Entrega: Viernes 15 de enero de 2021

Trabajo teórico: uso de aprendizaje por refuerzo para robótica móvil

Pascual Tornero Martín
Daniel Ruiz Rodríguez
Javier Navarro Martínez

Robots Móviles
Grado en Ingeniería Robótica
Escuela Politécnica Superior, Universidad de Alicante
Curso académico 2020-2021

Universitat d'Alacant
Universidad de Alicante

Índice

INTRODUCCIÓN **3**

1. DESCRIPCIÓN Y CLASIFICACIÓN DEL APRENDIZAJE AUTOMÁTICO **3**

1.1. Política y recompensa 4

1.2. Ejemplo práctico 5

2. APRENDIZAJE POR REFUERZO EN ROBOTS **6**

2.1. Ventajas del aprendizaje por refuerzo en robots móviles 6

2.2. Problemas al aplicar RL en robótica móvil 7

2.2.1. Solución al problema del espacio de estados continuo 8

2.2.2. Solución al problema de las recompensas sparse 8

3. ALGORITMOS DE RL PARA ROBÓTICA MÓVIL **9**

3.1. Q-Learning 9

3.2. Deep Q-Learning 12

4. PROYECTOS DE RL PARA ROBÓTICA MÓVIL **15**

4.1. Aprendizaje por refuerzo para enseñar a un robot humanoide a caminar más rápido 16

4.2. Generación de comportamientos de enjambre en robots móviles a través del uso de aprendizaje por refuerzo 18

4.3. Conducción autónoma mediante aprendizaje por refuerzo 21

4.4. Ejemplos aprendizaje por refuerzo con Gym de OpenAI 22

CONCLUSIONES **24**

BIBLIOGRAFÍA **24**

INTRODUCCIÓN

En el presente trabajo se realizará una introducción al aprendizaje por refuerzo para robots móviles. En primer lugar, el objetivo es exponer qué es el aprendizaje automático y dónde se ubica el aprendizaje por refuerzo dentro de él. También la razón por la que contemplar su utilización en robótica móvil. Además, se explicará el funcionamiento del aprendizaje por refuerzo y algunos algoritmos basados en él y las ventajas, desventajas y principales retos que existen. Por último, se mostrará diversos proyectos que lo aplican en robótica móvil y se realizará pruebas con algunos de ellos

1. DESCRIPCIÓN Y CLASIFICACIÓN DEL APRENDIZAJE AUTOMÁTICO

El aprendizaje automático se basa en la programación de un computador para que mejore en la realización de una tarea a partir de los datos de ejemplo o de la experiencia. Dentro del aprendizaje automático hay diversos tipos. A continuación, se explica brevemente los principales, para poder comprender donde se ubica el aprendizaje por refuerzo y qué lo caracteriza:

- Aprendizaje supervisado: se posee un conjunto de ejemplos para el entrenamiento, y la clase a la que pertenece cada ejemplo utilizado para el aprendizaje es conocida.
- Aprendizaje no supervisado: la clase de cada ejemplo no es conocida.
- Aprendizaje por refuerzo: no se utiliza un conjunto de datos de ejemplo, se aprende en base a la experiencia, pudiéndose recibir recompensas positivas o negativas en el cambio de un estado a otro.

Por lo tanto, el aprendizaje por refuerzo o *Reinforcement Learning (RL)* se caracteriza porque el agente recibe una serie de recompensas que le permiten aprender a partir de la experiencia. Además, no es posible indicar en cada situación qué acción debe realizar o es la correcta, como en aprendizaje supervisado, si no que tendrá que ir aprendiendo en su periodo de vida.

El aprendizaje por refuerzo posibilita diseñar para un agente comportamientos sofisticados que resultarían difíciles de programar manualmente. Permite resolver problemas de toma de decisiones secuenciales: el agente se encuentra en un estado concreto y debe tomar una decisión sobre qué acción va a realizar, la cual le lleva a un nuevo estado (o al mismo), en el que de nuevo tendrá que realizar una acción. Por lo tanto, se va moviendo entre estados, y el aprendizaje consiste en determinar con la experiencia qué acciones debe tomar en cada estado para maximizar la recompensa acumulada.

Mayoritariamente se aplica a problemas que se pueden modelar como procesos de decisión de Markov (MDP) que constan de un conjunto de estados S, entre los que pueden moverse mediante una serie de acciones del espacio A, y existe una función de transición que

determina, estando en un estado y realizando una determinada acción, qué probabilidad existe de ir a otro estado. También de una función de recompensa, que determina la probabilidad de recibir una recompensa u otra al estar en un estado y tomar una determinada acción. Por último, poseen la característica de que, cuando el agente está en un determinado estado, todos los estados anteriores por los que se ha pasado ya no importan, es decir, el pasado del agente no es tenido en cuenta.

Por lo tanto, existe un agente que comienza en un estado inicial y va tomando una serie de acciones, a partir de la cual el entorno le devuelve el nuevo estado en el que se encuentra y la recompensa obtenida de realizar esa acción. Respecto a los agentes, cabe mencionar que no son necesariamente una entidad completa: un agente es aquel dispositivo, objeto u ente en el que se puede tomar decisiones. Este concepto se entiende mejor con un ejemplo de un robot aspirador: el agente podría ser el control de los motores del robot, pero el microcontrolador de la batería que indica al robot que está casi sin batería (para que, por ejemplo, vuelva a su base de recarga), pese a ser parte del propio robot, pertenecería al entorno.

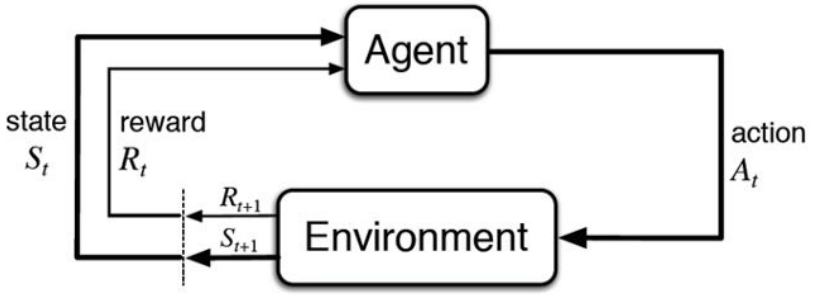

Figura 1. Diagrama que muestra al agente enviando (realizando) una acción, y el entorno devolviendo el nuevo estado en el que se encuentra el robot, y la recompensa obtenida por cambiar al nuevo estado.

1.1. Política y recompensa

La estrategia que sigue el robot para seleccionar una acción u otra se denomina política. Respecto a la recompensa, se trata de un valor escalar y es la ganancia inmediata que devuelve el entorno al realizar una acción. Puede ser positiva o negativa. No obstante, el objetivo es maximizar la ganancia acumulada. La recompensa a lo largo del proceso se conoce como retorno: es la suma de las recompensas a lo largo del proceso, multiplicadas por un factor de descuento con el objetivo de que pesen más las recompensas obtenidas más cercanas en el tiempo. Es la función de valor, que se modela con los entrenamientos, cuya estimación permite maximizar o escoger la mejor opción de las posibles para obtener la mejor recompensa acumulada al final del episodio.

Es necesario tener en cuenta que en entornos desconocidos el agente no puede saber cuál es la política óptima. No puede saber si hay un conjunto de decisiones que vayan a lograr una recompensa mejor.

1.2. Ejemplo práctico

A continuación, se va a mostrar un ejemplo simplificado en el que un robot móvil debe alcanzar un objetivo. La intención es explicar cómo se resolvería de forma tradicional, y cómo se resolvería con aprendizaje automático. En este ejemplo, aparece un robot (azul) que debe recorrer diversas casillas del mapa hasta alcanzar una posición objetivo (verde), siendo las casillas blancas camino libre, y las rojas paredes por las que no puede avanzar. Además, como resulta obvio, será mejor que las alcance recorriendo el menor número de casillas (y por tanto menos tiempo) posible. Cabe mencionar que la posición del objetivo es desconocida, y únicamente se conocerá que se ha llegado al objetivo una vez se alcance, al menos inicialmente que el entorno no es conocido. Además, el robot no ve más allá del estado en el que se encuentra y los siguientes estados a los que puede avanzar.

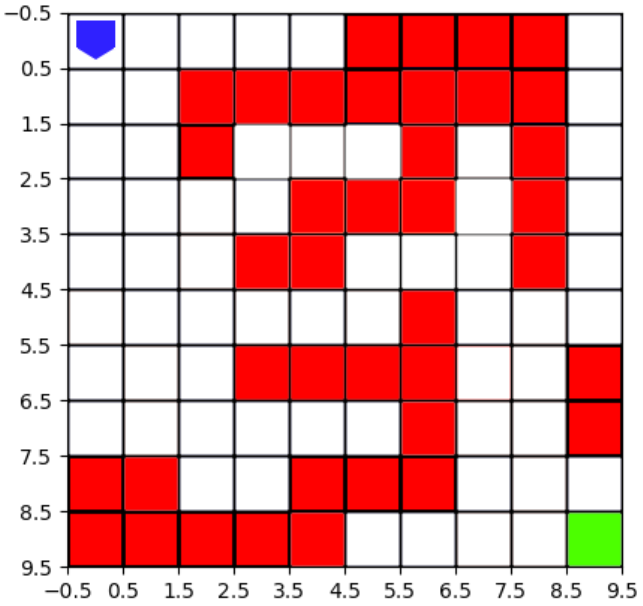

Figura 2. Ejemplo de tablero del problema descrito.

La solución tradicional, sin aplicar aprendizaje por refuerzo, podría ser programar al robot para que recorriera todas las casillas a las que puede avanzar. También tendría que tener algún tipo de almacenamiento de que casillas ha recorrido, para no repetir. Esto haría, a su vez, que debiera implementar un algoritmo que le permita la vuelta atrás por casillas ya recorridas cuando queda atascado en un camino sin salida. Por otro lado, si está almacenando el camino recorrido para futuros episodios, debería existir algún tipo de condicionante que eliminase los caminos sin salida del camino recorrido. Otra solución sería, en vez de almacenar el camino, almacenar la posición del objetivo y optar por una algoritmo voraz para alcanzarlo. Sin embargo, por sí solo no sería funcional para alcanzar la solución, pues podría quedar atascado en mínimos locales: se deberá entonces aplicar algún tipo de solución, como por ejemplo ocurría con el modo “Seguir pared” de método VFF, para poder salir de él.

Se observa entonces que con programación tradicional surgen dos problemas: la decisión de por qué solución optar, y el problema de la complejidad que pueda requerir esa solución para ser funcional. Sin embargo, la solución con aprendizaje por refuerzo para este

problema resulta más rápida de programar, pues únicamente se debe concretar el espacio de estados y las posibles acciones y aplicar algún algoritmo ya existente de aprendizaje por refuerzo como MonteCarlo o Q-Learning. También establecer las recompensas, que para este caso sería óptimo que fuese -1 (a excepción de cuando alcanza la casilla objetivo, que debería ser positiva) cada vez que avanza una casilla para que, como a cada movimiento de más empeora la recompensa acumulada, tenga en cuenta recorrer el menor número de estas. Después, solo quedará entrenar, siendo los movimientos que realiza el robot principalmente aleatorios al inicio, y más basados en el conocimiento adquirido conforme avance el número de entrenamientos realizados.

2. APRENDIZAJE POR REFUERZO EN ROBOTS

Una vez explicado el concepto de aprendizaje por refuerzo, se puede observar que resulta una elección adecuada para robots móviles. En vez de diseñar un control a bajo nivel de las acciones que debe realizar el robot dependiendo del estado en el que se encuentre y su objetivo, se diseña una política a alto nivel y una función de recompensa dependiendo de la adecuación de su acción a lograr cumplir el objetivo. Además, en muchas ocasiones las recompensas se corresponden con eventos del mundo, como se ha explicado, lo que ayuda a simplificar la fase de diseño.

2.1. Ventajas del aprendizaje por refuerzo en robots móviles

Por un lado, la programación de robots de forma tradicional es un proceso que puede requerir mucho tiempo. Las razones son varias: ser necesarias muchas iteraciones para ajustar el mapeo de bajo nivel de sensores a actuadores, resultar difícil plasmar las ideas de control a programación útil para el robot, e incluso existir conceptos erróneos sobre el funcionamiento de alguno de los sensores o actuadores que provoquen que la tarea falle. Sin embargo, con el uso de aprendizaje por refuerzo para robótica móvil el programador simplemente proporciona al robot especificaciones de la tarea que se quiere realizar a alto nivel, mientras que los detalles de bajo nivel se configuran de forma automática según el robot va realizando entrenamientos.

Por otro lado, los métodos actuales resuelven problemas como la realización de trayectorias de un robot móvil con la posesión de un mapa interno del mundo, utilizando localización y con planificadores de trayectorias para estos mapas. Esto se traslada a multitud de ejemplos, incluso coches autónomos, donde los mapas son el mapeado que se posee del mundo y las carreteras, la localización es el posicionamiento GPS y se consta de un planificador complejo para estos mapas. Además, este tipo de soluciones poseen una serie de problemas, como que realizan una serie de suposiciones como la inexistencia de objetos transparentes o el mapa estático. También requieren una gran cantidad de recursos computacionales, y no aprenden de fallos y errores cometidos en ejecuciones previas, por lo que no mejoran por sí solos. No ocurre así haciendo uso del aprendizaje por refuerzo, que

permite que los robots móviles mejoren continuamente su rendimiento, y permiten poder ir aprendiendo a funcionar en un gran número de entornos cuyas características concretas no podrían haberse estimado en una programación común.

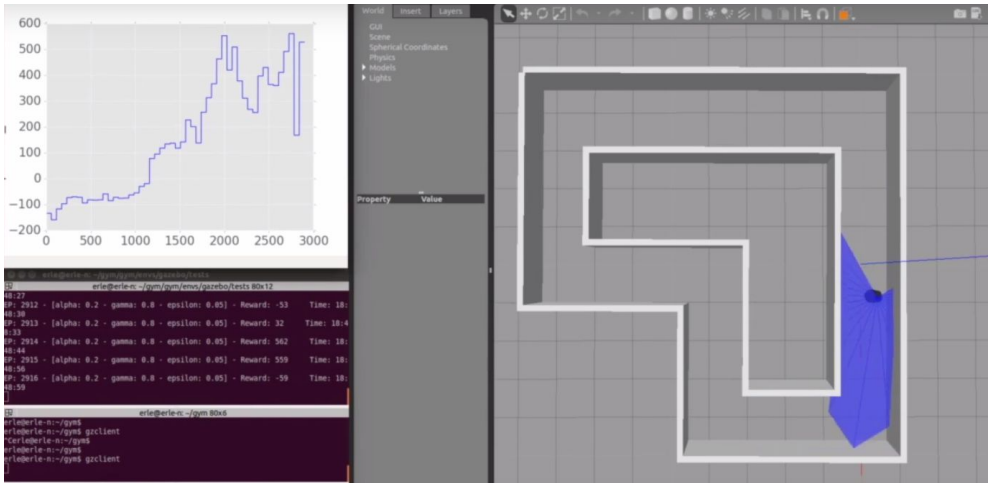

Figura 3. Robot recorriendo un pasillo en Gazebo, utilizando lo aprendido en aprendizaje por refuerzo en vez del *Navigation Stack* proporcionado por ROS. Se observa a la izquierda una gráfica que permite ver la mejora de la recompensa obtenida al haberse completado un mayor número de episodios.

Además, en la actualidad se avanza hacia un panorama en el que la conectividad entre todo tipo de dispositivos aumenta. Esto permite que lo aprendido por un robot pueda ser comunicado al resto, de forma que todos los robots con acceso a este conocimiento puedan mejorar en la realización de la tarea. Un sistema similar al que ya aplican empresas como Tesla, la cual recoge datos estadísticos de cada uno de los coches que posee circulando que le permiten mejorar la conducción autónoma de sus vehículos.

2.2. Problemas al aplicar RL en robótica móvil

Sin embargo, surgen una serie de cuestiones y problemas cuando se trata de aplicar el aprendizaje por refuerzo a la robótica móvil:

1. El espacio de estados en robótica móvil es continuo en tanto en cuanto un estado es una posición en la que se encuentra el vehículo. Sin embargo, los algoritmos de aprendizaje por refuerzo como Q-Learning requieren estados discretos. Intuitivamente, la solución pasa por discretizar el espacio, pero esto también puede generar problemas como introducir estados ocultos. También, si no se discretiza con una tolerancia muy pequeña entre una posición y otra, habrá un gran número de estados y se aumentará la cantidad de datos de entrenamiento necesarios.
2. El diseño de la función de recompensa, pues en ocasiones puede ser difícil asignar los valores de las recompensas para que el aprendizaje sea correcto. Principalmente hay dos tipos de recompensa, las *sparse* y las *dense*. En las primeras, la mayoría de recompensas son 0, a excepción de una recompensa negativa cuando se comete algún error (como por ejemplo, impactar con un objeto) y una positiva cuando se llega al objetivo. Respecto las segundas, más sofisticadas, consisten en estar dando tras cada

acción una recompensa distinta de 0, como puede ser la suma de distancias a los obstáculos dividida entre la distancia a la posición objetivo.

3. Las recompensas *sparse*, utilizadas por su implementación más sencilla, poseen un problema en el caso de la robótica móvil: si ocurre que en muchas iteraciones no se llegan a encontrar recompensas que no sean 0, el agente no aprenderá ya que las funciones de valor de cada estado no cambian. Esto resulta un inconveniente para entornos grandes.
4. Incorporar la prioridad de aplicar conocimiento respecto de aprender. La forma que tiene el aprendizaje por refuerzo de recoger información sobre su entorno y aprender es realizar acciones y observar lo que ocurre. Inicialmente, cuando desconoce su entorno, las acciones son arbitrarias. Cuanta más información, más puede mejorar el resultado. Sin embargo, también surge la cuestión de cuándo conviene comenzar a explotar el conocimiento que se posee.

2.2.1. Solución al problema del espacio de estados continuo

Pese a que algunos de los problemas citados son intrínsecos de la aplicación de aprendizaje por refuerzo a la robótica móvil, otros pueden ser y han sido solucionados. Respecto al problema que ocurre porque el espacio de estados en robótica móvil es continuo, se han encontrado diferentes soluciones.

La propuesta en el paper *Practical Reinforcement Learning in Continuous Spaces* pasa por sustituir los estados (posiciones) discretos en las tablas que almacenan las recompensas acumuladas promedio de realizar cada acción en cada estado. Se sustituyen por un aproximador de funciones, capaz de manejar variables continuas con más de una dimensión y generalizar en estados similares. Con este objetivo se introduce en el *paper* el algoritmo HEDGER. Este algoritmo permite una aproximación segura de la función de valor de Q-Learning utilizando *locally weighted regression* (LWR), que es una variación de las técnicas de regresión lineal estándar.

2.2.2. Solución al problema de las recompensas *sparse*

Respecto al problema de las recompensas *sparse* y el riesgo de que el agente no aprenda, el paper *Effective Reinforcement Learning for Mobile Robots* propone una solución consistente en proporcionar trayectorias de ejemplo al aprendizaje y, además, dividir este en dos fases.

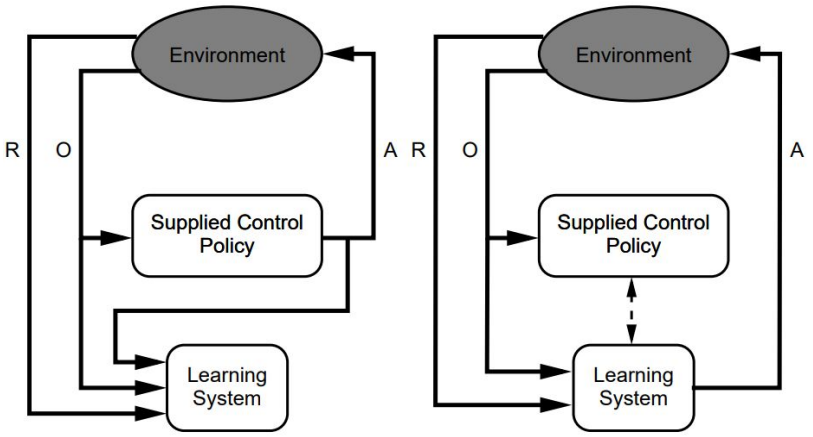

Figura 4. Aprendizaje en dos fases. En la primera fase se observa que las acciones a realizar salen de la *Supplied Control Policy*, que envía la acción al entorno e informa al sistema de aprendizaje de la acción escogida. En la segunda fase, la acción la genera el sistema de aprendizaje.

En la primera fase del aprendizaje, el robot es controlado por el *Supplied Control Policy*, que podría tratarse de un operador humano indicando el estado siguiente en cada caso, por ejemplo controlando el robot con un joystick. En esta fase, el sistema de aprendizaje del agente únicamente observa los estados, acciones y recompensas que se generan por la actuación del control. Esto se lleva a cabo para que el sistema de aprendizaje tenga una primera aproximación de las funciones de valor para poder suprimir el problema que provocaba que las funciones de valor no cambiasen por no encontrar estados con una recompensa diferente de cero. Es en este momento cuando se pasa a la segunda fase de aprendizaje, donde es el sistema de aprendizaje quien determina la política a seguir, y el el *Supplied Control Policy* no interviene, excepto que se desee mantener en segundo plano por si fuera aconsejable ejecutar alguna acción de control.

3. ALGORITMOS DE RL PARA ROBÓTICA MÓVIL

El objetivo en este apartado es explicar el funcionamiento de algunos de los principales algoritmos de aprendizaje por refuerzo utilizados en robótica móvil, como Q-Learning y una versión mejorada que recibe el nombre de Deep Q-Learning.

3.1. Q-Learning

Este es uno de los algoritmos más utilizados actualmente en el aprendizaje por refuerzo. El funcionamiento del algoritmo Q-Learning se basa en conseguir la estimación de una función de valor para cada uno de los pares estado-acción posibles para el entorno en el que se está desarrollando el aprendizaje. El valor de estos pares se almacena en una estructura de datos llamada Q(s, a) donde *s* es el estado, *a* es la acción y el valor contenido es una medida de la calidad observada al tomar la acción *a* en el estado *s*. A lo largo de los episodios, los valores de Q se actualizan de forma que aquellas acciones en cada estado que contengan un valor más alto sean la elección más adecuada para maximizar la recompensa

acumulada. Esta estructura funciona muy bien para los casos en los que el entorno tiene un número de estados y acciones razonables, sin embargo, cuanto mayores sean estos valores mayor necesidad de memoria y tiempo será necesaria para el almacenamiento de los valores.

En el aprendizaje por refuerzo, cada estado es consecuencia de los estados anteriores. Sin embargo, guardar toda esta información puede no resultar factible incluso para los escenarios en los que el número de episodios sea pequeño. Debido a esto, para el algoritmo se ha concebido un punto de vista de los procesos de decisión de Markov (MDP), según el cual el estado actual es solamente consecuencia del anterior y de la transición entre ellos mismos.

En cuanto a la actualización de los valores de la estructura Q(s,a) se sigue la fórmula siguiente que se deriva de la ecuación de Bellman:

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

La anterior fórmula muestra la actualización de la tabla al realizar una acción a_t desde el estado s_t, recibiendo una recompensa r_{t+1} y una transición al estado s_{t+1}. En la anterior ecuación, α es la tasa de aprendizaje que toma un valor entre 0 y 1, que determina cuánto afecta la información que se acaba de adquirir a la actualización de la información que se tenía anteriormente, prevaleciendo en mayor medida la información anterior cuanto más bajo es el valor de α. Además, también aparece γ, que indica la importancia que se le da a la recompensa futura. El valor que se le otorga a esta variable afecta a la convergencia ya que pondera la recompensa inmediata (r_{t+1}) con la futura ($Q(s_{t+1}, a)$). Esta variable toma un valor entre 0 y 1, que normalmente suele ser 0.99, y cuanto mayor es, más se pondera la recompensa futura frente a la inmediata, por lo que el agente se vuelve más "visionario", valorando en mayor medida las recompensas futuras. Otros valores a tener en cuenta son $Q(s_t, a_t)$, que se corresponden con la información que se tiene hasta ese momento para el estado s_t y la acción a_t, también r_{t+1} que es la recompensa obtenida al realizar la acción a_t en el estado s_t, y por último $\max_a Q(s_{t+1}, a)$ que es el valor máximo contenido en la estructura Q para el estado siguiente s_{t+1}, teniendo en cuenta todas las posibles acciones, a modo de mostrar la posible futura recompensa de tomar la acción a_t en el estado actual.

Enfocándose en la inicialización de la estructura Q, se pueden hallar distintas posibilidades, entre las cuales la más utilizada es inicializar todas sus celdas a un mismo valor positivo. Con esto se logra que, al actualizar los valores como se ha visto anteriormente, se alcancen valores negativos para el contenido de las celdas, los cuales serán más bajos cuanto peor se considere la acción a la que correspondan. Por otro lado, las acciones no probadas tendrán un valor positivo mientras no se prueben, por lo que se fuerza, de cierta manera, a explorar todas las posibilidades en cada estado. Esto pretende producir la exploración de un mayor número de opciones, objetivo del aprendizaje automático.

Como se puede ver, el algoritmo Q-Learning sirve para estimar el valor de realizar una determinada acción en un determinado estado. Sin embargo, es necesario definir una

política ($\pi_Q(s_t)$) para escoger qué acción tomar cuando el agente se encuentra en un determinado estado. Existen numerosas posibilidades para la elección de la política a aplicar, como pueden ser la aleatoria, consistente en elegir la acción de forma aleatoria, la voraz, que escoge la acción que maximice el valor de Q(s, a), y finalmente, la más empleada: ϵ-voraz, que calcula un número aleatorio entre 0 y 1, y si este es menor que ϵ se elige la acción de forma aleatoria; en caso contrario, se elige la acción con el valor máximo para ese estado en Q(s, a). Como se ha dicho, la política más usada para Q-Learning es la ϵ-voraz, y esto es así gracias a que permite obtener un equilibrio exploración-explotación, que es la idea consistente en que el algoritmo tiene la necesidad de explorar las diferentes acciones para cada estado y una vez se tiene un determinado conocimiento explotarlo para elegir la mejor opción en cada caso. El balance anterior permite que para los primeros episodios, al ser pobre la estimación de Q(s, a), se pueda explorar mucho. A medida que avanzan los episodios y se adquiere conocimiento es mejor explotar, pero sin dejar de explorar por completo en ningún momento. Con la política ϵ-voraz se logra este balance ya que al final de cada episodio se puede cambiar el valor de ϵ para que vaya disminuyendo, como por ejemplo con $\epsilon = \epsilon\delta$, donde δ es un valor entre 0 y 1 que regula la velocidad de decaimiento, es decir, que rápido se deja de explorar y se comienza explotar. Cabe destacar que Q-Learning no impone ningún requerimiento sobre la política utilizada, pero se debe tener en cuenta que esta puede tener impacto en la convergencia del algoritmo y en el balance exploración-explotación.

Una vez vistos varios conceptos que son importantes para el entendimiento de Q-Learning, se muestra con pseudo código la implementación del algoritmo:

1. $Q \leftarrow$ initialize(S, A);
2. **while** number of episodes is not reached **do**
3. | $s_t = s_0$;
4. | **while** episode is active **do**
5. | | $a_t \leftarrow \pi_Q(s_t)$;
6. | | $s_{t+1}, r_{t+1} \leftarrow$ perform(a_t);
7. | | $Q[s_t, a_t] = (1-\alpha)Q(s_t, a_t) + \alpha(r_{t+1} + \gamma \max_a Q(s_{t+1}, a))$;
8. | | $s_t = s_t + 1$;
9. | **end**
10. | change(ϵ)
11. **end**

A continuación, se realiza una explicación de la implementación mostrada anteriormente. En primer lugar, en la línea 1 se inicializa la estructura de datos Q(s, a) y en la línea 2 se comienza un bucle que ejecuta el resto del código mientras que no se hayan realizado el número de episodios fijados. Más adelante, en la línea 3 se inicializa el estado actual como el estado inicial del episodio que será siempre el mismo para cada uno de los episodios y en la línea 4 se comienza un bucle que estará activo hasta que se finalice el episodio actual en la línea 9. Una vez se ha entrado en este bucle se producen la mayor parte de las operaciones del algoritmo. En la línea 5 se escoge la acción a tomar en el estado actual según la política

definida, en la 6 se ejecuta la acción escogida y se obtiene el estado siguiente (s_{t+1}) y la recompensa de realizar la acción (r_{t+1}). En la línea 7, se actualiza el valor de la estructura $Q(s_t,a_t)$ como se ha visto anteriormente y en la línea 8 se elige como estado actual el estado siguiente obtenido en la línea 6, terminando el bucle y volviendo a la línea 5 mientras no haya terminado el episodio. Finalmente, en la línea 10 se cambia el valor de ϵ para actualizar el balance exploración-explotación como ya se ha mencionado en el párrafo anterior.

En resumen, el algoritmo Q-Learning sirve para estimar la mejor acción escogible para cada estado tras un número de episodios, y es uno de los algoritmos más utilizados de aprendizaje por refuerzo. Sin embargo, presenta inconvenientes tales como que el algoritmo está limitado por su tabla de índices, ya que con un gran número de acciones y estados esta estructura se vuelve inconcebible y no es factible para un número de estados muy grande. Se han propuesto mejoras a este algoritmo como estrategias de cuantización que añaden parámetros a ajustar y aproximaciones de la función de valor de Q(s, a) entre las cuales se encuentra Deep Q-Learning, algoritmo que se estudia en el siguiente subapartado.

3.2. Deep Q-Learning

Este algoritmo surge de la necesidad de cubrir las desventajas encontradas en el algoritmo Q-Learning debido a que al aumentar el número de estados y acciones posibles la memoria necesaria para poder almacenar los valores de Q(s, a) se vuelve demasiado alta, al igual que el tiempo necesario para explorar cada estado. El algoritmo Deep Q-Learning se trata de una mezcla entre el Q-Learning y la utilización de redes neuronales, que resultan muy útiles para aproximar funciones no lineales, que para el caso de este algoritmo permite aproximar la función de valor de Q(s, a). De esta manera, se introduce como entrada el valor del estado actual (s_t) a la red neuronal y se obtiene como salida los valores de $Q(s_t, a)$ para todas las acciones posibles. En la siguiente imagen se muestra la comparación entre Q-Learning y Deep Q-Learning.

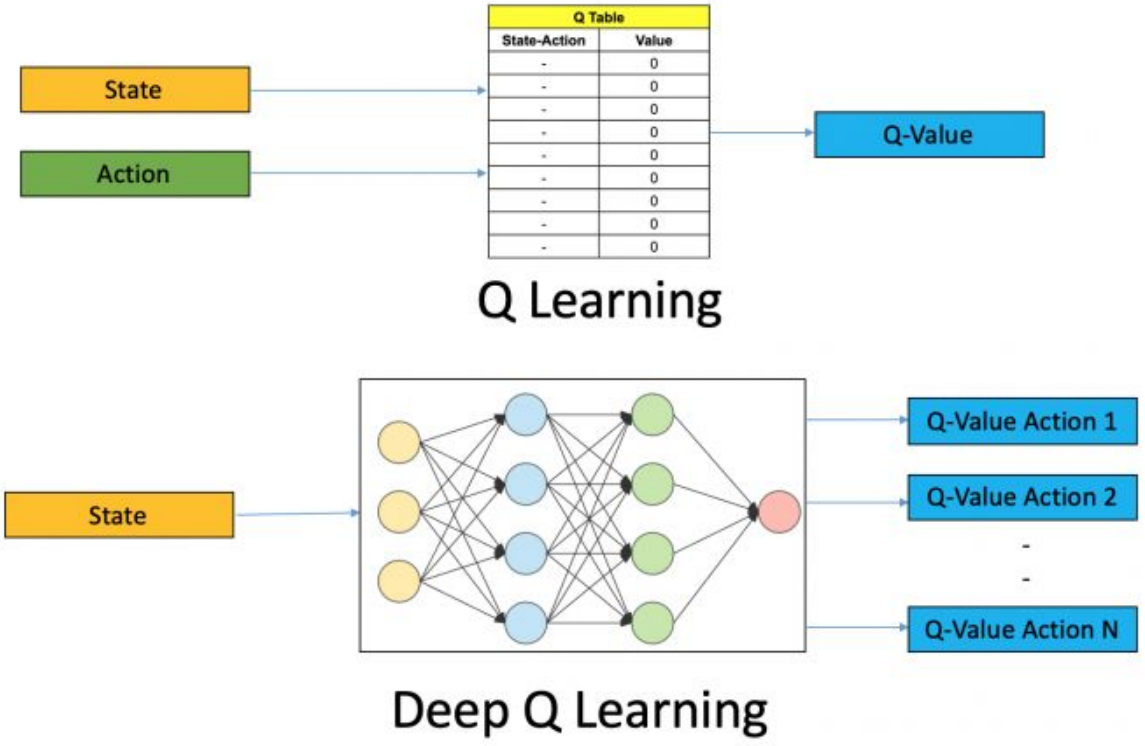

Figura 5. Comparación entre los algoritmos Q-Learning (arriba) y Deep Q-Learning (abajo)

Para poder aplicar la red neuronal es necesario establecer sus parámetros de entrenamiento. En primer lugar, se debe definir la función de coste de la misma, que está formada por el error cuadrático medio entre los dos lados de la ecuación de Bellman. Se tiene que la ecuación de Bellman para la red neuronal es la siguiente:

$$Q(s, a, \boldsymbol{\Theta}) = r + \boldsymbol{\gamma} \max_a Q(s', a', \boldsymbol{\Theta})$$

donde $Q(s, a, \boldsymbol{\Theta})$ es el valor predicho y $(r + \boldsymbol{\gamma} \max_a Q(s', a', \boldsymbol{\Theta}))$ el valor objetivo de la función de valor de Q por la red neuronal. Como se ha dicho, la función de valor es el error cuadrático medio entre ambos lados de la ecuación de Bellman y el resultado quedaría:

$$L(\boldsymbol{\Theta}) = E[(r + \boldsymbol{\gamma} \max_a Q(s', a', \boldsymbol{\Theta}) - Q(s, a, \boldsymbol{\Theta}))^2]$$

Por lo tanto, esta es la función que se minimiza usando el algoritmo de descenso por gradiente (gradient descent) típico de las redes neuronales.

Al introducir la red neuronal surgen nuevas premisas a seguir por el algoritmo. En primer lugar, la acción a escoger será la que maximice la salida de la red neuronal de Q, y en segundo lugar, toda la experiencia pasada será almacenada en memoria, cuya utilidad se explicará más adelante.

Explicado el funcionamiento de Deep Q-Learning y expuestas sus ventajas, surge un reto cuando se compara el Deep RL con el Deep Learning. Si se introduce la ecuación de la red neuronal en el algoritmo de Q-Learning anteriormente visto, se puede observar como el objetivo de la red $(r + \boldsymbol{\gamma} \max_a Q(s', a', \boldsymbol{\Theta}))$ va cambiando con cada iteración (cambio de estado

en cada episodio y entre episodios). En el Deep Learning se entrena con un objetivo definido que no cambia, por lo que el entrenamiento es estable, hecho que no ocurre con el aprendizaje por refuerzo. Por tanto, se depende de las funciones de política o valor en el aprendizaje por refuerzo para muestrear acciones. Sin embargo, esto cambia con frecuencia a medida que se aprende continuamente qué explorar. Según se avanzan los episodios se conoce más acerca de los valores de verdad básicos de los estados y acciones y, por lo tanto, la salida también cambia. Entonces, se está intentando aprender a mapear para una entrada y salida que cambia constantemente, por lo que es necesario encontrar una solución.

Dado que la misma red está calculando el valor previsto y el valor objetivo, podría existir una gran divergencia entre estos dos. Por ello, en lugar de usar una red neuronal para aprender, se deben usar dos. Se puede utilizar una red separada para estimar el objetivo. Esta red objetivo tiene la misma arquitectura que el aproximador de funciones pero con parámetros fijos temporalmente. Cada C iteraciones (un hiperparámetro ajustable), los parámetros de la red de predicción (valor previsto) se copian en la red del objetivo (valor objetivo). Esto conduce a un entrenamiento más estable porque mantiene fija la función objetivo, al menos durante un determinado período. En la siguiente imagen se ilustra un esquema del proceso descrito.

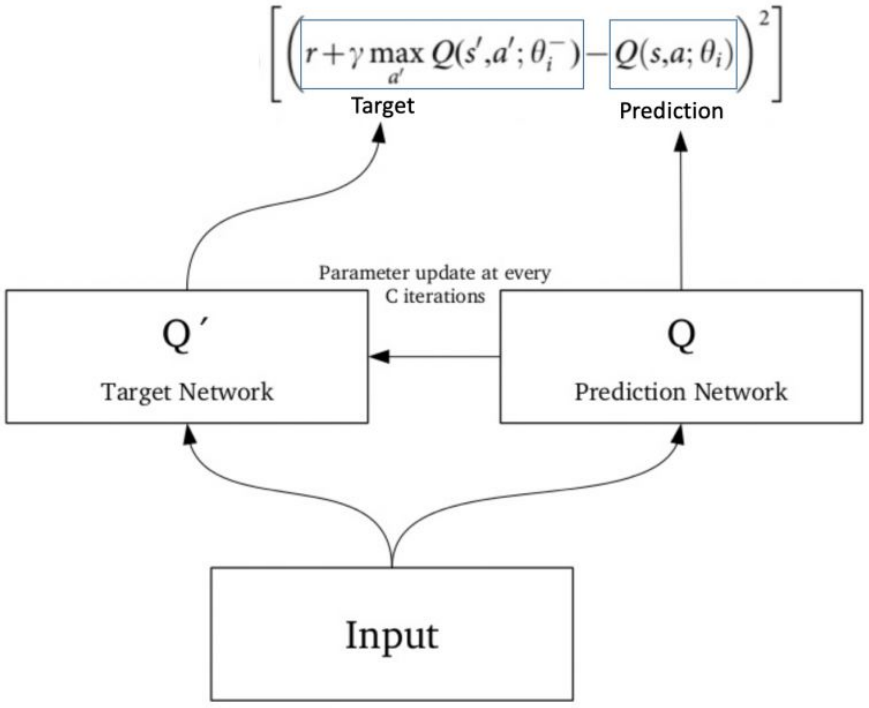

Figura 6. Esquema de funcionamiento con dos redes neuronales, una para el objetivo (izquierda) que cada C iteraciones cambia sus parámetros ($\boldsymbol{\Theta}_i^-$) por los de la red de predicción ($\boldsymbol{\Theta}_i$) que se visualiza a la derecha. La ecuación de arriba es la de coste con la cual se entrena la red y que se ha definido anteriormente.

Una vez conocido cómo se aplican las redes neuronales al Deep Q-learning, es necesario analizar otro concepto importante en el Deep Q-learning: la repetición de

experiencias. Una cosa que puede llevar a nuestro agente a malinterpretar el entorno son estados interdependientes consecutivos que son muy similares. Por ejemplo, si se le está enseñando a un robot móvil autónomo a circular correctamente y la primera parte de la carretera es solo una línea recta, es posible que el agente no aprenda a manejar las curvas del camino, y es aquí donde entra en juego el concepto de repetición de la experiencia. A partir del ejemplo del robot móvil autónomo, lo que sucede con la repetición de experiencias es que las experiencias iniciales de navegación en línea recta no pasan por la red neuronal de inmediato, si no que el agente guarda estas experiencias en la memoria como se ha mencionado anteriormente. Una vez que el agente alcanza un cierto umbral de repeticiones, hace una selección aleatoria de las experiencias de esta memoria y aprende de ellas. Cada experiencia se caracteriza por el estado en el que se encontraba, la acción que tomó, el estado en el que terminó y la recompensa que recibió ($[s_t, a_t, r_{t+1}, s_{t+1}]$). El muestreo aleatorio permite evitar que sólo aprenda de un tipo de experiencia, como por ejemplo, de conducir en línea recta.

Finalmente, una vez vistos varios conceptos que forman parte del algoritmo de Deep Q-Learning, se listan, a continuación, los pasos seguidos en el algoritmo:

1. Introducir a la red neuronal el estado actual s_t para obtener el valor de Q(s, a) de todas las posibles acciones a tomar.
2. Seleccionar la siguiente acción a realizar utilizando la política ϵ-voraz, cuyo funcionamiento se detalla en el apartado 3.1.
3. Hacer la acción que se ha escogido en el paso anterior y obtener el estado siguiente y la recompensa de haber desarrollado esta acción. Se almacena en la memoria la tupla $<s_t, a_t, r_{t+1}, s_{t+1}>$ para que se pueda utilizar más tarde en el mecanismo de repetición de experiencias.
4. Muestrear datos aleatorios de la memoria para la repetición de experiencias.
5. Calcular la función de coste de la red neuronal de la forma:
$$L(\boldsymbol{\Theta}) = E[(r + \boldsymbol{\gamma} \max_a Q(s', a', \boldsymbol{\Theta}) - Q(s, a, \boldsymbol{\Theta}'))^2]$$
6. Ejecutar el algoritmo de descenso por gradiente con los parámetros actuales de las redes.
7. Cada C iteraciones copiar los pesos de la red de predicción a los de la red objetivo.ç
8. Repetir lo anterior mientras no se haya terminado el episodio.
9. Realizar lo mismo que en los 8 pasos anteriores mientras queden episodios por experimentar.

4. PROYECTOS DE RL PARA ROBÓTICA MÓVIL

A continuación, se van a explicar algunos proyectos hallados en los que se aplica aprendizaje por refuerzo para soluciones de robótica móvil.

4.1. Aprendizaje por refuerzo para enseñar a un robot humanoide a caminar más rápido

Este proyecto posee como objetivo conseguir que un robot humanoide, en este caso el robot NAO, aprenda a caminar más rápido cuando ya ha aprendido a andar. Se hace uso del aprendizaje por refuerzo seguro, el cual asume la existencia de una política segura que permite aprender una nueva, que se representa con un enfoque basado en casos. Para la realización de este proyecto se ha hecho uso de la plataforma ROS que se ha estudiado en la asignatura, utilizando paquetes como el del robot NAO propio de ROS e intercambiando mensajes de distintos tipos, genéricos de ROS como *geometry_msgs, nao_msgs, humanoid_step_msgs* y *rl_msgs*. Con estos recursos el proyecto es capaz de simular el robot en un entorno como el que se ve en la imagen siguiente en la aplicación NAOsim.

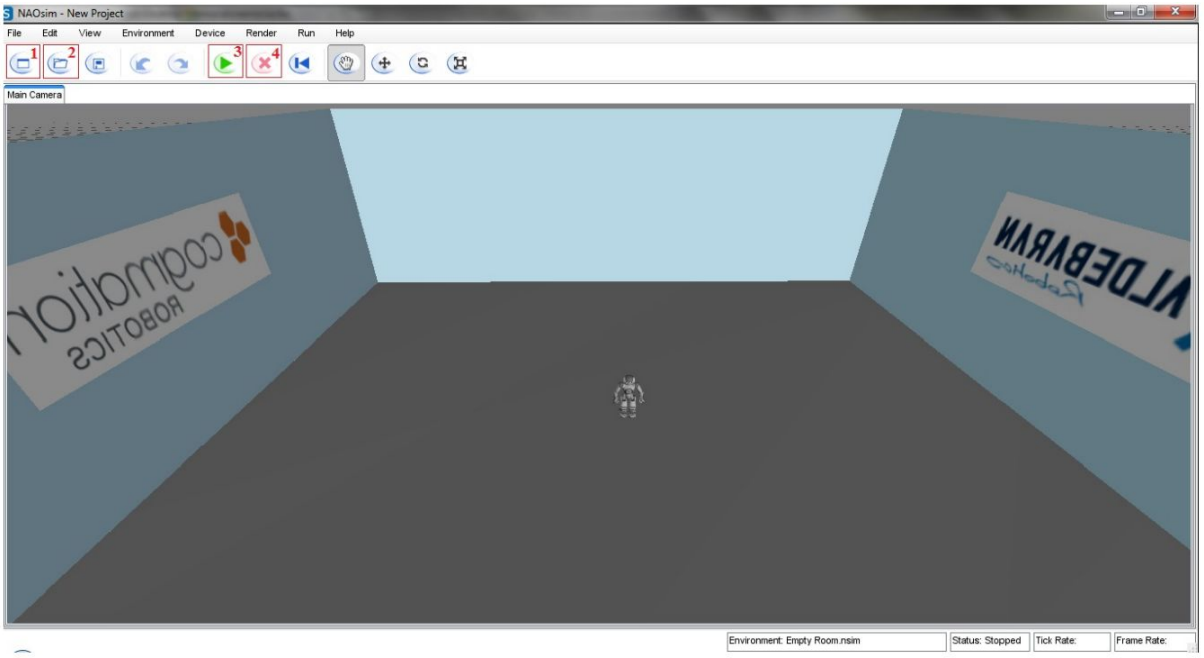

Figura 7. Entorno de simulación en NAOsim utilizado para el aprendizaje del robot.

Una vez vistas las bases del proyecto, se puede indagar en cómo se ha planteado este escenario para el aprendizaje por refuerzo. El comportamiento de caminar se puede tratar desde dos niveles de abstracción distintos:

- Abstracción de bajo nivel: el comportamiento de caminar se descompone en una sucesión de ángulos de las articulaciones del robot, compuestos por 26 variables, mostradas en la figura 8.
- Abstracción de alto nivel: el comportamiento de caminar se descompone en una secuencia de pasos, como se puede ver en la figura 9.

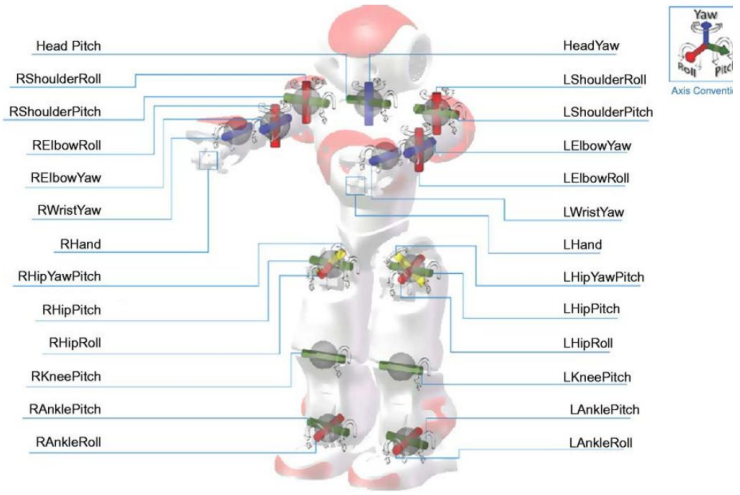

Figura 8. Conjunto de variables que entran en juego para la abstracción de bajo nivel.

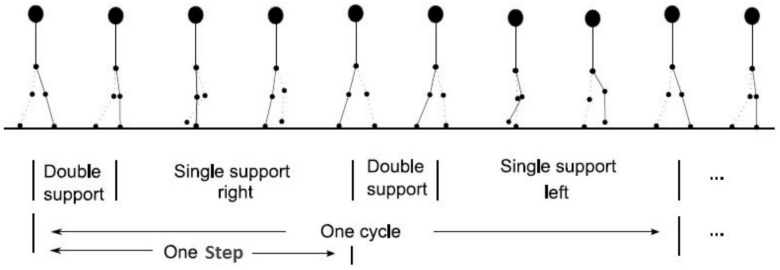

Figura 9. Comportamiento de caminar en la abstracción de alto nivel.

El nivel de abstracción en este proyecto es el de alto nivel, ya que para el dominio de bajo nivel no se dispone de los mecanismos suficientes para la obtención de experiencia válida y de mecanismos de evaluación, como puede ser la distancia recorrida por cada paso o acción. Esta abstracción de alto nivel representa el comportamiento de caminar como una sucesión de pasos hasta que el robot alcance la distancia fijada, que determina el final de cada episodio. Un episodio de este escenario está compuesto por dos fases, ya que al evaluar cada acción del proceso de aprendizaje se introduce una nueva medida de evaluación del episodio, lo que impide evaluar el episodio en una sola fase. Las fases son:

- Fase de aprendizaje: se realiza el proceso de aprendizaje del algoritmo. En esta fase, el número de pasos del episodio está determinado por los pasos realizados por el robot para completar la distancia fijada, o hasta su caída en caso de error.
- Fase de evaluación: al finalizar la fase de entrenamiento, se ejecutan secuencialmente todos los pasos seguidos y se obtiene la velocidad del episodio.

Para representar el escenario se dispone de un espacio de estados y acciones continuos. El espacio de estados está compuesto por una tupla de 3 componentes, $< x, y, \boldsymbol{\Theta} >$, siendo x e y, la separación longitudinal y transversal con respecto al pie contrario, y $\boldsymbol{\Theta}$, la rotación a través del eje vertical (Z) como se visualiza en la siguiente captura.

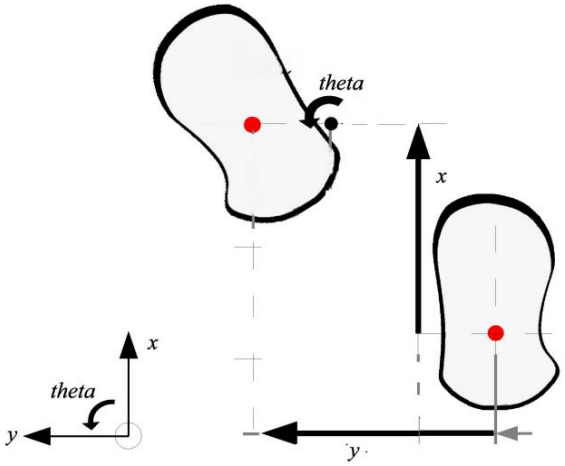

Figura 10. Representación de un estado.

El espacio de acciones se compone de una tupla de 4 componentes < Δx, Δy, Δ**Θ**, t >, que representa el desplazamiento a realizar por un pie respecto al contrario en cada una de las variables de estado (x, y, **Θ**), y el tiempo de realización del desplazamiento, t. Por otro lado, el refuerzo aplicado o recompensa a las acciones es el desplazamiento del robot en un paso ($x_{paso} - x_i$) donde x_{paso} es la x tras realizar la acción y x_i la x al principio del estado, por lo que de esta manera, el algoritmo intenta realizar pasos más largos. Adicionalmente, el refuerzo total del episodio se computa mediante la ecuación $recompensaTotal = \frac{Xf - Xi}{Tf - Ti}$, que se corresponde con la velocidad obtenida en la fase de evaluación del episodio.

Para el correcto funcionamiento en el proyecto se hace uso de los algoritmos PI-SRL y PR-SRL para poder ser comparados en cuanto a los resultados obtenidos, y además, también hace uso de Q-Learning, con los parámetros de aprendizaje **α** = 0.25 y **γ** = 0.9, que son los valores que mejor resultado arrojaban tras la experimentación. También cabe añadir que en los experimentos, se definía una distancia de 0.5 metros para cada episodio. Esta distancia se ha establecido por las limitaciones del simulador, ya que recorriendo una distancia superior a 2 metros se observa un balanceo hacia atrás en el robot que provoca la caída del mismo, hecho que no se ha observado en los experimentos con el robot NAO real. Por último, es necesario señalar que para el entrenamiento han llevado a cabo un total de 102 episodios y la política de selección de la acción ha sido similar a la **ϵ**-voraz, dando con estos parámetros un resultado satisfactorio para el experimento, tanto al simularlo como al probarlo en el robot real.

4.2. Generación de comportamientos de enjambre en robots móviles a través del uso de aprendizaje por refuerzo

En este trabajo hacen uso de técnicas de aprendizaje por refuerzo, en concreto del algoritmo Q-Learning, con el objetivo de entrenar un grupo de robots para generar comportamientos de enjambre. Se presentan dos posibles soluciones con diferentes enfoques. En la primera solución propuesta, analizada en este apartado, se establecen los estados del

robot en función de la distancia de sus dos vecinos más cercanos. En la segunda posible solución, se definen un radio de atracción y otro de repulsión, y los estados se establecen según la cantidad de vecinos dentro de cada uno de los radios divididos en los cuatro cuadrantes locales del robot, aunque esta propuesta no se desarrolla en este escrito. En este proyecto se asume que la comunicación de cada agente con sus vecinos ya está resuelta. Se realizan varias pruebas en Matlab para cada una de las soluciones propuestas variando el número de robots del enjambre.

Una vez presentado el proyecto, se centra la explicación en cómo se aplica este escenario al aprendizaje por refuerzo. Como se ha dicho anteriormente, para esta solución se asume que cada agente puede detectar la posición de sus dos vecinos más cercanos. Los estados se definen en función de la distancia de separación entre sus dos vecinos más cercanos y se toma como referencia al robot que ejecuta la acción, y a sus dos vecinos más cercanos se les llama robot A y robot B. La distancia del robot de referencia al Robot A o B se denota como RA o RB respectivamente, siendo RA < RB. Como los estados son discretos, las distancias RA y RB también se deben discretizar. En este caso se discretizan las distancias RA y RB en 3 niveles, siendo 0 una distancia muy cerca, 1 una distancia buena y 2 que el robot está muy lejos, quedando los estados que se ven en la siguiente tabla.

Estado	RA	RB
1	0	0
2	0	1
3	0	2
4	1	0
5	1	1
6	1	2
7	2	0
8	2	1
9	2	2

Tabla 1. Tabla con los diferentes estados posibles dadas las distancias RA y RB discretizadas.

Una vez vistos los posibles estados, se presentan las acciones disponibles. Estas, son el conjunto de todos los movimientos y decisiones que el agente puede tomar, como girar a la izquierda o a la derecha, avanzar o retroceder. En este proyecto se asume que los robots del enjambre son holonómicos (robots cuyo número de grados de libertad controlables es igual al número de grados de libertad totales). Para estos casos, las acciones normalmente son un número finito de posibles direcciones en las que el robot se puede mover, sin embargo para este planteamiento se definen 5 acciones que son: acercarse al robot A (acción 1), retirarse del robot A (acción 2), acercarse al robot B (acción 3), retirarse del robot B (acción 4) o permanecer quieto (acción 5).

En cuanto a la manera de otorgar las recompensas a cada acción, esta metodología se basa en reducir o aumentar la distancia de RA y RB lo suficiente hasta mantenerla en el nivel discretizado 1, que equivale a una distancia buena. La política de recompensa se representa como una función a trozos como sigue:

$$R_{t-1} = \begin{cases} RA - (Nd - 1), & a_t = 1 \\ -RA, & a_t = 2 \\ RB - (Nd - 1), & a_t = 3 \\ -RB, & a_t = 4 \\ -1, & a_t = 5, s_t \neq 5 \\ 100, & a_t = 5, s_t = 5 \end{cases}$$

donde a_t es la acción seleccionada, s_t es el estado actual y Nd son los niveles de discretización de la distancia RA y RB. Para este caso $Nd = 3$.

Una vez vistos los diferentes componentes del aprendizaje por refuerzo para la solución se muestran los pasos para desarrollarla:

1. Calcular todas las distancias entre los robots del enjambre y para cada robot mapear las distancias discretizadas de los dos robots más cercanos (distancia RA y RB, según el planteamiento descrito en esta sección).
2. Con las distancias discretizadas del paso 1 calcular el estado actual para cada robot.
3. Con la política de exploración, que en este caso es muy similar a la política ϵ-voraz vista en los algoritmos del apartado 3 del documento, se decide la acción a realizar para cada robot y se ejecuta.
4. De acuerdo con la política de recompensas, se actualizan los valores de la matriz Q con el estado actual y el estado anterior usando la ecuación de actualización de los valores de la función Q(s, a) que se vio en el apartado 3.1. Se actualizan los estados y se regresa al paso 1.

Conocidos los pasos seguidos para la implementación, solo queda hablar de los parámetros elegidos que han dado mejores resultados. En cuanto al ajuste de los parámetros del algoritmo Q-Learning, se ha definido la tasa de aprendizaje $\alpha = 0.2$ y la tasa de descuento $\gamma = 0.5$, donde se observa que α es parecida a la del proyecto anteriormente visto en el que se trabajaba con el robot NAO, y en contraste, γ resulta ser mucho menor que que la del caso citado. Además de estos parámetros se han debido definir los valores para discretizar las distancias, siendo una distancia corta cuando es menor que 0.8 unidades, una distancia buena cuando se encuentra entre 0.8 y 1.2 unidades, y finalmente, la distancia para la cual se encuentra lejos se da cuando esta es mayor que 1.2 unidades. Por último, cabe destacar que para la mayoría de los experimentos se han seleccionado un total de 250 episodios.

4.3. Conducción autónoma mediante aprendizaje por refuerzo

Como se ha explicado anteriormente, la conducción autónoma es un campo que requiere una exhaustiva programación a bajo nivel, con una gran número de parámetros que deben ser ajustados y que complican su programación y eficiencia. Por esta razón, el aprendizaje por refuerzo resulta de gran utilidad para esta tarea, puesto que no es necesario un mapa ni una reglas explícitas, confiando en un buen entendimiento del entorno inmediato.

Para poder realizar un entrenamiento del sistema de aprendizaje por refuerzo con un coche autónomo, se ha realizado primero un entrenamiento en un simulador de conducción, para después aplicarlo al mundo real. Para ello, es necesario que un agente humano esté presente en el vehículo en los primeros episodios de entrenamiento, de forma que pueda parar el automóvil y evitar accidentes y un mal entrenamiento.

El principal paradigma a resolver es formular la conducción como un problema de decisión de Markov, es decir, el estado actual depende únicamente del estado anterior y el siguiente estado dependerá solo del estado actual. Para ello, es necesario definir el espacio de estados (S), el espacio de acciones (A) y la función de recompensa (R). Una vez definidas S y A, la función de transición dependerá de los grados de libertad del agente, dictados por la morfología del vehículo utilizado (diferencial, holonómico,etc). A continuación se explican las consideraciones tomadas para definir estas variables:

a) **Espacio de estados:** La clave para definir el espacio de estados viene dada por las observaciones que el algoritmo recibe tras tomar una decisión y transitar a otro estado, es decir al realizar un paso o *step*. Se han desarrollado una gran variedad de sensores muy sofisticados para algoritmos de conducción, que no se limitan a sensores GPS, LIDAR, IMUs, o sensores infrarrojos. En este proyecto, se demuestra que con una cámara monocular unida a un sensor que mida la velocidad y dirección de las ruedas es suficiente para realizar una conducción segura y eficiente. También es necesario considerar cómo se trata la imagen debido a que esta puede ser enviada directamente al sistema de aprendizaje por refuerzo a través de una serie de convoluciones o bien, comprimir la representación de la imagen utilizando un VAE, sistema que reduce la información de las imágenes manteniendo las principales características o las que se requieran para la aplicación en concreto.

b) **Espacio de acciones:** La conducción posee ciertas acciones que le son intrínsecas como pueden ser: acelerar, frenar, señalizar, etc. Pero la representación de estas acciones no es algo tan trivial, por ejemplo: la aceleración se puede definir de forma discreta o continua en un rango [0,1]. Al discretizar la aceleración y fijar una velocidad son necesarias menos iteraciones, en cambio, al utilizar acciones continuas, a pesar de ser más complicadas de aprender, resulta en un control más suave. En este proyecto se usa un espacio de acciones bidimensional; donde los parámetros son la dirección de las ruedas y una velocidad fija en km/h.

c) **Función de recompensa**: su diseño podría consistir en una recompensa que trate de minimizar la distancia al centro del carril por el cual se está conduciendo. Sin embargo, este sistema está limitado por la intuición del humano que proporciona las recompensas. Por lo tanto, en este proyecto, en cada episodio el agente realizará una trayectoria por el carril y el episodio finalizará cuando comete alguna infracción de las leyes de circulación, entonces será recompensado en función de su comportamiento. En conclusión, el valor de cada par estado-acción será función de la distancia recorrida antes de cometer una infracción, por ejemplo, salirse del carril.

En este trabajo, se ha utilizado el algoritmo DDPG (deep deterministic policy gradients). DDPG es un algoritmo *off-policy*, es decir, que como Deep Q-learning, las acciones realizadas durante el entrenamiento provienen de una política distinta de la política óptima que habrá aprendido el agente al final del entrenamiento, debido al uso de dos redes neuronales convolucionales. De esta manera, se gana diversidad en el par estado-acción y en el balance exploración-explotación al no estar tan limitado por la política óptima.

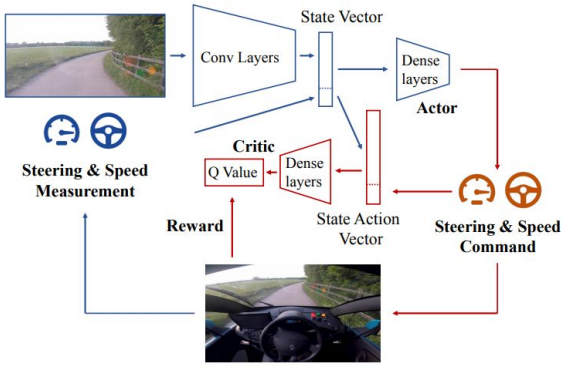

Figura 11. Esquema del algoritmo.

Como se puede observar en la figura 11, la imagen recibida por la cámara del vehículo junto a las medidas de la dirección y velocidad de las ruedas, son enviadas a una red neuronal que permite formular la política óptima para el aprendizaje, el agente elige una acción en un estado (acelerar, girar a la izquierda, etc) y recibe una recompensa. Así tras un número de episodios determinados el agente aprende a maximizar la distancia recorrida sin cometer ninguna infracción (recompensa), consiguiendo un comportamiento sofisticado y eficiente con un menor número de sensores que los necesarios en otras técnicas de aprendizaje automático supervisado. El algoritmo arroja muy buenos resultados, permitiendo que un coche equipado con una cámara y unos sensores para medir la velocidad y dirección de las ruedas, pueda llegar a aprender a conducir, manteniéndose en el carril, como se muestra en el siguiente vídeo: Learning to drive in a day.

4.4. Ejemplos aprendizaje por refuerzo con Gym de OpenAI

Open AI Gym es un conjunto de herramientas de código abierto para desarrollar algoritmos de aprendizaje por refuerzo que proporciona los entornos y técnicas necesarias

para crear proyectos, que permite definir las políticas, el número de episodios, la recompensa necesaria con una interfaz común para colecciones de problemas, también llamados entornos.

Haciendo uso de esta herramienta, se ha optado por realizar un proyecto implementando un ejemplo simple para demostrar el funcionamiento del aprendizaje por refuerzo. Cabe destacar que este proyecto sí que se ha implementado y ejecutado, no sólo se documenta. El entorno probado consiste en un robot bípedo humanoide que debe caminar por un terreno escarpado, como se ilustra en la siguiente figura:

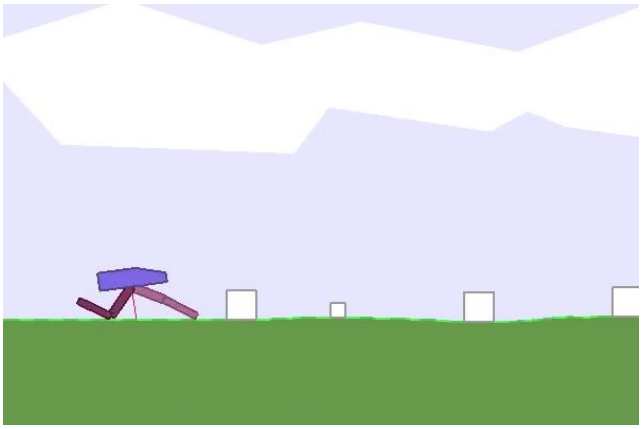

Figura 12.. Ejemplo del entorno.

Como se ha explicado a lo largo de este trabajo el aprendizaje por refuerzo está definido por la política, los pares estado-acción y la función de recompensa. El robot recibe una recompensa cada vez que consigue moverse hacia adelante, con un total de hasta más de 300 puntos si consigue llegar hasta el final. En cambio, si el robot cae recibe una recompensa de -100 puntos.

El número de estados por los que puede transitar el robot depende de muchas variables como la velocidad angular, la velocidad vertical y horizontal, el torque de los motores, la posición y el ángulo de las articulaciones y la medida del sensor LIDAR que posee el robot para calcular la distancia respecto al suelo. Al unir todas estas variables se define el estado del robot, por lo tanto, hay un gran número de estados posibles .

Las acciones que puede ejecutar el agente dependen de los grados de libertad del robot y en este caso, consisten en aumentar la velocidad de los motores de las articulaciones de la rodilla y de la cadera de cada lado. Es decir, tiene 8 posibles acciones: aumentar o disminuir la velocidad de la rodilla o de la cadera del lado izquierdo y del lado derecho.

Por lo tanto, el robot comenzará a caminar cayéndose rápidamente en los primeros episodios, tratando de explorar nuevas opciones que maximicen la recompensa (recorrer la máxima distancia). De esta forma, finalmente será capaz de implementar un comportamiento sofisticado que le permita caminar y mantenerse estable.

Cuando el robot es entrenado con un algoritmo de aprendizaje por refuerzo unido a redes neuronales convolucionales como Deep Q-Learning, es capaz de moverse en un terreno escarpado y complicado sin caerse tal y como haría un humano: esto se puede observar en el vídeo Bipedal Walker, o en la presentación que se ha realizado y complementa este trabajo, pues en ella se muestra la ejecución el ejemplo implementado.

CONCLUSIONES

A lo largo del trabajo se ha podido ver cómo funciona el aprendizaje por refuerzo y cómo se concreta su uso la robótica móvil. También se han estudiado dos de los algoritmos de aprendizaje por refuerzo más utilizados, destacando en especial Deep Q-Learning por la utilización de redes neuronales que permite mejorar la eficiencia para este tipo de tareas. Además, se han presentado proyectos prácticos de aprendizaje por refuerzo en robótica móvil realizados y se ha llegado a implementar uno de ellos, profundizando así en el proceso de aprendizaje para este tipo de tareas. Cabe destacar que el aprendizaje por refuerzo es una parte del aprendizaje automático que está al alza y que, seguramente, en los años venideros tendrá importancia en el sector de la robótica móvil, por lo que el conocer la base de su funcionamiento y poder entenderlo será de gran utilidad.

BIBLIOGRAFÍA

A continuación, se adjunta el contenido y documentos utilizados para la realización del trabajo:

[1] Gregory Kahn et al. (2018) Self-supervised Deep Reinforcement Learning with Generalized Computation Graphs for Robot Navigation
https://arxiv.org/pdf/1709.10489.pdf

[2] D.Smart,W. Pack Kaelbling, L. (2002) Effective Reinforcement Learning for Mobile Robots
https://www.researchgate.net/...Learning-for-Mobile-Robots.pdf

[3] Surmann, H. et al. (2020) Deep Reinforcement learning for real autonomous mobile robot navigation in indoor environments
https://arxiv.org/pdf/2005.13857.pdf

[4] Choudhary, A. (2019) A Hands-On Introduction to Deep Q-Learning using OpenAI Gym in Python
https://www.analyticsvidhya.com/blog/2019/04/introduction...

[5] Sanz, M. (2020) Introducción al aprendizaje por refuerzo. Parte 3: Q-Learning con redes neuronales, algoritmo DQN.
https://markelsanz14.medium.com/introducci%C3%B3n-al-aprendizaje...

[6] Foy, P. (2019) Deep Reinforcement Learning: Guide to Deep Q-Learning
https://www.mlq.ai/deep-reinforcement-learning-q-learning/

[7] Acera, D. (2013) Aprendizaje por Refuerzo Seguro para enseñar a un robot humanoide a caminar más rápido
https://e-archivo.uc3m.es/bitstream/handle/10016/17996/memoriaPFC
[8] Kendall, A. et al. (2018) Learning to Drive in a Day
https://arxiv.org/pdf/1807.00412.pdf
[9] Sokolkov, R. (2019) GitHub: Learning to Drive in a Day
https://github.com/r7vme/learning-to-drive-in-a-day
[10] (2020) Getting Started with Gym
https://gym.openai.com/docs/#getting-started-with-gym