruedas motrices, girando hacia el lado donde la rueda posea una menor velocidad angular. Este es uno de los modelos más usados por su simplicidad mecánica y su facilidad de control, pero presenta grandes limitaciones en cuanto a su movimiento por lo que su aplicación real es también limitada. Las ecuaciones que definen el movimiento de un robot diferencial son las siguientes:

$$\dot{x} = v\cos\theta \tag{2.15}$$

$$\dot{y} = v\sin\theta \tag{2.16}$$

$$\dot{\theta} = \omega \tag{2.17}$$

Donde la velocidad lineal v y la velocidad angular ω están definidas como:

$$v = \frac{v_r + v_l}{2} \tag{2.18}$$

$$\omega = \frac{v_r - v_l}{L} \tag{2.19}$$

Sustituyendo estas expresiones en las ecuaciones anteriores:

$$\dot{x} = \frac{v_r + v_l}{2} \cos\theta \tag{2.20}$$

$$\dot{y} = \frac{v_r + v_l}{2} \sin\theta \tag{2.21}$$

$$\dot{\theta} = \frac{v_r - v_l}{L} \tag{2.22}$$

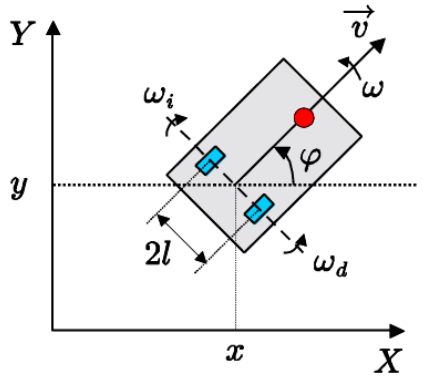


Figura 2.6: Modelo cinemático de un robot diferencial

Fuente: https://www.researchgate.net/figure/FIGURA-1-Diagrama-del-robot-movil_fig1_47297317

- **Modelo dirigido o triciclo**: este otro modelo utiliza una rueda direccional delantera, que puede ser también motriz, y una o dos ruedas traseras, que si bien no disponen de la tracción del vehículo serán pasivas. Es común en vehículos como triciclos o motocicletas. La estabilidad de estos vehículos es también muy limitada por lo que no será aptos para terrenos con una topografía compleja. La ecuación que rige el movimiento del robot viene dada por:

$$\dot{x} = v \cos\theta \tag{2.23}$$

$$\dot{y} = v \sin\theta \tag{2.24}$$

$$\dot{\theta} = \frac{v}{L} \tan \delta \tag{2.25}$$

Donde v siendo la velocidad lineal del vehículo, L es la distancia entre los ejes, θ es la orientación del vehículo y δ es el ángulo de dirección de la rueda delantera.

- **Modelo síncrono**: en este modelo todas las ruedas son motrices y pueden girar simultáneamente en el mismo ángulo, permitiendo que el robot se desplace en una dirección específica sin necesidad de cambiar su orientación. Permiten así un movimiento holonómico al robot, de tal forma que se tiene un control total de la dirección. Esta configuración es útil por ejemplo en entornos industriales donde los espacios son limitados y se requiere de alta maniobrabilidad. Sus ecuaciones de movimiento son las siguientes:

$$\dot{x} = -v \sin \phi \tag{2.26}$$

$$\dot{y} = v \cos \phi \tag{2.27}$$

$$\dot{\phi} = \omega \tag{2.28}$$

Donde v es la velocidad lineal, ω es la angular y ϕ representa la orientación en plano.

- **Modelo de Ackerman**: este modelo se basa en la cinemática utilizada en automóviles, donde las ruedas delanteras son las únicas direccionales, y el radio de giro se ajusta para cada una de ellas de tal manera que el CCI coincida en un punto concreto y así las ruedas traseras sigan la trayectoria deseada. Esta disposición presenta grandes dificultades a la hora de realizar giros cerrados en espacios reducidos, ya que también se trata de un modelo no holonómico. El robot sobre el cual se desarrollará el trabajo dispone de una distribución similar, por lo que se podrá ver las limitaciones mencionadas más adelante. Este modelo contempla por tanto ángulos de giro independientes para las dos ruedas delanteras (δ_i y δ_o) cumpliendo así la ya mencionada geometría. Para ello ha de cumplir

la siguiente relación:

$$\tan \delta_i = \frac{L}{R - \frac{w}{2}} \tag{2.29}$$

$$\tan \delta_o = \frac{L}{R + \frac{w}{2}} \tag{2.30}$$

dónde R es el radio de giro medido desde el eje trasero hasta el centro instantáneo de rotación. δ_i es el ángulo de la rueda interior (la que describe la circunferencia de menor radio), y δ_o es el ángulo de la rueda exterior.

Normalmente, este modelo se suele simplificar, usando el modelo bicicleta ya descrito, pues se asume un solo ángulo de giro efectivo δ para ambas ruedas delanteras ($\delta = \delta_i = \delta_o$). Para el robot MuSHR utilizado en la simulación, se sigue está simplificación pues a efectos prácticos, el modelo dinámico no se ha tenido en cuenta, por lo que fuerzas y fricciones no se han considerado, aunque sí lo hace la simulación.

- **Modelo omnidireccional**: este último modelo utiliza ruedas especiales, como ruedas Mecanum o esféricas, que permiten al robot moverse en cualquier dirección de forma independiente de su orientación, sin restricciones en los grados de libertad del movimiento, pudiendo realizar movimientos laterales, e incluso diagonales sin cambiar la orientación. Esta disposición suele ser muy útil pues el robot se convierte en holonómico, pudiendo alcanzar todos los puntos de su espacio de trabajo sin maniobrar. Las ecuaciones de movimiento de un robot omnidireccional con 4 ruedas se rigen de la siguiente forma:

$$v_x = \frac{R}{4}(-\omega_1 + \omega_2 + \omega_3 - \omega_4) \tag{2.31}$$

$$v_y = \frac{R}{4}(\omega_1 + \omega_2 - \omega_3 - \omega_4) \tag{2.32}$$

$$\omega = \frac{R}{4L}(\omega_1 + \omega_2 + \omega_3 + \omega_4) \tag{2.33}$$

Figura 2.7: RB-KAIROS: Plataforma omnidireccional con ruedas Mecanum

Fuente: `https://robotnik.eu/products/mobile-robots/rb-kairos-2/`

2.3.1.2 Modelado dinámico

A diferencia del modelado cinemático comentado anteriormente, que se enfoca únicamente en la geometría del movimiento sin considerar las fuerzas involucradas, es necesario mencionar la importancia del modelado dinámico, pues describe la relación entre las fuerzas y torques aplicados al sistema y el movimiento resultante. La dinámica permite una representación e implementación más realista al incluir efectos como la inercia, la fricción y las fuerzas externas.

El modelado dinámico es fundamental para comprender y predecir el comportamiento de un robot en todas sus posibles situaciones. En aplicaciones reales, los robots móviles están sujetos a restricciones físicas que afectan su desempeño, como la aceleración limitada por la potencia del motor, la influencia del rozamiento de las ruedas con el suelo o los efectos del propio peso del robot. Estos factores son cruciales en el diseño de sistemas de control y en la simulación de entornos más realistas. El movimiento del robot es posible describirlo con las ecuaciones de Newton-Euler, que relacionan fuerzas y torques con las aceleraciones respectivas:

$$F = m * a \tag{2.34}$$

$$\tau = I * \alpha \tag{2.35}$$

donde F es la fuerza neta, m la masa, a la aceleración lineal, y por otro lado, τ es el torque aplicado, I es el momento de inercia respecto del eje de giro y α la aceleración angular.

Estas ecuaciones constituyen la base de la dinámica, y se combinan con muchos otros modelos de contacto para definir el movimiento resultante en función de las diferentes configuraciones del robot, como las vistas en la cinemática. En el caso de este trabajo, la simulación de las dinámicas del robot se lleva a cabo mediante el motor físico MuJoCo, el simulador de físicas empleado para este trabajo y que se detallará en un capítulo posterior, el cual es capaz de simular con precisión los efectos de las fuerzas, torques y colisiones dentro del entorno. Esto permite que el agente de aprendizaje por refuerzo interactúe con un entorno que imita fielmente las condiciones físicas reales, sin necesidad de definir explícitamente las ecuaciones dinámicas en la implementación. Si bien la dinámica juega un papel clave en el comportamiento de cualquier objeto físico que posea movimiento, en este trabajo, la simulación física es gestionada por MuJoCo, lo que permite que el enfoque se centre en el desarrollo exclusivo del aprendizaje por refuerzo.

2.3.2 Localización

Uno de los pilares fundamentales de la robótica móvil es la localización de un robot dentro del entorno en el que se encuentra. Una persona para poder localizarse y orientarse, toma de referencia su entorno, y en muchos casos puntos característicos o conocidos. Extrapolando esta idea, el robot necesita una memoria en la que almacenar un mapa y poder comparar esa información interna con la percibida por los sensores, para así poder localizarse. Los mapas utilizados pueden ser basados en puntos característicos, de rejilla de ocupación o bien topológicos, los cuales tienen posiciones distintivas y relacionadas entre sí representados como si de un grafo se tratase.

Aplicar la robótica móvil a entornos reales no es tan sencillo, pues existen varias fuentes de las cuales emanan errores que al ser acumulados, puede provocar que el robot se pierda y navegue mal por el entorno. Una de las principales fuentes de error es la imprecisión de las propias lecturas de los diferentes sensores de los que dispone el robot. Tanto las cámaras,

como sensores ultrasonidos entre otros pueden dar falsas mediciones debido a reflexiones o interferencias. Por otro lado, el propio movimiento del robot es otra de las razones de dicho error. Más concretamente, la odometría, cuyo cálculo de posición se basa en los encoders de las ruedas de los robots, también es muy propensa a errores acumulativos, por deslizamiento, terrenos irregulares, funcionamiento de motores de forma no ideal, e inclusive pequeñas deformaciones en las ruedas. Esto hace que sea imposible saber con certeza la posición exacta del robot dentro del entorno. Para manejar este problema, es necesario modelar dicho error de alguna manera, de tal forma que se reduzca lo máximo posible y la localización sea aplicable a entornos reales.

2.3.2.1 Localización de Markov

Para la problemática de determinar la posición de un robot, se aplican métodos que utilizan modelos probabilísticos para manejar esa incertidumbre inherente de las propias aplicaciones reales. A través del filtro de Bayes, la localización de Markov combina información previa, modelos de movimiento y lecturas sensoriales para calcular la probabilidad de que el robot esté en cada posible posición. Esto permite que el robot ajuste continuamente su estimación de ubicación a medida que se mueve y recopila datos del entorno.

La posición del robot es representada como una variable aleatoria, que sigue una distribución de probabilidad, la cual indica la probabilidad de que el robot este en una determinada posición. Para calcular esa distribución de probabilidad del estado actual del robot se utiliza el conocido como Filtro de Bayes:

$$Bel(x_t) = \eta P(z_t \mid x_t, m) \int P(x_t \mid u_t, x_{t-1}) Bel(x_{t-1})\, dx_{t-1} \tag{2.36}$$

donde cada termino corresponde a lo siguiente:

- **Factor de normalización** η : para que la suma (o integral, en este caso) de todas las probabilidades sea igual a 1
- **Modelo de sensor** $P(z_t \mid x_t, m)$: modela el comportamiento del sensor, relacionando las lecturas con la posición y el mapa. Suponiendo que se sabe dónde están los obstáculos en el mapa m, y la posición estimada del robot x, se calcula una estimación de las

lecturas z. Existen dos modelos principalmente modelos para ello, como son el modelo *"beam range"*, que estima la intersección del haz del sensor con los obstáculos más cercanos, y modela las diferentes fuentes de error siendo más costoso computacionalmente, y por otro lado, el modelo basado en *"scan"* que compara directamente las lecturas con el mapa almacenado siendo menos preciso y más rápido. Para poder reducir el error, cada vez más se utilizan sensores que son capaces de detectar *landarks* como reflectores visuales o dispositivos que emiten señales radio entre otro muchos formatos. Esto ayuda a reducir el error y corregir desviaciones cuando son detectados.

Para manejar estos errores, el modelo del sensor se representa como una mezcla ponderada de diferentes distribuciones que modelan cada una de estas fuentes de error, donde la suma de todas ellas representan la incertidumbre total del sensor. Las fuentes a modelar son el ruido en las lecturas, los obstáculos inesperados por el hecho de trabajar en entornos dinámicos, las lecturas aleatorias y las lecturas que vienen limitadas por el rango máximo del sensor.

- **Modelo de movimiento** $P(x_t \mid u_t, x_{t-1})$: este se utiliza para predecir posiciones futuras del robot, basándose en la posición anterior y el comando de movimiento (acción) ejecutado u_t. Este modelado se puede basar en la propia odometría, usando los encoders, o bien, basándose en la velocidad utilizando estimaciones de estas sin depender de los encoders de las ruedas. El modelo de error en la odometría introduce términos probabilísticos para representar la incertidumbre en la estimación de la posición. Para ello, se suelen emplear distribuciones probabilísticas como una distribución normal (gaussiana) o bien triangular de manera más simple, obteniendo valores más compactos.

- **Conocimiento del estado anterior** $Bel(x_{t-1})$: representa el conocimiento previo sobre la posición del robot. Además se necesitará una creencia inicial ($bel(x_0)$) sobre la posición del robot. Si se conoce de manera exacta, esta probabilidad será igual a 1 en esa posición y 0 en todas las demás. Si en cambio es desconocida, se asume una distribución uniforme, significando que el robot podría estar en cualquier lugar con la misma probabilidad.

El algoritmo, de manera general, lleva a cabo una predicción de movimiento, actualizando

dicha distribución previa según el modelo de movimiento, y a su vez, realiza correcciones ajustando las predicciones mediante nuevas lecturas, que pasan el filtro del modelado de los valores tomados por el sensor.

2.3.2.2 SLAM (*Simultaneous Localization and Mapping*)

La mayoría de robots que dispongan de la capacidad de navegar por un entorno hacen uso de un mapa para poder localizarse. Este es un aspecto fundamental pues le da la capacidad al robot, no solo de poder localizarse, también le permite planificar trayectorias o tomar decisiones, en función de su entorno. Este mapa no siempre es conocido, o bien no se le puede proporcionar al robot, por lo que el hecho de mapear se vuelve una tarea extra dentro de este tipo de robots. Como tal, mapear consiste en calcular el mapa más probable a partir de la odometría y las lecturas de los sensores. A diferencia de los métodos tradicionales que requieren conocer previamente la ubicación del robot, SLAM permite que el robot se localice y explore el entorno sin información previa.

El mayor desafío del SLAM es la incertidumbre inherente al proceso de localización y mapeo simultáneo. A diferencia de la localización con poses conocidas, donde el filtro de Bayes puede aplicarse directamente, en SLAM no se puede utilizar de manera simple debido a la gran cantidad de hipótesis posibles sobre la trayectoria del robot y la estructura del mapa. Esto se debe a que no solo hay incertidumbre en las mediciones del sensor, sino también en la posición del robot, lo que genera un problema altamente no lineal y computacionalmente costoso. Para la representación espacial se utiliza generalmente un mapa de rejilla de ocupación independientes entre ellas, en el que se indica si está ocupada esa celda. Cuanto mayor tamaño de rejilla, menor computo pues menor será el nivel de discretización. Sin embargo, para SLAM este enfoque no es escalable porque el espacio de estados es demasiado grande pues se debe estimar tanto la trayectoria del robot (posición posible del robot) como la posición de cada elemento del mapa, lo que aumenta exponencialmente el número de combinaciones posibles. Además, un error en la estimación de la posición del robot afecta la ubicación de los elementos del mapa y viceversa por lo que las fuentes de error aumentan y se retroalimentan.

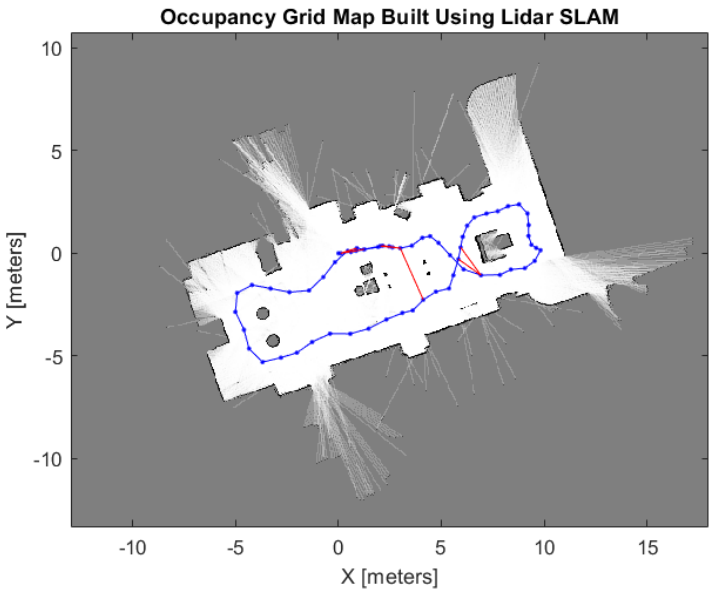


Figura 2.8: Mapa de ocupación creado usando técnicas SLAM

Fuente: `https://www.mathworks.com/help/nav/ug/implement-simultaneous-localization-and-mapping-with-lidar-scans.html`

Para abordar SLAM de manera eficiente, existen una serie de métodos que combinan técnicas probabilísticas con optimización matemática para mejorar la precisión y la escalabilidad del algoritmo. Los principales enfoques para resolverlo son los siguientes:

- **EKF-SLAM (Filtro de Kalman Extendido)**: Se representa el sistema como un vector que contiene la posición del robot y la de los landmarks (puntos de referencia), mediante una distribución gaussiana. En primer lugar, predice usando un modelo de movimiento como lo es la odometría para estimar la nueva posición, y a continuación, al detectar un landmark, se usa el Filtro de Kalman Extendido (EKF) para actualizar la estimación de su posición y reducir la incertidumbre. La complejidad computacional es exponencial con respecto del número de landmarks, lo que lo hace ineficiente para entornos con muchas referencias.

- **FastSLAM (Filtro de Partículas + EKF para el mapa)**: Este otro algoritmo extiende el anterior, utilizando un enfoque de filtro de partículas, separando la estimación de la trayectoria del robot y la del mapa. Cada partícula en el filtro representa una

posible trayectoria del robot y mantiene un conjunto de filtros de Kalman individuales para los landmarks, reduciendo así la dimensionalidad del problema.

- **SLAM basado en Grafos**: se representa el problema como un grafo, donde los nodos corresponden a poses del robot y los landmarks, y los bordes representan mediciones con incertidumbre. Se utiliza optimización de mínimos cuadrados para encontrar la configuración del grafo que minimiza los errores de medida y odometría. Este método permite gestionar grandes entornos y cerrar ciclos con alta precisión, a costa de una mayor carga computacional.

A pesar de ser uno de los principales problemas de la robótica móvil, la localización y el mapeo toman un segundo plano para el desarrollo de este trabajo, ya que la pose del robot es conocida en cada una de las iteraciones de manera exacta para la ejecución de la simulación. No solo del robot, si no también del objetivo a alcanzar, por lo que no será necesario aplicar este tipo de técnicas en nuestro robot simulado, como sí lo sería para aplicaciones reales de robots que espacios de interior donde no disponen de cobertura para poder localizarse con exactitud.

2.3.3 Control y navegación en la robótica móvil

Una vez se dispone de un robot con un modelado sólido y funcional, diseñado con la capacidad de realizar las tareas para las que fue creado, es necesario desarrollar e implementar las técnicas que le brinden al robot la capacidad de poder alcanzar un objetivo por un determinado camino, preferiblemente el más corto o de menor coste, evitando los obstáculos de su alrededor. Este conjunto de algoritmos reciben el nombre de navegación, y son los encargados de darle la autonomía al robot para moverse por su entorno.

A la hora de hablar de navegación, es necesario hacer una distinción entre dos tipos complementarios, navegación global y local. La primera de ellas se centra en encontrar, mediante un algoritmo determinado, el camino óptimo entre dos puntos haciendo uso de un mapa. Por otro lado la navegación local, centrada en la evitación de aquellos obstáculos, que como tal no estén referenciados en el mapa utilizado, utiliza la información de los sensores del robot para recalcular nuevas rutas sin alejarse en exceso del camino óptimo ya calculado en el pla-

nificado global. A lo largo de esta sección se van a introducir los principales algoritmos de planificación y optimización, llevando a cabo una introducción sobre ellos.

2.3.3.1 Planificación global de trayectorias

Para poder hacer un calculo efectivo del camino más corto por el que poder navegar con el robot, es necesario tratar la información de la que se dispone, como el mapa o las características de movimiento del robot. Por regla general, los algoritmos que planifican las rutas, trabajan con el espacio de configuraciones del robot, es decir, el espacio formado por todas las posibles poses del robot. Este conjunto de poses, también conocido como *CSpace*, tiene las mismas dimensiones que grados de libertad tenga el robot. Si hablamos de robótica móvil, la pose se define típicamente como (x, y, θ), sin embargo, se suele ignorar la θ, quedándose en un espacio 2D para poder simplificarlo.

El objetivo pasa por tanto por la búsqueda de una ruta en el mapa, que solo pase por el espacio libre en este, considerando siempre las dimensiones del robot con el que se trabaje. Para ello se dilatan los obstáculos con al menos un tamaño igual al radio del robot, y así poder suponer que el robot es un punto en el mapa, simplificando los cálculos. Esta operación de dilatación se ha formalizado matemáticamente a lo largo de los años y recibe el nombre de *Suma de Minkowski*, donde se realiza una suma entre la forma del robot y los propios obstáculos como ya se ha mencionado. Muchos algoritmos de preprocesamiento de mapas generan los conocidos como *costmaps*, donde con las misma idea, se genera un mapa de costes para el robot, donde cuanto más cerca se esté del obstáculo más peso tendrá esa posible ruta, promoviendo así navegar lo más alejado posible a estos. Inicialmente, hay infinitos caminos posibles entre dos puntos de origen y destino, pues el espacio está definido por los número reales. El espacio de búsqueda de una ruta no puede ser infinito por lo que se vuelve necesario restringir y discretizar el espacio, limitando las posibles rutas. Para ello se suele transformar el *CSpace* en un grafo que contenga todos los posibles caminos a evaluar, y posteriormente aplicar un algoritmo de optimización sobre ese grafo. Existen varias formas de transformar el *CSpace* en un grafo:

- **Rejilla de ocupación**: el mapa se discretiza en un tamaño de celda concreto, donde cada una de estas celdas viene representada por un nodo del grafo, el cual se conecta

con sus 8 vecinos colindantes. Los nodos pueden ser o no accesibles por el robot, en función de la dilatación previa aplicada al mapa, y si los obstáculos ocupan o no dicha celda, como se ha visto anteriormente.

- **Grafo de visibilidad**: se crea un grafo cuyos nodos son los vértices de los polígonos de los obstáculos ya dilatados, y cuyas uniones son todas las posibles conexiones entre ellos, siempre y cuando no se vea "bloqueado" por otro obstáculo existente.
- **Grafo de voronoi**: la elaboración del grafo pasa por maximizar la distancia mínima a los obstáculos del entorno, es decir, conseguir que los puntos tengan la misma distancia mínima entre dos o más objetos de su alrededor. El grafo resultante no asegura en ningún caso el camino más corto, pues se busca mantener una distancia que en la mayoría de ocasiones provoca la generación de rutas no óptimas. Sin embargo, esta transformación es la más conservadora, pues al maximizar la distancia a todos los objetos colindantes, se maximiza la seguridad.

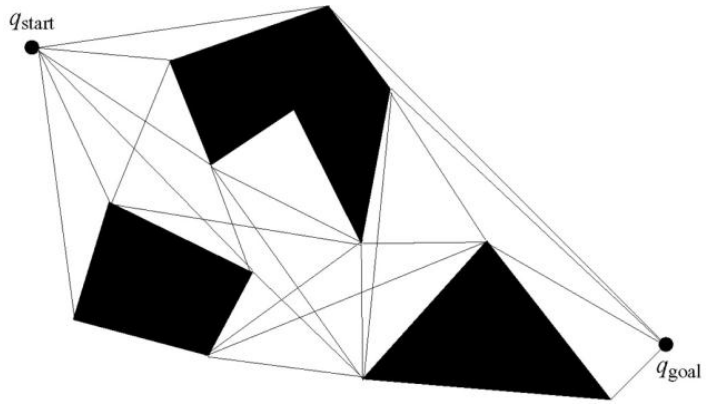


Figura 2.9: Grafo de visibilidad entre un punto origen y un destino

Fuente: `https://www.researchgate.net/figure/Grafo-de-visibilidad-en-un-entorno-de-dos-obstaculos-tomado-de-13_fig2_277756099`

Tras haber definido la forma del grafo que representará las posibles rutas y caminos a tomar por el robot en el mapa, se ha de aplicar un algoritmo que de como resultado la secuencia de nodos, que optimice el camino más corto posible en esa situación. Aunque existen numerosos algoritmos para resolver este problema, los 3 más utilizados en la robótica móvil suelen ser los siguientes:

- **Dijkstra**: es el algoritmo de búsqueda clásico más popular. Su funcionamiento se basa en inicializar una tabla con valores infinitos, e ir seleccionando el nodo con la menor distancia y actualizando los costes de cada vecino, si el camino es más corto que el almacenado. Este proceso se itera hasta visitar todos los nodos, o bien alcanzar el nodo destino. A pesar de funcionar correctamente, no dispone de un cálculo heurístico, por lo que explora más nodos de los necesarios en muchos casos, y no solo eso, si no que es muy costoso computacionalmente $O(n^2)$.

- A^*: este algoritmo surgió de la investigación realizada con el robot Shakey, y mejora el de Dijkstra, utilizando una heurística para guiar la búsqueda, reduciendo el número de nodos explorados, y expandiendo siempre el nodo de menor coste $f(n)$. Se utiliza una función de evaluación:

$$f(n) = g(n) + h(n) \tag{2.37}$$

  donde:

  - $g(n)$ es el coste acumulado desde el nodo inicial hasta el actual
  - $h(n)$ es una heurística que estima el costo desde el nodo actual hasta el destino, donde se hace una estimación optimista del camino restante. Para ello se suele utilizar la distancia de Manhattan o la euclidiana en función de las características y limitaciones del robot y su entorno.

  Este algoritmo es más eficiente, pues explora menos nodos, garantizando aun así el camino más corto. La heurística tomará gran importancia pues será la encargada de explorar un mayor o menor número de rutas.

- D^* es una variante del algoritmo A^*, pero diseñada para entornos dinámicos en los que el mapa está sujeto a cambios tras la planificación inicial. Este algoritmo es capaz de replanificar rutas cuando se detectan los mencionados cambios, y en lugar de recalcular todo desde cero, se actualizan solo aquellas partes que hayan cambiado, recalculando solo la zona afectada. Este método es idóneo para robots que operan en entornos dinámicos, sin embargo, presenta una mayor complejidad de implementación, así como de valorar cuando es necesario volver a planear la ruta, incurriendo en un mayor costo de cálculos.

En muchas ocasiones, estos algoritmos deterministas no son lo suficientemente prácticos ante espacios de búsqueda muy grandes, pues las rutas a calcular conllevan mucho tiempo de cálculo. Como alternativa, se desarrollan un tipo de algoritmos probabilísticos,basados en técnicas de muestreo para encontrar rutas viables en este tipo de situaciones. Principalmente exiten dos algoritmos de esta tipología:

- **PRM** *(Probabilistic Roadmap)*: Se generan puntos aleatorios en el espacio libre y se conectan para formar un grafo de caminos viables, donde no se colisione con ningún obstáculo. Luego, se usa un algoritmo clásico de búsqueda en grafos para encontrar la mejor ruta a través del grafo generado. A pesar de ser una opción más eficiente en espacios muy grandes, no es aplicable a entornos dinámicos, pues se debería muestrear y recalcular el grafo ante cualquier cambio. A su vez, la calidad de las rutas dependerá directamente de los puntos muestreados.
- **RRT** *(Rapidly-exploring Random Tree)*: en este otro algoritmo, se calcula un árbol de búsqueda expandiendo nodos de manera aleatoria dentro del espacio de configuración. Se selecciona un nodo aleatorio, y se extiende una nueva rama al punto más cercano en el árbol de manera iterativa hasta alcanzar el destino. Este algoritmo es sencillo y eficiente para espacios de alta dimensionalidad, encontrando posibles soluciones sin necesidad de explorar todo el entorno de posibilidades. Por contra, no garantiza el camino más corto global, y la trayectoria resultante suele ser muy irregular, por lo que necesita un postprocesado para suavizarse.

2.3.3.2 Planificación local y evitación de obstáculos

La robótica móvil posee unas características muy concretas ya que la gran totalidad de entornos y espacios de trabajo de este tipo de robots, son dinámicos y cambiantes. Hasta ahora se ha llevado a cabo una explicación de como poder planificar rutas en diferentes tipos de mapas según su complejidad y los objetivos buscados, pero es necesario tener unos algoritmos que brinde la capacidad al robot de evitar obstáculos que no estén contemplados mapa. Estas técnicas deben responder en tiempo real, de manera similar a como lo haría un robot reactivo sin planificación previa, utilizando única y exclusivamente, información local de sus sensores. Estos métodos se pueden clasificar en dos tipos principalmente:

- **Métodos de campo de fuerzas**: estos métodos se sustentan sobre la misma idea que la robótica reactiva basada en campos de potencial, donde existen dos tipos de fuerzas, atractiva y repulsiva, utilizadas para la replanificación de trayectorias. El algoritmo más utilizado que implementa dicho concepto es el *Vector Field Force (VFF)* de Borenstein y cols. (1991). Este combina el campo de potencial y fuerzas mencionadas anteriormente con una rejilla de ocupación local de un determinado tamaño alrededor del robot, donde las celdas incrementan su valor conforme se detectan lecturas en esa zona delimitada por una celda concreta. La fuerza de atracción F_t es constante y toma la dirección al objetivo a alcanzar. Por otro lado, la fuerza de repulsión total, es la suma resultante de las fuerzas de calculadas para cada celda respecto del robot, con sus respectivas magnitudes en función de la certeza del obstáculo detectado, teniendo en cuenta, aquellas que se encuentren dentro de la ventana local de análisis. Al robot se le asignará un comando de rotación δ, calculado como el ángulo del vector de la fuerza resultante con el eje X. A razón de evitar cambios bruscos, se implementa un filtro de paso bajo que atenúe los comandos de giro.

  Este algoritmo cuenta con un problema principalmente: la presencia de ruido en los sensores que provoca un comportamiento inestable, a raíz de lecturas instantáneas que no son del todo fiables. A esto se suma un problema de oscilación en el movimiento del robot, debido al filtro implementado y a la inercia intrínseca al propio movimiento. Para poder solventarlo, se suele hacer lo que se conoce como *damping*, es decir, reducir la fuerza repulsiva de los obstáculos si el robot no va en dirección a estos. En el caso de que la diferencia de ángulos entre el obstáculo y el movimiento sea mayor a 90º, puede provocar que el robot se ponga a dar vueltas "perdido" en un mínimo local, por lo que se suele implementar una modalidad para seguir paredes para así evitarlos.

  La robustez del algoritmo frente a ciertas situaciones es bastante débil. Esto es debido a que su implementación se puede considerar relativamente sencilla, donde por norma general, se reduce de una forma excesivamente drástica toda la información de los sensores. Algunas de estas situaciones son los pasillos estrechos, donde las oscilaciones se hacen más evidentes, o bien pequeñas aberturas y puertas en el mapa que repelen el robot haciendo su entrada inaccesible.

Como solución a todos los problemas ya comentados, se hace uso de los histogramas mediante un método conocido como *Vector Field Histogram (VFH)*. En lugar de usar fuerzas vectoriales directas como en VFF, VFH convierte el entorno en un histograma polar representando así, la densidad de obstáculos en diferentes direcciones alrededor del robot. Para construir dicho histograma, se transforma el espacio 2D de la rejilla del método VFF en solamente una dimensión, dividiéndola en sectores angulares y construyendo un vector con tantas componentes como en sectores se haya hecho la división. La magnitud de cada componente del histograma dependerá del número de celdas ocupadas en ese sector y la distancia a la que se encuentren. Posteriormente, umbralizando se logra determinar que direcciones están bloqueadas y cuales libres para así decidir cual es el valle que reúne las condiciones de amplitud necesarias para entrar el robot, y que esté lo más cerca posible con la dirección que se quiere tomar.

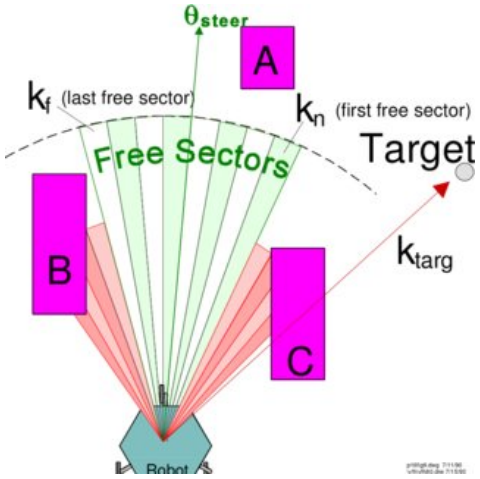


Figura 2.10: Vector Field Histogram (VFH)

Fuente: Borenstein, Johann Koren, Yoram. (1991). The Vector Field Histogram - Fast Obstacle Avoidance For Mobile Robots. Robotics and Automation

- **Métodos en el espacio de velocidades:** A la hora de que un robot móvil navegue, es necesario, e igual de importante que evitar obstáculos, tener en cuenta las limitaciones y diseño dinámico del robot, para así hacerlo de manera segura y eficiente. Los métodos vistos anteriormente trabajan en un espacio de coordenadas, pues separan la trayectoria óptima de los comandos de movimiento que se aplicarían a los actuadores. La técnica

más utilizada en el espacio de velocidades (v, w) es el método de la ventana dinámica, Fox y cols. (1997).

Cada ciertos intervalos de tiempos, se lleva a cabo un calculo de aquellas velocidades que el robot puede alcanzar en el siguiente instante, respetando sus limitaciones dinámicas. Esto da como resultado un conjunto de trayectorias semicirculares si se mantuviesen constantes unas ciertas con signas de movimiento durante un intervalo concreto. Por tanto, el principal objetivo pasa por encontrar las velocidades (v, w) designadas para el siguiente intervalo de tiempo Δt, las cuales sean capaces de obtener un movimiento hacia el objetivo, alejándose de los obstáculos y con la mayor velocidad posible. Es posible formalizarlo como:

$$G(v, w) = \alpha * heading(v, w) + \beta * dist(v, w) + \gamma * vel(v, w) \tag{2.38}$$

donde se quiere maximizar los siguientes tres parámetros ponderados por unos pesos ajustables α, β, γ:

- heading(v,w): es la componente que marca la dirección o alineación hacia el objetivo, en una futura posición una vez aplicadas las velocidades.

- dist(v,w): es la distancia al obstáculo más cercano que intersecte con esa posible trayectoria curva.

- vel(v,w): las velocidades del robot que se quieren maximizar para encontrar las trayectorias más óptimas y eficientes a nivel temporal.

A pesar de ser un algoritmo mucho más fiable, y eficiente a la hora de aplicarlo a situaciones reales de navegación, el programador deberá ajustar el instante de tiempo a evaluar, ya que es posible que si es demasiado grande, el robot tenga que retroceder por haber ido demasiado rápido. Por contra con intervalos más cortos, más cálculos y un mayor coste computacional en tiempo real.

2.4 Aplicación concreta del aprendizaje por refuerzo a un robot móvil

El campo de la robótica móvil tiene ya a sus espaldas un gran cantidad de años de investigación y desarrollo dentro del mundo de la propia robótica. Sin embargo, no ha sido hasta el reciente crecimiento de la inteligencia artificial y el machine learning, cuando las técnicas de navegación tradicional empleadas han sufrido una revolución. Tal y como se ha explicado, los métodos tradicionales buscan resolver problemas de una forma óptima o subóptima aplicando algoritmos basados en la información que disponen del mapa y de las lecturas de los sensores. La aplicación del aprendizaje por refuerzo en robótica móvil es un tema de investigación en el que se está invirtiendo tiempo y recursos económicos por todo el mundo ya que puede llevar consigo grandes ventajas frente a los enfoques tradicionales.

En un articulo publicado en la revista *Tsinghua Science and Technology* por los investigadores Zhu y Zhang (2021), se presenta una revisión sistemática y detallada sobre la aplicación del aprendizaje por refuerzo profundo para resolver tareas de navegación con plataformas móviles. Esta publicación respalda la idea de cómo estos nuevos métodos ofrecen una serie de ventajas significativas pues los métodos clásicos presentan debilidades frente a situaciones de entornos muy dinámicos o desconocidos. EL uso de DRL en la navegación se presenta como la solución más prometedora gracias a su capacidad para aprender estrategias directamente desde los datos recopilados por los sensores, sin necesidad de ser tratados y sin necesidad de mapas globales precisos. Estas características son las que definen aquellos entornos y situaciones donde es realmente interesante aplicar este nuevo paradigma. Algunos ejemplos serían la navegación social, donde los robots deben moverse en entornos con peatones, como centros comerciales, cumpliendo con las normas sociales y manejando la aleatoriedad del comportamiento humano; o bien la navegación en interiores donde la evitación de obstáculos sea muy común por el dinamismo del propio entorno. Otro ámbito donde es interesante este tipo de navegación es en flotas de robots conformadas por varios de ellos, donde por ejemplo, necesiten coordinarse sin comunicación directa entre ellos. También es aplicable para planificaciones globales de trayectorias, donde el robot se encuentre en una situación donde no se dispone de ninguna información en referencia al mapa donde se encuentre, y donde

a mayores, pueda necesitar de cierto grado de independencia. El caso más identificativo, y donde ya se ha comenzado a aplicar, es en la exploración espacial, donde los rovers cuentan con grandes limitaciones temporales para comunicarse con las bases terrestres, y donde los entornos espaciales son totalmente desconocidos. Las principales características del uso de aprendizaje por refuerzo profundo en el ámbito de la robótica móvil son las siguientes:

- Navegación sin necesidad de mapas globales *(mapless navigation)*, aprendiendo directamente a navegar usando exclusivamente datos de los sensores, permitiendo operar en entornos desconocidos o dinámicos sin necesidad de construir mapas o actualizarlos.

- La política de navegación es directa entre las entradas sensoriales hasta las acciones, por lo que se simplifica eliminando módulos de mapeo, localización o planificación.

- Pueden aprender comportamientos complejos y generalizar a situaciones no vistas durante el entrenamiento, especialmente si se entrena en entornos variados, pudiendo manejar situaciones cambiantes y dinámicas.

- Permite una toma de decisiones reactiva y eficiente incluso en presencia de cambios repentinos del entorno.

- Es robusto frente al ruido sensorial y a los errores de medición, lo cual reduce la necesidad de sensores de altas capacidades, ya que es posible modelar todos estos errores en el entrenamiento.

- Tiene la capacidad y flexibilidad de integrarse por partes específicas con los enfoques tradicionales de forma híbrida, pudiendo solo sustituir la planificación local, o funcionar de forma totalmente independiente.

- Los agentes pueden mejorar continuamente su política mediante la interacción con el entorno, algo difícil de lograr en métodos clásicos con lógica programada.Esta capacidad permite adaptarse a nuevas tareas o usuarios con entrenamiento adicional. Las políticas aprendidas pueden ser reutilizadas o adaptadas (transfer learning) a otros robots, entornos o tareas, reduciendo los costes de desarrollo.

- Capacidad de ser integrado fácilmente con datos visuales de cámaras, lo que lo hace ideal para navegación basada en visión o tareas como seguimiento de objetos o navegación guiada por imágenes.
- El uso de simuladores lo más fieles posibles permite entrenar y luego desplegar en el mundo real sin necesidad de datos del entorno real durante el entrenamiento, evitando daños y reduciendo tiempo de pruebas.

3 Objetivos

El objetivo de este Trabajo Final de Grado de manera general, es realizar un proceso de aprendizaje e investigación del reinforcement learning en su aplicación concreta en el ámbito de la robótica y en el control de robots. Se llevará a cabo un estudio profundo de las bases de este paradigma para su entendimiento, pudiendo así realizar una experimentación adecuada y una mejora progresiva del control, concretamente, de un robot móvil. En cuanto a los objetivos más específicos del trabajo tenemos:

- Asentar conceptos del aprendizaje por refuerzo y entender cómo su integración en la robótica móvil puede traer consigo ventajas frente al control tradicional.
- Realizar una investigación sobre las principales herramientas de software para el desarrollo de este paradigma en la robótica y su posterior aplicación.
- Elaborar un entorno de simulación acorde a las necesidades de la propia naturaleza del aprendizaje por refuerzo y las herramientas seleccionadas para su simulación.
- Llevar a cabo la implementación del aprendizaje y la simulación del robot en el entorno previamente elaborado.
- Entender cómo evaluar los resultados obtenidos de los modelos entrenados mediante diferentes métricas, como el tiempo medio de los episodios o la recompensa media obtenida en cada uno de ellos.
- Optimizar el modelo entrenado, ajustando parámetros y realizando cambios en el entorno de simulación, explorando nuevas mejoras para obtener mejores resultados.
- Documentar los resultados y las conclusiones, destacando limitaciones y propuestas para furas investigaciones.

4 Metodología

En este cuarto apartado del trabajo se definen las herramientas y procedimientos empleadas para llevar a cabo el desarrollo del presente trabajo. La metodología adoptada busca garantizar la obtención de resultados lo más fiables posibles, siguiendo las pautas de los objetivos establecidos. Para las simulaciones realizadas en este trabajo, el modelo del robot móvil en 3D fue seleccionado de forma libre, sin estar condicionado a un diseño específico. Se optó por utilizar un modelo disponible en internet con un modelado cinemático similar al de un automóvil (modelo de Ackerman), debido a su accesibilidad y características adecuadas para los objetivos planteados. Esta elección permitió enfocar los esfuerzos en la implementación y análisis de la metodología, sin la necesidad de desarrollar un modelo desde cero. La realización de este trabajo será simulada en su totalidad al no disponer de un robot real de estas características, por lo que el entrenamiento y la evaluación de los modelos obtenidos no se aplicarán a un entorno real. Para la implementación se utilizará el lenguaje de programación más popular entre implementaciones de aprendizaje por refuerzo como lo es Python, el cual cuenta con numerosas bibliotecas y frameworks diseñados específicamente para el aprendizaje por refuerzo como los que se presentarán a continuación. Además se hará uso de la herramienta Conda utilizada principalmente para la gestión de entornos y paquetes. Esta herramienta permite la gestión de entornos virtuales de manera aislada, previniendo conflictos de dependencias entre diferentes proyectos, pudiendo instalar y utilizar versiones específicas de Python y librerías, necesarias para garantizar que todo funciona correctamente, entre otras ventajas.

Para el desarrollo del trabajo se ha optado por realizar pruebas con otros entornos ya predefinidos por las herramientas utilizadas. Se mostrarán las características de cada uno de ellos para una mejor compresión del paradigma del aprendizaje por refuerzo, como vienen a ser los espacio de acciones y observaciones de cada uno de ellos. También se llevará a cabo una breve explicación, pero más concisa, comparando los diferentes algoritmos aplicables

mediante las librerías utilizadas, dónde se detallarán las características de cada uno de ellos.

4.1 Herramientas y librerías de software empleadas

En este siguiente apartado, se presentarán las librerías de Python utilizadas para la simulación e implementación del aprendizaje por refuerzo en un agente robótico capaz de interactuar con su entorno, logrando así el objetivo de aprender una determinada tarea. Gracias a ellas es posible confeccionar el entorno acorde a las necesidades de la tarea a aprender y al diseño del robot utilizado. Más concretamente, se han utilizado tres librerías "independientes", con funciones concretas, cuya compatibilidad además, es total entre ellas: *MuJoCo, OpenAI Gym y Stable Baselines3*.

4.1.1 MuJoCo

Multi-Joint dynamics with Contact, también conocido como MuJoCo (Todorov y cols. (2012)), es un simulador de físicas lanzado en 2015 por la empresa Roboti LLC, la cual fue fundada por Emo Todorov, destinado a la investigación y al desarrollo en diferentes campos donde se requiera realizar simulaciones con contactos complejos, como lo son por ejemplo el de la robótica, la biomecánica o la animación 3D para videojuegos, entre otros. Años más tarde, en 2021, pasó a ser gratuito y *opensource* tras ser adquirido por la empresa Deepmind, a la cual sigue perteneciendo en la actualidad. MuJoCo ofrece la capacidad de poder implementar simulaciones donde además sean necesarias una alta precisión y rapidez con el objetivo de replicar entornos reales de la manera más fiel posible. Este motor gŕafico pone a disposición todas las características esenciales y necesarias para el desarrollo de este trabajo.

MuJoCo no es interesante exclusivamente a nivel de cálculo de interacciones y contactos entre cuerpos físicos, si no que dispone de muchas otras característica, como que utiliza un formato de archivo basado en XML llamado MJCF (MuJoCo File) para describir y configurar los modelos que se usarán en las simulaciones. Además, incluye un compilador de modelos integrado, lo que significa que puede procesar estos archivos XML y convertirlos en representaciones internas optimizadas que el simulador puede utilizar de manera eficiente, eliminando la necesidad de realizar configuraciones complicadas en lenguajes de programación. El formato descrito es posiblemente el formato más generalizado entre aplicaciones de simulación

robóticas por su fácil legibilidad y posibilidades, permitiendo definir elementos del modelo, como cuerpos rígidos, articulaciones, sensores, actuadores, contactos y propiedades físicas de estos como su masa o su fricción. Originalmente MuJoCo fue desarrollada por completo en C y C++ pero proporciona un API nativa para Python que permite interactuar directamente con el simulador mediante este lenguaje. Otra de las cracterísticas claves es que dispone de una interfaz gráfica con visualización 3D interactiva en OpenGL, la cual permite cargar modelos de manera sencilla y probar las configuraciones que lo definen en el archivo correspondiente. Asimismo, la página web del simulador proporciona una documentación muy extensa y completa para entender su funcionamiento.

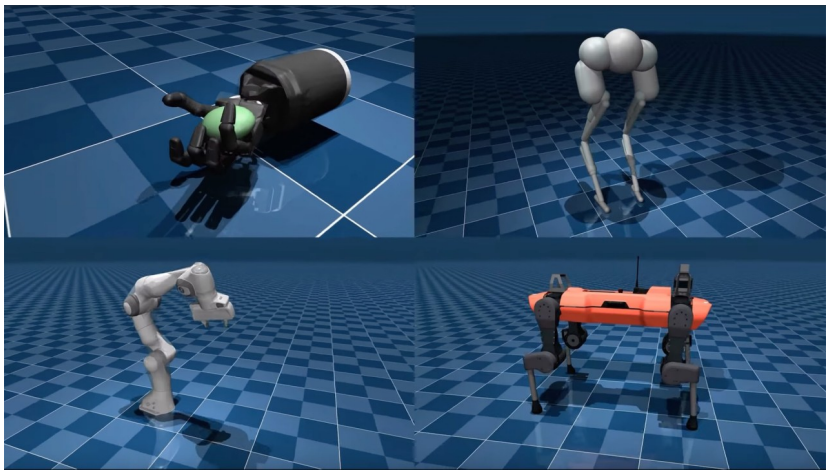


Figura 4.1: Ejemplos de entornos simulados en MuJoCo

Fuente: `https://www.linkedin.com/pulse/deepmind-releases-mujoco-menagerie-collection-simulated-kromme/`

4.1.2 OpenAI Gym / Gymnasium

Gym (Brockman (2016)) es una librería desarrollada por parte de la empresa OpenAI, que contiene las herramientas necesarias para la creación y uso de entornos de aprendizaje por refuerzo (RL). Recientemente evolucionó a Gymnasium, pues OpenAI dejó de dar soporte activo a Gym y la comunidad tomó la iniciativa de continuar con su desarrollo y mejora, solucionando y corrigiendo errores, ampliando su compatibilidad con nuevas versiones de Python entre otras muchas cosas. Esta evolución garantiza la compatibilidad retroactiva, por lo que los entornos y configuraciones diseñados para Gym funcionarán sin problemas en Gymna-

sium. Además una de sus grandes ventajas es que proporciona una interfaz estandarizada para interactuar con los entornos simulados creados, lo que facilita la comparación y prueba de diferentes algoritmos utilizando diferentes parámetros, e inclusive agentes.

Como se puede ver en la figura 4.2, Gym dispone de numerosos entornos ya implementados que pueden ser utilizados por el usuario para el aprendizaje de este paradigma así como para la implementación de unos nuevos entornos, ya sea partiendo de estos mismos o creando sus propios entornos desde cero. Los entornos existentes que recopila esta librería son muy variados y van desde entornos 2D para controlar a un agente simple hasta otros con agentes mucho más complejos en 3D, haciendo uso de otras librerías como la de MuJoCo. A su vez, Gymnasium ofrece una amplia documentación en su página web, donde recoge información básica del uso y funcionamiento de la librería, junto con tutoriales y explicaciones de cómo realizar las implementaciones de un nuevo entorno propio.

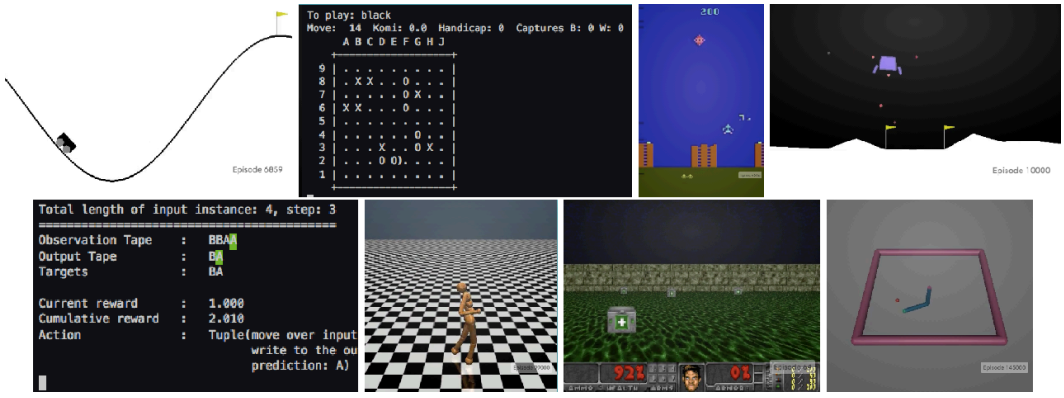


Figura 4.2: Ejemplos de entornos simulados mediante Gymnasium

Fuente: `https://shimmy.farama.org/environments/gym/`

4.1.3 Stable Baselines3

En último lugar, Stable-Baselines3 (SB3) (Raffin y cols. (2021)) es es una biblioteca de código abierto desarrollada en Python diseñada para implementar algoritmos de aprendizaje por refuerzo profundo (DRL) de la manera más simple, eficiente y accesible posible. Esta es una continuación y mejora de Stable-Baselines, que a su vez fue un *fork* de OpenAI Baselines, pero con un enfoque en mantener la calidad del código, mejorar la documentación y facilitar el desarrollo. Consigo, Stable-Baselines3 adoptó PyTorch como su *backend* principal

en lugar de TensorFlow debido a la capacidad de depurar modelos en tiempo real, así como su compatibilidad de integración de manera robusta con otras bibliotecas de Python, como NumPy, pandas y TensorBoard utilizadas en este tipo de implementaciones de aprendizaje por refuerzo profundo.

Tras la salida de diferentes actualizaciones y versiones, en la actualidad SB3 recoge los algoritmos más usados y robustos de RL, ya sean basados en política, en valor entre otros tipos... Estos algoritmos están diseñados para cubrir una amplia variedad de tipos de escenarios, desde entornos discretos donde las posibilidades son más definidas y limitadas hasta entornos continuos, como pueden llegar a ser aplicaciones reales. Stable-Baselines3, al igual que las herramientas anteriormente mencionadas, posee una amplia y detallada documentación que facilita el entendimiento de como emplear estos algoritmos que a su vez ayuda a entender como funcionan realmente. Cabe destacar que SB3 es totalmente compatible con Gymnasium por lo que se presenta como una gran herramienta para el desarrollo del objetivo de este trabajo.

4.2 Entornos

La construcción de un entorno de manera correcta es una de las partes fundamentales para el aprendizaje de un agente, pues la confección de este engloba desde el criterio de recompensas adoptado, el espacio de observaciones y acciones del agente a entrenar, entre otros muchos aspectos. A continuación se presentarán los entornos utilizados tanto para el aprendizaje del uso de las herramientas del anterior apartado, como también para el aprendizaje de los conceptos esenciales del aprendizaje por refuerzo en sí, de una manera progresiva en cuanto a complejidad y número de uso de estas herramientas. El primero de los entornos empleados es una simulación simple de aterrizaje de un cohete en una plataforma. La simplicidad de este entorno ayudará a comprender el funcionamiento de gymnasium y a como realizar entrenamientos de modelos para el agente con SB3. Tras ello, el segundo entorno también implementa MuJoCo, por lo que aportará el conocimiento necesario sobre la compatibilidad entre todas las librerías y como se confecciona un entorno tridimensional en un motor gráfico. Además, el objetivo de este es que un brazo de dos grados de libertad alcance un objetivo concreto en el espacio, si se extrapola, se puede ver como comparte muchas similitudes con

el entorno utilizado.

Posteriormente se hará lo mismo con el robot móvil utilizado para este trabajo, extrapolando todas las ideas aprendidas con la implementación de los otros entornos de ejemplo. Se creará desde cero el entorno utilizando un modelo de un robot sacado de internet, definiendo todas las partes que dan forma a la propia definición de un entorno en aprendizaje por refuerzo.

4.2.1 *Lunar Lander*

El primer entorno utilizado para el desarrollo de este trabajo se define y conforma de las siguientes partes y características:

- **Descripción del entorno:** Este entorno es un problema clásico de optimización de trayectoria de cohetes cuyo objetivo es conseguir que aterrice en la plataforma de manera correcta y de la forma más suave posible. Existen dos versiones del entorno: discreto o continuo, pero para simplificación y según el principio de máxima de Pontryagin (Pontryagin (2018)), que dicta que lo óptimo es encender el motor a máxima potencia o apagarlo, se ha decidido decantarse por las acciones discretas de encender o apagar el motor en diferentes orientaciones.

  En la figura 4.3 se puede observar la representación 2D de Gym de este problema donde la plataforma de aterrizaje siempre está en las coordenadas (0,0), siendo estas los dos primeros números del vector de estado. Existe la posibilidad de aterrizar fuera de la plataforma de aterrizaje, y cabe destacar que el combustible es infinito, por lo que un agente puede aprender a volar y luego aterrizar en su primer intento.

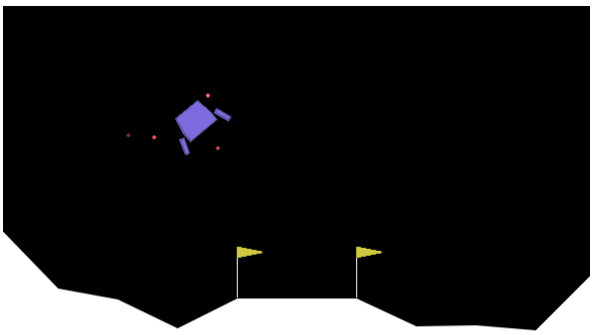


Figura 4.3: Entorno *Lunar Lander*

Fuente: `https://www.gymlibrary.dev/`

- **Espacio de acciones**: El espacio de acciones de este entorno, es muy sencillo y limitado, ya que se trata de un espacio discreto que puede tomar valores del rango de 0 a 3. Las acciones a realizar por el agente son encender cada uno de los motores de manera independiente en diferentes orientaciones o bien no hacer nada y caer (tabla 4.1). Para definir este espacio de acciones se utiliza *Box*, una estructura matemática que permite definir rangos de valores continuos posibles, propia de uno de los módulos de la librería de Gym. Gracias a esta herramienta se definen los espacios continuos que se necesiten, especificando los límites máximos y mínimos, su forma o tamaño, y el tipo de datos del espacio.

Pos. vector	Acción
`0`	No hacer nada
`1`	Encender motor orientado hacia la izquierda
`2`	Encender motor de sustentación (central)
`3`	Encender motor orientado hacia la derecha

Cuadro 4.1: Espacio de acciones del entorno *Lunar Lander*

- **Espacio de observaciones y estados**: Los estados y las observaciones de este entorno estan compuestos de los mismos parámetros ya que el entorno es totalmente observable. El espacio de observaciones del *Lunar Lander* está formado por un vector de tamaño ocho, siendo estas las coordenadas x e y del cohete, sus velocidades lineales en estos

dos ejes, su ángulo, su velocidad angular y dos valores booleanos que representan si cada pata está o no en contacto con el suelo. En la siguiente tabla 4.2 se muestran las observaciones junto su rango de posibles valores que pueden tomar.

Pos. vector	Observación	Min	Max
0	Posición X del cohete	-1.5	1.5
1	Posición Y del cohete	-1.5	1.5
2	Velocidad lineal X del cohete	-5.0	5.0
3	Velocidad lineal Y del cohete	-5.0	5.0
4	Ángulo del cohete	$-\pi$	π
5	Velocidad angular del cohete	-5.0	5.0
6	Contacto con la primera pata	FALSE	TRUE
7	Contacto con la segunda pata	FALSE	TRUE

Cuadro 4.2: Espacio de observaciones del entorno *Lunar Lander*

- **Estado inicial**: El cohete de la simulación comienza en el centro superior de la ventana con una fuerza inicial aleatoria aplicada a su centro de masa. La plataforma de aterrizaje tendrá la misma posición, pero el entorno generado si que cambiará de manera aleatoria de la misma forma.
- **Terminación del episodio**: Un episodio de este entorno acaba si se da alguna de las siguientes situaciones:
  - Si el cohete se estrella (el cuerpo del módulo de aterrizaje entra en contacto con la luna)
  - El módulo de aterrizaje sale de la ventana gráfica (la coordenada x es mayor que 1)
  - Si el cohete no está "despierto", según la documentación de Box2D (herramienta para la vectorizacion del entorno), un cuerpo que no está despierto es un cuerpo que no se mueve y no choca con ningún otro cuerpo.

4.2.2 *Reacher*

El segundo entorno utilizado para el desarrollo de este trabajo, se trata de un agente, que incluyendo MuJoCo, su tarea o comportamiento es "similar" al objetivo se define y conforma

de las siguientes partes y características:

- **Descripción del entorno:** Este entorno, es uno de los entornos disponibles que forma parte de los muchos entornos de MuJoCo disponibles en Gymnasium, el cual presenta una mayor complejidad que el anterior expuesto. Se trata de un brazo robótico con dos articulaciones, cuyo objetivo es mover el efector final del robot cerca de un objetivo que se genera en una posición aleatoria (figura 4.4). Este efector final se encuentra siempre en un plano determinado, pues a pesar de ser un entorno tridimensional, el robot siempre tiene su rango de movimiento limitado a un plano XY.

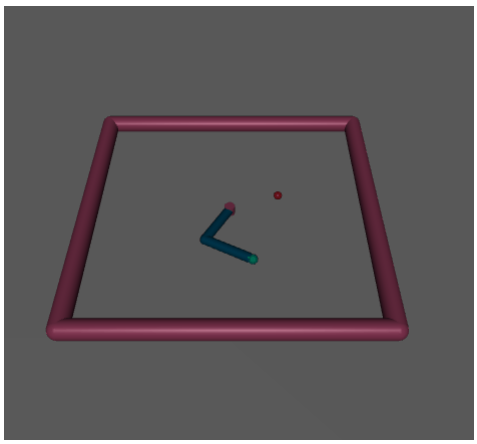


Figura 4.4: Entorno *Reacher*

Fuente: `https://www.gymlibrary.dev/`

- **Espacio de acciones:** El espacio de acciones está formado también por la herramienta *Box*. En este entorno, es un vector de tamaño dos de valores continuos entre -1 y 1 (tabla 4.3), que hacen referencia al torque $(N * m)$ aplicado en cada una de las articulaciones del robot.

Pos. vector	Acción	Min	Max
0	Torque aplicado en la primera articulación	-1.0	1.0
1	Torque aplicado en la segunda articulación	-1.0	1.0

Cuadro 4.3: Espacio de acciones del entorno *Reacher*

- **Espacio de observaciones y estados:** El espacio de observaciones, también continuo

y formado por un *Box*, posee 11 elementos. Este vector incluye las posiciones articulares (senos y cosenos de las articulaciones), las velocidades articulares del robot, así como la posición del efector final del mismo y la posición del punto objetivo a alcanzar. En la siguiente tabla 4.4 se muestra que termino corresponde a cada posición del vector.

Pos. vector	Observación	Min	Max
0-1	Coseno de los ángulos de las articulaciones	$-\infty$	∞
2-3	Seno de los ángulos de las articulaciones	$-\infty$	∞
4-5	Posición del punto objetivo	$-\infty$	∞
6-7	Velocidad angular de cada articulación	$-\infty$	∞
8-10	Posición del extremo del robot	$-\infty$	∞

Cuadro 4.4: Espacio de observaciones del entorno *Reacher*

- **Estado inicial**: El estado inicial del entorno se define como un conjunto de valores que describen las posiciones y velocidades iniciales del brazo junto con su objetivo. De esta forma se genera un entorno controlado pero con variabilidad suficiente para que el agente aprenda a alcanzar el objetivo desde diferentes configuraciones iniciales. Inicialmente, comienza con posiciones y velocidades aleatorias pero limitadas, para la posición acotadas por un rango de [-0.1, 0.1] y para la velocidad de un rango de [-0.005, 0.005]. Para la posición del objetivo, se fija siempre dentro de una distribución esférica uniforme (S) con un radio 0.2, es decir, el objetivo se puede encontrar en cualquier punto dentro de una esfera de radio 0.2 respecto del origen.

- **Terminación del episodio**: El episodio de esta simulación nunca termina como tal, sin embargo, existe algo llamado 'truncamiento', donde se define como duración predeterminada de un episodio 50 iteraciones. Esto se hace para evitar bucles infinitos en los que se pueda quedar atrapado el agente sin posibilidad de alcanzar el objetivo, así como para acelerar tiempos de entrenamientos, descartando modelos que se alejen del objetivo final, haciéndolo muchísimo más eficientes. En caso de que en ese número de iteraciones el brazo alcance el objetivo, se reiniciará de nuevo apareciendo este en una nueva ubicación aleatoria.

4.2.3 Entorno robot móvil

Para el desarrollo de este trabajo se ha realizado una búsqueda e investigación sobre alguna plataforma robótica móvil sobre la que centrar la aplicación del aprendizaje por refuerzo. Concretamente para ello se ha utilizado el *Multi-agent System for non-Holonomic Racing (MuSHR)* (Srinivasa y cols. (2019)). Esta proyecto de código abierto engloba todo el desarrollo *hardware* y *software* de un pequeño robot móvil completamente funcional, cuya finalidad es precisamente idéntica a la de este trabajo como es aprender e investigar sobre la aplicación de IA en el entorno de vehículos autónomos y robótica móvil.

Figura 4.5: *Multi-agent System for non-Holonomic Racing*

Fuente: `https://mushr.io/`

A continuación, se describen las principales características del robot utilizado:

- El robot está montado sobre un chasis de escala 1/10 de un coche de rally, lo que le proporciona una estructura compacta y robusta. Cuenta con tracción a cuatro ruedas, lo que asegura una mayor capacidad de movimiento en diferentes tipos de terreno. El sistema de propulsión está basado en un motor de corriente continua con escobillas, combinado con un controlador electrónico de velocidad, adecuado para velocidades entre 0-30 mph (0-48km/h).

- El robot está diseñado para ser accesible económicamente pues el costo oscila entre los 610 dólares para la versión sin sensores, y hasta 930 dólares para la versión totalmente equipada.

- La plataforma está basada en ROS (*Robot Operating System*), lo que permite probar

sistemas de control clásicos ya existentes, así como otros tipos de algoritmos de una manera sencilla en simulación.

- Incluye un sensor LiDAR de 360 grados, específicamente el modelo YDLiDAR X4, particularmente útil para tareas de mapeo y navegación, proporcionando una visión precisa del entorno manteniendo la premisa del bajo costo.

- También disponde de una IMU (Unidad de Medición Inercial), que permite medir el movimiento y la orientación del vehículo en todo momento, especialmente útil para implementación de algoritmos de SLAM (Simultaneous Localization and Mapping) para la localización y mapeo.

- Para el procesamiento visual y la interacción con el entorno, el robot utiliza cámaras Intel RealSense RGBD, que proporcionan nubes de puntos en 3D. Estas cámaras son fundamentales para tareas como la evitación de obstáculos y el seguimiento de objetos, ya que permiten una percepción detallada del entorno que ayuda en la toma de decisiones del robot.

Además de las características mencionadas, en la página oficial del robot, es posible encontrar tutoriales para la construcción física del mismo, así como para su simulación en diferentes entornos. Además, al ser de código abierto, es relativamente sencillo obtener el código del modelado del robot en diferentes formatos como XML o URDF.

Para este trabajo, se han utilizado unos archivos como base del modelado, y a partir de ellos se ha confeccionado el entorno propio en Gym utilizando sus propias herramientas, ya que no existía previamente ningún entorno que ya implementase este robot móvil o alguno de similares características. En un repositorio de GitHub, del autor Schmittle (2020), integrante del grupo de investigación que ha desarrollado el robot utilizado, se puede encontrar una serie de archivos XML que contienen el robot ya adaptado para ser simulado con MuJoCo. Estos archivos han servido de base para todo el desarrollo del trabajo, pues sobre ellos se ha ido trabajando para añadir y modificar funcionalidades.

- **Descripción del entorno:** En lo referente al entorno, se trata de una simulación del robot MuSHR en un plano bidimensional. El objetivo del agente es minimizar la

distancia al punto rojo objetivo, que se genera aleatoriamente dentro de un anillo definido por un radio mínimo y máximo alrededor del origen como se puede ver en la figura 4.6. Para añadir el punto que el agente debe alcanzar, se ha recurrido a las líneas correspondientes del modelo del Reacher (archivo XML) que incluyen dicho punto, y han sido añadidas al propio modelo del robot móvil de tal forma que sea visible, y su posición sea modificable para cada reseteo del entorno. El agente tendrá como objetivo aprender gracias a recibir recompensas basadas en el progreso hacia el objetivo entre otros aspectos que se detallarán en el desarrollo del trabajo.

Figura 4.6: Entorno Robot Móvil

- **Espacio de acciones:** El espacio de acciones es continuo y está definido como un *Box* bidimensional, sin embargo, su definición no se hace de manera explicita en el código del entorno de Gym, si no que este se construye de manera automática a raíz de las etiquetas *<actuator>*, las cuales definen los actuadores disponibles para controlar en el modelo del archivo XML del propio robot. Cada acción incluye dos componentes: el giro aplicado a las ruedas delanteras y la velocidad general aplicada al robot, cuyos valores de las acciones están acotados entre los rangos de la tabla 4.5. Para poder limitar cada componente del espacio de acciones, también se ha de limitar los actuadores definiendo en el XML añadiendo el parámetro *ctrllimited* como *true*, e indicando el rango de valores que puede tomar con *ctrlrange*.

- **Espacio de observaciones y estados:** El espacio de observaciones también es continuo y viene definido por un*Box* que contiene 11 elementos. Este sí se define de manera

Pos. vector	Acción	Min	Max
0	Giro de las ruedas delanteras	-0.38	0.38
1	Velocidad aplicada al robot	-0.3	0.3

Cuadro 4.5: Espacio de acciones del entorno *MuSHR*

explicita en la definición del entorno de Gym a diferencia del espacio de acciones. El vector incluye información sobre la posición relativa entre el robot y el objetivo, la distancia al objetivo, la posición absoluta y orientación del robot, y la posición del objetivo. La tabla 4.6 detalla las observaciones:

Pos. vector	Observación	Min	Max
0-2	Posición relativa al objetivo (XYZ)	$-\infty$	∞
3	Distancia al objetivo	0	∞
4-6	Posición absoluta del robot (XYZ)	$-\infty$	∞
7	Orientación del robot (ángulo)	$-\pi$	π
8-10	Posición del objetivo (XYZ)	$-\infty$	∞

Cuadro 4.6: Espacio de observaciones del entorno *MuSHR*

- **Estado inicial:** El estado inicial del entorno comienza con el robot en una posición de partida considerada como origen y partiendo de una condición de reposo, es decir, sin ninguna velocidad inicial. Por otro lado, la posición del objetivo se define en un anillo entre un radio mínimo de 1.5 y máximo de 4.0 unidades, asegurando una distribución uniforme en ese rango. Esta distribución ha sido escogida ya que al tratarse de un robot no holonómico debido a sus características, para posiciones muy cercanas del objetivo a la posición inicial de referencia u origen, el robot encuentra grandes dificultades para alcanzarlo, ya que para ello tiene que realizar una gran cantidad de maniobras lo que dificulta el proceso de entrenamiento.

- **Terminación del episodio:** Un episodio termina si el robot se acerca a menos de 0.3 unidades del objetivo, momento en el que recibe una recompensa adicional y el episodio finaliza exitosamente. En caso de no darse esta situación, los episodios están limitados a una duración máxima de 1000 pasos, donde si un episodio alcanza este límite sin que se cumpla ninguna condición de terminación (*terminated o truncated*), Gym considerará

que el episodio ha terminado. Este límite es útil para evitar que los episodios duren indefinidamente si el objetivo no se alcanza o si no hay un criterio de terminación claro, pudiendo agilizar el proceso de entrenamiento del agente y evitando bucles infinitos.

4.3 Algoritmos

Existen una gran cantidad de algoritmos aplicables en el aprendizaje por refuerzo, concretamente Stable-Baselines3, que es la herramienta utilizada para estos aprendizajes, ofrece una amplia y casi completa gama de algoritmos de entrenamiento. Cada uno de estos algoritmos está diseñado para abordar diferentes tipos de problemas, según la naturaleza del entorno y el tipo de espacio de observación y acción. En la tabla 4.7 se pueden observar todos los algoritmos disponibles en esta herramienta de entrenamiento y su compatibilidad con diferentes tipos de entornos. Dichos algoritmos pueden ser clasificados en función de si permiten alguno de los siguientes tipos de espacio de acciones o características:

- **Continuo**: representa un rango infinito de posibles acciones dentro de límites específicos. Estas están representadas como valores reales (números continuos), y se definen en Gym con la herramienta *Box* como ya se ha visto. Estos son típicos de aplicaciones reales como la propia robótica, o la propia aplicación sobre la que se está investigando en este trabajo.

- **Discreto**: son un conjunto finito de acciones posibles, donde cada acción está etiquetada con un número entero o índice. Sus aplicaciones son propias de juegos de mesa o el control de un personaje en un videojuego.

- **Multidiscreto**: es una generalización del anterior tipo, permitiendo definir varios conjuntos discretos de acciones independientes. Más concretamente son sistemas que requieren múltiples decisiones independientes al mismo tiempo, de carácter discreto cada una de ellas.

- **Multibinario**: representa un conjunto de acciones binarias únicamente como *true o false*. Al igual que en el anterior tipo, son sistemas donde múltiples elementos deben ser controlados de forma independiente, pero exclusivamente con opciones binarias.

- **Multiprocesado**: este termino se refiere a la capacidad de ejecutar varios entornos de simulación en paralelo, utilizando múltiples procesos de la CPU. Esto puede acelerar el entrenamiento al permitir que un agente recopile datos de experiencias simultáneamente en diferentes instancias de un mismo entorno. El objetivo pasa por tanto por conseguir que el entrenamiento aproveche al máximo los recursos del procesador, ya que no espera a que una simulación termine antes de iniciar otra, repercutiendo en el tiempo de entrenamiento y la cantidad de datos de los que dispone el agente para este proceso.

Nombre	Continuo	Discreto	MultiDiscreto	MultiBinario	MultiProcesado
ARS	✓	✓	✗	✗	✓
A2C	✓	✓	✓	✓	✓
CrossQ	✓	✗	✗	✗	✓
DDPG	✓	✗	✗	✗	✓
DQN	✗	✓	✗	✗	✓
HER	✓	✓	✗	✗	✓
PPO	✓	✓	✓	✓	✓
QR-DQN	✗	✓	✗	✗	✓
RecurrentPPO	✓	✓	✓	✓	✓
SAC	✓	✗	✗	✗	✓
TD3	✓	✗	✗	✗	✓
TQC	✓	✗	✗	✗	✓
TRPO	✓	✓	✓	✓	✓
Maskable PPO	✗	✓	✓	✓	✓

Cuadro 4.7: Algoritmos de entrenamiento disponibles en SB3

De todos los algoritmos disponibles en esta herramienta, se suelen emplear dos por norma general debido a sus características: *A2C y PPO*. Estos algoritmos son muy populares porque ofrecen un buen balance entre simplicidad, estabilidad, rendimiento y velocidad durante todo el proceso de entrenamiento. Además estos algoritmos, son los dos únicos disponibles, compatibles con todos los espacios de acciones, lo que los hace a su vez, mucho más versátiles.

Para el desarrollo del entrenamiento de los entornos de *Lunar Lander y Reacher* se emplearán ambos algoritmos, sin embargo, para el entrenamiento del entorno del robot móvil únicamente se utilizará PPO. La capacidad de este algoritmo para manejar entornos dinámicos sin perder estabilidad hace que sea más adecuado frente a A2C, que, aunque efectivo,

podría ser menos robusto en situaciones complejas. Esto también se verá reflejado en una comparativa que se realizará sobre las recompensas obtenidas en función del tiempo para valorar cual de los dos es más adecuado. Por estas razones, el desarrollo del proyecto se realizará entorno al algoritmo de entrenamiento PPO. A continuación se exponen sus características y los detalles sobre su funcionamiento:

4.3.1 *Proximal Policy Optimization* (PPO)

El algoritmo PPO (Schulman y cols. (2017)) es una técnica de aprendizaje por refuerzo que pertenece a la familia de los algoritmos *model-free* basados en política. PPO nace como una extensión de TRPO (*Trust Region Policy Optimization*), centrándose en mejorar la estabilidad y eficiencia del entrenamiento al optimizar la política de forma más controlada, utilizando un enfoque que evita grandes actualizaciones en los parámetros del modelo, lo que podría llevar a un rendimiento inestable.

Como ya se ha visto a lo largo del marco teórico (2), en el aprendizaje por refuerzo, el agente interactúa con su entorno y aprende a tomar decisiones mediante recompensas. Para lograrlo, los agentes pueden usar políticas para determinar la acción a seguir en cada estado. Los métodos más tradicionales, como el Método de Gradiente de Política y Q-learning, normalmente se ven afectados por una alta variabilidad en las actualizaciones de la política, lo que puede generar entrenamientos ineficientes. Por su lado, PPO se basa en la optimización de la política proximal, lo que significa que la política que se optimiza no puede alejarse demasiado de la política anterior. De esta manera, se asegura que las actualizaciones de la política sean lo suficientemente grandes como para mejorar el rendimiento, pero no tanto como para empeorar el rendimiento debido a un cambio brusco. Además introduce el concepto de coeficiente de clip, cuya función es precisamente restringir esas actualizaciones de la política. Este coeficiente actúa como un límite que impide que los valores de probabilidad de las acciones cambien demasiado, ayudando a prevenir grandes oscilaciones y mejorando la estabilidad del entrenamiento de manera global.

En conclusión, PPO es un algoritmo de optimización de políticas muy eficiente y estable para el aprendizaje por refuerzo, cuya eficacia está muy demostrada en problemas prácticos complejos. A través de la optimización proximal y el método de recorte, PPO logra evitar

grandes cambios en la política, manteniendo un entrenamiento robusto y rápido, para entornos y espacios de acciones de todo tipo.

5 Desarrollo

Este apartado se centra en el desarrollo y el proceso de experimentación del trabajo en cuestión, siguiendo todas y cada una de las pautas y objetivos mencionados a lo largo de este. Para facilitar el entrenamiento del robot móvil, se ha realizado una implementación previa de los entornos expuestos en el apartado 4.2, pudiendo trabajar en un primer lugar, con entornos ya predefinidos que funcionan correctamente y que a su vez son más sencillos. El proceso de experimentación por tanto consistirá en la modificación de ciertos parámetros y aspectos del entorno con la finalidad de aproximar cada vez más el comportamiento del robot al deseado. La recompensa que obtenga el agente en cada iteración, jugará el papel principal en su entrenamiento, pues es el aspecto del entorno que mejor define cómo se comporta en un entorno determinado. De manera iterativa e intentando abordar diferentes estrategias se añadirán diferentes componentes cuyo sumatorio formen la recompensa total que obtendrá el robot. Asimismo, los resultados obtenidos en la experimentación se irán recopilando y reflejando de manera gráfica para mejorar la interpretación del comportamiento del robot.

5.1 Entrenamiento entorno *Lunar Lander*

El primero de los entornos con el que se ha comenzado el proceso de experimentación de este trabajo es el entorno *LunarLander*. En resumen, este entorno tiene como objetivo el de modelar el comportamiento de un módulo lunar para aterrizarlo suavemente en una plataforma. Para ello, el entorno implementa una función de recompensa que incentiva comportamientos deseables y penaliza acciones que no lo son.

El entorno en si y como se construye la recompensa tiene cierto nivel de complejidad, sin embargo, la lógica con la que están implementados cada una de los componentes que la conforman hace que sea un entorno ideal para entender como funciona el aprendizaje por

refuerzo. Simplemente se ejecutará un código de entrenamiento genérico sobre el entorno, sin modificar ningún parámetro de este o de las recompensas, utilizándolo tal y como se define por defecto. Para cada paso se otorga una recompensa determinada conformada (sumatorio) por diferentes aspectos del agente dentro de la simulación. Asimismo, la suma de las recompensas de todos los pasos de un episodio, conformará la recompensa total de este, cuyo valor será muy importante para valorar el comportamiento de manera directa del agente en cuestión.

El entorno está definido de tal manera que utiliza una recompensa "principal" para evaluar diferentes aspectos del estado del *lander* e incentivar un aterrizaje eficiente y controlado. Estos valores se recogen en una variable con el nombre de *shaping* y a continuación se desglosará cada una de sus componentes.

- **Recompensa por posición respecto del objetivo:** Esta parte de la recompensa penaliza al agente si este se encuentra lejos del objetivo, que es la base de aterrizaje, en el espacio de coordenadas del entorno. La fórmula empleada para ello, calcula la distancia del lander a dicha base utilizando Pitágoras, la diferencia en las coordenada x e y, y multiplicando dicha distancia por -100. De esta forma se fomenta que el *lander* se acerque al objetivo, ya que cuanto más lejos esté del objetivo, mayor será la penalización.

$$R_{\text{posición}} = -100 \cdot \sqrt{x^2 + y^2} \tag{5.1}$$

- **Recompensa por velocidades de la nave:** de una forma similar a la anterior, el agente es penalizado si tiene velocidades horizontales o verticales, ya que el objetivo es aterrizar con velocidad lo más cercana a cero en ambas direcciones, para así fomentar un aterrizaje suave, sin movimiento horizontal o vertical. La fórmula utilizada sigue el mismo patrón, calculando el módulo de la velocidad y multiplicándolo por -100.

$$R_{\text{velocidad}} = -100 \cdot \sqrt{v_x^2 + v_y^2} \tag{5.2}$$

- **Penalización por inclinación de la nave:** el agente es penalizado si el ángulo que presenta en ese paso no es 0, es decir, no se encuentra en horizontal. El objetivo es aterrizar con el *lander* completamente horizontal, por lo que cualquier ángulo distinto

a cero recibe una penalización acorde a este valor.

$$R_{\text{ángulo}} = -100 \cdot |\theta| \tag{5.3}$$

- **Recompensa por contacto con el suelo (aterrizaje):** La última componente que conforma la recompensa *shaping* otorga recompensas de +10 cuando las piernas de la nave hacen contacto directo con el suelo. Así se consigue incentivar el aterrizaje, asegurando que intente aterrizar sobre el suelo, estabilizándose en esa posición, pues cuanto mayor tiempo tenga en contacto su tren de aterrizaje con el suelo, mayor será su recompensa obtenida en ese episodio.

$$R_{\text{patas}} = 10 \cdot P_{izq} + 10 \cdot P_{dcha} \tag{5.4}$$

La recompensa *shaping* estará formada por todas las componentes mostradas hasta ahora de la siguiente manera:

$$\text{shaping} = R_{\text{posición}} + R_{\text{velocidad}} + R_{\text{ángulo}} + R_{\text{patas}} \tag{5.5}$$

Asimismo, el entorno del *lander* modela otras situaciones que también son incluidas en el cálculo de la recompensa final:

- **Penalización por uso de combustible:** El *lander* utiliza combustible tanto para controlar la orientación como para mantener la velocidad vertical y horizontal durante el aterrizaje. El uso excesivo de combustible está penalizado, y para ello emplea una fórmula acumulativa, en la que resta el combustible utilizado por el motor principal (vertical) ”m” por 0.30. Así se consigue incentivar el uso eficiente del combustible, consumiendo lo menos posible para conseguir el objetivo de un aterrizaje controlado y eficiente:

$$R_{\text{motor\_principal}} = -0.30 \cdot m \tag{5.6}$$

De la misma manera que para el motor principal, también se calcula una penalización, aunque en una menor proporción, para el uso de los motores laterales ”s”, cuya función

principal es la de ajustar la orientación de la nave:

$$R_{\text{motores\_orientación}} = -0.03 \cdot s \tag{5.7}$$

Ambas penalizaciones por el uso de los diferentes motores de la nave se aplican de forma conjunta en la recompensa final que el agente recibirá:

$$R_{\text{combustible}} = R_{\text{motor\_principal}} + R_{\text{motores\_orientación}} \tag{5.8}$$

- **Recompensa por finalización de la simulación:** la simulación se puede terminar bajo dos posibles circunstancias: la nave se aleja demasiado de la plataforma de aterrizaje o bien, aterriza correctamente en el lugar objetivo. Si el *lander* se desplaza demasiado lejos de la zona de aterrizaje, es decir, si la coordenada X de la nave en valor absoluto es mayor o igual a 1, significará que la nave ha salido de la ventana gráfica y el juego terminará y se le asignará al agente una penalización de -100. Con ello se impide que el agente se aleje demasiado del objetivo de aterrizaje, acotando la zona especifica de movimiento.

-

$$R_{\text{fin}} = \begin{cases} -100 & \text{si } |x| \geq 1.0 \\ 0 & \text{en caso contrario} \end{cases} \tag{5.9}$$

La otra posibilidad de finalización es que el *lander* aterrice exitosamente como ya se ha comentado. Para ello se comprueba que el robot no esté ”despierto”, un indicativo de gym que denota que el agente se encuentra parado y estabilizado en su simulación. En tal caso, la nave habrá aterrizado, y el juego terminará, recibiendo el agente una recompensa de +100. El aterrizaje exitoso será recompensado pues es el objetivo que en definitiva ha de aprender el agente.

$$R_{\text{fin}} = \begin{cases} +100 & \text{si está estabilizado y ha aterrizado correctamente} \\ 0 & \text{en caso contrario} \end{cases} \tag{5.10}$$

Cabe destacar que en el entorno, el valor de *shaping* se actualiza en cada paso, y de la misma manera, para cada paso se aplican las penalizaciones por el uso de combustible, mientras que Rfin se aplica solo cuando la simulación finaliza. Además un episodio será considerado como solución, si puntúa al menos 200 puntos, gracias a la función de recompensa final que viene definida como:

$$R_{\text{total}} = \text{shaping} + P_{\text{combustible}} + R_{\text{fin}}* \tag{5.11}$$

Tras haber explicado cada una de las diferentes componentes que conforma la recompensa final obtenida por el agente, se detallará el proceso de entrenamiento llevado a cabo para el *lander*. Como ya se ha explicado en el apartado 4.3 se utilizarán los algoritmos A2C y PPO implementados por la librería SB3, con el objetivo de poder compararlos y realizar un análisis, y de esta forma poder escoger cual se adecúa más a las necesidades del entorno que se va a implementar posteriormente, y en definitiva, descubrir cual tiene un mejor rendimiento.

Al tratarse de un entorno relativamente simple, no es necesario realizar una gran cantidad de pasos para sacar conclusiones satisfactorias, ya que el agente aprenderá un comportamiento que cumpla con los objetivos con rapidez. Para este entrenamiento en concreto se han ejecutado tres simulaciones en paralelo con A2C y otras tres con PPO, llegando para cada una de ellas, al menos, hasta las 900.000 iteraciones. El número es bastante alto, y con menos se pueden obtener ya conclusiones, sin embargo, debido a la rapidez de entrenamiento por la falta de requisitos en los recursos gráficos de la simulación, se ha decidido hacer un entrenamiento algo más prolongado para así poder ver el funcionamiento y la estabilidad de ambos algoritmos a largo plazo.

En un primer lugar, se han obtenido dos gráficas tras el entrenamiento con A2C, mostrando por un lado la longitud media de los episodios, indicativo de si el *lander* aterriza cada vez más rápido, y por otro, la recompensa media obtenida por episodio como tal.

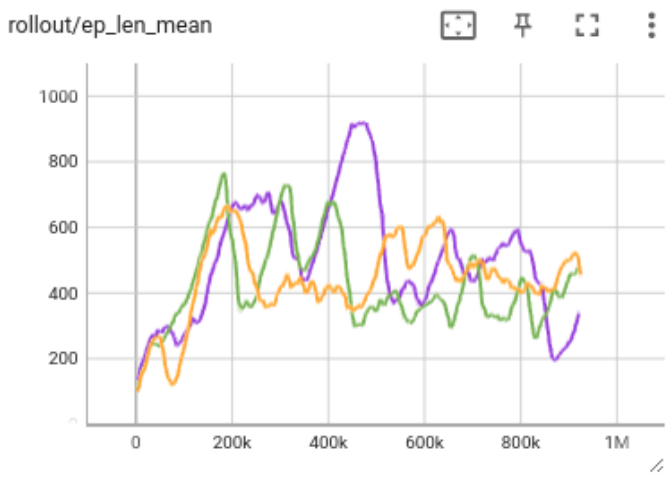


Figura 5.1: Longitud por episodio media de LunarLander con A2C

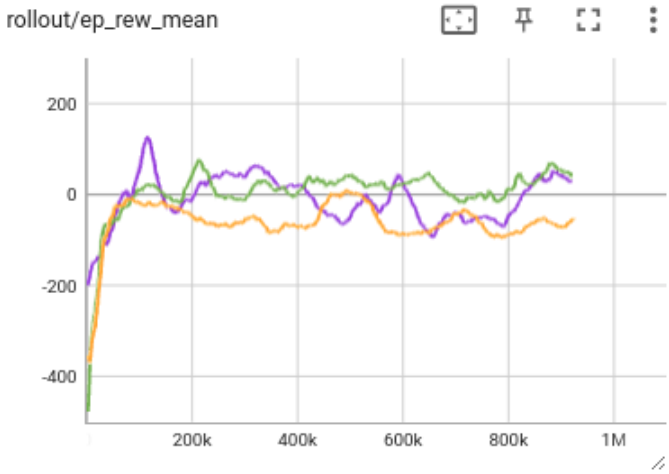


Figura 5.2: Recompensa por episodio media de LunarLander con A2C

De la misma manera se han obtenido otros resultados para ambos parámetros evaluados utilizando el algoritmo de PPO.

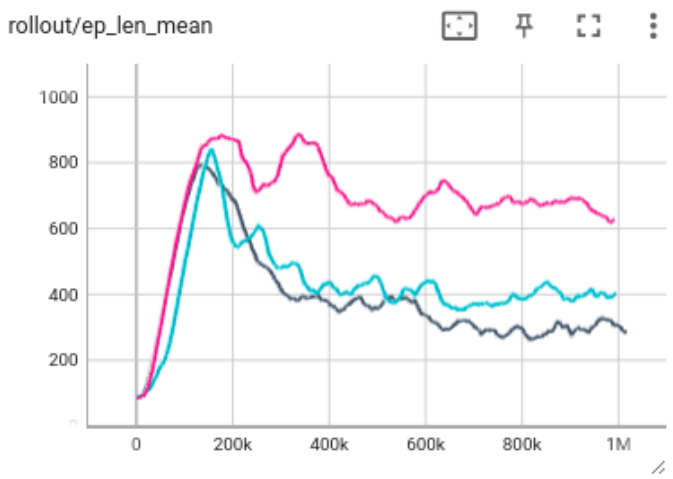


Figura 5.3: Longitud por episodio media de LunarLander con PPO

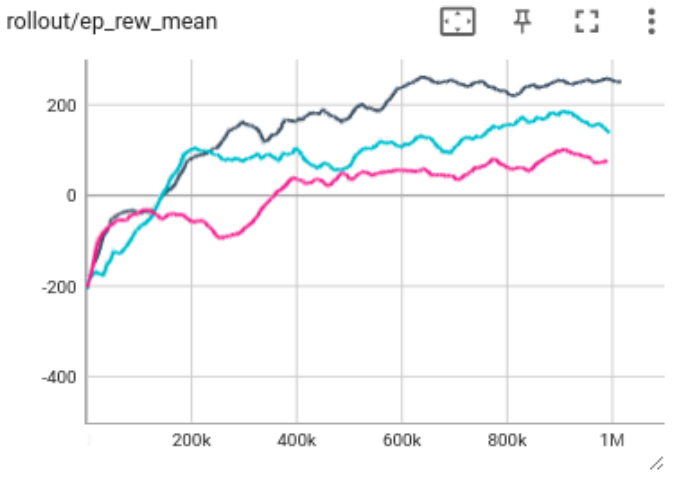


Figura 5.4: Recompensa por episodio media de LunarLander con PPO

De un vistazo general, es posible observar como ambos algoritmos consiguen que la recompensa aumente a medida que avanzan los pasos del entrenamiento de la simulación, por lo que se está consiguiendo que el agente aprenda un comportamiento. También se puede destacar, que de manera general, tanto el tiempo medio empleado para un episodio, como la recompensa media obtenida, son mucho más inestables para el algoritmo de A2C. Esto tiene sentido, ya que como se ha visto en el apartado 4.3.1, PPO mejora precisamente esa inestabilidad presente en otros algoritmos, limitando lo que puede cambiar la política entre los

episodios del entrenamiento. El otro aspecto clave de estos entrenamientos, es que se puede observar como PPO obtiene mejores recompensas duraante todo el proceso de entrenamiento y sobretodo a largo plazo, donde PPO es capaz de conseguir recompensas en torno a 200 y A2C consigue máximos regulares de en torno a solamente 50.

En resumen, gracias al estudio realizado para el entorno *LunarLander*, es posible asegurar que PPO consigue un comportamiento mucho más estable frente a A2C. Y no solo eso, si no que además consigue resolver el problema consigue obtener recompensas entorno a 200 e incluso superarlas tal y como se establece en la definición del entorno.

5.2 Entrenamiento entorno *Reacher*

Continuando con el el proceso de experimentación previo al que se va a realizar con el entorno del robot móvil, se llevó a cabo el entrenamiento del entorno *Reacher*. Además de seguir indagando sobre el aprendizaje por refuerzo, este entorno toma vital importancia, ya que implementa la librería Gym junto con MuJoCo, de la misma manera a como se hará con la implementación del robot sobre el que se centra este trabajo, sirviendo como estructura base para desarrollar el del robot móvil. En cuanto al entorno *Reacher*, consiste en un brazo robótico de dos grados de libertad cuyo objetivo es el de alcanzar una posición objetivo generada aleatoriamente en cada episodio con su extremo.

Tal y como se ha actuado el anterior entorno, se ha ejecutado un programa de entrenamiento genérico sobre el entorno, utilizando los parámetros de las recompensas por defecto. En lo referente a como se confecciona la recompensa, está es incluso más simple que la anteriormente vista, pues unicamente depende de dos componentes. La recompensa que el agente obtiene en cada paso de los diferentes episodios viene definida por tanto de la siguiente manera:

- **Distancia respecto del objetivo:** Esta parte de la recompensa penaliza al agente en función de la distancia entre la punta del brazo (*fingertip*) y el objetivo (*target*), y la diferencia de magnitud que hay entre ellas. Para calcular esta distancia, se utiliza la norma euclidiana entre las coordenadas de ambos puntos en el espacio 2D. La penalización es proporcional al valor negativo de dicha distancia, incentivando que el brazo robótico reduzca al mínimo esta separación, y por tanto, se acerque lo máximo posible

al objetivo.

$$R_{\text{distancia}} = -|\vec{d}| = -\sqrt{(x_{\text{fingertip}} - x_{\text{target}})^2 + (y_{\text{fingertip}} - y_{\text{target}})^2} \tag{5.12}$$

Donde $\vec{d}$ es el vector que representa la diferencia entre la posición del *fingertip* y el *target*.

- **Control del movimiento:** Para evitar movimientos innecesarios o bruscos del agente, se introduce una penalización basada en el cuadrado de todas las acciones realizadas en cada paso, evitando que aplique torques muy elevados sobre las articulaciones. Este término fomenta que el agente utilice las menores fuerzas posibles para alcanzar el objetivo, logrando un comportamiento lo más eficiente y estable posible.

$$R_{\text{control}} = -\sum_i a_i^2 \tag{5.13}$$

Donde a_i representa las acciones aplicadas a los actuadores del brazo robótico en ese paso.

- **Recompensa total:** La recompensa total en este entorno es una combinación lineal de las dos componentes descritas anteriormente: la penalización por la distancia al objetivo y por el torque aplicado en cada articulación, asegurando así alcanzar el objetivo de una manera controlada y eficiente.

$$R_{\text{total}} = R_{\text{distancia}} + R_{\text{control}} \tag{5.14}$$

Una vez explicada la recompensa final obtenida por el agente y las dos componentes que la conforman, se llevará a cabo el proceso de entrenamiento del agente. De la misma manera que para el anterior entorno, se utilizarán los algoritmos A2C y PPO implementados por la librería SB3 para tomar una decisión definitiva de que algoritmo emplear en el entorno del robot móvil. Concretamente en este entorno, el modelo unicamente depende de dos parámetros a la hora de obtener una recompensa por lo que es mucho más sencillo modelar el comportamiento y poder visualizarlo gráficamente. A priori, puede parecer un entorno más sencillo, sin embargo,

la implementación de una simulación de físicas en MuJoCo hace que el entorno sea mucho más pesado computacionalmente, y que por consiguiente, los entrenamientos requieran de un *hardware* más potente. El entrenamiento que se ha llevado a cabo alcanza las 300.000 iteraciones para cada una de las simulaciones ejecutadas en paralelo, tres con A2C y otras tres con PPO.

Primeramente, para el entrenamiento con el algoritmo A2C, se han obtenido los siguientes resultados de la recompensa media obtenida por episodio:

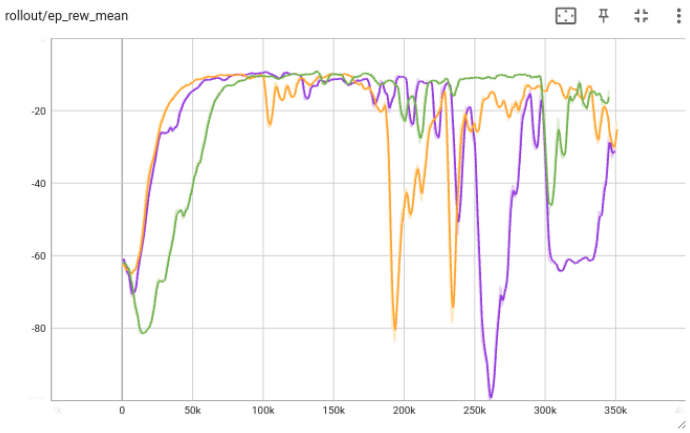


Figura 5.5: Recompensa por episodio media de Reacher con A2C

En ella se puede observar como los diferentes modelos entrenados con A2C alcanzan rápidamente una recompensa valle de entorno a un valor de -10. La inestabilidad que caracteriza a este algoritmo debido a como esta confeccionado hace que se encuentre una posible solución de manera muy eficaz en los primeros pasos del entrenamiento. Por consiguiente, esta inestabilidad, provoca que el algoritmo sea muy poco fiable en entrenamientos muy largos, ya que se puede observar como alcanza grandes picos negativos en repetidas ocasiones, y como nunca es capaz de alcanzar una recompensa mayor que la obtenida en los inicios.

De la misma manera, se ha realizado un entrenamiento con PPO, cuyos resultados de recompensa media por episodio son los siguientes:

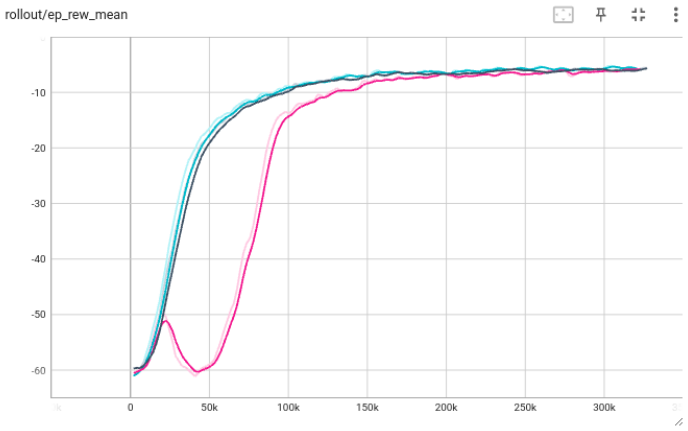


Figura 5.6: Recompensa por episodio media de Reacher con PPO

Es sencillo destacar como el comportamiento mostrado por el modelo de PPO en el entrenamiento es mucho más suave y estable. A su vez PPO es capaz de asemejarse prácticamente a A2C en cuanto a la obtención de una recompensa relativamente buena en poco tiempo, pues antes de las 100.000 iteraciones es capaz de alcanzar una recompensa de -10 en al menos dos de los tres entrenamiento ejecutados en paralelo. La característica que más destaca de este algoritmo como ya se vio en el anterior entorno, es que es capaz de mantener una estabilidad a lo largo de muchas iteraciones. La recompensa a partir de esas 100.000 iteraciones se estabiliza y aumenta progresivamente, mejorando conforme pasas las iteraciones dele entrenamiento.

En definitiva, PPO se presenta como un algoritmo mucho más robusto frente a A2C. A pesar de que pueda ser efectivo para encontrar un comportamiento relativamente óptimo para un agente en un entorno sencillo, para un numero limitado de iteraciones, PPO es claramente una mejor solución para el entrenamiento del robot móvil que se va a implementar. Dicho entorno es más complejo y no depende unicamente de dos variables para definir su recompensa. Además, al tratarse de un entorno que utiliza las físicas de MuJoCo para la simulación, el entrenamiento será más pesado que los anteriores, por lo que se deberá entrenar con hardware con un cierto nivel de exigencia y muchas iteraciones para encontrar un comportamiento que se asemeje al esperado. Por tanto, el algoritmo escogido para el entrenamiento del robot móvil será PPO, pues gracias a él se obtendrán unas recompensas mucho más estables y fiables en un número muy alto de episodios del entrenamiento.

5.3 Entrenamiento entorno Robot Móvil

Una vez implementados los dos anteriores entornos, y visto el funcionamiento de las herramientas que se van a emplear en lograr los objetivos del trabajo, se va a integrar todo lo aprendido con ellos, en el entrenamiento del robot móvil. En este entorno usaremos MuJoCo como simulador de físicas, y para su entrenamiento se usará el algoritmo de PPO ya que se han obtenido resultados mucho más sólidos para entornos que requieran de gran entrenamiento gracias a su estabilidad. En referencia al propio objetivo del entorno, este pasa por emular un planificador de trayectorias, simulando el robot en una superficie plana, teniendo que alcanzar un objetivo en el mapa representado como un punto de color rojo.

La forma de proceder con el entrenamiento es idéntica a los otros entornos previos, pues se limitará a entrenar el algoritmo en diferentes ramas, con el objetivo de obtener un modelo que obtenga el mejor comportamiento posible del agente. La recompensa obtenida de la interacción del agente con el entorno, y su toma da decisiones a lo largo de la simulación y los episodios, definirá el comportamiento del modelo entrenado, por lo que será vital elaborar una recompensa lógica y adecuada a los objetivos. La función de recompensa inicial que se plantea dispone de los siguientes términos:

- **Penalización por distancia al objetivo:** cuanto más lejos esté el robot del objetivo, la recompensa será más negativa y por tanto perjudicará de manera directa a la recompensa obtenida en cada iteración del episodio. De esta manera se consigue incentivar estar lo más cerca posible del objetivo durante todo el episodio. Para cacular esta distancia, de manera similar a como se hizo con el *Reacher* se utiliza la norma euclidiana entre las coordenadas del robot y el objetivo:

$$R_{\text{distancia}} = -|\vec{d}| = -\sqrt{(x_{\text{robot}} - x_{\text{target}})^2 + (y_{\text{robot}} - y_{\text{target}})^2} \tag{5.15}$$

- **Penalización por control:** penaliza de manera moderada la toma de acciones muy grandes, es decir, cambios bruscos de giro o velocidades muy altas. Se realiza un cálculo donde esta recompensa aumenta cuadráticamente con la magnitud de las acciones (velocidad lineal y angular), y es multiplicada por una constante, que puede ser modificada para encontrar el mejor equilibrio, castigando esos movimientos bruscos. De

esta manera se incentiva una conducción más suave y eficiente, evitando también así inestabilidades en el sistema.

$$R_{control} = -\sum_{i} a_i^2 * 5 \tag{5.16}$$

- **Recompensa por progreso:** a mayores se calcula cuánto se ha reducido la distancia al objetivo con respecto al paso anterior del mismo episodio, es decir se calcula un progreso. Si el robot se aleja, el progreso será negativo y también penalizará como las anteriores componentes, en cambio si se acerca, se otorgará una recompensa positiva proporcional a esa distancia avanzada. Mediante una constante ajustable se le da peso a esta componente haciendo que tome un mayor o menor protagonismo.

$$R_{progreso} = (Dist_{i-1} - Dist_i) * 500 \tag{5.17}$$

- **Penalización por tiempo:** se realiza una pequeña penalización fija en cada paso, introduciendo así un coste temporal en la tarea del agente, y de esta manera fomentar que el objetivo se intente alcanzar más rápidamente y de manera eficiente. La idea también considera intentar que el agente no se quede girando en las simulaciones sin avanzar, haciendo así que el tiempo juegue en su contra de alguna manera.

$$R_{tiempo} = -0.1 \tag{5.18}$$

- **Recompensa total:** La recompensa total en este entorno es la acumulación de las componentes descritas anteriormente:

$$R_{\text{total}} = R_{\text{distancia}} + R_{\text{control}} + R_{\text{progreso}} + R_{\text{tiempo}} \tag{5.19}$$

- **Recompensa por alcanzar el objetivo:** si el robot se encuentra a una distancia menor a 0.3 unidades del objetivo, se considera que ha sido alcanzado. Siendo este el principal objetivo del agente se otorga una gran recompensa extra (+500) y el episodio

se finaliza para comenzar de nuevo con otro.

$$distancia < 0.3 => R_{total} += 500; \quad terminated = true \tag{5.20}$$

El diseño de esta función de recompensa impulsa al agente a acercarse progresivamente al objetivo, evitando en la medida de lo posible acciones bruscas o muy extremas. Además se tiene en cuenta la eficiencia de la trayectoria tomada, penalizando la recompensa en función de la duración del episodio. De la misma manera se premia el finalizar el episodio con éxito, pues es el objetivo final del comportamiento buscado.

Tras explicar como se ha confeccionado la recompensa total del entorno creado para el desarrollo del trabajo, se entrenará el agente, al igual que con los entornos anteriores, para comprobar si es capaz de aprender el comportamiento descrito de alcanzar el objetivo de una manera "relativamente" eficiente. Cabe destacar la importancia vital de la elaboración de un entorno de calidad que se ajuste a las especificaciones y a cada objetivo según el agente empleado. El entorno definirá el funcionamiento del agente, por lo que dependerá en gran parte lograr replicar el comportamiento buscado de su construcción. Inicialmente se utilizarán estas componentes para el entrenamiento, para así llevar a cabo el proceso de entrenamiento, sin embargo, será inevitable ajustar ciertos parámetros y añadir mejoras para afinar el funcionamiento del robot. Como ya se ha mencionado anteriormente, el algoritmo de entrenamiento utilizado será PPO al ser uno de los más extendidos en este paradigma y los buenos resultados obtenidos con los entornos anteriores. El modelo entrenado dependerá de forma directa del entorno y de como ha sido elaborado, por lo que al probar el mejor modelo obtenido se obtendrá una prueba fiable de como de correcta ha sido la implementación de la recompensa, el espacio de observaciones entre otros aspectos. Al tratarse un entorno mucho más complejo, tanto a nivel de físicas y simulación como de cálculo de acciones, observaciones y recompensas, requerirá de un entrenamiento mucho más extenso en cuanto a nivel de número de iteraciones y episodios. De la misma forma, el entrenamiento será más costoso y por tanto implicará un mayor tiempo de entrenamiento para un mismo número de episodios con respecto los anteriores entornos.

Para la primera toma de contacto, se ha entrenado el algoritmo de PPO entrono a los 6

millones de iteraciones obteniendo como resultado los siguientes datos:

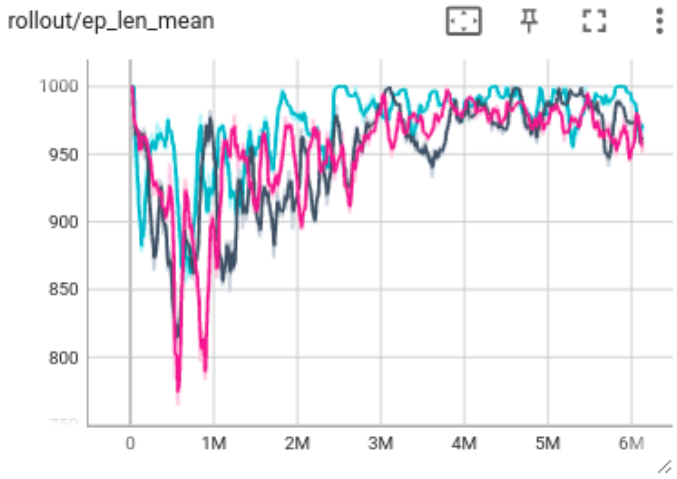


Figura 5.7: Longitud por episodio media de MuSHR con PPO

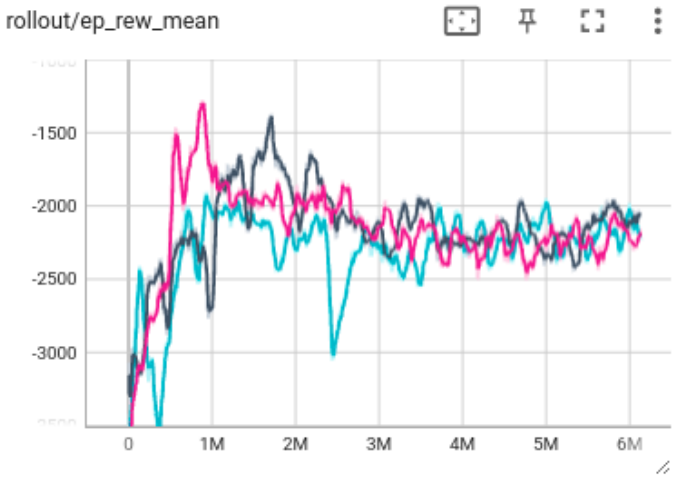


Figura 5.8: Recompensa por episodio media de MuSHR con PPO

En cuanto al entrenamiento podemos ver como a partir de 3 millones de iteraciones, el comportamiento se ”estabiliza” dentro de unos rangos concretos, tanto en tiempo medio de episodio como de recompensa media obtenida acumulada por episodio. Respecto del estado inicial, donde la toma de decisiones es aleatoria, se obtiene una recompensa mayor de forma notable, por lo que de alguna manera el agente sí que está aprendiendo y mejorando, obte-

niendo recompensas menos negativas. Esto significa que el progreso es mayor, y que se acerca de alguna manera al objetivo buscado en este trabajo. Es cierto que la recompensa mejora, y se maximiza, sin embargo no es una prámetro del todo fiable, ya que se puede modificar y dar falsas expectativas del funcionamiento del entorno, pues bastaría con aumentar la magnitud y el orden de alguna de las recompensas para ver como obtenemos una mayor recompensa. El parámetro que nos dará más fiabilidad acerca del funcionamiento del agente será la longitud del episodio. Esto se debe a que existe una limitación de tiempo de la simulación de cada episodio configurable, que ha sido añadida al resgistro del mismo, inicialmente definida en 1000 pasos. Al llegar a ese número de pasos, el entorno se resetea, motivo por el cual se ve como la gŕafica tiene un límite máximo marcado en ese rango. El agente, alcanza el mínimo de tiempo para una recompensa máxima, cosa la cual tiene sentido, aunque no tiene porque siempre cumplirse. En este caso cuando el agente, reduce el tiempo de episodio a menos de 800 iteraciones, está alcanzando el objetivo pues el episodio no se trunca por ese límite máximo, en cambio, es una solución subóptima pues aprende, reduce el tiempo, pero no todo lo que debería ni de forma estable.

Si nos ceñimos a la simulación probando el entorno con uno de los mejores modelos, podemos observar como el robot hace por acercarse al objetivo, pero a pesar de ello, no consigue alcanzarlo en todos los episodios. La mejora y el entrenamiento es notable, sin embargo se requerirá de nuevas metodologías para medir el éxito del robot en las pruebas. Esto toma gran importancia, ya que el siguiente paso consistirá en ajustar y modificar las recompensas ya existentes a favor de obtener un mejor comportamiento. Al hacerlo, como ya se ha mencionado, la magnitud de la recompensa obtenida puede diferir en gran manera en la implementación de mejoras y normalizaciones respecto del entorno presentado inicialmente.

5.3.1 Proceso de experimentación, mejora y nuevas implementaciones

Tras haber hecho una primera aproximación del entorno a replicar, relizando las definiciones correspondientes en los diferentes ámbitos que lo conforman, como la recompensa, el espacio de observaciones y el espacio de acciones, se ha continuado con la mejora e integración de nuevas metodologías que aseguren un comportamiento del agente óptimo a la hora de elaborar una trayectoria al punto objetivo. A continuación se detalla el proceso de experimentación

seguido y las mejoras en cuestión.

5.3.1.1 Inclusión de una semilla fija para la simulación

En el contexto del aprendizaje por refuerzo, la aleatoriedad juega un papel fundamental. Gran parte de los elementos que intervienen en el entrenamiento, como puede ser la inicialización del entorno o la propia red entrenada, están sujetos a procesos estocásticos. Esta aleatoriedad es deseable hasta cierto punto, ya que permite generalización ante nuevos entornos, pero presenta un problema clave como lo es la falta de reproducibilidad. La generalización ante nuevas posiciones del objetivo se cubre, precisamente, con la aparición de este objetivo de forma pseudoaleatoria durante gran número de episodios.

Para abordar esta cuestión, se ha incorporado una semilla fija (*seed*) en el entenamiento, tanto para la generación de la posición del objetivo en cada episodio, como para la confección de la propia red. Al establecer una semilla específica, se fuerza a que los generadores de números aleatorios, produzcan secuencias idénticas en cada ejecución, de tal manera que al ejecutar de nuevo el entrenamiento del entorno, el comportamiento por parte del agente será el mismo en caso de no haber cambiado ningún parámetro del entorno. De esta manera se asegura que los cambios en el entorno, como pueden ser la confección de la recompensa o el espacio de observaciones afectar de forma directa a este, puediendo repetir los entrenamientos y compararlos bajo exactamente las mismas condiciones iniciales, para analizar el impacto de una modificación concreta.

En definitiva haciendo esto se garantiza que dos versiones distintas del entorno o de la política del agente se evalúan frente a las mismas condiciones iniciales, lo cual reduce el sesgo y permite conclusiones más fiables. En su integración con el entorno, la semilla se define en un valor específico y constante, y se establece de forma explícita tanto en el entorno como en los módulos auxiliares como Numpy, torch o la propia creación del entorno.

5.3.1.2 Transición a gymnaisum

Durante este periodo de experimentación también se ha llevado a cabo una migración del entorno desde la librería gym de OpenAI, inicialmente utilizada en los entornos de prueba y los primeros entrenamientos del entorno propio, a su sucesora gymnasium, desarrollada y

mantenida activamente en la actualidad. Esta transición no solo obedece a una cuestión de mantenimiento, sino que también incorpora mejoras sustanciales en términos de compatibilidad, estabilidad y flexibilidad.

Aunque gym ha sido durante años el estándar de para el desarrollo de entornos de aprendizaje por refuerzo, su desarrollo activo finalizó en 2022. Desde entonces gymnasium ha asumido el relevo con el objetivo de dar una mejor compatibilidad con versiones más modernas de Python, MuJoCo o SB3. Aunque el funcionamiento el entorno era correcto con gym la migración asegura una reducción de posibles fallos y *warnings* en la ejecución del código, siguiendo la línea de prácticas más actuales.

Esta transición ha permitido mantener el entorno actualizado con las herramientas más recientes de DRL, pero también a provechar funciones avanzadas como la integración nativa para creacción de entornos vectorizados (VecEnv) y normalizados (VecNormalize), así como simplificar la interoperabilidad entre los módulos de entrenamiento o prueba del entorno con una integración limpia y estable con gymnasium.

En lo que a la migración respecta, esta se ha realizado de manera relativamente sencilla, teniendo que cambiar la librería importada para hacer uso de gymnaisum. A su vez se han trasladado todos los archivos de programación, como el propio entorno, el robot modelado o archivos de entrenamiento a la instalación del módulo de gymnaisum. A mayores, al igual que como se hizo con gym, el entorno se ha vuelto a registrar en los archivos de inicialización para que así gymnasium lo puediera interpretar.

5.3.1.3 Elaboración de un entorno de pruebas

Con el objetivo de evaluar de forma precisa el comportamiento del agente y ajustar adecuadamente la función de recompensa, se ha desarrollado un script auxiliar denominado *play.py*. Este entorno interactivo ha sido diseñado específicamente para ejecutar la política del agente en tiempo real mediante control manual con teclado, permitiendo observar la evolución de la recompensa obtenida en cada paso en diferentes situaciones del agente en el entorno,y su acumulado durante cada episodio.

A través de esta herramienta se ha podido comprobar si los términos que componen la función de recompensa están dentro del mismo orden de magnitud, un aspecto esencial para que

ninguno de ellos condicione en exceso la política aprendida. De no realizar esta comprobación, es habitual que los algoritmos de aprendizaje por refuerzo prioricen aquellas señales de recompensa con valores absolutos más grandes, incluso si no están directamente relacionadas con el objetivo real del comportamiento. Gracias a esto se ha podido comprobar si los términos que componen la función de recompensa están dentro del mismo orden de magnitud, un aspecto esencial para que ninguno de ellos predomine frente al resto. Cada cierto número de pasos (en este caso, cada 2 segundos) se imprimen por la terminal las componentes individuales que conforman la recompensa en ese step concreto, junto con la suma acumulada de toda la simulación.

Para su integración se empleó la librería *pygame*, la cual permite capturar las flechas del teclado al ser pulsadas. Cada vez que una de las teclas se pulsa se incrementa el valor de velocidad o giro del robot en función de si se pulsan las teclas verticales u horizontales respectivamente. Este valor acumulado se limita, acorde a los límites del espacio de acciones que se estableción en el entorno para así poder obtener datos reales. En cada iteración del código se ejecuta un paso en el entorno con los valores del espacio de acciones generado por las teclas, obteniendo un control totalmente manual del robot.

5.3.1.4 Definición y justificación de observaciones

Como ya se vió en el apartado (2.2.2.3), en el paradigma del aprendizaje por refuerzo, el espacio de observaciones representa toda la información que el agente percibe del entorno en cada instante de tiempo. Esta es la base sobre la cual el agente toma decisiones, por lo que un diseño adecuado y óptimo del espacio de observaciones es crucial. Este debe contener información suficiente y relevante para que el agente pueda aprender una política efectiva, pero también debe evitar la inclusión de datos redundantes o irrelevantes que puedan entorpecer el proceso de aprendizaje o ralentizar la convergencia.

El espacio o vector de observación tiene dimensión 10 y se ha confeccionado para que contenga la información mínima pero suficiente para que el agente pueda tomar decisiones eficaces. A continuación se detallan los componentes de este vector:

- **Error en posición**: se compone como la diferencia de posición del objetivo respecto del robot, en coordenada XY. Se trata de un vector relativo que le permite al agente

conocer en qué dirección se encuentra el objetivo, independientemente de su posición absoluta. Esta información le permiete comprender que al reducir estos valores obtendrá una mejor recompensa de manera directa.

$$P_{\text{error}} = [x_{\text{robot}} - x_{\text{target}}, y_{\text{robot}} - y_{\text{target}}] \tag{5.21}$$

- **Error en orientación**: Este valor se define como la diferencia entre el ángulo deseado, obtenido con la tangente aplicada sobre el vector de error en posición, y la orientación actual del robot. Para evitar discontinuidades (saltos de 2π), este valor se normaliza en el rango $[-\pi, \pi]$. Este es un factor fundamental en un entorno con un agente con cinemática no holonómica, pues debe conocer como está orientado respecto del objetivo para también poder reducir este error.

$$\theta_{\text{error}} = (\theta_{\text{target}} - \theta_{\text{robot}}) \tag{5.22}$$

- **Posición absoluta del robot**: incluye las coordenadas XY del robot dentro del entorno global. Aunque parte de esta información ya está contenida en el vector relativo, se ha incluido de forma explícita para facilitar que el agente aprenda patrones espaciales globales durante la fase de entrenamiento. Este error permite cuantificar de forma precisa cuánto debe girar el robot para estar orientado hacia el objetivo. Su valor es esencial tanto para el proceso de aprendizaje como para la definición de una nueva componente añadida a la recompensa.

$$P_{\text{robot}} = [x, y] \tag{5.23}$$

- **Orientación absoluta**: La orientación absoluta del robot sobre el plano horizontal, también conocido como yaw, se obtiene a partir de la información de orientación codificada como cuaternión en el vector de posiciones de la simulación. Esta representación se transforma a un ángulo en radianes, extrayendo únicamente la componente corres-

pondiente a la rotación alrededor del eje Z, acotado dentro del intervalo $[-\pi, \pi]$.

$$\text{yaw} = \theta_{\text{robot}} \in [-\pi, \pi] \tag{5.24}$$

- **Posición absoluta del objetivo**: Coordenadas XY del punto objetivo en el entorno. Esta información, junto con la posición del robot, permite al agente reconstruir tanto el vector relativo como el contexto espacial global.

$$P_{\text{target}} = [x, y] \tag{5.25}$$

- **Acción previa**: Se incluyen los valores de la acción aplicada en el step anterior, concretamente el ángulo de dirección (steering) y la velocidad (speed). Esta información dota al agente de cierta memoria de corto plazo, permitiéndole relacionar la formulación de la recompensa por control basada en la diferencia entre acciones consecutivas que se explicará más adelante, la cual fue modificada.

$$A_{\text{previa}} = [steering_{\text{t-1}}, speed_{\text{t-1}}] \tag{5.26}$$

La estructura final del vector proporciona al agente una representación balanceada del estado actual del entorno, incluyendo tanto información local (error de orientación) como global (osición del objetivo) y dinámica (acción pasada). El diseño final del espacio de observaciones que utiliza el robot se compone de la siguiente forma:

$$Observacin = [P_{\text{error}}, \theta_{\text{error}}, P_{\text{robot}}, \text{yaw}, P_{\text{target}}, A_{\text{previa}}] \tag{5.27}$$

5.3.1.5 Redefinición de la recompensa final

Con el objetivo de obtener un comportamiento más estable, preciso y eficaz por parte del agente, se han rediseñado varios componentes de la función de recompensa original. Estos cambios buscan mejorar la alineación entre las señales de aprendizaje y los objetivos reales de navegación, eliminando redundancias, ajustando escalas y añadiendo términos, que en definitiva, otorguen un mejor feedback al agente para aprender una buena política. A continuación

se describen en detalle las recompensas actualmente implementadas en el entorno MuSHR:

- **Penalización por distancia al objetivo:** se rediseñó esta componente para evitar una penalización lineal simple, que resultaba poco representativa cuando el agente se encuentra cerca del objetivo. En su lugar, se implementó una función de recompensa de tipo exponencial, que otorga una penalización más fuerte cuando el agente está lejos y refuerza más finamente los acercamientos a corta distancia, penalizando en una considerable menor medida. Además, se añadió un coeficiente que permite controlar y ajustar la pendiente de la función exponencial, facilitando como se quiere que sea ese ajuste.

$$R_{\text{distancia}} = w_{dist} \cdot e^{-k \cdot d} \tag{5.28}$$

  donde d es la distancia al objetivo, w_{dist} es el peso asignado, y k es el coeficiente de inclinación de la función exponencial.

- **Recompensa por heading (orientación hacia el objetivo):** este nuevo término se basa en el error angular entre la orientación del robot y la dirección al objetivo. Esta componente tiene como objetivo alinear progresivamente al agente con su rumbo deseado, algo especialmente importante en plataformas no holonómico como el robot utilizado para la simulación. El error angular se normaliza en el rango $[-\pi, \pi]$, y la recompensa varía entre $-w_{heading}$ (cuando el agente está completamente desalineado) y $+w_{heading}$ (cuando apunta directamente al objetivo y el error es próximo a 0).

$$R_{\text{heading}} = w_{heading} \cdot \left(1 - 2 \cdot \frac{|\theta_{\text{error}}|}{\pi}\right) \tag{5.29}$$

- **Penalización por control:** se modificó sustancialmente este término. En versiones anteriores se penalizaba toda acción en función de su magnitud, lo que impedía la exploración eficaz del espacio de acciones, y sumaba términos con unidades heterogéneas (velocidad lineal y angular). La nueva formulación penaliza exclusivamente la diferencia entre la acción actual y la acción del paso anterior, fomentando trayectorias más suaves

sin castigar el uso de acciones grandes cuando son necesarias.

$$R_{\text{control}} = -w_{ctrl} \cdot (\Delta a_{\text{steering}})^2 \tag{5.30}$$

donde $\Delta a_{\text{steering}}$ es la diferencia en el valor de giro (steering) entre dos pasos consecutivos. La velocidad ya no se penaliza directamente.

- **Penalización por tiempo:** se mantiene una penalización constante por cada paso del episodio, incentivando que el agente alcance el objetivo de forma eficiente. A su vez previene comportamientos pasivos o bucles de comportamiento estático como girar indefinidamente sin avanzar. De esta manera también se intenta evitar máximos y mínimos locales, promoviendo la exploración.

$$R_{\text{tiempo}} = -w_{time} \tag{5.31}$$

- **Recompensa por alcanzar el objetivo:** esta componente se mantiene, pues si el agente se aproxima a una distancia inferior a un umbral (actualizado a 0.25), se considera que el objetivo ha sido alcanzado. En ese caso, se asigna una recompensa elevada que marca el éxito del episodio, y se establece el estado del episodio como terminado para su posterior reseteo y continuar así con el entrenamiento.

$$distancia < 0.25 => R_{total} += 2000; \quad terminated = true \tag{5.32}$$

- **Recompensa total:** la recompensa final se calcula como la suma ponderada de todos los términos anteriormente definidos:

$$R_{\text{total}} = R_{\text{distancia}} + R_{\text{heading}} + R_{\text{control}} + R_{\text{tiempo}} + \delta_{\text{éxito}} \cdot R_{\text{objetivo}} \tag{5.33}$$

donde $\delta_{\text{éxito}}$ vale 1 si el objetivo ha sido alcanzado, y 0 en caso contrario.

Estos cambios, fruto de un proceso iterativo de análisis y ajuste, han permitido obtener un comportamiento mucho más coherente con los objetivos planteados, reduciendo oscilacio-

nes, mejorando la estabilidad durante la navegación y facilitando la convergencia del agente durante el entrenamiento.

6 Resultados y conclusiones

6.1 Resultados

Durante el desarrollo del trabajo se han llevado a cabo múltiples entrenamientos del agente en el entorno simulado, variando distintos parámetros de la función de recompensa con el fin de encontrar una configuración óptima. Estas son algunas de las diferentes combinaciones que se han realizado tras la redefinición del entorno:

Código	**weight_dist**	**(coef)**	**weight_ctrl**	**weight_time**	**weight_heading**	**reward_goal**
190425_0	3	0.7	100	0.5	1	100
190425_1	4	0.6	100	0.5	1	100
190425_2	5	0.5	100	0.5	1	100
190425_3	2	0.7	100	0.5	1	100
200425_0	1	0.5	1	0.5	1	100
200425_1	2	0.5	1	0.5	1	100
200425_2	3	0.5	1	0.5	1	100
200425_3	4	0.5	1	0.5	1	100
200425_4	5	0.5	1	0.5	1	2000
200425_5	6	0.5	1	0.5	1	2000
210425_0	5	0.4	1	0.5	2	2000
210425_1	5	0.3	1	0.5	2	2000
220425_0	5	0.2	1	0.5	2	2000
220425_1	5	0.2	1	0.5	3	2000

Cuadro 6.1: Historial de configuraciones de parámetros y recompensas utilizadas durante el entrenamiento del agente.

En un primer momento se llevaron a cabo 4 entrenamientos simultaneos (código empezando por 19), con ligeras modificaciones en el peso de la recompensa de distancia, así como en el coeficiente que cambia la pendiente de la función exponencial. Los resultados obtenidos fueron los siguientes:

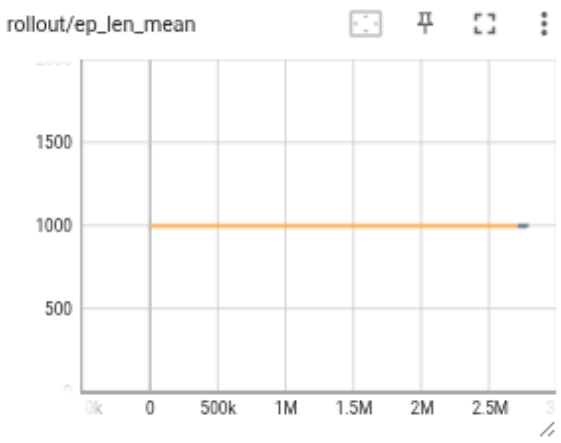


Figura 6.1: Longitud media por episodio de MuSHR con redefinicion de recompensa (19)

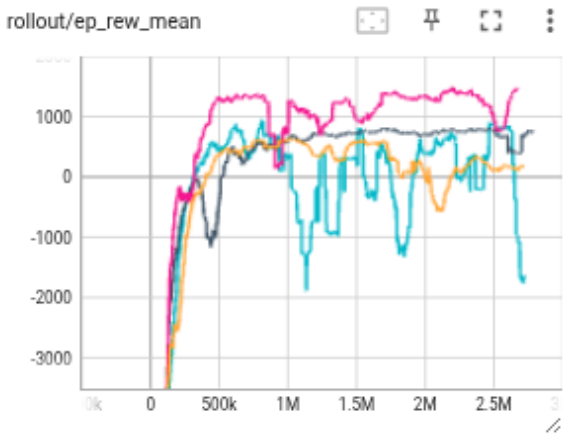


Figura 6.2: Recompensa media por episodio de MuSHR con redefinicion de recompensa (19)

Como se puede ver en la primera de las gráficas, la longitud de los episodios no disminuye en nigún momento, lo que significa que el robot no ha llegado al objetivo en nigún episodio del entrenamiento. Esto es un claro indicativo de que alguno, o varios de los parámetros no están bien configurados. La recompensa obtenida en cambio, si presenta un aumento progresivo, e incluso cierta estabilidad en alguno de los entrenamientos, sin embargo, esto no es un indicativo de éxito ya que si el robot consigue maximizar la recompensa pero no alcanzar el objetivo, se deberán aplicar nuevos cambios puesto que se encuentra relativamente lejos de la política buscada. Gracias al entorno creado para manejar el robot manualmente, se vió como la recompensa de control no estaba balanceada con el resto de componentes, lo

que provocaba que el robot las omitiese, y encima no se moviera prácticamente, ya que le recompensaba mucho el hacer movimientos muy suaves. Tras extraer estas conclusiones, se redujo drasticamente el peso de la recompensa relacionada con el control, y junto con ello se aumentó el margen de tiempo, con el objetivo de que el robot tuviera más tiempo dentro de un mismo episodio para poder aprender, pasando de 1000 pasos por episodio iniciales, a 2000 pasos. Además, se mantuvo el coeficiente de la pendiente de distancia en 0.5 de cara a futuros aprendizajes, al ser el entrenamiento donde se obtuvo una mayor recompensa:

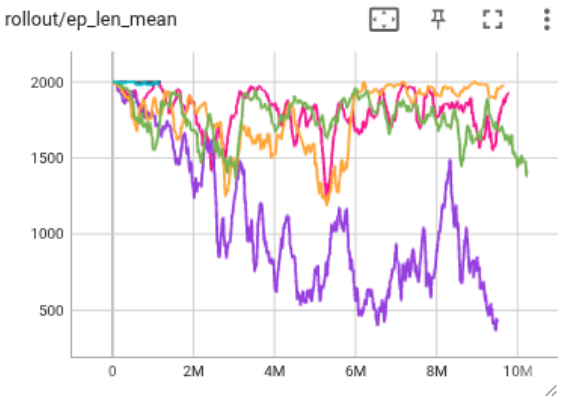


Figura 6.3: Longitud media por episodio de MuSHR con redefinicion de recompensa (20)

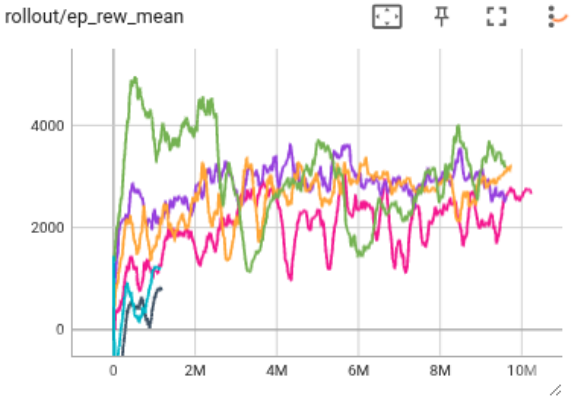


Figura 6.4: Recompensa media por episodio de MuSHR con redefinicion de recompensa (20)

Para los dos primeros entrenamientos de esta serie, se cortó rapidamente su ejecución ya que el robot seguía sin alcanzar el objetivo, pues el timpo por episodio estaba estancado en el límite. Adoptando un comportamiento más ”agresivo” se siguió aumentando el peso para la

recompensa de distancia. De esta manera si que se obtuvieron resultados algo mejores, pues además de maximizar la recompensa, el robot comenzó a a minimizar el tiempo medio por episodio, llegando hasta los 1200 pasos aproximadamente, de los 2000 marcados como límite.

Dado que el robot parecía mejorar más aún su funcionamiento aumentando las recompensas vinculadas a la distancia, se continuó con esta metodología, y se le dió un mayor valor a la recompensa terminal, pasando de 100 a 2000, de acuerdo a los valores que se estaban teniendo de media por episodio, para que así realmente tuviera mucho peso alcanzar el objetivo, y que no fuera pasado por alto a la hora del aprendizaje. Este cambio trajo consigo una gran mejora con el entrenamiento "200425-4" (gráfico morado), ya que los episodios se redujeron hasta los 500 pasos de media en varios puntos del entrenamiento, y manteniendose por debajo de los 1000 pasos en gran parte de estos, lo que aseguraba que ahora el robot sí estaba alcanzando el objetivo de manera consistente. Si se grafican las trayectorias seguidas por el robot con el mejor modelo de este entrenamiento cargado, se observa como los caminos seguidos por este oscilan de tal manera que el robot tarda más en alcanzar el punto de lo deseable. También se puede ver como para el punto objetivo concreto en el episodio 4 el robot no es capaz de alcanzar el punto, quedandose en un bucle infinito dando vueltas alrededor de este.

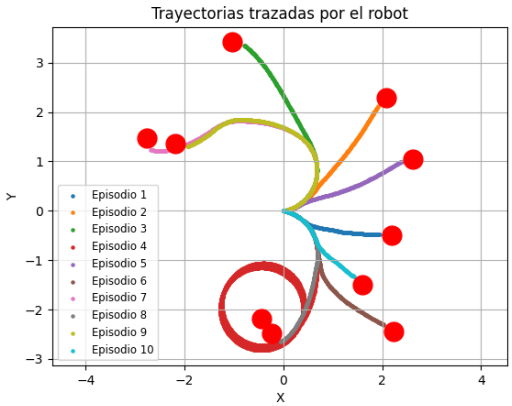


Figura 6.5: Trayectorias realizadas por el modelo "200425-4" en el step 9.420.000

Con este trabajo no solo se buscaba alcanzar el punto de destino, si no que lo hiciera de manera óptima, requisitos los cuales no se estaba cumpliendo, pues controlando de forma manual el robot, se podían obtener tiempo inferiores a estos. Lo más óptimo para llegar de un punto a otro (si no hay obstáculos de por medio) es ir en línea recta, se buscó lograr una

política del agente que cumpliera que al inicio del episodio girará lo necesario para ponerse totalmente orientado hacia el objetivo, y una vez apuntara hacia este, que avanzara lo más rápido posible. Para conseguir este comportamiento se aumentó el peso del *heading* a la vez que se redujo el coeficiente de la función exponencial de laa recompnesa de distancia, reduciendo su pendiente, y provocando que al principio no variase en exceso, es decir, que no se tuviera tanto en consideración la distancia como si la orientación. El agente comprendió que a una distancia más lejana la recompensa por la lejanía seguía siendo muy negativa aunque avanzara, dandole prioridad a la orientacion respecto del objetivo. Reducir el coeficiente de la exponencial además contribuye a tener un mayor ajuste, incentivando que una vez este orientado y avance, continue hasta lograr alcanzar el objetivo de una manera más exacta. A continuación se muestran los resultados obtenidos en la siguiente serie de entrenamientos:

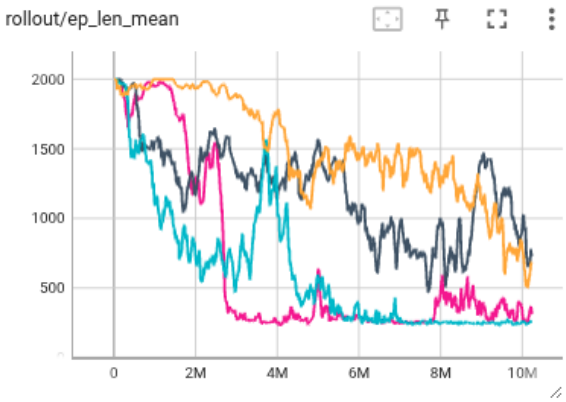


Figura 6.6: Longitud media por episodio de MuSHR con redefinicion de recompensa (21)

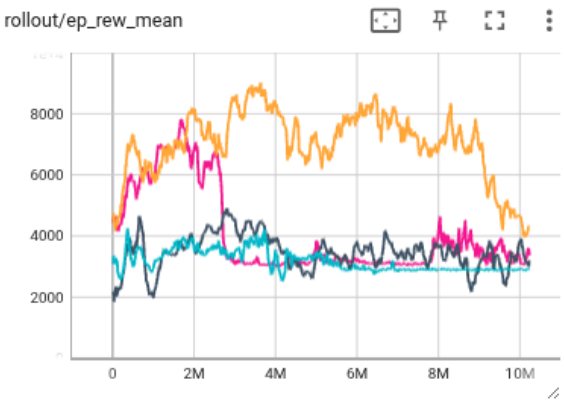


Figura 6.7: Recompensa media por episodio de MuSHR con redefinicion de recompensa (21)

En dos de los entrenamiento no se consiguieron mejores resultados que los obtenidos hasta el momento, pues la longitud media de los episodios seguía siendo alta. Sin embargo, se puede observar como en dos de ellos sí que se obtuvo un resultado muy satisfactorio, ya que los episodios comenzaron a tener una longitud por debajo de los 300 pasos prolongadamente y de manera muy estable. Especialmente, el mejor resultado se consiguió para el entrenamiento ”210425-1” (azul celeste), que a pesar de alcanzar esa cifra tan reducida de duración por episodio más tarde que el otro entrenemiento, pudo mantener de forma muy estable ese valor a lo largo de millones de pasos.

Cargando este modelo obtenido en la simulación, se aprecia de forma visual como el robot alcanza el objetvo en todas las ocasiones, además de hacerlo de manera óptima demostrando que ha aprendido la política que se buscaba. A continuación se muestran las trayectorias seguidas por el robot para 10 episodios consecutivos:

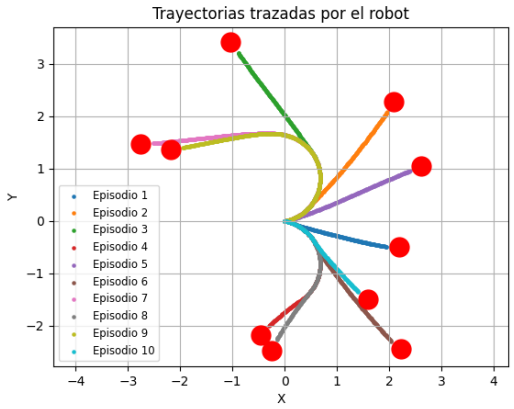


Figura 6.8: Trayectorias realizadas por el modelo "210425-1" en el step 9.450.000

6.2 Conclusiones

A lo largo del presente trabajo se ha desarrollado un entorno de simulación personalizado basado en el modelo del robot móvil MuSHR, integrando herramientas como MuJoCo para la simulación, Gymnasium para la confección del entorno de aprendizaje por refuerzo y el algoritmo PPO de Stable-Baselines3 para el entrenamiento del agente. Tras un proceso iterativo de diseño, pruebas y mejoras, se ha logrado que el agente aprenda a navegar hacia un objetivo fijo, planificando una ruta óptima y mejorando progresivamente su comportamiento mediante este entrenamiento por refuerzo.

Los entrenamientos iniciales en entornos clásicos como Lunar Lander y Reacher permitieron validar el correcto funcionamiento del pipeline de aprendizaje y comparar el rendimiento entre dos de los algoritmos más utilizados, A2C y PPO, donde este segundo demostró mayor estabilidad y eficacia frente al otro, motivo por el cual fue seleccionado para el entorno final del robot móvil.

En el anterior apartado se ha visto como en el entorno personalizado, el agente fue capaz de aprender una política que tiende a acercarse al objetivo, reduciendo el tiempo medio por episodio significativamente y aumentando la recompensa acumulada, consiguiendo llegar al objetivo en todos los episodios. El modelo entrenado mostró un comportamiento coherente con los objetivos planteados, especialmente tras la redefinición de la función de recompensa y del espacio de observaciones.

Este trabajo ha demostrado que es posible aplicar con éxito técnicas de aprendizaje por refuerzo profundo a la navegación de un robot móvil en un entorno simulado. Se ha desarrollado un entorno completo desde cero, incluyendo modelado físico, definición del espacio de observaciones, espacio de acciones y función de recompensa personalizada. Además, se ha experimentado con distintos enfoques de evaluación y visualización del comportamiento aprendido tal y como se marcaba en los objetivos.

El código fuente del entorno y la simulación se encuentra disponible en el siguiente repositorio de github:

`https://github.com/izanvines/Deep-RL-for-mobile-robots-gymnasium`

6.3 Líneas de trabajo futuro

Se ha podido demostrar la viabilidad del uso de técnicas de aprendizaje por refuerzo profundo en el control de un robot móvil en un entorno simulado, cumpliendo con los objetivos preestablecidos. Sin embargo, la naturaleza del entorno desarrollado y la versatilidad del aprendizaje por refuerzo abren múltiples líneas de investigación y mejora que podrían abordarse en futuros trabajos:

- **Transferencia a untorno real:** El punto más interesante generalmente, posterior a una simuación de un robot, es el de transferir la política entrenada en simulación al robot físico concreto, o a uno con cinemática similar. Este proceso, presenta retos significativos debido a la diferencia entre el comportamiento ideal que se presenta en la simulación y la complejidad del entorno real por factores como el ruido sensorial, una dinámica imprecisa, o errores de modelado entre otros.

  Para llevar a cabo esta integración, sería interesante de igual manera hacer ajustes de la política preentrenada en simulación mediante un entrenamiento adicional en el robot real. Además sería casi imprescindible introducir nuevas variables a la simulación, para entrenar una política más robusta a las discrepancias del mundo real, intentando

conseguir que la simulación se asemeje lo máximo posible a la realidad teneindo en cuenta todos los inconvenientes que afectan al funcionamiento.

La validación en un entorno real, gracias a la posible disponibilidad de una plataforma con características similares, permitiría evaluar no solo la eficacia de la política, sino también su robustez, su capacidad de generalización y la seguridad para su posterior integración y aplicación en un entorno donde desempeñar una tarea con una aplicación real, como pudiera ser el ámbito de la robótica espacial o la robótica industrial.

- **Ampliación del entorno:** Actualmente, el entorno simulado presenta un escenario estático y sin obstáculos, lo cual simplifica el aprendizaje pero limita su aplicación a casos reales. Una mejora sustancial sería la incorporación de obstáculos tanto estáticos, como móviles si fuera necesario. El robot debería evitar colisionar con estos elementos del entorno, permitiendo evaluar la capacidad del agente para planificar rutas alternativas, inclsuo en tiempo real.

  También seria interesante plantear entornos más complejos, con zonas estrechas, giros cerrados o bifurcaciones, que obliguen al agente a tomar decisiones más estratégicas. Estas modificaciones requerirían una redefinición de las observaciones y posiblemente de una extensión del espacio de acciones para contemplar situaciones de planificación local para poder evitar dichos obstáculos.

- **Integración de sensores y entorno parcialmente observable:** En la simulación se asume un conocimiento completo del estado, pues el espacio de observaciones se compone de la propia posición del robot, entre otros parámetros, distando de unas condiciones reales. Poder incorporar sensores simulados, como LIDARs para detectar obstáculos en su entorno inmediato sin conocer su posición exacta, cámaras o IMUs para ayudar a estimar la orientación, velocidad y posición del robot, conducirían a trabajar bajo un paradigma de entorno parcialmente observable,que haría que el agente dependa de percepciones ruidosas y parciales, exigiendo un aprendizaje más complejo, pero más cercano a aplicaciones reales, complementando así los puntos anteriores.

- **Hibridación con ténicas clásicas de navegación:** A pesar de haber conseguido resultados satisfactorios, no deja de ser una línea más de investigación que a priori

puede presentar también grandes retos, por lo que combinar lo mejor del enfoque clásico de navegación con las ventajas del aprendizaje por refuerzo podría ser otro punto sobre el que trabajar. Algunas posibles estrategias serían las de combinar por ejemplo, una planificación global con métodos de RL junto con una planificación local clásica para evitar obstáculos, o bien al contrario, donde se usen algoritmos globales de búsqueda de grafos como las que ya se han explicado, junto con técnicas que apliquen una política para evitar obstáculos dinámicos, y que permitan redirijirse por la ruta ya planificada. Otra opción sería la de combinar el clásico SLAM con RL, pues como ya hemos visto, el espacio de observaciones utilizado por el entorno, contiene la posición del robot, datos los cuales pueden ser adquiridos de utilizar SLAM con su respectiva sensórica, y que aporte esa infromación que sería necesaria. A su vez, mediante RL dotaríamos a SLAM de ese módulo de navegación que necesita para planificar posibles rutas.

Bibliografía

Abeliuk, A., y Gutiérrez, C. (2021). Historia y evoluación de la inteligencia artificial. *Revista Bits de Ciencia*(21), 14–21.

Andrychowicz, M., Wolski, F., Ray, A., Schneider, J., Fong, R., Welinder, P., … Zaremba, W. (2017). Hindsight experience replay. *Advances in neural information processing systems*, *30*.

Borenstein, J., Koren, Y., y cols. (1991). The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE transactions on robotics and automation*, *7*(3), 278–288.

Brockman, G. (2016). Openai gym. *arXiv preprint arXiv:1606.01540*.

Feinberg, V., Wan, A., Stoica, I., Jordan, M. I., Gonzalez, J. E., y Levine, S. (2018). Model-based value estimation for efficient model-free reinforcement learning. *arXiv preprint arXiv:1803.00101*.

Fox, D., Burgard, W., y Thrun, S. (1997). The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, *4*(1), 23–33.

Fujimoto, S., van Hoof, H., y Meger, D. (2018). *Addressing function approximation error in actor-critic methods.*

GALIPIENSO, A., ISABEL, M., CAZORLA QUEVEDO, M. A., Colomina Pardo, O., Escolano Ruiz, F., y LOZANO ORTEGA, M. A. (2003). *Inteligencia artificial: modelos, técnicas y áreas de aplicación*. Ediciones Paraninfo, SA.

Haarnoja, T., Zhou, A., Abbeel, P., y Levine, S. (2018). *Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.*

Lillicrap, T. (2015). Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*.

Mnih, V., Badia, A. P., Mirza, M., Graves, A., Lillicrap, T. P., Harley, T., … Kavukcuoglu, K. (2016). *Asynchronous methods for deep reinforcement learning.*

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., y Riedmiller, M. (2013). *Playing atari with deep reinforcement learning.*

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., … others (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529–533.

Pontryagin, L. S. (2018). *Mathematical theory of optimal processes.* Routledge.

Qin, S. J., y Badgwell, T. A. (1997). An overview of industrial model predictive control technology. En *Aiche symposium series* (Vol. 93, pp. 232–256).

Racanière, S., Weber, T., Reichert, D., Buesing, L., Guez, A., Jimenez Rezende, D., … others (2017). Imagination-augmented agents for deep reinforcement learning. *Advances in neural information processing systems*, *30*.

Raffin, A., Hill, A., Gleave, A., Kanervisto, A., Ernestus, M., y Dormann, N. (2021). Stable-baselines3: Reliable reinforcement learning implementations. *Journal of Machine Learning Research*, *22*(268), 1–8.

Schmittle, M. (2020). *Mujoco ros simulator for mushr.* `https://github.com/prl-mushr/mushr_mujoco_ros`.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., y Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., … others (2018). A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science*, *362*(6419), 1140–1144.

Srinivasa, S. S., Lancaster, P., Michalove, J., Schmittle, M., Summers, C., Rockett, M., … Sadeghi, F. (2019). MuSHR: A low-cost, open-source robotic racecar for education and research. *CoRR*, *abs/1908.08031*.

Sutton, R. S., y Barto, A. G. (1998). *Reinforcement learning: An introduction* (Revised Edition (2018) ed.). MIT Press.

Todorov, E., Erez, T., y Tassa, Y. (2012). Mujoco: A physics engine for model-based control. En *2012 ieee/rsj international conference on intelligent robots and systems* (pp. 5026–5033).

Turing, A. M. (1950). Computing machinery and intelligence. *Mind*, *59*(236), 433–460. doi: 10.1093/mind/LIX.236.433

Zhu, K., y Zhang, T. (2021). Deep reinforcement learning based mobile robot navigation: A review. *Tsinghua Science and Technology*, *26*(5), 674–691.