Universidad de Alicante
Ingeniería Robótica

APRENDIZAJE POR REFUERZO EN ROBÓTICA MÓVIL

ROBOTS MÓVILES

Javier Bascuñana Giner
Blanca Martínez Gálvez
Carlos Martínez Suárez
Paula Perales Izquierdo

1. INTRODUCCIÓN

En la última década se ha observado un aumento importante en las aplicaciones relacionadas con la robótica a nivel mundial. Estas aplicaciones ya no se encuentran únicamente en laboratorios o en fábricas, donde se pueden mantener condiciones controladas, sino que también se presentan en distintas situaciones cotidianas. Entre las distintas categorías de robots, destaca fuertemente la robótica móvil, debido a su alto potencial de impacto social.

La robótica móvil es la rama de la robótica encargada del diseño y construcción de máquinas automáticas capaces de trasladarse en cualquier ambiente dado. Están pensadas para realizar tareas autónomamente, de manera independiente a ciertas órdenes de los humanos. Pese a que es un campo bastante avanzado que ya se emplea en variedad de tareas en la industria, sigue siendo un ámbito en continua evolución e investigación, dado que va muy ligado a la optimización de los algoritmos que se utilizan para navegación, localización, etc.

En este sentido, la capacidad de resolución de problemas de un robot móvil suele estar limitada por el conocimiento y las habilidades del diseñador de sus algoritmos. Se identifica entonces la necesidad de incorporar metodologías generales que les permita adquirir las habilidades necesarias para poder realizar las labores que les son asignadas. En este punto entra en juego el aprendizaje automático (*machine learning* en inglés).

El aprendizaje automático es un subcampo de las ciencias de la computación y una rama de la inteligencia artificial, cuyo objetivo es desarrollar técnicas que permitan que las computadoras aprendan. En este campo aparece el concepto de agente inteligente, que es una entidad capaz de percibir su entorno, procesar tales percepciones y responder o actuar de manera racional, tendiendo a maximizar el resultado esperado. Este proceso de aprendizaje mejora con la experiencia y con el uso de datos, es decir, en un principio el agente no conoce nada, pero con el tiempo y ayuda de una gran variedad de algoritmos y técnicas, el agente acaba por ser un experto en el entorno en que trabaja. Dentro del campo del aprendizaje automático existe una gama muy amplia de aplicaciones: desde motores de búsqueda, clasificación de secuencias de ADN, reconocimiento de habla y lenguaje escrito hasta alcanzar la perfección en videojuegos o incluso la robótica.

Dentro de las aplicaciones mencionadas anteriormente, existe una amplia gama de algoritmos que llevan a cabo dicho aprendizaje. No obstante, cada uno de ellos lo realiza de una manera distinta, ya que como es obvio, no todos los campos tienen una manera única de resolverse. Respecto a los algoritmos principales, cabe destacar los siguientes:

- **Aprendizaje supervisado**: el algoritmo produce una función que establece una correspondencia entre las entradas y las salidas deseadas del sistema. Se utilizan conjuntos de datos de entrenamiento etiquetados para comenzar el aprendizaje con la finalidad de poder obtener dicha función de clasificación para que cuando se envíe un dato de entrada no conocido, sea capaz de designar correctamente su clase.
- **Aprendizaje no supervisado**: Es muy similar al anterior, no obstante, los datos de entrada no están clasificados y se trabaja más con las características de estos para hacer agrupaciones.
- **Transducción**: similar al aprendizaje supervisado, pero no construye de forma explícita una función, sino que trata de predecir la clase de los futuros ejemplos basándose en los ejemplos de entrada, sus clases y las entradas nuevas del sistema.
- **Aprendizaje por refuerzo**: este consiste, de manera muy breve (ya que se profundizará en él a lo largo del trabajo) en algoritmos que tratan de seleccionar las acciones que van a maximizar la recompensa que recibirá el agente inteligente en función de la tarea que realiza.

En concreto, el aprendizaje por refuerzo se basa en una señal de recompensa que le indica al sistema qué hacer. En este tipo de algoritmos el feedback que obtiene del entorno no es instantáneo, sino que está retrasado, por lo que el tiempo es un factor muy importante. Este tipo de aprendizaje es el que más se asemeja al del ser humano y otros animales. Esto se debe a que el aprendizaje por refuerzo funciona de forma que aprende qué hacer y la manera de asignar las diversas situaciones a acciones para maximizar una recompensa numérica. Por tanto, no se indica qué acciones debe tomar, sino que debe descubrir la mayor recompensa a base de probar distintas acciones (es decir, mediante prueba y error).

El problema del aprendizaje por refuerzo se formaliza gracias a los procesos de decisión de Markov. La idea general es recoger los aspectos más importantes del problema real mediante un agente que interactúa con el entorno para así llegar a un objetivo. Además, el agente debe de ser capaz de percibir el estado en el que se encuentra y ser capaz de realizar una acción que lo altere. Una de las aplicaciones más conocidas de este tipo de aprendizaje es la del control del péndulo invertido, el cual se suele aplicar a videojuegos.

El funcionamiento de este tipo de algoritmos se basa en dos elementos principales, el agente y el entorno, además de otros que pueden intervenir, como una política, una recompensa, una función de valor y, opcionalmente, un modelo del entorno.

La política define el comportamiento del agente en un momento dado. Es la relación de los estados del entorno percibidos por el agente, con las acciones a realizar por el agente en tal estado. En general, deben ser políticas estocásticas o deterministas.

La recompensa define la meta. En cada paso, el entorno manda al agente una señal numérica. El objetivo es maximizar la recompensa total por episodio, por lo que la recompensa define tanto las buenas como las malas acciones en cada ocasión.

La función de valor, a diferencia de la recompensa, determina qué es bueno a largo plazo. El valor de un estado es la recompensa total que el agente espera obtener en el futuro empezando desde ese estado.

Por último, el modelo del entorno es una representación del comportamiento del entorno hecha por el agente. Así pues, se pueden planear con antelación futuras situaciones antes de experimentarlas.

2. ALGORITMOS

2.1. Q-LEARNING

Q-Learning es un algoritmo de aprendizaje por refuerzo diseñado por Watkins en 1989, propuesto como su tesis doctoral con el nombre de "Learning from Delayed Rewards", en el que se introducía el estudio del aprendizaje del algoritmo en base a recompensas y castigos. Más tarde en 1992 se logró demostrar su convergencia hacia la optimalidad, dada una serie de restricciones.

El nombre de Q-Learning proviene de *Quality* (calidad en inglés), que representa cómo de útil es una acción dada a la hora de las futuras recompensas. Por esto, se denomina Q a la matriz, de tamaño [estados, acciones], donde tras cada acción se va actualizando la estimación que viene dada por la nueva experiencia. Esta tabla se inicializa a cero y se va actualizando siguiendo la ecuación:

$$Q[s_t, a_t] += \alpha\left(r_{t+1} + \gamma \max_a Q[s_{t+1}, a] - Q[s_t, a_t]\right);$$

Figura 1. Actualización de la tabla Q.

De esta forma, el valor de la posición de la tabla que representa el estado actual (s_t) y la acción tomada (a_t), se actualiza con la combinación especificada en la Figura x de los siguientes elementos:

- $\mathbf{r_{t+1}}$: recompensa recibida tras tomar la acción.
- $\mathbf{s_{t+1}}$: estado al que transiciona.

- γ: factor de descuento. Este valor se encarga de balancear los futuros valores de recompensa que se puedan tomar. Consiste en una especie de ajuste para la aproximación de las recompensas.
- α: tasa de aprendizaje. Es el ratio que, al igual que en otros tipos de algoritmos de aprendizaje automático, indica cómo de grandes son los pasos del aprendizaje.

Este es uno de los factores clave del algoritmo, ya que es la principal diferencia con otros como el de Montecarlo, comentado en el siguiente apartado. A diferencia de este, en Q-Learning la actualización de la "puntuación" se realiza inmediatamente después de la realización de la acción, resultando en un algoritmo de convergencia mayor pero que requiere de una buena relación entre exploración y explotación para su correcto funcionamiento, un buen ajuste de los parámetros.

Aquí también entra en juego un factor muy relevante como es el de la política. Esta suele realizarse mediante una técnica denominada ε-voraz (*épsilon-voraz*), que consiste en dos posibles decisiones a tomar en base a una probabilidad que varía según avanza el aprendizaje. Estas dos posibilidades son elegir una acción de manera aleatoria, o escoger aquella que, según nuestra experiencia previa, nos lleve a un mejor cómputo de recompensas.

Al factor de probabilidad de elección se le aplica un factor (δ), llamado factor de decaimiento, cuya función es reducir en cada ejecución la probabilidad de que la acción sea aleatoria para que, según vaya cogiendo más experiencia, elija aquellas que ya conoce en lugar de explorar nuevas opciones. Es un factor muy importante y que se debe ajustar adecuadamente para evitar tanto que el agente explore demasiadas opciones nuevas y no converja nunca el algoritmo, como que este converja demasiado rápido y se quede con un conjunto de acciones que le lleven a un camino subóptimo.

```
Q ← initialize(S, A);
while number of episodes is not reached do
    s_t = s_0;
    while episode is active do
        a_t ← π_Q(s_t);
        s_{t+1}, r_{t+1} ← perform(a_t);
        Q[s_t, a_t] += α (r_{t+1} + γ max_a Q[s_{t+1}, a] − Q[s_t, a_t]);
        s_t = s_{t+1};
    end
end
```

Figura 2. Pseudocódigo de Q-Learning.

Este algoritmo resulta que, pese a que ya existe desde hace unos años y puede considerarse “antiguo”, obtiene muy buenos resultados en aquellos problemas cuyo espacio de trabajo sea discreto. No obstante, esta se considera de hecho una de sus limitaciones, ya que la tabla Q debe tener únicamente índices discretos y, además, no resulta un algoritmo factible en problemas con un número demasiado grande de estados.

2.2. MONTECARLO

El método de MonteCarlo se refiere a un conjunto de algoritmos que utilizan el muestreo y la aleatoriedad para aproximar soluciones, es decir, genera episodios recogiendo acciones de forma pseudo-aleatoria. Una vez acabado el episodio y recibidas todas las recompensas, se utiliza la experiencia acumulada para aproximar el valor de los pares estado-acción.

MonteCarlo ofrece tres importantes ventajas sobre los métodos típicos de programación dinámica. En primer lugar, se puede emplear para estimar el mejor comportamiento directamente de la experiencia, sin ningún modelo de la dinámica del entorno. Además, se puede usar con un simulador o modelo de muestras, ya que en algunos tipos de algoritmos es complicado constituir el tipo de modelo explícito de probabilidades de transición requerido. Por último, es fácil y eficiente enfocar los métodos de MonteCarlo en un pequeño subconjunto de estados.

A pesar de las grandes ventajas que ofrece este método, también presenta diferentes desventajas como resultados poco fiables o incluso incorrectos, ya que con pocos datos representativos o no actualizados puede fallar. Además, en casos donde la relación entre variables pueda modificar el resultado final del proyecto o inversión, este método no va a ser útil ya que no tiene en cuenta la dependencia entre los datos.

Para la implementación del algoritmo de MonteCarlo se adjunta un pseudocódigo el cual se explica a continuación:

```
Q ← initialize(S, A) arbitrarily;
M ← initialize(S, A) as empty list;
while number of episodes is not reached do
    Generate s_0, a_0, r_1, s_1, a_1 ... r_T, s_T ~ π_Q;
    G ← 0;
    for t = T − 1 to 0 do
        G ← γ G + r_{t+1};
        M[s_t, a_t] ← append(G);
    end
    for s ∈ S, a ∈ A do
        Q[s, a] ← mean(M[s, a]);
    end
end
```

Figura 3. Pseudocódigo MonteCarlo.

El funcionamiento del algoritmo es muy sencillo: se deben generar el estado, la acción y la recompensa en función de una política. Esta política puede ser aleatoria, escogiendo una acción al azar, voraz, seleccionando la acción que mejor resultado otorga, o ϵ-voraz, la cual cuenta con una probabilidad que ayudará con la decisión a la hora de seleccionar una acción: se genera un número aleatorio, normalmente entre 0 y 1, y se compara con la constante ϵ que se deberá ajustar adecuadamente; si el número generado es mayor que ϵ, se escoge la opción voraz; por lo contrario, se selecciona una acción aleatoria.

Una vez conocida la acción a realizar, se debe almacenar el total de las ganancias que se han producido hasta el momento. Para calcular este valor, se debe recorrer en sentido opuesto el almacenamiento de las recompensas. Esto se debe a que la ganancia se calcula mediante la acumulación de recompensas. Además, se aplica un factor de descuento, γ, para provocar una convergencia más óptima y rápida.

$$G_t = r_t + \gamma\, r_{t+1} + \gamma^2\, r_{t+2} + \ldots = \sum_{k=0}^{\infty} \gamma^k\, r_{t+k}$$

Figura 4. Cálculo de la ganancia.

Por último, simplemente se debe acumular la ganancia en una matriz que se compone con los datos que ayudarán en la decisión de la acción en la política.

Al finalizar todos los episodios establecidos, el método presentará una aproximación al resultado deseado. En función de varios parámetros, como el número de episodios o el ajuste de las distintas constantes, esta aproximación será más o menos acertada.

Además, este algoritmo suele tener un balance exploración-explotación: en las primeras iteraciones, interesa explorar para conocer el entorno, por lo que la estimación es

pobre y no se obtendrán buenos resultados. En cambio, a medida que la estimación mejora y el algoritmo gana experiencia, interesa explotar, por lo que se obtendrán resultados mejores que los vistos hasta el momento.

2.3. DEEP Q NETWORK

Una vez comentado el algoritmo Q-Learning, se va a explicar en qué consiste una variante del mismo basada en redes neuronales, la cual es conocida como Deep Q-Learning Networks.

A pesar de que Q-Learning es un algoritmo simple a la vez que potente que permite elaborar una guía de ayuda en la toma de decisiones del agente, este se queda obsoleto si el problema a resolver consta de gran cantidad de estados y acciones posibles, por ejemplo, un entorno con 10.000 estados y 1.000 acciones por estado supondría una matriz Q con 10 millones de celdas. Además, los valores asociados a cada una de las celdas no se pueden inferir basándose en los valores del resto, sino que es necesario recorrer cada estado y probar cada acción posible para evaluar el resultado del mismo, lo cual supone 2 problemas principales: la cantidad de memoria requerida para guardar y actualizar la matriz Q que aumentará conforme el número de estados aumente y en segundo lugar, la cantidad de tiempo que sería necesario emplear para explorar todo el espacio de estados-acciones para poder completar la matriz.

Una solución propuesta para este problema es el de aproximar los valores de la matriz a través de modelos de aprendizaje automático como por ejemplo las redes neuronales. Esta idea se le ocurrió a la compañía de inteligencia artificial DeepMind, cuyo algoritmo fue comprado por Google por 500 millones de dólares.

Las diferencias principales entre Deep Q-Learning Network y Q-Learning es la salida que proporciona el algoritmo y la manera de “almacenar” los valores relativos a cada par estado-acción. Con estas técnicas de redes neuronales se sustituye la matriz de valores de Q-Learning por el aprendizaje de una función de generalización, la cual servirá para seleccionar la mejor acción a tomar en cada caso a la vez que permite tomar una buena decisión si se topara con un estado que no ha sido explorado (cosa que en Q-Learning supone un problema ya que si el estado no está explorado, no se dispone de ningún valor para evaluar qué acción tomar). Además, permite manejar casos en los que los estados son más complejos que un simple tablero con dimensiones XYZ. Respecto a la salida devuelta, en el caso de Q-Learning, se devuelve un valor que hace referencia a la acción que maximiza la recompensa, como se ha comentado anteriormente. Por otra parte, Deep Q-Learning Network devuelve todo un conjunto de valores asociados a cada una de las acciones para el estado en el que se encuentra y finalmente se escoge la mejor. Esto permite llamar a la red

neuronal una vez por cada estado en vez de una vez para cada par estado-acción. Como entrada a la red neuronal se introduce el estado y como salida se tienen n valores (n acciones posibles). De este modo, toda la experiencia pasada es almacenada en la memoria (función de generalización), la acción a tomar se escoge en función de la salida máxima de la red neuronal.

Respecto a la fase de entrenamiento, al igual que se ha comentado anteriormente en otros algoritmos como Montecarlo ó Q-Learning, aquí también se usa un algoritmo ε-voraz, el cual permitirá conseguir un balance entre exploración y explotación, es decir, se explorará la mayor parte de estados a la vez que se va almacenando en memoria cómo de buena es cada acción para cada uno de los estados. Otra similitud que tiene con Q-Learning es el uso de la misma ecuación de actualización de valores:

$$\mathrm{Q}(s_t, a_t) \coloneqq \mathrm{Q}(s_t, a_t) + \lambda \cdot \left[r_t + \gamma \cdot \max_a \mathrm{Q}(s_{t+1}, a) - \mathrm{Q}(s_t, a_t)\right]$$

Figura 5. Ecuación de actualización de valores

No obstante, en este algoritmo se introduce el concepto de aprendizaje por diferencia temporal, lo cual hace referencia a una clase de métodos que aprenden mediante el arranque a partir de la estimación actual de la función de valor. En este caso, para entrenar la red se utiliza el método de descenso de gradiente para minimizar el término TD-residual, de la Figura 6.

$$\left[r_t + \gamma \cdot \max_a \mathrm{Q}(s_{t+1}, a) - \mathrm{Q}(s_t, a_t)\right]$$

Figura 6. Descenso por gradiente.

Algunas de las ventajas de Deep Q-Learning Network son: la reducción de la memoria necesaria para almacenar la matriz de valores Q para entornos muy grandes, ya que ahora no es necesaria, es capaz de afrontar estados no vistos durante la fase de entrenamiento gracias a la generalización que permite hacer la red neuronal. Respecto a las desventajas, es determinista y por tanto no puede aprender políticas estocásticas (incertidumbre), no se puede aplicar directamente a espacios de acción continuos, es necesario discretizar, se requiere añadir el algoritmo epsilon-voraz de manera externa a la red para conseguir el balance de exploración-explotación comentado.

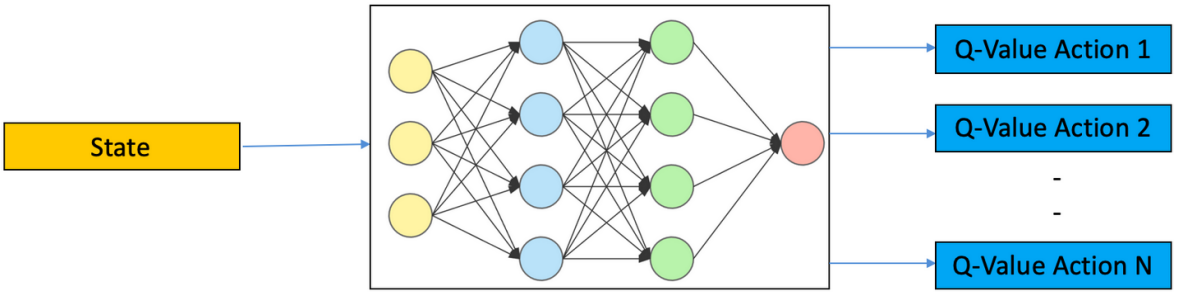

Figura 7. Esquema de la red neuronal en el algoritmo Deep Q Network

2.4. A3C: Asynchronous Advantage Actor-Critic

Este algoritmo, al igual que el anterior, fue desarrollado por DeepMind de Google, que es la división de Inteligencia Artificial de Google. Es una modificación del algoritmo llamado Actor-Critic (AC).

Actor Critic es un paradigma de aprendizaje por refuerzo en el que existen múltiples procesos con funciones bien diferenciadas:

- En primer lugar, el proceso "crítico" tiene como función modificar la red a partir de las experiencias recogidas y estimar la función de valor que después se comunicará al resto de procesos.
- El "actor" se encarga de ejecutar las políticas estimadas por el crítico sobre una copia del entorno a resolver. Este proceso cuenta con una red neuronal independiente que toma como función de entrada la estimación del crítico para devolver la acción a ejecutar.

El concepto de ventaja presente en el nombre del algoritmo se debe a que, a diferencia de algoritmos que utilizan técnicas de descenso por gradiente, como DQN (Deep Q Network), este tipo de algoritmos introducen este concepto para estimar no solamente cómo de buenas o malas fueron las acciones tomadas, sino también en qué medida estas acciones serían mejores o peores de lo que se pudiera esperar. Esto permite al algoritmo centrarse en la parte de la red neuronal central (crítico) donde las predicciones se ajustan menos a la realidad e intentar corregirlas.

A diferencia de DQN, donde un agente individual representado por una red neuronal interacciona con un solo entorno, A3C utiliza múltiples agentes para aprender de manera más eficiente. Al tratarse de un algoritmo del tipo actor-crítico, cuenta con una red global para estimar las funciones de valor y una serie de workers independientes a los que se les asigna su propia serie de parámetros de aprendizaje. Cada uno de estos workers interacciona

con su propia copia del entorno de manera simultánea. La premisa que justifica esta solución como una mejora ante A2C (versión anterior a A3C que no contiene la parte asíncrona) es que la experiencia de cada agente tomará decisiones independientes al resto, lo que permitirá unas experiencias más diversas, y por tanto, permitirá una exploración del entorno mucho más en profundidad. No obstante, diversos estudios han comprobado que ambos algoritmos producen un rendimiento similar, siendo en muchos casos A2C una opción más eficiente.

En la siguiente imagen se muestra el esquema general de funcionamiento del algoritmo A3C.

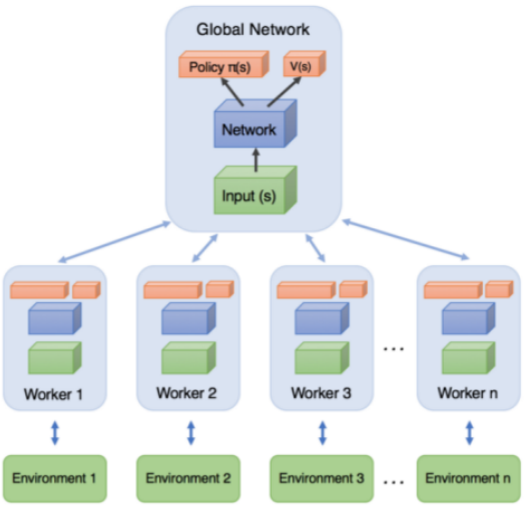

Figura 8. Diagrama de arquitectura del algoritmo A3C.

En esencia, A3C se trata de uno de los algoritmos más competitivos que existen en la actualidad. Esencialmente supera a DQN en todos los aspectos, siendo este más rápido, simple, robusto y capaz de conseguir puntuaciones mucho más altas en una gran mayoría de tareas típicamente asignadas para la validación de estos algoritmos.

3. APLICACIONES

3.1. ENTORNO REAL

Un claro ejemplo de utilización de este tipo de aprendizaje automático en el mundo real es el caso de la RoboCup. Este es un proyecto internacional de promoción del estudio de la inteligencia artificial mediante la competición entre distintos equipos de investigación en un torneo de fútbol con robots móviles, cuyo objetivo final es tener listo para 2050 un equipo de robots autosuficientes capaces de ganar al campeón de la Copa Mundial de la FIFA de ese mismo año. Existen gran variedad de categorías:

- Liga de simulación, donde no existen robots físicos, sino que se enfrentan en un terreno virtual.
- Liga de robots de tamaño pequeño.
- Liga de robots de tamaño medio.
- Liga de robots con cuatro patas.
- Liga de robots humanoides.

La primera edición se llevó a cabo en 1997 y desde entonces se han seguido realizando ediciones de la misma año tras año, incrementando la cantidad y calidad de datasets y algoritmos en este ámbito.

La gran mayoría de trabajos se destinan a la mejora de tareas que ya existen, como puede ser correr o chutar. En varios estudios se habla de la mejoría que el aprendizaje por refuerzo puede conseguir en este tipo de objetivos que, a priori, pueden parecer sencillos. A continuación se detallan algunos de estos trabajos.

3.1.1 Lanzamiento de pelota en un robot sin extremidades

Este estudio, llevado a cabo por la Universidad Carnie Mellon de Pittsburgh, USA, en 2018, detalla cómo llevaron a cabo la implementación de una serie de habilidades (*skills*) en unos robots móviles diseñados por ellos mismos denominados CMDragons, Figura 9.

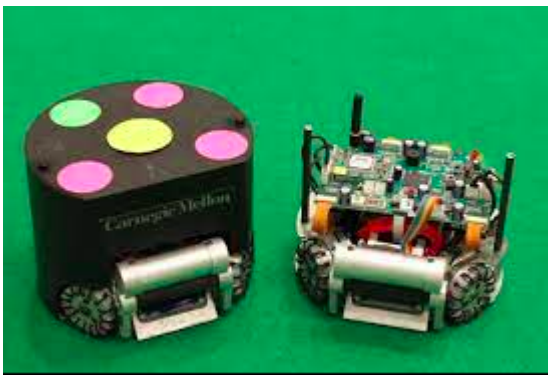

Figura 9. CMDragons con y sin carcasa.

Para ello, utilizaron un algoritmo de aprendizaje por refuerzo para espacios continuos que no se ha detallado anteriormente, el Deep Deterministic Policy Gradient (DDPG). Este está íntimamente relacionado con Q-Learning pero resulta más complejo que este, podría describirse como el Deep Q-Learning para espacios continuos.

Así pues, en un entorno de simulación realista, se representó mediante vectores de estado las habilidades *go-to-ball* (ir a la pelota) y *aim-to-shoot* (apunta para disparar).

En el primer caso, el vector estaba compuesto de variables como la posición de la pelota (P_x^B, P_y^B), la velocidad del robot (V_x^R, V_y^R, ω^R), la distancia entre el robot y la pelota ($d_{r\text{-}b}$), entre otras características:

$$s = (P_x^B, P_y^B, V_x^R, V_y^R, \omega^R, d_{r-b}, x_{top}, y_{top}, x_{bottom}, y_{bottom}, x_{left}, y_{left}, x_{right}, y_{right})$$

Figura 10. Vector de estados del go-to-ball.

Además, utilizaban recompensas ajustadas y complejas como las de la Figura 11, donde la recompensa total era la suma de las 3 que aparecen.

$$r_{contact} = \begin{cases} 100 & \text{ball on the dribbler} \\ 0 & \text{ball not on the dribbler} \end{cases}$$

$$r_{distance} = \frac{5}{\sqrt{2\pi}} \exp(\frac{-d_{r-b}^2}{2}) - 2$$

$$r_{orientation} = \frac{1}{\sqrt{2\pi}} \exp\left(-2\frac{\theta_{r-b}}{\pi^2}\right)$$

Figura 11. Recompensas del go-to-ball.

En el segundo caso, las variables abarcaban también casos como la velocidad de la pelota (V_x^B, V_y^B), la distancia del robot a la portería ($d_{r\text{-}g}$) o los senos y cosenos de los ángulos que formaba con los postes izquierdo y derecho de la portería.

$$s = (P_x^B, P_y^B, V_x^B, V_y^B, \omega^R, d_{r-g}, \sin(\theta_l), \cos(\theta_l), \sin(\theta_r), \cos(\theta_r))$$

Figura 12. Vector de estados del aim-to-shoot.

Aquí las recompensas son más sencillas y se basan en la velocidad de la pelota y si esta se acerca a la portería o no:

$$r = \begin{cases} 0.05(\alpha - \beta)|V^B| & \alpha > \beta \\ (\alpha - \beta)|V^B| & \alpha < \beta \end{cases}$$

Figura 13. Recompensas del aim-to-shoot.

Finalmente, ambas habilidades se juntaron en una tarea con la que el robot era capaz de aproximarse a la pelota, apuntar a la portería y disparar, logrando resultados muy prometedores para futuros avances en el ámbito.

3.1.2 Correr y parar en seco con un humanoide

Una de las habilidades más importantes en el fútbol es correr. El primer jugador en alcanzar el balón tiene ventaja sobre el equipo contrario y lo mismo pasa en la RoboCub. Para mejorar las probabilidades de que esto ocurra, es decir, que un robot llegue primero a la pelota, es necesario desarrollar mecanismos de funcionamiento rápidos y estables (que el robot no pierda estabilidad al correr y pueda frenar rápido). Para esta competición, se usó una versión del robot humanoide NAO de la empresa SoftBank Robotics

Para este caso, también se ha utilizado un algoritmo de aprendizaje por refuerzo para estados continuos que no se ha visto anteriormente llamado Proximal Policy Optimization (PPO). Es un algoritmo de Descenso por Gradiente, al igual que DDPG. La premisa principal de PPO consiste en evitar grandes variaciones en las políticas aplicadas haciendo cambios de manera progresiva con la técnica de descenso por gradiente. De esta manera, se pretende influenciar al agente para que a medida que avance la exploración, ejecute acciones que aumente la recompensa total y evite el resto.

No se va a explicar el algoritmo completo ya que tiene partes complejas, pero lo más importante es saber que usa el "objetivo sustitutivo recortado" (*clipped surrogate objective*), el cual se define mediante la siguiente fórmula:

$$L(\theta) = \hat{\mathbb{E}}_t[min(r_t(\theta)\hat{A}_t, clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t],$$
$$where \;\; r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{old}(a_t|s_t)}},$$

Figura 14. Fórmula de optimización del algoritmo PPO.

donde épsilon es un hiperparámetro con valor 0.1 o 0.2 normalmente. El primer término dentro de min es $r_t(\theta)\,\widehat{A}_t$ es el ratio de probabilidad bajo la nueva y antigua política multiplicado por una ventaja estimada en el tiempo t. El segundo término $clip(r_t(\theta), 1-\epsilon, 1+\epsilon)\,\widehat{A}_t$ modifica el objetivo recortando el ratio de probabilidad, el cual elimina el incentivo de mover r_t fuera del intervalo $[1-\epsilon, 1+\epsilon]$. Finalmente se elige el mínimo entre el objetivo recortado y no recortado (segundo y primer término respectivamente), así el objetivo final es un valor más bajo y puede servir como cota pesimista.

En la simulación, se anima al robot a correr lo más rápido posible en línea recta, ya que la recompensa solo viene dada por la diferencia en la coordenada x del torso del robot

desde la última medida de tiempo. A su vez, como restricción del aprendizaje, el episodio termina si el robot se cae y si logra superar la distancia estipulada para la prueba.

En cuanto al tema de entrenar la parada, ocurre lo mismo que para el tema de correr. Se puede suponer que el estado inicial del robot es corriendo y el estado final deseado es la posición de reposo. El robot recibe una recompensa por acabar en un estado parecido a la posición deseada y es penalizado si acaba en una posición alejada a esta. Además, también recibe un bonus si evita la caída. El episodio termina si tras un tiempo de 1s el robot no es estable y cae al suelo.

3.2. ENTORNO SIMULADO

Como ya se ha comentado anteriormente, este tipo de aprendizaje automático se emplea en múltiples tipos de videojuegos. Uno de los más conocidos es la resolución de laberintos. En este, el robot se encuentra en un entorno cerrado del cual debe localizar la salida. Para complicar dicho objetivo, el espacio cuenta con trampas, que provocan una recompensa negativa, y paredes, por las cuales el robot no puede pasar.

En este tipo de casos, el agente cuenta con 4 acciones posibles: izquierda, derecha, arriba y abajo. Los estados dependerán del tamaño del tablero escogido, ya que cada casilla representa un posible estado del agente. Las recompensas se otorgarán en base al mapa establecido.

Una vez conocido el funcionamiento de esta clase de videojuegos, se procede a profundizar en uno concreto: la resolución de un laberinto mediante el algoritmo Q-Learning. Este algoritmo ha sido seleccionado debido a que presenta mejores resultados que, por ejemplo, MonteCarlo, ya que este es equiprobable, mientras que Q-Learning prioriza la exploración.

Para la implementación de este ejemplo, se ha creado un environment mediante la librería Gym de OpenAI que permite el desarrollo de un entorno de manera sencilla. La interfaz escogida ha sido la siguiente:

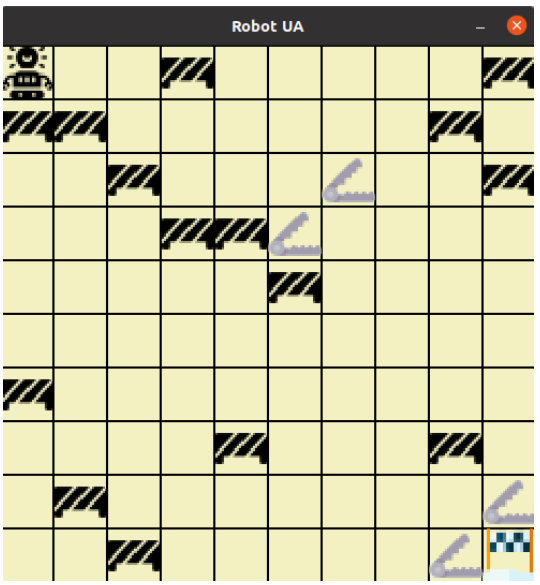

Figura 15. Interfaz diseñada para el laberinto.

Como se puede observar, el laberinto cuenta con distintas paredes por las cuales el robot no puede pasar, además de varias trampas, las cuales deberán proporcionar una recompensa negativa para que el robot encuentre la trayectoria óptima hasta la meta, consiguiendo así evitar dichos obstáculos.

La implementación del algoritmo se basa en el pseudocódigo aportado en el apartado 2, concretamente en la sección de Q-Learning.

Conocido el tamaño del tablero gracias a las funciones que proporciona la librería Gym y establecido el número de episodios para resolver el problema, en este caso 200, se procede a implementar dicho código.

El primer paso es decidir la acción a realizar siguiendo una política. En este caso, se ha escogido una política ϵ-voraz. Para mejorar su eficacia, a este valor de ϵ se le aplica un factor de descuento, δ. Ambos valores han sido ajustados para lograr el mejor resultado. Seguidamente, se actualiza la posición en el tablero y se actualiza la matriz Q. Esta almacena el valor que corresponde al estado actual. Con dicha información, se actualiza la recompensa, el estado y ϵ.

Estos pasos se repiten hasta completar el número de episodios establecido. Para comprobar la eficacia de este algoritmo, se muestra por pantalla las recompensas en el estado actual, además de la media de estas hasta el momento.

Se puede observar que en las primeras iteraciones el robot no se aproxima al resultado esperado, pero a lo largo de los episodios consigue una pronta convergencia,

gracias al ajuste de los parámetros necesarios y a la experiencia que proporciona este tipo de algoritmo.

Para un mejor entendimiento del funcionamiento de este algoritmo, se adjunta un vídeo donde se observa la correcta ejecución del mismo, además del resultado que proporciona:

Vídeo laberinto Q-Learning

Como se muestra en el vídeo, el robot comienza explorando todo el mapa y va convergiendo conforme avanzan los episodios. Por pantalla, se muestra el número del episodio en el que se encuentra, la recompensa actual y la media total de las recompensas acumuladas hasta el momento.

Gracias a este algoritmo, el robot consigue una trayectoria óptima con una convergencia rápida en un entorno con suficientes obstáculos.

4. CONCLUSIONES

Las técnicas de aprendizaje por refuerzo han demostrado ser muy útiles en la última década para la resolución de problemas que hasta hace pocos años eran inabordables. Además de la eficiencia que proporcionan, se componen de algoritmos fácilmente programables en comparación con otros que requieren una formación muy avanzada en programación y aprendizaje automático. Otra ventaja es la gran cantidad de información disponible que se puede encontrar y la fácil accesibilidad a la misma.

Con referencia a los algoritmos presentados en el proyecto, cabe destacar que la mayoría de ellos son muy usados y eficientes en sus respectivos campos. No obstante, para temas de robótica móvil en los que el entorno suele componerse de estados contínuos y gran cantidad de acciones posibles, los que destacarían son Deep Q-Learning Network y A3C debido a que tratan con mayor eficiencia estos entornos. Por otra parte, además de los algoritmos abordados en este trabajo, existe un gran número de algoritmos junto a otros que siguen apareciendo con el paso de los años que también son capaces de resolver estos problemas de manera eficiente.

Respecto al tema que se aborda, el aprendizaje por refuerzo proporciona una alternativa a la planificación de trayectorias tradicional, o en cualquier caso, una ayuda para abordar dicha tarea. Este punto se viene demostrando desde hace unos años con la RoboCup que, además, proporciona año tras año una enorme base de datos de los diferentes proyectos presentados, muy útil para avanzar cada vez más en esta dirección.

5. BIBLIOGRAFÍA

1. Aplicaciones del aprendizaje reforzado en robótica móvil. KENZO IGNACIO LOBOS TSUNEKAWA, Universidad de Chile, 2018.
2. Aprendizaje por refuerzo en robótica móvil. Sandra Rudkowskyj Hernanz, 2019.
3. Estrategias de aprendizaje automático aplicadas a videojuegos. Universidad Politécnica de Valencia, 2019.
4. Métodos value-based: Deep Q-Network. Medium.2021.
5. DeepMind.
6. DQN, Vídeo Youtube
7. Transparencias Manipuladores 2021-2022. Ingeniería Robótica, Universidad de Alicante.
8. Learning from Delayed Rewards.Christopher John Cornish Hellaby Watkins, 1989.
9. Q-Learning.Christopher John Cornish Hellaby Watkins, 1992.
10. Simple Reinforcement Learning: Q-learning. TowardsDataScience, 2019.
11. Deep Deterministic Policy Gradient. OpenAI.
12. Learning to run faster in a humanoid robot soccer environment through reinforcement learning. Miguel Arbreu, Luis Paulo Reis, Nuno Lau. Universidad de Porto, Universidad de Aveiro, 2019.
13. Learning Skills for Small Size League RoboCup.Devin Schwab, Yifeng Zhu, and Manuela Veloso. Carnegie Mellon University, Pittsburgh, 2018.