Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Deep Reinforcement Learning para la cobertura y patrullaje informativo en escenarios hidrológicos parcialmente observables

Autor: Dame Seck Diop

Tutor: Daniel Gutiérrez Reina, Samuel Yanes Luis

Dpto. Ingeniería Eléctrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Deep Reinforcement Learning para la cobertura y patrullaje informativo en escenarios hidrológicos parcialmente observables

Autor:
Dame Seck Diop

Tutor:
Daniel Gutiérrez Reina, Samuel Yanes Luis
Profesor Titular

Dpto. Ingeniería Eléctrónica
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2022

Trabajo Fin de Grado: Deep Reinforcement Learning para la cobertura y patrullaje informativo en escenarios hidrológicos parcialmente observables

Autor: Dame Seck Diop
Tutor: Daniel Gutiérrez Reina, Samuel Yanes Luis

El tribunal nombrado para juzgar el trabajo arriba indicado, compuesto por los siguientes profesores:

Presidente:

Vocal/es:

Secretario:

acuerdan otorgarle la calificación de:

El Secretario del Tribunal

Fecha:

Agradecimientos

No puedo estar más agradecido con todas las personas que me han acompañado en esta gran etapa de mi vida, la cuál estoy dando por terminada con este proyecto.

En primer lugar, quiero agradecer a mi familia, en especial a mis padres, por la paciencia y la confianza que siempre han tenido en mi. A mi tío, por el gran apoyo que me ha dado cuando mas lo necesitaba.

A mis amigos, tanto los anteriores como los conocidos en la carrera, que han hecho estos años más amenos y me han dado fuerza para conseguir todo lo que me proponga.

A mis abuelos, tíos y primos, por creer siempre en mí y ser una parte imprescindible en mi vida.

Y por supesto, a mis tutores, Samuel Yanes Luis y Daniel Gutiérrez Reina por confiar en mi para este trabajo y guiarme en este proceso.

Dame Seck Diop
Escuela Técnica Superior de Ingeniería de Sevilla.

Sevilla, 2022

Resumen

En este trabajo se lleva a cabo la supervisión y monitorización de la contaminación del lago Ypacaraí mediante vehículos autónomos de superficies (ASV). Se pretende diseñar un planificador de alto nivel basado en técnicas de *Deep Reinforcement Learning* capaz de resolver un problema de patrullaje informativo en escenarios parcialmente observables.

En primer lugar, se han sintonizado los parámetros de los dos escenarios proporcionados: completamente y parcialmente observable. Este escenario, basado en la geometría espacial del lago, contiene zonas de mayor importancia que representan las zonas más contaminadas del lago. Se ha implementado un algoritmo Double-DQN en estos dos entornos y se ha estudiado como este algoritmo ha resuelto el problema del patrullaje informativo y como ha evolucionado la política del agente durante el entrenamiento.

Para mejorar los resultados obtenidos en los experimentos del anterior algoritmo, especialmente en el escenario parcialmente observable, se ha rediseñado la función de recompensa para que el agente se atreva a visitar lugares no descubiertos y desconocidos por completo. Se ha vuelto a entrenar al agente Double- DQN con esta nueva ley de recompensa y se han realizado varios experimentos en donde se estudia como influye ésta en el comportamiento del agente.

Por último, se ha comparado nuestro agente con varios algoritmos basados en heurísticas usados habitualmente en casos de planificación de rutas de cobertura no homogénea.

Abstract

In this work, the supervision and monitoring of the contamination of Lake Ypacaraí using autonomous surface vehicles (ASV) is reallised. The aim is to design a high-level planner based on Deep Reinforcement Learning techniques capable of solving an informative patrolling problem in partially observable scenarios.

First, the parameters of the two scenarios provided have been tuned: fully and partially observable. This scenario, based on the spatial geometry of the lake, contains areas of higher importance representing the most polluted areas of the lake. A Double-DQN algorithm has been implemented in these two environments and it has been studied how this algorithm has solved the informative patrolling problem and how the agent's policy has evolved during training.

In order to improve the results obtained in the experiments with the previous algorithm, especially in the partially observable scenario, the reward function has been redesigned so that the agent dares to visit undiscovered and completely unknown places. The Double-DQN agent has been retrained with this new reward law and several experiments have been performed to study how the reward law influences the agent's behaviour.

Finally, our agent has been compared with several heuristic-based algorithms commonly used in non-homogeneous coverage path planning cases.

Índice

Resumen III
Abstract V

1 Introducción 1
1.1 Antecedentes 1
1.2 Uso de ASVs 3
1.3 Objetivos 6

2 Estado del arte 7
2.1 Inteligencia Artificial 7
2.1.1 Tipos de aprendizaje 7
Aprendizaje profundo 8
2.2 Robótica 8
2.2.1 Inteligencia Artificial aplicada a la planificación de rutas y movimientos 9
2.3 Path Planning 10
2.3.1 Algoritmos clásicos de Path Planning (PP) 11
2.4 Monitorización y patrullaje 12

3 Planteamiento del problema 17
3.1 Problema del vigilante 17
3.2 Descripción del escenario 19

4 Metodologia 21
4.1 Reinforcement Learning 21
4.1.1 Definiciones basicas 21
4.1.2 Proceso de decisión de Markov 23
4.1.3 Funciones de valor y la ecuación de Bellman 24
Funciones de valor 24
Ecuación de Bellman 24
Ecuación de optimalidad de Bellman 24
4.1.4 Políticas épsilon-greedy 25
4.1.5 Aprendizaje por diferencia temporal: Q-Learning 26
Q-Learning 26
4.2 Redes neuronales artificiales(RNAs) 27
4.2.1 Neurona artificial 27
4.2.2 Perceptrón multicapa 28

4.2.3 Redes Neuronales Convolucionales 30
4.2.4 Retropropagación 32
4.3 Deep Reinforcement Learning 32
4.3.1 Deep Q-Learning 32
Experience Replay 33
Target Network 34
Dueling Network 35
4.4 Ley de Recompensa 36
4.4.1 Recompensa a la exploración 39
4.5 Representación del estado 40
4.5.1 Estado del Entorno completamente observable 40
4.5.2 Estado del Entorno parcialmente observable 42
4.6 Nuestro agente 43
4.6.1 Red neuronal del agente 43
4.6.2 Programación del agente 44
4.6.3 Colisiones 45

5 Resultados 47
5.1 Generación de los mapas 48
5.2 Métricas de funcionamiento 49
5.3 Escenario Completamente Observable 51
5.3.1 Evolución de las métricas 51
5.3.2 Evolución de la política 54
5.3.3 Conclusiones 57
5.4 Escenario Parcialmente Observable 57
5.4.1 Evolución de las métricas 58
5.4.2 Evolución de la política 61
5.4.3 Exploración del agente 61
5.4.4 Conclusiones 63
5.5 Escenario parcialmente observable con recompensa de cobertura 64
5.5.1 Experimentos 64
5.5.2 Conclusiones 67
5.6 Comparación con otros algoritmos basados en heurísticas 70
5.6.1 Algoritmo cortacésped 70
5.6.2 Algoritmo de dirección aleatoria 70
5.6.3 Algoritmo de movimientos aleatorios 71
5.6.4 Experimentos 71

6 Discusión y conclusiones 73
6.1 Conclusiones 73
6.2 Líneas futuras 74

Índice de Figuras 77
Índice de Tablas 81
Bibliografía 83
Glosario 87

1 Introducción

No se aprecia el valor del agua hasta que se seca el pozo.

PROVERBIO INGLÉS

1.1 Antecedentes

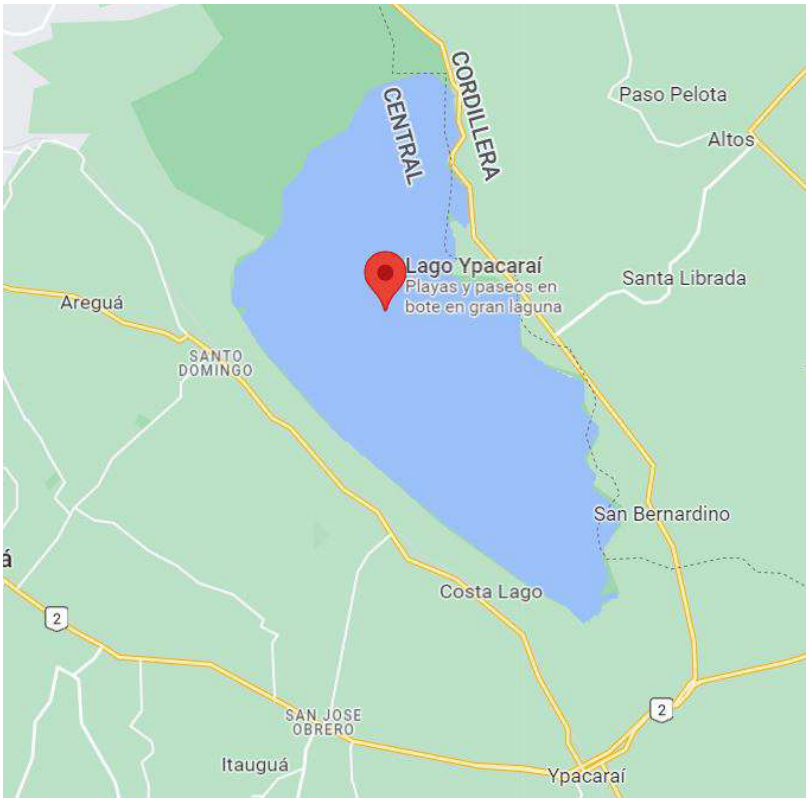


Figura 1.1 Vista cenital del lago Ypacaraí.

El lago Ypacaraí es el mayor cuerpo de agua de Paraguay con más de $60km^2$ de superficie navegable y $3m$ de profundidad. Está situado entre las ciudades de San Bernardino (en el este), Areguá (en el oeste) y Ypacaraí (en el sur) como principal fuente de abastecimiento de agua en

la zona. A lo largo de los años, su importancia se ha hecho mayor con el turismo ya que ha sido como lago recreativo para que la gente se bañe y para la navegación de pequeñas embarcaciones. Su importancia también está relacionada con la vida natural desarrollada en los humedales de la cuenca que rodea al lago.

No obstante, en los últimos 40 años, la continua expansión de la agricultura en los alrededores del lago, la falta de de sistemas de alcantarillado en las ciudades cercanas y los vertidos de residuos de las industrias situadas en la orilla, entre otros factores, han provocado en el lago un proceso de eutrofización anormal [25].

Esta eutrofización artificial (enriquecimiento no natural de las aguas) provoca el florecimiento de cianobacterias(Figura 1.2). Estos son organismos que consumen muy rápidamente el oxígeno disuelto propiciando la anoxia del medio. Adicionalmente, desde 2012, ha estado en una forma agresiva con varias floraciones de algas verde-azuladas. Estas colonias de algas vienen acompañadas de olores fétidos y la generación de toxinas como la microsis, perjudicial para la fauna y los humanos, incluso mortales en algunas ocasiones [2].

Los residuos humanos de las ciudades junto con los provenientes de las industrias que rodean la zona, han causado un aumento considerable de colonias de cianobacterias en el agua (de ahí su característico color verde esmeralda). Los vertidos, ricos en nitrógeno, fósforo y metales constituyen un catalizador para las colonias de bacterias y suciedad en toda la superficie navegable y en las zonas de humedales que conforman el lago. Este efecto compromete no solo al ecosistema animal de la cuenca sino también la salubridad del suministro de agua de las ciudades anexas, cuya única fuente de recursos hídricos está en el propio Ypacaraí.

Figura 1.2 Efecto de las cianobacterias.

Hay que tener en cuenta que la naturaleza de los brotes de cianobacterias es dinámica y bastante caótica, lo que dificulta la predicción de dónde y y cuándo surgirán [3]. Además el comportamiento de la floración no es estático y cambia su tamaño con el tiempo, lo que hace que la tarea de medir la calidad del agua sea un proceso arduo.

Como primera medida de prevención de enfermedades humanas y animales, es necesaria una tarea de monitorización continua del lago. Cabe destacar que la contaminación se distribuye de forma irregular por la superficie del Ypacaraí. En las zonas cercanas a las ciudades de Areguá y San Bernardino, donde la ausencia de infraestructura de alcantarillado y filtrado séptico supone un repunte de vertidos y de bacterias, hay especial concentración. Así como en zonas portuarias recreativas o zonas industriales cercanas como el Puerto de San Blas, en la parte sur del lago.

El gobierno de Paraguay ha intentado combatir este problema y ha dedicado numerosos esfuerzos técnicos y de investigación:

- Ha instalado 3 estaciones de monitorización del bioma
- Se realizaron 12 estudios de campo para el análisis de los brotes agresivos surgidos entre el 2015 y el 2016.
- Desde la perspectiva de la infraestructura: construcción de desvíos de los ríos que desembocan en el Ypacaraí, cordones flotante para la contención de las algas de superficie, mejora del saneamiento de residuos de las industrias colindantes..

Este problema de contaminación es un reto interdisciplinar en el que diversos perfiles de ingeniería y científicos deben trabajar juntos. Las soluciones deben venir de la ayuda de técnicas multidisciplinares que no sólo se centren en la reducción de los vertidos y las inversiones en infraestructuras, sino también en el seguimiento y estudio del estado actual del lago para encontrar una forma eficaz de revertir la situación.

1.2 Uso de ASVs

Es vital hacer un seguimiento eficiente del estado del lago para tener una imagen actualizada del estado biológico de las floraciones de algas. Este mapa de contaminación permite analizar el rendimiento de las medidas medioambientales adoptadas por las autoridades e investigadores. Sin embargo, la tarea de seguimiento manual requiere un gran esfuerzo y recursos humanos, ya que exige constantes desplazamientos desde la orilla hasta las principales floraciones con embarcaciones a motor y un muestreo manual de las aguas.

Figura 1.3 ASV realizando la monitorización en la superficie del lago.

En esta línea, un proyecto de cooperación de la Universidad de Sevilla (grupo ACETI del Departamento de Ingeniería Electrónica de la misma universidad) y la Universidad Nacional de Asunción (FIUNA), denominado "Autonomous Surface Vehicle for the study of water quality in Lakes" ha planteado la utilización de Vehículos Autónomos de Superficie, ASV de sus siglas en inglés, para sustituir las inspecciones humanas. Estos vehículos, que son similares a los barcos

pero más pequeños y sin tripulación que los maneje, se han utilizado ampliamente para tareas de vigilancia ambiental [2]. Presentan muchas ventajas en comparación con los barcos normales: como ser más baratos en gastos de combustible, más respetuosos con el medio ambiente debido a el uso de motores eléctricos que pueden conectarse a una batería y a un sistema de alimentación por paneles solares, y por supuesto, la capacidad de operar sin tripulación a bordo.

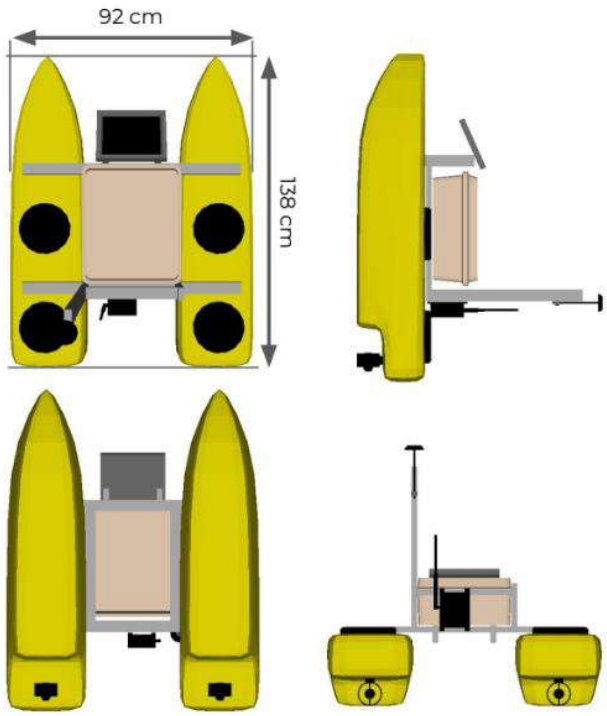


Figura 1.4 Diseño del Cormorán-II.

El último prototipo de ASV que se diseñó para esta tarea es de tipo catamarán. Tiene dos motores que permiten el manejo en una configuración diferencial y un tamaño de 1.38 m de eslora y 0.92 m de manga. También va equipado con actuadores y sensores que permite su manejo y caracterización del entorno hídrico. Los motores (BLDC) tienen una capacidad máxima de 32 amperios y ejercen, a máxima potencia, una fuerza de 67N cada uno. Además, están controlados con un variador de frecuencia (ESC) de 30 Amperios. La velocidad máxima de seguridad del vehículo es de 2 m/s. En cuanto al sensore, este de tipo *Smart Water*(Gama de productos para la monitorización del agua) que consta de un sistema de lectura multi-paramétrica. Cuenta con un sistema de alimentación independiente de una duración mayor de 1 día. Asimismo, se comunica con el vehículo mediante dataframes a través de un puerto USB. En un diseño paralelo, el sistema cuenta con una bomba de elevación por sifón para tomar muestras aisladas. Adicionalmente, se ha diseñado una placa base (PCB) para el conexionado de todos los componentes. La PCB permite la conexión flexible de los dispositivos de navegación y potencia. Cuenta con puertos extras para alimentar nuevo hardware o cambiar el existente.

Se han fabricado varios drones acuáticos de pequeño tamaño y mismas características, de los cuales, dos están en las instalaciones de la Escuela Técnica Superior de Ingeniería de Sevilla.

En resumen, el prototipo servirá para navegar por el lago Ypacaraí mientras se toman muestras de

Tabla 1.1 Sistemas y subsistemas del vehículo.

Módulo	Componente
Sistema de navegación	• Un controlador de bajo nivel: Navio2 • Un out-board companion: Jetson Xavier • Con un sistema GPS ultrapreciso: EMLID
Sistema de distribución de potencia	• Configuración diferencial. • Alimentación redundante. • Una batería de larga duración y capacidad.
Sistema de sensores de calidad del agua	• Sistema de monitorización independiente. • Múltiples sensores.
Sistema de visión	• Cámara estereoscópica de última generación. • Inteligencia Artificial para la detección de obstáculos.
Sistema de Interfaz	• Sistema on-cloud para el control de usuario de la flota. • Sistema de recolección y representación de datos abierta al mundo.

la calidad del agua y su nivel de contaminación. El ASV tiene una arquitectura hardware-software que proporcionará la capacidad de seguir una secuencia de waypoints (previamente calculados e interpolados por un generador de trayectorias, como un interpolador spline).Así, una vez que la arquitectura de bajo nivel está lista, la tarea de exploración se realiza seleccionando los waypoints que el ASV debe visitar para obtener la mayor eficiencia.

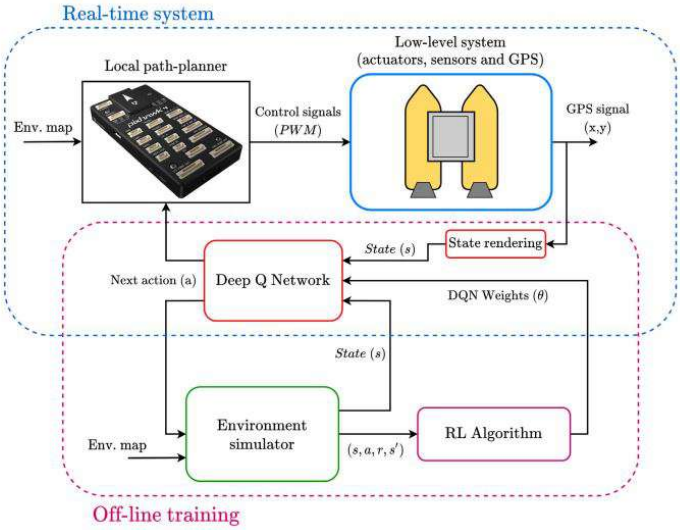


Figura 1.5 Diagrama del control ASV en el caso real (arriba) y el bucle de aprendizaje de refuerzo (abajo) [26].

1.3 Objetivos

En este trabajo el planificador de alto nivel calcula el siguiente punto a visitar dado el estado actual. Estamos ante un problema de decisión secuencial, esto es, en cada instante, el agente recibe información limitada del ambiente, y debe tomar la mejor decisión de acuerdo a sus objetivos. Dependiendo de los objetivos de la misión, se pueden tener en cuenta una gran cantidad de caminos posibles. En nuestro caso, queremos adquirir la mayor información posible del nivel de contaminación del lago.

En trabajos previos, como [26], se expuso este problema y se consiguió abordarlo bajo dos perspectivas:

- Problema de patrullaje con importancia homogénea, que supone que cada zona del lago es igualmente importante; por lo tanto, el ASV debe visitar todos los puntos de paso con la misma frecuencia.
- Problema de patrullaje con una importancia no homogénea, considerando una importancia relativa diferente para cada waypoint del lago.

Los autores de [26] han sido capaz de devolver resultados positivos en la cobertura homogénea de la superficie navegable del lago (con mejoras del 30 % en el tiempo medio de espera de cada zona a ser visitada) respecto de otros métodos clásicos como el algoritmo de cortacésped o de búsqueda aleatoria. Para el caso de la exploración y cobertura dada una importancia no homogénea del lago, tenemos que se mejora la recompensa obtenida del orden del 20 % respecto del método cortacésped.

Además de los resultados se tiene un algoritmo capaz de aprender para cualquier entorno lacustre independientemente de su morfología, lo que supone un punto a favor de las técnicas empleadas.

En este proyecto se intenta seguir el trabajo de [26] y obtener un algoritmo con mayor resolución, así como abordar el problema desde un punto de vista más realista. Hay que tener en cuenta que no se conocerá toda la información del lago en todo momento, por ello, el tema principal de este trabajo se centra en el paradigma de la no-observabilidad del lago. Es necesario una política que decida explorar el lago o visitar las zonas de interés incluso antes de saber cómo se distribuye la contaminación (interés).

Los objetivos principales son los siguientes:

1. Configuración de un entorno fiel a la representación del problema.
2. Estudio y ajuste de parámetros de un algoritmo de exploración adaptativo basado en *Deep Reinforcement Learning* que consiga mejores resultados.
3. Comparación del anterior algoritmo frente a otros métodos de exploración.
4. Estudiar principalmente el caso de un Escenario Parcialmente Observable, donde el agente comienza a aprender sin ninguna información previa de la contaminación del lago.
5. Estudiar la disposición del agente, en el caso del Escenario Parcialmente Observabl, a explorar las zonas que todavía no ha descubierto.

2 Estado del arte

2.1 Inteligencia Artificial

La Inteligencia Artificial (IA) es una disciplina de ciencias de computación que intenta replicar y desarrollar la inteligencia y sus procesos implícitos a través de computadoras [51]. Existen varios elementos que componen la ciencia de la IA dentro de las cuáles se pueden distinguir tres grandes ramas:Lógica difusa, Redes Neuronales Artificiales, Algoritmos genéticos

Los objetivos centrales de la IA son, en primer lugar, estudiar y entender mejor el funcionamiento de la inteligencia y humana, y en segundo lugar, desarrollar programas capaces de automatizar el comportamiento inteligente [18].

Uno de los paradigmas principales de investigación en IA es el paradigma conexionista, surgido a finales de la década de los 80. La IA conexionista sostiene que los fenómenos mentales pueden ser descritos por redes de unidades sencillas y a menudo uniformes que se interconectan. La forma de las conexiones y de las unidades puede variar de un modelo a otro. Estos enfoques se han hecho muy populares recientemente, siendo las redes neuronales las más destacables. Actualmente se utilizan en varios trabajos de aprendizaje automático (Machine Learning) y han resultado ser extremadamente buenos para llevar a cabo determinados tipos de tareas inteligentes, como el reconocimiento de patrones, la categorización y la coordinación.

2.1.1 Tipos de aprendizaje

El Machine Learning es el campo de la Inteligencia Artificial que permite a las máquinas elaborar predicciones. Dependiendo del esquema de aprendizaje, se pueden distinguir varios tipos de aprendizaje en el Machine Learning:

- Aprendizaje supervisado: En este tipo de aprendizaje, los datos de entrenamiento contienen la solución deseada. Es usado normalmente para clasificar objetos, personas etc. Por ejemplo, si se desean clasificar frutas, los datos de entrenamiento contendrán las imágenes de las frutas y cada una junto con la etiqueta que la identifica(nombre de la fruta). Cada vez que se procesa una entrada y se produce una salida, se compara esa salida con la etiqueta y se modifican los pesos de la red dependiendo del error cometido. Si el entrenamiento de la red es óptimo, esta debería acertar las etiquetas de datos de entrada no vistos anteriormente.
- Aprendizaje no supervisado: En este esquema de aprendizaje los datos de entrenamiento no contienen las etiquetas que las identifica. Por lo tanto, la red neuronal intenta determinar la salida correcta por sí mismo y haciendo uso de información interna. En esta situación, la red se ve obligada a aprender rasgos y características de los datos de entrada.
- Aprendizaje por refuerzo: en inglés Reinforcement Learning (RL), es un caso especial de aprendizaje no supervisado en el que los pesos de la red se modifican en función de la

Figura 2.1 Sophia, robot humanoide con Inteligencia Artificial desarrollado por Hanson Robotics.

recompensa obtenida por producir las salidas. Por lo tanto, aprenderá por si mismo en un entorno desconocido gracias a las recompensas y penalizaciones que obtiene de sus acciones. El agente debe actuar y crear la mejor estrategia posible para obtener la mayor recompensa en tiempo y forma [16].

Aprendizaje profundo

Por otro lado , el aprendizaje profundo (en inglés Deep Learning (DL)) es una colección de algoritmos usados en redes neuronales para resolver tareas de aprendizaje automático (Machine Learning), ya sea aprendizaje supervisado, aprendizaje no supervisado o aprendizaje por refuerzo. Los algoritmos de Deep Learning se basan en redes neuronales artificiales(redes neuronales profundas) capaces de aprender múltiples niveles de representación que corresponden con diferentes niveles de abstracción [50].
El uso de DL es muy popular debido a la manera en la que representan los datos. Las redes neuronales profundas tienen varias capas (de ahí lo de "profundas"), lo que permite que el modelo aprenda representaciones de los datos de entrada en capas. Esta representación en capas posibilita que los datos complejos se representen como la combinación de componentes más elementales, y esos componentes pueden ser divididos en componentes aún más simples, y así sucesivamente, hasta llegar a unidades atómicas. En consecuencia, los componentes de más alto nivel se derivan de los componentes de nivel inferior para formar una representación jerárquica.
Varias arquitecturas de aprendizaje profundo, como redes neuronales profundas y redes neuronales profundas convolucionales, han conseguido avances en aplicaciones como: clasificación de imágenes, visión por computador, reconocimiento automático del habla, reconocimiento de señales de audio y música, transcripción de escritura a mano etc.

2.2 Robótica

La robótica es la rama de la ingeniería mecánica, de la ingeniería electrónica y de las ciencias de la computación, que se ocupa del diseño, construcción, operación, estructura, manufactura y aplicación de los robots [52].

Un robot es una máquina electromecánica con sensores y actuadores que puede desplazarse (en su totalidad o una parte de su construcción) en su entorno y realizar las tareas previstas de forma autónoma o semiautónoma [18]. Se emplean para sustituir algunas labores humanas puesto que en muchas aplicaciones pueden realizar tareas de forma más eficaz y eficiente, y disminuir el riesgo de la seguridad humana. Adicionalmente, pueden alcanzar lugares inaccesibles para el ser humano, aumentando la influencia humana en estos sitios.

Cuando un robot móvil puede operar con cierto grado de autonomía y por lo tanto sin intervención humana, se dice que es un robot autónomo. Este tipo de robots pueden comprender y navegar por un entorno de forma independiente. Para ello recaban información de su entorno mediante sensores como cámaras, detectores de proximidad etc. Luego procesan los datos captados y evalúan la situación para realizar una acción especifica. El comportamiento de la gran mayoría de robots autónomos es el resultado de un aprendizaje realizado mediante Machine Learning (ML), que les ha servido para planificar rutas e interpretar la mejor opción, sin depender de vías predefinidas.

El patrullaje es la vigilancia continua de un entorno para llevar a cabo una misión o mantener el orden. Es una actividad importante para las aplicaciones relacionadas con seguridad y vigilancia.

Los vehículos autónomos no tripulados equipados con sensores han sufrido un gran avance en la capacidad de entender y predecir lo que ocurre en su entorno. Esto les hace grandes candidatos para mantener una conciencia situacional precisa y actualizada de la situación de una cierta zona, especialmente en entornos sujetos a cambio continuo. Asimismo, el uso de vehículos autónomos reduce la necesidad de exponer a las personas a entornos hostiles, intransitables o contaminados.

Es por ello que el patrullaje de entornos mediante robots móviles autónomos ha recibido una atención creciente en los últimos años. Varios proyectos de investigación [2, 25, 26, 21] se centran principalmente en el desarrollo de estrategias de patrullaje eficaces. Para funcionar bien, los vehículos tienen que patrullar su entorno de forma continua y visitando repetidamente algunos puntos de interés para vigilarlos.

Igualmente, el uso de varios robots para patrullar ayuda claramente a mejorar el rendimiento de la tarea. Por lo tanto, el patrullaje multirobot es un problema que se ha estudiado desde muchas perspectivas diferentes en los últimos años, incluyendo representaciones formales del problema, soluciones óptimas análisis teóricos, implementación y validación experimental.

Ahora, operando como un equipo, en lugar de un conjunto de individuos, estos vehículos no tripulados pueden proporcionar una cobertura actualizada de una gran área coordinando sus movimientos, y mejorar la robustez de esta cobertura compensando el fallo de uno o más vehículos.

A continuación se listan algunos usos principales que se le da al patrullaje con robots autónomos:

- Vigilancia y seguridad: Los investigadores también han identificado a los vehículos aéreos no tripulados como un medio eficaz para vigilancia en entornos urbanos complejos [38] y la vigilancia e inspección de líneas eléctricas. En [20] se presenta un sistema autónomo de patrulla y vigilancia que es una alternativa viable a la patrulla humana tradicional y al sistema de vigilancia por CCTV. Los resultados han demostrado que el sistema propuesto es adecuado y factible para ser utilizado como agente autónomo de patrulla y vigilancia para uso en interiores.
- Control del medio ambiente: Se está disponiendo de la robótica móvil para controlar los niveles de contaminación a una escala bastante grande y recoger información sobre las condiciones ambientales [45]. La integración de las capacidades de planificación de trayectorias de los sistemas robóticos con los modelos de previsión de diversos parámetros ambientales permite mejorar el rendimiento operativo en entornos dinámicos [10]. Así como la capacidad de seguir la evolución de los procesos de interés.

2.2.1 Inteligencia Artificial aplicada a la planificación de rutas y movimientos

La planificación de rutas o trayectorias (en inglés path planning) en robótica consiste en encontrar una trayectoria de un punto inicial a un punto destino sin colisionar con obstáculos. El objetivo de

Figura 2.2 Los robots de seguridad de la serie S5 de SMP Robotics están diseñados para sustituir a los guardias de seguridad que patrullan y para proporcionar una supervisión móvil de CCTV.

la planificación de trayectorias es encontrar una trayectoria óptima o casi óptima desde el estado inicial hasta el estado objetivo que evite los obstáculos basándose en uno o algunos indicadores de rendimiento (como el menor coste de trabajo, camino más corto, ruta más rápida, etc.) en el espacio de movimiento.

La implementación de algoritmos de control y planificación de trayectorias basados en la IA en los robots autónomos mejora drásticamente la eficacia y la practicidad de estos, ya que los entornos en los que deben operar son muy dinámicos y necesitan una adaptación constante. Por ejemplo, los almacenes automatizados en los que los robots están destinados a trabajar en conjunto con los humanos, se benefician directamente de los avances en la planificación de trayectorias complejas y la toma de decisiones autónomas basadas en algoritmos impulsados por la IA. Asimismo, los robots de limpieza y los de reparto también se están convirtiendo en parte de nuestra vida cotidiana.

2.3 Path Planning

La monitorización de entornos con vehículos no tripulados es cada vez más común. La planificación de la trayectoria es un componente esencial en el desarrollo de esta tarea, puesto que determina el nivel de autonomía del vehículo para hacer frente a los cambios del entorno y se considera esencial para la fiabilidad y el éxito de la misión.

La planificación de rutas o trayectorias (en inglés, PP) se considera un problema de optimización multiobjetivo con restricciones, cuyo objetivo es generar una trayectoria factible y eficiente para guiar al robot de forma autónoma hacia el objetivo de interés en el área de operación correspondiente evitando los obstáculos [29]. Hay varios tipos de PP dependiendo del propósito de interés, los mas populares son: de buscar el camino más corto, adaptativo, informativo.

Definición 1. *Problema del Path Planning: Sea un camino* $P(t) = \{p_0, p_1, p_2, p_3, ...P_N\}$*, el path planning es:*

$P^{optimo} = \min J(P)$*, con J un índice o coste de optimalidad.*

Como queremos captar con los sensores la contaminación del lago, nos interesa recorrer una ruta eficiente con el fin de maximizar la información recogida, respetando al mismo tiempo restricciones como los presupuestos de energía, tiempo o distancia de viaje, esto es, Informative Path Planning (IPP). A diferencia de la planificación métrica de trayectorias, la IPP suele producir trayectorias que son mucho más largos que el camino de longitud mínima desde el inicio hasta la meta, es decir,

caminos que pretenden explorar la totalidad de un entorno determinado [13]. Además, nos interesa una IPP en un entorno desconocido o parcialmente observable, donde la planificación se basa en la información del entorno que se ha conseguido hasta el momento, a esto se le denomina Adaptive Informative Path Planning (AIPP) [23, 13, 9].

2.3.1 Algoritmos clásicos de PP

Los planificadores de trayectoria que se centran en la resolución de las soluciones suelen utilizar enfoques de búsqueda de grafos, mientras que los que se preocupan por el tiempo de computación suelen emplear algoritmos evolutivos con el objetivo de satisfacer las restricciones de tiempo real de las operaciones autónomas [29].

La característica idéntica de los algoritmos de búsqueda de grafo es que estiman una solución de coste mínimo con una función heurística. Después, se realizan iteraciones donde se hace un seguimiento de las posibles soluciones para encontrar el que tenga menor coste que el estimado. La trayectoria será entonces la unión de de vértices de coste mínimo desde la posición inicial a la final. Para que estos algoritmos puedan ser aplicables al problema de planificación de trayectorias, el entorno debe transformarse en un grafo para definir las zonas prohibidas y los límites de colisión. El entorno, ahora llamado espacio de búsqueda, es un grafo direccional o bidireccional ponderado.

El algoritmo Dijkstra [36] es un muy útil y popular que se suele aplicar en problemas de problemas de enrutamiento y cálculo del camino más corto. El inconveniente de este algoritmo es que es computacionalmente caro debido a que calcula todos los posibles caminos en un grafo hasta encontrar la de menor coste.

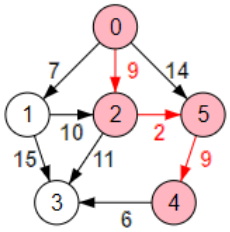


Figura 2.3 Algoritmo de búsqueda de grafos.

Otro algoritmo de búsqueda de grafos es A*, que actúa más eficientemente comparado con Dijkstra debido a su capacidad de búsqueda heurística [22]. En [6] se consigue obtener caminos eficientes con un coste mínimo en un espacio de operaciones de quadtrees utilizando A*. Sin embargo, este algoritmo también sufre de un alto costo computacional en espacios de búsqueda más grandes.

D* [7] es una versión derivada de A* aunque no mejora considerablemente la complejidad computacional de A* en problemas de alta dimensión. Este algoritmo utiliza un mecanismo basado en la interpolación lineal y se ha aplicado ampliamente al problema de planificación de trayectorias [35].

Un enfoque totalmente distinto es el de campos de potenciales artificiales(APF, por sus siglas en inglés). La idea básica de este método es que los robots se consideran partículas que se mueven en el campo potencial virtual. Entonces, a los obstáculos se les asigna un campo de potencial que repele al robot y a la meta se le da un campo de potencial que lo atrae. La resultante de todas las fuerzas aplicadas sobre el robot determinará la dirección y la velocidad de desplazamiento posteriores que guiarán al robot hacia la meta evitando eficazmente cualquier colisión.

Los autores de [40] aplicaron el APF al problema de planificación del movimiento de múltiples vehículos aéreos considerando obstáculos estáticos para probar la capacidad de evitar colisiones del método.

Aunque el algoritmo APF es computacionalmente rápido y eficiente, tiene varios inconvenientes, el más conocido es el de los mínimos locales: el vehículo puede quedar fácilmente atrapado en un obstáculo en forma de U.

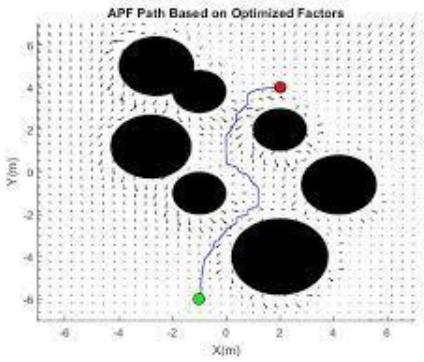


Figura 2.4 Trayectoria de un algoritmo de campos de potenciales artificiales.

Los algoritmos evolutivos(*Evolutionary algorithms*) son otro enfoque empleado con éxito para hacer frente a los problemas NP-hard de planificación de rutas. En muchos aplicaciones, tener un camino rápido y aceptable que satisfaga todas las restricciones es más deseable que tomar un largo tiempo computacional para encontrar el mejor camino.

Los Algoritmos genéticos(GA, por sus siglas en inglés) [14] utilizan operadores genéticos para buscar trayectorias óptimas en una población de posibles caminos que evolucionan de forma iterativa. En [1] se aplican GAs para abordar el problema de planificación de la trayectoria de un UAV, destacaron la eficiencia energética como factor de optimización de la trayectoria.

2.4 Monitorización y patrullaje

La tarea de vigilancia y patrullaje del lago Ypacaraí se centra en la utilidad de la información adquirida a lo largo del camino. Hay varios datos importantes que vigilar relacionadas con calidad del agua (nivel de pH, oxígeno disuelto, etc.) y éstas pueden cambiar con la hora del día y del año, las trayectorias resultantes generadas por cualquier enfoque adecuado deben tener en cuenta no sólo las diferentes zonas (con diferentes niveles de importancia) sino también la revisión de las que no se han muestreado durante mucho tiempo. Este problema, descrito más adelante como el *Patrolling Problem*, requiere un enfoque reactivo y adaptativo debido a las numerosas soluciones.

El problema de la cobertura total (Complete Coverage Path Planning (CCPP)) consiste en determinar la trayectoria que debe seguir un robot para para pasar por todos los puntos de un espacio de trabajo determinado. En [2] se modela elo CCPP mediante Circuitos Hamiltonianos (Hamiltonian Circuit (HC)) y Circuitos Eulerianos (Eulerian Circuit (EC)) para maximizar el valor de la cobertura utilizando una técnica metaheurística como el Algoritmo genético. La principal diferencia entre HC y EC es que un HC es un recorrido que visita cada vértice sólo una vez mientras que el EC visita cada arista sólo una vez.

El Problema del viajante (*Travelling Salesman Problem (TSP)*) es un famoso problema matemático que consiste en encontrar el HC con la distancia más corta para un gráfico dado. Para el caso particular de la planificación de trayectorias del lago Ypacaraí se ha formulado el CCPP como un HC en [2]. Sin embargo, es un algoritmo offline y, en consecuencia, no tiene reactividad a las incertidumbres del modelo.

Una combinación de TSP con *path planning* se muestra en [5] donde se estudia para el caso de un vehículo aéreo no tripulado que intenta visitar ciertos puntos en el mapa mientras evita ciertas zonas prohibidas, que representan, por ejemplo, el radar de un enemigo. En [3] se utiliza este último

enfoque para resolver el problema de patrullaje del lago Ypacarai. Mediante un enfoque GA y un modelo TSP con un número variable de waypoints, el objetivo es seleccionar los waypoints en función de las zonas de mayor interés.

Para hacer frente a la complejidad del problema, también se han considerado Algoritmos de Aprendizaje por Refuerzo [34]. Últimamente, muchos algoritmos de aprendizaje como Q-Learning [21], Deep Q Learning [55] y Reinforcement Learning [47] se han propuesto para la solución de la planificación de la trayectoria. Los métodos más usados en RL, como Q-Learning o Policy Iteration [42], sufren de problemas de dimensionalidad y de falta de generalización a la hora de resolver un problema de *path planning*. Por otra parte, en [30] se demuestra que un algoritmo de Deep Reinforcement Learning puede alcanzar un nivel comparable al de un humano profesional jugando a un conjunto de 49 juegos de la famosa consola Atari 2600. El agente de la red Q profunda fue entrenado recibiendo sólo los píxeles y la puntuación del juego.

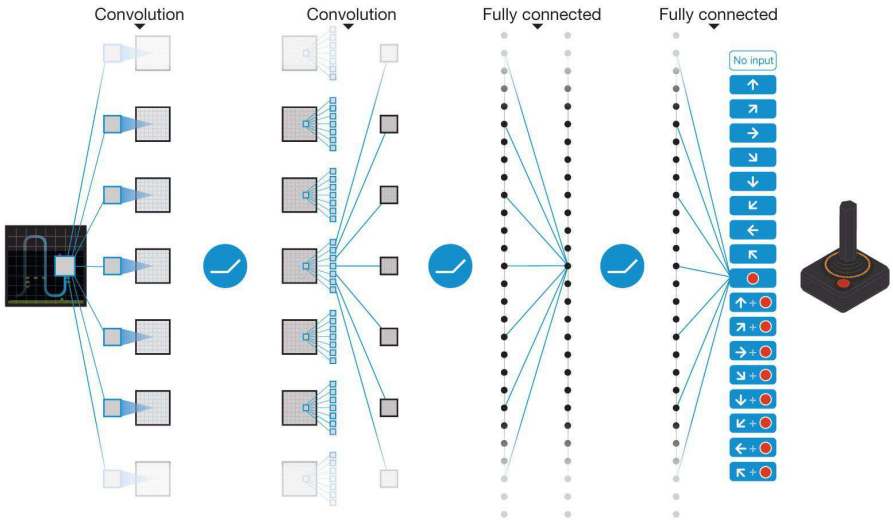
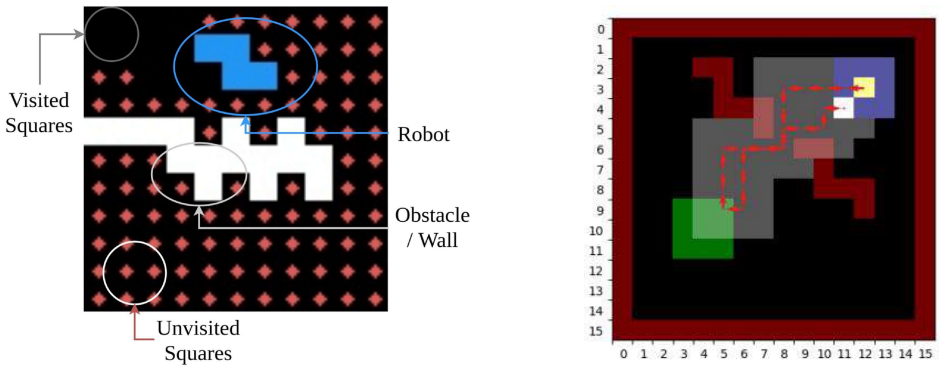


Figura 2.5 Ilustración esquemática de la red neuronal convolucional usada en [30].

La definición del entorno en los problemas donde se aplica Deep Reinforcement Learning (DRL) es esencial dado que el escenario define como el agente percibe el problema a resolver y las condiciones de contorno. En CCPP varios autores definieron el entorno como un mapa cuadriculado de dos dimensiones obteniendo un equivalente en celdas cuadradas del mapa real. Pueden haber zonas transitables, zonas no transitables (obstáculos), asi como casillas sin visitar, visitadas o desconocidas [19]. El robot puede ocupar una celda [44], o varias celdas a la vez [31]. En el *patrolling problem* se intenta no gastar demasiado tiempo en revisitar ciertas zonas y además visitar con más frecuencia a lugares que tienen mayor interés, esto se conoce como patrullaje no homogéneo. Los investigadores de [53] y [34] han abordado este problema dando a cada celda un valor de interés con un rango de valores posibles variable con el tiempo y el número de visitas recibidas. Por otra parte, el entorno puede ser determinista, esto es, el agente conoce valor del estado en el que se encuentra y por tanto la recompensa [19]-[27]. Por otro lado en un entorno estocástico el valor del estado y la recompensa es aleatorio [39].

El aprendizaje de una red neuronal también puede verse afectado por como recibe la información del entorno mediante el estado. En un Proceso de Decisión de Markov(MDP), distinguimos dos tipos de observación del entorno:



(a) [19] (b) [44]

Figura 2.6 Representación del entorno en celdas en [19] y [44]..

- Completamente Observable(*Fully Observable MDP*: El estado completo del escenario es accesible [44]-[53]-[19].
- Parcialmente observables(*Partially Observable MDP(POMDPs)*: Todo el estado no es accesible y el agente no sabe por completo en que estado se encuentra y debe estimarlo [34].

Una representación común del estado para los enfoques de RL es una o varias imágenes RGB del estado que muestren la posición y los obstáculos del mapa. Para poder procesar esta información es necesario utilizar Redes Neuronales Convolucionales. En [24] se propone tomar como estado las posiciones en coordenadas cartesianas de los robots a fin de reducir el coste computacional del algoritmo.

Asimismo, puede haber un sólo agente en el escenario, caso mono-agente, o pueden haber varios agentes, caso multiagente. En el último caso, se debe asegurar tanto que se resuelve el problema de cobertura como que se eviten las colisiones de los robots [53]-[11]. Para el caso multiagente se observan dos enfoques principales: el aprendizaje distribuido [11], [43], como el aprendizaje Q independiente (IQL por sus siglas en ingles), en el que cada agente intenta buscar una estrategia óptima basándose sólo en su propia experiencia, y el aprendizaje centralizado: una red centralizada que intenta optimizar el comportamiento de todos los agentes.

Los algoritmos tipo Value Optimization, como el Double Deep Q-Learning [46] han demostrado ser eficientes en la tarea de exploración de entornos [34]-[39]. En [48] se presenta otra variante dentro del Deep Q-Learning denominada Dueling Network Q-Learning, que viene a mejorar la convergencia del anterior y supuso un avance importante en los resultados obtenidos por DeepMind (Google) en [30].

Sin embargo, las estrategias Actor-Critic [4] fueron capaces de mejorar los resultados de los métodos anteriores. Estos algoritmos son de tipo Policy Optimization y son bastante más efectivos tanto en tiempo de entrenamiento como en coste computacional [19]-[31].

También hay enfoques novedosos como en [27], donde una combinación de DQL y una estructura de exploración paralela mejora el rendimiento de la planificación de la trayectoria y la velocidad de convergencia del aprendizaje.

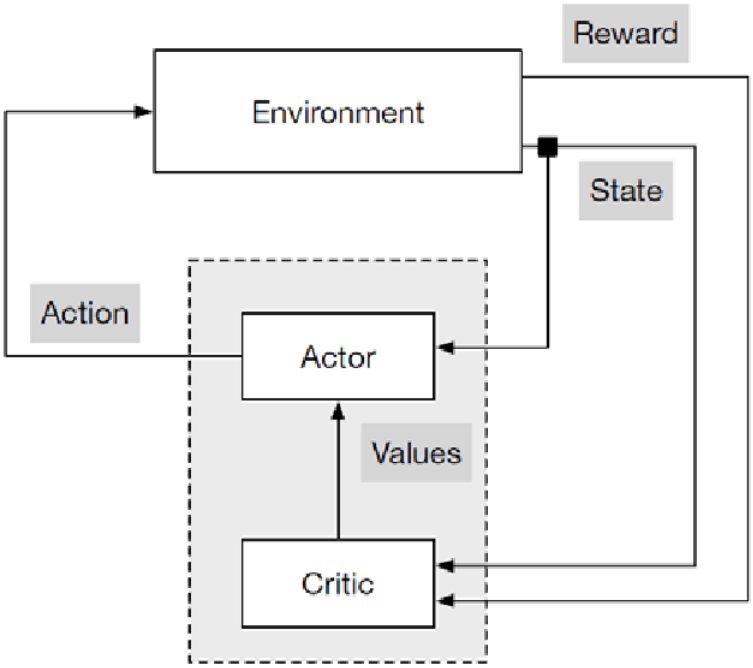


Figura 2.7 Arquitectura de los algoritmos Actor-Critic.

3 Planteamiento del problema

3.1 Problema del vigilante

El *problema del vigilante* o *patrolling problem* consiste en continuamente visitar zonas relevantes de un entorno para supervisar, controlar o protegerlo. En dicho entorno, pueden haber zonas más importantes que otras y por tanto deben recibir más visitas que las otras. Pero no se debe dejar de visitar las zonas menos importantes ya que debe mantenerse vigilado todo el entorno. En definitiva, el objetivo es no dejar zonas del mapa sin cubrir durante demasiado tiempo.

Si queremos una recogida de información eficiente, el objetivo del ASV que supervisa el lago es doble:

- Efectuar una cobertura total del lago, llegando a visitar en algún momento todas las zonas, para tener información completa del estado del lago.
- Volver a visitar aquellas zonas que hace tiempo que no visitamos para tener una imagen actualizada de la métrica del lago.

Queda claro que tenemos una especie de patrolling problem: queremos visitar varias veces zonas interesantes del mapa para afianzar nuestro conocimiento (redundancia útil), pero puede ir en contra de la eficiencia de la cobertura ya que hay que llegar a explorar todo el territorio, y además visitar muchas veces una zona es perder el tiempo (redundancia inútil).

En [28] se demuestra que el entorno que se quiere cubrir puede ser representado como un grafo no dirigido $G(V,E,W)$ donde $V = \{1,2,.....n\}$ es el conjunto de nodos del grafo y E el conjunto de aristas de G. Cada arista tiene un coste que significa la distancia entre nodos.

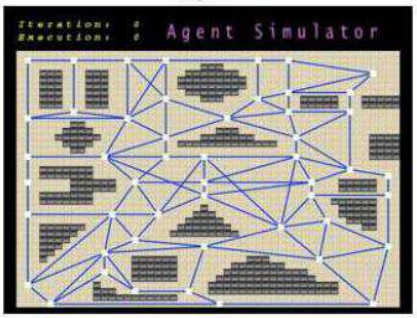


Figura 3.1 Posible representación en grafo propuesto en [28].

El *idleness*, W (en español ociosidad), de un nodo es la cantidad de tiempo transcurrido desde que ese nodo ha recibido la visita de un agente [8]. El *idleness* puede ser ponderado dependiendo de la importancia de cada zona.

Una vez definido el grafo, el objetivo el objetivo es minimizar el *idleness* media de cada nodo, es decir, visitar regularmente cada zona para reducirla.

Atendiendo a la importancia de cada zona, se pueden considerar dos variantes del problema de patrullaje:

- **Patrullaje homogéneo**: Todas las casillas en el mapa tienen la misma importancia. En este caso hay cobertura homogénea ya que se busca cubrir todas las zonas.
- **Patrullaje no homogéneo**: Existen zonas con más importancia que otras y que deben ser revisadas con una mayor frecuencia (en función de la importancia que se dé a cada zona).

En el caso del lago Ypacarai, hemos comentado que hay zonas de alta concentración de contaminación, como por ejemplo, las floraciones. Estas zonas coinciden con áreas portuarias recreativas o zonas industriales cercanas a la orilla como el Puerto de San Blas, en la parte sur del lago. Por lo tanto, el caso de patrullaje no homogéneo es más adecuado para el escenario de Ypacaraí [54].

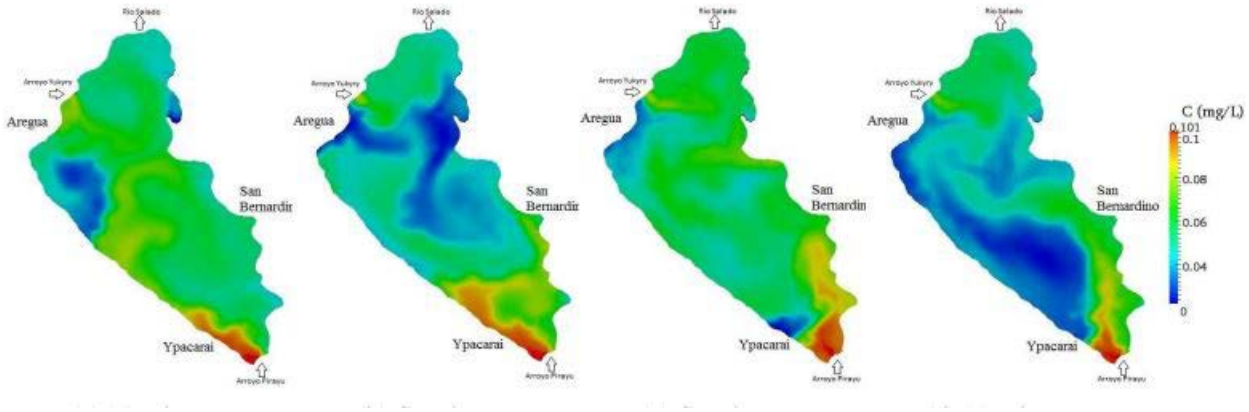


Figura 3.2 Distribución de la contaminación en el lago dependiendo de la dirección del viento. De izquierda a derecha: Noreste, Sureste, Suroeste, Noroeste [32].

Además, el tiempo medio entre visita y visita de una casilla en las zonas de mayor importancia debe ser menor y en las de importancia mínima puede ser mayor. Se crea una matriz de importancia I para ponderar la ociosidad de cada célula (nodo).

Dicho esto, el problema de patrullaje se puede reformular en encontrar una política π que minimice la media de W dado un número time step:

$$G(V,E,W) \longrightarrow \pi(E) | \min \frac{1}{M} \sum_{k=1}^{M} W_k \times I \tag{3.1}$$

Se dice que el problema de patrullaje se resuelve una vez que el planificador de rutas es capaz de elegir los movimientos que dan lugar a un tiempo medio mínimo entre visitas. En nuestro problema, queremos desarrollar un planificador de rutas capaz de proporcionar los waypoints (el siguiente waypoint, dada una posición del ASV en el mapa discreto) que dará como resultado, tras un número de movimientos (timesteps), en la minimización del tiempo medio de espera de cada célula en función de los criterios de importancia de cada zona.

Una vez completado el entrenamiento, la política del ASV decidirá en cada paso de tiempo el siguiente movimiento dado su estado en el mapa.

3.2 Descripción del escenario

El entorno es la representación del problema a resolver determina lo bien o mal que realiza su misión. Asimismo, define cómo el robot o agente puede moverse y las condiciones de contorno y optimalidad. Por otro lado, el escenario suele estar compuesto por zonas de distinta naturaleza: zonas transitables y zonas no transitables. En nuestro caso, el escenario está formado por los siguientes elementos:

- **Mapa**: El lago Ypacaraí tiene una superficie total de 60 km2 con una profundidad media de 1,31 m. para la simplicidad y eficiencia computacional, el mapa se ha discretizado en un grid map en el que cada celda cuadrada es equivalente a una fracción de espacio discretizado del lago. Cuanto más pequeño sea el área, más preciso será el escenario pero más difícil de calcular cada iteración del algoritmo.
  No es conveniente ampliar excesivamente la resolución porque el tiempo de ejecución del entrenamiento podría explotar fácilmente y ralentizar el entrenamiento.
  Existen celdas que se pueden ocupar, que son aquellas que están dentro de la superficie navegable del lago y celdas no ocupables , que son las que representan el espacio de la tierra. No obstante, en la simulación no se considera ningún obstáculo en el interior de el lago, ya que los sensores de obstáculos del ASV (Lidar + Cámara) proporcionan la capacidad de evitarlos (con un planificador local reactivo de la trayectoria).

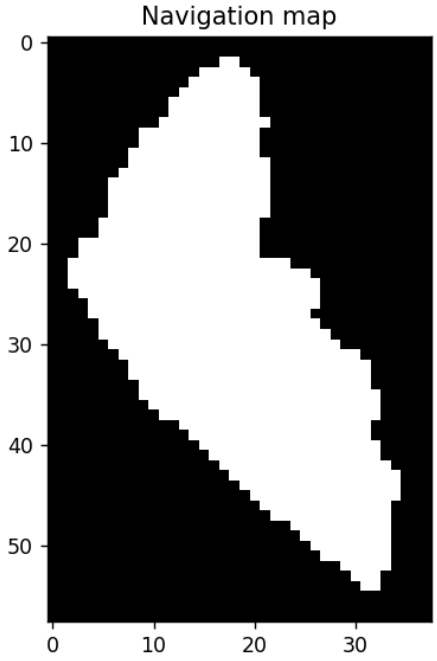


Figura 3.3 Mapa discretizado del lago Ypacaraí.

- **Acciones del agente**: El ASV puede moverse en una de las ocho direcciones diferentes de las celdas contiguas (Siguiendo los puntos cardinales:N, E, S, W y NE, SE, NW, SW) en cada paso. Esto es representativo de las capacidades, más o menos realistas, de movimiento del ASV.
  Las acciones sirven de interacción entre el agente y el entorno, que responderá con una recompensa. Por simplicidad, se supone que la duración de cada movimiento es la misma y en las simulaciones cada time step equivale a un movimiento.

El modulo de alto nivel se centra en la planificación de la trayectoria, es decir, en el cálculo de una secuencia de waypoints a seguir. Un generador local de trayectorias y un piloto automático llevarán a cabo el movimiento.
Una acción ilegal, un waypoint en una celda que no se puede ocupar, no se realizará ni en la simulación ni la aplicación real, por tanto el ASV permanecerá en el mismo lugar.

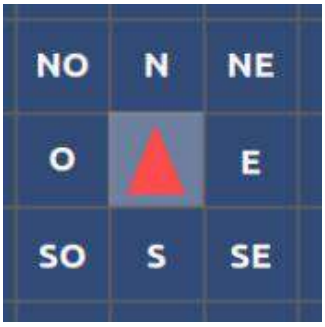


Figura 3.4 Posibles acciones que puede tomar el ASV.

- **Recompensa**: La recompensa es la respuesta del medio a la acción realizada por el agente y puede servir como premio o penalización. La ley de recompensa, que se va a explicar Sección 4.4, estará diseñada para premiar acciones o estrategias que entendemos como adecuados para la monitorización. Penalizará acciones ilegales o estrategias poco adecuadas. También debe ser acorde a los objetivos del ASV si queremos una recogida de información eficiente. Según el tipo de observación, como se comentará más adelante, se va a estudiar el diseño de la función de recompensa para que incentive al agente a visitar zonas no descubiertas.
- **Observación**: La observación del entorno, el estado, es la manera que tiene el agente de obtener datos del entorno, es decir, es la información observable del entorno. El estado puede ser entregado, según el algoritmo, de varias formas. En este trabajo se estudia como se comporta el agente según la observación del entorno:
  - Entorno completamente observable: El agente conoce, mediante imágenes, las dimensiones del mapa, su posición y la información de cada casilla del lago.
  - Entorno parcialmente observable: En este caso se conoce, también mediante imágenes, las dimensiones del mapa y la posición, pero sólo se conocen la información de las zonas del lago que ya se han visitado. Este caso no fue estudiado anteriormente en [54] y es un reto mayor.

4 Metodologia

4.1 Reinforcement Learning

En el aprendizaje por refuerzo el agente buscará las acciones, mediante prueba y error, que le llevarán a alcanzar su objetivo: maximizar la recompensa. Además, el agente aprende interactuando con el entorno, esto es debido a que en el RL, los datos son el mismo entorno. Asimismo, las secuencias de observaciones y acciones, llamadas trayectorias, son datos que se crean interactuando con el entorno y son muy útiles para entrenar a los agentes. Como ocurre en la vida real, las acciones tomadas afectan tanto a la recompensa inmediata como a las recompensas, ya que las acciones actuales condicionan las situaciones futuras.

4.1.1 Definiciones basicas

En esta sección, vamos a explicar los conceptos básicos del aprendizaje por refuerzo.

Agente

El agente (*agent*) es programa de software que toma las decisiones(acciones a realizar) y es el que aprende en RL. Este agente interactúa con el entorno mediante acciones y recibe recompensas en base a ellas.

Entorno

El entorno(*Environment*) es la representación del problema a resolver. Responde a las acciones tomadas por el agente recompensándole positiva o negativamente. Si el agente conoce el modelo del entorno, el aprendizaje es por refuerzo basado en modelo, o *model-based* en inglés. Por el contrario, cuando el agente no conoce el modelo del entorno, el aprendizaje es por refuerzo sin modelo, o *model-free* en inglés.

Estado

El entorno está representado por un conjunto de variables relacionadas con el problema [16], pues bien, el estado del entorno indica cómo están los diversos elementos que lo componen en cada momento. Para el agente, el estado refleja el conocimiento del entorno en un momento determinado. El conjunto de todos los estados posibles se llama espacio de estado y puede llegar a ser infinito. Se denomina observación (*observation*) a la parte del estado que el agente puede observar debido en algunos casos el agente no tiene acceso al estado completo del entorno [16].

Acción

El conjunto de acciones (*actions space*) está formado por las posibles acciones que puede tomar en un momento determinado el agente y son puestas a disposición por el entorno. El agente elegirá una acción de ese conjunto de acciones y puede cambiar el estado del entorno. El conjunto de acciones dependen del estado y puede haber estados que tengan diferentes acciones disponibles. El espacio de acciones puede ser de acción discreto, por ejemplo movimiento arriba o abajo, o puede ser de acción continua, por ejemplo girar un volante.

Función de transición

La función de transición (*transition function*) es la representación matemática del cambio de estado del entorno como respuesta a la acción del agente. También representa la probabilidad de moverse de un estado a otro y por ello es también llamado probabilidades de transición (*transition probabilities*) entre estados. La función de transición puede ser determinista o estocástico. En el caso del escenario completamente observable es determinista, ya que conocemos el valor de interés de todo el mapa. En el caso del escenario parcialmente observable es estocástico porque no sabemos el valor que nos vamos a encontrar de interés.

Recompensa

La recompensa(*reward*) es la respuesta del entorno como consecuencia de las acciones del agente. Esta recompensa puede simbolizar un premio o una penalización y en ese sentido servirá al agente como medida de bondad. El entorno utiliza la función de recompensa (*reward function*) para decidir que recompensa reproducir. En RL el objetivo del agente es maximizar la recompensa recibida, por lo que recibir una recompensa positiva al efectuar una acción en un estado determinado, le 'refuerzaá volver a realizar esa acción en el futuro.

Experiencia

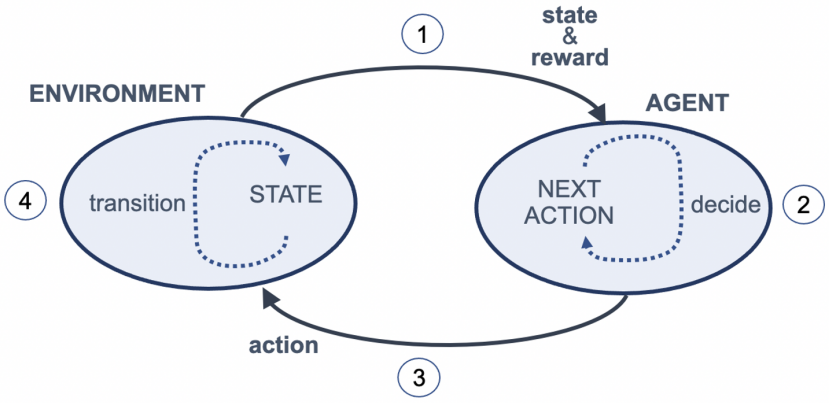


Figura 4.1 Ciclo de aprendizaje por refuerzo.

Llamamos *time step* o iteración a una interacción entre el agente y el entorno en el tiempo, esta interacción se hace por ciclos(Figura 4.1). En cada ciclo, el agente observa el entorno a partir de la información obtenida en el anterior ciclo: estado y recompensa. Con esta información, realiza una acción con el objetivo de obtener la máxima recompensa. El entorno responde con el nuevo estado y recompensa que ha causado la acción del agente. El conjunto del estado, la acción, la recompensa y el nuevo estado en cada *time step* se llama experiencia.

Episodio y retorno

Las tareas episódicas son aquellas que tienen un estado terminal y por ende un número finito de estados, como por ejemplo un juego. La secuencia de *time step*s desde el principio hasta el final de una tarea episódica se denomina episodio. Por el contrario, las tareas que no tienen un final natural y no acaban se denominan tareas continuas, como por ejemplo aprender a programar. El retorno (*return*) es la suma de las recompensas recolectadas en un solo episodio y sirve de guía para el agente para saber si va bien encaminado.

Política

La política es la estrategia que sigue el agente para maximizar el retorno. En la práctica es una función que aplica el agente para elegir la acción óptima en cada estado.

4.1.2 Proceso de decisión de Markov

El Proceso de decisión de Markov (MDP) es un formalismo intuitivo y fundamental para la planificación teórica de decisiones (en ingles DTP) y otros problemas de aprendizaje en dominios estocásticos [33]. En este modelo, un entorno se modela como un conjunto de estados y se pueden realizar acciones para controlar el estado del sistema. Utilizando el marco matematico del MDP se pueden formalizar los elementos del problema de RL descritos en la Subsección 4.1.1 de este capítulo.

Para poder utilizar el MDP en un problema se debe cumplir la propiedad de Markov(en inglés *Markov Property*). Esta se cumple cuando los estados futuros dependen del estado actual y no del historial de estados pasados, es decir, el futuro depende solo del presente y no del pasado. En este sentido, la recompensa obtenida al realizar una acción en un estado no depende del camino seguido hasta llegar a ese estado.

En el caso de la exploración del lago Ypacaraí podríamos sostener que se cumple, puesto que habiendo visitado un conjunto de celdas y dada una posición del agente, el estado futuro solo depende del estado actual (y de la acción).

A continuación definimos los distintos elementos:

- **Estado**: Un conjunto de estados S. El estado actual se define como s.
- **Acción**: Un conjunto de acciones A. Una acción dentro de ese espacio de acciones se define como a.
- **Función de transición**: Una Función de transición P. En el caso de que se pase del estado s al siguiente estado $s\prime$ realizando la acción a, se define como $P^a_{ss\prime}$.
- **Recompensa**: Una Función de recompensa R. En el caso de que se pase del estado s al $s\prime$ realizando la acción a, se define como $R^a_{ss\prime}$.
- **Factor de descuento**: El factor de descuento γ es un valor entre $[0,1]$ para ajustar la importancia de las recompensas a lo largo del tiempo. Cuanto antes se obtenga una recompensa, más probable será obtenerla, por lo tanto más importancia se le da. El retorno con descuento R_t en un *time step* t al realizar una acción a_t se puede calcular como:

$$R_t = r_{t+1} + \gamma * r_{t+2} + \gamma^2 * r_{t+3} + \gamma^3 * r_{t+4} + \gamma^4 * r_{t+5} = \sum_{k=t+1}^{T} R_t = \gamma^{k-t-1} r_k \tag{4.1}$$

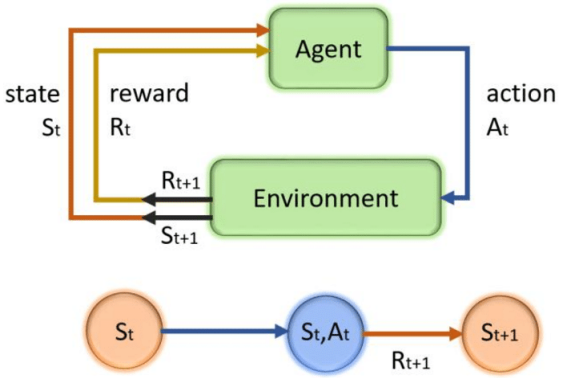


Figura 4.2 Ilustración visual del Proceso de decisión de Markov.

4.1.3 Funciones de valor y la ecuación de Bellman

Funciones de valor

Una función de valor devuelve información sobre un sistema y en RL se utiliza para medir lo que es bueno para el agente a largo plazo. En RL se utilizan dos tipos de función de valor: la función de valor del estado y la función de valor estado-acción.

La función de valor del estado, también llamado función V , mide como de bueno es cada estado según el retorno con descuento al seguir una política $\pi(s)$ determinada. Si la función de transición del entorno puede actuar de forma estocástica, calculamos la esperanza matemática (expected value), del retorno con descuento para un estado promediando un gran número de episodios.

$$V_\pi = \mathbb{E}_\pi[R_t|S_t = s] = \mathbb{E}_\pi[\sum_{j=0}^{T} \gamma^j r_{t+j+1}|S_t = s] \tag{4.2}$$

La expresión Ecuación 4.2 describe la función V como el valor esperado del retorno con descuento R_t, comenzando desde el estado s en el *time step t* y siguiendo la política π.

La función de valor estado-acción, también llamado función Q , mide como de bueno es realizar una acción a en un estado s según el retorno con descuento al seguir una política $\pi(s)$ determinada. Al igual que en la función de valor del estado, calculamos la esperanza matemática (expected value) del retorno con descuento para un estado promediando un gran número de episodios.

$$Q_\pi(s,a) = \mathbb{E}_\pi[R_t|S_t = s,A_t = a] = \mathbb{E}_\pi[\sum_{j=0}^{T} \gamma^j r_{t+j+1}|S_t = s,A_t = a] \tag{4.3}$$

La expresión Ecuación 4.3 describe la función Q como el valor esperado del retorno con descuento R_t al realizar una acción a, desde el estado s en el *time step t* y siguiendo la política π.

Ecuación de Bellman

La ecuación de Bellman para la función V (Ecuación 4.4) y la función de valor estado-acción Q (Ecuación 4.10), se pueden expresar como una suma de la recompensa inmediata y el valor descontado del siguiente estado.

$$V_\pi(s) = \sum_a \pi(s,a) * \sum_{s\prime} P^a_{ss\prime}[R^a_{ss\prime} + \gamma V_\pi(s\prime)] \tag{4.4}$$

$$Q_\pi(s,a) = \sum_{s\prime} P^a_{ss\prime}[R^a_{ss\prime} + \gamma Q_\pi(s\prime,a\prime)] \tag{4.5}$$

Ecuación de optimalidad de Bellman

Conseguir el comportamiento óptimo de un agente es equivalente a encontrar la política óptima, esta es aquella política que maximice las ecuaciones de Bellman Ecuación 4.4 y Ecuación 4.10.

Si denotamos $V_*(s)$ como la función de valor de estado óptima, tenemos que:

$$V_*(s) = max_\pi V_\pi(s) \tag{4.6}$$

La ecuación de Bellman óptima se calcula seleccionando la acción que da el valor máximo:

$$V_*(s) = \max_a \sum_a \pi(s,a) * \sum_{s\prime} P^a_{ss\prime}[R^a_{ss\prime} + \gamma V_*(s\prime)] \tag{4.7}$$

Por otro lado,Si denotamos $Q_*(s,a)$ como la función de valor de estado óptima, tenemos que:

$$Q_*(s,a) = max_\pi Q_\pi(s,a) \tag{4.8}$$

De nuevo, la ecuación de Bellman óptima se calcula seleccionando la acción que da el valor máximo:

$$Q_*(s,a) = \sum_{s\prime} P^a_{ss\prime}[R^a_{ss\prime} + \gamma \max_{a\prime} Q_*(s\prime,a\prime)] \tag{4.9}$$

Por último, se define la ecuación de optimalidad de Bellman como:

$$V_*(s) = \max_{a\prime} Q_*(s\prime,a\prime) \tag{4.10}$$

Una solución para un MDP es la *Programación dinámica basada en la ecuación de Bellman*. Se basa en calcular el valor óptimo de un estado mejorando iterativamente la estimación de $V(s)$ aprovechando que la ecuación de Bellman nos permite definir una función de valor de forma recursiva. Para poder emplear este algoritmo es necesario conocer las probabilidades de transición $P^a_{ss\prime}$ y las recompensas $R^a_{ss\prime}$. En otras palabras, se supone conocido el modelo completo del entorno y por ello se trata de un método basado en modelo(*model-based*). Esta última suposición limita el rango de aplicaciones de la programación dinámica ya que en la mayoría de los casos no se conoce por completo el modelo del entorno.

4.1.4 Políticas épsilon-greedy

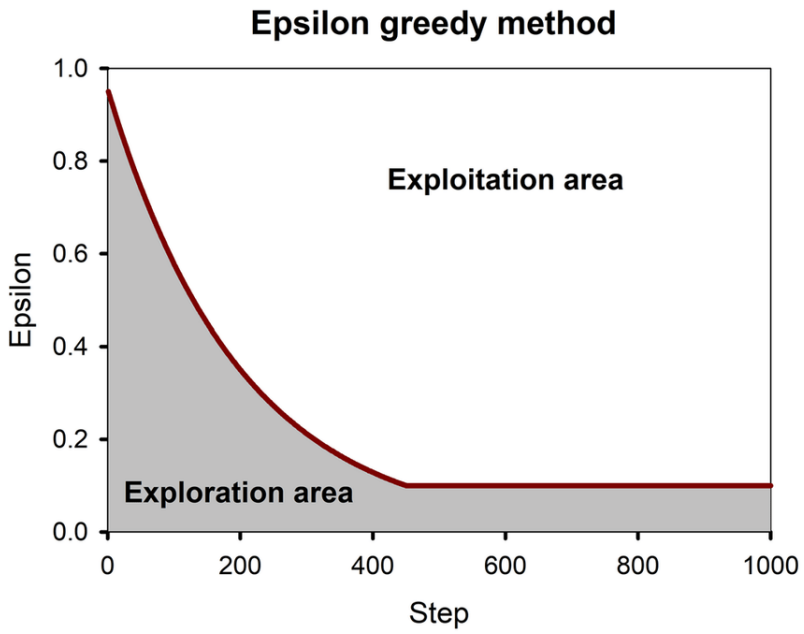


Figura 4.3 Posibles valores de ε a lo largo de un entrenamiento de 1000 episodios.

Un agente está explotando su conocimiento cuando, mantiene estimaciones de los valores de todas las acciones que ya había tomado y elige la que tiene mayor recompensa. Por el contrario, un agente está explorando cuando está eligiendo realizar una acción que nunca había tomado y no optar por la mejor acción estimada.

Parece conveniente que el agente elija siempre la acción con la que mayor recompensa obtenga, pero contradictoriamente, tiene que aventurarse a buscar más acciones que pueden ser potencialmente mejores. El dilema exploración-explotación es un gran desafio que surge en el aprendizaje por refuerzo y consiste en intentar equilibrar la exploración y la explotación del agente para aprovechar las ventajas de ambos y deshacerse de sus inconvenientes. De esta manera, el agente conseguiría aprovechar al máximo su sabiduría mientras amplia su conocimiento.

La acción que tiene mayor recompensa de un conjunto de acciones se denomina acción greedy.Por consiguiente, si el agente siempre elige este tipo de acciones, seguiría una política greedy. Por otro lado, en una política ε-greedy se elige la acción no greedy (una al azar) con una probabilidad ε. Mientras que se elige la opción greedy, la que es mejor según nuestra experiencia, con una probabilidad $(1-\varepsilon)$.

Utilizar una política ε-greedy puede solventar el dilema exploración-explotación si se decrece gradualmente el valor de ε a medida que el agente vaya adquiriendo mas conocimiento. Esto es debido a que es conveniente que el agente comience su entrenamiento optando por la exploración en lugar de la explotación, y probando diversas estrategias para maximizar el retorno. A lo largo del entrenamiento debe ir quedándose cada vez más con las acciones que maximizan la recompensa hasta que al final actúe según un valor muy bajo de épsilon y preferir la explotación, aunque siempre sigue intentando mejorar.

Para implementar esto se introduce un factor que llamaremos decay rate con un valor menor, aunque cercano, a 1 que multiplique el hiperparámetro ε en cada iteración.

4.1.5 Aprendizaje por diferencia temporal: Q-Learning

Para poder superar las limitaciones de la programación dinámica, tenemos que acudir a métodos *model-free*, es decir, que puedan aplicarse en los entornos cuya dinámica desconocemos. Los métodos Monte Carlo se basan en el aprendizaje a partir solo de la experiencia y tienen como objetivo aproximar las funciones V y Q promediando los retornos de las muestras. No obstante, en el método de control Monte Carlo constant-alpha para obtener una política óptima, se aplica la siguiente ecuación de actualización después de un episodio completo [16]:

$$Q(s,a) = Q(s,a) + \alpha * (R_t - Q(s,a)) \tag{4.11}$$

El hiperparámetro α es denominado *step size* y es el que marca la rapidez con la que se actualiza $Q(s,a)$ en cada iteración. El retorno R_t sólo se puede obtener al final de episodio, lo que dificulta la implementación de este método en tareas continuas.

En [41] se presenta una mejora a los anteriores métodos, llamado Diferencia Temporal (TD). A diferencia del método Monte Carlo, el cual utiliza el error entre los resultados predichos y los reales, los métodos TD se basan en el error o la diferencia entre las predicciones sucesivas en el tiempo. Esto es posible debido a que el aprendizaje se produce siempre que hay un cambio en la predicción a lo largo del tiempo [41]. En consecuencia, los métodos TD actualizan la tabla Q en cada *time step*.

Para poder aplicar un algoritmo TD hay que predecir el retorno en cada *time step*, esto es, usar predicción para realizar otra predicción y en RL se llama bootstrapping.

Q-Learning

El algoritmo Q-Learning es un caso de TD y la idea principal es que el algoritmo predice el valor de un par estado-acción $Q(s,a)$, y luego compara esta predicción con las recompensas acumuladas observadas en algún momento posterior. Después, actualiza los parámetros del algoritmo, para que la próxima vez haga mejores predicciones.

La ecuación de actualización es la siguiente:

$$Q(s,a) = Q(s,a) + \alpha * (r + \gamma * \max_a Q(s\prime,a) - Q(s,a)) \tag{4.12}$$

Esta actualización se asemeja al descenso de gradiente estocástico, actualizando el valor actual $Q(s,a)$ hacia un valor objetivo $r + \gamma * \max_a Q(s\prime,a)$.

Como se puede observar en la Ecuación 4.12, el retorno en cada *time step* se estima como la suma de la recompensa inmediata r y el valor descontado por el factor gamma del siguiente par estado-acción siguiendo una política greedy $\max_a Q(s\prime,a)$. El resultado que devuelve el algoritmo es

una tabla de valores de estimación para cada estado y cada acción. A continuación se muestra el algoritmo:

Algorithm 1 Algoritmo Q-Learning

```
Q ← Tabla vacía de A×S valores;
end ← 0;
while end == 0 do
    step ← 0;
    Reseteamos el escenario y obtenemos el estado s;
    while step < step_MAX do
        Tomamos una acción a mediante una política ε-greedy basada en Q(s,a);
        Aplicamos a y adquirimos el nuevo estado del escenario s' y la recompensa r;
        Actualizamos la tabla Q:
        Q(s,a) = Q(s,a) + α * (r + γ * máx_a Q(s',a) − Q(s,a))
        step ← step + 1;
        if ΔQ<Umbral then
            end ← 1;
        end if
    end while
end while
```

4.2 Redes neuronales artificiales(RNAs)

Las RNAs imitan el funcionamiento de las redes de neuronas biológicas construyendo un modelo computacional y matemático. Están constituidos por neuronas artificiales interconectadas masivamente en paralelo. Permiten a los programas informáticos reconocer patrones y resolver problemas comunes en los campos de la IA, el aprendizaje automático y el aprendizaje profundo.

4.2.1 Neurona artificial

En la Figura 4.4 se muestra la estructura de una neurona artificial marcando las equivalencias entre estas con las neuronas biológicas en la parte superior de la figura.

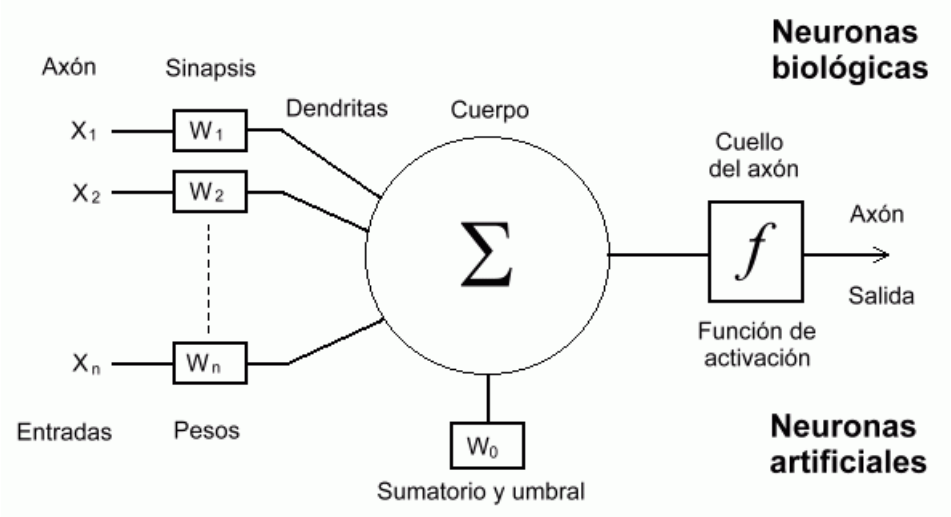


Figura 4.4 Modelo de una neurona artificial.

Las n neuronas x_i están enviando señales de entradas a la neurona artificial, estas entradas, definidas por un vector $\vec{X}$, corresponden a las señales de la sinapsis de una neurona biológica. Los valores w_i representan los pesos sinápticos en las dendritas [17], es decir, la fuerza de una conexión sináptica, y se representa por un vector $\vec{W}$. Las existencias de conexiones determinan si es posible que una unidad influya sobre otra, El sumatorio Σ se corresponde al cuerpo de la neurona, suma todas las entradas multiplicadas por su peso sináptico correspondiente más el sesgo (w_0).

$$E = w_0 + x_1 w_1 + x_2 w_2 + x_3 w_3 ... x_n w_n \tag{4.13}$$

$$E = \vec{X}^T * \vec{W} + w_0 \tag{4.14}$$

Las señales E luego son procesadas por una función de activación F, de la cuál depende la señal de salida S. La función de activación permite a las neuronas cambiar de nivel de activación a partir de las señales que recibe. El nivel de activación(estado interno de la neurona) de una célula depende de las entradas recibidas y de los valores sinápticos, pero no de anteriores valores de activación. La función de activación es la característica que mejor define el comportamiento de la neurona. Existen varios tipos de funciones: lineal, de umbral, o cualquier función no lineal.

La salida de la neurona es el resutado de aplicar la función de activación al sumatorio de los valores de entrada.

$$S = F(E) \tag{4.15}$$

4.2.2 Perceptrón multicapa

En los años 50, Frank Rosenblatt crea el modelo del perceptrón, que es una neurona artificial de la misma arquitectura que la presentada en la Subsección 4.2.1, y que es capaz de realizar tareas de clasificación automática. El perceptrón multicapa (o red neuronal densa) es una generalización del perceptrón simple y se caracteriza por tener las neuronas agrupadas en capas de distintos niveles. Como ya se ha demostrado en [15], el perceptrón multicapa es un aproximador universal. Esto le permite, entre otras cosas, aprender de ejemplos y aproximar relaciones no lineales y por lo tanto, abordar problemas reales.

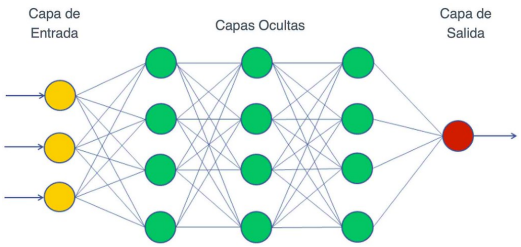


Figura 4.5 Perceptrón Multicapa.

En cuanto a la arquitectura del perceptrón multicapa, esta está formada por tres tipos de capas: la capa de entrada, que es la que recibe los datos del exterior y los propagan hacia todas las neuronas de la siguiente capa. Las siguientes son las capas ocultas, que son las que realizan un procesamiento no lineal de los datos entregados por la capa de entrada. Por último, la capa de salida Recibe la información de las capas anteriores y proporciona al exterior la respuesta o decisión de la red.

Para el perceptrón multicapa, las funciones de activación más usadas son:

1. Función identidad (identity): La neurona artificial propaga el mismo valor sin alterarlo.

$$f(x) = x \tag{4.16}$$

2. Función escalón (threshold):Es una función de tipo binaria donde cualquier salida con valor negativo es propagado como un 0, mientras que todo valor positivo incluyendo el 0 es propagado como 1.

$$f(x) = \begin{cases} 0 \quad si \quad x < 0 \\ \\ 1 \quad si \quad x \geq 0 \end{cases} \tag{4.17}$$

3. Función sigmoide(sigm): Función continua que devuelve un valor de salida real mayor o igual a 0 y menor o igual a 1 para cualquier valor de entrada.

$$f(x) = \frac{1}{1+e^{-}x} \tag{4.18}$$

4. Función rectificadora (ReLU): La función de activación ReLU activa la salida de una neurona solo si la entrada está por encima de cierto umbral. El comportamiento por defecto es que toma el valor 0 para cualquier valor negativo de entrada, mientras que la salida será una relación lineal con la variable de entrada si esta última es mayor que cero. Esta función, a diferencia de las dos anteriores, no está acotada.

$$f(x) = max(0,x) \tag{4.19}$$

5. Función tangente hiperbólica (tanh). Esta función es muy similar a la sigmoidal aunque está acotada entre -1 y 1.

$$f(x) = \frac{1-e^{-}2x}{1+e^{-}2x} \tag{4.20}$$

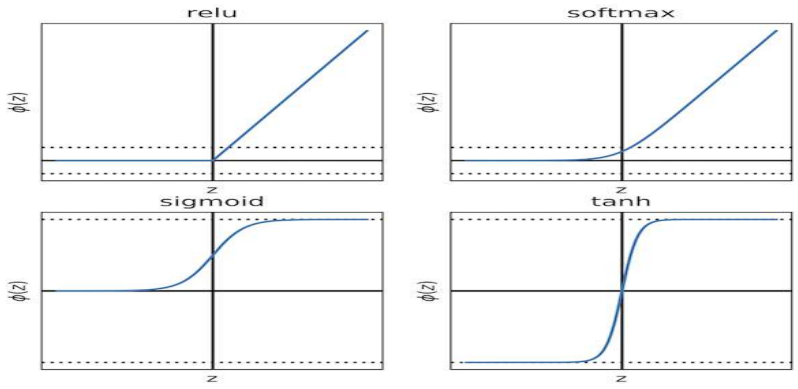


Figura 4.6 Funciones de activación.

Leaky ReLU es un tipo de función de activación basada en una ReLU, pero tiene una pequeña pendiente para los valores negativos en lugar de una pendiente plana. Este tipo de función de activación es popular en tareas en las que podemos sufrir gradientes dispersos.

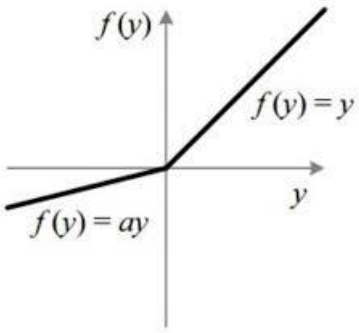


Figura 4.7 Leaky ReLU.

4.2.3 Redes Neuronales Convolucionales

Una red neuronal convolucional (en inglés,Convolutional Neural Network (CNN)) es un tipo de red que recibe como entrada una imagen, de la que es capaz de extraer diversas características aplicando la operación de convolución en ella. Están especialmente diseñadas para el procesamiento de datos de dos dimensiones. En la actualidad, son muy usadas para resolver problemas de clasificación, localización e identificación de objetos dentro de imágenes.

Las CNN son normalmente compuestas por varias capas, dos de ellas son: capas de convolución y la capa de agrupación(pooling layer).

Capas de convolución

Es la capa principal de las redes de neuronas convolucionales, siendo necesario el uso de una o de más capas de este tipo en dichas redes. Básicamente, la operación que realiza la red en esta capa es la convolución. Esta es una operación lineal que en el caso bidimensional se puede formular como:

$$x * K = y[i,j] = \sum_{m=0}^{M} [\sum_{n=0}^{N} x(m,n) \times w(i-m, j-n)] \tag{4.21}$$

En la Ecuación 4.21, **x** sería la imagen de $N \times M$ píxeles de entrada, **w** el filtro o kernel e **y** la salida producida por la convolución, la cual denominaremos mapa de características. Este resultado de la capa de convolución es lo que se utilizará en la siguiente capa. El valor de los pesos de la CNN son los valores del filtro **w**. De esta forma ,la red tendrá como objetivo el aprendizaje de filtros que se activan en presencia de alguna feature o característica en una región específica de la entrada.

En el caso de las imágenes a color RGB, la convolución equivaldría a la realización de tres convoluciones bidimensionales, una a la capa de rojo, una a la verde y otra a la azul, y sumar los resultados.

Normalmente, se pasan los mapas de características de una CNN a una red neuronal densa normal, reduciendo la dimensión de 2D a 1D. Esta combinación de arquitecturas se le conoce como Red Neuronal Convolucional Densa.

Capas de agrupación (pooling layer)

Las capas de agrupación se colocan usualmente entre dos capas convolucionales. Esta capa se suele utilizar para reducir información espacial y evitar el sobreajuste (overfitting) en la red y algo de invarianza a la traslación. Es decir, lo que hacen las capas de agrupación es simplificar la información en la salida de la capa convolucional y reducir la cantidad de parámetros de la red

La operación de pooling actúa del mismo modo que una operación convolucional, haciendo operaciones en pequeñas regiones de la imagen de entrada. Una técnica muy utilizada en la agrupación es el *Max-pooling*. Se realiza realiza aplicando un filtro de máximos a subregiones no superpuestas de la matriz inicial.

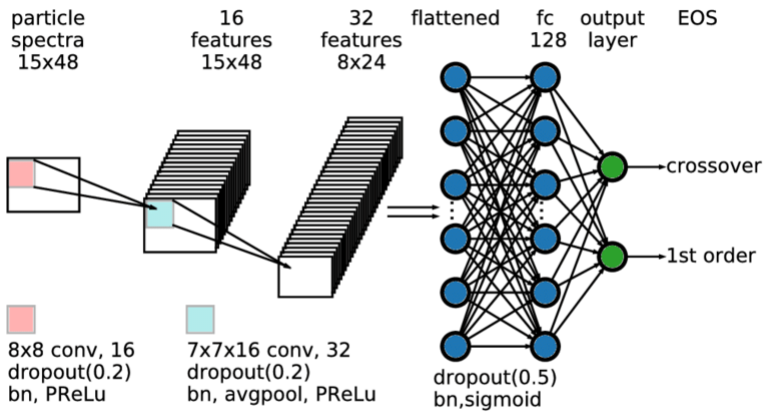


Figura 4.8 Ejemplo de una Red Neuronal Convolucional Densa.

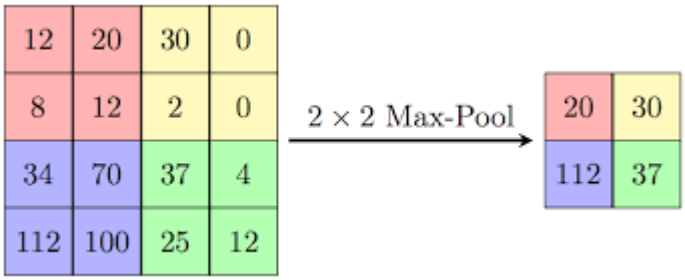


Figura 4.9 Operación de Maxpool 2X2.

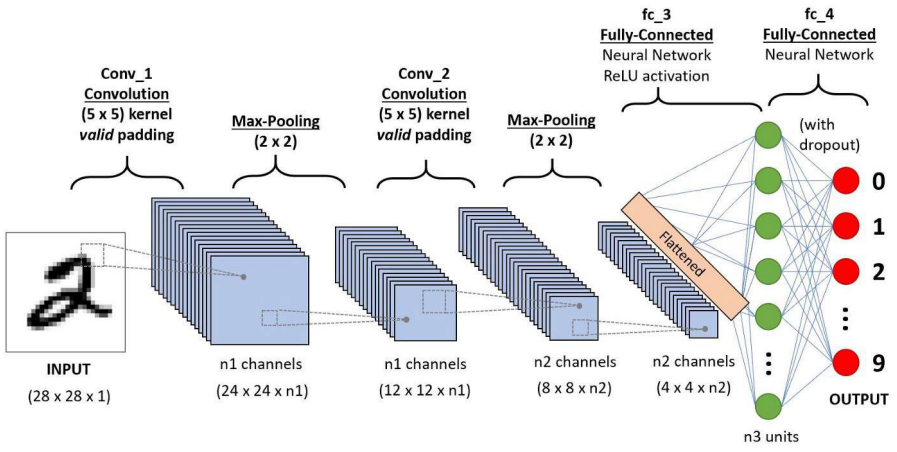


Figura 4.10 Ejemplo de una Red Neuronal Convolucional Densa con Operación de Maxpool 2X2.

4.2.4 Retropropagación

El proceso de aprendizaje de una red neuronal consiste en adaptar los parámetros de la red(pesos y sesgos) para conseguir la salida deseada. Los parámetros se inicializan aleatoriamente, y a medida que la red se vaya exponiendo a los datos de entrenamiento, se modifican los parámetros para que la salida sea lo más próxima posible a la deseada. Para cuantificar la diferencia entre la predicción y el objetivo se utiliza una función de pérdida(loss function).
El aprendizaje de una red neuronal es equivalente a encontrar el mínimo de la función de pérdida, ya que, en dicho punto el error es próximo a cero. Lo cuál implica que la salida de la red es cercana a la deseada. Para ello, se debe utilizar un algoritmo de optimización capaz de resolver un problema de minimización no lineal. El algoritmo de descenso de gradiente se basa en calcular el gradiente de la función de pérdida con respecto a los parámetros de la red neuronal y ajustar iterativamente los parámetros en la dirección opuesta al gradiente, disminuyendo así el error.
El algoritmo de descenso del gradiente usa la primera derivada (gradiente) de la función de pérdida y la encadena con las derivadas de cada capa de la red neuronal. Esto da lugar al algoritmo back propagation, que es una propagación hacia atrás de la señal de error, esto tiene sentido ya que el error de una capa anterior depende de la capa posterior. La retropropagación comienza con el valor de pérdida final y recorre las capas hacia atrás, aplicando la regla de la cadena para calcular la 'culpa'que tuvo cada parámetro en el valor de pérdida.

4.3 Deep Reinforcement Learning

El Aprendizaje por refuerzo profundo, en ingles DRL, es una téncica de aprendizaje automático o Machine Learning y resulta de la combinación de técnicas de aprendizaje por refuerzo (RL) y técnicas de aprendizaje profundo (DL).

4.3.1 Deep Q-Learning

En la práctica, la aplicabilidad del algoritmo Q-Learning se ve limitado hasta ahora en dominios en los que los espacios de estados son de alta dimensión. En estos casos, obtener una tabla de valores para cada par estado-acción puede llegar a ser insostenible computacionalmente. En su lugar, es habitual utilizar un aproximador no lineal de funciones para estimar la función estado-acción, como una red neuronal:

$$Q(s,a) \approx Q(s,a;\theta) \tag{4.22}$$

El conjunto de parámetros θ son los pesos de la red neuronal.

En cambio, utilizar una red neuronal, para representar la función en el RL, puede llegar a provocar inestabilidad o incluso pueden divergir los pesos de la red. Una de las causas de esto es la correlación entre acción y estado y/o secuencia de observaciones en un entorno complejo. Otra posible causa es el hecho de que pequeñas actualizaciones de la función pueden cambiar significativamente la política y, por tanto, cambiar la distribución de los datos.

La ecuación de actualización en ese caso sería:

$$Q(s,a;\theta) = Q(s,a;\theta) + \alpha * (Q_{target} - Q(s,a;\theta)) \tag{4.23}$$

Q_{target} es la función objetivo o target:

$$Q_{target} = r + \gamma * \max_{a\prime} Q(s\prime,a\prime;\theta) \tag{4.24}$$

. La red se puede entrenar minimizando una función de pérdida $L(\theta)$ en cada iteración :

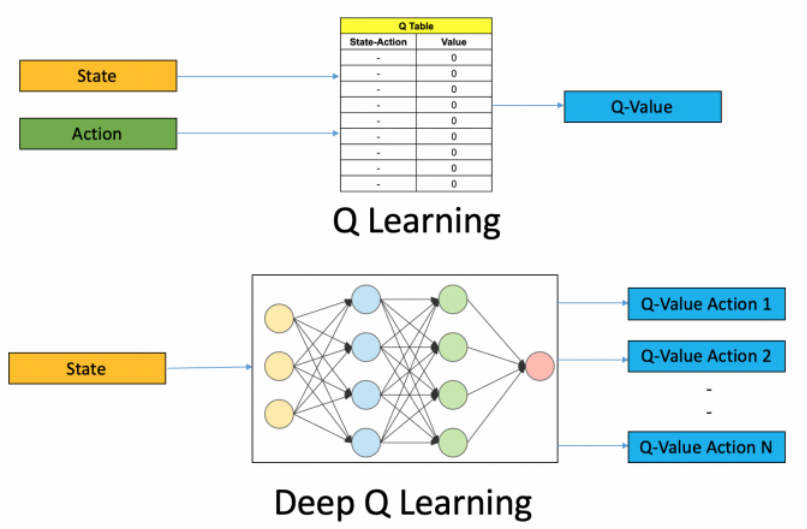


Figura 4.11 Q-Learning vs Deep Q-Learning.

$$L(\theta) = L(Q_{target}, Q(s,a;\theta)) \tag{4.25}$$

En [30] se utiliza una red de neuronas convolucional como aproximador, Deep Q-Network (DQN), para aplicar un algoritmo de Deep Q-Learning (DQL) capaz de alcanzar un nivel de rendimiento humano en los juegos del Atari 2600. Esta publicación superó al estado del arte debido a que abordaron las inestabilidades antes mencionadas con una variante novedosa del aprendizaje Q, que utiliza dos ideas clave: experience replay y target network.

Experience Replay

Surge como una técnica para mitigar el efecto negativo de la correlación entre observaciones sucesivas. La idea principal es crear un buffer de memoria que recolecte las experiencias pasadas formadas por la tupla $(s,a,r,s\prime)$. Una vez se agrupen las experiencias suficientes, se eligen al azar varias experiencias, y la más reciente, para actualizar los parámetros de la red. . El búfer es de tamaño fijo, y mientras el agente va interactuando con el entorno, se guardan los datos más recientes y se eliminan las tuplas de las experiencias más antigua. Recordemos que estamos en un entrenamiento basado en el descenso del gradiente y on-line , esto es, hacemos una propagación hacia atrás cada vez realizamos una acción. Además, estamos tratando de considerar que tenemos un problema de aprendizaje supervisado al calcular los valores estimados usando la ecuación de Bellman para aproximar la función Q con una red neuronal. En un entrenamiento supervisado se utiliza la técnica de batch learning/update en el que no se actualizan los pesos hasta que se hayan iterado a través de un subconjunto aleatorio de datos de entrenamiento y se hayan calculado la suma o gradiente medio del lote. Esto es necesario porque promedia hacia la salida objetiva y estabiliza el aprendizaje.

Sin embargo, en nuestro caso, podemos sufrir de interferencia catastrófica u olvido catastrófico(catastrophic forgetting). Cuando estados estados similares devuelven una respuesta distinta ante la misma acción, el agente reemplaza lo aprendido anteriormente con los datos nuevos. En otras palabras, un modelo puede perder drásticamente su capacidad de generalización en una tarea tras ser entrenado en una nueva tarea [49]. En un escenario de aprendizaje continuo, un agente no podría adaptarse por completo porque olvida el conocimiento existente cuando aprende cosas nuevas.

Experience replay nos proporciona básicamente la actualización por lotes(minibatch update) en un esquema de aprendizaje/entrenamiento en línea(on-line learning). Asimismo, esta técnica nos permite aprender de una experiencia varias veces y recordar ocasiones peculiares, evitando el olvido catastrófico.

La estructura de datos llamada deque de la librería collections de Python es la que usaremos para almacenar las experiencias del agente. Funciona como una lista con un tamaño máximo, si está llena y se intenta añadir un elemento a la lista, se agregará al final de la lista y se eliminará el primer elemento.

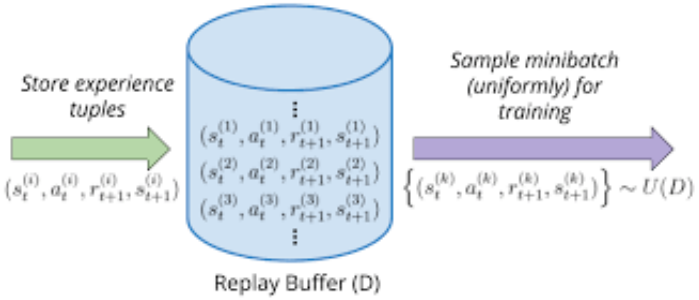


Figura 4.12 Buffer Replay.

Target Network

La correlación entre observaciones consecutivos también puede causar inestabilidad en el aprendizaje. Una vez más, al agente le resulta difícil distinguir estados sucesivos muy parecidos, y si la recompensa estimada es muy diferente, los parámetros de la red sufren una oscilación al actualizarlos en cada *time step*.

Por otra parte, en el algoritmo de Q-Learning estamos estimando $Q(s,a)$ con $Q(s\prime,a\prime)$ (bootsrapping). Si una actualización incrementa $Q(s,a)$, muy probablemente incremente $Q(s\prime,a)$ para todo a y por tanto aumente $Q(s\prime,a\prime)$. Asimismo, podemos a alterar estados cercanos y producir inestabilidad.

Los investigadores en [30] aliviaron este efecto con la introducción deliberada de un retraso entre el momento en que se produce la actualización de Q y el momento en que la actualización afecta a la función objetivo Q_{target}. Básicamente, se copian los parámetros de la red al inicializar el entrenamiento, θ^-, y sólo se actualizan con los parámetros de la red Q, θ, cada C pasos y se mantienen fijos entre las actualizaciones individuales. θ^- serán los parámetros usados para calcular Q_{target}, llamada ahora *target network*. En otras palabras. Q_{target} es ahora una versión desactualizada de Q durante C pasos, con esto se disminuye el efecto que tienen las actualizaciones recientes en la selección de acciones, con lo que se espera mejorar la estabilidad.

Si denominamos θ_i a los pesos de la red Q en la iteración(*time step*) i, $U(D)$ el conjunto de experiencias, La actualización de Q-learning en la iteración i utiliza la siguiente función de pérdida [30]:

$$L_i(\theta_i) = \mathop{\mathbb{E}}_{(s,a,r,s\prime)\backsim U(D)}[(r + \gamma * \max_{a\prime} Q(s\prime,a\prime;\theta_i^-) - Q(s,a;\theta_i))^2] \tag{4.26}$$

Sin embargo, se sabe que a veces en Q-Learning, se aprenden valores de acción poco realistas porque incluye un paso de maximización sobre los valores de acción estimados, que tiende a preferir los valores sobreestimados a los subestimados. Los autores de [46] muestran en ese misma publicación que incluso el algoritmo DQN a veces sobreestima sustancialmente los valores de las acciones.

La solución que propusieron, también en ese mismo artículo, fue crear otra red neuronal para que sea la target network. Esta variante la denominaron Double Deep Q-Network (Double-DQN).

$$Q_{target}(s,a) \approx Q_{target}(s,a;\theta^*) \tag{4.27}$$

El objetivo es el mismo que en [30], con esta segunda red se calculan la estimación de los valores de Q y se utilizan para retropropagar y entrenar a la Q-Network principal. Igualmente, se sincronizan

los parámetros de la target network, θ^*, periódicamente con los parámetros de la red neuronal principal. Cabe destacar que nunca se retropropaga en la target network, sólo en la principal.

La función de pérdida quedaría como sigue:

$$L_i(\theta_i) = \mathop{\mathbb{E}}_{(s,a,r,s\prime)\backsim U(D)}[(r + \gamma * \max_{a\prime} Q_{target}(s\prime,a\prime;\theta_i^*) - Q(s,a;\theta_i))^2] \tag{4.28}$$

El algoritmo Double-DQN, es el que usaremos en este trabajo y a continuación se muestra el pseudocódigo:

Algorithm 2 Algoritmo Double Deep Q-Network

Inicialización aleatoria de los pesos θ de la función $Q(s,a;\theta)$;
Copiamos los pesos θ de la función $Q(s,a;\theta)$ en θ en θ^* de $Q_{target}(s\prime,a\prime;\theta_i^*)$;
$Inicializamosepoch \leftarrow 0$;
while $epoch < epoch_{MAX}$ **do**
  $step \leftarrow 0$;
  Reseteamos el escenario y obtenemos el estado s;
  while $step < step_{MAX}$ **do**
    Tomamos una acción a mediante una política ε-greedy basada en $Q(s,a;\theta)$;
    Aplicamos a y adquirimos el nuevo estado del escenario s$\prime$ y la recompensa r;
    Guardamos $(s,a,r,s\prime)$ en el replay memory;
    if replay memory tiene más de N experiencias **then**
      experiencias $\leftarrow$ N experiencias $(s,a,r,s\prime)$ de replay memory;
      for $(s,a,r,s\prime)$ **in** expxeriencias **do**
        if hemos llegado al estado terminal **then**
          $y_i = r$
        else
          $y_i = r + \gamma * \max_{a\prime} Q_{target}(s\prime,a\prime;\theta_i^*)$
        end if
        Actualizamos los parámetros θ usando el algoritmo de descenso del gradiente para minimizar la función de pérdida: $L_i(\theta_i) = (y_i - Q(s,a;\theta_i))^2$
      end for
    end if
    $step \leftarrow step + 1$;
  end while
  Actualizamo θ^* cada n episodios: $\theta^* \leftarrow \theta$;
  $epoch \leftarrow epoch + 1$;
end while

Dueling Network

La arquitectura de red Dueling propuesta en [48], separa explícitamente la representación de los valores de estado y las ventajas de las acciones (dependientes del estado). Estos dos flujos comparten un módulo común de aprendizaje de características. Luego se combinan a través de una capa especial de agregación para producir una estimación de la función de valor estado-acción Q:

$$Q(s,a;\theta) = y[i,j] = V(s;\theta\prime) + A(s,a;\theta\prime\prime) \tag{4.29}$$

$V(s;\theta\prime)$ es la función de valor del estado actual y la función de ventaja(*advantage function*) $A(s,a;\theta\prime\prime)$ evalúa la recompensa esperada al realizar una acción a en un estado s con respecto a las demás acciones posibles.

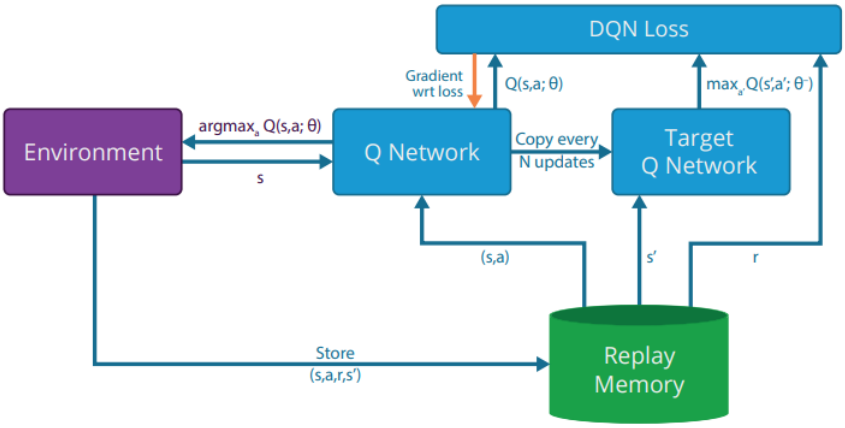


Figura 4.13 Esquema del algoritmo DQN.

En cambio, la Ecuación 4.29 no es identificable en el sentido de que dado Q no podemos recuperar V y A de forma única(*issue of identifiability*). En [48] sugirieron añadir una línea de base de A, que es el valor medio de A para cada posible a en el estado dado:

$$Q(s,a;\theta) = y[i,j] = V(s;\theta\prime) + (A(s,a;\theta\prime\prime) - \frac{1}{A}[\sum_{a\prime} A(s,a;\theta\prime\prime))] \tag{4.30}$$

Intuitivamente, la arquitectura Dueling puede aprender qué estados son (o no son) valiosos, sin tener que aprender el efecto de cada acción para cada estado. Esto es especialmente útil en los estados en los que sus acciones no afectan al entorno de ninguna manera relevante [48].

4.4 Ley de Recompensa

La recompensa que recibe el agente marca el comportamiento que él entiende como óptimo. Por ello, la ley de recompensa se debe de formular para guiar al agente al objetivo principal: cubrir todas las zonas del mapa, con mayor reiteración a zonas más relevante pero al mismo tiempo visitar las áreas que llevan tiempo sin visitarse. Como el lago está delimitado naturalmente por orillas, el agente también debe aprender a no colisionar con los límites del escenario.

Formalmente, una función de recompensa es una función matemática que mapea un estado y una acción tomada en una recompensa, que simboliza un premio o un castigo:

$$R_e : (s,a) \rightarrow R \tag{4.31}$$

Para diseñarlo hay que tener en cuenta no sólo las acciones buenas o malas, sino también la preferencia de una frente a otra. Adicionalmente hay que considerar no dar valores muy altos para que el agente no desarrolle fobias frente a acciones penalizadoras o se vuelva muy avaricioso. En la práctica, la función de recompensa se suele diseñar mediante prueba y error.

Como se adelantó en [54], se definirá una matriz de interés R_{abs}(i,j) que para cada (i,j) posee el valor de la importancia absoluta de esa zona a nivel estático, es decir, el interés de esa zona independientemente del tiempo. El rango de valores de esta matriz es de $[0,1]$, un valor de 0 significa que la importancia de la celda es nula y un valor de 1 significa que esta celda tiene la máxima importancia y debe ser visitada. En nuestro código desarrollado, este rango permite presentar visualmente esta matriz como un mapa de calorFigura 4.14.

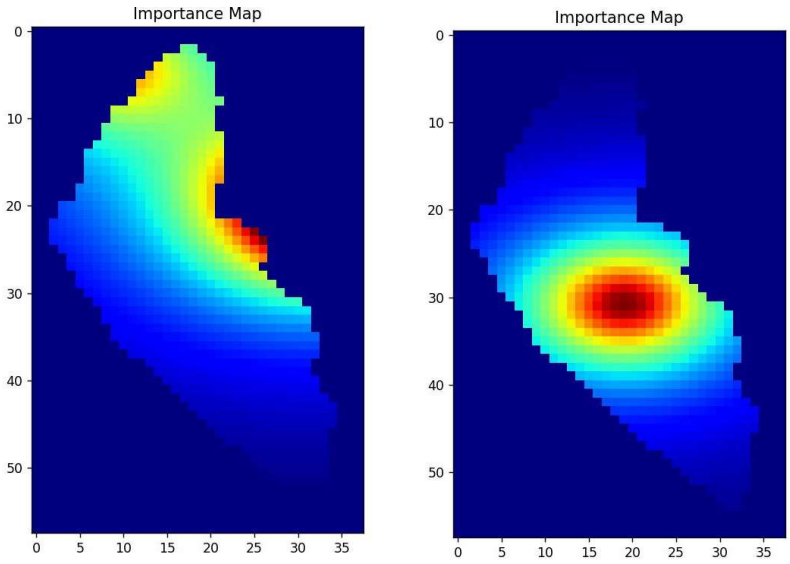


Figura 4.14 Ejemplo de matriz de interés R_{abs}(i,j) presentado visualmente como un mapa de calor.

Para ponderar cada celda dependiendo del tiempo que ha pasado sin visitarse, se utiliza la matriz W, el cuál comienza en su valor máximo al inicio. Si $[x,y]_{agent}$ son las coordenadas de la posición del agente y $W(i,j)^t$ el valor de las casillas de la región de deteccion de los sensores del robot en $[x,y]_{agent}$:

$$W = \begin{cases} W(i,j)^{t+1} = \text{mín}[W(i,j)^t + \delta, 1] & \text{si } [x,y]_{agent} \neq [i,j] \\ W(i,j)^{t+1} = 0 & \text{si } [x,y]_{agent} = [i,j] \end{cases} \tag{4.32}$$

Cuando una casilla se visita, el valor de las casillas de la región de detección en W se pasan a 0. A medida que pasa el tiempo recupera su ponderación hasta llegar a un máximo de 1. El parámetro δ indica cuandos time steps tienen que pasar hasta que una celda recupere su valor máximo. Por ejemplo, si δ vale 0.05, y esa celda no ha sido visitada, la ponderación se actualiza durante 20 pasos hasta que se recupere al completo. Así, aunque una casilla tenga un interés muy alto, este se anula al visitarlo y las mayores recompensas se encontrarán en casillas visitadas hace mucho tiempo. Esto favorece a la resolución del patrolling problem ya que empuja al agente a, por lo menos durante un instante, volver a zonas no frecuentadas.

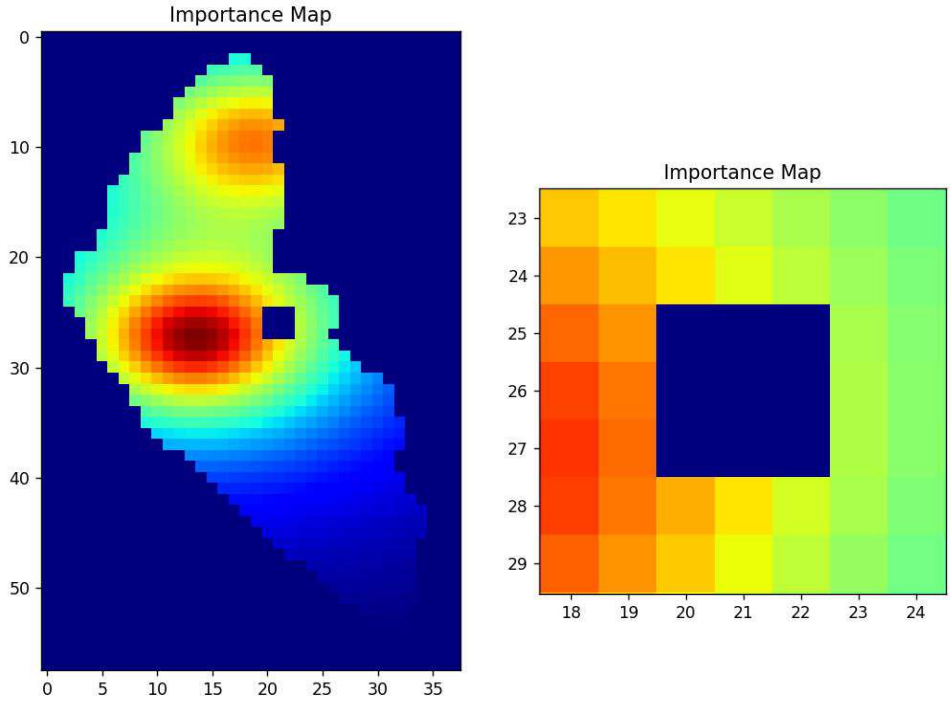
La matriz de importancia relativa, R, será el producto elemento a elemento(producto Hadamard) de las matrices W y R_{abs}(i,j)

$$R = W \circ R_{abs} \tag{4.33}$$

La matriz R puede representarse como un mapa de calor en el que las zonas más frías indican un bajo interés relativo y las más calientes representan un interés relativo alto.

Examinar una región del mapa varias veces afianza la información obtenida en visitas anteriores(redundancia útil), pero se sabe que en el problema del patrullaje es importante no estar enfocado en sólo un lugar.

Para evitar que el vehículo visite una zona con una frecuencia que afectaría a la optimalidad del patrullaje, se resta el valor estático de interés de una celda cada vez que esta es visitada. Para ello, se define el parámetro atrición (v), que simboliza el porcentaje de interés que pierde una casilla en cada visita. Si la atrición de una casilla con mucho interés vale 0.10, entonces se tiene que visitar 10 veces para que pierda por completo su interés frente a las demás.



(a) Mapa de importancia en la posición inicial

(b) Zoom en la región de interés del mapa de importancia en la posición inicial

Figura 4.15 Región de detección del vehículo en la posición [26, 21] con radio r=2. Se observa como en este instante el interés de las celdas de dicha región es 0..

Los sensores del ASV tienen un rango de detección que les permite medir hasta un cierto radio estando en una posición. Esto se modela en nuestro código definiendo el valor de esta distancia de detección como el radio que define el tamaño de una máscara cuadrada que representa el área de alcance los sensores en el grid map. Si definimos x como las coordenadas en el eje x de las casillas ocupables del grid map, e y como las coordenadas en el eje y, [i,j] la posición del ASV y r la distancia de detección, entonces la máscara ω se define como:

$$\omega(i,j) = (x-i)^2 + (y-j)^2 < r^2 \tag{4.34}$$

ω es una matriz binaria que vale 1 en las posiciones en las que se cumple la Ecuación 4.35. Dicho esto, podemos definir la suma de interés relativo que se recolecta en una posición (i,j) como:

$$\rho(i,j) = \sum_{m=0}^{M} [\sum_{n=0}^{N} \omega(m,n) \times R(m,n)] \text{ Siendo NxM el tamaño del escenario} \tag{4.35}$$

Este valor se puede normalizar para que no dependa del valor de distancia de detección:

$$\rho(i,j)_N = \frac{\rho(i,j)}{\pi * r^2} \tag{4.36}$$

La Ecuación 4.36 define la recompensa inmediata que obtendrá el agente en la posición (i,j) con un radio de alcance de los sensores de r. Este valor será mayor cuanto más interés haya en esa zona, esto es deseable ya que esas zonas representan las regiones más contaminadas. En cambio, a medida que se repasa varias veces un espacio, este perderá interés de manera temporal y por tanto las celdas con menos interés, pero con más tiempo sin ser visitados, tendrán mayor recompensa.

Se considera que una acción es ilegal cuando la posición a la que intenta dirigirse forma parte de la zona no transitable, es decir, fuera del lago (casillas de color negro en la Figura 3.3). Para penalizar una acción ilegal se emplea el parámetro C_{ilegal} cuyo valor cuantifica la penalización de este tipo de acciones. Es importante no utilizar un valor muy alto que haga que el agente tenga fobia a los bordes del lago, los cuáles pueden estar muy contaminados.

En Ecuación 4.37 se resume la ley de recompensa:

$$S = \begin{cases} S^{t+1} = \frac{\sum_{m=0}^{M}[\sum_{n=0}^{N}\omega(m,n)\times R(m,n)]}{\pi * r^2} & \text{si } la\ accin\ es\ legal \\ S^{t+1} = -C_{ilegal} & \text{si } la\ accin\ no\ es\ legal \end{cases} \tag{4.37}$$

4.4.1 Recompensa a la exploración

La función de recompensa propuesta en la anterior sección no favorecía explícitamente a la exploración. Es decir, aunque con la matriz W los picos de interés desaparecían durante un cierto tiempo y las recompensas mayores se situasen en casillas poco visitadas, una mal elección de parámetros puede hacer que esto no siempre se cumpla. El objetivo del agente siempre va a ser obtener la mayor recompensa posible, Por ejemplo, si el valor de recuperación δ es muy grande, el comportamiento óptimo del agente sería quedarse siempre cerca de los picos de interés porque se regeneran muy rápido. Esto, por supuesto, no es una buena solución al problema del patrullaje.

En el caso del estado parcialmente observable, este problema es más evidente. Puesto que, como el agente ni conoce toda la información del mapa, una vez que haya encontrado un pico de interés, no se esfuerza en explorar más. Surge entonces la idea de añadir un término en la Ecuación 4.37 que empuje al agente a la exploración.

Primero, debemos crear una matriz que contenga la información de las casillas visitadas(*Visited*), y una matriz que guarde las nuevas celdas visitadas en un paso(*NewVisited*). Esta última, es una matriz binaria que vale 1 en las nuevas casillas descubiertas.

Por último, se ha creado un parámetro que pondera el valor que se le atribuye a descubrir nuevas zonas, que se llamará *discovery reward* denotado como λ. El nuevo término que recompensa explícitamente a la exploración es el siguiente:

$$\psi(i,j) = \sum_{m=0}^{M}[\sum_{n=0}^{N}\lambda \times NewVisited(m,n)] \text{ Siendo NxM el tamaño del escenario} \tag{4.38}$$

$$\psi(i,j)_N = \frac{\psi(i,j)}{\pi * r^2} \text{ Normalizado frente al área de detección} \tag{4.39}$$

El valor del parámetro *discovery reward* determina como se va a comportar el agente en cuanto a si preferirá quedarse en los picos de interés o explorar más, porque ahora si que se obtiene bastante recompensa de esa manera. Retomando nuestro objetivo principal, se quiere conseguir un balance entre explorar y la intensidad del patrullaje (ir a zonas de interés alta). Por lo tanto, se estudiará el efecto de este parámetro, entrenando para varios valores de este en el escenario parcialmente observable.

La nueva ley de recompensa se resume en la Tabla 4.1:

Tabla 4.1 Resumen de la ley de recompensa.

Ley de recompensa	
$S = \begin{cases} S^{t+1} = \dfrac{\rho(i,j) + \psi(i,j)}{\pi * r^2} & \text{si } la\ accin\ es\ legal \\ S^{t+1} = -C_{ilegal} & \text{si } la\ accin\ no\ es\ legal \end{cases}$	(4.40)
Recolección de interés	$\rho(i,j) = \sum_{m=0}^{M} [\sum_{n=0}^{N} \omega(m,n) \times R(m,n)]$ (4.41)
Recompensa a visitar nuevas zonas	$\psi(i,j) = \sum_{m=0}^{M} [\sum_{n=0}^{N} \lambda \times NewVisited(m,n)]$ (4.42)

4.5 Representación del estado

El estado es la representación de la información disponible del escenario para el entrenamiento. En otros términos, el estado define como se encuentra el entorno y como nos encontramos respecto a él. Por tanto, La forma en la que se representa afecta al rendimiento del proceso de aprendizaje porque define qué información está disponible para que el agente aprenda a buscar acciones óptimas. En nuestro enfoque, esto se traduce a optimizar la función acción-estado Q.

Vamos a utilizar el marco matemático del MDP para formalizar los elementos del problema de RL y por tanto el estado. Para poder utilizar el MDP en un problema se debe cumplir la propiedad de Markov: el futuro depende solo del presente y no del pasado. Esto requiere que el entorno no tenga memoria, lo que no es adecuado para todos los casos reales y por supuesto el nuestro.

Como se expuso en [54], se pretende tratar con la propiedad de Markov con una información completa. Esto es, el estado contiene toda la información de los *time step*s anteriores, de modo que el estado agrupa la información de las acciones pasadas sin que queden dependencias temporales.

4.5.1 Estado del Entorno completamente observable

En este escenario se proporciona al algoritmo toda la información disponible en el escenario. Para ello, el estado será representado por tres imágenes:

- Posición del agente: Imagen binaria que vale cero en todos las celdas excepto en la casilla que se encuentra el agente.
- Mapa del lago: grid map de la superficie del lago Ypacaraí y el territorio que lo rodea. La superficie navegable serán representados con valor 1 y el territorio adyacente de tierra como 0's. Para generar el gridmap se ha creado un programa en Python que, con ayuda de la biblioteca OpenCV permitirá tomar de un mapa la forma de una masa de agua y mediante una ajuste manual tomar solo la superficie navegable discretizada. El resultado se almacenará como un fichero CSV de ceros y unos.
- Importancia relativa: En la Sección 4.4 se presentó la matriz de importancia relativa R

  A medida que el agente interactúa con el entorno, las matrices W y R_{abs}(i,j) se irán actualizando y modificando(en cada *time step*).

1. Las celdas de las que el agente recolecta información(máscara ω(Ecuación 4.35)) pasan a valer 0 en W. Sea i el *time step* en el que nos encontramos:

$$W_i = W_{i-1} - \omega \circ W_{i-1} \tag{4.43}$$

   De la anterior ecuación, se acotan a 0 algunos valores que pueden resultar menor que 0.

2. El mapa de interés estático R_{abs} se ve afectado por el parámetro attrition(ν), que simboliza el porcentaje de interés que pierde una casilla en cada visita. Llamamos $\hat{R_{abs}}$ al mapa de interés resultante

$$\hat{R_{abs}} = \hat{R_{abs}} - \nu * \omega \circ R_{abs} \tag{4.44}$$

   De la anterior ecuación, se acotan a 0 algunos valores que pueden resultar menor que 0.

3. La última imagen del estado será entonces la matriz de interés ponderada:

$$R = W \circ \hat{R_{abs}} \tag{4.45}$$

4. Por último, en cada *time step* las casillas de W recuperan su ponderación con el parámetro δ hasta llegar a un máximo de 1:

$$W_i = W_i + \delta * W_i \tag{4.46}$$

   De la anterior ecuación, se acotan a 1 algunos valores pueden resultar mayor que 1.

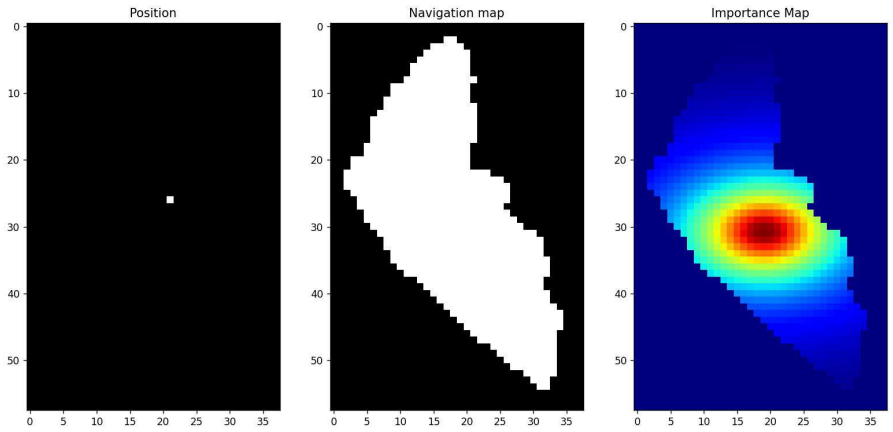


Figura 4.16 Estado del Escenario Completamente Observable. De izquierda a derecha: Posición del agente, Mapa del lago e Importancia relativa..

4.5.2 Estado del Entorno parcialmente observable

En este escenario se proporciona al algoritmo sólo la información de las zonas visitadas en el escenario. Para ello, el estado será representado por cuatro imágenes:

- Posición del agente: Imagen binaria que vale cero en todos las celdas excepto en la casilla que se encuentra el agente.
- Mapa del lago: grid map de la superficie del lago Ypacaraí y el territorio que lo rodea. La superficie navegable serán representados con valor 1 y el territorio adyacente de tierra como 0's.
- Matriz de importancia: Como puede que todavía no se haya visitado todo el mapa, se muestra en una imagen el mapa de interés de las casillas descubiertas. El procedimiento es el siguiente:

  1. Se crea un mapa de las casillas visitadas *visited_map*

  $$visited\_map = \begin{cases} \omega & \text{si primer } time\ step \\ visited\_map + \omega & \text{si ya no es el primer } time\ step \end{cases} \tag{4.47}$$

  2. Calculamos $\hat{R_{abs}}$ como en la Subsección 4.5.1.

  3. La imagen del estado será entonces el interés descubierto:

  $$visited\_map \circ \hat{R_{abs}} \tag{4.48}$$

- Mapa temporal: En esta imagen se muestra la ponderación de cada celda dependiendo del tiempo que ha pasado sin visitarse. Se pretende que la importancia temporal de la información de las últimas celdas visitadas sean superiores, es decir que valga 1.

  1. Las celdas de las que el agente recolecta información(máscara ω(Ecuación 4.35)) pasan a valer 0 en W. . Sea i el *time step* en el que nos encontramos:

  $$W_i = W_{i-1} - \omega \circ W_{i-1} \tag{4.49}$$

  De la anterior ecuación, se acotan a 0 algunos valores que pueden resultar menor que 0.

  2. La matriz W_i ya pondera cada casilla, pero las casillas más recientemente visitadas valen 0. Si llamamos al mapa del escenario *scenario_map*:

  $$(1 - W_i) \circ * scenario\_map \tag{4.50}$$

  La Ecuación 4.50 es la cuarta imagen del estado, en ella, cuanto más reciente ha sido la visita de una casilla, mayor importancia tendrá, justo al contrario que W_i, de ahí la resta. Los lugares nunca visitados o las zonas examinadas hace mucho tiempo valdrán 0. Se multiplica por el mapa del escenario para hacer 0 los lugares no visitables.

  3. Por último, en cada *time step* las casillas de W recuperan su ponderación con el parámetro δ hasta llegar a un máximo de 1:

  $$W_i = W_i + \delta * W_i \tag{4.51}$$

  De la anterior ecuación, se acotan a 1 algunos valores pueden resultar mayor que 1.

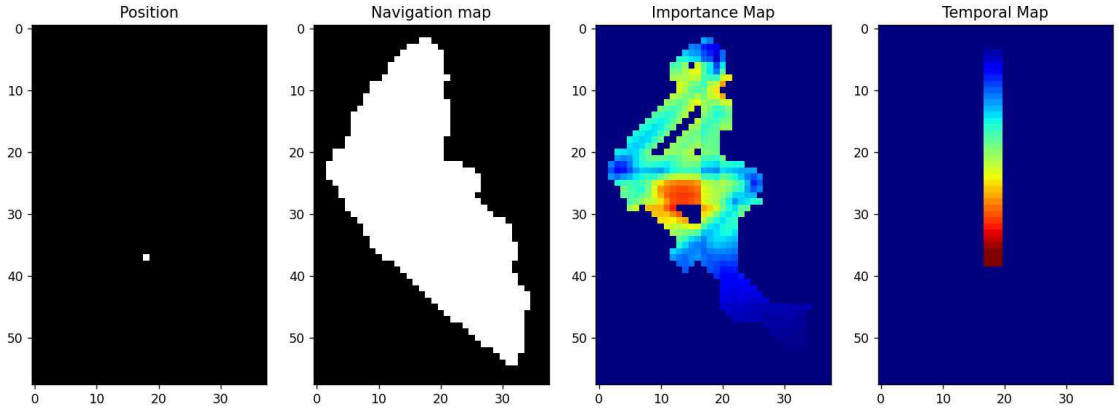


Figura 4.17 Estado del Escenario Parcialmente Observable. De izquierda a derecha: Posición del agente, Mapa del lago, Matriz de importancia y Mapa temporal.

4.6 Nuestro agente

4.6.1 Red neuronal del agente

La red neuronal que intentaremos optimizar es una red convolucional densa con arquitectura Dueling, por lo tanto el agente será un Agente Dueling-DQN.

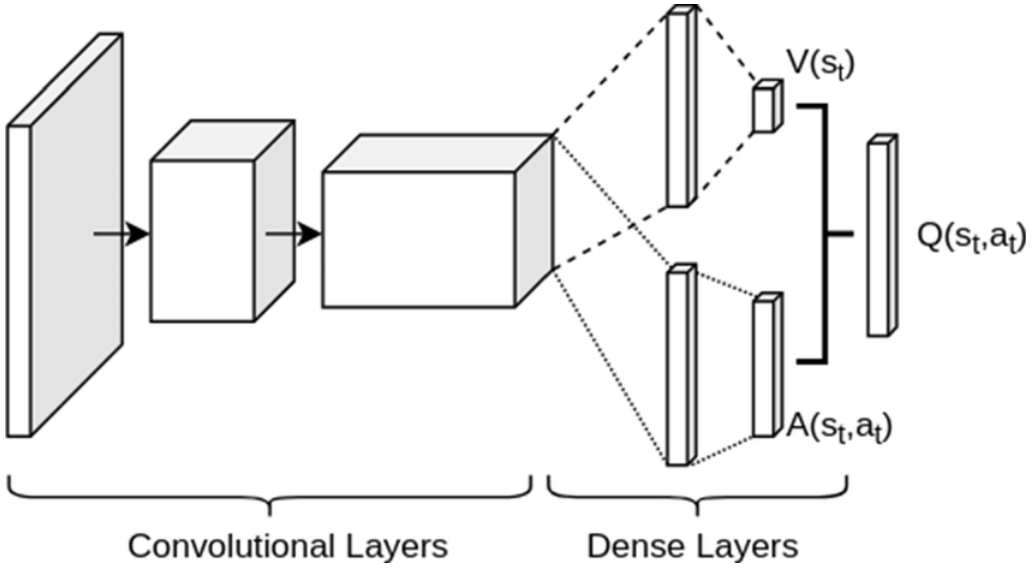


Figura 4.18 Dueling DQN.

La red convolucional densa está formada por varias capas:

1. **Feature extractor**: Capas convolucionales, al final de estas capas se sacan 1024 features o características de las imágenes de entrada:
   a) Capa de entrada que recibe las imágenes del estado y devuelve una salida de 64 canales.
   b) Función de activación Leaky ReLu..
   c) Recibe las imágenes de la capa anterior y devuelve una salida de 32 canales.
   d) Recibe las imágenes de la capa anterior y devuelve una salida de 16 canales.
   e) Función de activación Leaky ReLu.
   f) Capa *flatten*, que sirve para unidimensionalizar la entrada multidimensional y se usa como transición a la capa totalmente conectada
2. Capa lineal que recibe la salida de *Feature extractor* y retorna una salida de tamaño 256
3. Función de activación Leaky ReLu.
4. Capa lineal 256x256
5. Función de activación Leaky ReLu.
6. Capa lineal 256x256
7. Función de activación Leaky ReLu.

La salida de estas capas son las entradas a dos flujos distintos de capas:

1. Capas densas lineales que estima el valor del estado actual y por lo tanto la dimensión de salida es 1
2. Capas densas lineales que estima el valor de la *advantage function* y por lo tanto la dimensión de salida es el número de acciones

Por último, se aplica la Ecuación 4.30 a la salida de estos dos últimos flujos y se obtiene el valor estimado de la función Q.

Como mejora, en el último se ha añadido una capa de Maxpool a cada salida de las capas convolucionales para disminuir los parámetros de la red y poder entrenar mas rápido

4.6.2 Programación del agente

El Agente Dueling-DQN ha sido programado en el fichero *DuelingDQNAgent.py* en la clase DuelingDQNAgent. En ella se implementa todo lo necesario para el aprendizaje del agente siguiendo el algoritmo Double-DQN

Los parámetros de entrada destacables de esta clase son las siguientes:

- ***env***: el escenario del que aprenderá el agente e intentará optimizar la red para conseguir las mayores recompensas.
- ***memory size*** Tamaño del buffer de experience replay.
- ***batch size*** Tamaño de los lotes usados para la optimización de la red. La elección de un tamaño de mini-lotes de potencias de 2 ayuda porque estos van a ser vectorizados y procesados paralelamente en la GPU.
- ***target update*** Número de episodios entre la actualización de los parámetros de la red target con la red principal.

- ***gamma*** Factor de descuento γ.
- ***number of features*** Número de features o características de las imágenes que debe extraer la capa convolucional.
- ***epsilon values*** Tupla que sirve para definir el intervalo de valores de epsilon durante el aprendizaje.
- ***epsilon interval*** Tupla que define el porcentaje del entrenamiento en el que decrece el valor de epsilon hasta el mínimo.
- ***learning rate*** Hiperparámetro α, al que habíamos denominado *step size*.
- ***max pool*** Parámetro booleano que vale True si queremos aplicar Max-pooling a la salida de las capas convolucionales, si no, vale False.
- ***train every*** Se optimiza la red cada train every steps, en nuestro caso 15. Esto hace que tarde menos el entrenamiento.
- ***save every*** Se guarda la política de la red cada save every episodio, en nuestro caso 2000. Esto lo hacemos para estudiar el avance del aprendizaje.

Para la optimización se ha elegido el algoritmo de optimización de descenso del gradiente Adam(Adaptive Moment Estimation). La estimación de momento adaptativo es un algoritmo de tasa de aprendizaje adaptativo que sigue un conjunto de estrategias heurísticas para una mayor rapidez en el proceso de optimización.

La función de pérdida a usar para la regresión es el error cuadrático medio, en concreto la Ecuación 4.28.

4.6.3 Colisiones

Es vital para nuestro agente aprender a identificar la orilla del lago y evitar tomar acciones que le lleven a colisionar con esta. Es por ello que este tipo de acciones ilegales son penalizadas en la ley de recompensa con el parametro C_{ilegal}. Sin embargo, cuantas más veces se dé esta situación durante en el entrenamiento, más rápido aprenderá a reconocerla y evitar colisiones. Hemos configurado los escenarios para que un episodio se termine después de un número predeterminado de colisiones. Si permitimos más de una colisión por episodio, el buffer de experiencias se llenará de experiencias de colisión al principio y por tanto se aprenderá primero a esquivar los bordes. Si solo permitimos una colisión, entonces solo habrá episodios muy malos y cortos y aprenderemos más lentamente. Si el entrenamiento dura 20.0000 episodios y sólo se deja que haya 1 colisión por partida, el buffer que tiene 1.000.000 de estados sólo verá, como máximo, un 2 % de movimientos de colisión. Si añadimos más colisiones, conseguiremos aprender más controlando la cantidad de experiencias de replay.

5 Resultados

En este Capítulo se analizan los resultados de los diversos experimentos llevados a cabo. En ellos se ha estudiado el comportamiento del agente en las siguientes circunstancias:

- Escenario Completamente Observable : Análisis del problema del vigilante con la función de recompensa sin premio a la exploración.
- Escenario Parcialmente Observable: Análisis del problema del vigilante con la función de recompensa sin premio a la exploración.
- Escenario Parcialmente Observable: Análisis del problema del vigilante con la función de recompensa con premio a la exploración. Con hincapié en estudiar el efecto del parámetro *discovery_reward*.
- Comparación de la performance del algoritmo frente a otros algoritmos basados en heurísticas.

El código y resultados pueden encontrarse en el repositorio de GitHub *https://github.com/derpberk/DRLPatrollingProblemYpacarai*

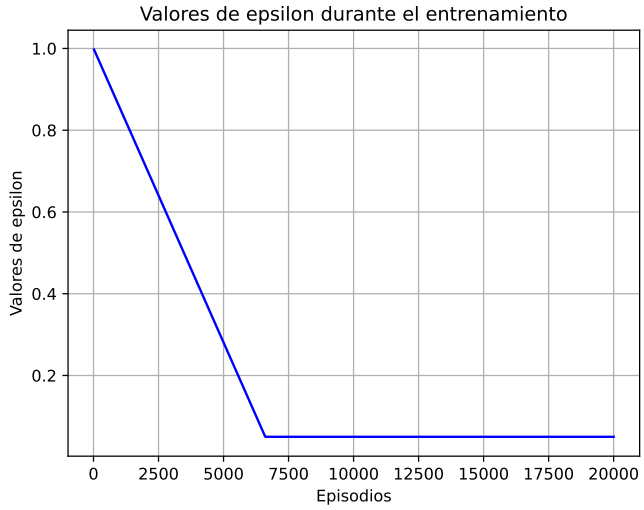
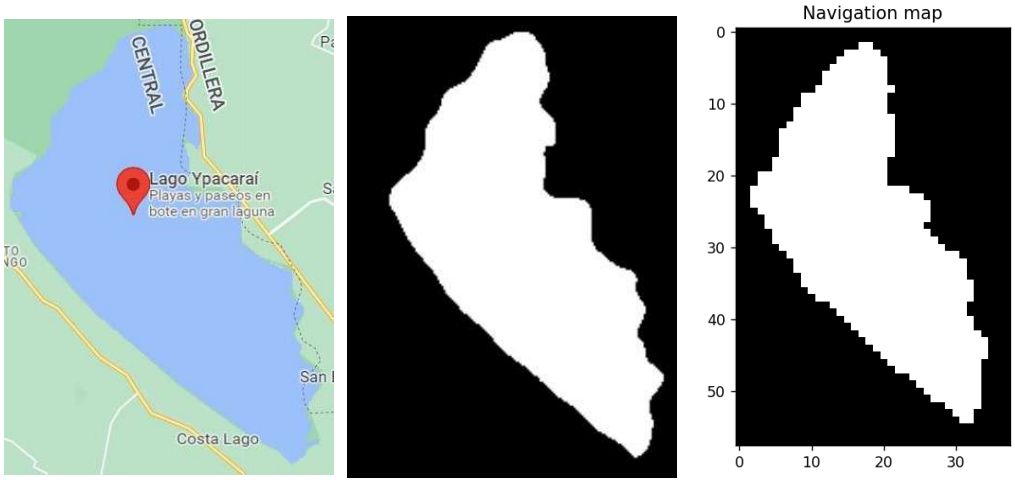


Figura 5.1 Valores de ε durante el entrenamiento.

Los hiperparámetros de entrenamiento se muestran en la Tabla 5.1.En la Figura 5.1 se muestra la evolución de épsilon a lo largo del entrenamiento. Como se puede apreciar , va a ir de 1 a 0.1 en el intervalo del 0 % del entrenamiento al 33 %. A partir del cuál epsilon es 0.1 para explotar el conocimiento.

Tabla 5.1 Hiperparámetros de entrenamiento.

Hiperparámetros	Valores	Comentarios
Batch size	64	Con un tamaño pequeño la estimación del gradiente en cada época es más ruidosa, pero ayuda al algoritmo a evitar los mínimos locales.
Replay memory size	100000	Cuanto mayor sea este parámetro, menos posible es sufrir *catastrophic forgetting*, pero podría ralentizar el entrenamiento si ocupa demasiada memoria.
Target update	1000	
$\varepsilon - value$	[1.0,0.1]	Elegimos 0.1 como el valor mínimo de ε para que siempre exista una probabilidad de encontrar acciones mejores.
$\varepsilon - interval$	[0,0.33]	El valor de ε decrece sólo en el primer tercio del entrenamiento porque queremos que el agente sea muy explotativo
γ	0.99	
learning rate	1e-4	
Train every	15	Se optimiza la red cada 15 steps. Así que de media se ejecutan 20-30 pasos cada partida. Esto reduce el tiempo de entrenamiento y cada experimento de 20.000 episodios tarda unas 2 horas y media.
Save every	2000	Comparamos 10 políticas generadas cada 2000 episodios a lo largo de los 20.000 episodios de entrenamiento.



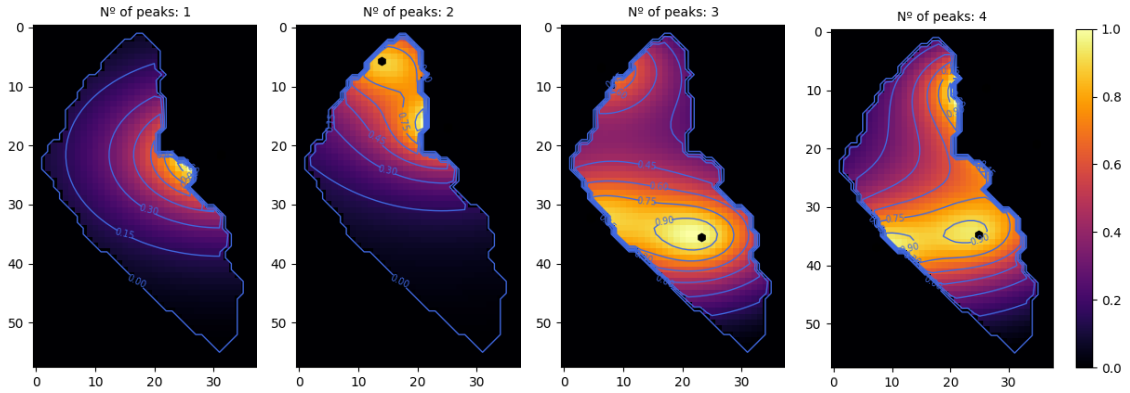
(a) Mapa satélite del lago (b) Mapa binarizado (c) Mapa discretizado

Figura 5.2 De izquierda a derecha los distintos pasos que se han hecho para discretizar el mapa del lago..

5.1 Generación de los mapas

Para generar el grid-map del lago Ypacaraí se ha tomado una imagen de satélite del lago y se ha creado un programa en Python que, con ayuda de la biblioteca OpenCV, toma solo la superficie navegable discretizada de la imagen según una relación de resolución espacial configurable.

En primer lugar, se aplica un filtro por umbral de la imagen, que binariza la imagen construyendo dos segmentos: la superficie del lago(en blanco) y el fondo de la imagen(en negro), que es el terreno alrededor del lago. La asignación de píxeles (0 ó 1) se consigue comparando su tono, color y brillo



(a) (b) (c) (d)

Figura 5.3 Cuatro ejemplos de las funciones aleatorias de Shekel tomadas como *ground-truth*.

concreto (valor plano) con el valor umbral. Finalmente, tras un filtro de mediana para corregir posible ruido, se ha aplica un escalado y se ha tomado que 1 píxel de en el mapa escalado se corresponda a 0.125 metros (Figura 5.2).

Los parámetros de calidad del agua se modelan mediante la optimización funciones de referencia, en nuestro caso la función Shekel. La función real de los mapas de calidad del agua, como la temperatura, el pH, niveles de turbidez, será suave debido al comportamiento real de la dinámica de los fluidos y las condiciones del viento. La función Shekel es una función multidimensional, multimodal, continua y determinista que suele utilizarse como función de prueba para comprobar las técnicas de optimización. Por tanto, consiste en un conjunto de picos suaves, similares a cómo se comportan las variables de calidad del agua en los lagos y mares. La forma matemática de una función en n dimensiones con m máximos es:

$$f(\vec{x}) = \sum_{i=0}^{m} \left(\sum_{j=0}^{n} \left(x_j - a_{ij}\right)^2 + c_i \right)^{-1} \tag{5.1}$$

La función Shekel tiene m máximos locales n-dimensionales, C es un vector de m dimensiones, y A es una matriz de m por n dimensiones. Los elementos de las matrices C y A son respectivamente c_i y a_{ij}. La matriz A define el número y las posiciones de las localizaciones máximas y en la matriz C su respectiva importancia inversa c_i.

5.2 Métricas de funcionamiento

Se han definido las siguientes métricas de funcionamiento para valorar la bondad del experimento:

- **Suma de interés**: ω es una matriz binaria que vale 1 en las posiciones en las que el agente recoge información. Dicho esto, podemos definir la suma de interés relativo que se recolecta en una posición (i,j) en el *time step* t como:

$$\rho(i,j)_t = \sum_{m=0}^{M} [\sum_{n=0}^{N} \omega(m,n) \times R(m,n)] \tag{5.2}$$

  Siendo NxM el tamaño del escenario y R la matriz de importancia relativa del escenario

  La suma de interés de todo el episodio es la suma de los valores de $\rho(i,j)_t$ en todos los *time steps*:

$$SOI = \sum_{t=0}^{T} \rho(i,j)_t \tag{5.3}$$

- **Importancia relativa media del escenario**: Para evaluar si el algoritmo está repasando con mayor frecuencia las zonas con mayor interés, se hace una media de la importancia relativa de las celdas en cada *time step* y nos quedamos con el valor mínimo que se ha conseguido alcanzar. Llamamos a la matriz que contiene el mapa del escenario *SM*, el cuál vale 0 en la casillas no visitables y vale 1 en las casillas visitables.

$$\text{mín}\, R_t = \frac{(W \circ R_{abs})}{\sum_{m=0}^{M} \sum_{n=0}^{N} SM(m,n)} \tag{5.4}$$

- **Porcentaje visitado del mapa**: Para saber si el algoritmo está visitando, al menos una vez todas las casillas del mapa, se evalúa en cada entrenamiento el porcentaje de la superficie del lago transitable que ha visitado el vehículo. Llamamos a la matriz que registra las casillas visitadas *VM*, el cuál vale 0 en la casillas no visitadas y vale 1 en las casillas ya visitadas:

$$PV = \frac{\sum_{m=0}^{M} \sum_{n=0}^{N} VM(m,n)}{\sum_{m=0}^{M} \sum_{n=0}^{N} SM(m,n)} \tag{5.5}$$

- **Porcentaje recompensa a la exploración**: En el último entrenamiento se busca empujar al agente a explorar. Para estudiar qué valor de discovery reward es el mejor y cuánto afecta a la función de recompensa se añade otra métrica. $\psi(i,j)$ es el término que premia explícitamente a la exploración en la función de recompensa. En cada *time steps* t, este valdrá:

$$\psi(i,j)_t = \frac{\sum_{m=0}^{M} [\sum_{n=0}^{N} \lambda \times NewVisited(m,n)]}{\pi * r^2} \quad \lambda = \text{discovery reward} \tag{5.6}$$

  Siendo r el hiperparámetro *longitud de detección*, *NxM* el tamaño del escenario, NewVisited una matriz binaria que guarda las nuevas celdas visitadas en un paso y que vale 1 en las nuevas casillas descubiertas.

  Si la recompensa acumulada al final del episodio es denotado como A, entonces el porcentaje debido a la exploración es:

$$PE = \frac{\sum_{t=0}^{T} \psi(i,j)_t}{A} \tag{5.7}$$

5.3 Escenario Completamente Observable

El primer entrenamiento del agente fue llevado a cabo en el *Escenario Completamente Observable*. Se realizaron 20000 episodios con los siguientes parámetros de entrada:

Tabla 5.2 Configuración del Escenario Completamente Observable.

Hiperparámetros	Valores
Presupuesto de la batería	100
Longitud de detección	2
Distribución de información aleatoria	True
Número de kilómetros	100
Recuperación	0.03
Atrición	0.05
Colisiones permitidas	True
Número de colisiones permitidas	20

A continuación se estudian algunas métricas del problema que se consideran indicativos de un buen entrenamiento e interesantes de cara a la puesta en práctica de nuestro algoritmo. En las figuras de estas métricas se muestran superpuestas: los valores reales de las métricas y un filtrado exponencial de media móvil(en inglés EMA) con $\alpha = 0.05$ para poder ver con claridad la tendencia de los valores.

5.3.1 Evolución de las métricas

Recompensa acumulada

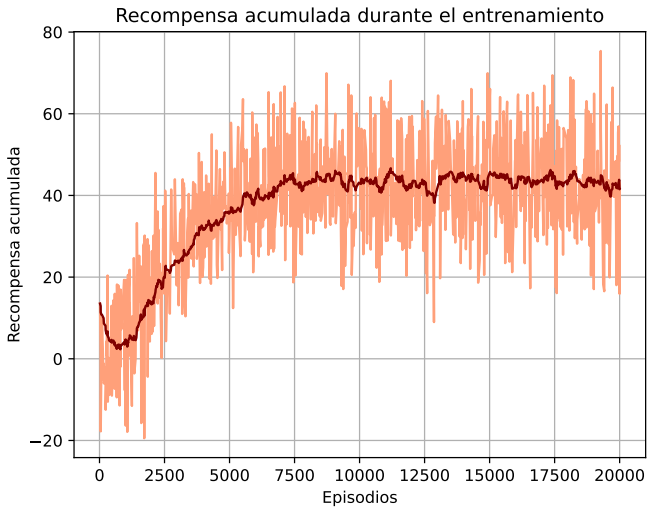


Figura 5.4 Valores de la recompensa acumulada durante el entrenamiento.

En la Figura 5.4 se puede observar como el agente ha ido aprendiendo a aumentar la media de la recompensa en el tramo decreciente de $\varepsilon - greedy$. En cambio, en el tramo explotativo, no solo aprende mejorando su recompensa media sino que aprende a mejorar su performance en nuevos escenarios.

Longitud de los episodios

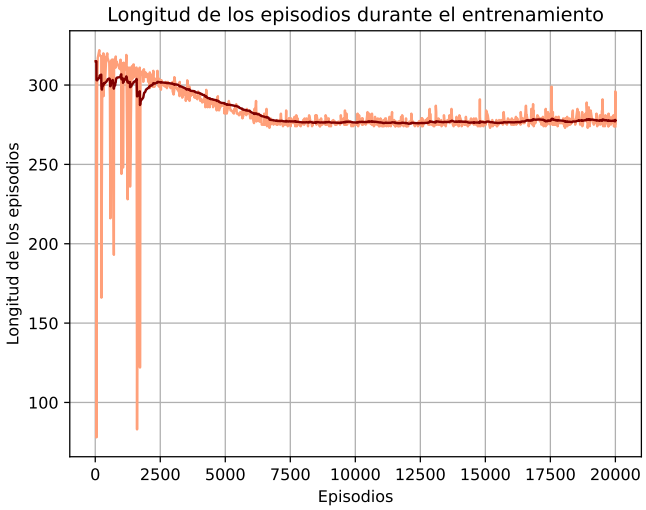


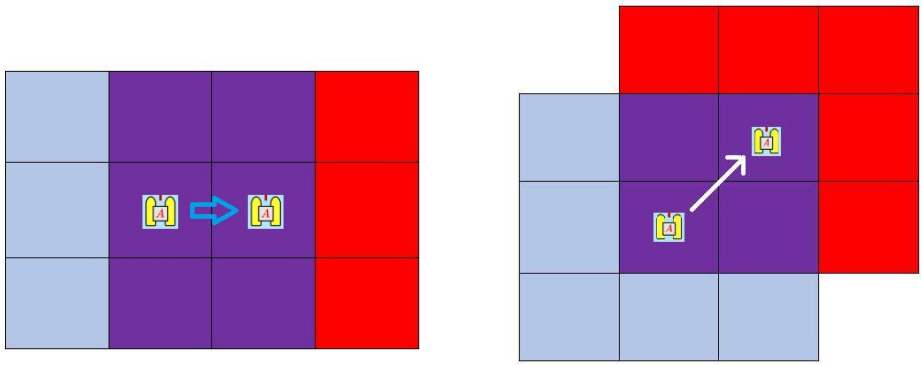
Figura 5.5 Longitud de los episodios durante el entrenamiento.

En la Figura 5.5 se puede observar como el agente ha tardado unos 2500 episodios en aprender por completo a no colisionarse. También se aprecia la disminución de la longitud de los episodios en el tramo de exploración. A la misma vez, la recompensa acumulada aumenta, como se comentó anteriormente. Entonces, cada vez está recolectando más información en menos tiempo.

La reducción de la duración de los episodios es debido a que cada vez dura menos la batería. Teniendo en cuenta que una traslación en diagonal gasta $\sqrt{2}$ veces más de batería que una traslación horizontal o vertical, se ve que cada vez el algoritmo prefiere desplazarse en diagonal.

En la Figura 5.6 se aprecia que en un movimiento diagonal se recoge información de 5 casillas nuevas (celdas rojas), mientras que en un movimiento horizontal sólo se descubre la información de 3 nuevas celdas. Por lo tanto, el desplazamiento en diagonal es más rentable tanto para acumular recompensa como para descubrir zonas no visitadas

En conclusión, el agente ha ido descubriendo que moviéndose en diagonal recoge más información del entorno que trasladándose en vertical u horizontal.



(a) Movimiento horizontal (b) Movimiento diagonal

Figura 5.6 Movimientos del vehículos.

Media de interés relativa

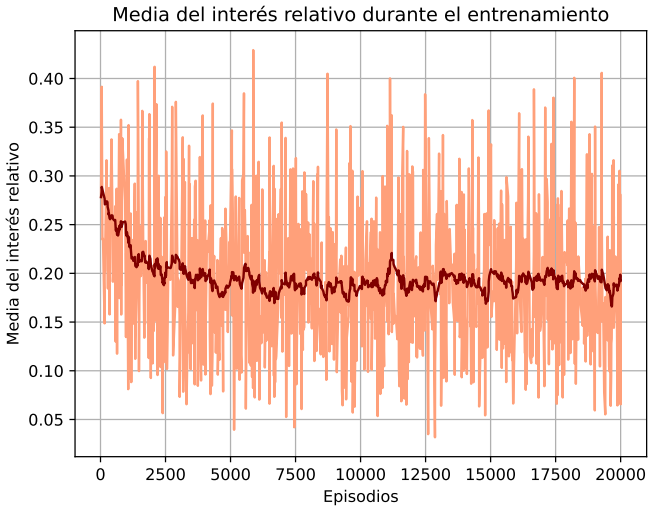


Figura 5.7 Media de interés relativa.

Como el objetivo de nuestro algoritmo es también minimizar el *idleness* de cada casilla se estudia como ha ido mejorando la capacidad de la red de resolver esto. En la Figura 5.7 se ve como se ha ido reduciendo la media de la importancia relativa de las casillas, la cuál no ha mejorado en el tramo explotativo del aprendizaje aunque no se ve una tendencia creciente. Además, a lo largo del tramo explorativo también se explota, lo cuál es suficiente. Cabe pensar que 20.000 son demasiados episodios de entrenamiento, pero esto no es perjudicial si no se manifiestan sus efectos negativos: *overfitting* y/o la degradación de la red.

Función de pérdida

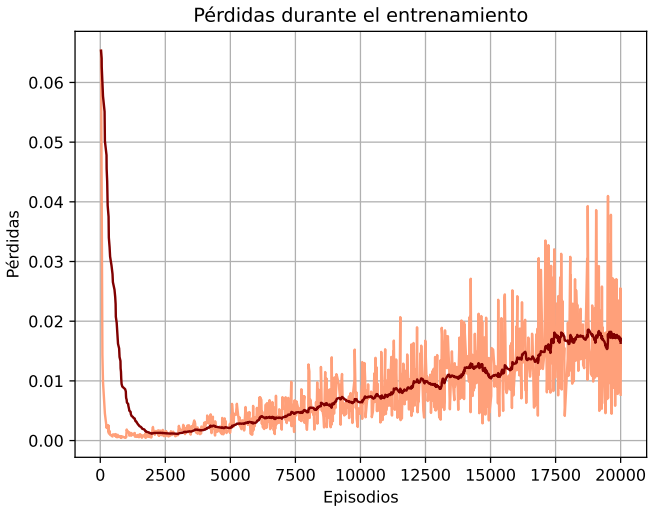


Figura 5.8 Gráfica de la función de pérdida.

En un algoritmo de aprendizaje de una red neuronal, el objetivo de la optimización de esta red es minimizar la función de pérdida. En la Figura 5.8 se nota como al principio se ha ido disminuyendo de forma pronunciada. No obstante, ha comenzado a crecer y cada vez con más alteración, lo que

en un principio parece un indicativo negativo. Pero en nuestro caso, la recompensa acumulada no ha dejado de crecer, luego nuestro algoritmo si está optimizando de manera correcta.

Lo que ocurre es que a medida que el agente aprende a no colisionar y las partidas son más largas, la recompensa futura aumenta. Análogamente, los valores de la función Q, representada por la red neuronal principal, aumentan. Como se adelantó previamente, la función de pérdida en el algoritmo de Double-DQN se obtiene comparando las salidas de la red neuronal principal con la *target-network*. Los parámetros de ésta última red van desactualizados con la principal siempre, excepto en el momento de sustituir sus pesos. Por lo tanto, que la red aprenda demasiado rápido produce un crecimiento de la función de pérdida en nuestro caso.

Porcentaje del escenario visitado

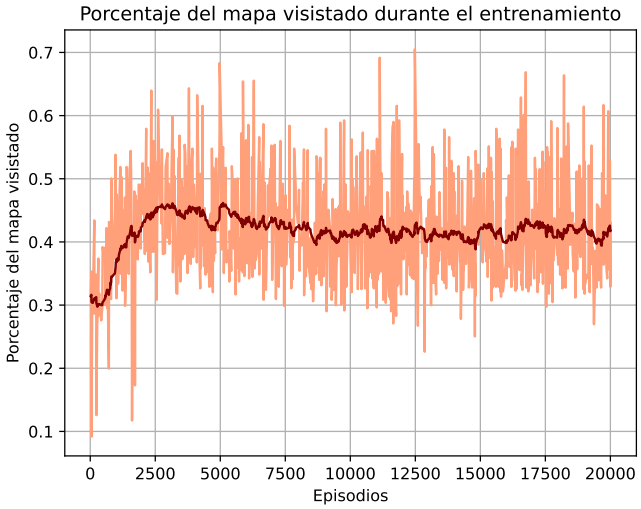


Figura 5.9 Evolución del porcentaje del mapa que se ha visitado.

Medimos el porcentaje de casillas navegables visitadas durante el episodio para conocer como de explorativo es nuestro agente. En la Figura 5.9 se muestra como la performance del algoritmo en este sentido es muy pobre y que hasta el porcentaje va disminuyendo en el tramo de explotación. Es decir, como conocemos el interés de todas las zonas del mapa, el agente se ha interesado en la cobertura de las zonas de alta importancia. Por lo tanto, es una ventaja que la cobertura sea baja, porque eso indica que visitamos sólo los picos.

5.3.2 Evolución de la política

Se ha guardado la política del agente cada 2000 episodios de entrenamiento para estudiar la capacidad de la política, sin ningún tipo de exploración.

Se han evaluado con los mismos 100 episodios a cada política y se ha hecho una media de las métricas que se consideran interesantes. A continuación se mostrarán estos valores en una gráfica, donde el episodio 0 se corresponde con un agente que toma acciones totalmente aleatorias, para modelar la política inicial del agente.

Media de la recompensa acumulada

En la Figura 5.10 se puede contemplar la media de las recompensas y en azul claro la desviación típica, que simboliza el intervalo de confianza. La tendencia de la recompensa es semejante a como se había visto en la sección anterior, pero aquí se ve con más claridad que, a la vez que pasan los episodios, la media de la recompensa disminuye.

Los picos de recompensa se encuentran en el intervalo de [2500,10000] episodios, cuando el agente llega al episodio 2500 ha visto una media de 750.000 experiencias ($2500 * 300$ si suponemos que la media de pasos es 300) y ha entrenado con $\frac{BSxsteps}{steps\_per\_episode} = 1.600.000$. Esto es indicativo de que se ha entrenado muy intensivamente y de que el aprendizaje ha sido rápido para la dificultad que tiene el problema. Pero demasiado entrenamiento ha llegado a degradar la política ligeramente, aunque la diferencia de media en la gráfica sea escaso.

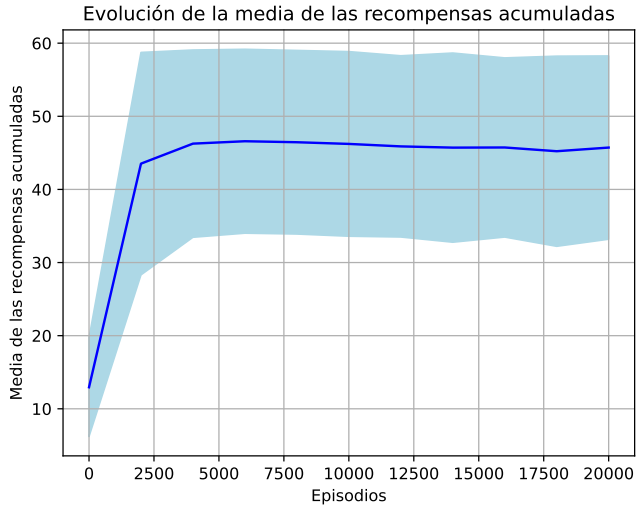


Figura 5.10 Evolucion de la media de la recompensa acumulada.

Media de la suma de interés

El interés recogido durante los episodios es proporcional a la recompensa obtenida, por lo tanto, tienen un comportamiento semejante. Una vez más, se observa en la Figura 5.10 como empeora la política en respecto a este aspecto.

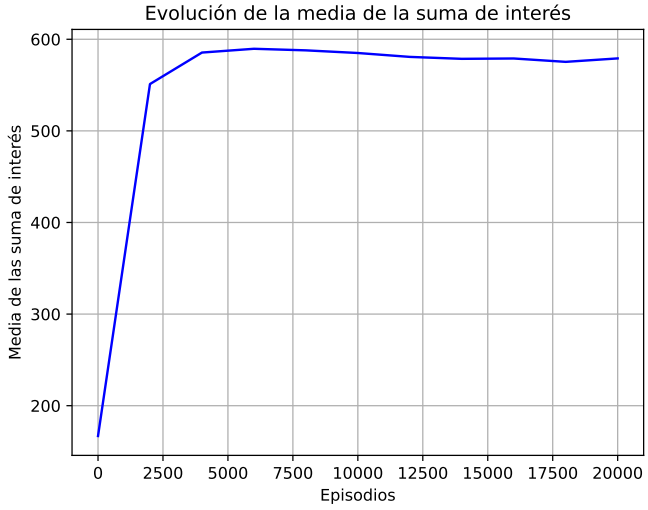


Figura 5.11 Evolucion de la media de la suma de interés.

Media de la importancia relativa

En la Figura 5.12 se muestran, en cada política, la mínima de las medias de la importancia relativa de los diversos mapas de interés. El agente ha conseguido ir disminuyendo este valor, como se desea, pero se observa como empeoran una vez más las políticas a la vez que se entrenan más episodios.

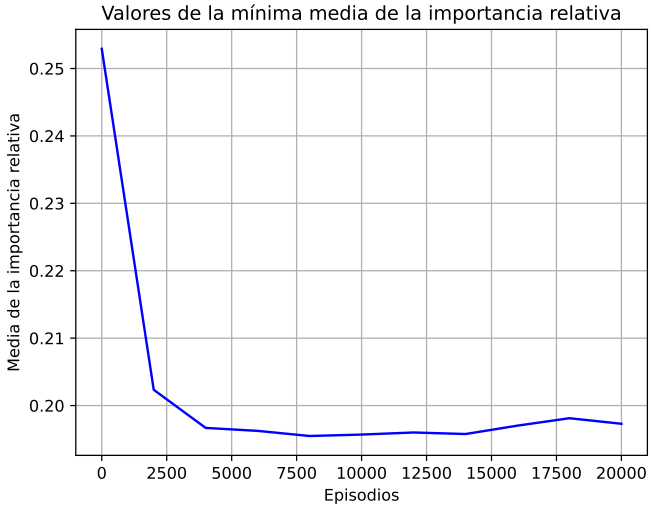


Figura 5.12 Valores mínimos de la media de la importancia relativa.

Media del porcentaje del mapa visitado

En la Figura 5.13 se observa como esta medida va disminuyendo mientras la red va aprendiendo. Hay que tener en cuenta que al principio las políticas $\varepsilon - greedy$ obligan al agente a tomar acciones aleatorias, de modo que es probable que se descubran forzosamente nuevas casillas. El hecho de que el porcentaje visitado disminuya a lo largo del aprendizaje significa que el agente se enfoca cada vez más en una zona. Esto por supuesto es una señal de que la política tiende a ser poco explorativa y conformista con el interés inmediato.

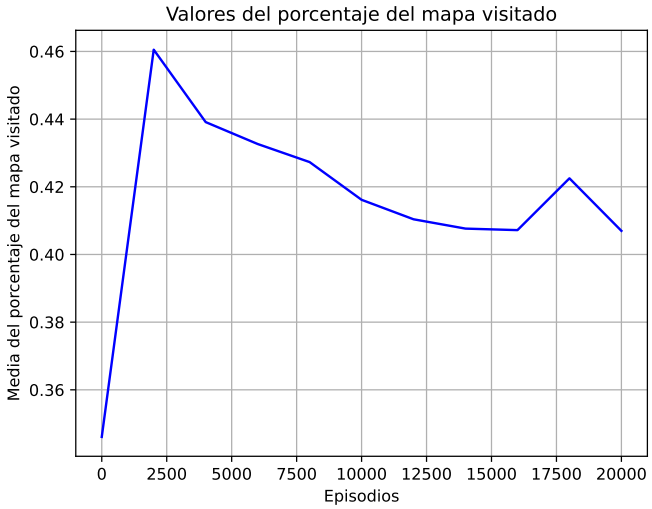


Figura 5.13 Valores del porcentaje del mapa visitado.

5.3.3 Conclusiones

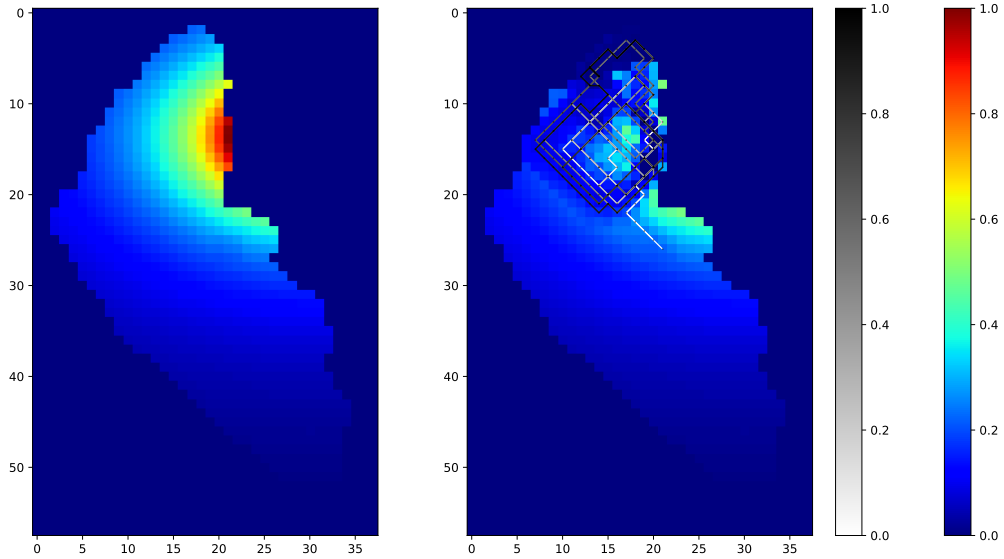


Figura 5.14 Trayectoria del vehículo en el entorno Completamente Observable..

El resultado de las métricas de este entrenamiento muestra como la política óptima para el agente ha sido enfocarse en los picos de interés y esperar a que se regeneren para obtener la mayor recompensa. Este comportamiento se puede apreciar visualmente en la Figura 5.14, en esta imagen se muestra la trayectoria del vehículo en la política que ha conseguido mayor recompensa en el entrenamiento. Los colores más cálidos de la línea que representa la trayectoria son los últimos tramos.

También se ve claramente como el agente prácticamente sólo se mueve en diagonal para poder obtener más información por paso. En consecuencia, los episodios duran menos *time steps* debido a que se gasta más rápido la batería del vehículo. Cabe destacar que los episodios duran lo mismo exactamente en tiempo y distancia ya que el presupuesto de la batería es fijo.

La firmeza del agente de focalizarse en el pico de interés hace que disminuye cada vez la suma de interés porque cada vez que visita una casilla su interés absoluto disminuye según el parámetro *atrición*, por ello, cada vez hay menos interés en esas casillas.

Aún así, parece que cada vez reitera más este comportamiento y deja de visitar las demás partes del lago. Es por ello que disminuye el porcentaje de cobertura aunque solo se disminuye de un 46 % a un 41 % al final.

Por último, el comportamiento de la política, aún cuando maximiza su recompensa, no es óptimo desde la perspectiva del diseño. Por tanto, la estrategia del agente no cumple los objetivos principales del problema del patrullaje y parece que debe ser más premiado a explorar por lo que se necesita rediseñar la propia ley de recompensa.

5.4 Escenario Parcialmente Observable

Para este entrenamiento del agente, llevado a cabo en el Escenario Parcialmente Observable. Se han realizado 20000 episodios con los mismos parámetros listados en la Tabla 5.1. Recordemos que el

agente no tiene ninguna información previa de la distribución del interés en el lago, por consiguiente debe, al menos al principio, explorar el mapa para encontrar las zonas de mayor interés.

También se ha configurado el escenario igual que en el Escenario Completamente Observable(Tabla 5.2). A continuación se estudian las mismas métricas del problema, asimismo, en las figuras de estas métricas se muestran superpuestas: los valores reales de las métricas y un filtrado exponencial de media móvil(en inglés EMA) con $\alpha = 0.05$ para poder ver con claridad la tendencia de los valores.

5.4.1 Evolución de las métricas

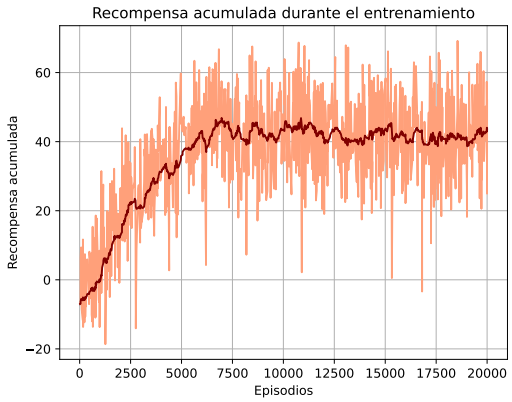


Figura 5.15 Valores de la recompensa acumulada durante el entrenamiento.

Recompensa acumulada

Se muestra en la Figura 5.15 la evolución de la recompensa, la cuál es equivalente a la obtenida en el escenario anterior. Igualmente, tanto los valores medios como máximos y mínimos son semejantes. Entonces se puede llegar a la conclusión de que el agente consigue encontrar los picos de interés.

Longitud de los episodios

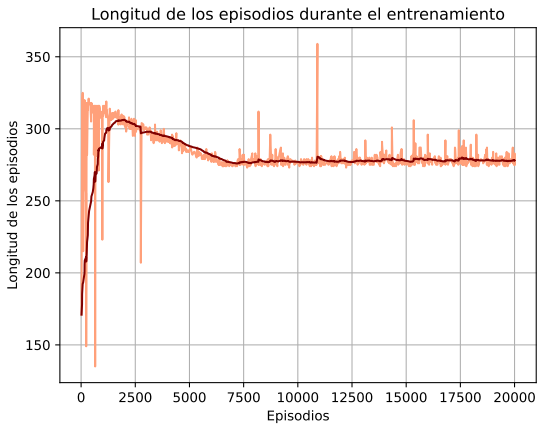


Figura 5.16 Longitud de los episodios durante el entrenamiento.

Análogamente al primer entrenamiento, el agente consigue darse cuenta de que se recolecta más información desplazándose en diagonal y por tanto gastando más batería.

Media de interés

En este escenario en vez de estudiar cuál la media del interés relativo, el cuál ya no es un estado como en el caso anterior, se estudia la media del interés estático descubierto. En la Figura 5.7 se observa que al principio se va descubriendo más interés pero a medida que se cruza el 33 % de episodios se va disminuyendo este valor. Como el agente no conoce toda la información, puede no descubrir toda las zonas de alto interés. Dicho de otro modo, puede que haya más de un pico de interés, pero el agente deja de buscar cuando encuentra uno. Por otro lado, el interés estático se va disminuyendo según el parámetro *atrición*, luego cuanto más tiempo se quede el agente repasando una zona, dejará cada vez menos interés .

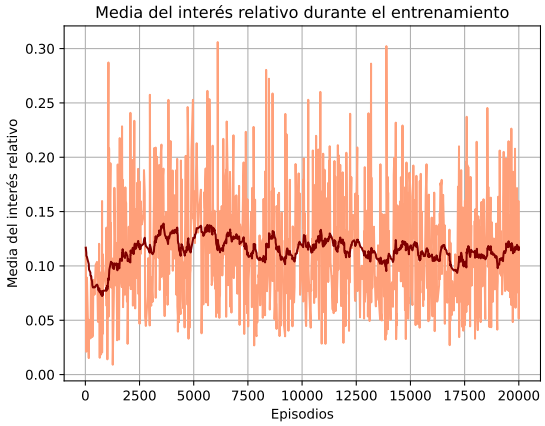


Figura 5.17 Media de interés.

Función de pérdida

La función de pérdida evoluciona de la misma manera que en el escenario previo, esto tiene sentido teniendo en cuenta que, con los datos obtenidos de las anteriores figuras, al final el agente sigue la misma estrategia de quedarse repasando una zona de alto interés.

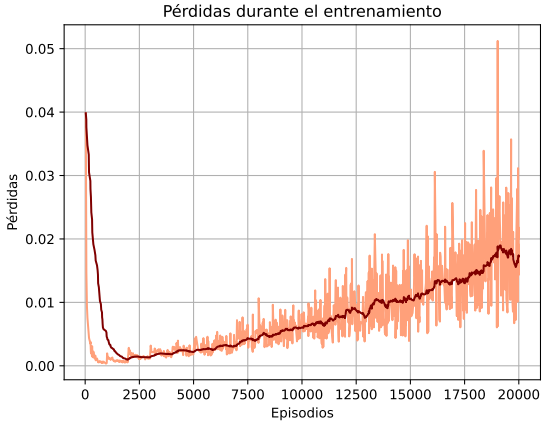


Figura 5.18 Gráfica de la función de pérdida.

Porcentaje del escenario visitado

En la Figura 5.19 se muestra como, una vez más, cada vez se exploran menos zonas.

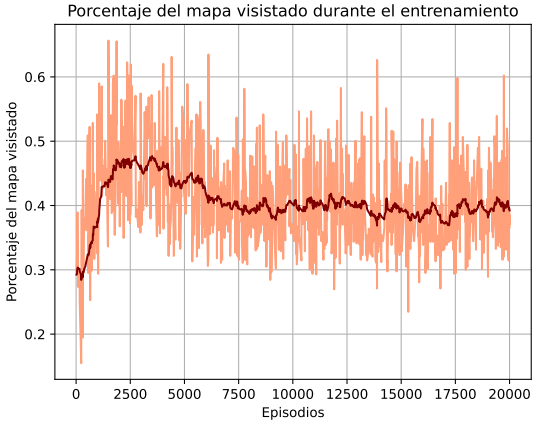
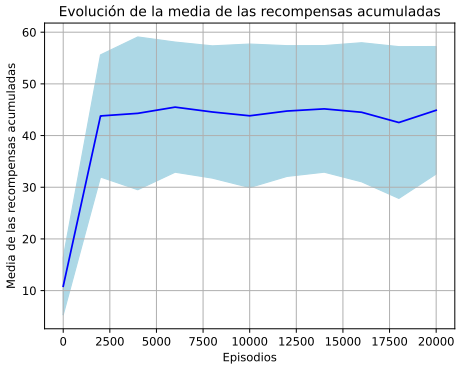
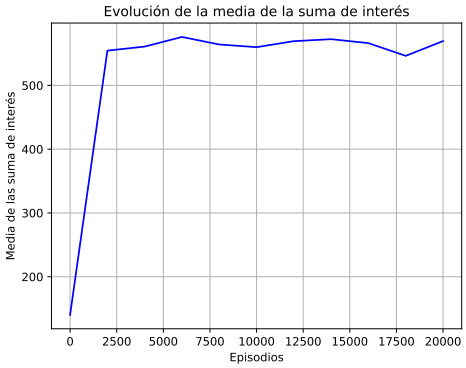


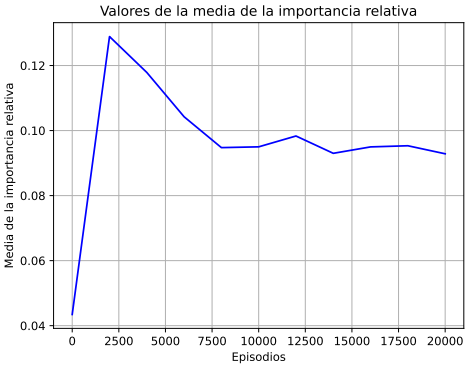
Figura 5.19 Evolución del porcentaje del mapa que se ha visitado.



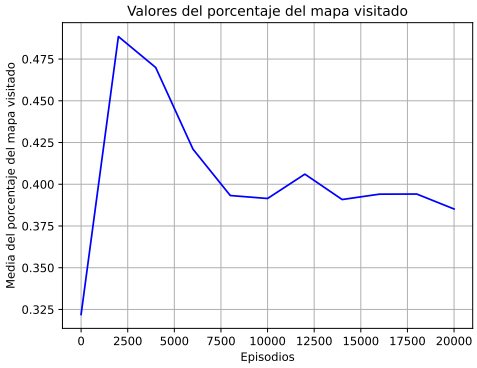
(a) Media de las recompensas acumuladas



(b) Media de las sumas de interés



(c) Importancia relativa



(d) Media de los porcentajes del mapa visitado

Figura 5.20 Evolución de la política durante el aprendizaje. Se ha guardado la política del agente cada 2000 episodios de entrenamiento y se han evaluado con los mismos 100 mapas de interés a cada política y se ha hecho una media de las métricas que se consideran interesantes..

5.4.2 Evolución de la política

La política en este segundo entrenamiento progresa de la misma manera que en el primero aunque varía ligeramente debido a la estocasticidad del entrenamiento. No obstante, el aprendizaje es estable y no se da ningún fenómeno de *catastrophic forgetting* (degradación severa del performance). La obtención de la recompensa se mantiene casi constante con los episodios en el periodo de explotación, sólo se reduce ligeramente y no cambia la media drásticamente. El valor mínimo de la media de recompensa acumulada del episodio 5000 al 20000 sólo es 6.5 % menos que el máximo. La suma de interés es proporcional a la recompensa y por tanto evolucionan de manera similar.

La media de la importancia relativa se con el mapa visitado, se muestra en la subfigura (c) de la Figura 5.20 y parece que es la única métrica que mejora. Esta vez no cogemos el mínimo siempre es el primer *time step* siempre ya que no conocemos ninguna información todavía. La tendencia decreciente de este valor es signo de que cada vez el agente repasa mejor el interés que encuentra. Sin embargo, el porcentaje del mapa visitado disminuye, por lo que se deduce que cada vez se mantiene más concentrado en los picos de interés.

5.4.3 Exploración del agente

Cómo no se tiene ningún conocimiento del entorno al principio del episodio, es muy importante que el agente encuentre una estrategia de exploración eficiente.

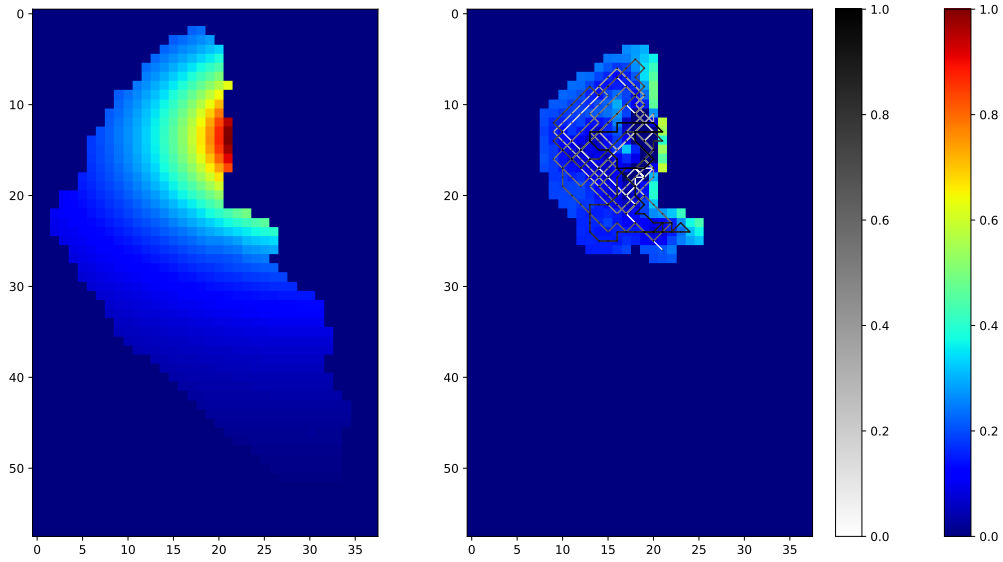


Figura 5.21 Trayectoria del vehículo en el entorno Parcialmente Observable.

En la Figura 5.21 se muestra la trayectoria del vehículo en un mapa del lago y se observa como una vez que encuentra una concentración de interés deja de explorar y no descubre más zonas. Las casillas no visitadas son las que están en azul oscuro.

Sin embargo, en este mapa sólo existía un pico, en la Figura 5.22 vemos como consigue encontrar el otro pico cuando el primero pierde por completo su interés temporal. Esto es porque la estrategia de este agente es seguir la dirección en la que aumenta el interés de las casillas. Este es un buen método a seguir especialmente si se está explorando una zona completamente desconocida. Además, es un comportamiento que, en principio, asegura una recompensa futura mayor.

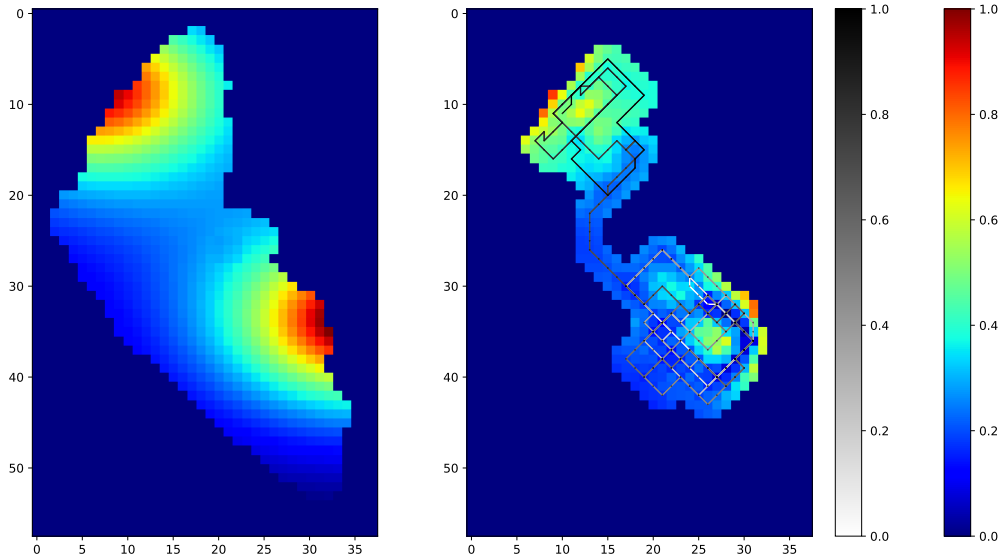


Figura 5.22 Trayectoria del vehículo en el entorno Parcialmente Observable con dos picos de interés.

Para comprobar la robustez del agente frente encontrar picos muy alejados, se ha creado un pico artificial con distribución Gaussiana(Figura 5.23) en la otra parte del lago. Efectivamente, el agente no ha sido capaz de descubrirlo porque no está lo suficientemente extendido como para que el agente note un crecimiento de interés.

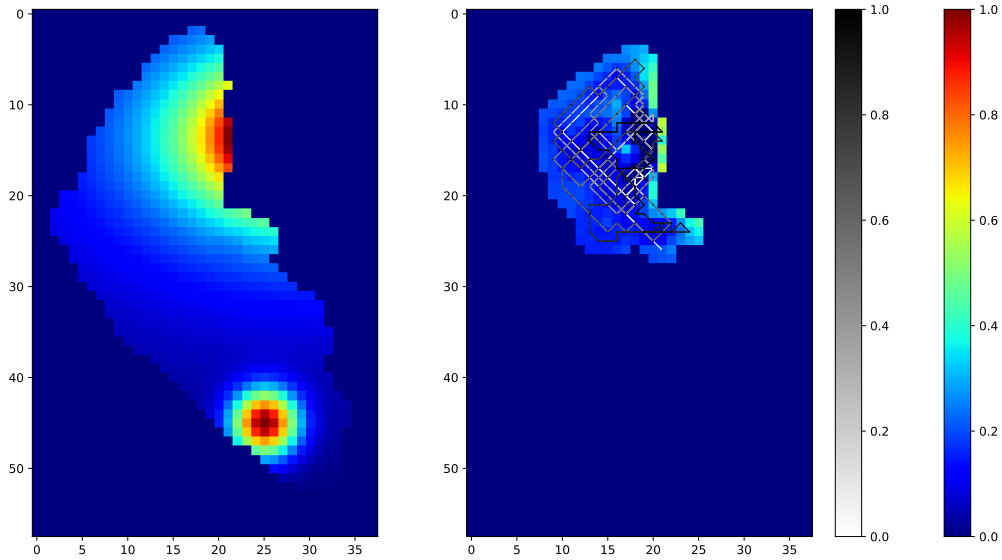


Figura 5.23 Trayectoria del vehículo en el entorno Parcialmente Observable con un pico de interés de distribución Gaussiana creado con una máscara de filtro gaussiano de tamaño $23x23$, $\mu = 0$ y $\sigma = 2.5$.

Por último, para tomar como referencia cómo se comportaría el agente entrenado en el Entorno

Completamente Observable en el mapa de la Figura 5.23, se ha evaluado la última política. Como se puede observar en la Figura 5.24, el agente no es capaz de recolectar el interés del pico de tipo gaussiano aunque sepa claramente que es una zona de alto interés y se dirige hacia ella. Como se le ha proporcionado la distribución de la información al agente, este ha aprendido a cubrir los picos de interés ajustándose al modelo de estos y es incapaz de cubrir otros tipos. Por tanto, se ha producido *overfitting* y el agente ha perdido la capacidad de generalización. Cabe destacar que la longitud del episodio ha sido de 273 pasos aunque el recorrido haya sido corto, esto es porque al final el vehículo se ha quedado repitiendo los mismos dos pasos hasta agotar la batería.

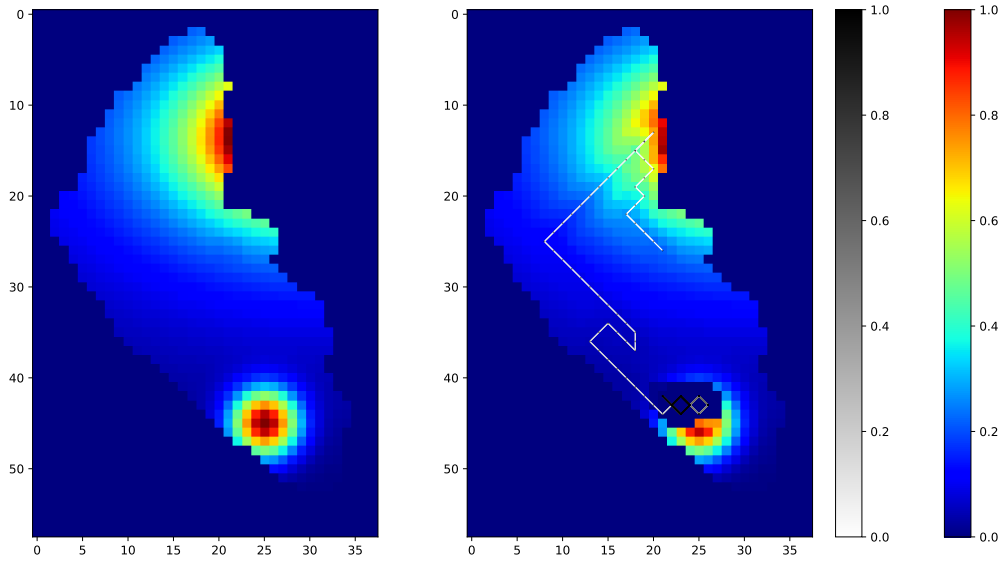


Figura 5.24 Trayectoria del vehículo en el entorno Completamente Observable con un pico de interés de distribución Gaussiana creado con una máscara de filtro gaussiano de tamaño $23x23$, $\mu = 0$ y $\sigma = 2.5$.

5.4.4 Conclusiones

Quedarse en los picos de interés asegura recompensa futura y la estrategia del agente es, primero encontrarlo siguiendo el camino por el que aumenta el interés y luego repasar esta zona. Esta estrategia ha sido muy efectiva ya que, como se observa en las figuras, se encuentran casi las mismas sumas de interés que en el anterior entrenamiento. Sin embargo, debido a la poca exploración del agente, una concentración de interés lejana al que el agente ha detectado primero, puede no ser nunca descubierta. El objetivo de nuestro estudio es minimizar el *idleness* de cada casilla y por tanto se debe incentivar al agente a descubrir todas las zonas para descubrir otras posibles concentraciones de interés. Es por ello que en el siguiente experimento se utiliza la función de recompensa que añade un término de exploración ponderado por el parámetro *discovery reward*.

5.5 Escenario parcialmente observable con recompensa de cobertura

De los anteriores experimentos se puede sacar conclusión de que necesitamos que el agente sea empujado a explorar el mapa. Con la función de recompensa, Ecuación 4.40, podemos controlar el peso de la recompensa a la exploración con el parámetro *discovery reward*, denotado como λ en la Tabla 4.1.

5.5.1 Experimentos

Se han simulado 20.000 episodios en el Escenario Parcialmente Observable para distintos valores de *discovery reward* y se han guardado la últimas y las mejores políticas. Se han usado los hiperparámetros listados en la Tabla 5.1. Recordemos que el agente no tiene ninguna información previa de la distribución del interés en el lago y estamos intentando hacer que explore. También se ha configurado el escenario como en la Tabla 5.2).

Para evaluar los entrenamientos se han simulado 100 episodios en cada caso y se ha hecho una media de las métricas. En las gráficas que se mostrarán a continuación se trazan estas medias para distintos valores de *discovery reward*: 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1. Se desea conocer el valor de λ que balancee perfectamente la exploración con la suma de interés recogido.

Media de las recompensas

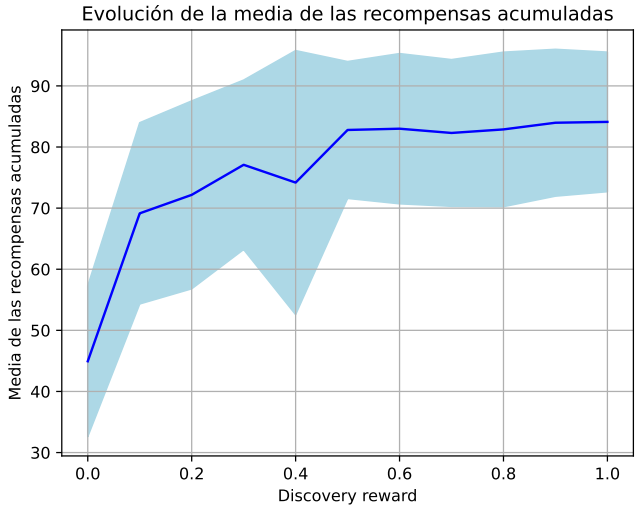


Figura 5.25 Media de las recompensas.

Como era de esperar, a medida que aumenta discovery reward, aumenta la recompensa debido a que la función de recompensa devuelve valores mayores. Sin embargo, en *discovery reward = 0.4* se aprecia como aumenta considerablemente el intervalo de confianza y que la recompensa disminuye respecto al valor anterior. Pero para entender mejor como se está comportando el agente hay que estudiar las demás métricas.

Media del porcentaje del mapa visitado

Tal y como se pretendía, el porcentaje del mapa visitado aumenta porque la exploración es cada vez más premiada. En *discovery reward = 1* casi se alcanza el 100 % de cobertura y en *discovery reward = 0.8* aproximadamente el 80 %, pero en la Figura 5.25 se ve que las recompensas obtenidas son similares en ese intervalo. El hecho de que ahora obtenga valores mayores de recompensa al visitar nuevas celdas hace que vea más importante explorar que visitar casillas de interés alto.

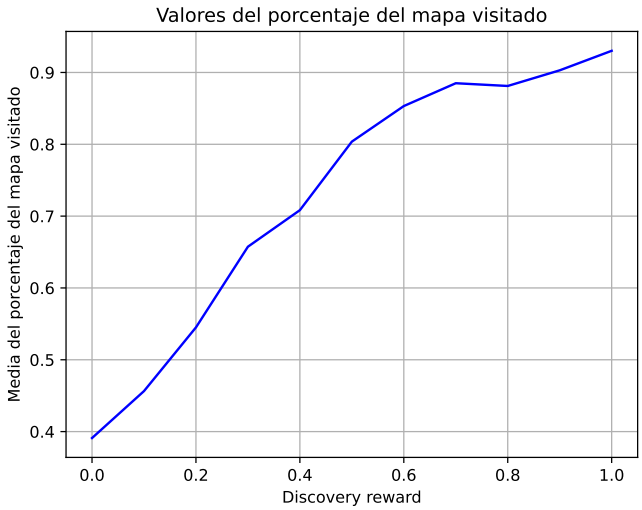


Figura 5.26 Media del porcentaje del mapa visitado.

Conseguimos una cobertura total pero ahora no es imparcial con respecto a los picos de interés, por tanto este comportamiento es indeseado.

Media del porcentaje de la recompensa a la exploración

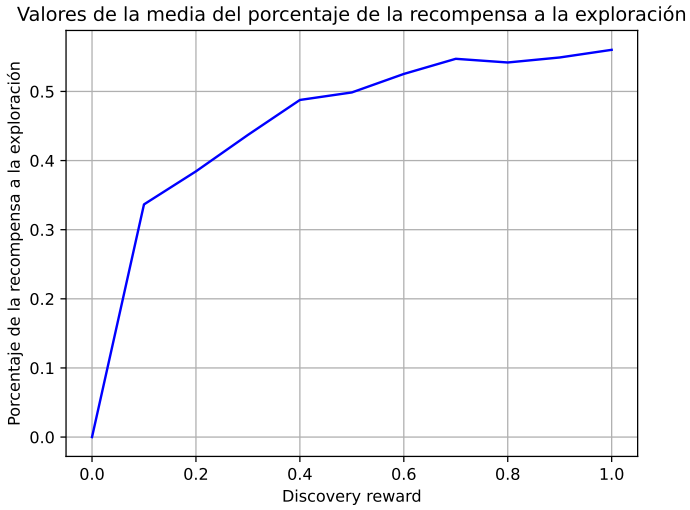


Figura 5.27 Media del porcentaje de la recompensa a la exploración.

Para estudiar el peso que tiene la exploración en la función de recompensa hacemos media del porcentaje del término de la exploración en la suma de recompensas en cada *discovery reward*. Vemos en la Figura 5.27 que el peso es cada vez mayor.

Media de la importancia relativa

La Figura 5.28 nos sirve para reafirmar que cada vez el agente se preocupa menos en minimizar el idleness, sino en visitar más celdas.

Recompensa a lo largo de un episodio

En un episodio el agente no deja de recibir recompensas aunque se marche a zonas de menor interés, lo que incentiva a la exploración. Por eso se ve en la Figura 5.29 que a medida que aumenta

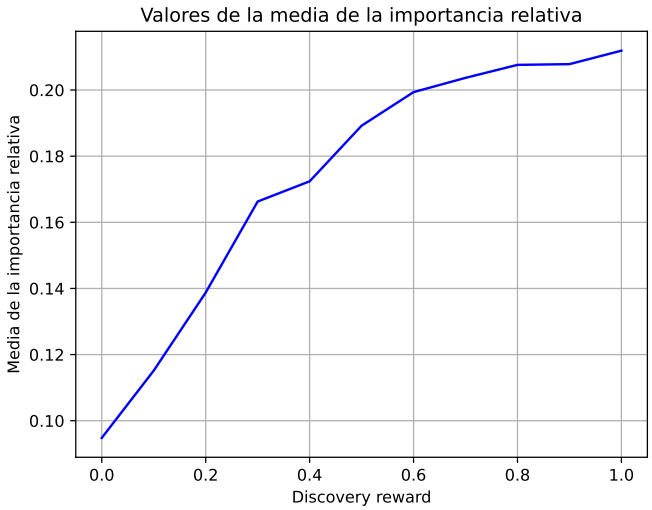
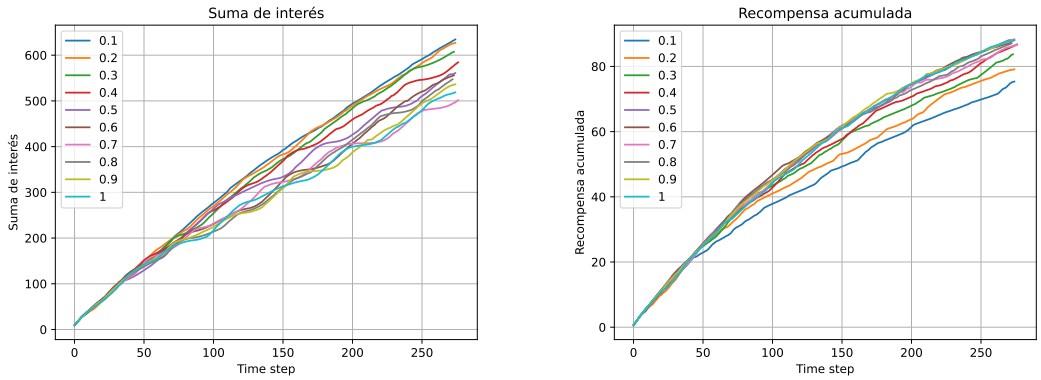
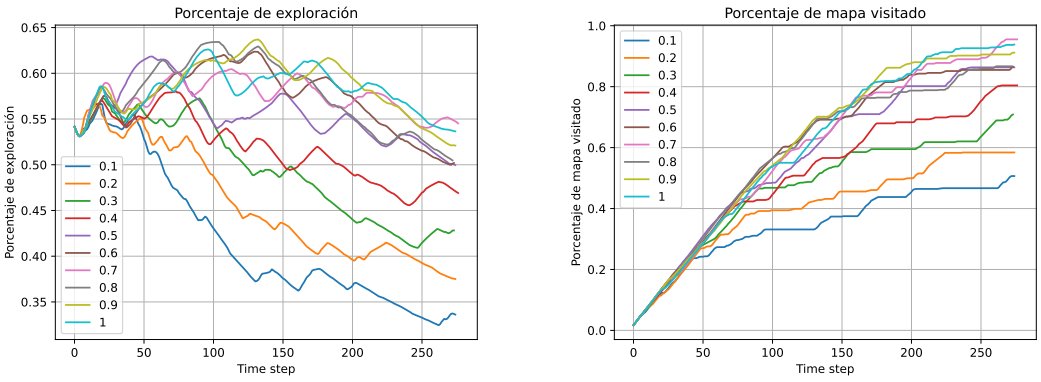


Figura 5.28 Media de la importancia relativa.



(a) Suma del interés recolectado

(b) Suma de la recompensa obtenida



(c) Porcentaje del término de exploración en la recompensa total

(d) Porcentaje del mapa visitado

Figura 5.29 Evolución de las métricas con el término *discovery reward* a lo largo de un episodio para cada política. En la leyenda se muestra cada valor de *discovery reward* con el color de su curva correspondiente.

discovery reward las curvas son más suaves. En los primeros valores donde no pesa demasiado descubrir nuevas zonas, el comportamiento seguido sigue siendo permanecer en la concentración de casillas de alto interés. Por consiguiente, hay tramos de poca obtención de recompensa mientras se regenera el interés, como se puede observar en la curva de *discovery reward = 0.1*.

Suma de interés obtenido a lo largo de un episodio

Cuanto más alto es el valor de *discovery reward*, hay menos suma de interés porque al explorar más el escenario no se puede focalizar continuamente en las zonas de mayor interés. Esto es una característica típica de un problema multiobjetivo: no se puede explorar e intensificar a la vez perfectamente. También hay más tramos planos en las curvas de la Subfigura (a) de la Figura 5.29. Estos tramos representan los instantes en dónde el agente explora las zonas de menor interés mientras las zonas más importantes recuperan su interés temporal. Es por ello que estas curvas son más rectas y llegan a valores más altos cuanto más bajo es el valor de *discovery reward* en el entrenamiento. Cabe destacar que aunque la curva azul claro (*discovery reward = 1*) de la Subfigura (a) consigue alcanzar los valores más bajos en cuanto a suma de interés, es el que más recompensa total ha obtenido (Subfigura (b)). Eso es signo de que aunque la función de recompensa devuelva valores más altos cuando mayor es *discovery reward*, el agente no tiene porqué estar comportándose mejor ni cumpliendo los objetivos de diseño.

Porcentaje de la recompensa a exploración a lo largo de un episodio

Esta medida muestra el peso de la recompensa a exploración en la recompensa total y como evoluciona esta a lo largo de un episodio(Subfigura(c) de la Figura 5.29). Cuanto mayor es *discovery reward*, el porcentaje es cada vez mayor. La tendencia de las curvas es decreciente desde el principio para valores de *discovery reward < 0.5* debido a que prefieren ser menos explorativos. Para los demás valores, observamos como sus curvas tienden a crecer inicialmente, aunque luego decrecen porque cada vez hay menos casillas que descubrir. Los tramos de aumento, los cuáles son más pronunciados cuanto mayor es *discovery reward*, representan los instantes en dónde el agente explora nuevas zonas.

Porcentaje del mapa visitado a lo largo de un episodio

Como era de esperar, cuanto mayor es *discovery reward*, más se enfoca el agente en aumentar las zonas visitadas.

5.5.2 Conclusiones

El valor de *discovery reward* puede provocar que buscar nuevas casillas sea más rentable que buscar casillas de mayor interés. En consecuencia, el agente será más explorativo si *discovery reward* es muy grande, y si es al contrario, será más avaricioso en términos del interés.

De las gráficas anteriores se puede deducir que el valor de *discovery reward* que más balancea la exploración con la suma de interés recogido debe estar cerca del 0.5. Puesto que el porcentaje del mapa visitado es 80 % y además el valor de la media de la importancia relativa es de aproximadamente 0.19. Este último es similar al valor que obtenían las mejores políticas del primer entrenamiento. Para un mejor análisis, se muestra en la Figura 5.31 la trayectoria del vehículo en el mismo mapa de interés para cada valor de *discovery reward*.

Se puede observar que cada vez se aleja más de las casillas de alto interés y realiza más visitas a otras áreas. A partir de $\lambda = 0.5$ la cobertura es casi total y al parecer, cada vez que un pico de interés pierde importancia temporal, el vehículo se decanta a explorar otros lugares y después volver. Este comportamiento no aparecía en los entrenamientos anteriores, donde el vehículo nunca salía de las zonas de alto interés.

A medida que aumenta *discovery reward* disminuye la suma de interés, como se muestra en la Figura 4.18 y en la Subfigura (a) de la Figura 5.29. Pero el aprendizaje con $\lambda = 0.5$ tiene de media

aproximadamente 50 unidades de interés más que $\lambda = 0.7$, siendo este último sólo un 7 % más explorativo. Además de que es un máximo local porque recoge más interés que sus valores vecinos.

Por otro lado, en la Figura 5.27 se aprecia que cuando $\lambda = 0.5$ el 50 % de la recompensa obtenida se ha conseguido visitando nuevas casillas, En conclusión, los valores del término referente a la exploración tienen que ir del orden del valor de la recompensa que da visitar una zona de interés alto.

Por otra parte, en la Subsección 5.4.3 se mostró como, al añadir un pico gaussiano, ninguno de los dos agentes entrenados con la función de recompensa sin término de exploración fueron capaz de cubrir el mapa eficientemente. Recordemos que el agente del primer entrenamiento (Escenario Completamente Observable (Sección 5.3)) había sufrido de sobreajuste(*overfitting*) y no era capaz de cubrir el pico artificial. Mientras que el agente del segundo entrenamiento (Escenario Parcialmente Observable (Sección 5.4)) no había sido capaz de encontrar el pico artificial.

No obstante, en la Figura 5.30 se observa como nuestro agente entrenado con *discovery reward=0.5* ha sido capaz de encontrar el pico artificial y además ha sido capaz de cubrirlo.

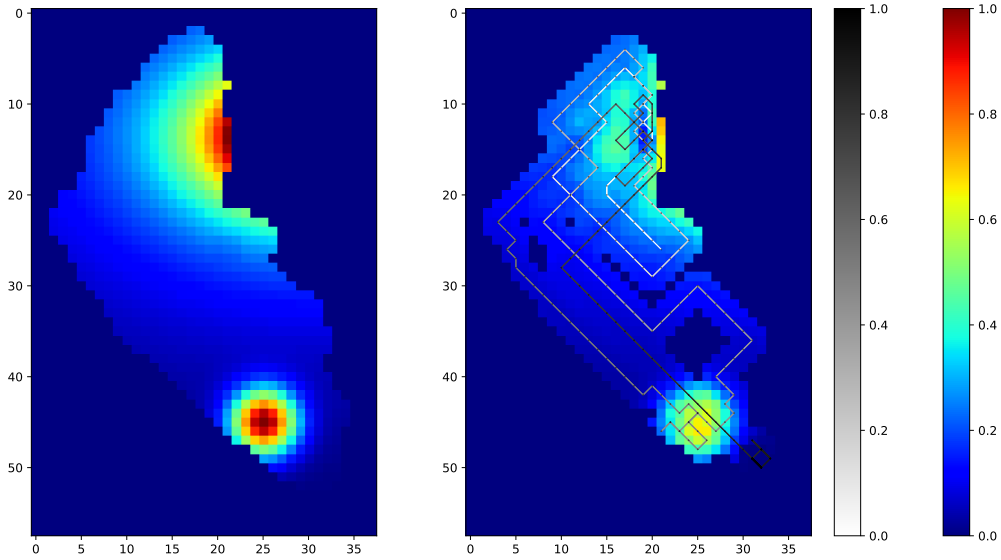
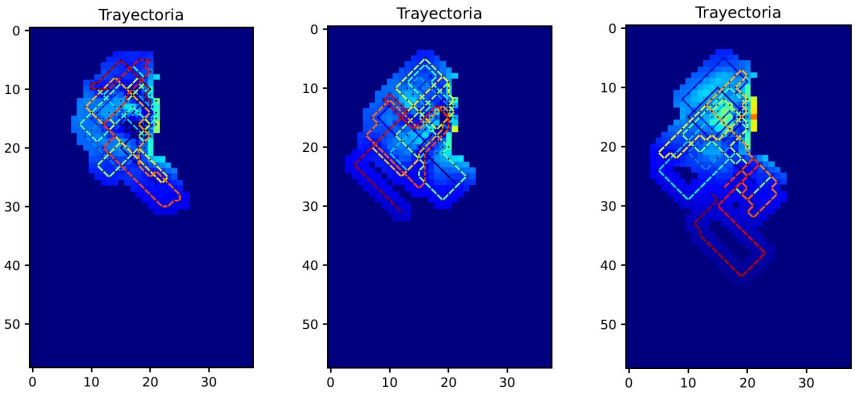


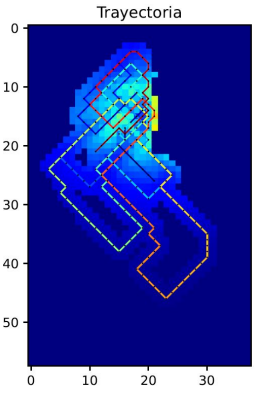
Figura 5.30 Trayectoria del vehículo en el entorno Parcialmente Observable con un pico de interés de distribución Gaussiana creado con una máscara de filtro gaussiano de tamaño 23*x*23, $\mu = 0$ y $\sigma = 2.5$.



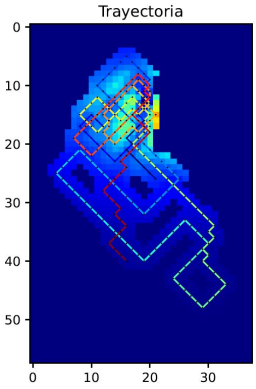
(a) $\lambda = 0.1$

(b) $\lambda = 0.2$

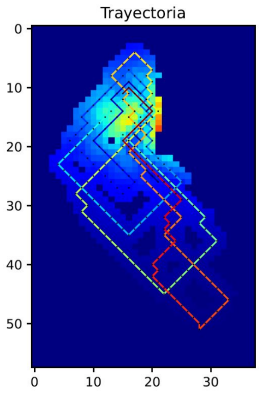
(c) $\lambda = 0.3$



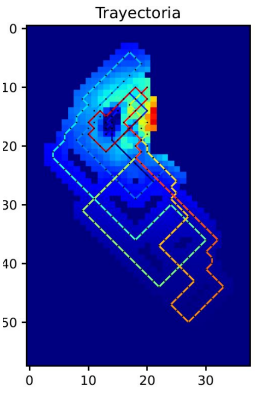
(d) $\lambda = 0.4$



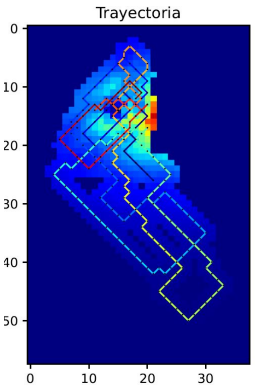
(e) $\lambda = 0.5$



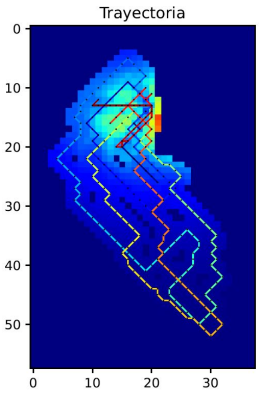
(f) $\lambda = 0.6$



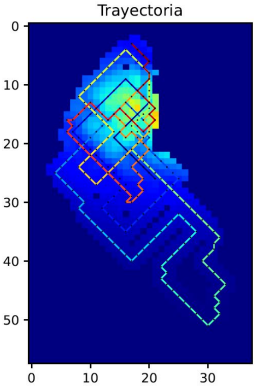
(g) $\lambda = 0.7$



(h) $\lambda = 0.8$

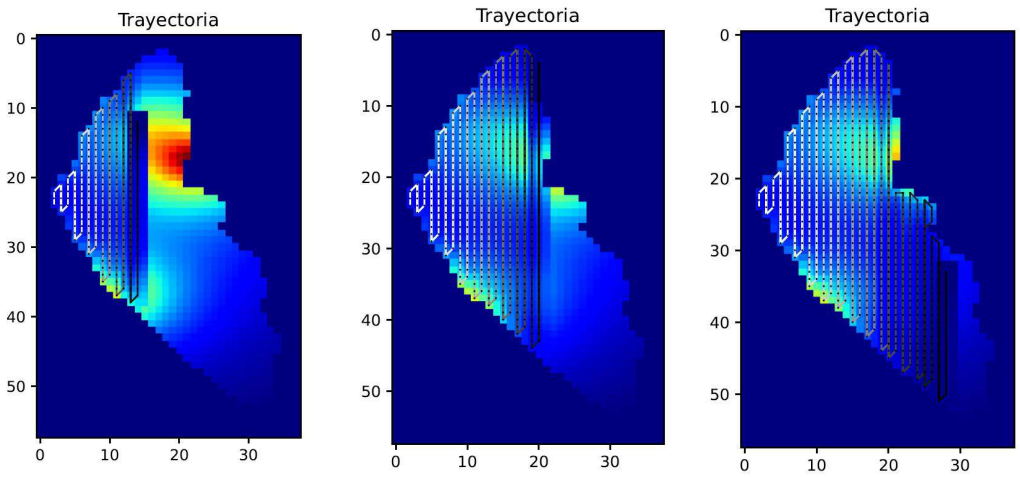


(i) $\lambda = 0.9$



(j) $\lambda = 1$

Figura 5.31 Evolución de la política durante el aprendizaje.



(a) (b) (c)

Figura 5.32 Visualización de la trayectoria de un algoritmo cortacésped, las partes más oscuras de la trayectoria son las más recientes.

5.6 Comparación con otros algoritmos basados en heurísticas

5.6.1 Algoritmo cortacésped

El algoritmo cortacésped es un método típico de pathplanning de tipo barrido (lawn mower - cortacésped). Consiste en visitar una vez todas las casillas del mapa, ya sea en trayectorias totalmente verticales o totalmente horizontales. El algoritmo de tipo barrido garantiza tener una cobertura total del mapa, pero no tiene en cuenta la importancia de las zonas ni la redundancia útil.

5.6.2 Algoritmo de dirección aleatoria

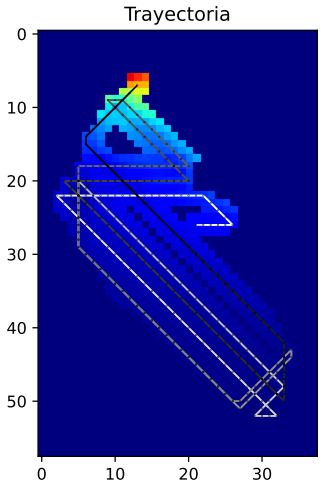


Figura 5.33 Visualización de la trayectoria de un algoritmo de dirección aleatoria, las partes más oscuras de la trayectoria son las más reciente.

En este algoritmo, el agente escoge una acción aleatoria y la mantiene hasta que llega a una orilla. Entonces escoge otra acción aleatoria que no sea la que vuelva sobre sus pasos u otra que colisione. Visualmente, el vehículo va rebotando en los bordes como una bola en una mesa de billar.

5.6.3 Algoritmo de movimientos aleatorios

En este algoritmo las acciones del agente son totalmente aleatorias, excepto cuando tome una acción que le lleve a colisionarse, entonces escoge otra acción aleatoria que tampoco le lleve a estrellarse.

5.6.4 Experimentos

Se han hecho media de las métricas en 50 episodios para comparar nuestro agente con otros métodos de *path-planning*. Se ha escogido la política entrenada con *discovery reward* $= 0.5$

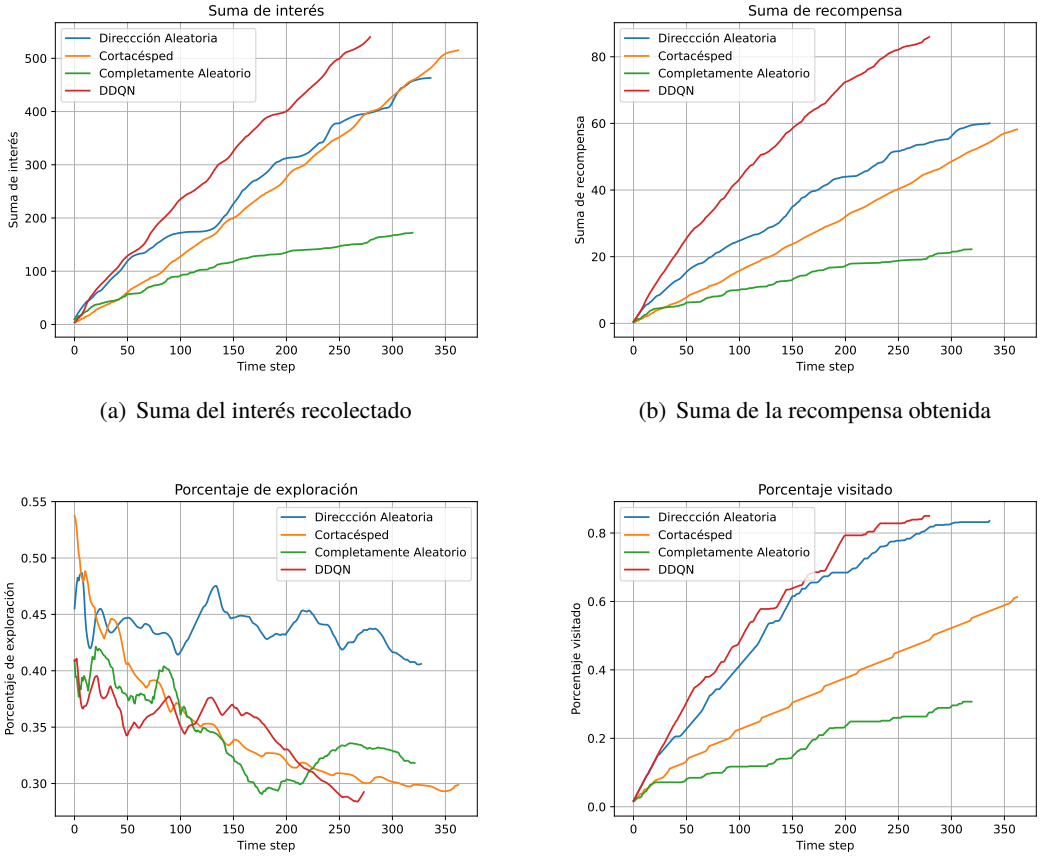
Tabla 5.3 Comparación de métricas de interés.

	Cortacésped	**Direcciones aleatorias**	**Aleatorio**	**Double-DQN**
Recompensa	39.25	47.8	21.48	63.8
Desviación típica	7.74	8.4	6.6	10.5
Suma de interés	276	293	153	484
Min. importancia relativa	0.20	0.21	0.23	0.18
Porcentaje visitado	61 %	85 %	34 %	82 %
Duración del episodio	379	326	320	276

Como tenemos en cuenta la duración de la batería, la cobertura del algoritmo cortacésped es siempre 61 % en 379 pasos. En cambio, nuestro agente ha conseguido en 276 pasos de media, 103 menos, 20 % más de cobertura. Esto es debido al desplazamiento diagonal del agente frente al vertical del algoritmo cortacésped. La suma de interés y por tanto la recompensa es también mayor debido a la pasividad de este último algoritmo de visitar con más asiduidad a las zonas de mayor interés. Esto último también es el motivo por el que nuestro agente consigue mejor minimizar la importancia relativa, consiguiendo un valor de 0.18 frente a los 0.20 del algoritmo cortacésped.

En cuanto a los algoritmos aleatorios, estos realizan una performance peor que nuestro algoritmo. El método de direcciones aleatorias visita de media el 86 % del mapa, pero suma casi la mitad de interés y tiene recompensas menores, además realiza más pasos. Recordemos que la red neuronal había aprendido a recoger más interés por desplazamiento moviéndose en diagonal, con el inconveniente de que la batería se agotaba más rápido. El algoritmo completamente aleatorio es el que peores resultados ha obtenido en todas las métricas. En general, nuestro agente ha conseguido una performance tres veces mejor que este último con 44 pasos menos. En resumen, nuestro algoritmo es capaz de cubrir más eficientemente el lago y obtener mayores recompensas y es capaz de discriminar las zonas con mayor interés concentrado.

En cuánto a la evolución de las métricas a lo largo de un episodio(Figura 5.34), vemos como el algoritmo Double-DQN es claramente superior en suma de interés obtenido (Subfigura(a)) y porcentaje de la recompensa obtenida (Subfigura(b)). También se observa que aunque el algoritmo cortacésped alcanza a recolectar más interés, obtiene menos recompensa que el algoritmo de direcciones aleatorias, esto es debido a que este último ha cubierto más zonas del mapa, como se puede notar en la Subfigura(d). En cuanto al porcentaje del mapa visitado, nuestro agente ha conseguido el mejor resultado junto con el algoritmo de direcciones aleatorias. Sin embargo, el algoritmo Double-DQN ha obtenido 10 % menos de porcentaje de la recompensa a exploración (Subfigura(c)), lo cuál era de esperar ya que ha recolectado más interés. Por lo tanto, en este episodio, nuestro agente ha realizado una cobertura y recompensa mayor que los demás algoritmos.



(a) Suma del interés recolectado

(b) Suma de la recompensa obtenida

(c) Porcentaje del término de exploración en la recompensa total

(d) Porcentaje del mapa visitado

Figura 5.34 Evolución de las métricas con el término *discovery reward* a lo largo de un episodio para cada política. En la leyenda se muestra cada valor de *discovery reward* con el color de su curva correspondiente.

6 Discusión y conclusiones

El uso de Vehículos Autónomos de Superficie (ASV)s para la monitorización de entornos contaminados ha sufrido un auge en estas últimas décadas debido a que disminuye la necesidad de arriesgar la salud de las personas en estos ambientes. Para ello el vehículo debe ser capaz de generar y seguir waypoints que determinan la trayectoria ideal a seguir para cumplir su labor. El DRL ha sido recientemente uno de los enfoques más comunes para la planificación reactiva de trayectorias y la evitación de colisiones. Esto es debido, en parte, a la capacidad de las redes neuronales a ser aproximadores paramétricos no lineales, lo que ha producido que DRL se haya convertido en la forma más común para tratar la aproximación de funciones complejas. De esta forma, los robots son capaces de saber responder con eficacia a la lectura de los sensores, entendiendo, predeciendo y actuando en entornos dinámicos.

Existen tareas de soluciones complejas con espacio de acciones grandes y estados prácticamente infinitos, pero en DRL la red conserva los pesos para observar determinados patrones en los estados para todas las acciones. Por consiguiente, la red aprende de experiencias previas y no necesita modelar el entorno.

6.1 Conclusiones

En el caso del lago Ypacaraí, hemos aplicado *Deep Reinforcement Learning* con un agente Dueling-DQN en dos escenarios distintos. Para el caso del escenario Completamente Observable, la estrategia que asegura una mayor recompensa, derivado de como está formulada la ley de recompensa, es siempre permanecer en una zona de alto interés. Lo cuál es un comportamiento no deseable, ya que se busca realizar una cobertura de todo el lago aunque visitando con más frecuencia las zonas de mayor interés. Por otra parte, el desplazamiento ideal del vehículo a lo largo del mapa es de forma diagonal porque de esta manera se recolecta más información en cada paso. No obstante, la duración de la batería se ve afectada puesto que un movimiento en diagonal gasta más batería que cualquier otro. Por lo que se refiere al aprendizaje, se ha ido degradando en el tramo explotativo de éste, debido a que se ha aprendido muy rápido como para llegar a tener una generalización suficiente que permita seguir aprendiendo siendo explotativo.

Para el caso del escenario Parcialmente Observable, la estrategia que asegura una mayor recompensa, derivado otra vez de como está formulada la ley de recompensa, es análogo al del primer experimento, que es siempre permanecer en una zona de alto interés. Se observan las mismas evoluciones en las gráficas con valores similares.

En el tercer experimento llevado a cabo en el escenario Parcialmente Observable, se ha diseñado una ley de recompensa que premia explícitamente el descubrimiento de nuevas zonas y se estudia el efecto de ésta última contribución. El valor del hiperparámetro *discovery reward* que mejores resultados ha conseguido ha sido 0.5. Con ese valor, el agente visita zonas no descubiertas cada

vez que las zonas de mayor interés pierden su importancia temporal. En consecuencia, la media de porcentajes de los mapas visitados en la evaluación es del 80 %, el doble que antes y consiguiendo una media de idleness similar. Cuanto más se ha aumentado *discovery reward*, menos se ha tomado en cuenta la importancia de visitar con más reiteración a las casillas más importantes, lo que es un comportamiento indeseado. Además, se observa como la contribución de la recompensa a la exploración frente a la total es del 50 % en el caso de *discovery reward=0.5* y por tanto los valores del término referente a la exploración tienen que ser iguales a los valores de la recompensa que da visitar una zona de interés alto.

Por último, nuestro algoritmo es capaz de realizar un mejor desempeño que los algoritmos basados en heurísticas en el patrullaje no homogéneo. Consiguiendo una cobertura del 80 % del mapa frente a los 60 % del algoritmo cortacésped, el cual es un algoritmo de barrido que se enfoca sólo en cubrir todo el mapa sin tener en cuenta las zonas de mayor importancia. Además, obtiene en menos pasos más información y recompensa tanto en el algoritmo cortacésped como en los métodos aleatorios: Direcciones aleatorias y totalmente aleatorio. Frente a éste último llegamos a recoger el triple de interés en menos pasos y descubrir más del doble de la superficie navegable.

6.2 Líneas futuras

En este trabajo nos hemos enfocado en la monitorización de un lago llevado a cabo por un ASV, pero varios enfoques tratan de resolver este ‘problema haciendo uso de varios vehículos y entrenando a las redes para que no sólo tengan en cuenta la planificación de trayectorias sino también la evitación de obstáculos entre robots. En este nuevo enfoque, tambien hay que tomar en consideración la comunicación y sincronización entre el grupo de robots y la constante actualización del mapa de interés, que ahora depende de la lectura de los sensores de todos los vehículos.

Por otro lado, el algoritmo de aprendizaje puede ser mejorado con varias técnicas

- *Prioritized replay* [37]: Sirve para priorizar la experiencia, con el fin de reproducir las transiciones importantes con más frecuencia y, por tanto, aprender de forma más eficiente. La idea clave es que un agente de RL puede aprender más eficazmente de algunas transiciones que de otras. Las transiciones pueden ser más o menos sorprendentes, redundantes o relevantes para la tarea.
- Capa ruidosa [12]: Una capa lineal ruidosa es una capa lineal con ruido paramétrico añadido a los pesos. Esta estocasticidad inducida puede utilizarse en redes de aprendizaje por refuerzo para que la política del agente ayude a una exploración eficiente. Los parámetros del ruido se aprenden con el descenso de gradiente junto con los demás pesos de la red. El ruido gaussiano factorizado es el tipo de ruido que se suele emplear.

Por último, en futuro paso de investigación se ha considerado entrenar la red neuronal con algoritmos genéticos. Estos últimos, son métodos adaptativos basados en el proceso genético de los organismos vivos. En la naturaleza, los individuos luchan por los recursos limitados y aquellos más adaptados tendrán más probabilidad de sobrevivir y por tanto de reproducirse. El cruce entre dos individuos adaptados puede llegar a tener unos genes mejores que sus ancestros. Los algoritmos genéticos usan una analogía directa con el comportamiento natural. Generan una población de individuos en el que cada uno representa una solución y tiene una puntuación, que simboliza la bondad de dicha solución. Los individuos con mayor puntuación tendrán más probabilidad a ser seleccionados y por tanto sobrevivir. A continuación se resumen los pasos del algoritmo:

1. Generar una población inicial de soluciones.
2. Seleccionar las soluciones mejor adaptadas, es decir, las que tienen mayor puntuación.
3. Cruzar algunas soluciones con un operador de cruce para obtener su descendencia

4. Mutar algunas soluciones con operador de mutación para obtener las soluciones mutadas.
5. Elegir las soluciones que sobreviven y formarán la nueva generación.
6. Si no se alcanza el criterio de parada volver al paso 2.

La idea principal sería tratar los parámetros de la red neuronal como los individuos de una población. Los GAs son más explorativos que los demás métodos porque una población ya abarca diversas soluciones al mismo tiempo, y además, por efectos de cruzamiento amplían mucho más el espacio de búsqueda. En cambio, es muy probable que vaya a tardar en entrenar bastante, pero si funciona, el entrenamiento será muy estable porque los algoritmos genéticos tienen una convergencia estable.

Índice de Figuras

1.1 Vista cenital del lago Ypacaraí 1
1.2 Efecto de las cianobacterias 2
1.3 ASV realizando la monitorización en la superficie del lago 3
1.4 Diseño del Cormorán-II 4
1.5 Diagrama del control ASV en el caso real (arriba) y el bucle de aprendizaje de refuerzo (abajo) [26] 5

2.1 Sophia, robot humanoide con Inteligencia Artificial desarrollado por Hanson Robotics 8
2.2 Los robots de seguridad de la serie S5 de SMP Robotics están diseñados para sustituir a los guardias de seguridad que patrullan y para proporcionar una supervisión móvil de CCTV 10
2.3 Algoritmo de búsqueda de grafos 11
2.4 Trayectoria de un algoritmo de campos de potenciales artificiales 12
2.5 Ilustración esquemática de la red neuronal convolucional usada en [30] 13
2.6 Representación del entorno en celdas en [19] y [44]. 14
2.7 Arquitectura de los algoritmos Actor-Critic 15

3.1 Posible representación en grafo propuesto en [28] 17
3.2 Distribución de la contaminación en el lago dependiendo de la dirección del viento. De izquierda a derecha: Noreste, Sureste, Suroeste, Noroeste [32] 18
3.3 Mapa discretizado del lago Ypacaraí 19
3.4 Posibles acciones que puede tomar el ASV 20

4.1 Ciclo de aprendizaje por refuerzo 22
4.2 Ilustración visual del Proceso de decisión de Markov 23
4.3 Posibles valores de ε a lo largo de un entrenamiento de 1000 episodios 25
4.4 Modelo de una neurona artificial 27
4.5 Perceptrón Multicapa 28
4.6 Funciones de activación 29
4.7 Leaky ReLU 30
4.8 Ejemplo de una Red Neuronal Convolucional Densa 31
4.9 Operación de Maxpool 2X2 31
4.10 Ejemplo de una Red Neuronal Convolucional Densa con Operación de Maxpool 2X2 31
4.11 Q-Learning vs Deep Q-Learning 33
4.12 Buffer Replay 34
4.13 Esquema del algoritmo DQN 36
4.14 Ejemplo de matriz de interés R_{abs}(i,j) presentado visualmente como un mapa de calor 37

4.15 Región de detección del vehículo en la posición [26, 21] con radio r=2. Se observa como en este instante el interés de las celdas de dicha región es 0. 38
4.16 Estado del Escenario Completamente Observable. De izquierda a derecha: Posición del agente, Mapa del lago e Importancia relativa. 41
4.17 Estado del Escenario Parcialmente Observable. De izquierda a derecha: Posición del agente, Mapa del lago, Matriz de importancia y Mapa temporal 43
4.18 Dueling DQN 43

5.1 Valores de ε durante el entrenamiento 47
5.2 De izquierda a derecha los distintos pasos que se han hecho para discretizar el mapa del lago. 48
5.3 Cuatro ejemplos de las funciones aleatorias de Shekel tomadas como *ground-truth* 49
5.4 Valores de la recompensa acumulada durante el entrenamiento 51
5.5 Longitud de los episodios durante el entrenamiento 52
5.6 Movimientos del vehículos 52
5.7 Media de interés relativa 53
5.8 Gráfica de la función de pérdida 53
5.9 Evolución del porcentaje del mapa que se ha visitado 54
5.10 Evolucion de la media de la recompensa acumulada 55
5.11 Evolucion de la media de la suma de interés 55
5.12 Valores mínimos de la media de la importancia relativa 56
5.13 Valores del porcentaje del mapa visitado 56
5.14 Trayectoria del vehículo en el entorno Completamente Observable. 57
5.15 Valores de la recompensa acumulada durante el entrenamiento 58
5.16 Longitud de los episodios durante el entrenamiento 58
5.17 Media de interés 59
5.18 Gráfica de la función de pérdida 59
5.19 Evolución del porcentaje del mapa que se ha visitado 60
5.20 Evolución de la política durante el aprendizaje. Se ha guardado la política del agente cada 2000 episodios de entrenamiento y se han evaluado con los mismos 100 mapas de interés a cada política y se ha hecho una media de las métricas que se consideran interesantes. 60
5.21 Trayectoria del vehículo en el entorno Parcialmente Observable 61
5.22 Trayectoria del vehículo en el entorno Parcialmente Observable con dos picos de interés 62
5.23 Trayectoria del vehículo en el entorno Parcialmente Observable con un pico de interés de distribución Gaussiana creado con una máscara de filtro gaussiano de tamaño $23x23$, $\mu = 0$ y $\sigma = 2.5$ 62
5.24 Trayectoria del vehículo en el entorno Completamente Observable con un pico de interés de distribución Gaussiana creado con una máscara de filtro gaussiano de tamaño $23x23$, $\mu = 0$ y $\sigma = 2.5$ 63
5.25 Media de las recompensas 64
5.26 Media del porcentaje del mapa visitado 65
5.27 Media del porcentaje de la recompensa a la exploración 65
5.28 Media de la importancia relativa 66
5.29 Evolución de las métricas con el término *discovery reward* a lo largo de un episodio para cada política. En la leyenda se muestra cada valor de *discovery reward* con el color de su curva correspondiente 66
5.30 Trayectoria del vehículo en el entorno Parcialmente Observable con un pico de interés de distribución Gaussiana creado con una máscara de filtro gaussiano de tamaño $23x23$, $\mu = 0$ y $\sigma = 2.5$ 68
5.31 Evolución de la política durante el aprendizaje 69

5.32 Visualización de la trayectoria de un algoritmo cortacésped, las partes más oscuras de la trayectoria son las más recientes 70
5.33 Visualización de la trayectoria de un algoritmo de dirección aleatoria, las partes más oscuras de la trayectoria son las más reciente 70
5.34 Evolución de las métricas con el término *discovery reward* a lo largo de un episodio para cada política. En la leyenda se muestra cada valor de *discovery reward* con el color de su curva correspondiente 72

Índice de Tablas

1.1	Sistemas y subsistemas del vehículo	5
4.1	Resumen de la ley de recompensa	40
5.1	Hiperparámetros de entrenamiento	48
5.2	Configuración del Escenario Completamente Observable	51
5.3	Comparación de métricas de interés	71

Bibliografía

[1] Alberto Alvarez, Andrea Caiti, and Reiner Onken, *Evolutionary path planning for autonomous underwater vehicles in a variable ocean*, IEEE Journal of Oceanic Engineering **29** (2004), no. 2, 418–429.

[2] Mario Arzamendia, Derlis Gregor, Daniel Gutierrez Reina, and Sergio Luis Toral, *An evolutionary approach to constrained path planning of an autonomous surface vehicle for maximizing the covered area of ypacarai lake*, Soft Computing **23** (2019), no. 5, 1723–1734.

[3] Mario Arzamendia, Daniel Gutierrez, Sergio Toral, Derlis Gregor, Eleana Asimakopoulou, and Nik Bessis, *Intelligent online learning strategy for an autonomous surface vehicle in lake environments using evolutionary computation*, IEEE Intelligent Transportation Systems Magazine **11** (2019), no. 4, 110–125.

[4] Shalabh Bhatnagar, Mohammad Ghavamzadeh, Mark Lee, and Richard S Sutton, *Incremental natural actor-critic algorithms*, Advances in neural information processing systems **20** (2007).

[5] Murat Cakir, *2d path planning of uavs with genetic algorithm in a constrained environment*, 2015 6th International Conference on Modeling, Simulation, and Applied Optimization (ICMSAO), IEEE, 2015, pp. 1–5.

[6] K.P. Carroll, S.R. McClaran, E.L. Nelson, D.M. Barnett, D.K. Friesen, and G.N. William, *Auv path planning: an a* approach to path planning with consideration of variable vehicle speeds and multiple, overlapping, time-dependent exclusion zones*, Proceedings of the 1992 Symposium on Autonomous Underwater Vehicle Technology, 1992, pp. 79–84.

[7] Joseph Carsten, Dave Ferguson, and Anthony Stentz, *3d field d: Improved path planning and replanning in three dimensions*, 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2006, pp. 3381–3386.

[8] Yann Chevaleyre, *Theoretical analysis of the multi-agent patrolling problem*, Proceedings. IEEE/WIC/ACM International Conference on Intelligent Agent Technology, 2004.(IAT 2004)., IEEE, 2004, pp. 302–308.

[9] Shushman Choudhury, Nate Gruver, and Mykel J Kochenderfer, *Adaptive informative path planning with multimodal sensing*, Proceedings of the International Conference on Automated Planning and Scheduling, vol. 30, 2020, pp. 57–65.

[10] Matthew Dunbabin and Lino Marques, *Robots for environmental monitoring: Significant advancements and applications*, IEEE Robotics & Automation Magazine **19** (2012), no. 1, 24–39.

[11] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson, *Stabilising experience replay for deep multi-agent reinforcement learning*, International conference on machine learning, PMLR, 2017, pp. 1146–1155.

[12] Meire Fortunato, Mohammad Gheshlaghi Azar, Bilal Piot, Jacob Menick, Ian Osband, Alex Graves, Vlad Mnih, Remi Munos, Demis Hassabis, Olivier Pietquin, Charles Blundell, and Shane Legg, *Noisy networks for exploration*, 2017.

[13] Gregory Hitz, Enric Galceran, Marie-Ève Garneau, François Pomerleau, and Roland Siegwart, *Adaptive continuous-space informative path planning for online environmental monitoring*, Journal of Field Robotics **34** (2017), no. 8, 1427–1449.

[14] John H Holland, *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*, MIT press, 1992.

[15] Kurt Hornik, Maxwell Stinchcombe, and Halbert White, *Multilayer feedforward networks are universal approximators*, Neural Networks **2** (1989), no. 5, 359–366.

[16] Jordi Torres i Viñals, *Introducción al aprendizaje por refuerzo profundo: Teoría y práctica en python*, 1 ed., Independently published, 14 abril 2021.

[17] Fernando Izaurieta and Carlos Saavedra, *Redes neuronales artificiales*, Departamento de Física, Universidad de Concepción Chile (2000).

[18] Philip Jansen, Stearns Broadhead, Rowena Rodrigues, David Wright, Philip Brey, Alice Fox, and Ning Wang, *SIENNA D4.1: State-of-the-art Review: Artificial Intelligence and robotics*, June 2019.

[19] Anirudh Krishna Lakshmanan, Rajesh Elara Mohan, Balakrishnan Ramalingam, Anh Vu Le, Prabahar Veerajagadeshwar, Kamlesh Tiwari, and Muhammad Ilyas, *Complete coverage path planning using reinforcement learning for tetromino based cleaning and maintenance robot*, Automation in Construction **112** (2020), 103078.

[20] Kian Seng Lee, Mark Ovinis, T Nagarajan, Ralph Seulin, and Olivier Morel, *Autonomous patrol and surveillance system using unmanned aerial vehicles*, 2015 IEEE 15th International Conference on Environment and Electrical Engineering (EEEIC), IEEE, 2015, pp. 1291–1297.

[21] Siding Li, Xin Xu, and Lei Zuo, *Dynamic path planning of a mobile robot with improved q-learning algorithm*, 2015 IEEE international conference on information and automation, IEEE, 2015, pp. 409–414.

[22] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun, *Anytime dynamic a*: An anytime, replanning algorithm*, ICAPS'05, AAAI Press, 2005.

[23] Zhan Wei Lim, David Hsu, and Wee Sun Lee, *Adaptive informative path planning in metric spaces*, The International Journal of Robotics Research **35** (2016), no. 5, 585–598.

[24] Bo Liu, Yue Zhang, Shupo Fu, and Xuan Liu, *Reduce uav coverage energy consumption through actor-critic algorithm*, 2019 15th International Conference on Mobile Ad-Hoc and Sensor Networks (MSN), IEEE, 2019, pp. 332–337.

[25] Mario Eduardo Arzamendia López, *Reactive evolutionary path planning for autonomous surface vehicles in lake environments*, Ph.D. thesis, University of Seville, 2018.

[26] Samuel Yanes Luis, Daniel Gutiérrez Reina, and Sergio L Toral Marín, *A deep reinforcement learning approach for the patrolling problem of water resources through autonomous surface vehicles: The ypacarai lake case*, IEEE Access **8** (2020), 204076–204093.

[27] Liangheng Lv, Sunjie Zhang, Derui Ding, and Yongxiong Wang, *Path planning via an improved dqn-based learning policy*, IEEE Access **7** (2019), 67319–67330.

[28] Aydano Machado, Geber Ramalho, Jean-Daniel Zucker, and Alexis Drogoul, *Multi-agent patrolling: An empirical analysis of alternative architectures*, International workshop on multi-agent systems and agent-based simulation, Springer, 2002, pp. 155–170.

[29] Somaiyeh Mahmoudzadeh, David Powers, and Reza Zadeh, *State-of-the-art in uvs' autonomous motion planning: Augmented reactive mission and motion planning architecture*, pp. 31–40, 01 2019.

[30] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al., *Human-level control through deep reinforcement learning*, nature **518** (2015), no. 7540, 529–533.

[31] Farzad Niroui, Kaicheng Zhang, Zendai Kashino, and Goldie Nejat, *Deep reinforcement learning robot for search and rescue applications: Exploration in unknown cluttered environments*, IEEE Robotics and Automation Letters **4** (2019), no. 2, 610–617.

[32] Laura C Oporto, Derlis A Ramírez, Jhabriel D Varela, and Christian E Schaerer, *Analysis of contaminant transport under wind conditions on the surface of a shallow lake*, Mecánica Computacional **34** (2016), no. 31, 2155–2163.

[33] Martijn Otterlo and Marco Wiering, *Reinforcement learning and markov decision processes*, Reinforcement Learning: State of the Art (2012), 3–42.

[34] Claudio Piciarelli and Gian Luca Foresti, *Drone patrolling with reinforcement learning*, Proceedings of the 13th International Conference on Distributed Smart Cameras, 2019, pp. 1–6.

[35] Maria-Cecilia Rivara and Cristian Levin, *A 3-d refinement algorithm suitable for adaptive and multi-grid techniques*, Communications in applied numerical methods **8** (1992), no. 5, 281–290.

[36] Alvaro Salas, *Acerca del algoritmo de dijkstra*, arXiv preprint arXiv:0810.0075 (2008).

[37] Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver, *Prioritized experience replay*, 2015.

[38] Eduard Semsch, Michal Jakob, Dušan Pavlicek, and Michal Pechoucek, *Autonomous uav surveillance in complex urban environments*, 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology, vol. 2, IEEE, 2009, pp. 82–85.

[39] Rishi Shah, Yuqian Jiang, Justin Hart, and Peter Stone, *Deep r-learning for continual area sweeping*, 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2020, pp. 5542–5547.

[40] Laura Sorbi, Graziano Pio De Capua, Laura Toni, and Jean-Guy Fontaine, *Target detection and recognition: A mission planner for autonomous underwater vehicles*, OCEANS'11 MTS/IEEE KONA, IEEE, 2011, pp. 1–5.

[41] Richard S Sutton, *Learning to predict by the methods of temporal differences*, Machine learning **3** (1988), no. 1, 9–44.

[42] Richard S Sutton, Andrew G Barto, et al., *Introduction to reinforcement learning*, (1998).

[43] Ardi Tampuu, Tambet Matiisen, Dorian Kodelja, Ilya Kuzovkin, Kristjan Korjus, Juhan Aru, Jaan Aru, and Raul Vicente, *Multiagent cooperation and competition with deep reinforcement learning*, PloS one **12** (2017), no. 4, e0172395.

[44] Mirco Theile, Harald Bayerlein, Richard Nai, David Gesbert, and Marco Caccamo, *Uav coverage path planning under varying power constraints using deep reinforcement learning*, 2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), IEEE, 2020, pp. 1444–1449.

[45] Marco Trincavelli, Matteo Reggente, Silvia Coradeschi, Amy Loutfi, Hiroshi Ishida, and Achim J Lilienthal, *Towards environmental monitoring with mobile robots*, 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, IEEE, 2008, pp. 2210–2215.

[46] Hado Van Hasselt, Arthur Guez, and David Silver, *Deep reinforcement learning with double q-learning*, Proceedings of the AAAI conference on artificial intelligence, vol. 30, 2016.

[47] Tian Wang, Ruoxi Qin, Yang Chen, Hichem Snoussi, and Chang Choi, *A reinforcement learning approach for uav target searching and tracking*, Multimedia Tools and Applications **78** (2019), no. 4, 4347–4364.

[48] Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Hasselt, Marc Lanctot, and Nando Freitas, *Dueling network architectures for deep reinforcement learning*, International conference on machine learning, PMLR, 2016, pp. 1995–2003.

[49] MingYue Wei, *forgetting in deep learning*, Dec. 2020.

[50] Wikipedia, *Aprendizaje profundo*, Jun. 2020.

[51] ———, *Inteligencia artificial*, Jun. 2020.

[52] ———, *Robótica*, Jun. 2020.

[53] Jian Xiao, Gang Wang, Ying Zhang, and Lei Cheng, *A distributed multi-agent dynamic area coverage algorithm based on reinforcement learning*, IEEE Access **8** (2020), 33511–33521.

[54] Samuel Yanes Luis, Daniel Gutiérrez, and Sergio Toral, *A multiagent deep reinforcement learning approach for path planning in autonomous surface vehicles: The ypacarac-lake patrolling case.*, IEEE Access **PP** (2021).

[55] Zhao Yijing, Zheng Zheng, Zhang Xiaoyi, and Liu Yang, *Q learning algorithm based uav path learning and obstacle avoidence approach*, 2017 36th Chinese control conference (CCC), IEEE, 2017, pp. 3397–3402.

Glosario

AIPP Adaptive Informative Path Planning. 11

ASV Vehículos Autónomos de Superficie. 73, 74

CCPP Complete Coverage Path Planning. 12, 13

CNN Convolutional Neural Network. 30

DL Deep Learning. 8, 32

Double-DQN Double Deep Q-Network. 34, 35, 44, 54, 71

DQL Deep Q-Learning. 33

DQN Deep Q-Network. 33

DRL Deep Reinforcement Learning. 13, 32, 73

EC Eulerian Circuit. 12

HC Hamiltonian Circuit. 12

IA Inteligencia Artificial. 7, 10

IPP Informative Path Planning. 10, 11

MDP Proceso de decisión de Markov. 23, 25, 40

ML Machine Learning. 9

PP Path Planning. VII, 10, 11

RL Reinforcement Learning. 7, 13, 21–24, 26, 32, 40

TD Diferencia Temporal. 26