Comparación de Algoritmos de Aprendizaje por Refuerzo en la Navegación de Robots Móviles: Q-Learning, Montecarlo y SARSA

David Fuentelsaz Rodríguez
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
davfuerod@alum.us.es

Miguel Galán Lerate
Dpto. Ciencias de la Computación e Inteligencia Artificial
Universidad de Sevilla
Sevilla, España
miggaller@alum.us.es

***Resumen*—El objetivo principal de este trabajo es comparar tres algoritmos de aprendizaje por refuerzo: Q-Learning, Montecarlo y SARSA, en el contexto de la planificación de rutas para un robot móvil en un entorno con obstáculos. El estudio se centra en evaluar la eficiencia, la convergencia, la longitud de las rutas generadas y la capacidad de evitar colisiones de cada algoritmo al navegar hacia un destino. Los resultados obtenidos muestran que cada algoritmo presenta ventajas y desventajas específicas en diferentes aspectos del problema planteado. Q-Learning demostró una rápida convergencia y generó rutas más directas en la mayoría de los escenarios, mientras que Montecarlo ofreció una mejor exploración del espacio de estados. SARSA, por su parte, destacó en entornos altamente estocásticos debido a su enfoque de aprendizaje on-policy, aunque presentó peores resultados en general y una convergencia más lenta.**

Estas conclusiones ofrecen una guía para la selección del algoritmo más adecuado según las características del entorno y los requisitos del sistema, y se sugieren mejoras futuras basadas en el aprendizaje transferencial y otras técnicas avanzadas.

***Palabras clave*—Inteligencia Artificial, Aprendizaje por Refuerzo, Q-Learning, Montecarlo, SARSA, Procesos de Decisión de Markov, Entornos Estocásticos, Política, Exploración y Explotación.**

I. Introducción

El aprendizaje por refuerzo es un tipo de aprendizaje automático que se enfoca en la toma de decisiones en entornos dinámicos y no deterministas. En este tipo de aprendizaje, el agente interactúa con el entorno y recibe recompensas o penalizaciones en función de sus acciones. El objetivo es maximizar las recompensas y minimizar las penalizaciones para encontrar la política óptima.

Esta técnica tiene una gran variedad de aplicaciones entre las que se encuentran predicciones financieras, robótica, videojuegos, medicina o cualquier problema de optimización [2] [3] [6].

En el contexto de nuestro trabajo, nos centraremos en la aplicación del aprendizaje por refuerzo a la planificación de rutas para robots móviles. En este problema, un robot con ruedas debe encontrar una ruta segura y eficiente en un entorno con obstáculos. Aunque este problema puede parecer simple a primera vista, la presencia de obstáculos, la estocasticidad en el efecto de las acciones y la necesidad de optimizar la ruta para minimizar el tiempo y los recursos hacen que sea un desafío significativo.

Para evaluar la eficacia de los algoritmos de aprendizaje por refuerzo en este contexto, utilizamos tres mapas diferentes que varían en tamaño y porcentaje de obstáculos. Estos mapas nos permitirán comparar y analizar el rendimiento de los algoritmos en una variedad de escenarios.

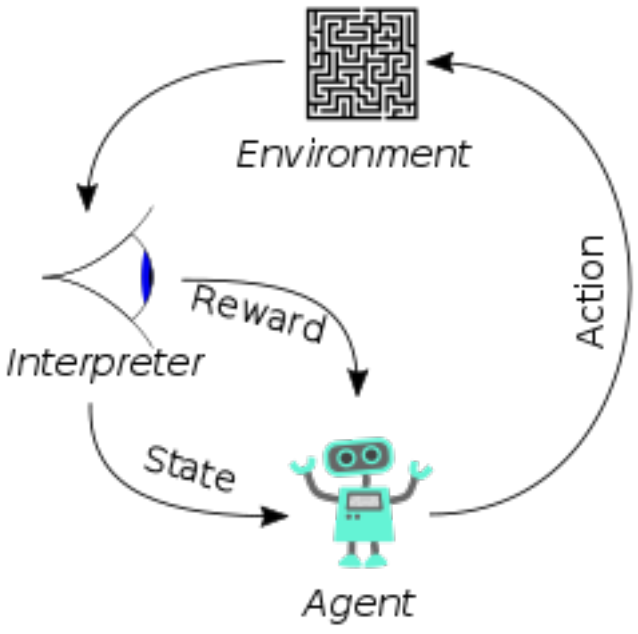

Fig. 1. Esquema del funcionamiento del aprendizaje por refuerzo.

II. Preliminares

En esta sección, proporcionamos una introducción a los conceptos fundamentales del aprendizaje por refuerzo, así como una descripción de los algoritmos de aprendizaje por refuerzo que utilizaremos en nuestro trabajo.

A. Conceptos clave del aprendizaje por refuerzo

El aprendizaje por refuerzo (RL) es un tipo de aprendizaje automático que entrena al software para tomar decisiones y maximizar una recompensa en un entorno dado a través de un proceso de ensayo y error [7].

Los componentes principales de un sistema de aprendizaje por refuerzo son:

- **Agente**: Es la parte del sistema que toma decisiones e interactúa con el entorno. En nuestro caso, el agente es el robot móvil que busca planificar rutas en un entorno con obstáculos.
- **Entorno**: Es el espacio en el que el agente se mueve e interactúa. En nuestro caso, el entorno es un mapa con obstáculos.
- **Acción**: Una acción es la decisión tomada por el agente en un estado particular. En nuestro caso, tenemos tenemos ocho acciones que representan cada uno de los posibles movimientos que puede efectuar el robot y una acción que representa que el agente permanezca en el mismo estado tras realizarla.
- **Recompensa**: La recompensa es la retroalimentación que el agente recibe del entorno después de tomar una acción en un estado dado. En nuestro caso, la recompensa puede ser positiva o negativa (también llamada penalización).
- **Política**: La política es la estrategia utilizada por el agente para seleccionar acciones en función de los estados del entorno.
- **Exploración**: Es el proceso de probar nuevas acciones y recopilar información sobre el entorno. La exploración es esencial para que el agente descubra nuevas oportunidades y estrategias que podrían llevar a recompensas a largo plazo más altas.
- **Explotación**: Es el proceso de utilizar la información aprendida para tomar decisiones que maximicen la recompensa inmediata. La explotación es importante para que el agente obtenga la mayor recompensa posible en el momento presente.

B. Descripción de los algoritmos

A continuación, describimos los algoritmos de aprendizaje por refuerzo que utilizados en nuestro trabajo:

*1) **Q-Learning**:* El algoritmo Q-Learning es un método de aprendizaje por refuerzo basado en valores que se utiliza para encontrar la política óptima de selección de acciones en un entorno determinado. Este algoritmo es una variante del aprendizaje por refuerzo que utiliza una función de valor, denominada función Q, para determinar la mejor acción en cada estado del entorno. No requiere un modelo del entorno y puede manejar problemas con transiciones estocásticas y recompensas sin requerir adaptaciones [5].

El agente explora el entorno y actualiza la tabla Q utilizando la ecuación de Bellman modificada:

$$Q(s,a) \leftarrow Q(s,a) + \alpha[R(s,a) + \gamma \max_{a'} Q(s',a') - Q(s,a)]$$

Donde:

- α es el factor de aprendizaje.
- $R(s,a)$ es la recompensa inmediata por tomar la acción a en el estado s.
- γ es el factor de descuento.
- s' es el estado siguiente al tomar la acción a en el estado s.
- $\max_{a'} Q(s',a')$ es el valor máximo esperado para cualquier acción posible en el estado s'.

Se trata de un algoritmo con un enfoque *off-policy*, lo que significa que aprende la función de valor óptima independientemente de la política que el agente esté siguiendo en el momento. Q-Learning actualiza la estimación de la función Q utilizando la recompensa inmediata y el valor máximo de la función Q para el siguiente estado y acción [1] [4] [5].

Hiperparámetros en Q-Learning

- **Número de episodios:** La cantidad total de episodios que se utilizan para entrenar el algoritmo. Un mayor número de episodios puede permitir un mejor aprendizaje, aunque también incrementa el tiempo de entrenamiento.
- **Número de pasos máximo por episodio:** El número máximo de pasos que el agente puede tomar dentro de un solo episodio antes de que termine. Esto es útil para evitar que los episodios se alarguen indefinidamente y asegurar que el agente tenga múltiples oportunidades de aprender dentro de un tiempo razonable.
- **Tasa de aprendizaje (α):** Controla cuánto se actualiza la función Q con nuevas experiencias. Un valor alto de α permite un aprendizaje rápido pero puede causar oscilaciones, mientras que un valor bajo conduce a un aprendizaje más estable pero más lento.
- **Factor de descuento (γ):** Determina la importancia de las recompensas futuras. Un valor alto de γ hace que el agente valore más las recompensas futuras, mientras que un valor bajo de γ se centra más en las recompensas inmediatas.
- **Estrategia de exploración (ϵ en ϵ-greedy):** Controla el balance entre exploración y explotación. Un valor alto de ϵ fomenta la exploración, mientras que un valor bajo de ϵ fomenta la explotación de acciones conocidas.

*2) **Algoritmo de Montecarlo**:* El algoritmo de Montecarlo es otro método de aprendizaje por refuerzo basado en valores que se utiliza para encontrar la política óptima de selección de acciones en un entorno determinado. A diferencia de Q-Learning, que actualiza sus estimaciones de valor de manera

incremental y en cada paso del episodio, los métodos de Montecarlo actualizan sus estimaciones solo al final de un episodio completo, utilizando las recompensas acumuladas desde el inicio del episodio hasta el final.

El algoritmo de Montecarlo no requiere un modelo del entorno y puede manejar problemas con transiciones estocásticas y recompensas. Sin embargo, a diferencia de Q-Learning, necesita episodios completos para actualizar las estimaciones, lo que puede hacer que sea menos eficiente en entornos donde los episodios son largos o difíciles de completar [5].

En el método de Montecarlo, el agente explora el entorno y actualiza la estimación del valor de una acción en un estado específico utilizando la recompensa acumulada observada. La ecuación para el cálculo de la utilidad es:

$$U \leftarrow \sum_{i=t}^{T} \gamma^{i-t} R_i$$

Donde:

- γ es el factor descuento que determina la importancia de las recompensas futuras en comparación con las recompensas inmediatas.
- R_i es el retorno (recompensa acumulada) del i-ésimo episodio en el que se ha visitado el par (s, a).

Hiperparámetros en el Algoritmo de Montecarlo

- **Número de episodios:** La cantidad total de episodios que se utilizan para entrenar el algoritmo. Un mayor número de episodios puede permitir un mejor aprendizaje, aunque también incrementa el tiempo de entrenamiento..
- **Descuento de recompensas (γ):** Determina la importancia de las recompensas futuras. Un valor alto de γ hace que el agente valore más las recompensas futuras, mientras que un valor bajo de γ se centra más en las recompensas inmediatas.

*3) **SARSA**:* El algoritmo SARSA (State-Action-Reward-State-Action) es un algoritmo de aprendizaje por refuerzo que se utiliza para aprender una política óptima en un entorno de Markov discreto. Es bastante similar al algoritmo Q-Learning, aunque tiene una serie de diferencias. En primer lugar, SARSA es un algoritmo con un enfoque *on-policy*, lo que significa que el agente aprende una política óptima mientras sigue la misma política que está aprendiendo. Este algoritmo también utiliza una tabla Q(s, a) que almacena el valor esperado de tomar la acción a en el estado s. No obstante, la forma en la que se actualiza Q presenta una diferencia clave respecto a Q-Learning, como se puede comprobar en la siguiente expresión:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[R(s, a) + \gamma Q(s', a') - Q(s, a)]$$

Donde:

- α es el factor de aprendizaje.
- $R(s, a)$ es la recompensa inmediata por tomar la acción a en el estado s.
- γ es el factor de descuento.
- s' es el estado siguiente al tomar la acción a en el estado s.
- a' es la acción tomada en el estado siguiente s'.

En este caso, se puede apreciar que la acción utilizada para actualizar la tabla Q en el siguiente estado es la seleccionada por la política actual. Al seguir un enfoque *on-policy*, SARSA, por lo general, suele ser un algoritmo más lento en la convergencia a la política óptima, especialmente en entornos complejos donde se requiere la exploración de numerosas acciones [4].

Hiperparámetros en SARSA

- **Número de episodios:** La cantidad total de episodios que se utilizan para entrenar el algoritmo. Un mayor número de episodios puede permitir un mejor aprendizaje, aunque también incrementa el tiempo de entrenamiento.
- **Número de pasos máximo por episodio:** El número máximo de pasos que el agente puede tomar dentro de un solo episodio antes de que termine. Esto es útil para evitar que los episodios se alarguen indefinidamente y asegurar que el agente tenga múltiples oportunidades de aprender dentro de un tiempo razonable.
- **Tasa de aprendizaje (α):** Controla cuánto se actualiza la función Q con nuevas experiencias. Un valor alto de α permite un aprendizaje rápido pero puede causar oscilaciones, mientras que un valor bajo conduce a un aprendizaje más estable pero más lento.
- **Factor de descuento (γ):** Determina la importancia de las recompensas futuras. Un valor alto de γ hace que el agente valore más las recompensas futuras, mientras que un valor bajo de γ se centra más en las recompensas inmediatas.
- **Estrategia de exploración (ϵ en ϵ-greedy):** Controla el balance entre exploración y explotación. Un valor alto de ϵ fomenta la exploración, mientras que un valor bajo de ϵ fomenta la explotación de acciones conocidas.

C. Trabajos relacionados

Para un mejor entendimiento de cómo realizar la implementación de los algoritmos, se han revisado algunos trabajos realizados:

- Reinforcement Learning: An Introduction [1].

III. Implementación

En esta sección, describiremos el método implementado en nuestro trabajo. Abordaremos los siguientes dos aspectos:

- Descripción del entorno de trabajo.
- Detalles de la implementación de los algoritmos de aprendizaje por refuerzo (Q-Learning, Montecarlo y SARSA).

A. Descripción del entorno de trabajo

En el desarrollo de este trabajo, se empleó un entorno de simulación implementado en un Jupyter Notebook. El entorno de simulación se basó en un mapa de ejemplo proporcionado en el trabajo, el cual consiste en una cuadrícula que representa el entorno en el que se moverá el robot móvil. La cuadrícula

está compuesta por casillas, algunas de las cuales contienen obstáculos (representadas con unos), mientras que otras están libres de ellos (representadas con ceros).

A partir del mapa de ejemplo, se generaron dos mapas adicionales para aumentar la variabilidad en los escenarios de prueba. Estos mapas proporcionaron diferentes configuraciones de entorno para evaluar el desempeño de los algoritmos de aprendizaje por refuerzo.

Las casillas inicial y de destino fueron seleccionadas aleatoriamente dentro de cada mapa generado, asegurándose de que ninguna de ellas contuviera obstáculos. Esta selección aleatoria garantizó la variabilidad en los escenarios de prueba y permitió evaluar el desempeño de los algoritmos en diferentes configuraciones del entorno.

En este entorno simulado, el robot móvil parte de una casilla inicial aleatoria y tiene como objetivo llegar a una casilla destino específica. Durante la simulación, el robot puede moverse entre casillas adyacentes en la cuadrícula, siguiendo una política determinada por los algoritmos de aprendizaje por refuerzo implementados. La interacción del robot con el entorno se realiza mediante acciones discretas.

Este enfoque de simulación proporcionó un entorno controlado y reproducible para evaluar el rendimiento de los algoritmos de aprendizaje por refuerzo en la planificación de rutas de robots móviles. Además, permitió explorar diferentes estrategias de navegación y analizar su efectividad en la resolución de problemas de planificación de rutas en entornos con obstáculos.

```
16 3
1111111111111111111
1000000010000000001
1000000010000000001
1000000000001000001
1000000000001000001
1111111111111111111
```

Fig. 2. Ejemplo mapa generado con la casilla objetivo.

B. Implementación de los algoritmos de aprendizaje por refuerzo

Funciones auxiliares

Aquí se describen las funciones auxiliares que han sido utilizadas en la implementación de los algoritmos de aprendizaje por refuerzo.

1. Método para leer el mapa de un fichero de texto

lee_mapa(*fichero*)
Entrada: Nombre del archivo *fichero*
Salida: Matriz que representa el mapa y el objetivo
Algoritmo:
Leer las líneas del archivo *fichero*
y convertirlas a matriz
Devolver la matriz y sus dimensiones

/

2. Método para comprobar si una casilla contiene un obstáculo

hay_colision(*estado*)
Entrada: Casilla en la que se encuentra el robot *estado*
Salida: Valor booleano que indica si hay colisión
Algoritmo:
Verificar si hay un obstáculo en *estado*
Devolver true si hay colisión, false en caso contrario

3. Método para aplicar una acción en un estado

aplica_accion(*estado*, *accion*)
Entrada: *estado* y *accion*
Salida: Nuevo *estado*
Algoritmo:
Si hay colisión entonces
  Devolver *estado*
$x, y \leftarrow$ **Coordenadas de *estado***
Si *accion* es norte entonces
  $y \leftarrow y + 1$
Si no, si *accion* es sur entonces
  $y \leftarrow y - 1$
Si no, si *accion* es este entonces
  $x \leftarrow x + 1$
Si no, si *accion* es oeste entonces
  $x \leftarrow x - 1$
Sino, si *accion* es noreste entonces
  $x, y \leftarrow x + 1, y + 1$
Si no, si *accion* es sureste entonces
  $x, y \leftarrow x + 1, y - 1$
Si no, si *accion* es suroeste entonces
  $x, y \leftarrow x - 1, y - 1$
Si no, si *accion* es noroeste entonces
  $x, y \leftarrow x - 1, y + 1$
Devolver (x, y) **como nuevo *estado***

4. Método para obtener los estados libres de obstáculos

estados_sin_obstaculos()
Salida: Lista de estados sin obstáculos
Algoritmo:
Inicializar una lista vacía llamada *estados_sin_obstaculos*
Para cada **estado *en nav_estados*** *hacer*
  Si no hay_colision(*estado*) *entonces*
    Agregar el estado a la lista *estados_sin_obstaculos*
Devolver la lista *estados_sin_obstaculos*

5. Método para obtener los posibles desvíos del agente al realizar una acción

obtiene_posibles_errores(*accion*)
Entrada: *accion*
Salida: Lista de posibles acciones erróneas
Algoritmo:
Si *accion* es norte entonces
  errores ← **['NE', 'NO']**
Si no, si *accion* es sur entonces
  errores ← **['SE', 'SO']**
Si no, si *accion* es este entonces
  errores ← **['NE', 'SE']**
Si no, si *accion* es oeste entonces
  errores ← **['NO', 'SO']**
Si no, si *accion* es noreste entonces
  errores ← **['N', 'E']**
Si no, si *accion* es noroeste entonces
  errores ← **['N', 'O']**
Si no, si *accion* es sureste entonces
  errores ← **['S', 'E']**
Si no, si *accion* es suroeste entonces
  errores ← **['S', 'O']**
Si no
  errores ← **[]**
Devolver *errores*

6. Método para escoger una acción a partir de la política $\epsilon - greedy$

escoger_accion(*estado*, *epsilon*)
Algoritmo:
Si *estado* no está en la tabla Q entonces
  Inicializar los valores de *estado* a cero en la tabla Q
Si un número aleatorio entre 0 y 1 es menor que *epsilon*
  Seleccionar una acción aleatoria
Si no
  Seleccionar la mejor acción conocida
Devolver la acción seleccionada

7. Método para obtener la política a partir de los valores de la tabla Q

obtener_politica(Q)
Entrada: Tabla Q
Salida: Política basada en Q
Algoritmo:

Inicializar *politica* como diccionario vacío
Para cada estado *en* nav_estados hacer
  Si estado *está en* Q_table entonces
    politica[estado] ← **índice de max(*Q_table[estado]*)**
  Sino
    politica[estado] ← **0**

Retornar *politica*

8. Método para obtener la recompensa de aplicar una acción a un estado

obtiene_recompensa(*estado*, *accion*)
Entrada: Estado actual *estado*, acción a tomar *accion*
Salida: Recompensa correspondiente
Algoritmo:
x, y ← ***estado***
Si *estado* es igual a *destino* entonces
  devolver *recompensa_objetivo_alcanzado*
Si hay colisión en *estado* entonces
  devolver *-penalizacion_colision*
Si *accion* es igual a 'esperar' entonces
  devolver *-penalizacion_esperar*
Para cada *accion_error*
en obtiene_posibles_errores(*accion*) hacer
  estado_vecino ← **aplica_accion(*estado*, *accion_error*)**
  Si hay colisión en *estado_vecino* entonces
    devolver *-penalizacion_casilla_adyacente_obstaculo*

En este caso, *recompensa_objetivo_alcanzado*, *penalizacion_colision*, *penalizacion_esperar* y *penalizacion_casilla_adyacente_obstaculo* son variables globales que se usan en el método para definir el valor de la recompensa en diferentes situaciones.

Tras presentar todas las funciones auxiliares utilizadas en la implementación de los algoritmos, procedemos a detallar la implementación de Q-Learning, Montecarlo y SARSA.

Q-Learning

En primer lugar, nos encontramos con el algoritmo de Q-Learning. Los valores de los hiperparámetros se han seleccionado de manera que optimicen el rendimiento del algoritmo. No obstante, en la siguiente sección se detallarán los experimentos y pruebas realizadas.

Q-Learning:
Inicializar Q **como diccionario vacío**
Parámetros: $\alpha = 0.2$**,** $\gamma = 0.9$**,** $\epsilon = 0.3$**,**
$epocas = 5000$**,** $max\_pasos = 100$
Para cada época:
  Estado aleatorio sin obstáculos
  Para cada paso:
    accion_index ← **escoger_accion(*estado*,** ϵ**)**
    accion ← **nav_acciones[*accion_index*]**
    Aplicar acción, obtener nuevo estado y recompensa
Si *estado* no está en *Q_table* entonces
    Inicializar *Q_table*[*estado*] con ceros
Fin Si
Si *nuevo_estado* no está en *Q_table* entonces
    Inicializar *Q_table*[*nuevo_estado*] con ceros
Fin Si
mejor_accion_nueva ← **max*Q_table*[*nuevo_estado*]**
***Q_table*[*estado*][*accion*]** ← ***Q_table*[*estado*][*accion*] +**
  alpha* * (*recompensa* + *gamma* * *mejor_accion_nueva
  - *Q_table*[*estado*][*accion*])
  Si estado destino o terminal, terminar

Montecarlo

Pasamos ahora con el algoritmo de montecarlo. Los valores de los hiperparámetros se han seleccionado de manera que optimicen el rendimiento del algoritmo. No obstante, en la siguiente sección se detallarán los experimentos y pruebas realizadas.

Montecarlo:
Inicializar *racum* como diccionario de listas vacío
Inicializar *policy* como un diccionario
Inicializar *Q_table* como diccionario vacío
Param: $\gamma = 0.4$**,** $epocas = 5000$**,** $max\_pasos = 100$
Para cada época:
  estado $\leftarrow$ **estado aleatorio sin obstáculos**
  accion_index $\leftarrow$ **escoger_accion(*estado*)**
  accion $\leftarrow$ **nav_acciones[*accion_index*]**
  episodio $\leftarrow$ **genera_secuencia**
  Para cada paso t **en *episodio*:**
    st*, *at*, *rt $\leftarrow$ ***episodio*[t]**
    nuevo_estado $\leftarrow$ **aplica_accion(*st*, *at*)**
    Si *st* no está en *Q_table* entonces
      Inicializar *Q_table*[*st*] con ceros
    Fin Si
    Si *first_visit*(t, *episodio*) entonces
      $U \leftarrow \sum_{i=t}^{len(episodio)} \gamma^{(i-t)} \times r_i$
      index $\leftarrow$ **índice de *at* en *nav_acciones***
      ***racum*[(*st*, *at*)].append(**U**)**
      ***Q_table*[*st*][*index*]** $\leftarrow$ **promedio(*racum*[(*st*, *at*)])**
      ***policy*[*st*]** $\leftarrow$ ***nav_acciones*[argmax(*Q_table*[*st*])]**
    Fin Si

SARSA

A continuación se muestran los detalles de la implementación realizada del algoritmo SARSA. De igual manera que en Q-Learning, el valor de los hiperparámetros se han seleccionado de manera que optimicen el rendimiento del algoritmo.

SARSA:
Inicializar Q_table **como diccionario vacío**
Parámetros: $\alpha = 0.1$**,** $\gamma = 0.9$**,** $\epsilon = 0.3$**,**
$num\_episodios = 100000$**,** $max\_pasos = 100$
Para cada episodio:
  Estado aleatorio sin obstáculos
  accion_index $\leftarrow$ **escoger_accion(*estado*,** ϵ**)**
  accion $\leftarrow$ **nav_acciones[*accion_index*]**
  Para cada paso:
    Aplicar acción, obtener nuevo estado y recompensa
    nueva_accion_index $\leftarrow$
      escoger_accion(*nuevo_estado*, ϵ**)**
    nueva_accion $\leftarrow$ **nav_acciones[*nueva_accion_index*]**
Si *estado* no está en *Q_table* entonces
      Inicializar *Q_table*[*estado*] con ceros
Fin Si
Si *nuevo_estado* no está en *Q_table* entonces
      Inicializar *Q_table*[*nuevo_estado*] con ceros
Fin Si
***Q_table*[*estado*][*accion*]** $\leftarrow$ ***Q_table*[*estado*][*accion*] +**
$\alpha \cdot$ (***recompensa*** $+$
$\gamma \cdot$ ***Q_table***$[$***nuevo_estado***$][$***nueva_accion***$] -$
Q_table$[$***estado***$][$***accion***$]$)
estado $\leftarrow$ ***nuevo_estado***
accion $\leftarrow$ ***nueva_accion***
Si *estado* == destino, terminar

En esta ocasión, al actualizar los valores de la tabla Q, el robot tiene en cuenta la política que está siguiendo. Esto es debido a su enfoque *on-policy*, diferenciándolo de esta manera del algoritmo de Q-Learning.

IV. Pruebas y experimentación

En esta sección, se detallan los experimentos realizados y los resultados obtenidos al aplicar tres algoritmos de aprendizaje por refuerzo (Q-Learning, Montecarlo y SARSA) en el contexto de la planificación de rutas por parte de un robot móvil.

A. Experimentos realizados

Para llevar a cabo los experimentos, se diseñaron tres mapas de prueba, cada uno con un tamaño diferente, un porcentaje distinto de obstáculos y una casilla objetivo única. La posición inicial del robot se selecciona de manera aleatoria en cada experimento. El objetivo es evaluar la capacidad del agente para planificar una ruta segura hacia el objetivo, evitando colisionar con los obstáculos presentes en el entorno. Se espera que el agente se mueva por zonas alejadas de los pasillos para maximizar la seguridad durante la navegación.

A continuación, se presentan los resultados obtenidos de los experimentos realizados con cada uno de los algoritmos de aprendizaje por refuerzo. Se probaron diferentes configuraciones en los hiperparámetros de cada algoritmo para evaluar su rendimiento en una variedad de escenarios.

1) ***Q-Learning****:* Como se describió en la sección de *Implementación*, los hiperparámetros utilizados para ajustar el rendimiento del algoritmo Q-Learning son los siguientes:

Tasa de aprendizaje(α) = 0.2, Factor de descuento(γ) = 0.9, Tasa de exploración(ϵ) = 0.3, Épocas de entrenamiento = 5000, Máximo de pasos por episodio = 100

Se optó por valores bajos de α y ϵ para priorizar la explotación de la mejor acción conocida y permitir que el agente aprenda de manera gradual, evitando posibles desbordamientos en los resultados que podrían surgir de un aprendizaje demasiado rápido. Adicionalmente, probamos con un valor alto de γ para que el agente priorizara las

recompensas futuras sobre las inmediatas.

A continuación, se presentan los resultados obtenidos en cada uno de los mapas de prueba:

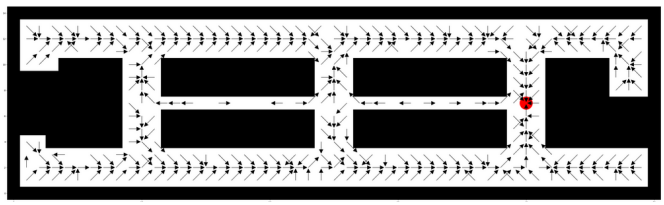

Fig. 3. Resultados Q-Learning mapa 1

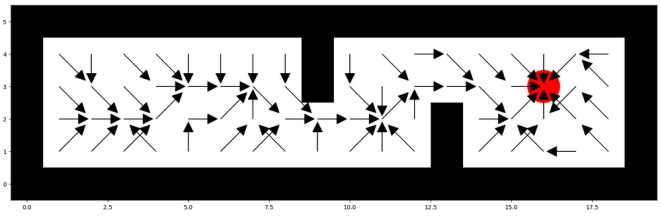

Fig. 4. Resultados Q-Learning mapa 2

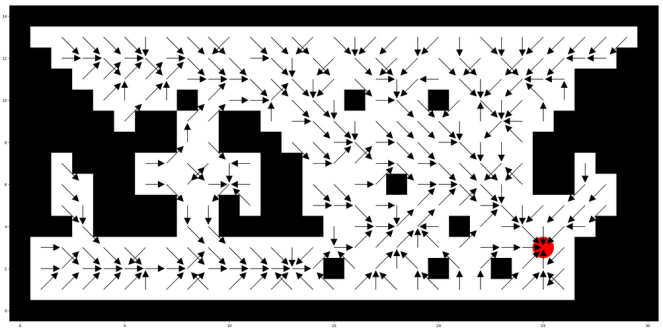

Fig. 5. Resultados Q-Learning mapa 3

Las imágenes de las rutas diseñadas por el robot en cada uno de los tres mapas muestran resultados muy satisfactorios por varias razones. En primer lugar, el robot ha sido capaz de generar rutas cortas y directas, minimizando el tiempo y los recursos necesarios para alcanzar el objetivo. Además, el robot ha demostrado una habilidad consistente para evitar colisiones con los obstáculos presentes en los mapas, incluso en aquellos con una mayor densidad de obstáculos. Finalmente, se destaca la robustez del algoritmo, ya que los resultados han sido satisfactorios en todos los mapas, a pesar de las variaciones en el tamaño, la densidad de obstáculos y la posición del estado objetivo.

Por otro lado, tras realizar numerosas pruebas con variaciones en el valor de los hiperparámetros, llegamos a la conclusión de que las modificaciones en los valores de la tasa de aprendizaje, el factor de descuento y la tasa de exploración no producían cambios significativos en los resultados, e incluso en algunos casos se observaron pequeñas mejoras. No obstante, se pudo verificar que al reducir significativamente el *número de épocas de entrenamiento* del algoritmo, los resultados fueron mucho peores. Por ejemplo, al establecer el número de épocas en 1000, estos son los resultados obtenidos en cada uno de los mapas:

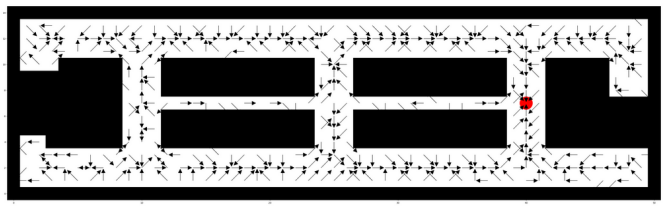

Fig. 6. Resultados Q-Learning mapa 1 tras modificar el número de épocas

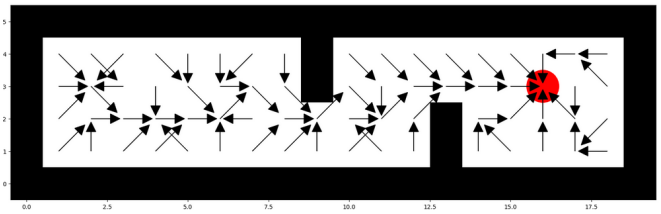

Fig. 7. Resultados Q-Learning mapa 2 tras modificar el número de épocas

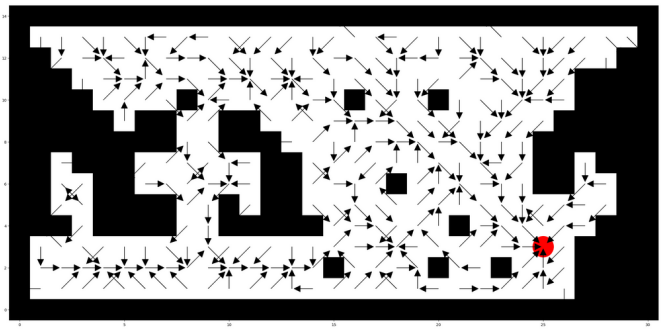

Fig. 8. Resultados Q-Learning mapa 3 tras modificar el número de épocas

Como se puede apreciar, los resultados no son tan satisfactorios, ya que el robot colisiona con los obstáculos en ciertas ocasiones y no suele seleccionar la ruta más eficiente para llegar a la casilla objetivo. Este empeoramiento de los resultados es más notorio en los mapas 1 y 3, lo cual tiene sentido porque, al ser mapas más grandes, requieren un mayor número de épocas de entrenamiento para que el agente aprenda la política óptima.

*2) **SARSA**:* En la implementación del algoritmo SARSA, los valores de los hiperparámetros se ajustaron según el mapa. En el mapa 2 (el más simple de todos), aumentamos la tasa de aprendizaje a 0.5 para que el agente pudiera aprender rápidamente sin una excesiva preocupación por desbordamientos en los valores Q. Además, reducimos el

factor de descuento a 0.5, ya que en un mapa más pequeño las recompensas futuras no son tan importantes como las inmediatas, dado que el objetivo está más cerca. Mantenemos el valor de epsilon muy bajo, en 0.1, pues la simpleza del mapa permite que este pueda ser explorado completamente de manera rápida.

En los otros dos mapas, al ser entornos más complejos, disminuimos la tasa de aprendizaje y aumentamos tanto el factor de descuento como la tasa de exploración. Estas modificaciones permiten que el agente explore más y valore mejor las recompensas a largo plazo, necesarios en escenarios más complicados.

Finalmente, es importante destacar que el número de episodios de entrenamiento es significativamente mayor que en Q-Learning. Esto se debe a que SARSA, al tener un enfoque *on-policy*, normalmente converge más lentamente hacia una política óptima, especialmente cuando la estrategia de exploración no se maneja adecuadamente [4]. Por ello, consideramos necesario aumentar el número de iteraciones del algoritmo.

A continuación, se presentan los resultados obtenidos en cada uno de los mapas de prueba:

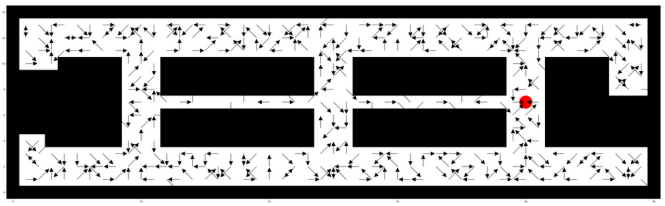

Fig. 9. Resultados SARSA mapa 1

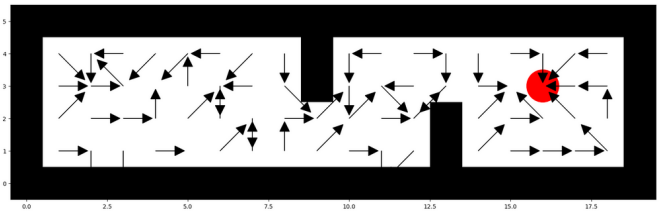

Fig. 10. Resultados SARSA mapa 2

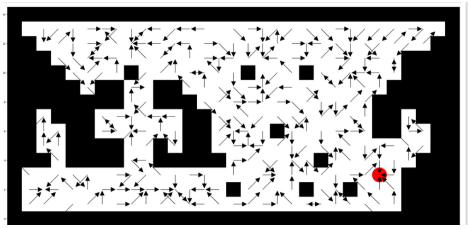

Fig. 11. Resultados SARSA mapa 3

*3) **Montecarlo**:* Los hiperparámetros utilizados para la ejecución del algoritmo de montecarlo fueron los siguientes: Factor de descuento(γ) = 0.3, Épocas de entrenamiento = 5000, Máximo de pasos por episodio = 100

A diferencia de Q-Learning, se optó por un factor de descuento bajo.

A continuación, se presentan los resultados obtenidos en cada uno de los mapas de prueba:

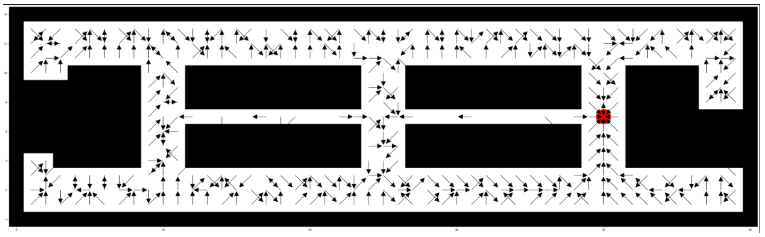

Fig. 12. Resultados Montecarlo mapa 1

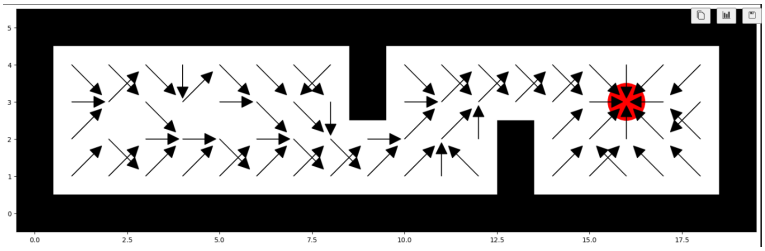

Fig. 13. Resultados Montecarlo mapa 2

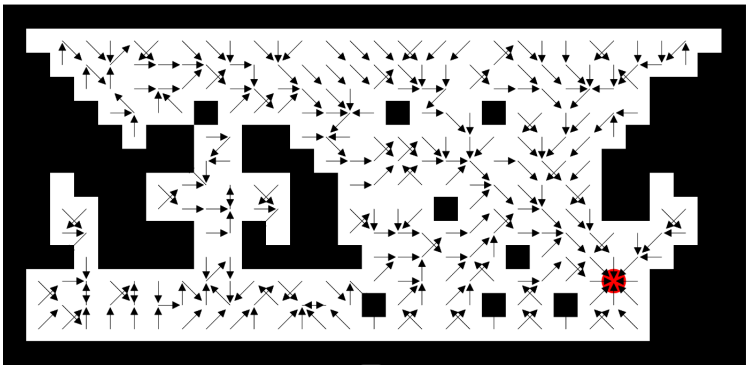

Fig. 14. Resultados Montecarlo mapa 3

Tras implementar y ejecutar el algoritmo Montecarlo con los hiperparámetros descritos anteriormente, los resultados obtenidos fueron bastante satisfactorios, comparativamente mejores que los obetenidos en SARSA.

Al igual que en Q-Learning,el valor del hiperparámetro correspondiente al número de épocas de entrenamiento es sensible a modificaciones. Al reducirse considerablemente las épocas de entrenamiento se obtuvieron resultados

notablemente peores.

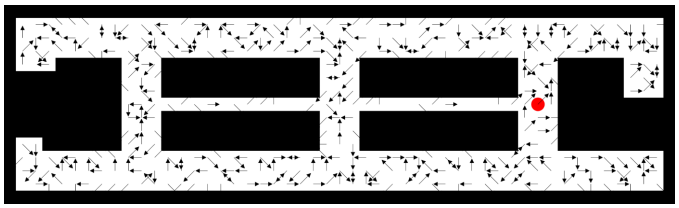

Fig. 15. Resultados Montecarlo mapa 1 tras modificar el número de épocas

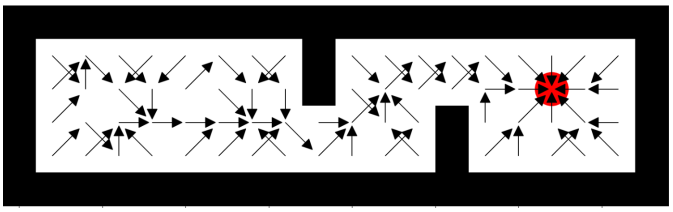

Fig. 16. Resultados Montecarlo mapa 2 tras modificar el número de épocas

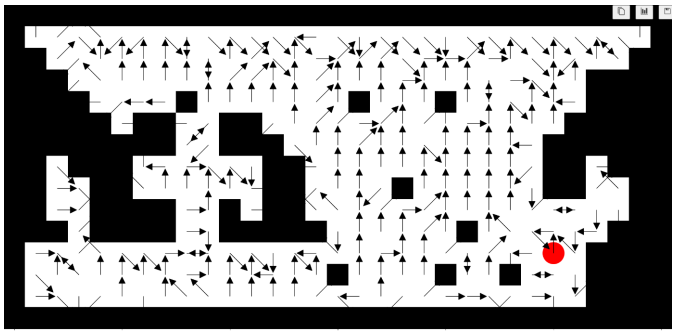

Fig. 17. Resultados Montecarlo mapa 3 tras modificar el número de épocas

Como se puede apreciar, los resultados no son tan satisfactorios, ya que el robot no suele seleccionar la ruta más eficiente para llegar a la casilla objetivo. Este empeoramiento de los resultados es más notorio en los mapas 1 y 3, lo cual tiene sentido porque, al ser mapas más grandes, requieren un mayor número de épocas de entrenamiento para que el agente aprenda la política óptima. En el mapa 2, no se notaron cambios significativos.

Tras realizar pruebas durante un extenso periodo de tiempo, destacamos una observación interesante es que, a diferencia de Q-Learning, se obtienen mejores resultados en los mapas dados con un factor de descuento pequeño. Esto puede ser debido a que en Montecarlo se aprende de episodios completos mientras que en Q-Learning se aprende en cada paso del episodio. A continuación se muestran algunas pruebas Factor de descuento(γ) = 0.9

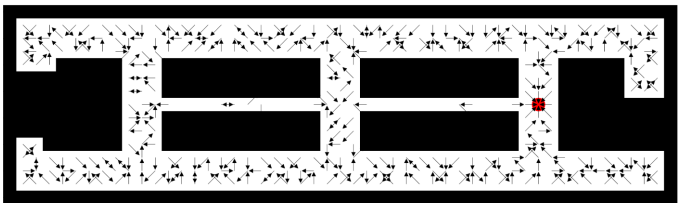

Fig. 18. Resultados Montecarlo mapa 1 tras modificar el factor de descuento

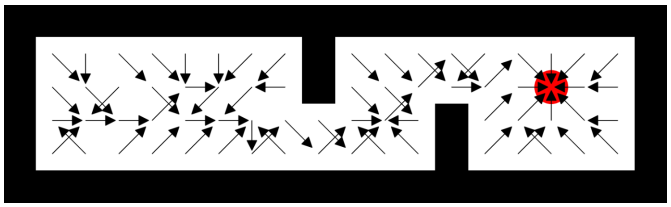

Fig. 19. Resultados Montecarlo mapa 2 tras modificar el factor de descuento

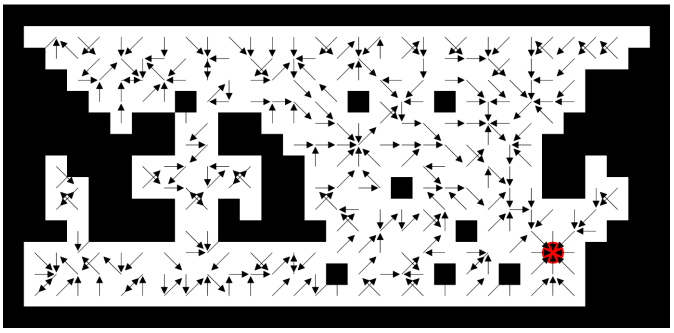

Fig. 20. Resultados Montecarlo mapa 3 tras modificar el factor de descuento

Como se puede apreciar, los resultados siguen siendo satisfactorios, pero peores que con un factor de descuento bajo ya que a pesar que el robot suele seleccionar la ruta más eficiente, en ciertos estados selecciona una acción que difiere de la más óptima para llegar a la casilla objetivo.

Una observación interesante es que, a diferencia de Q-Learning, se obtienen mejores resultados en los mapas dados con un factor de descuento pequeño. Esto puede ser debido a que en Montecarlo se aprende de episodios completos mientras que en Q-Learning se aprende en cada paso del episodio. A continuación se muestran algunas pruebas Factor de descuento(γ) = 0.9

V. Conclusiones

En este trabajo, se ha llevado a cabo una comparación exhaustiva de tres algoritmos de aprendizaje por refuerzo (Q-Learning, Montecarlo y SARSA) aplicados a la planificación de rutas para un robot móvil en un entorno con obstáculos.

Se diseñaron y probaron tres mapas diferentes, variando en tamaño y densidad de obstáculos, para evaluar la capacidad de cada algoritmo para generar rutas seguras y eficientes. Los resultados obtenidos se analizaron en términos de la longitud de las rutas generadas, la frecuencia de colisiones y la convergencia hacia políticas óptimas.

Los experimentos demostraron que Q-Learning y Montecarlo proporcionaron resultados superiores en comparación con SARSA. Q-Learning generó rutas más cortas y directas, y mostró una alta capacidad para evitar colisiones incluso en entornos complejos. Montecarlo también tuvo un buen rendimiento, aunque su convergencia fue más lenta en comparación con Q-Learning. Por otro lado, SARSA presentó rutas más largas y menos directas, con una mayor frecuencia de colisiones y una convergencia más lenta, incluso con un mayor número de episodios de entrenamiento. Además, los resultados de SARSA fueron más sensibles a los valores de los hiperparámetros, mostrando una caída significativa en el rendimiento cuando se alejaban de los valores óptimos.

Las conclusiones principales de este trabajo indican que Q-Learning es el algoritmo más robusto y eficiente para la tarea de planificación de rutas en entornos con obstáculos, seguido de Montecarlo. SARSA, aunque conceptualmente sólido, mostró limitaciones en términos de eficiencia y robustez en comparación con los otros dos algoritmos. La sensibilidad de SARSA a los hiperparámetros sugiere que requiere una mayor afinación para obtener resultados competitivos.

Como ideas de mejora y trabajo futuro, se podría investigar la combinación de estos algoritmos con técnicas de aprendizaje profundo, como Deep Q-Learning, para manejar entornos más complejos y de mayor dimensionalidad. Además, se podría explorar la integración de mecanismos de aprendizaje transferencial para permitir al robot aplicar conocimientos previos a nuevos entornos, mejorando así su capacidad de generalización y adaptabilidad.

REFERENCIAS

[1] Richard S. Sutton y Andrew G. Barto. Reinforcement Learning: An Introduction. MIT Press, 2018.
[2] Russell, Stuart and Norvig, Peter. Artificial Intelligence: A Modern Approach (2010). Chapter 21.
[3] https://www.aprendemachinelearning.com/aprendizaje-por-refuerzo/
[4] https://www.javatpoint.com/sarsa-reinforcement-learning
[5] https://es.wikipedia.org/wiki/Q-learning
[6] https://www.codificandobits.com/curso/aprendizaje-por-refuerzo-nivel-basico/2-ejemplos-reales-aplicacion-aprendizaje-por-refuerzo/
[7] https://aws.amazon.com/es/what-is/reinforcement-learning/