Trabajo Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática

Aprendizaje por Refuerzo Profundo Aplicado al Patrullaje Informativo Multiagente en Escenarios Hidrológicos

Autor: Dame Seck Diop

Tutor: Daniel Gutiérrez Reina, Samuel Yanes Luis

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

Sevilla, 2023

Trabajo Fin de Máster
Máster en Ingeniería Electrónica, Robótica y Automática

Aprendizaje por Refuerzo Profundo Aplicado al Patrullaje Informativo Multiagente en Escenarios Hidrológicos

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, 2023

Trabajo Fin de Máster: Aprendizaje por Refuerzo Profundo Aplicado al Patrullaje Informativo Multiagente en Escenarios Hidrológicos

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

Quiero expresar mi profundo agradecimiento a todas las personas que han estado a mi lado durante esta etapa significativa de mi vida. En primer lugar, me gustaría agradecer a mi familia, en particular a mis padres, por su inquebrantable paciencia y confianza en mí. A mi tío, por brindarme un apoyo invaluable en los momentos más difíciles.

A mis amigos, tanto los de toda la vida como aquellos que he conocido durante mis estudios, quienes han hecho que estos años sean más agradables. A mis abuelos, tíos y primos, por su constante creencia en mí y por ser una parte esencial de mi vida.

Y, por supuesto, a mis tutores, Samuel Yanes Luis y Daniel Gutiérrez Reina, por confiar en mí para llevar a cabo este trabajo y por su valiosa orientación a lo largo de todo el proceso.

Dame Seck Diop
Escuela Técnica Superior de Ingeniería de Sevilla.

Sevilla, 2023

Resumen

En este proyecto de fin de carrera, se presenta la implementación de un algoritmo de Aprendizaje por Refuerzo Profundo Multiagente para abordar el problema de patrullaje informativo en un escenario hidrológico parcialmente observable. El objetivo es supervisar y monitorear los recursos hídricos utilizando vehículos autónomos equipados con sensores de calidad del agua. El patrullaje se divide en dos fases: exploración y explotación. Para ello, se utilizan redes neuronales con dos "*heads*" o terminaciones paralelas, donde cada una estima la función Q correspondiente a cada fase. Se introduce una variable para determinar qué función Q utilizar en cada momento para tomar decisiones. El trabajo se centra en el diseño e implementación del algoritmo, evaluando su rendimiento en un escenario hidrológico y analizando los resultados obtenidos. En definitiva, el objetivo final es mejorar la eficiencia y efectividad del patrullaje informativo en entornos hidrológicos mediante el uso de técnicas de aprendizaje por refuerzo profundo multiagente.

Abstract

In this thesis project, the implementation of a Multiagent Deep Reinforcement Learning algorithm to address the informational patrolling problem in a partially observable hydrological scenario is presented. The objective is to supervise and monitor water resources using autonomous vehicles equipped with water quality sensors. The patrolling is divided into two phases: exploration and intensification. For this purpose, neural networks with two parallel heads, where each one estimates the Q function corresponding to each phase. A variable is introduced to determine which Q function to use at each time to make decisions. The work focuses on the design and implementation of the algorithm, evaluating its performance in a hydrological scenario and analyzing the results obtained. Ultimately, the final objective is to improve the efficiency and effectiveness of informative patrolling in hydrological environments through the use of multi-agent deep reinforcement learning techniques.

Índice

1 Introducción

1.1 Antecedentes

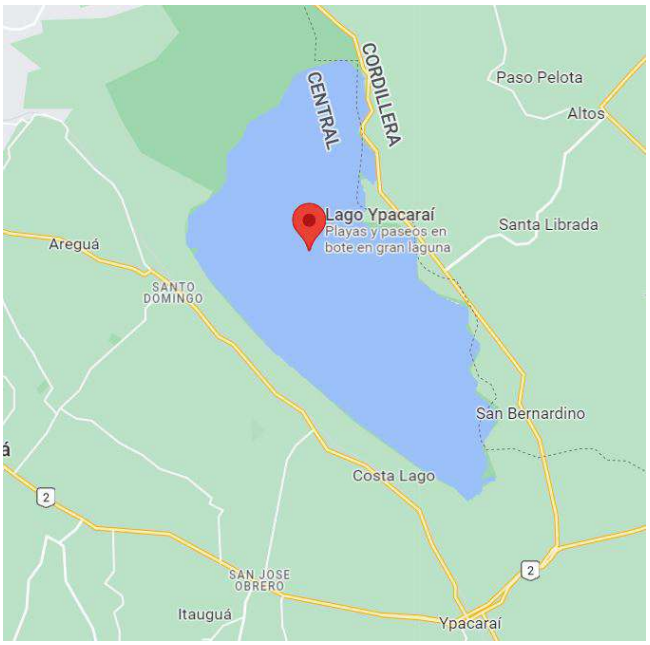


Figura 1.1 Vista cenital del lago Ypacaraí.

El lago Ypacaraí es el cuerpo de agua más grande de Paraguay con más de $60km^2$ de superficie navegable y $3m$ de profundidad. Este lago desempeña un papel fundamental como fuente de abastecimiento de agua en las ciudades que lo rodean. Con el paso de los años, su importancia ha aumentado gracias al turismo, ya que se utiliza como un lago recreativo para que las personas se bañen y para la navegación en pequeñas embarcaciones. Además, su valor también está relacionado con la rica vida natural que se desarrolla en los humedales de la cuenca que rodea al lago.

A causa del desarrollo económico de las ciudades cercanas, la falta de sistemas de alcantarillado durante muchos años y el aumento del turismo, la cantidad de desechos vertidos al agua ha experimentado un aumento significativo durante estos últimos años [1]. Debido a la grave contaminación en el lago, se produjo un fenómeno conocido como eutrofización, que es un proceso antinatural caracterizado por una acumulación excesiva de nutrientes en el agua.

Debido al enriquecimiento artificial de nutrientes en el agua, se ha producido un crecimiento excesivo de cianobacterias en el lago (Figura 1.2). Estos organismos consumen rápidamente el oxígeno disuelto en el agua, lo que provoca la falta de oxígeno y la pérdida de vida acuática. La contaminación causada por los desechos humanos de las ciudades y las industrias circundantes ha resultado en un aumento significativo de las colonias de cianobacterias en el lago. Estos desechos, ricos en nitrógeno, fósforo y metales, actúan como un catalizador para el crecimiento de bacterias y suciedad en toda el área navegable del lago, incluyendo las zonas de humedales. Esta situación no solo afecta negativamente al ecosistema acuático de la cuenca, sino que también pone en peligro la calidad del suministro de agua de las ciudades cercanas, ya que el lago Ypacaraí es su única fuente de recursos hídricos.

Figura 1.2 Efecto de las cianobacterias. Desde 2012, ha habido una proliferación agresiva de algas verde-azuladas en el lago, lo cual ha generado olores desagradables y la producción de toxinas, como la microcistina, que es perjudicial para la fauna y los humanos, y en ocasiones puede ser mortal.

La aparición de brotes de cianobacterias es impredecible y su comportamiento es caótico, lo que dificulta predecir dónde y cuándo ocurrirán. Además, las floraciones de cianobacterias no son constantes y su tamaño varía con el tiempo, lo que dificulta el proceso de medir la calidad del agua de manera precisa y constante.

Es importante destacar que la contaminación en el lago Ypacaraí se distribuye de manera desigual en su superficie. Existen áreas específicas donde se concentra en mayor medida debido a diferentes factores. Por ejemplo, en zonas cercanas a ciudades donde la falta de infraestructura de alcantarillado y sistemas de filtrado séptico provoca un aumento en los vertidos y la presencia de bacterias. Asimismo, las áreas portuarias recreativas y las zonas industriales cercanas,

como el Puerto de San Blas en la parte sur del lago, también contribuyen a la concentración de contaminantes.

La situación se vuelve aún más complicada para las autoridades, ya que se ven obligadas a cerrar el lago a los turistas debido a las floraciones de cianobacterias, las cuales son cíclicas e impredecibles. La única solución para abordar este problema radica en adoptar un enfoque multidisciplinario que involucre inversiones públicas en infraestructuras adecuadas y estudios bioquímicos de las aguas. Es necesario llevar a cabo un monitoreo constante y realizar estudios exhaustivos para comprender en detalle el estado actual del lago. Esto permitirá encontrar soluciones efectivas para revertir la situación. También es necesario implementar medidas que aborden tanto la fuente de contaminación como las consecuencias de las floraciones, con el fin de restaurar y mantener la calidad del agua en el lago Ypacaraí.

1.2 Uso de ASVs

Es esencial realizar un muestreo periódico y efectivo en áreas específicas de interés, donde se produzcan floraciones de algas o haya una acumulación de residuos y basura en el lago. Esto proporcionará información actualizada sobre el estado biológico de las aguas. Además, este mapa de contaminación resultante permitirá evaluar la eficacia de las medidas medioambientales implementadas por las autoridades y los investigadores.

Sin embargo, la monitorización manual requiere una cantidad considerable de recursos humanos y esfuerzo, ya que implica desplazamientos constantes en embarcaciones motorizadas desde la orilla hasta las áreas principales de floración para realizar muestreos manuales de las aguas. Además, debido al gran tamaño del lago Ypacaraí (60 km2), el muestreo manual sería ineficiente y lento para los biólogos, especialmente considerando que la situación de las cianobacterias puede cambiar diariamente. Por lo tanto, se necesita un sistema de monitoreo reactivo que permita un seguimiento continuo de la contaminación del lago.

Figura 1.3 ASV en las orillas del lago.

Se ha planteado la utilización de Vehículos Autónomos de Superficie, ASV de sus siglas en inglés, para sustituir las inspecciones humanas. El uso ASVs ha ganado impulso en áreas como el patrullaje [2][3] y la planificación informativa de trayectorias [4][5]. Estos vehículos están equipados con diversos sensores, como pH y oxígeno disuelto, que les permiten realizar mediciones de contaminación y podrían sustituir fácilmente a una estación de muestreo estática, proporcionando una forma más eficiente de medir la contaminación. Estos vehículos presentan muchas ventajas en comparación con los barcos normales:

- Son más respetuosos con el medio ambiente, ya que utilizan motores eléctricos y pueden ser alimentados mediante paneles solares, lo que reduce su impacto ambiental y los costos de combustible.
- Capacidad para operar de manera autónoma, es decir, sin necesidad de una tripulación a bordo. Esto los hace especialmente adecuados para tareas que podrían ser peligrosas o difíciles de realizar por humanos.
- Gracias a su tamaño reducido, propulsión eléctrica y autonomía, los ASVs también pueden realizar misiones de larga duración.

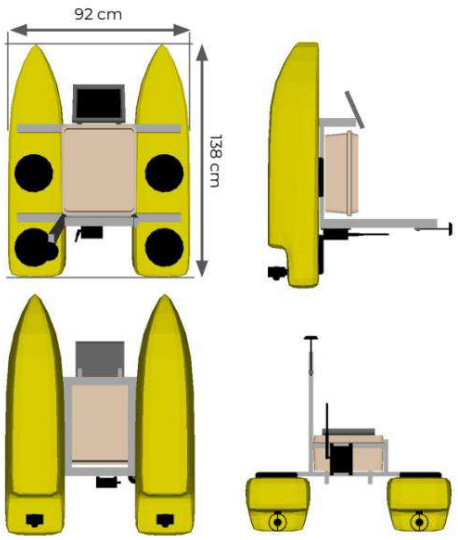


Figura 1.4 Diseño del Cormorán-II.

Para aumentar la eficiencia, se propone desplegar una flota de ASVs para que cada uno pueda explorar diferentes áreas y medir la calidad del agua. Sin embargo, esto requiere una coordinación adecuada entre los vehículos autónomos para evitar colisiones y asignar eficientemente las zonas de exploración. El objetivo es maximizar la cobertura del muestreo y optimizar la utilización de

los recursos disponibles.Por lo tanto, se busca establecer una estrategia de coordinación entre los ASVs para llevar a cabo la tarea de monitoreo de manera eficiente y efectiva.

Se han adquirido cuatro drones acuáticos de pequeño tamaño con características similares, uno de los cuales se encuentra en las instalaciones de la Escuela Técnica Superior de Ingeniería de Sevilla. Estos drones acuáticos son de tipo catamarán y se ha diseñado una arquitectura funcional que consta de varios sistemas, como se muestra en la Tabla 1.1. Además, en la Figura 1.4, se puede apreciar el diseño del ASV desde cuatro ángulos diferentes.

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.

En cuanto a la misión de los vehículos autónomos, cada uno está programado para realizar una planificación local de trayectorias y una planificación global de trayectorias. La planificación global de trayectorias es una tarea de alto nivel que consiste en decidir los mejores puntos de paso para alcanzar el objetivo, teniendo en cuenta diferentes criterios de optimización, como la redundancia de visitas o la maximización del área de cobertura.Por otro lado, la planificación local implica trazar la ruta física desde un punto de partida hasta un punto de destino, evitando obstáculos en el camino. Para adaptar la trayectoria calculada por el planificador global a situaciones imprevistas, como la presencia de obstáculos móviles, se utilizan sensores como LIDAR y cámaras. La arquitectura hardware-software del ASV permite seguir una secuencia de puntos de paso (waypoints) que son previamente calculados e interpolados por un generador de trayectorias, como un interpolador spline.En la Figura 1.5 se muestra diagrama del control ASV así como el bucle de aprendizaje de refuerzo que servirá como planificador global.

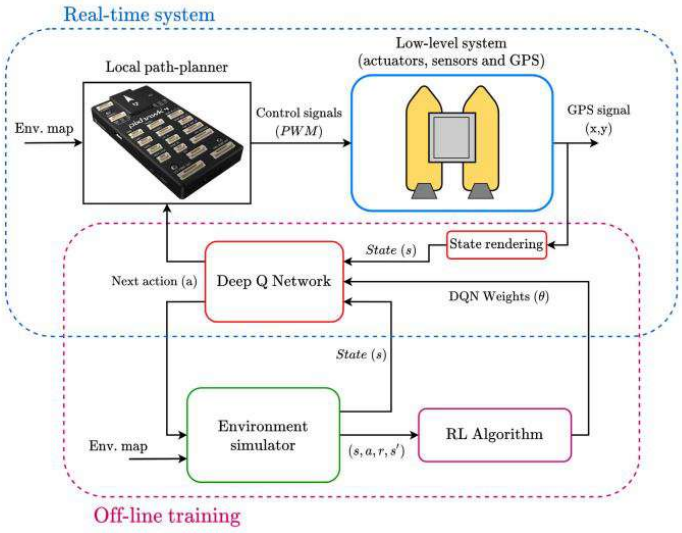


Figura 1.5 Diagrama del control ASV en el caso real (arriba) y el bucle de aprendizaje de refuerzo (abajo).

1.3 Objetivos

En este proyecto, se busca diseñar un planificador de alto nivel que determine el siguiente punto a visitar para el vehículo autónomo. Se enfrenta a un problema de decisión secuencial, donde en cada momento el agente recibe información limitada del entorno y debe tomar la mejor decisión de acuerdo con sus objetivos.

Este problema se puede abordar como un problema de patrullaje, como se ha hecho en trabajos anteriores como [3] y [5]. Hay dos perspectivas desde las cuales se puede analizar este problema:

- Importancia homogénea: Se considera que todas las zonas del lago tienen la misma importancia y por lo tanto, el ASV debe visitar todos los puntos de paso con la misma frecuencia para garantizar una cobertura equitativa.
- Importancia no homogénea: se considera una importancia diferente para cada zona del lago y las que son más importantes deben ser más visitadas con más frecuencia.

Como se mencionó anteriormente, hay zonas del lago más contaminadas que otras, por lo que la monitorización en esos lugares debe ser más frecuente, estamos ante un caso de patrullaje no homogéneo. Sin embargo, al inicio de la misión, los vehículos parten sin tener información sobre la contaminación del lago y deben descubrirla para luego intensificar la vigilancia en las zonas más contaminadas.

En el trabajo [5], los autores lograron resultados positivos en la cobertura no homogénea utilizando múltiples vehículos, en comparación con métodos clásicos como el algoritmo de cortacésped o la búsqueda aleatoria. Para abordar el caso multiagente, se empleó una red neuronal convolucional centralizada, compartida por todos los agentes, para extraer características y permitir

que los agentes tomen decisiones. Esta arquitectura se benefició de que las experiencias de los agentes son equivalentes, lo que condujo a obtener mayores recompensas en la mayoría de los casos en comparación con la arquitectura contrapartida, donde cada vehículo utiliza una red neuronal entrenada de forma independiente. Sin embargo, es importante destacar que en el trabajo [5] no abordaron explícitamente el problema de la falta de información sobre la contaminación al inicio de la misión. Esto implica que los vehículos deben descubrir la información sobre la contaminación durante el transcurso del episodio antes de poder tomar decisiones para intensificar la vigilancia en las zonas más relevantes.

En este proyecto, se busca seguir el trabajo realizado en [5] con el objetivo de desarrollar un algoritmo de patrullaje con una mayor resolución y abordar el problema desde una perspectiva más realista. Para lograrlo, se plantean dos fases principales en el patrullaje:

1. **Fase de exploración**: En esta fase, los vehículos se encargan de visitar todas las zonas del mapa y tomar medidas de contaminación.
2. **Fase de explotación**: Una vez que se han identificado las zonas más importantes del lago en términos de contaminación, se intensifica la vigilancia y monitoreo en estas áreas.

Los objetivos principales son los siguientes:

1. Diseñar un método para un planificador de rutas multiagente basado en Deep Reinforcement Learning escalable en el contexto del patrullaje no homogéneo del Lago Ypacaraí.
2. Presentar un enfoque novedoso para supervisar Escenarios Dinámicos Parcialmente Observables que consistirá en un mecanismo de transición suave entre la fase explorativa y explotativa.

2 Estado del arte

2.1 Deep Reinforcement Learning

Deep Reinforcement Learning (DRL) es una técnica de Machine Learning (ML) que combina Reinforcement Learning (RL) [6] y Deep Learning (DL) para entrenar a un agente a tomar las acciones que maximizan una recompensa en un entorno. En 2015, DRL dufrió una gran popularización debido al algoritmo Deep Q-Network (DQN), este fue capaz de alcanzar un nivel comparable al de un humano profesional jugando a un conjunto de 49 juegos de la famosa consola Atari 2600. En 2016, [7] propusieron la Double Deep Q-Network (Double-DQN) para mitigar el efecto de la sobreestimación de los valores estado-acción. En ese mismo año, se introdujo la Dueling Deep Q-Network (Dueling-DQN), que separa la función de valor Q en dos flujos para estimar mejor los valores de estado y la ventaja de toma una acción por separado. Por último, Rainbow DQN [8], fue introducido en 2017, como una combinación de varias de las mejoras antes mencionadas (Double-DQN, Dueling-DQN,Prioritized Buffer Replay etc...) para lograr un rendimiento de mejor en los juegos Atari. El algoritmo DQN y sus variantes se han usado en un amplio rango de aplicaciones ya que es capaz de aprender políticas de control directamente a partir de entradas sensoriales de alta dimensión usando RL.

En particular, en el ámbito del patrullaje informativo se han llevado a cabo numerosas aplicaciones [9, 10, 11]. En [9], se entrena una Double-DQN a aprender una política de control que se generaliza en función de las distintas restricciones de potencia un vehículo aéreo no tripulado (UAV). Esta red neuronal debe tomar decisiones de control para el UAV, equilibrando el presupuesto de potencia limitado y el objetivo de cobertura mediante el uso de canales de entrada similares a mapas para proporcionar al agente información espacial a través de capas de redes convolucionales. En [10] el entorno contiene espacios no navegables y además las zonas tienen diferentes requisitos de cobertura Figura 2.1. Así que, dado un mapa de relevancia que representa los requisitos de cobertura, el algoritmo DRL elige de forma autónoma las mejores acciones del dron para optimizar la cobertura. Por el contrario, en [11] el robot no conoce de antemano la distribución de los eventos relevantes y, sin embargo, debe aprender a maximizar la tasa de detección de eventos de interés.

También se han combinado DRL con patrullaje informativo para la monitorización de espacios

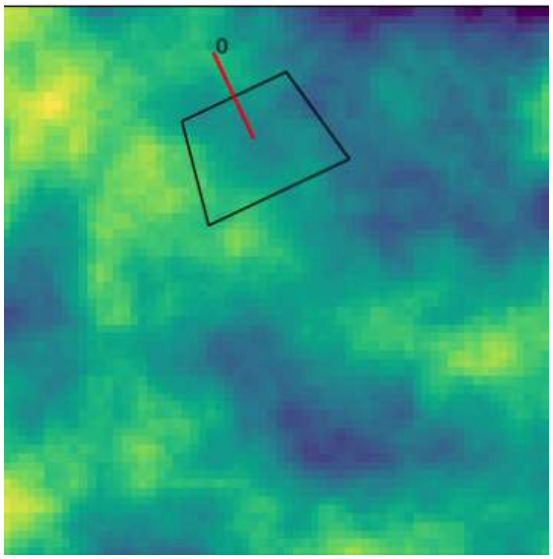


Figura 2.1 Mapa de relevancia con superposición de la posición del agente (marcada con un 0) y la zona observada.

medioambientales. En [3], los autores modelaron el mapa de contaminación del lago Ypacaraí y construyeron una imagen que representa un mapa de interés que tenía valores más altos en las zonas más contaminadas. Esto último, junto con una imagen que indica las zonas navegables del lago y otras imágenes que proporcionan información temporal sobre el recorrido del vehículo, eran las entradas a una Double-DQN. Esta red fue capaz de aprender a visitar las zonas más contaminadas con mayor frecuencia e incluso aprender a encontrar estas zonas si estas le eran desconocidas al principio del patrullaje. En este trabajo utilizaremos esta metodología de representación del entorno junto con otras adiciones que se mencionarán más adelante.

Recientemente, el patrullaje informativo con DRL se ha extendido al caso multiagente (Multi-agent DRL (MADRL)) debido a que mejora las prestaciones del caso monoagente. Sin embargo, introduce nuevos retos: descoordinación, asignación de créditos, exploración global, sobregeneralización relativa y la escalabilidad [5]. Además, el caso multiagente implica una complejidad mucho mayor en la planificación, ya que necesita una optimización de la planificación de la trayectoria y la consideración de la posible interacción entre los vehículos. Por otra parte, el espacio de estados y acciones crece exponencialmente con el número de agentes y por ello la complejidad computacional puede volverse insostenible incluso para un enfoque de aprendizaje profundo. El entrenamiento de los agentes se puede distinguir en dos paradigmas [12]: distribuido y centralizado. En el entrenamiento distribuido [13][14][15][16] los agentes aprenden sin comunicarse entre ellos y cada uno optimiza una política propia sin intercambio de información con otros agentes. Sin embargo, la principal limitación de este este enfoque es que para un único agente el estado se vuelve no estacionario debido a que percibe a los demás agentes como parte del estado. Por otro lado, la búsqueda de una política óptima influye en la toma de decisiones de los demás agentes, lo que puede puede provocar *action shadowing*. Por lo tanto, cada acción seleccionada se empareja con una acción arbitraria elegida por el otro agente y se producen un conjunto de

políticas subóptimas que hacen que se subestimen los valores de utilidad de las acciones óptimas. Para aliviar esto, en [14] se propuso un mecanismo de intervalos de actualización negativos Double-DQN que elimina los datos del *Buffer Replay* que conducen a la descoordinación. En el entrenamiento centralizado las políticas se optimizan en base al intercambio mutuo de información entre los agentes [17][18][19].

Si la ejecución es centralizada (CTCE), los agentes se modelan como un meta-agente con un conjunto de acciones A': A_i es el conjunto de acciones del agente i y $|A^i| = \prod_{i=1}^{N}(A_i) = |\{a_1, a_2, a_3, ..., a_N\}|$ es la dimensión conjunta de acciones de N agentes. Esta formulación está limitada por ser poco escalable ya que explota dimensionalmente con un gran número de agentes [18]. Por otro lado, la ejecución decentralizada (CTDE) es el enfoque más exitoso en la actualidad [12]. Esta es, los agentes comparten información en el entrenamiento, pero cada uno tiene su propia política a la hora de ejecutar las acciones. En [17], los autores aprovecharon la homogeneidad entre los agentes para hacer el entrenamiento en una única red y compartir los parámetros. Demostraron que las políticas que emergían de esta manera eran más eficientes y podían llegar a dar mejor resultados que las políticas aprendidas de forma autónoma. Algunos autores han aplicado métodos en los que se aprende una aproximación de la función de valor a partir de la información mutua. Los agentes se aprovechan de las acciones conjuntas, la información mutua y las políticas de otros agentes que están disponibles durante el entrenamiento e incorporan esta información extra en funciones de valor centralizadas, aunque dicha información se descarta en el momento de la prueba [20].

En este trabajo se diseña y se entrena una única red neuronal compartida por todos los agentes y cada agente será la copia exacta de la misma red. También se utilizan métodos de Iteración de Valores (DDQL) porque suelen ser más eficientes en cuanto a muestras que los métodos de Iteración de Políticas como [21]. Sin embargo, las acciones se toman en base a la aproximación de la función de valor estado-acción y dicha información no se descarta como en [20]. Además, sólo existe una red y una función Q que optimizar para cualquier número de agentes, lo cual aumenta la escalabilidad de nuestro método considerablemente.

2.2 Multi-Objective Reinforcement Learning

Varios problemas a resolver en el mundo real exigen el cumplimiento de múltiples objetivos en conflicto que deben equilibrarse. En RL se optimiza una política en base a una recompensa escalar, por tanto en Multi-Objective Reinforcement Learning (MORL) la recompensa se extiende a un vector donde cada componente representa la recompensa escalar de cada objetivo [22]. Los métodos MORL se pueden clasificar en dos categorías: *single-policy* y *multi-policy*.

Los enfoques *single-policy* [23, 24] consisten en aprender una única política basada en una escalarización lineal de los objetivos. Esto convierte el MORL en un problema de RL con un único objetivo, que es una suma lineal ponderada de los valores objetivo, donde las ponderaciones indican la importancia relativa de los objetivos. La escalarización lineal sólo puede encontrar políticas en frentes de pareto convexos, por ello, en [23] proponen una escalarización no lineal utilizando la función de Chebyshev. Esto permitía obtener una mejor dispersión entre el conjunto de soluciones óptimas pertenecientes al frente de Pareto. Sin embargo, los enfoques *single-policy* requieren de información previa sobre las preferencias del usuario que pueden ser variantes y un

pequeño cambio en ellas puede producir variaciones significantes en la solución [25]. Técnicas como Distillation of a Mixture of Experts (DiME) [26] y MO-MPO [27] toman un enfoque totalmente ortogonal a esto y codifican las preferencias mediante restricciones sobre la influencia de cada objetivo en la actualización de la política, en lugar de mediante la escalarización.

Por otro lado, las técnicas *multi-policy* se basan en encontrar el conjunto de políticas que forman parte del frente de Pareto. Varios trabajos como [28, 29, 30] buscan puntos pareto dominantes resolviendo una serie de problemas de objetivo único para diferentes vectores de escalarización lineal. El conjunto resultante se denomina Convex coverage set (CCS): el conjunto de soluciones óptimas para todas las ponderaciones posibles para objetivos ponderados linealmente. También se han intentado buscar las políticas óptimas de forma simultánea adaptando el algoritmo Q-learning al caso multiobjetivo [31, 32, 33]. Adicionalmente, existen otras técnicas que buscan múltiples políticas en el espacio de parámetros de políticas para generar infinitas soluciones Pareto-óptimas en una sola ejecución [34, 35] o combinando enfoques de política única con un objetivo general [36].

En nuestro trabajo se reciben dos recompensas en cada paso y por tanto la recompensa es un vector de dos componentes, pero no se intenta encontrar una política que actúe en base a ponderaciones y preferencias. En contraste, existen dos fases, exploratoria e intensificatoria, y en cada una se debe de actuar en base a su respectiva recompensa.

2.3 Multi-Task Reinforcement Learning

En Multi-Task Reinforcement Learning (MTRL) el agente tiene como objetivo aprender varias tareas a la vez. En la mayoría de los casos, estas tareas comparten, al menos parcialmente, el espacio de estados y acciones pero difieren en la recompensa obtenida [37]. Por lo tanto, el objetivo principal del aprendizaje multitarea es entrenar en paralelo las tareas individuales relacionadas con una representación compartida del entorno. Para llevar esto a cabo, en [38] modelaron la distribución sobre los Proceso de decisión de Markov (MDP) utilizando un modelo jerárquico bayesiano de mezcla infinita. Para cada nueva distribución MDP, utilizaron la distribución aprendida previamente como información previa. Esto aceleraba el aprendizaje de nuevas características de un MDP nuevo y además, el uso de un modelo no paramétrico (jerárquico bayesiano de mezcla infinita) les permitía adaptarse rápidamente a entornos que no habían encontrado antes.

En cuanto a entornos parcialmente observables, en [39] introdujeron Regionalized Policy Representation (RPR) para caracterizar el comportamiento del agente en cada entorno y un proceso de Dirichlet sobre las RPRs en múltiples tareas para agruparlas. En [40] hacen uso de *policy sketches*, que anotan tareas con secuencias de subtareas nombradas, proporcionando información sobre las relaciones estructurales de alto nivel entre las tareas, pero no sobre cómo implementarlas. Para aprender de los *policy sketches*, presentaron un modelo que asocia cada subtarea con una subpolítica modular y maximiza conjuntamente la recompensa sobre políticas de tareas completas vinculando parámetros a través de subpolíticas compartidas. La optimización se logra mediante un entrenamiento de una red actor-crítico desacoplado que facilita el aprendizaje de comportamientos comunes a partir de múltiples funciones de recompensa diferentes.

Para aprovechar las similitudes en los contextos de diferentes brazos del entorno Multi-armed bandit, los autores de [41] presentaron una técnica MTRL para estimar las semejanzas entre tareas

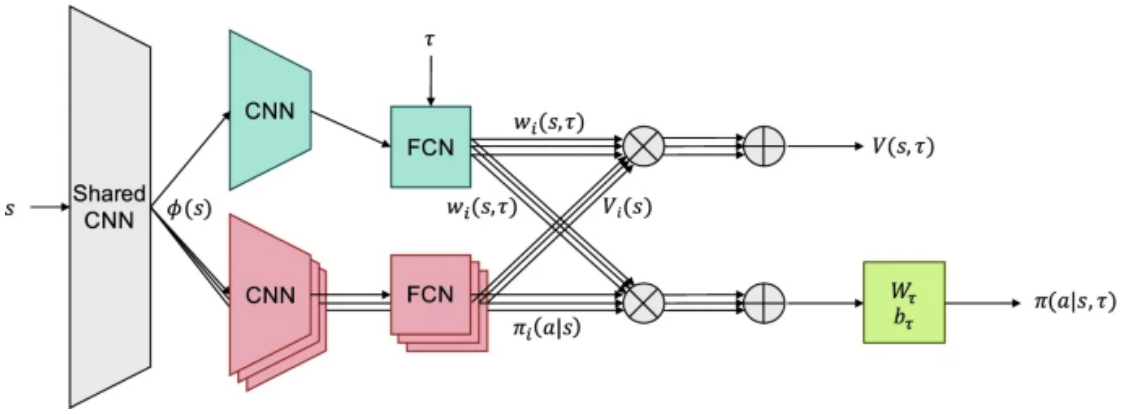


Figura 2.2 Las primeras capas convolucionales son compartidas entre la red de atención y las subredes. Las políticas ponderadas de las subredes se transforman mediante una capa específica de la tarea (en verde) para tener en cuenta el diferente número de acciones posibles en las distintas tareas.

a partir de los datos. Esto permitió mejorar la predicción de las recompensas dado un contexto. En contraste, en [42] propusieron un enfoque MTRL basado en una *attention-network* sin utilizar ninguna información previa sobre las relaciones entre tareas. Esta *attention-network* (Figura 2.2) agrupa automáticamente el conocimiento obtenido de las tareas en subredes con una granularidad a nivel de estado. La salida final es una combinación lineal de las salidas de las subredes, esto permite que todas las subredes que son útiles contribuyan a la política y a la función de valor.

En [43] presentaron un marco de intercambio de experiencia para detectar, mediante las recompensas de cada tarea, las semejanzas entre tareas, llamadas región compartida. Con esto se podía decidir qué experiencias podían compartir las tareas. También han surgido varios métodos basados en nuevos diseños [44, 45, 46]. En [44] proponen un marco de trabajo para entrenar agentes a aprender una política jerárquica que es capaz de decidir cuando entrenar una nueva política o cuando reutilizar una política aprendida previamente. Esto es posible gracias a que se le proporciona una gramática temporal estocástica que le indica cuándo debe confiar en habilidades previamente aprendidas y cuándo debe ejecutar nuevas habilidades. La gramática temporal es un conjunto de reglas que indican cuándo y cómo se deben utilizar ciertas habilidades en función del contexto temporal.

Por otra parte, en [45] introducen Predictions of Bootstrapped Latents (PBL), un algoritmo de aprendizaje de representación autosupervisado que se basa en representaciones predictivas multipaso de observaciones futuras y se centra en capturar información estructurada sobre la dinámica del entorno. Esta técnica utiliza Redes Neuronales Recurrentes (RNNs por sus siglas en inglés) para comprimir historias completas H_t (estados del agente) como B_t e historias parciales $H_{t,k}$ como $B_{t,k}$, como entradas para tareas auxiliares de predicción. En concreto, PBL consta de dos tareas auxiliares de predicción: una predicción directa, condicional a la acción, de historias parciales comprimidas a futuras observaciones latentes y una predicción inversa, de observaciones latentes a estados del agente.

Por otro lado, en [46] estudian los problemas del enfoque del aprendizaje multitarea. Concretamente la necesidad de encontrar un equilibrio entre las necesidades de múltiples tareas, que compiten por los limitados recursos de un único sistema de aprendizaje. La importancia de una tarea para el agente aumenta con la escala de las recompensas observadas en esa tarea y estas pueden diferir arbitrariamente de una tarea a otra. Esto hace que el algoritmo se centre en esas tareas más importantes a expensas de la generalidad. Para evitar esto, los autores utilizan la normalización PopArt [47] para hacer una actualización de una red *actor critic* invariante la escala y escasez (*sparsity*) de las recompensas, permitiendo grandes mejoras de rendimiento en agentes multitarea paralelos. Además, propusieron adaptar automáticamente la contribución de cada tarea a las actualizaciones del agente para que todas las tareas tengan impactos comparables en la dinámica de aprendizaje.

En este trabajo el agente aprende a optimizar dos tareas pero no se efectúan en paralelo, como los trabajos anteriormente mencionados. En el primer tramo del episodio, se toman decisiones en base a una política optimizada sólo para la tarea de exploración, mientras que en el segundo tramo se toman acciones siguiendo una política explotativa. Como se ha mencionado anteriormente, cada agente tiene una copia idéntica de la red neuronal, la cuál tiene unas primeras capas convolucionales compartidas por ambas políticas, y luego diverge en dos subredes que estiman una función Q cada una.

3 Planteamiento del problema

3.1 Problema del vigilante: caso de un único agente

El *problema del vigilante* o *patrolling problem* consiste en encontrar una estrategia óptima de patrullaje que determine la mejor ruta para visitar continuamente zonas relevantes de un entorno para supervisarlo o protegerlo. La relevancia de cada zona puede variar dependiendo del objetivo del patrullaje. Atendiendo a esta importancia, se pueden considerar dos variantes del problema de patrullaje:

- **Patrullaje homogéneo**: se busca cubrir todas las zonas por igual ya que la importancia es homogénea.
- **Patrullaje no homogéneo**: existen zonas más importantes que otras y deben ser cubiertas con una mayor frecuencia.

En [48] el problema del vigilante es modelado como un Traveling Salesman Problem (TSP). El problema del vendedor viajero (en español) es un problema de optimización combinatoria que originalmente se formuló de la siguiente manera: dado un conjunto de ciudades en un mapa y las distancias entre cada par de ellas, encontrar el camino más corto tal que cada ciudad sea visitada solo una vez y se finalice en la ciudad de origen. 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 este trabajo, el objetivo del ASV que supervisa el lago es similar al TSP con zonas de mayor importancia, pero los ASVs también deben primero obtener el mapa del lago mediante exploración (cobertura total). Para representar una superficie continua, se puede recurrir a la técnica de la *esqueletonización* del terreno [49]. Esto es, 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. Con esta formulación se pueden representar diferentes tipos de problemas dependiendo del significado de los costes asociados a las aristas. En nuestro caso este coste es la distancia entre nodos. En la Figura 3.1 se muestra una posible representación en grafo propuesto en [50].

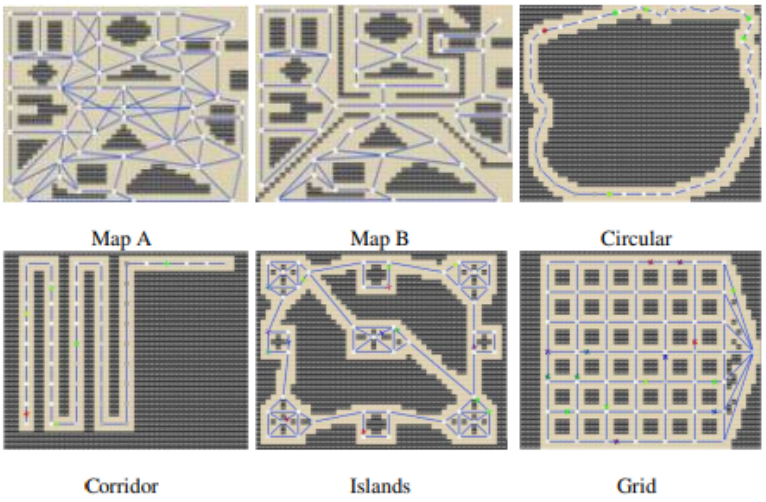


Figura 3.1 Posible representación en grafo.

En [49] también se introduce el concepto de *idleness*, W (en español ociosidad), de un nodo. Este es la cantidad de tiempo transcurrido desde que ese nodo ha recibido la visita de un agente. A partir de esto, se pueden definir varios criterios de evaluación:

- *Idleness* medio del grafo: Media del *idleness* de todos los nodos.
- Peor *idleness*: Valor del *idleness* más alto entre los nodos del grafo.

En este proyecto, nuestro objetivo es minimizar el *idleness* medio del grafo y por tanto de cada nodo.

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

Sin embargo, el lago Ypacarai tiene zonas de alta concentración de contaminación, como por ejemplo, las floraciones. Por lo tanto, se crea una matriz de importancia I para ponderar el *idleness* de cada nodo dependiendo de si es una zona muy contaminada o no. Por consiguiente, el caso de patrullaje no homogéneo es el más adecuado para el escenario del lago Ypacaraí [5].

En este caso, el problema de patrullaje se puede reformular en encontrar una política π que minimice la media de W ponderado por una matriz de importancia I dado un número time steps:

$$G(V,E,W) \longrightarrow \pi(E) | \min \frac{1}{M} \sum_{k=1}^{M} W_k \times I \tag{3.2}$$

3.2 Problema del vigilante: caso multiagente

El problema del patrullaje es una tarea naturalmente adecuada para ser compartida en el espacio y en el tiempo por varios agentes. Existe una gran variedad de problemas que pueden reformularse

como una tarea particular de patrulla multiagente, como la cobertura de espacios hídricos. El despliegue de una flota de ASVs tiene varias ventajas frente al uso de un sólo dron:

- Con una buena coordinación, se puede llegar a visitar todo el lago en menos tiempo.
- Varios drones realizarán medidas simultáneamente en puntos diferentes del mapa, esto permitirá tener un mapa de contaminación más completo y detallado, lo que a su vez facilitará la toma de decisiones y la implementación de medidas para abordar y controlar la contaminación en el lago Ypacaraí de manera más eficiente.
- Monitorización más eficiente en el caso de que exista más de un pico de contaminación.

Como se comentó anteriormente, el entorno es representado por un grafo y el objetivo es minimizar el *idleness*. Pero ahora existe un conjunto de agentes **A** que toman acciones basadas en sus percepciones propias sobre el entorno y en su conocimiento sobre el *idleness* y la importancia de los nodos. Existen dos hipótesis sobre el *idleness* en el caso multiagente:

- *Idleness* individual: cada agente considera sólo sus propias visitas para restablecer su *idleness* estimado del nodo.
- *Idleness* compartida: todos los agentes consideran las visitas de todos los agentes para restablecer el *idleness* estimado del nodo. En caso de comunicación instantánea perfecta, el *idleness* compartido corresponde a la real.

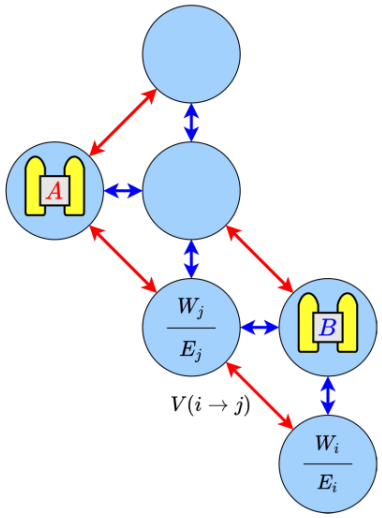


Figura 3.2 Posible representación en grafo del caso multiagente.

Suponiendo que el grupo de agentes es homogéneo, es decir, comparten la misma arquitectura, y existe comunicación sobre el *idleness*, se pueden distinguir dos tipos de estrategias [49]: cognitiva y reactiva. En el primer caso, el grupo de agentes trabaja en conjunto para alcanzar el mismo

objetivo, minimizar globalmente el *idleness* compartido. Mientras que en el segundo caso cada agente toma sus acciones en base a minimizar globalmente su *idleness* individual.

En cuánto a la coordinación de movimientos multiagente, se puede hacer uso de un coordinador central, que elige el nodo objetivo de cada agente (cognitivo) para garantizar que ningún agente se dirija al mismo nodo. O de un coordinador descentralizado en la que la coordinación surge de la interacción. En la práctica, se van a utilizar drones de arquitectura idéntica por lo tanto, nuestro grupo de agentes va a ser homogéneo. Asimismo, seguiremos una estrategia cognitiva con un coordinador central, que será un servidor al que los drones mandarán datos de posiciones y medidas y decidirá las siguientes acciones de los drones. En la Figura 3.2 se muestra una posible representación en grafo del caso multiagente propuesto en [5].

En nuestro trabajo, hemos planteado el problema de patrullaje de manera híbrida, dividiéndolo en dos fases: exploración y explotación. Durante la fase de exploración, los vehículos tienen la tarea de visitar todas las áreas del mapa y tomar medidas de contaminación. En esta etapa, se realiza un patrullaje homogéneo, ya que no se considera la importancia relativa de las zonas y el objetivo principal es lograr una cobertura completa del lago. Una vez que se han identificado las zonas más importantes en términos de contaminación, se pasa a la fase de explotación. En esta etapa, se realiza un patrullaje no homogéneo, centrándose en las áreas más relevantes. Los esfuerzos se intensifican en estas zonas para monitorear y vigilar de cerca su estado de contaminación. Este enfoque plantea un desafío, ya que implica que un mismo agente debe adaptarse a los cambios de fases y tomar decisiones óptimas en cada una de ellas. Cada fase tiene sus propias características y objetivos, lo que requiere que el agente sea capaz de ajustar su comportamiento y acciones de acuerdo con las necesidades específicas de cada fase.

3.3 Descripción del escenario

La definición del entorno juega un papel muy importante en un problema de patrullaje porque define cómo el robot o agente puede moverse. También define las condiciones de contorno y por tanto las zonas transitables o no. En nuestro caso, el escenario está formado por los siguientes elementos:

- **Mapa**: Un grid map es análogo a un grafo, cada celda es equivalente a un nodo, y la distancia recorrida al moverse de una celda a otra adyacente es equivalente al peso de una arista. Se ha discretizado el mapa del lago Ypacaraí en un grid map (Figura 3.3).

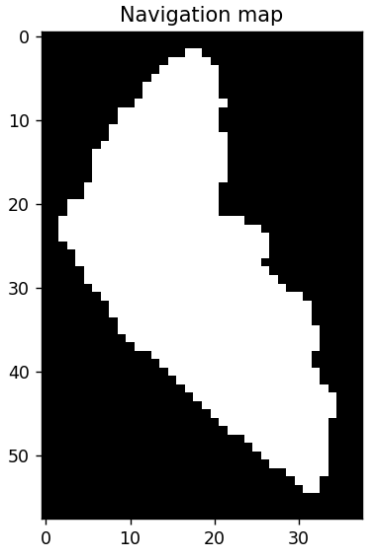


Figura 3.3 Mapa discretizado del lago Ypacaraí.

Teniendo en cuenta que la superficie total del lago es de 60 km2 y tiene una profundidad media de 1,31 m, cada celda cuadrada es equivalente a una fracción de espacio discretizado del lago. En cuanto a las celdas que no se pueden ocupar, que son aquellas que están fuera de la superficie navegable del lago o las celdas que representan el espacio de la tierra, éstas tienen un valor nulo en el grid map. 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).
Cuando el agente explora el lago y recoge muestras, va construyendo un mapa de la contaminación del lago, este también es parte del escenario. En nuestros experimentos sólo hemos tenido en cuenta la contaminación debida a las floraciones de algas verde-azuladas. El comportamiento de la floración es dinámica, bastante caótica y cambia su tamaño con el tiempo. Todo esto se ha tenido en cuenta al crear el modelo del mapa de contaminación en el entorno.

La dinámica del movimiento de las algas verde-azuladas se modela mediante un proceso discreto de difusión. En este modelo, la concentración de las algas se representa mediante partículas discretas que tienen permitido el movimiento dentro del espacio del lago. Partimos de que las partículas se colocan de forma aleatoria sobre el mapa, concentradas en torno a varios puntos iniciales. Cada partícula i posee una posición y una velocidad concreta $(\overline{p}_i(t), \overline{v}_i(t)) \in \mathbb{R}^2$. La velocidad total de las partículas en el modelo de difusión puede descomponerse en una parte estocástica $\overline{v}_i^{aleat}$ que representa la energía de difusión. Esta velocidad de dirección y magnitud aleatoria se evalúa en cada instante de tiempo t y para cada partícula. A la velocidad total se le incorpora una componente derivada de la corriente del agua $\overline{v}_i^{corr}$ proveniente de evaluar la posición de la partícula con un campo vectorial $\hat{V}(x,y)^{corr} : \mathbb{R}^2 \to \mathbb{R}^2$ precalculado con lo que podrían ser las corrientes típicas del lago Ypacaraí. Finalmente, la última componente de la velocidad de las partículas de algas responde a una velocidad constante del viento $\overline{v}^{viento}$, con una magnitud invariante a lo largo de un episodio. De este modo, la velocidad total de una partícula puede expresarse como:

$$\overline{v}_i(t) = \alpha_1 \times \overline{v}_i^{aleat} + \alpha_2 \times \overline{v}_i^{corr} + \alpha_3 \times \overline{v}^{viento} \tag{3.3}$$

En esta ecuación, $\alpha_1, \alpha_2, \alpha_3$ se corresponden con constantes que modulan la predominancia de cada fenómeno en el movimiento total. Para obtener la posición nueva de cada partícula, se computa de forma discreta, dado una constante de tiempo Δt de la siguiente forma:

$$\overline{p}_i(t+1) = \overline{p}_i(t) + \Delta t \times \overline{v}_i(t+1) \tag{3.4}$$

Con la posición de cada partícula, componemos un mapa de partículas P_t que sigue la misma discretización que el *grid map*, en el que cada posición o casilla indica el número de partículas que hay en él. Para pasar del mapa de partículas P_t a un campo escalar de contaminación I_t, aplicamos un filtro gaussiano sobre la imagen de las partículas sobre la cuadrícula, lo que conlleva el paso de un mapa de partículas a un mapa de densidad de partículas al que llamaremos finalmente **mapa de contaminación** (Figura 3.4):

$$I_t = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \otimes P_t \tag{3.5}$$

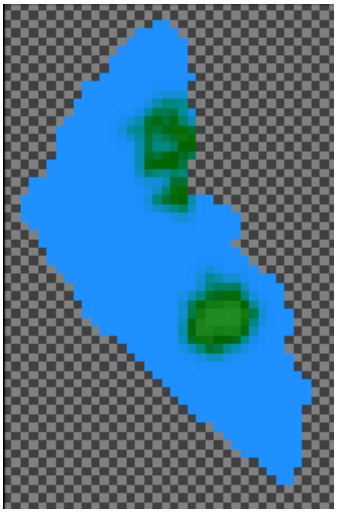


Figura 3.4 Ejemplo del modelado de la contaminación del lago Ypacaraí.

- **Acciones del agente**: Una vez definido el número direcciones que puede tomar el agente, se calculan los ángulos de estas direcciones, espaciadas uniformemente en un intervalo de ángulos $[0,2\pi]$.

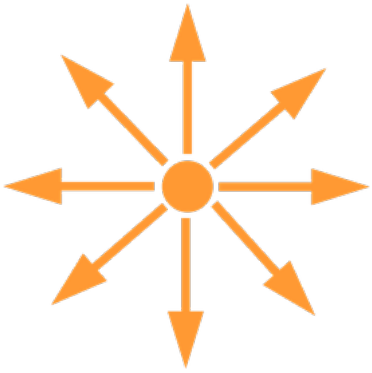


Figura 3.5 Posibles acciones que puede tomar el ASV.

Para lograr una representatividad de las capacidades realistas de movimiento del ASV, suponemos ocho direcciones diferentes (Siguiendo los puntos cardinales: N, E, S, W y NE, SE, NW, SW) en cada paso. Por simplicidad, cada paso de la simulación es un movimiento a una dirección.
Se definen las acciones ilegales como:

  - Movimiento a celda ocupada por otro agente.
  - Movimiento a una celda que forma parte de las zonas no navegables: obstáculo u orilla.

Las acciones ilegales no se realizará ni en la simulación ni la aplicación real, por tanto el agente permanecerá en el mismo lugar.

- **Recompensa**: La recompensa es la forma que tenemos de cuantificar las consecuencias de tomar una acción y puede servir como premio o penalización. La ley de recompensa, que se va a explicar más adelante en la Sección 4.5, penalizará los siguientes casos:
  - Acciones ilegales.
  - El solape de las zonas de detección de dos o más ASVs.
  - Estrategias inadecuadas como visitar zonas con menos *idleness* ponderado respecto a otros lugares.

  En cambio estará diseñada para premiar acciones o estrategias que entendemos como adecuados para la monitorización como:
  - Visitar celdas que nunca se han visitado.
  - No compartición las zonas de detección de dos o más ASVs.
  - Visitar zonas con más contaminación con respecto a otros lugares.
- **Observación**: La observación del entorno permite al agente obtener información sobre el entorno. Hay dos tipos de entornos:
  - Entorno completamente observable: El agente conoce toda la información del entorno: las dimensiones del mapa, su posición, la posición de los demás agentes y la contaminación e *idleness* de cada casilla del lago.
  - Entorno parcialmente observable: Este es nuestro caso, el agente no conoce la contaminación de las celdas que nunca ha visitado y debe mantener un modelo de sensor para estimar la distribución de probabilidad de diferentes observaciones. Sin embargo, en nuestro escenario, el agente solo utiliza la última información conocida sobre la contaminación de una celda.

4 Metodologia

4.1 Reinforcement Learning

El aprendizaje por refuerzo (*RL*), es una técnica de aprendizaje automático en el que un agente aprende a través de la interacción con el entorno. Mediante prueba y error, el agente debe aprender a tomar acciones que maximicen su recompensa, que es la retroalimentación que recibe del entorno. Este enfoque es útil en situaciones en las que no existen datos etiquetados disponibles y el agente debe aprender a partir de la experiencia.

4.1.1 Elementos del Reinforcement Learning

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

Definiciones

- El agente (*agent*) es el programa de software que aprende a tomar las decisiones basadas en las acciones a realizar en el entorno. A medida que toma estas acciones, recibe recompensas que lo ayudan a mejorar su comportamiento en el futuro.
- El entorno (*Environment*) es el mundo en el que el agente se encuentra y donde lleva a cabo sus acciones. El entorno representa todas las posibles situaciones que pueden ocurrir en un momento determinado y responde a las acciones del agente, dándole recompensas positivas o negativas según corresponda. Si el agente conoce el modelo del entorno (*model-based* en inglés), su aprendizaje se basa en el modelo. En cambio, si el agente no conoce el modelo del entorno, su aprendizaje es sin modelo (*model-free* en inglés).
- El *estado* es la descripción del entorno en un momento dado. El espacio de estado es el conjunto de todos los posibles estados, que puede ser infinito. El conjunto de todos los estados posibles se llama espacio de estado y puede llegar a ser infinito [51]. La observación es la parte del estado que el agente tiene acceso. Si el agente es capaz de observar todo el entorno, entonces el entorno es completamente observable. En cambio, si el agente no tiene acceso al estado completo del entorno, éste es parcialmente observable.

- En cada paso, el agente realiza una *acción* en el entorno, lo que hace que el entorno cambie. El conjunto de acciones que el agente puede realizar en un momento dado se llama espacio de acciones. Diferentes entornos permiten diferentes tipos de acciones. El conjunto de acciones disponibles depende del estado actual del entorno, lo que significa que diferentes estados pueden tener diferentes acciones disponibles. El espacio de acciones puede ser discreto, por lo que el agente sólo dispone de un número finito de movimientos. Por otra parte, el espacio de acciones puede ser continuo, como cuando el agente controla un robot en un mundo físico.

- La recompensa (*reward*) es una señal que el entorno proporciona al agente que le indica lo bueno o malo que es el estado actual del mundo debido a sus acciones. Esta señal puede representar una recompensa o una penalización y se utiliza para medir el éxito del agente en su tarea. La función de recompensa es la regla que el entorno utiliza para determinar qué señal de recompensa debe proporcionar. En el aprendizaje por refuerzo, el objetivo del agente es maximizar su recompensa acumulada, que es la suma de las recompensas que recibe en cada paso, llamada retorno.

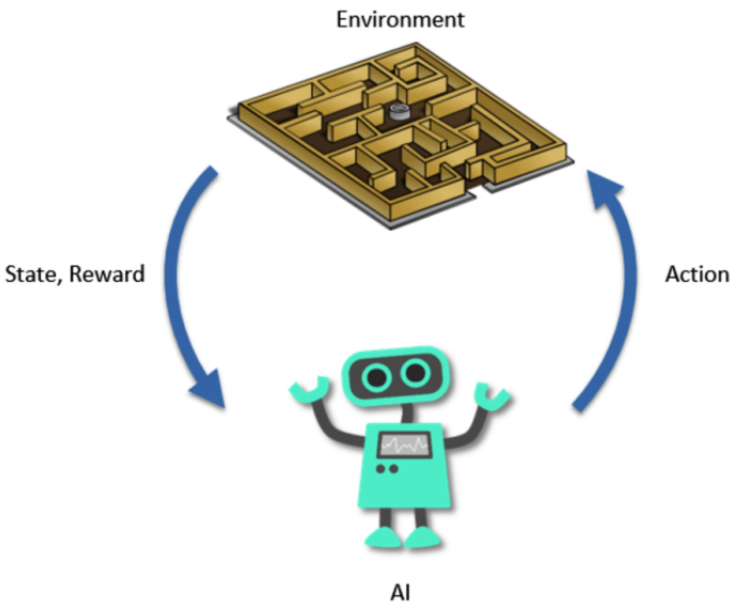


Figura 4.1 Ciclo de aprendizaje por refuerzo.

- La función de transición (*transition function*) es una fórmula matemática que describe cómo el entorno cambia en respuesta a la acción del agente. También indica la probabilidad de moverse de un estado a otro, lo que se llama probabilidades de transición. La función de transición puede ser determinista o estocástica. En un escenario completamente observable es determinista, porque se conoce todo el mapa. En un escenario parcialmente observable es estocástico, porque no se sabe el valor que se encontrará.

- La *experiencia* en el aprendizaje por refuerzo es la secuencia de estados, acciones y recompensas que el agente experimenta mientras interactúa con el entorno. Esta experiencia es fundamental para que el agente pueda aprender a tomar decisiones óptimas en situaciones similares en el futuro. La calidad de la experiencia es importante para el aprendizaje del agente, ya que una experiencia más rica y variada puede ayudar al agente a descubrir patrones y relaciones útiles en los datos.
- El *episodio* es la secuencia completa de interacciones entre el agente y el entorno. Comienza en un estado inicial y termina cuando se alcanza un estado final o se alcanza un límite de tiempo predefinido. El objetivo del agente es maximizar la recompensa acumulada obtenida durante el episodio.

Política

La política es la estrategia que sigue el agente para decidir qué acción elegir dado un estado. La política puede ser determinista o estocástica y puede ser representada por una función que mapea el estado del agente a una acción específica que debe tomar en ese estado. En una política determinista, el agente siempre realiza la misma acción en un estado dado. En una política estocástica, el agente selecciona una acción de acuerdo a una distribución de probabilidad. El objetivo del agente es aprender una política óptima que maximice la cantidad total de recompensas que recibe a lo largo del tiempo. La política óptima es aquella que maximiza la recompensa esperada para cualquier estado dado. En el aprendizaje por refuerzo, el agente utiliza la experiencia para actualizar su política

4.1.2 Proceso de decisión de Markov

El MDP es un modelo matemático que se utiliza para formalizar los problemas de aprendizaje por refuerzo en dominios estocásticos. Asimismo, es la formulación estándar de los elementos del problema de RL.

La propiedad de Markov (en inglés *Markov Property*) establece que los estados futuros dependen solo del estado actual y no de la historia pasada de los estados. En otras palabras, el futuro depende solo del presente y no del pasado. Esta propiedad es esencial para usar el Proceso de decisión de Markov (MDP) en el aprendizaje por refuerzo, ya que permite que el agente aprenda una política que maximiza la cantidad total de recompensas recibidas a lo largo del tiempo. La recompensa que se obtiene al realizar una acción en un estado no depende del camino recorrido para llegar a ese estado.

Un proceso de decisión de Markov es una tupla de 5 elementos (S,A,P,R,γ):

- Un conjunto de estados S. El estado actual se define como s.
- Un conjunto de acciones A. Una acción dentro de ese espacio de acciones se define como a.
- 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 $P^a_{ss\prime} = \mathbb{P}[S_{t+1} = s\prime | S_t = s, A_t = a]$.
- 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 $R^a_{ss\prime} = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$.

- El factor de descuento $\gamma \in (0,1]$ se utiliza para ajustar la importancia de las recompensas a lo largo del tiempo. Representa la importancia relativa de las recompensas futuras en comparación con las recompensas inmediatas.

El objetivo del agente es actuar de tal manera que maximice el rendimiento esperado en una perspectiva a largo plazo con respecto a una función de transición desconocida P. El rendimiento se define como el valor esperado de las recompensas descontadas, es decir, el retorno con descuento G.

$$G_t = R_{t+1} + \gamma * R_{t+2} + \gamma^2 * R_{t+3} + \gamma^3 * R_{t+4} + ... + \gamma^{T-1} * R_{t+T} = \sum_{k=t+1}^{T} G_t = \gamma^{k-t-1} R_k \quad (4.1)$$

La Ecuación 4.1 muestra el retorno descontado G_t en un *time step* t al realizar una acción a_t. Por lo tanto, el agente aprende una política de comportamiento $\pi : S \rightarrow P(A)$ que optimiza el rendimiento esperado J a lo largo del aprendizaje. Este rendimiento se define como el valor esperado de las recompensas descontadas sobre la distribución de estado inicial ρ_0 mientras que las acciones seleccionadas se rigen por la política π [12].

$$J = \mathbb{E}_{s_0 \sim \rho_0, s_{t+1} \sim P, a_t \sim \pi} \left[\sum_{t=0}^{\infty} \gamma^t R_t \right] \quad (4.2)$$

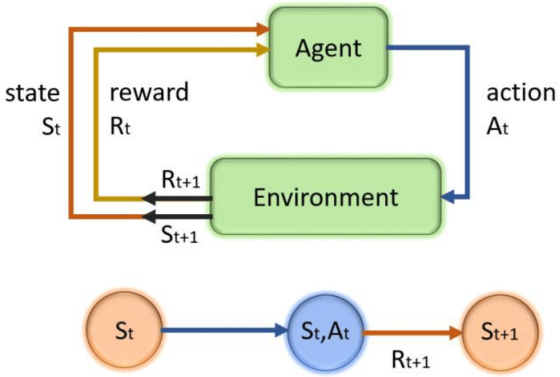


Figura 4.2 Ilustración visual del Proceso de decisión de Markov.

4.1.3 Funciones de valor y la ecuación de Bellman

Una función de valor en RL es una medida utilizada para evaluar la calidad de una acción o estado en términos de la recompensa esperada a largo plazo. Hay dos tipos de funciones de valor en RL: la función de valor del estado y la función de valor estado-acción.

La función de valor del estado, también llamado función V, define la utilidad de estar en un estado particular siguiendo una política determinada $\pi(s)$.Como se muestra en la Ecuación 4.3, la función V es la estimación del retorno descontado esperado que se puede obtener desde un estado dado s siguiendo una política específica π.

$$V_\pi(s) = \mathbb{E}_{s_{t+1}\sim\mathbb{P},a_t\sim\pi}\left[\sum_{j=0}^{T}\gamma^j R_{t+j+1}|S_t = s\right] \tag{4.3}$$

Por otro lado, la función de valor estado-acción, también llamado función Q, describe la utilidad de estar en el estado s, realizar la acción a y seguir la política $\pi(s)$. Esta función estima el valor esperado del retorno descontado que se puede obtener al tomar una acción específica desde un estado dado, y luego seguir una política determinada a partir de ese punto. Sirve para determinar cuál es la mejor acción a tomar en un estado dado.

$$Q_\pi(s,a) = \mathbb{E}_{s_{t+1}\sim\mathbb{P},a_t\sim\pi}\left[\sum_{j=0}^{T}\gamma^j R_{t+j+1}|S_t = s,A_t = a\right] \tag{4.4}$$

Ecuación de Bellman

La ecuación de Bellman es una fórmula matemática que se utiliza en el aprendizaje por refuerzo para actualizar la función de valor de un estado o acción. Para la función V (Ecuación 4.5) y la función de valor estado-acción Q (Ecuación 4.6), la ecuación de Bellman se pueden expresar como una suma de la recompensa inmediata y el valor descontado del siguiente estado.

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

$$Q_\pi(s,a) = \sum_{s\prime} P^a_{ss\prime}[R^a_{ss\prime} + \gamma Q_\pi(s\prime,a\prime)] \tag{4.6}$$

Optimalidad de Bellman

El principio del óptimo de Bellman establece que una política óptima en RL debe cumplir con la propiedad de optimalidad de Bellman, la cual se basa en la ecuación de Bellman (Ecuación 4.5 y Ecuación 4.6). Esto significa que el valor de un estado o acción debe ser igual a la recompensa inmediata más el valor máximo esperado de los estados o acciones sucesores, ponderado por el factor de descuento. Si denotamos $V_*(s)$ como la función de valor de estado óptima, tenemos que:

$$V_*(s) = max_\pi V_\pi(s) \tag{4.7}$$

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

$$V_*(s) = \max_a \sum_a \pi(s,a) * \sum_{s\prime} P^a_{ss\prime}[R^a_{ss\prime} + \gamma V_*(s\prime)] \tag{4.8}$$

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

$$Q_*(s,a) = max_\pi Q_\pi(s,a) \tag{4.9}$$

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

$$Q_*(s,a) = \sum_{s\prime} P^a_{ss\prime}[R^a_{ss\prime} + \gamma \max_{a\prime} Q_*(s\prime,a\prime)] \tag{4.10}$$

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

$$V_*(s) = \max_{a\prime} Q_*(s\prime,a\prime) \tag{4.11}$$

El algoritmo de programación dinámica conocido como iteración de valor o iteración de política se basa en la optimalidad de Bellman para encontrar la política óptima. Este algoritmo utiliza la ecuación de Bellman para iterativamente actualizar los valores de los estados o acciones hasta converger a la solución óptima.

4.1.4 Q-Learning

Los métodos de aprendizaje por diferencia temporal Diferencia Temporal (TD) [52] se basan en el error o la diferencia entre las predicciones sucesivas en el tiempo para actualizar la función Q, en cada *time step*. Esto se debe a que el aprendizaje se produce cuando hay un cambio en la predicción a lo largo del tiempo [52]. El algoritmo Q-Learning es un caso de TD y su objetivo es actualizar los valores de la tabla Q iteративamente. En cada paso, el agente toma una acción $a \in A$ en un estado $s \in S$, recibe la recompensa immediata $r(s,a)$ y actualiza los valores de Q según la siguiente ecuación:

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

Donde $0 < \alpha \geq 1$ es el *learning rate*, que marca la rapidez con la que se actualiza $Q(s,a)$ en cada iteración y $0 \geq \gamma < 1$ es el factor de descuento. Como se puede observar en la Ecuación 4.12, es necesario predecir el retorno en cada *time step*, lo que implica realizar una predicción basada en otra predicción previa, lo cual se conoce como *bootstrapping* en RL

En resumen, el algoritmo calcula el retorno esperado en cada *time step* como la suma de la recompensa inmediata y el valor descontado por el factor gamma del siguiente par estado-acción, utilizando una política greedy ($\max_a Q(s\prime,a)$). El resultado final del algoritmo es una tabla que contiene los valores de estimación para cada estado y cada acción. La política aprendida puede diferir de la política óptima porque se basa en estimaciones del valor Q. Además, el algoritmo Q-Leaning es *offline*, es decir, se puede estimar el valor $Q(s,a)$ utilizando acciones tomadas con una política diferente a la política que se está optimizando. En Algorithm 1 se muestra el algoritmo.

4.1.5 Políticas épsilon-greedy

Cuando un agente está explotando su conocimiento, está utilizando su experiencia previa para seleccionar la acción que le ha dado la mayor recompensa en el pasado. Por otro lado, cuando un agente está explorando, está eligiendo una acción que nunca ha tomado antes y no necesariamente la mejor acción estimada. En resumen, la explotación se basa en la experiencia pasada, mientras que la exploración busca nuevas oportunidades. En el aprendizaje por refuerzo, el agente debe encontrar un equilibrio entre elegir la acción que le ha dado la mayor recompensa en el pasado y

Algorithm 1 Algoritmo Q-Learning

Q ← Tabla vacía de A×S valores;
$end \leftarrow 0$;
while $end == 0$ **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);
    Aplicamos a y adquirimos el nuevo estado del escenario s′ y la recompensa r;
    Actualizamos la tabla Q:
    $Q(s,a) = Q(s,a) + \alpha * (r + \gamma * \max_a Q(s',a) - Q(s,a))$
    $step \leftarrow step + 1$;
    if ΔQ<Umbral **then**
      $end \leftarrow 1$;
    end if
  end while
end while

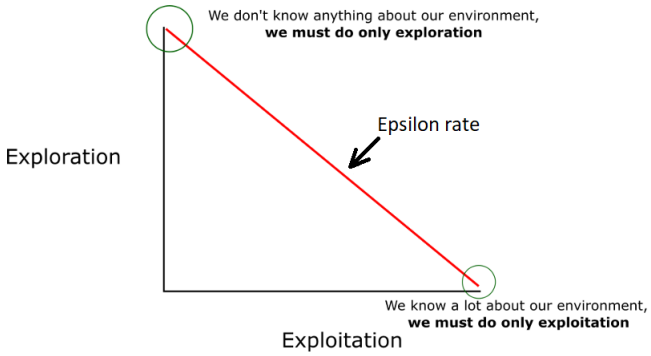


Figura 4.3 Evolución y etapas de los valores de ε.

explorar nuevas acciones que podrían darle una recompensa aún mayor. Este equilibrio se conoce como el dilema exploración-explotación. Si el agente solo se enfoca en la explotación, podría perder oportunidades para obtener una recompensa aún mayor. Por otro lado, si solo se enfoca en la exploración, podría perder la oportunidad de aprovechar su experiencia pasada. Por lo tanto, el objetivo es encontrar un equilibrio para aprovechar lo mejor de ambos mundos y maximizar la recompensa a largo plazo.

La acción que tiene la mayor recompensa se conoce como acción "greedy". Si un agente siempre elige esta acción, se dice que está siguiendo una política "greedy". Por otro lado, en una política "ε-greedy", el agente elige una acción no "greedy" al azar con una probabilidad ε, mientras que elige la acción "greedy" que ha sido mejor en el pasado con una probabilidad $(1-\varepsilon)$. Esto

permite al agente equilibrar la exploración y la explotación para encontrar el equilibrio adecuado entre elegir la mejor acción y explorar nuevas acciones. Una política ε-greedy puede ayudar a equilibrar la exploración y la explotación. Es importante reducir gradualmente el valor de ε a medida que el agente adquiere más conocimiento para que pueda centrarse en las acciones que maximizan la recompensa. Al principio, es recomendable que el agente explore diferentes acciones para encontrar posibles mejoras. Si bien el agente puede preferir la explotación al final del entrenamiento, si se mantiene un valor bajo de ε, siempre habrá un margen de mejora, aunque pequeño. En la Figura 4.3 se muestra un ejemplo comentado de como evolución ε a lo largo de un episodio.

4.2 Multi-agent Reinforcement Learning

Cuando varios agentes interactúan simultáneamente en un entorno y posiblemente entre ellos, el MDP se extiende a los *Markov Games* [53].

Un *Markov Game* es una extensión de los MDPs y esa formalizado por la tupla de seis elementos $(\mathbb{N},\mathbb{S},\{\mathbb{A}^i\},P,\{R^i\},\gamma)$:

- $\mathbb{N} = \{1,2,...,N\}$ N denota el conjunto de $N > 1$ número de agentes
- Un conjunto de estados $\mathbb{S}$ observado por todos los agentes. El estado actual se define como s.
- El espacio de acción conjunto se denota por $\mathbb{A} = A_1 \times , ... , \times A_N$ que es la colección de espacios de acción individuales de los agentes $i \in N$.
- Una Función de transición $P : \mathbb{S} \times \mathbb{A} \rightarrow P(\mathbb{S})$.
- Cada agente posee una función de recompensa asociada $R^i : \mathbb{S} \times \mathbb{A} \times \mathbb{S} \rightarrow \mathbb{R}$.
- El factor de descuento $\gamma \in (0,1]$ se utiliza para ajustar la importancia de las recompensas a lo largo del tiempo.

El objetivo es que cada agente busca optimizar su política individual $\pi^i : S \times A_i$, es decir, maximizar su recompensa acumulada esperada a lo largo del tiempo. Esto implica aprender una política teniendo en cuenta las acciones de otros agentes y su impacto en el entorno. En cada paso, cada agente $i \in N$ selecciona y ejecuta una acción en función de la política individual. El sistema evoluciona desde el estado s_t bajo la acción conjunta a_t con respecto a la función de probabilidad de transición P al siguiente estado s_{t+1}. Cada agente recibe R^i como respuesta inmediata a la transición de estado. Se define π como la estrategia conjunta de todos los agentes y π^{-i}como la estrategia conjunta de todos los agentes excepto el agente i. Utilizaremos la notación $< \pi^i, \pi^{-i} >$ para referirnos a la estrategia conjunta en la que el agente i sigue la política π mientras que los demás agentes siguen su política π^{-i}.

En cuanto a la función de valor de estado de cada agente i, ésta depende ahora de las políticas de los demás agentes:

$$V^i_{\pi^i,\pi^{-i}}(s) = \mathbb{E}_{s_{t+1}\sim\mathbb{P},a_t\sim\pi}\left[\sum_{j=0}^{T}\gamma^j R_{t+j+1}|S_t = s\right] \tag{4.13}$$

Por consiguiente, la política óptima viene determinada por la política individual y las estrategias de los demás agentes. La política que aprende el agente depende de varios factores. En [12] se presentan los dos principales factores: tipo de tarea y la información disponible. Atendiendo al tipo de tarea:

- Entorno completamente cooperativo: Los agentes están motivados a cooperar, maximizar el rendimiento del grupo e intentar evitar el fallo de un individuo. En este entorno todos los agentes reciben la misma recompensa ($R = R^i = ... = R^N$) en la transición de estados. Si los agentes son motivados a cooperar pero no reciben la misma recompensa, se dice que están en un entorno cooperativo.
- Entorno completamente competitivo: Los agentes intentan maximizar el rendimiento individual mientras minimizan el de los demás. Este entorno, formalmente denominado *zero-sum* Markov Game, la suma de las recompensas de los agentes en la transición de estados es 0 ($R = \sum_{i=1}^{N} = R^i$). Si los agentes son motivados a competir pero la suma de las recompensas no es 0, se dice que están en un entorno competitivo.
- Entorno mixto: El entorno no es ni completamente competitivo ni completamente cooperativo y por tanto no hay restricciones en los objetivos de los agentes.

Atendiendo a la información disponible:

- Agentes independientes: Los agentes no conocen las acciones ni las recompensas de los demás agentes y por tanto estos forman parte del entorno.
- Agentes de acción conjunta: Los agentes observan las acciones tomadas de todos los demás a-posteriori.

4.2.1 Definiciones

Cuando las políticas de los demás agentes están determinadas, el agente i puede maximizar su propia función de valor del estado encontrando la mejor respuesta.

Definition 4.2.1 (Mejor respuesta)**.** *La mejor respuesta de un agente i* $\pi^i_* \in \Pi^i$ *con respecto a las estrategias de los demás agentes* π^{-i} *en todos los estados* $s \in \mathbb{S}$*.*

$$V^i_{\pi^i_*,\pi^{-i}}(s) \geq V^i_{\pi^i,\pi^{-i}}(s) \tag{4.14}$$

Sin embargo, cuando los agentes aprenden simultáneamente, la mejor respuesta encontrada puede no ser única [12]. Desde el punto de vista de la teoría de juegos, se utilizan dos conceptos comunes de equilibrio para definir soluciones en los juegos. El primero se denomina equilibrio de Nash:

Definition 4.2.2 (Equilibrio de Nash)**.** *La solución en la que la política $\pi^i_* \in \Pi^i$ es la mejor respuesta de cada agente i con respecto a las estrategias de los demás agentes π^{-i}_* tal que la siguiente inecuación*

$$V^i_{\pi^i_*,\pi^{-i}_*}(s) \geq V^i_{\pi^i,\pi^{-i}_*}(s) \tag{4.15}$$

se cumple en todos los estados $s \in \mathbb{S}$ y en todas las políticas $\pi^i \in \Pi^i$, $\forall i$.

Por lo tanto, ningún agente i puede aumentar su recompensa esperada si se desvía unilateralmente de la política π^i_* que cumple un equilibrio de Nash. Sin embargo, pueden haber múltiples equilibrios de Nash y no está garantizado que sea siempre la mejor solución grupal [54]. Por ello el concepto de Pareto-optimalidad podría ser útil.

Definition 4.2.3 (Pareto-dominancia)**.** *Una política conjunta π domina a otra política conjunta $\hat{\pi}$ si y sólo si:*

$$V^i_{\pi}(s) \geq V^i_{\hat{\pi}}(s) \;\; \forall i, \forall s \in \mathbb{S} \quad y \quad V^j_{\pi}(s) > V^j_{\hat{\pi}}(s) \;\; \exists j, \exists s \in \mathbb{S} \tag{4.16}$$

Definition 4.2.4 (Pareto-optimalidad)**.** *Si una política conjunta π_* no es Pareto-dominada por ninguna otra política conjunta, entonces π_* es Pareto-óptima.*

Por lo tanto, una solución que cumple el equilibrio de Nash es pareto-óptimo si no se puede mejorar la recompensa esperada de ningún agente sin que disminuya la recompensa esperada de ningún otro agente.

4.2.2 Multi-agent Q-Learning

En un sistema multiagente, cada agente toma una acción individualmente en función de su propia política (representada por π^i), y estas acciones se combinan en un vector $a = [a_1,a_2,..,a_N] \;\; a_i \in A_i$ que representa las acciones conjuntas de todos los agentes en un paso. Del mismo modo, el estado del sistema se representa mediante un vector $s = [s_1,s_2,..,s_N] \;\; s_i \in S_i$ que contiene los estados individuales de cada agente. En cada paso, cada agente toma una acción $a_i : \pi^i, \;\; \pi^i \in \pi$, recibe una recompensa $R^i(s_i,a)$ que depende del estado del agente s_i y las acciones conjuntas a de todos los agentes. Sin embargo, a pesar de que la recompensa recibida depende de todas las acciones conjuntas, cada agente actualiza su propia tabla $Q_i(s_i,a_i)$, donde Q_i es una función de valor que asigna un valor a cada par estado-acción individual del agente i. Esto implica que varias acciones conjuntas se asocian a una única percepción de la acción del agente, ya que cada agente solo actualiza su propia tabla basándose en su acción individual y la recompensa recibida

Es por ello que resulta útil utilizar una función $Q_i(s_i,a)$ que mantuviera un valor Q separado para cada acción conjunta [55]. La función de valores estado-acción conjuntos $Q_i(s_i,a)$ de un agente se define como el promedio (sobre todos los estados conjuntos que contienen s_i) de la recompensa esperada recibida por el agente i cuando la acción conjunta a se ejecuta en s_i siguiendo una política $\pi^*_i : S_i \rightarrow A$ (política conjunta que proporciona la máxima recompensa con descuento para el agente i sobre todos los s_i). La relación entre los valores Q conjuntos y los valores Q reales del agente $Q_i(s_i,a_i)$ se describe mediante:

$$Q_i(s_i,a_i) = \sum_a p(a|a_i) Q_i(s_i,a) \tag{4.17}$$

donde $p(a|a_i)$ es la probabilidad de que el conjunto de acciones a sea ejecutada cuando el agente i toma la acción a_i. Los valores Q conjuntos óptimos del agente se definen como:

$$Q_i^*(s_i,a) = R^i(s_i,a) + \sum_t \sum_{s_i(t)} \gamma^t * p(s_i(t)|s_i(t-1),\, a(t-1)) * R^i(s_i(t),\pi_i^*(s(t)) \tag{4.18}$$

donde $t = \{0,1,...,\infty\}$ y $p(s_i(t)|s_i(t-1),\, a(t-1))$ es la probabilidad de transición al estado $s(t)$ dado la acción y el estado anterior.

En un sistema cooperativo, una solución se considera óptima si cumple dos condiciones: es un equilibrio de Nash y también es Pareto-óptimo [55]. Sin embargo, en un entorno cooperativo, podemos restringir aún más esta definición de optimalidad a un subconjunto de equilibrios de Nash llamados equilibrios de coordinación. Un equilibrio de coordinación se caracteriza por tener una propiedad adicional: si existe una acción conjunta en la cual todos los agentes reciben su recompensa máxima, y solo hay una acción conjunta que cumple esta condición, entonces el equilibrio de coordinación se considera estricto. En otras palabras, en un equilibrio de coordinación estricto, solo hay una acción conjunta que maximiza las recompensas de todos los agentes, y no hay otras acciones conjuntas que tengan el mismo efecto. En los casos en los que existen múltiples acciones conjuntas que permiten que todos los agentes obtengan su recompensa máxima, se requiere alguna forma de selección coordinada para determinar cuál de estas acciones se considera el equilibrio de coordinación. Esto implica que en un equilibrio de coordinación no estricto, es necesaria alguna forma de acuerdo o coordinación entre los agentes para seleccionar una de las acciones conjuntas óptimas.

4.3 Redes neuronales artificiales

las RNAs son herramientas que permiten a los programas informáticos emular el procesamiento de información realizado por las redes de neuronas biológicas, lo cual resulta útil en la resolución de problemas y el reconocimiento de patrones. Estas redes están compuestas por neuronas artificiales interconectadas en gran cantidad y funcionan en paralelo. Las RNAs se comportan como un modelo de caja negra y son funciones no lineales que aplican a un vector de entradas un vector de salida.

4.3.1 Descripción y funcionamiento

Hay muchos tipos de redes neuronales pero el más usado es el tipo perceptrón. Esta es una red acíclica de nodos, densamente conectada y organizada en capas. En [56] se demostró que el perceptrón multicapa es capaz de aproximar universalmente funciones, lo que significa que puede aprender de ejemplos y aproximar relaciones no lineales. Esto hace que sea adecuado para abordar problemas del mundo real. La arquitectura del perceptrón multicapa consta de tres tipos de capas. Primero, está la capa de entrada, que recibe los datos externos y los transmite a todas las neuronas de la siguiente capa. Luego, están las capas ocultas, las cuales procesan los datos de manera no

lineal provenientes de la capa de entrada. Estas capas ocultas permiten al perceptrón realizar un procesamiento complejo y capturar características importantes de los datos. Por último, está la capa de salida, que recibe información de las capas anteriores y proporciona la respuesta o decisión final de la red hacia el exterior.

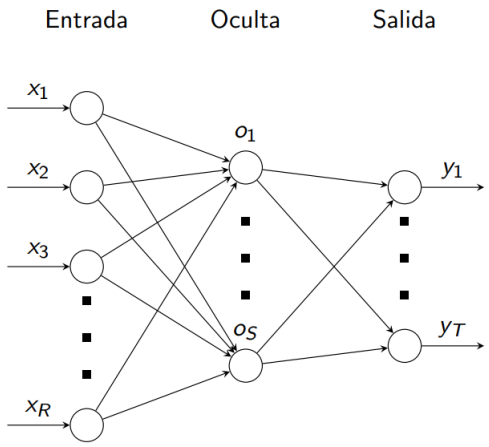


Figura 4.4 Perceptrón Multicapa.

En una red neuronal, cada unidad individual se denomina neurona y se representa como un nodo en el grafo de la red. La Figura 4.5 muestra la estructura de una neurona artificial y también se destacan las similitudes con las neuronas biológicas en la parte superior de la figura.

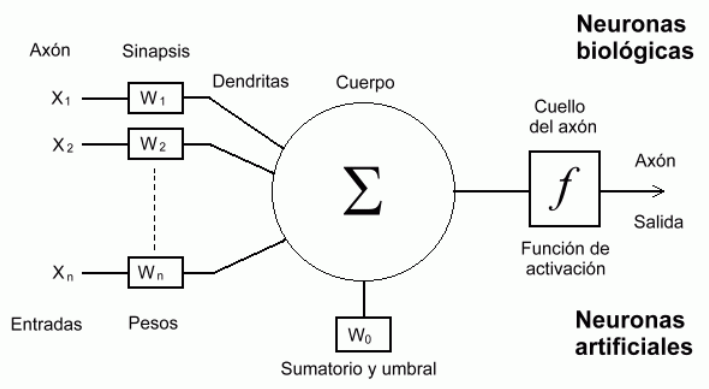


Figura 4.5 Modelo de una neurona artificial.

Cada nodo realiza una suma ponderada de sus entradas y luego son procesadas por una función de activación, denotada como f. Esta función determina la señal de salida de la neurona, denotada como y. La función de activación permite que las neuronas cambien su nivel de activación en base a las señales que reciben. El nivel de activación, que representa el estado interno de la neurona, depende de las entradas recibidas y de los pesos sinápticos, pero no de los valores anteriores de

activación. La ecuación general para el cálculo de la salida de una neurona artificial:

$$y = f\left(\sum_{i=1}^{n} w_i \cdot x_i + b\right) \tag{4.19}$$

Donde:

- x_i son las entradas a la neurona artificial.
- y es la salida de la neurona artificial.
- $f(\cdot)$ es la función de activación.
- w_i son los pesos sinápticos asociados a las entradas x_i.
- b es el sesgo (bias) de la neurona.
- n es el número de entradas a la neurona.

La función de activación es un componente fundamental de una neurona, ya que define su comportamiento, las más usadas son:

1. Función identidad (identity): La neurona artificial propaga el mismo valor aplicar ninguna transformación. Es muy usado en problemas de regresión en la capa de salida.

$$f(x) = x \tag{4.20}$$

2. Función escalón (threshold): también conocida como función de Heaviside, 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.

$$f(x) = \begin{cases} 0 & si \quad x < 0 \\ \\ 1 & si \quad x \geq 0 \end{cases} \tag{4.21}$$

3. Función sigmoide (sigm): Función continua que devuelve un valor de salida en el rango de 0 a 1 para cualquier valor de entrada. Es muy utilizada en problemas de clasificación binaria o problemas en los que se requiere una salida en forma de probabilidad.

$$f(x) = \frac{1}{1 + e^{-}x} \tag{4.22}$$

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 retorna el valor 0 para cualquier valor negativo de entrada, mientras que retorna el valor de entrada si es positivo. Es una función no lineal muy utilizada en redes neuronales debido a

su eficiencia computacional y su capacidad para resolver problemas de aprendizaje profundo. Esta función, a diferencia de las dos anteriores, no está acotada.

$$f(x) = max(0,x) \tag{4.23}$$

La función de activación Leaky ReLU es una variante de la función ReLU (Rectified Linear Unit) que introduce una pequeña pendiente positiva para los valores negativos en lugar de tener una pendiente plana. Esta función es especialmente útil en tareas donde se pueden presentar gradientes dispersos o problemas de desvanecimiento de gradientes. En lugar de hacer que los valores negativos sean directamente cero, como lo hace la función ReLU, la función Leaky ReLU permite que una fracción pequeña de la entrada negativa pase a través de la función. Esto ayuda a prevenir la saturación de la neurona y permite un flujo más suave de los gradientes durante el proceso de retropropagación en el entrenamiento de una red neuronal.

5. Función tangente hiperbólica (tanh). Esta función es muy similar a la sigmoidal pero produce una salida en el rango de -1 a 1.

$$f(x) = \frac{1 - e^{-}2x}{1 + e^{-}2x} \tag{4.24}$$

4.3.2 Redes Neuronales Convolucionales

Una red neuronal convolucional (en inglés, Convolutional Neural Network (CNN)), es un tipo de red que se utiliza principalmente para procesar datos de dos dimensiones, como imágenes. Está diseñada para extraer características relevantes de las imágenes mediante la aplicación de la operación de convolución. En la actualidad se utilizan ampliamente en diversas aplicaciones, como reconocimiento de objetos, segmentación semántica, superresolución de imágenes. Las CNN son normalmente compuestas por varias capas:

- **Capas de convolución**: la capa principal y fundamental es la capa de convolución. Esta capa es esencial en las CNNs y se utiliza una o varias capas de este tipo en estas redes. La operación central realizada por esta capa es la convolución. La convolución es una operación matemática que combina los valores de píxeles de una imagen deslizando un filtro o núcleo (una matriz de pesos) sobre la entrada y se realiza una combinación lineal de los valores del filtro con los valores correspondientes de la región de la entrada cubierta por el filtro. El resultado de esta combinación lineal se coloca en una nueva matriz llamada mapa de características. Este mapa de características resalta patrones locales en la imagen, como bordes, texturas o formas, que son importantes para tareas como el reconocimiento de objetos o la clasificación de imágenes.

$$x * K = y[i,j] = \sum_{m=0}^{M} [\sum_{n=0}^{N} x(m,n) \times w(i-m, j-n)] \tag{4.25}$$

En la ecuación 4.25, se muestra la operación de convolución en una red neuronal convolucional. En esta ecuación, **x** representa la imagen de entrada con dimensiones $N \times M$ píxeles,

w es el filtro o kernel que se aplica en la convolución, y **y** es el resultado de la convolución, también conocido como mapa de características. El objetivo de la red neuronal convolucional es aprender los valores de los pesos del filtro **w** durante el proceso de entrenamiento. Estos pesos representan los filtros que se activan en presencia de características específicas en regiones particulares de la imagen de entrada. A medida que la red se entrena con un conjunto de datos, ajusta los valores de los pesos para identificar y resaltar las características relevantes en la imagen.

- **Capa de activación**: Después de la capa de convolución en una red neuronal convolucional, se aplica una función de activación, como ReLU (Rectified Linear Unit), a la salida de la convolución. Esta etapa es importante para introducir no linealidades en la red y permitir a la red neuronal convolucional aprender y modelar relaciones no lineales en los datos de entrada.
- **Capa de agrupación**: se suelen colocar capas de agrupación (también conocidas como capas de pooling) entre dos capas convolucionales. Estas capas tienen como objetivo reducir la información espacial y prevenir el sobreajuste en la red, al mismo tiempo que proporcionan cierta invarianza a la traslación. La operación de agrupación se realiza de manera similar a la convolución, pero en lugar de utilizar filtros ponderados, se aplica una operación específica en pequeñas regiones de la imagen de entrada. Una técnica comúnmente utilizada es el *Max-pooling*, en la cual se selecciona el valor máximo dentro de cada subregión no superpuesta de la matriz inicial. El *Max-pooling* reduce la dimensionalidad de la imagen de entrada al tomar solo el valor máximo de cada subregión, descartando así información menos relevante y conservando características importantes. Esto ayuda a mantener la invarianza a pequeñas traslaciones y a reducir el costo computacional de la red.
- **Capa densamente conectada**: En las redes neuronales convolucionales (CNN), es común utilizar una combinación de arquitecturas al pasar los mapas de características a una red neuronal densa. Esto implica reducir la dimensionalidad de los datos de 2D a 1D. A esta combinación de arquitecturas se le conoce como Red Neuronal Convolucional Densa. Una vez que se obtienen los mapas de características a través de las capas convolucionales, se utilizan como entrada en una red neuronal densa. En la red neuronal densa, todas las neuronas de la capa anterior están conectadas a todas las neuronas de la siguiente capa. Esto permite la combinación de características de alto nivel extraídas por las capas convolucionales.

4.3.3 Entrenamiento

El entrenamiento de una red neuronal consiste en ajustar los parámetros (pesos y sesgos) de la red para que la salida se acerque lo más posible al resultado deseado. Por razones matemáticas es un proceso iterativo. Inicialmente, los parámetros se establecen de forma aleatoria y, a medida que la red se expone a los datos de entrenamiento, se actualizan para reducir la diferencia entre la predicción y el objetivo. Esta diferencia se cuantifica utilizando una función de pérdida.

Las iteraciones del proceso de entrenamiento son un recorrido sobre la superficie de error y el objetivo del entrenamiento es minimizar la función de pérdida. Para lograr esto, se utiliza un algoritmo de optimización capaz de resolver un problema de minimización no lineal. El algoritmo más comúnmente utilizado es el descenso de gradiente. En el descenso de gradiente, se calcula la

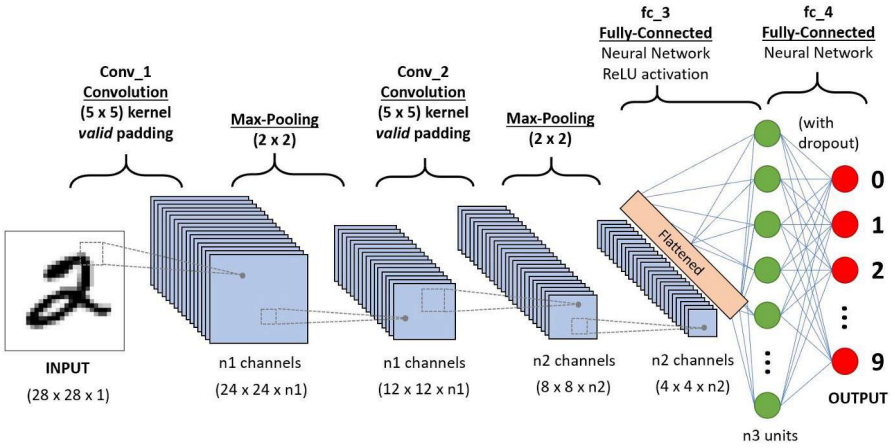


Figura 4.6 Ejemplo de una Red Neuronal Convolucional Densa con Operación de Maxpool 2X2.

dirección y la magnitud del cambio necesario para reducir el error de la red neuronal. Esto se logra mediante el cálculo del gradiente de la función de pérdida, que representa cómo varía la función de pérdida con respecto a cada uno de los parámetros de la red. A continuación, se ajustan los parámetros de la red en la dirección opuesta al gradiente, lo que significa que se actualizan de manera que el error disminuya. El proceso se repite iterativamente, calculando el gradiente y actualizando los parámetros en cada iteración. A medida que avanza el descenso de gradiente, los parámetros se ajustan en pequeños pasos, lo que permite que la red se acerque gradualmente al mínimo de la función de pérdida. Esto lleva a una reducción progresiva del error y una mejora en el rendimiento de la red neuronal. Este algoritmo utiliza la propagación hacia atrás, también conocida como backpropagation. Consiste en calcular el error de la capa final y propagarlo hacia atrás a través de las capas anteriores, utilizando la regla de la cadena para calcular la contribución de cada parámetro en el error. La retropropagación se basa en la idea de que el error de una capa anterior depende de la capa posterior. En conclusión, el aprendizaje de una red neuronal implica ajustar los parámetros de la red utilizando el algoritmo de descenso de gradiente. Este algoritmo utiliza la retropropagación para calcular el error de cada capa y actualizar los parámetros en la dirección que reduce el error. De esta manera, la red neuronal puede aprender a aproximar las salidas deseadas a partir de los datos de entrenamiento.

4.4 Deep Q-Learning

El algoritmo Q-Learning puede sufrir la maldición de la dimensionalidad en dominios donde los espacios de estados son de alta dimensión [57]. Esto se debe a que la construcción y mantenimiento de una tabla de valores para cada par estado-acción puede volverse computacionalmente

inmanejable. Para abordar este desafío, es común utilizar un aproximador no lineal de funciones, como una red neuronal, para estimar la función estado-acción. En lugar de almacenar los valores Q en una tabla, la red neuronal puede aprender a mapear directamente los estados a sus respectivas acciones y estimar los valores Q correspondientes,

$$Q(s,a) \approx Q(s,a;\theta) \tag{4.26}$$

donde el conjunto de parámetros θ son los pesos de la red neuronal.

El uso de una red neuronal como aproximador de funciones permite gestionar eficientemente espacios de estados de alta dimensión, ya que la red puede generalizar y aprender patrones complejos en los datos de entrada. La red neuronal toma el estado actual como entrada y produce una salida que representa los valores Q para cada acción posible en ese estado. Al utilizar una red neuronal en lugar de una tabla de valores, el algoritmo Q-Learning puede ser aplicado en problemas más complejos y con mayores dimensiones de estado, lo que amplía su aplicabilidad en una variedad de dominios de aprendizaje por refuerzo. La ecuación de actualización en este caso se representa mediante la ecuación 4.27.

$$Q(s,a;\theta) = Q(s,a;\theta) + \alpha * (Q_{target} - Q(s,a;\theta)) \tag{4.27}$$

En esta ecuación, $Q(s,a;\theta)$ representa la función de valor Q para un par estado-acción dado, parametrizada por los pesos de la red neuronal θ. La actualización se realiza ajustando los pesos θ en la dirección del gradiente descendente de la función de pérdida, que es la diferencia entre el valor objetivo Q_{target} y el valor actual $Q(s,a;\theta)$ multiplicada por el *learning rate* α. El valor objetivo Q_{target} se calcula utilizando la ecuación 4.28. Donde r representa la recompensa obtenida al tomar la acción a en el estado s, γ es el factor de descuento que pondera la importancia de las recompensas futuras, y $\max_{a\prime} Q(s\prime,a\prime;\theta)$ es el máximo valor de Q para el siguiente estado $s\prime$ considerando todas las posibles acciones $a\prime$:

$$Q_{target} = r + \gamma * \max_{a\prime} Q(s\prime,a\prime;\theta) \tag{4.28}$$

La red neuronal se entrena minimizando una función de pérdida $L(\theta)$ en cada iteración, como se muestra en la ecuación 4.29. Esta función de pérdida representa la discrepancia entre el valor objetivo Q_{target} y el valor actual $Q(s,a;\theta)$, y se utiliza para actualizar los pesos θ de la red mediante técnicas de optimización como el descenso de gradiente:

$$L(\theta) = L(Q_{target},Q(s,a;\theta)) \tag{4.29}$$

Cuando se utiliza una red neuronal para representar la función Q en el aprendizaje por refuerzo, pueden surgir problemas de inestabilidad o divergencia de los pesos de la red. Esto puede ocurrir debido a varias razones. Una de las razones es la correlación existente entre la acción tomada y el estado actual, así como la secuencia de observaciones en un entorno complejo. Esto significa que las actualizaciones de los pesos de la red pueden basarse en datos altamente correlacionados, lo que puede dificultar la convergencia y provocar inestabilidad. Además, pequeñas actualizaciones en la función Q pueden tener un impacto significativo en la política de toma de decisiones, lo que a su vez puede cambiar la distribución de los datos utilizados para entrenar la red. Esto puede

generar cambios abruptos en los valores estimados por la red y dificultar la convergencia hacia una solución estable. En el artículo mencionado [58], se utiliza una red neuronal convolucional como aproximador en un algoritmo de aprendizaje por refuerzo llamado DQN (Deep Q-Network) para lograr un rendimiento comparable al humano en los juegos de Atari 2600. Este trabajo fue considerado un avance significativo en el campo, ya que abordaron las inestabilidades y desafíos asociados con el uso de redes neuronales en el aprendizaje por refuerzo. Para superar las dificultades mencionadas anteriormente, los autores propusieron dos ideas clave: el *experience replay* y la *target network*.

4.4.1 Experience Replay

En el contexto del aprendizaje continuo, puede ocurrir un fenómeno llamado olvido catastrófico. Esto sucede cuando un modelo de aprendizaje, como una red neuronal, olvida completamente el conocimiento adquirido previamente al aprender nuevas tareas o ejemplos. En otras palabras, el modelo pierde la capacidad de generalizar adecuadamente en una tarea después de ser entrenado en una nueva tarea. Este problema de interferencia catastrófica puede ser problemático en escenarios de aprendizaje continuo, donde se requiere que el agente se adapte y aprenda nuevas tareas sin olvidar completamente el conocimiento previo. Si el modelo reemplaza completamente lo que ha aprendido anteriormente con nuevos datos, no podrá utilizar ese conocimiento previo para mejorar su desempeño en nuevas situaciones. Para abordar este problema, es importante utilizar técnicas como el Experience Replay. Consiste en almacenar las experiencias pasadas del agente en un búfer de memoria, formadas por tuplas que contienen el estado actual (s), la acción tomada (a), la recompensa obtenida (r), y el estado siguiente $(s\prime)$. El *Experience Replay* también se utiliza para abordar el problema de la correlación entre observaciones sucesivas en el aprendizaje por refuerzo. En lugar de actualizar los parámetros de la red neuronal inmediatamente después de cada interacción con el entorno, se seleccionan al azar varias experiencias del búfer de memoria, incluyendo algunas más recientes. Estas experiencias seleccionadas se utilizan para actualizar los parámetros de la red, lo que permite que el agente aprenda de una variedad de situaciones pasadas.

El tamaño del búfer de memoria es fijo, y a medida que el agente interactúa con el entorno, se van guardando las experiencias más recientes y se eliminan las más antiguas para mantener el tamaño constante. El deque (doble-ended queue) es una estructura de datos proporcionada por la librería *collections* de *Python* que utilizaremos para almacenar las experiencias del agente en el *Experience Replay*. Se comporta como una lista con un tamaño máximo predefinido. Cuando el deque está lleno y se intenta añadir un nuevo elemento, se desplaza automáticamente su contenido hacia la izquierda, eliminando el primer elemento y haciendo espacio para el nuevo elemento al final de la lista. Esta funcionalidad permite mantener un historial limitado de experiencias pasadas en el deque, asegurando que siempre tengamos las últimas y más relevantes experiencias almacenadas. Además, al ser un deque, también podemos acceder eficientemente tanto al principio como al final de la lista.

4.4.2 Target Network

Si el agente encuentra estados muy similares en secuencia pero con recompensas distintas, el proceso de actualización de los parámetros de la red puede volverse inestable. Esto se debe a que la

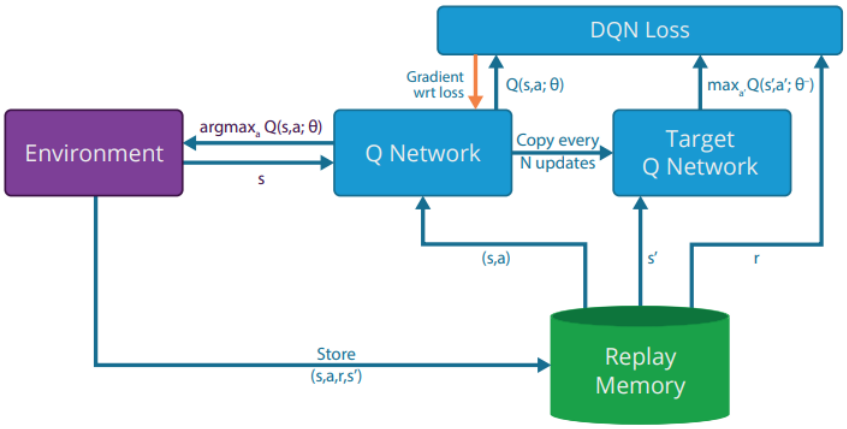


Figura 4.7 Esquema del algoritmo DQN.

red tiene dificultades para distinguir entre estos estados similares y puede producir actualizaciones erráticas de los pesos en cada paso temporal. Para lograr esto, crearon una copia de los parámetros de la red neuronal al inicio del entrenamiento, denotada como θ^-. Estos parámetros se actualizan solo cada C pasos y se mantienen fijos entre las actualizaciones individuales, mientras que los parámetros de la red principal Q, denotados como θ, se actualizan regularmente. Este retraso en la actualización permite disminuir el impacto de las actualizaciones recientes en la selección de acciones, lo que conduce a una mayor estabilidad en el aprendizaje.

Los parámetros θ^- serán usados para calcular Q_{target}, que ahora es otra red neuronal. Cabe destacar que nunca se retropropaga en la target network, sólo en la principal.

$$Q_{target}(s,a) \approx Q_{target}(s,a;\theta^-) \tag{4.30}$$

La función de pérdida quedaría como sigue:

$$L_i(\theta_i) = \mathop{\mathbb{E}}_{(s,a,r,s\prime)\backsim U(D)}[(r+\gamma * \max_{a\prime} Q_{target}(s\prime,a\prime;\theta_i^-) - Q(s,a;\theta_i))^2] \tag{4.31}$$

En el algoritmo Q-Learning existe un problema conocido como sobreestimación de los valores de acción. Esto ocurre debido a que se utiliza una operación de maximización sobre los valores de acción estimados, lo que tiende a favorecer los valores sobreestimados en lugar de los subestimados. Esto puede llevar a que el agente aprenda valores de acción poco realistas. Los autores de [7] demuestran que incluso el algoritmo DQN, que utiliza una red neuronal profunda para estimar los valores de acción, puede sufrir de sobreestimación sustancial de los valores. La solución que propusieron, fue utilizar la red neuronal principal para seleccionar la acción óptima en el estado siguiente y luego utiliza la target network para estimar el valor correspondiente a esa acción, esta variante la denominaron Double-DQN. La fórmula de Double-DQN se puede expresar de la siguiente manera:

$$Q(s,a;\theta) = Q(s,a;\theta) + \alpha \left(r + \gamma * Q \left(s\prime, \arg\max_{a\prime} Q(s\prime,a\prime;\theta); \theta^{-} \right) - Q(s,a;\theta) \right) \quad (4.32)$$

El algoritmo Double-DQN se usará en este trabajo, en la Figura 4.7 se muestra el esquema del algoritmo DQN y en Algorithm 2 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
          $y_i = r + \gamma * Q(s\prime, \arg\max_{a\prime} Q(s\prime,a\prime;\theta); \theta^{-})$
        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
  Actualizamos θ^* cada n episodios: $\theta^* \leftarrow \theta$;
  $epoch \leftarrow epoch + 1$;
end while

4.4.3 Dueling Network

La arquitectura de red denominada Dueling Network, propuesta en el artículo [59], tiene como objetivo separar explícitamente la estimación de los valores de estado y las ventajas de las acciones en el aprendizaje por refuerzo. En esta arquitectura, se utilizan dos flujos de información que comparten un módulo común de aprendizaje de características. Estos flujos representan la función

de valor del estado actual y la función de ventaja, respectivamente. Luego, se combinan utilizando una capa especial de agregación para obtener la estimación de la función de valor estado-acción Q. La fórmula general para el cálculo de Q en la arquitectura Dueling Network es la siguiente:

$$Q(s,a;\theta) = V(s;\theta') + \left(A(s,a;\theta'') - \frac{1}{|A|} \sum_{a'} A(s,a';\theta'') \right) \tag{4.33}$$

Donde:

- $V(s;\theta')$ es la estimación de la función de valor del estado actual.
- $A(s,a;\theta'')$ es la estimación de la función de ventaja al realizar la acción a en el estado s con respecto a las demás acciones posibles.
- θ representa los parámetros de la red neuronal.
- θ' y θ'' son los conjuntos de parámetros separados utilizados para estimar V y A respectivamente.

La adición de la línea de base de A en la fórmula (4.33) permite obtener una estimación más precisa de la función Q al tener en cuenta el promedio de las ventajas de todas las acciones posibles en el estado dado. Esto ayuda a separar la estimación del valor del estado de la influencia de las diferentes acciones. La arquitectura Dueling Network permite aprender de manera más eficiente qué estados son valiosos y cuáles no, sin necesidad de estimar el efecto de cada acción para cada estado. Esto resulta especialmente útil en situaciones en las que las acciones no tienen un impacto relevante en el entorno [59].

4.5 Ley de Recompensa

Para guiar al agente hacia el comportamiento óptimo, es necesario diseñar una función de recompensa que incentive a los agentes a cumplir con los siguientes objetivos:

1. La primera fase del patrullaje debe ser completamente exploratoria y la flota de agentes debe visitar de forma coordinada todo el mapa.
2. Después de medir la información en todo el lago en la primera fase, los agentes deben pasar a una fase explotativa donde se tiene en cuenta la importancia de cada zona.
3. Penalizar que los agentes tomen medidas en la misma zona del mapa.
4. Se deben evitar colisiones entre los vehículos.

4.5.1 Retos del problema multiagente

Cuando hay un solo agente, éste puede aprender de manera estable en un entorno estocástico porque puede atribuir las transiciones de estado a sus propias acciones. Sin embargo, cuando múltiples agentes interactúan en el mismo entorno sin conocimiento sobre las acciones conjuntas futuras, cada agente ve a los demás agentes como parte del entorno. Como resultado, el entorno

se vuelve no estacionario, ya que las transiciones de estado ahora dependen de las acciones de todos los agentes, pero cada agente solo tiene conocimiento de su propia acción. El aprendizaje se vuelve más complejo debido a la interacción entre los agentes y la falta de conocimiento sobre las acciones conjuntas futuras. Surge así el problema de la *no estacionariedad*, que es debido a que los agentes actualizan sus políticas durante el proceso de aprendizaje simultáneamente, de modo que el entorno parece no estacionario desde la perspectiva de un único agente.

Definition 4.5.1 (No estacionariedad)**.** *Un único agente se enfrenta a un problema de objetivo móvil (moving target) cuando la función de probabilidad de transición cambia debido a la coadaptación* $\pi^i \neq \hat{\pi}^i \;\; \exists i \in N$

$$\mathbb{P}(s\prime|a,\pi^1,...,\pi^N) \neq \mathbb{P}(s\prime|a,\hat{\pi}^1,...,\hat{\pi}^N) \tag{4.34}$$

En el artículo [55], se analiza la convergencia de algoritmos multiagente hacia soluciones óptimas. Los resultados indican que los agentes pueden converger a soluciones subóptimas o pueden quedarse estancados entre diferentes soluciones, a pesar de la alta aleatoriedad en la selección de acciones. En el artículo, se introduce el concepto de *shadowed equilibrium* (equilibrio ensombrecido).

Definition 4.5.2 (*Shadowed equilibrium*)**.** *Una política conjunta* $\bar{\pi}^i$ *está en la sombra de otra política conjunta* $\hat{\pi}$ *en un estado x si y sólo si:*

$$V_{\pi^i,\bar{\pi}^{-i}}(s) < \min_{j,\pi^j} V_{\pi^j,\hat{\pi}^{-j}}(s) \;\; \exists i,\, \pi^i \tag{4.35}$$

Un equilibrio está ensombrecido por otro cuando existe al menos un agente que, al desviarse unilateralmente de la política $\bar{\pi}^i$, no obtendrá una mejora mayor que al desviarse de la política $\hat{\pi}$. En otras palabras, al menos un agente podría obtener una recompensa mayor al desviarse de la política $\bar{\pi}^i$ y seguir la política $\hat{\pi}$. Esto implica que la política $\hat{\pi}$ es preferible para al menos un agente sobre la política $\bar{\pi}^i$. Como una forma de equilibrio en la sombra, la patología de la sobregeneralización relativa, describe que un equilibrio de Nash subóptimo en el espacio de acciones conjuntas es considerado como solución óptima. Esto sucede porque cada agente puede estar actuando de manera óptima en su propio contexto y en combinación con acciones arbitrarias de otros agentes, pero en conjunto, esto no garantiza una solución global óptima. En el contexto de la cooperación total, se busca que los agentes maximicen una señal de recompensa compartida de manera equitativa. Sin embargo, incluso en entornos completamente observables, es difícil determinar qué agentes y acciones contribuyeron al resultado final de la recompensa cuando los agentes no tienen acceso a la información sobre las acciones conjuntas. Un problema destacado en este contexto es el problema de asignación de créditos, donde se busca asociar las recompensas a los agentes de manera justa.

Definition 4.5.3 (Asignación de créditos)**.** *En el entorno totalmente cooperativo con señales de recompensa conjuntas un agente individual no puede concluir el impacto de su propia acción en el éxito del equipo y, por tanto, se enfrenta a un problema de asignación de créditos.*

Los autores de [60] demostraron que los algoritmos de aprendizaje independientes no pueden distinguir entre la exploración realizada por sus compañeros de equipo y la estocasticidad del

entorno, incluso en juegos de matriz simple. Esto dificulta el aprendizaje, ya que los agentes necesitan recibir retroalimentación sobre su desempeño en la tarea para poder aprender de manera efectiva. Además, el problema de asignación de créditos se vuelve más desafiante debido a la naturaleza secuencial del aprendizaje por refuerzo. Los agentes deben comprender no solo el impacto de sus acciones individuales, sino también las secuencias completas de acciones que conducen al resultado de la recompensa final. En nuestro sistema multiagente, los agentes tienen acceso a toda la información sobre los demás agentes, excepto la siguiente acción que tomarán. Esto plantea desafíos relacionados con la asignación de créditos y la determinación de la contribución de cada agente a los ajustes de recompensa compartidos. Para abordar estos desafíos, es necesario diseñar una función de recompensa que promueva la coordinación entre los agentes y proporcione a los agentes una señal de retroalimentación clara y significativa. En el artículo [61], se exploran las propiedades de la función de recompensa que conducen a un comportamiento efectivo del sistema multiagente. Las dos propiedades estudiadas fueron:

- Factorización (*Factoredness*): El grado de factorización se refiere a la fracción de estados en los que un cambio en el estado de un agente tiene el mismo impacto tanto en la recompensa del agente como en la recompensa global del sistema. Un alto grado de factorización significa que la recompensa del agente se mueve en la misma dirección que la recompensa global en respuesta a cambios en el estado del sistema. Si todas las recompensas de los agentes son iguales a la recompensa global, entonces el sistema se considera "totalmente" factorizado, lo que indica que los agentes están alineados en sus objetivos y acciones.
- Aprendibilidad (*learnability*): Una mayor aprendibilidad significa que es más fácil para un único agente tomar acciones (cambiar su estado) que maximicen su recompensa. Cuando la aprendibilidad es demasiado baja, muchos otros agentes están afectando la recompensa, por lo tanto, es difícil para un agente discernir los efectos de sus acciones de las acciones de todos los demás agentes. Esto significa que el agente tiene una mejor relación señal-ruido, lo que implica que el valor de la recompensa de un agente está más fuertemente influenciado por el estado actual del mismo agente.

En este trabajo, no se investigan nuevamente las propiedades de aprendibilidad y factorización, sino que se utilizan dos de las tres recompensas diferentes previamente analizadas en [61]. Estas recompensas ofrecen diferentes equilibrios entre aprendibilidad y factorización. Cada agente i realiza acciones para maximizar su propia recompensa R^i . El rendimiento del sistema se mide por la recompensa global R. Para cualquier agente i, el estado del sistema s se descompone en un componente que depende del estado del agente i, denominado s_i y un componente que no depende del estado del agente i, denominado s_{-i}

$$P_i(s) = R(s_i) \tag{4.36}$$
$$D_i(s) = R(s_i) - R(s_{-i}) \tag{4.37}$$

- **Recompensa Local (Ecuación 4.36)**: la recompensa local refleja la contribución de los estados del agente i a la recompensa global. Debido a que no depende de los estados de otros agentes, esta recompensa es perfectamente aprendible por cada agente. Sin embargo, en ciertos dominios, puede tener un bajo grado de factorización, lo que significa que los

cambios en el estado del agente i pueden no tener un impacto directo en la recompensa global. Aunque un agente puede maximizar fácilmente esta recompensa local, sus acciones pueden no ser óptimas para maximizar la recompensa global en su totalidad.

- **Diferencia de recompensas (Ecuación 4.37)**: logra un equilibrio entre la factorización y la aprendibilidad. Tiene alto grado de aprendabilidad porque solo depende de los estados del agente i y no de los estados de los otros agentes. Esto es porque al restar $G(s_{-i})$ se elimina el ruido de los otros agentes de la recompensa del agente i. El alto grado de factorización de D_i se debe a que su valor $G(s_{-i})$ no se ve afectado por los estados del agente i. Esto significa que cualquier cambio en la recompensa D_i se debe únicamente al impacto del agente en la recompensa global R. En resumen, cualquier acción que tenga un efecto positivo o negativo en D_i también tendrá el mismo efecto en R.

Uno de los objetivos de este trabajo es determinar cuál de estas recompensas es más adecuada para nuestro sistema en particular. Se busca evaluar y comparar los efectos de estas recompensas en términos de cuál consigue resolver mejor el problema del patrullaje planteado.

4.5.2 Matrices y parámetros en las funciones de recompensa

Formalmente, una función de recompensa es una función matemática que asigna un valor de recompensa a cada combinación de estado y acción tomada por el agente. Esta función puede reflejar premios o castigos y debe diseñarse teniendo en cuenta tanto las acciones deseables como las preferencias entre ellas.

$$R_e : (s,a) \rightarrow R \tag{4.38}$$

Es importante considerar el equilibrio en la asignación de valores de recompensa, evitando valores demasiado altos que puedan generar un comportamiento excesivamente avaricioso por parte del agente o fobias hacia acciones penalizadoras. La función de recompensa generalmente se diseña a través de un proceso de prueba y error, ajustando los valores y observando el comportamiento del agente en el entorno. Aunque nuestra misión de patrullaje sufre dos fases, exploración y explotación, el entorno proporciona recompensas independientemente de la fase en la que se encuentre la misión. En cada paso, se devuelven dos recompensas, una correspondiente al desempeño en la tarea de exploración y al desempeño en explotación. Estas recompensas no están condicionadas por la fase actual, sino que se entregan simultáneamente como si la misión estuviera en ambas fases al mismo tiempo. Este enfoque permite que el agente reciba información sobre el desempeño tanto en la exploración como en la explotación en cada paso, lo que puede ser útil para guiar su comportamiento y aprendizaje en ambas fases. La asignación y diseño de estas recompensas dependerá de los objetivos y criterios específicos de la misión de patrullaje, y se ha ido ajustando mediante iteración y prueba para lograr el comportamiento deseado del agente.

Para ambas fases, se define una matriz de *idleness* W. El propósito de esta matriz es tener un registro del tiempo transcurrido desde la última visita a cada celda del mapa. Inicialmente, al inicio de la misión, todas las celdas tendrán asignado el valor máximo en la matriz W, lo que indica que ninguna celda ha sido visitada aún. A medida que el agente explora y visita diferentes celdas, se actualizará la matriz W disminuyendo el valor correspondiente a las celdas visitadas. A medida que pasa el tiempo sin visitar una celda, su peso en la matriz W aumentará, lo que indicará

que esa celda lleva más tiempo sin ser visitada. Cuando los agentes realizan una acción conjunta a en el paso t y detectan las zona de interés en las coordenadas $[x,y]_{fleet}$ del mapa en el paso $t+1$, se procede a actualizar la matriz W en W_{t+1} de la siguiente manera:

$$W_{t+1} = \begin{cases} W(i,j)_{t+1} = \text{mín}\left[W(i,j)_t + \dfrac{1}{FF * MaxNOM} \ , \ 1\right] & \text{si } [x,y]_{fleet} \neq [i,j] \\ W(i,j)_{t+1} = 0 & \text{si } [x,y]_{fleet} = [i,j] \end{cases} , W \in \mathbb{R}^{M \times N} \tag{4.39}$$

Donde:

- (i,j): La matriz W se indexa mediante los índices (i,j), que representan las diferentes ubicaciones en el mapa. $i = 1,2,...,M$. $j = 1,2,...,N$.
- Forget Factor (FF): Determina el porcentaje de tiempo en relación a la duración náxima total del episodio que debe transcurrir antes de que una celda recupere su máximo valor de *idleness*. Es un valor en el rango de $(0,1]$, donde un valor de 1 indica que una celda tarda todo el episodio en recuperarse por completo, y un valor menor indica que una celda se recupera más rápidamente. Es importante destacar que el valor de FF no puede ser 0, ya que eso implicaría que las celdas nunca pierden su *idleness* y conservan su valor máximo de *idleness* de forma indefinida.
- Número máximo de pasos (MaxNOM): se calcula teniendo en cuenta la carga inicial de la batería del vehículo, la distancia máxima que puede recorrer el vehículo si partiera con la batería al máximo y la distancia recorrida por el agente en cada paso. Este número determina la duración máxima del episodio. Junto con el Forget Factor, establece cuántos pasos deben transcurrir desde la visita de una celda para que recupere su máximo valor de *idleness*. En la Ecuación 4.40 se muestra como se calcula este parámetro:

$$MaxNOM = \frac{Presupuesto\ de\ distancia}{radio\ de\ deteccin\ de\ los\ sensores} \tag{4.40}$$

  Donde el presupuesto de distancia son los kilómetros de autonomía que le queda al dron en función de la carga de la batería.

A continuación se muestran las matrices y los parámetros usados en todas las funciones de recompensas:

- MxN: tamaño de las matrices que contienen información del escenario.
- r: es el radio de detección de los sensores de los agentes.
- ω_t^i: Máscara binaria que vale 1 en las celdas donde alcanza la medida de sensores del agente $i \in N$ en el paso t. Se define como un círculo centrado en la posición del ASV, y su tamaño está determinado por el radio de detección r. Si consideramos las coordenadas en el eje x de las casillas ocupables del mapa de cuadrícula como $x_{navigable}$, las coordenadas en el

eje y como $y_{navigable}$, la posición del ASV como (i,j), y la distancia de detección como r, entonces la máscara ω^i se define de la siguiente manera:

$$\omega(i,j)^i = (x_{navigable} - i)^2 + (y_{navigable} - j)^2 < r^2 \tag{4.41}$$

- Redundancy mask (RM): Para penalizar que más de un agente realice la medida en la misma celda, la matriz de Redundancy mask (RM) registra en cada celda cuantas medidas se están realizando. Como en cada paso, cada agente realiza sólo una medida dentro de la máscara de detección:

$$RM_t = \sum_i^N \omega_t^i \tag{4.42}$$

  Esto nos proporciona información sobre la redundancia de las medidas en cada celda.

- Collective mask (CM): Es una matriz booleana que vale *false* en los valores donde RM vale 0 y *true* en los valores donde RM es mayor que 1. Representa las posiciones del mapa cubiertas por toda la flota.

- C_{ilegal}: Se utiliza para penalizar las acciones ilegales en la función de recompensa. Es una variable de diseño que determina la magnitud de la penalización. Es importante elegir un valor adecuado para C_{ilegal} para evitar que el agente desarrolle un temor excesivo a las penalizaciones, lo cual podría tener un impacto negativo en el proceso de patrullaje.

- Para referirse a una matriz exclusiva para un agente i, se utiliza un superíndice en la matriz colectiva para indicar que se deben enmascarar los valores que no están contenidos en la máscara de detección del agente i. El resultado es la matriz individual del agente i, obtenida al multiplicar elemento a elemento la matriz colectiva y la máscara del agente i. Por ejemplo, para la matriz RM:

$$RM_t^i = RM_t \circ w_t^i$$

4.5.3 Recompensa explorativa

Dado que el estado en el problema es parcialmente observable, es beneficioso tener una medida global del interés en todas las áreas del mapa. Esto permite identificar las zonas en las que se debe prestar una mayor atención o intensificar el monitoreo. Además, para construir un modelo preciso del mapa de contaminación del lago, es necesario tener información completa sobre todas las zonas. Esto implica que el agente debe explorar y recolectar datos en todas las áreas del mapa para obtener una representación completa y precisa de la contaminación. Por lo tanto, en la fase explorativa se busca visitar el máximo número de celdas posibles, esto en la literatura se llama *Coverage Path Planning* [3].

Para incentivar a los agentes a recoger información en todas las celdas del mapa, se ha realizado un seguimiento de las casillas que se visitan por primera vez en un paso. La matriz *NV (NewVisited)* almacena las casillas que se descubren en un paso específico. Es una matriz binaria donde el valor es 1 en las casillas recién descubiertas por toda la flota en ese paso y 0 en las casillas que ya habían sido visitadas anteriormente.

A continuación se muestran la recompensa de exploración que recibe un agente $i \in N$ al realizarse una acción conjunta a en el paso t según el tipo de recompensa:

1. Recompensa local:

$$R_t^i = \begin{cases} R_t^i = 0.2 * \sum_{m=0}^{M} \sum_{n=0}^{N} \dfrac{\left(1 + NV^i(m,n)\right) \times W_t^i(m,n)}{r \times RM_t^i(m,n)} & \text{si } \textit{la accion es legal} \\ R_t^i = -C_{ilegal} & \text{si } \textit{la accion no es legal} \end{cases} \tag{4.43}$$

2. Diferencia de recompensas:

$$R_t = 0.2 * \sum_{m=0}^{M} \sum_{n=0}^{N} \frac{(1 + NV(m,n)) \times CM_t(m,n) \times W_t(m,n)}{r \times RM_t(m,n)} \tag{4.44}$$

$$R_t^{-i} = 0.2 * \sum_{m=0}^{M} \sum_{n=0}^{N} \frac{\left(1 + NV^{-i}(m,n)\right) \times W_t^{-i}(m,n)}{r \times RM_t^{-i}(m,n)} \tag{4.45}$$

$$R_t^i = \begin{cases} R_t^i = R_t - R_t^{-i} & \text{si } \textit{la accion es legal} \\ R_t^i = -C_{ilegal} & \text{si } \textit{la accion no es legal} \end{cases} \tag{4.46}$$

Tanto en la recompensa local como en la diferencia de recompensas se establecen varios aspectos importantes del diseño de la función de recompensa que contribuyen a la optimización y eficacia del proceso de exploración:

1. La visita a una zona nueva tiene el doble de beneficio en comparación con visitar una zona de máximo "idleness" que ya ha sido visitada anteriormente. Esto se logra mediante el término $\left(1 + NV^i(m,n)\right)$, donde si la celda es visitada por primera vez, el valor es 2, mientras que si no es la primera visita, el valor es 1. Esto incentiva al agente a explorar y visitar todas las celdas al menos una vez.

2. Cuando varios agentes toman medidas en la misma celda, comparten la información contenida en esa celda. Esto implica que los agentes se distribuyen y evitan estar muy cerca unos de otros para obtener una mejor cobertura y aprovechar al máximo la información disponible. Esto se logra mediante el uso de la matriz RM, donde las celdas en las que varias máscaras de detección se solapan tienen un valor igual al número de máscaras que se solapan en esa celda. Esto permite que cada agente obtenga una parte proporcional de la información y la suma de todas las partes obtenidas por todos los agentes es igual a la información originalmente presente en esa celda.

3. Se normaliza la suma de las recompensas obtenidas con diferentes radios de detección dividiendo entre r. Esto se realiza para tener una medida comparativa equitativa de las recompensas, independientemente del tamaño del radio de detección utilizado.

4. Aunque se utilizan matrices globales en el cálculo de las recompensas, solo se considera la información contenida dentro del área de detección del vehículo (NV^i, W_t^i, RM_t^i). Esto significa que solo se tiene en cuenta la información relevante para el agente dentro de su rango de detección.

5. Se toman en cuenta las acciones ilegales mediante la asignación de una recompensa negativa (-1 en este caso, $C_{ilegal} = 1$) cada vez que se realizan acciones ilegales. Esto sirve como una penalización para desincentivar al agente a realizar acciones no permitidas.
6. Se multiplica por 0.2 el sumatorio de las recompensas obtenidas en la fase exploratoria. Esto se hace para que las recompensas de la fase exploratoria no sean significativamente mayores que las recompensas de la fase explotativa. El valor de 0.2 se ha determinado empíricamente para lograr un equilibrio adecuado entre las dos fases del proceso de patrullaje.

4.5.4 Recompensa a la explotación

A medida que los vehículos toman medidas en cada celda, guardan el valor de la última medida en cada celda en una matriz que denominaremos I. Esta será el modelo del mapa de contaminación del lago y se denominará matriz de importancia. La matriz de importancia relativa, I_R en el instante t, será el producto elemento a elemento (producto Hadamard) de las matrices W_t y I_t

$$I_{Rt} = W_t \circ I_t \tag{4.47}$$

La matriz I_R se puede visualizar como un mapa de calor, donde las zonas más frías representan un bajo interés relativo, mientras que las zonas más calientes indican un alto interés relativo. Esta representación gráfica proporciona información sobre las zonas del mapa que son más relevantes en términos de importancia. A continuación, se muestra la recompensa de explotación que recibe un agente i (pertenece a un conjunto de agentes N) cuando se realiza una acción conjunta a en el paso t.

1. Recompensa local:

$$R_t^i = \begin{cases} R_t^i = \sum_{m=0}^{M} \sum_{n=0}^{N} \dfrac{I_{Rt}^{\,i}(m,n)}{r \times RM_t^i(m,n)} & \text{si } \textit{la accion es legal} \\ R_t^i = -C_{ilegal} & \text{si } \textit{la accion no es legal} \end{cases} \tag{4.48}$$

2. Diferencia de recompensas:

$$R_t^i = \sum_{m=0}^{M} \sum_{n=0}^{N} \frac{CM_t(m,n) \times I_{Rt}(m,n)}{r \times RM_t(m,n)} \tag{4.49}$$

$$R_t^{-i} = \sum_{m=0}^{M} \sum_{n=0}^{N} \frac{I_{Rt}^{\,-i}(m,n)}{r \times RM_t^{-i}(m,n)} \tag{4.50}$$

$$R_t^i = \begin{cases} R_t^i = R_t - R_t^{-i} & \text{si } \textit{la accion es legal} \\ R_t^i = -C_{ilegal} & \text{si } \textit{la accion no es legal} \end{cases} \tag{4.51}$$

La ecuación (4.48) describe los aspectos clave en el diseño de la función de recompensa para mejorar la eficacia de la fase del patrullaje informativo:

1. Se tiene en cuenta la importancia de cada zona porque se utiliza la matriz I_{Rt} y por tanto no se busca una cobertura homogénea el mapa, sino visitar con más frecuencia las zonas contaminadas

2. Se han implementado los siguientes aspectos de la misma manera que en la recompensa de exploración: en el caso de que varios agentes realicen medidas en la misma celda, se comparte la información contenida en esa celda. Además, se realiza una normalización de las recompensas obtenidas con diferentes radios de detección para que sean comparables entre sí. Aunque se utilizan matrices globales en el cálculo de las recompensas, solo se tiene en cuenta la información dentro del área de detección del vehículo.
3. Al igual que en la fase explorativa: se toman en cuenta las acciones ilegales mediante la asignación de una recompensa negativa (-1 en este caso, $C_{ilegal} = 1$) cada vez que se realizan acciones ilegales.

4.6 Representación del estado

El estado es una representación del entorno en la que el agente se encuentra y es la información que tiene disponible para tomar decisiones. Es a través del estado que el agente percibe y entiende su entorno, y utiliza esta información para interactuar y tomar acciones. La forma en que se representa el estado es crucial, ya que afecta directamente el desempeño del agente y su capacidad para aprender y adaptarse al entorno. Una representación adecuada del estado proporciona al agente la información necesaria para tomar decisiones efectivas y lograr buenos resultados en su tarea. Primeramente, en nuestro entorno de simulación, el agente tiene información parcial sobre la contaminación del lago, ya que no conoce la contaminación de las celdas que nunca ha visitado. El agente mantiene un modelo de sensor para estimar la distribución de probabilidad de las observaciones. Sin embargo, en nuestro escenario, el agente solo utiliza la última información conocida sobre la contaminación de una celda para tomar decisiones. Además, para simular la dinámica de la floración de algas y su arrastre por el viento, hemos implementado una simulación en la que el mapa de contaminación del lago cambia de un paso a otro. El pico de interés se desplaza y algunas zonas cercanas al pico pueden experimentar un crecimiento en la concentración de algas. Esto introduce un elemento de cambio y dinamismo en el entorno, lo que requiere que el agente se adapte y tome decisiones actualizadas en cada paso. Por otro lado, los agentes deben saber la posición suya y la de los demás agentes, además de que tanto el modelo del mapa de contaminación, como la importancia relativa en el entorno debe ser el de la flota de agentes. Cada estado de cada agente se diferenciará por la posición, ya que es ilegal que dos o más agentes tomen la misma celda, y la posición de los demás agentes.

En nuestro escenario, cada agente debe tener conocimiento de su propia posición y la posición de los demás agentes. Esto es crucial para evitar colisiones entre agentes y garantizar un comportamiento seguro y eficiente. Cada agente tiene un estado que se compone de su posición individual, así como la posición de los demás agentes en el entorno y el modelo del mapa de contaminación y la importancia relativa. Todos los agentes tienen acceso a la misma información sobre el mapa de contaminación y la importancia relativa, por lo cual estos son compartidos por todos los agentes de la flota.

4.6.1 Estados comunes para todos los agentes

- Mapa del lago: Representa la superficie del lago Ypacaraí y las áreas circundantes en forma de una cuadrícula. Cada celda del mapa tiene un valor binario que indica si es navegable

(valor 1) o si corresponde a tierra (valor 0). El mapa se genera a partir de un archivo CSV que se crea utilizando una combinación de un mapa original y un ajuste manual para seleccionar solo las áreas navegables.

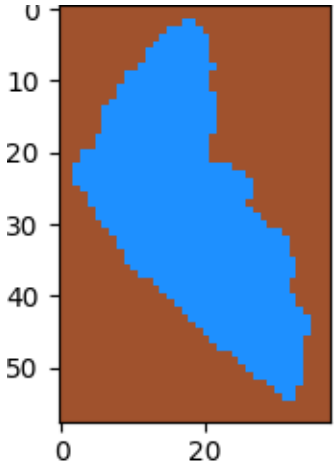


Figura 4.8 Mapa del lago.

- *Idleness* del mapa: Se genera una imagen del mismo tamaño que el grid-map y se calcula el *idleness* de cada celda según las visitas que han realizado los agentes en la flota. Esto permite que un agente pieda interés en una zona que ya ha visitado otro agente recientemente. La Figura 4.9 muestra un mapa donde las posiciones más recientemente visitadas están representadas por colores cálidos o rojos, mientras que las posiciones que no han sido visitadas en mucho tiempo están representadas por colores más fríos o azules.

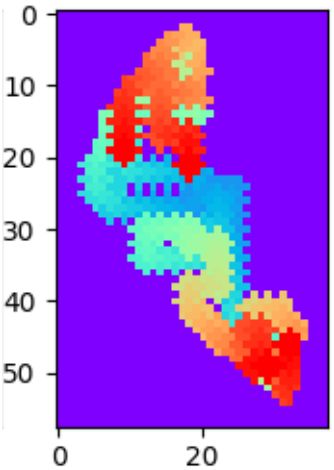


Figura 4.9 Mapa del *idleness* en el lago.

- Modelo de la contaminación del lago: A medida que los agentes visitan cada celda, recopilan información sobre el nivel de contaminación en esa zona. Este modelo de contaminación se actualiza a medida que se realizan nuevas mediciones. Se asume que las mediciones son precisas y que las zonas no visitadas tienen la misma información que la última medición realizada. Este estado permite a los agentes tener conocimiento de la contaminación del lago y tomar decisiones informadas durante la fase de explotación.

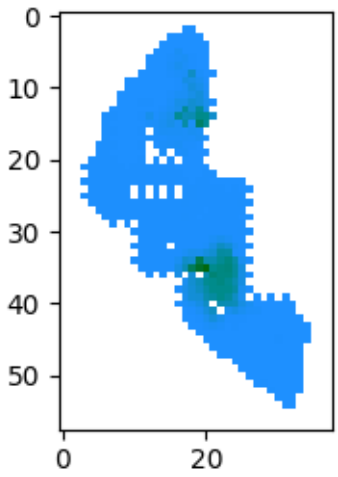


Figura 4.10 Mapa del *idleness* en el lago.

En la Figura 4.10, se puede observar cómo el agente registra la información en cada celda que ha visitado. Las celdas en blanco representan las que el agente aún no ha explorado, mientras que las celdas verdes indican un mayor nivel de interés. Cuanto más intenso sea el color verde, mayor será el interés en esa zona específica del entorno. Aunque en las zonas azules existe un interés casi nulo, es importante destacar siguen teniendo cierto nivel mínimo de interés. Esto se debe a que el agente no recibiría ninguna recompensa en una zona donde el interés es cero, incluso si el nivel de *idleness* es alto.

Estos estados comunes garantizan que todos los agentes tengan acceso a la misma información sobre el entorno y puedan coordinar sus acciones de manera efectiva.

4.6.2 Estados únicos a cada agente

En este escenario, cada agente tiene acceso a información única que se refiere a su propia posición y la posición de los demás agentes. Estos estados únicos se representan mediante imágenes binarias, que se describen a continuación:

- Posición del agente: Es una imagen binaria donde todas las celdas tienen un valor de cero, excepto la celda en la que se encuentra el agente. Esta imagen proporciona información específica sobre la ubicación actual del agente y le permite tomar decisiones basadas en su propia posición.

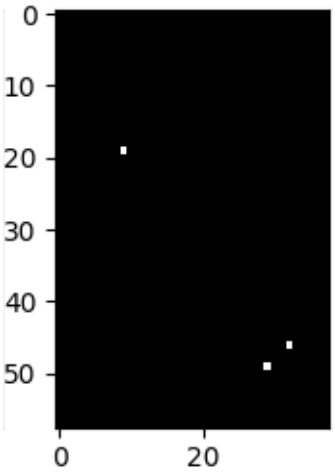


Figura 4.12 Posición de los otros agentes.

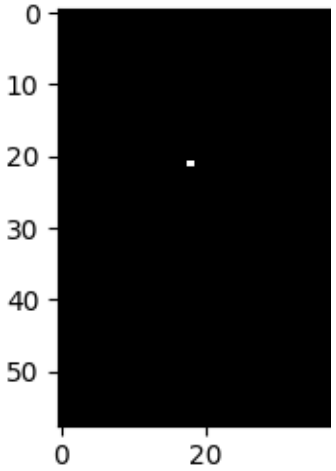


Figura 4.11 Posición del agente.

- Posición de los demás agentes: Es una imagen binaria donde todas las celdas tienen un valor de cero, excepto las celdas que corresponden a la ubicación de los demás agentes en el entorno. Esta imagen permite que cada agente tenga conocimiento de la posición de los demás agentes y evite colisiones o coordine sus acciones de manera adecuada.

4.7 Agente

4.7.1 Red neuronal del agente

En nuestra arquitectura de red neuronal, llamada Dueling Multihead, utilizamos una red convolucional densa con dos *heads* (terminaciones paralelas), donde cada *head* está asociada a una tarea

específica: la fase de exploración y la fase de explotación. Cada *head* tiene su propia capa de salida y parámetros, lo que permite que la red aprenda y optimice estas dos tareas de forma independiente. Esto permite que la red neuronal aprenda y optimice diferentes tareas simultáneamente. Cada *head* tiene su propia función de pérdida y se ajustan los pesos de cada *head* independientemente durante el entrenamiento. En el bloque compartido de la red convolucional (*Feature extractor*), se extraen características comunes que son útiles para ambas tareas. Estas características compartidas se aprenden durante el entrenamiento y se utilizan en las *heads* individuales para generar las salidas correspondientes a cada tarea. Se han explorado diferentes arquitecturas de redes neuronales para evaluar su rendimiento en el algoritmo y determinar si el número de capas compartidas tiene algún impacto en el desempeño. Se han evaluado dos arquitecturas de capa convolucional, denominadas "Net0" y "Net1". La principal diferencia entre ambas radica en que "Net1" cuenta con una capa adicional, y además la capa de entrada de esta arquitectura posee el doble de filtros de salida en comparación con "Net0". Esto significa que "Net1" debería tener una mayor capacidad de aprendizaje y procesamiento de características en las capas convolucionales.

Tabla 4.1 Arquitecturas de la red "Net0".

Net0	**Nº Filtros de entrada**	**Nº Filtros de salida**	***Kernel size***
Primera capa	5	64	3
Segunda capa	64	32	3
Tercera capa	32	16	3

Tabla 4.2 Arquitecturas de la red "Net1".

Net1	**Nº Filtros de entrada**	**Nº Filtros de salida**	***Kernel size***
Primera capa	5	128	3
Segunda capa	128	64	3
Tercera capa	64	32	3
Cuarta capa	32	16	3

En las *heads* de las arquitecturas consideradas, denominadas "Arch0" y "Arch1", existen diferencias en la forma en que se organizan las capas densas. En "Arch0", la capa densa que recibe la salida de las capas convolucionales es compartida entre ambas *heads*. Luego, cada *head* utiliza una arquitectura Dueling para calcular su propia función Q, donde se separa la estimación de la función de valor del estado actual y la estimación de la función de ventaja. En cambio, en "Arch1", las dos *heads* no comparten ninguna capa densa, lo que implica que cada *head* tiene su propia capa densa independiente.

Después de recibir la salida de las capas convolucionales, la capa densa aplica una función de activación ReLU a esa salida. A continuación, se agregan dos capas más a esta capa densa.

Tabla 4.3 Arquitecturas de la capa densa.

Dense	Nº Neuronas de entrada	Nº neuronas de salida	*Función de activación*
Primera capa	-	-	ReLU
Segunda capa	256	256	ReLU
Tercera capa	256	256	ReLU

La parte del Dueling que calcula función de ventaja de acciones es la siguiente:

Tabla 4.4 Arquitecturas de la capa función de ventaja.

Advantage	Nº Neuronas de entrada	Nº neuronas de salida	*Función de activación*
Primera capa	256	64	ReLU
Segunda capa	64	5	ReLU

Por otro lado, la parte del Dueling que calcula función de valor del estado es la siguiente:

Tabla 4.5 Arquitecturas de la capa función de valor del estado.

Value	Nº Neuronas de entrada	Nº neuronas de salida	*Función de activación*
Primera capa	256	64	ReLU
Segunda capa	64	1	ReLU

Por lo tanto, la arquitecturas "Arch0" y "Arch1" serían:

- **Arch0**: La capa densa de la Tabla 4.3 es única y la separación en dos *heads* comienza con la arquitectura Dueling.
  - *Head* de fase exploración/explotación: Calcular la función Q tal y como indica la Ecuación 4.33 con los valores de las capas de las tablas 4.4 y 4.5.
- **Arch1**: La capa densa de la Tabla 4.3 no es compartida y cada *head* tiene su propia capa densa. La separación en dos *heads* comienza a la salida de la red convolucional.

Para referirse a una red completa, que sería una combinación de las arquitecturas, se utiliza la notación "Net**x**-Arch**x**", siendo **x** 0 o 1.:

4.7.2 Transición entre fases exploratoria y explotativa

Nuestro enfoque innovador consiste en utilizar una variable llamada ν para controlar la transición suave entre las fases exploratoria y explotativa. Esta variable determina la probabilidad de seleccionar una acción de la política exploratoria o explotativa. Cuando ν es igual a 1, se elige la acción con el mayor valor de las salidas de la *head* que calcula la función Q en la fase exploratoria. Por otro lado, cuando ν es igual a 0, se eligen las acciones de la salida de la *head* explotativa. Un valor distinto de ν de 0 o 1 permite ajustar la probabilidad de seleccionar una acción de la

política exploratoria o explotativa. Este enfoque nos permite controlar de forma flexible y suave la transición entre las fases, adaptando la exploración y la explotación según sea necesario.

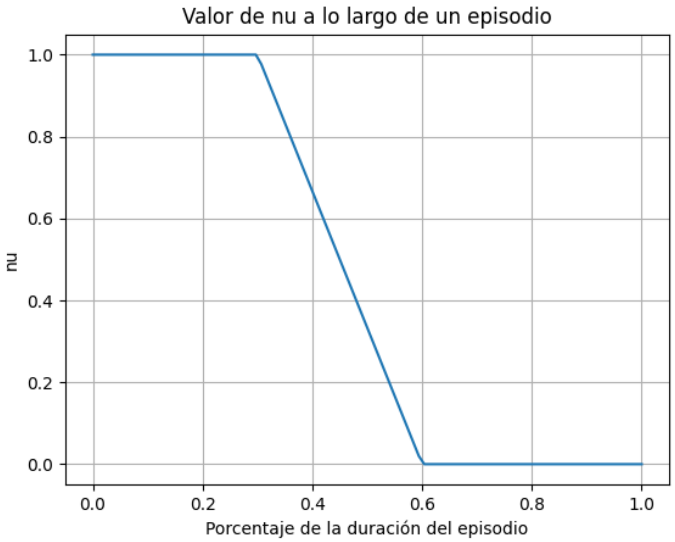


Figura 4.13 Valores de *nu* a lo largo del episodio.

Este mecanismo de transición basado en ν es una contribución científica original, ya que hasta donde tenemos conocimiento, no se había propuesto anteriormente en la literatura. Proporciona una forma novedosa y efectiva de manejar la transición entre las fases exploratoria y explotativa en el contexto del aprendizaje por refuerzo. Además, este enfoque cumple con la propiedad de que si se seleccionan intervalos equitativos entre la duración de la fase de exploración y la fase de explotación durante el entrenamiento, el agente aprende a generalizar y puede adaptarse a otros intervalos sin haber sido entrenado específicamente con ellos. En la Figura 4.13 se muestra cómo evoluciona ν a lo largo de un episodio. Durante los primeros 30 % del episodio, ν es igual a 1, lo que indica que se seleccionan exclusivamente acciones exploratorias. Luego se produce una transición gradual hacia acciones más explotativas. A partir del 60 % del episodio, ν se mantiene constante en 0 hasta el final, lo que implica que se sigue una política completamente explotativa.

4.7.3 Evitación de colisiones

Hemos implementado medidas para garantizar que nuestros agentes puedan identificar y evitar la orilla del lago, así como evitar la colisión con otros agentes en el escenario multiagente. Para evitar la colisión con la orilla del lago, hemos creado una máscara que impide que el agente elija acciones que lo lleven a celdas no navegables. Si el agente intenta ir a una celda no navegable, se selecciona la siguiente acción con la mayor valoración Q que no resulte en una colisión. Este enfoque nos permite garantizar que el agente evite la orilla y se mantenga en áreas navegables. En cuanto a la colisión entre agentes, hemos establecido que las acciones que resulten en colisiones sean consideradas ilegales. Estas acciones ilegales se penalizan en la ley de recompensa mediante

el parámetro C_{ilegal}. Además, hemos configurado el entorno para que un episodio se termine después de un número predeterminado de colisiones. Esto ayuda a evitar que los agentes intenten ir al mismo lugar y colisionen entre sí.

5 Resultados

En este Capítulo se analiza el comportamiento de nuestro sistema en las siguientes circunstancias:

- Recompensa Local: Analizaremos el problema de patrullaje cuando cada agente recibe una recompensa local.
- Diferencia de recompensas: Análisis del problema del patrullaje en el caso en el que cada agente recibe una diferencia de recompensas.
- Cambio de valores de ν: Comprobar si el algoritmo puede generalizar y adaptarse a otros intervalos de ν distintos de los utilizados durante el entrenamiento.

El algoritmo ha sido implementado en Python, utilizando la librería PyTorch para la construcción de la red neuronal. El entorno de simulación se ha basado en la librería Gym, mientras que las operaciones numéricas y matriciales se han realizado con las librerías NumPy y SciPy. Para la visualización gráfica del entorno y las métricas, se ha utilizado Matplotlib junto con Pandas y Seaborn. El buffer de experiencias se ha implementado utilizando la estructura de datos deque de la librería collections. El código y los resultados del algoritmo se encuentran disponibles en el repositorio de GitHub: *https://github.com/dsdiop/MultiAgentPatrollingProblem*.

Los hiperparámetros de entrenamiento se muestran en la Tabla 5.1. El parámetro $\nu - intervals$ define cuatro puntos específicos a lo largo de la duración total de un episodio en los cuales se debe establecer un valor particular para el parámetro ν. Cada punto está representado por un par de valores: el primero es el instante en el que se debe alcanzar ese punto, expresado como un porcentaje de la duración total del episodio, y el segundo es el valor de ν que se debe utilizar en ese instante. Estos puntos se utilizan para definir una función escalonada que determina cómo se ajusta el valor de ν a lo largo del tiempo en un episodio. La función escalonada se crea estableciendo los valores de ν en los puntos específicos y luego interpolando linealmente entre ellos. En la Figura 5.1 se muestra gráficamente como evoluciona el valor *nu* en un episodio con el valor de $\nu - intervals$ elegido en la Tabla 5.1.

Tabla 5.1 Hiperparámetros de entrenamiento.

Hiperparámetros	Valores	Comentarios
Batch size	64	Mejor tamaño con el que funciona el algoritmo
Replay memory size	1000000	Cuanto mayor sea este parámetro, menos posible es sufrir *catastrophic forgetting*
Target update	1000	
Polyak update constant	0.001	
$\varepsilon - value$	[1.0,0.05]	Con un valor mínimo de 0.05 siempre existe un 5 % de probabilidad de encontrar una acción mejor
$\varepsilon - interval$	[0,0.5]	El valor de ε decrece hasta 0.05 en la primera mitad del entrenamiento
γ	0.99	
learning rate	1e-4	
Train every	15	Se optimiza la red cada 15 steps.
Save every	2000	
$\nu - intervals$	[[0., 1], [0.30, 1], [0.60, 0.], [1., 0.]]	
Masked actions	True	Se enmascaran las acciones que llevan al agente fuera del espacio navegable del lago

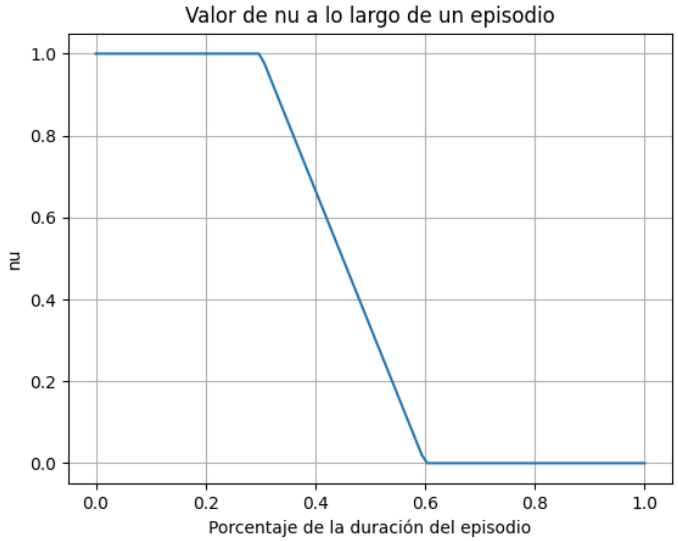


Figura 5.1 Valores de *nu* a lo largo del episodio.

Los entrenamientos fueron llevados a cabo en un *Escenario Parcialmente Observable* donde el mapa de contaminación simula el comportamiento del florecimiento dinámico de algas. Se realizaron 20000 episodios con los parámetros de entrada mostrados en la Tabla 5.2.

Tabla 5.2 Configuración del Escenario Completamente Observable.

Hiperparámetros	Valores
Presupuesto de la batería	100 %
Longitud de detección (r)	2 celdas
Longitud del movimiento	2 celdas
Distribución de información aleatoria	True
Número de kilómetros	200
Factor de olvido (FF)	0.5
Atrición	0.1
Colisiones permitidas	True
Número de colisiones permitidas	15

5.1 Métricas de funcionamiento

Se han definido las siguientes métricas de funcionamiento para evaluar el desempeño de nuestro algoritmo:

- **Interés ponderado**: Representa la media de la suma del interés ponderado recolectado por toda la flota durante el episodio. En la fase de exploración el íntéres es el *idleness* de cada celda y en la fase de explotación el interés ponderado es la importancia relativa de cada celda:

  1. Fase de exploración:

$$SOI_W = \frac{\sum_{t=0}^{T}\left[\sum_{m=0}^{M}\sum_{n=0}^{N}\frac{CM_t(m,n)\times W_t(m,n)}{r\times RM_t(m,n)}\right]}{T} \tag{5.1}$$

  2. Fase de explotación:

$$SOI_I = \frac{\sum_{t=0}^{T}\left[\sum_{m=0}^{M}\sum_{n=0}^{N}\frac{CM_t(m,n)\times W_t(m,n)\times I_{Rt}(m,n)}{r\times RM_t(m,n)}\right]}{T} \tag{5.2}$$

- **Importancia relativa media del escenario**: Para cada fase, se evalúa si el algoritmo está priorizando las zonas más importantes durante cada fase del patrullaje. Se considera que la fase exploratoria termina cuando el valor de v alcanza 0. Se utiliza el mapa de navegabilidad (*MAP*), que indica las celdas navegables (1) y no navegables (0). Se calcula la importancia relativa media del escenario para cada fase:

1. Fase de exploración: se hace una media del *idleness* durante la fase de exploración

$$M_W = \frac{1}{T}\sum_{t=0}^{T}\frac{\sum_{m=0}^{M}\sum_{n=0}^{N} MAP(m,n)\times W_t(m,n)}{\sum_{m=0}^{M}\sum_{n=0}^{N} MAP(m,n)} \tag{5.3}$$

2. Fase de explotación: se hace una media de la importancia relativa de las celdas en cada *time step* y se calcula el valor medio a lo largo del esta fase.

$$M_I = \frac{1}{T}\sum_{t=0}^{T}\frac{\sum_{m=0}^{M}\sum_{n=0}^{N} MAP(m,n)\times W_t(m,n)\times I_{Rt}(M,N)}{\sum_{m=0}^{M}\sum_{n=0}^{N} MAP(m,n)} \tag{5.4}$$

- **Porcentaje visitado del mapa**:

  Definimos la matriz *VM* como el registro de las casillas visitadas, donde el valor es 0 para las casillas no visitadas y 1 para las casillas ya visitadas. El porcentaje visitado del mapa en el instante t es:

$$PV_t = \frac{\sum_{m=0}^{M}\sum_{n=0}^{N} VM_t(m,n)}{\sum_{m=0}^{M}\sum_{n=0}^{N} MAP(m,n)} \tag{5.5}$$

  Se realiza un registro del porcentaje del mapa visitado en dos instantes:

  1. Porcentaje visitado durante la fase de exploración: Esta métrica mide el rendimiento del algoritmo en términos de exploración del mapa durante la fase de exploración. Si llamamos T_{exp} al instante en el que acaba la fase de exploración ($v < 0.5$):

$$PV_{exp} = \frac{\sum_{t=0}^{T_{exp}} PV_t}{T_{exp}} \tag{5.6}$$

  2. Porcentaje visitado durante a lo largo del episodio: Para medir el rendimiento del algoritmo en términos de exploración del mapa al finalizar del episodio:

$$PV = \frac{\sum_{t=0}^{T} PV_t}{T} \tag{5.7}$$

5.2 Comparación de recompensas

5.2.1 Recompensa Local

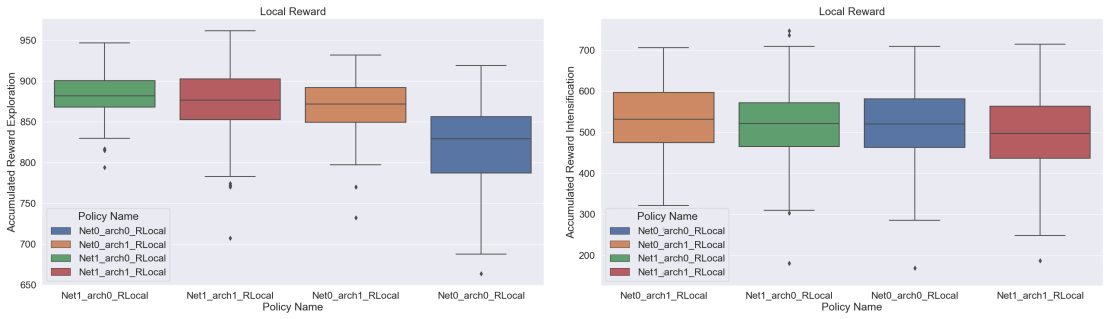
Tabla 5.3 Mediana de las métricas en 200 episodios. (↑): Cuanto mayor es la métrica mejor. (↓): Cuanto menor es la métrica mejor.

Política	SOI_I ↑	SOI_W ↑	M_I ↓	M_W ↓	$PV_{exp}(\%)$ ↑	$PV(\%)$ ↑
Net0-Arch0	520	**829**	**0.120**	**0.574**	**86.03**	**91.77**
Net0-Arch1	**531**	871	0.086	0.549	88.75	96.01
Net1-Arch0	521	**882**	**0.083**	**0.548**	88.75	**97.09**
Net1-Arch1	**496**	877	0.087	0.555	**89.48**	95.28

En la Tabla 5.3 se muestra la mediana de las métricas para las diferentes políticas evaluadas en 200 episodios. Se puede observar que la arquitectura "Net1-Arch0" tiene los mejores valores en general, alcanzando un 97 % de porcentaje del mapa visitado. Por otro lado, la arquitectura "Net0-Arch0" presenta los peores valores y un rendimiento inferior en comparación con las otras políticas.

A pesar de que existen diferencias sutiles entre las diferentes políticas en términos de rendimiento, ninguna de ellas presenta un rendimiento pobre (todas alcanzan al menos un 90 % de cobertura). Por ejemplo, las políticas "Net1-Arch0" y "Net0-Arch1" solo difieren en milésimas o centésimas en algunas métricas, lo cual es insignificante. Por lo tanto, la elección de la arquitectura no es un factor determinante que condicione por completo el rendimiento del algoritmo.

A continuación, se presentan los diagramas de caja, que proporcionan una representación visual de la distribución de las métricas del conjunto de datos estadísticos. Estos diagramas permiten tener una visión clara y rápida de cómo se distribuyen los valores en cada métrica, mostrando los cuartiles, la mediana y posibles valores atípicos.

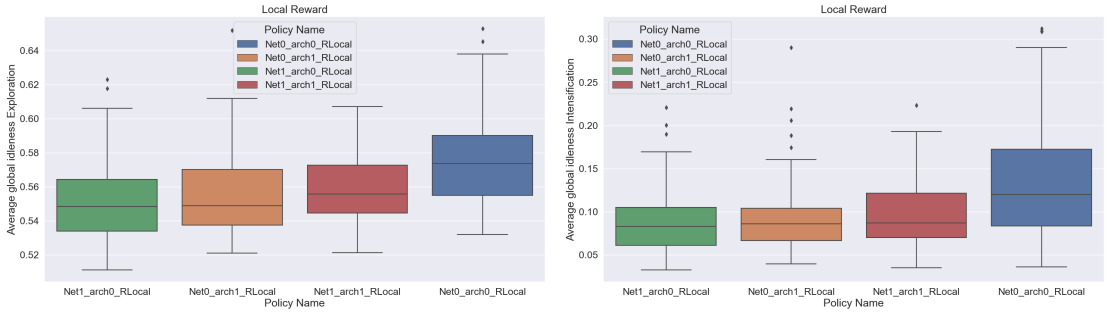


(a) Suma del *idleness* recolectado por toda la flota durante un episodio

(b) Suma del interés ponderado recolectado por toda la flota durante un episodio

Figura 5.2 Suma del interés ponderado recolectado.

En la Figura 5.2 se puede apreciar que la arquitectura "Net0-Arch0" presenta un rendimiento notablemente inferior en la fase explorativa en comparación con las demás arquitecturas mientras que las demás arquitecturas presentan en rangos similares. Sin embargo, todas las arquitecturas muestran un rendimiento similar en la fase intensificativa.

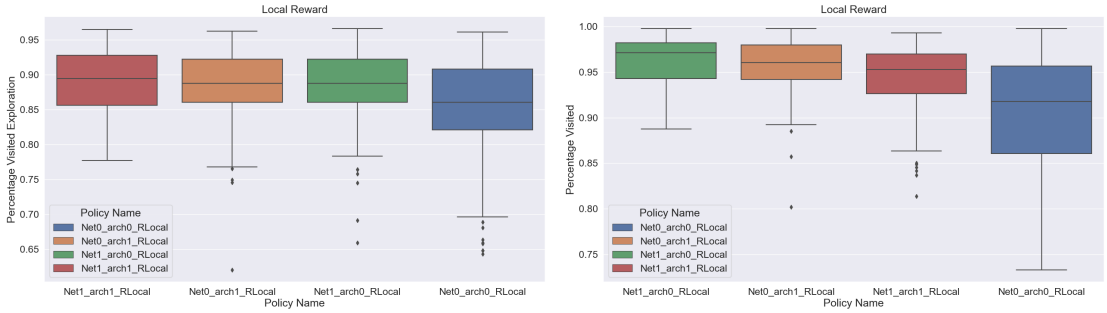


(a) Importancia relativa media del escenario en la fase de exploración

(b) Importancia relativa media del escenario en la fase de intensificación

Figura 5.3 Importancia relativa media del escenario.

Similar a las demás métricas, se aprecia un rendimiento pobre en la arquitectura "Net0-Arch0" en ambas fases en la Figura 5.3.



(a) Porcentaje del mapa visitado en la fase de exploración

(b) Porcentaje del mapa visitado al final del episodio

Figura 5.4 Porcentaje del mapa visitado.

Los diagramas de la Figura 5.4 indican que el algoritmo logra completar la fase de exploración del lago con éxito antes de pasar a la fase de intensificación. Incluso en el peor de los casos, se logra una cobertura de aproximadamente el 70 %, lo cual demuestra un rendimiento aceptable.

A continuación, se utilizan las matrices de número de visitas para mostrar la homogeneidad de la cobertura durante la fase de exploración y la concentración de visitas en zonas más relevantes durante la fase de intensificación. Estas matrices registran la cantidad de veces que cada celda ha sido visitada, proporcionando una representación visual de cómo el algoritmo explora y se enfoca en áreas específicas del escenario en cada una de las fases del proceso.

El mapa de contaminación en mitad del episodio es el siguiente, cabe destacar que, como el mapa es dinámico, la zona de concentración de interés se va desplazando hacia la derecha:

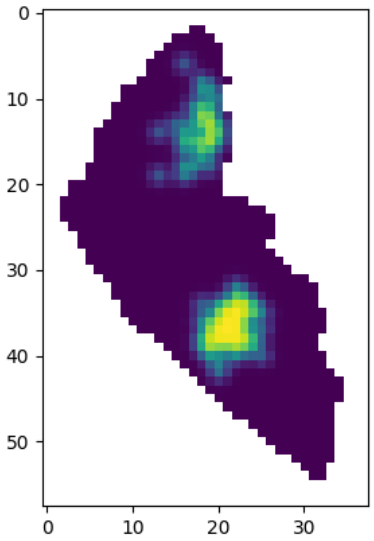
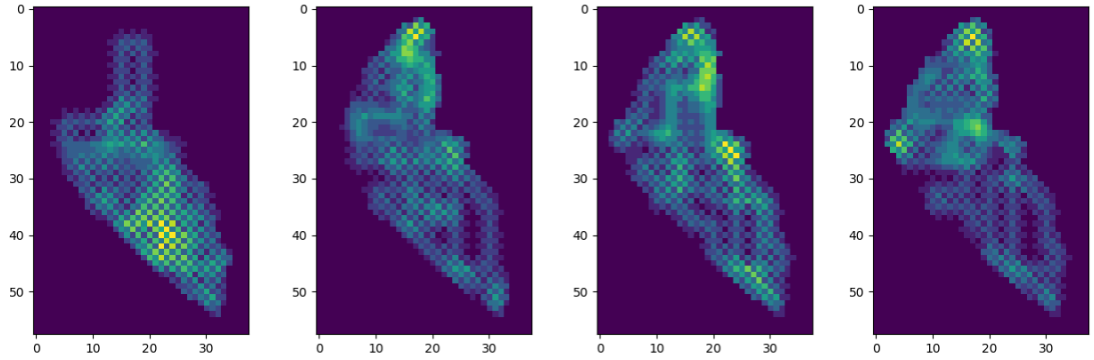


Figura 5.5 Mapa de contaminación.

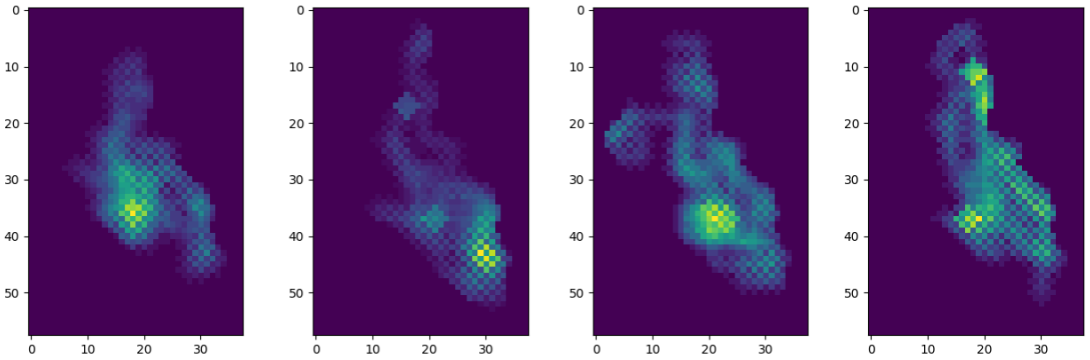


(a) Mapa de visitas Net0-Arch0 (b) Mapa de visitas Net0-Arch1 (c) Mapa de visitas Net1-Arch0 (d) Mapa de visitas Net1-Arch1

Figura 5.6 Mapa de visitas fase de exploración.

En la Figura 5.6 se puede observar que todos los algoritmos logran cubrir exitosamente todas las zonas del mapa durante la fase de exploración. Sin embargo, se puede notar que la arquitectura

"Net0-arch0" tiene una menor cobertura en comparación con las otras arquitecturas. Por otro lado, aunque la cobertura es total, se puede apreciar que no es perfectamente homogénea, ya que en algunos casos los agentes se dirigen al mismo lugar, lo que indica que la coordinación entre ellos no es óptima.



(a) Mapa de visitas Net0-Arch0 (b) Mapa de visitas Net0-Arch1 (c) Mapa de visitas Net1-Arch0 (d) Mapa de visitas Net1-Arch1

Figura 5.7 Mapa de visitas fase de explotación.

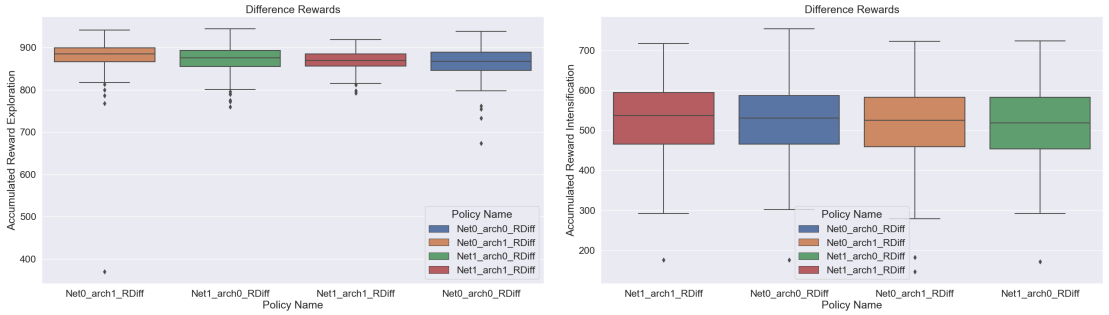
En la Figura 5.7 se puede observar que los agentes tienden a concentrarse en algunas zonas más que en otras durante la fase de exploración. Sin embargo, se puede apreciar que la transición de una fase a otra se ha realizado con éxito, lo que indica que los agentes han logrado identificar las zonas relevantes y han intensificado su patrullaje en esas áreas.

5.2.2 Diferencias de recompensa

Tabla 5.4 Mediana de las métricas en 200 episodios.

Política	$SOI_I \uparrow$	$SOI_W \uparrow$	$M_I \downarrow$	$M_W \downarrow$	$PV_{exp}(\%) \uparrow$	$PV(\%) \uparrow$
Net0-Arch0	530	**868**	0.086	**0.543**	90.02	95.89
Net0-Arch1	524	**884**	**0.066**	0.544	**92.14**	**96.86**
Net1-Arch0	**518**	878	0.088	0.548	89.90	94.92
Net1-Arch1	**536**	869	**0.095**	**0.554**	**87.66**	**94.43**

La Tabla 5.4 muestra la mediana de las métricas obtenidas en 200 episodios, y se puede observar que la arquitectura "Net0-Arch1" presenta un rendimiento notablemente superior en comparación con las demás arquitecturas. Esta arquitectura logra una cobertura de mediana del 92 % en la fase de exploración y reduce el interés ponderado a una mediana de 0.066, que es el valor más bajo de todos los experimentos. Sin embargo, es importante destacar que las demás arquitecturas también tienen un desempeño aceptable, con diferencias mínimas en algunas métricas. Por ejemplo, la arquitectura "Net1-Arch1" obtiene una mediana del 94.43 % en el porcentaje de mapa cubierto al final del episodio, lo cual es un resultado muy positivo. Aunque hay diferencias entre las arquitecturas, la elección de la arquitectura no es determinante para el éxito total del algoritmo. En este caso, la arquitectura "Net0-Arch1" destaca como la mejor opción.

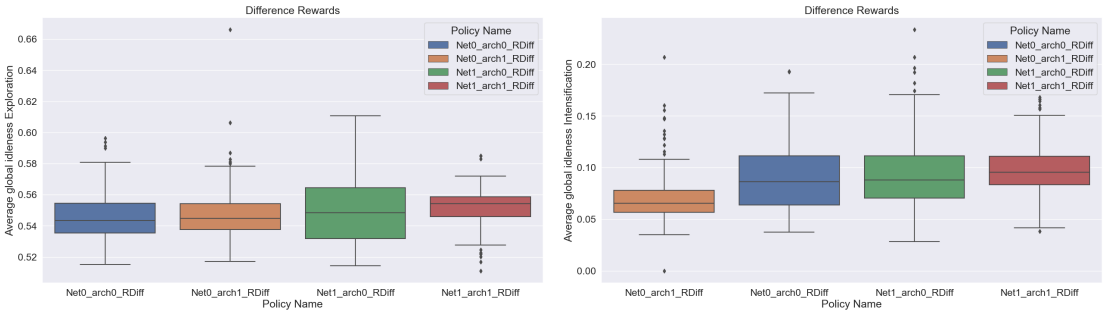


(a) Suma del *idleness* recolectado por toda la flota durante un episodio

(b) Suma del interés ponderado recolectado por toda la flota durante un episodio

Figura 5.8 Suma del interés ponderado recolectado.

En la Figura 5.8 se ve con más claridad que las arquitecturas muestran una tendencia a obtener resultados cercanos entre sí, ya que las cajas en el diagrama muestran una distribución similar del *idleness*/*interés ponderado*. Esto sugiere que, a pesar de algunas diferencias en el rendimiento entre las arquitecturas, en general, todas ellas logran resultados competitivos y consistentes en los experimentos realizados.

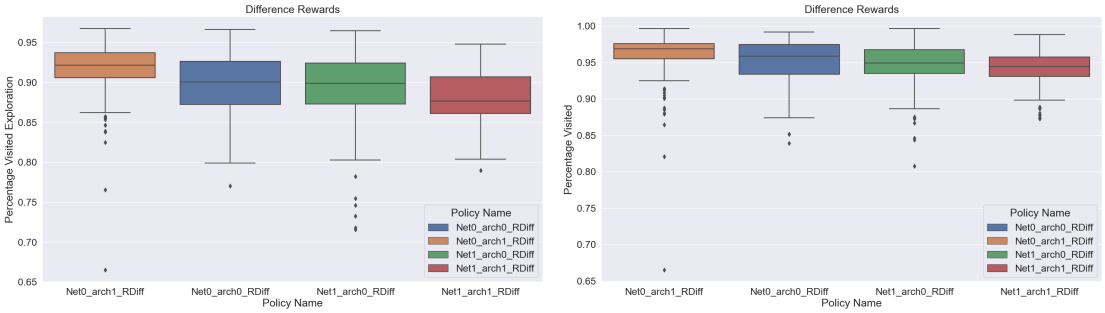


(a) Importancia relativa media del escenario en la fase de exploración

(b) Importancia relativa media del escenario en la fase de intensificación

Figura 5.9 Importancia relativa media del escenario.

En la Figura 5.9 se ve como, durante la fase de exploración, las arquitecturas muestran un rendimiento comparable en cuanto a la importancia relativa media del escenario. Sin embargo, en la fase explotativa, la arquitectura "Net0-Arch1" se destaca como la más eficiente, ya que logra mantener una importancia relativa media del escenario más baja y con una variabilidad más reducida en comparación con las demás arquitecturas.



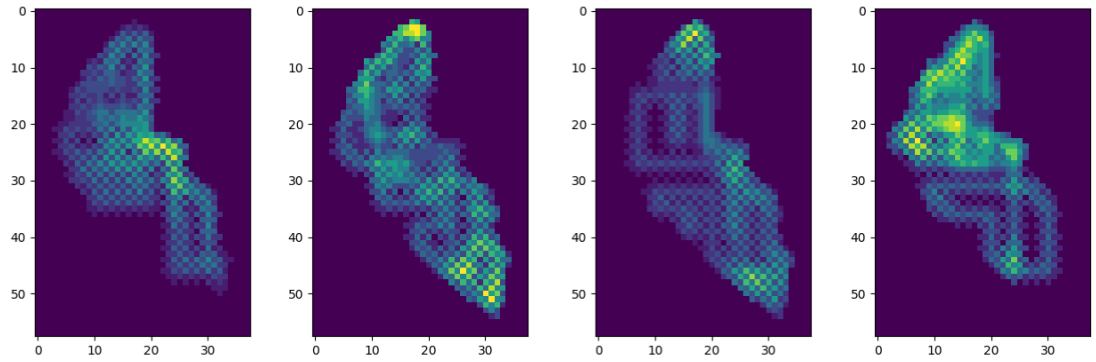
(a) Porcentaje del mapa visitado en la fase de exploración

(b) Porcentaje del mapa visitado al final del episodio

Figura 5.10 Porcentaje del mapa visitado.

Los diagramas de la Figura 5.10 nuevamente confirman la superioridad de la arquitectura "Net0-Arch1". Sin embargo, es importante destacar que los demás algoritmos también presentan un desempeño destacado.

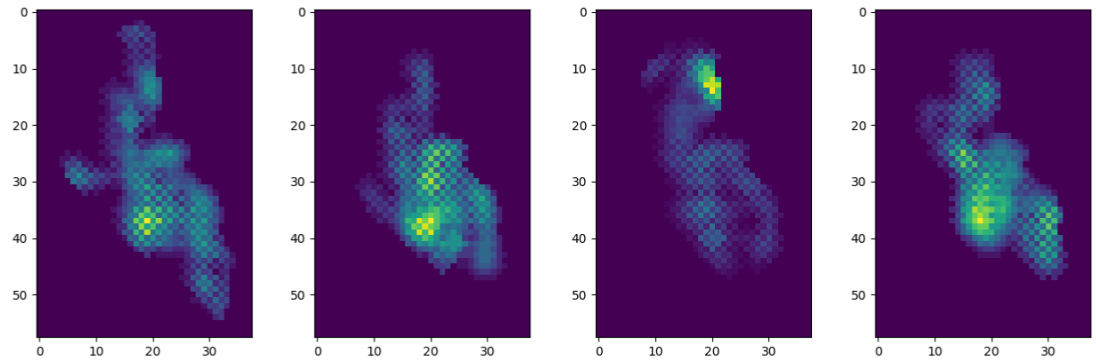
En la Figura 5.11, se puede observar que la arquitectura "Net0-Arch0" no logra realizar una exploración completa, ya que no descubre completamente el pico de interés ubicado en la parte inferior izquierda del mapa (Figura 5.5). Sin embargo, en la Figura 5.12 se aprecia que en la fase explotativa logra intensificar en esa zona. Aunque la fase explorativa no es óptima, se complementa con la fase explotativa, lo cual demuestra la ventaja de tener dos fases y refuerza el éxito de nuestra técnica para realizar la transición de una fase a otra utilizando la misma red neuronal.



(a) Mapa de visitas Net0-Arch0 (b) Mapa de visitas Net0-Arch1 (c) Mapa de visitas Net1-Arch0 (d) Mapa de visitas Net1-Arch1

Figura 5.11 Mapa de visitas fase de exploración.

Por otro lado, la "Net0-Arch1" muestra, una vez más, los mejores resultados.



(a) Mapa de visitas Net0-Arch0 (b) Mapa de visitas Net0-Arch1 (c) Mapa de visitas Net1-Arch0 (d) Mapa de visitas Net1-Arch1

Figura 5.12 Mapa de visitas fase de explotación.

5.2.3 Comparación entre las dos recompensas

En la tabla combinada, denominada Tabla 5.5, se fusionan los resultados de las dos tablas anteriores de la forma (diferencia de recompensas)-(recompensa local). Se resalta en negrita el valor más grande entre los dos y se agrega un asterisco al valor más alto de la mediana de la métrica. Observando de manera general, se puede afirmar que en las arquitecturas que contienen "Net0", el entrenamiento con la *diferencia de recompensas* funciona mejor, mientras que en las

Tabla 5.5 Mediana de las métricas en 200 episodios.

Política	$SOI_I \uparrow$	$SOI_W \uparrow$	$M_I \downarrow$	$M_W \downarrow$	$PV_{exp}(\%) \uparrow$	$PV(\%) \uparrow$
Net0-Arch0	**530**-520	**868**-829	**0.086**-0.120	**0.543***-0.574	**90.02**-86.03	**95.89**-91.77
Net0-Arch1	524-**531**	**884***-871	**0.066***-0.086	**0.544**-0.549	**92.14***-88.75	**96.86**-96.01
Net1-Arch0	518-**521**	878-**882**	0.088-**0.083**	**0.548-0.548**	**89.90**-88.75	94.92-**97.09***
Net1-Arch1	**536***-496	869-**877**	0.095-**0.087**	**0.554**-0.555	87.66-**89.48**	94.43-**95.28**

arquitecturas con "Net1", el entrenamiento con la *recompensa local* es más efectiva. En términos de los mejores resultados absolutos, la arquitectura "Net0-Arch1" entrenada con la *diferencia de recompensas* muestra un rendimiento destacado, con diferencias mínimas respecto a los demás mejores resultados absolutos. Por lo tanto, se considera que esta arquitectura y recompensa son las más adecuadas para nuestro estudio.

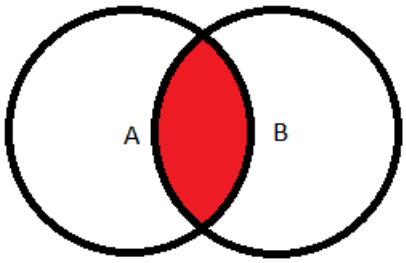


Figura 5.13 Solape del área de detección.

Por otra parte, la diferencia entre utilizar la *diferencia de recompensas* y la *recompensa local* en este sistema solo tiene efecto cuando dos agentes toman medidas en la misma área de detección. En la Figura 5.13 se ven dos agentes A y B, el área de alcance representado por un círculo concéntrico a la posición del agente, y el área de solape de estos círculos en color rojo. En las recompensas locales, la recompensa se calcula considerando tanto el área no solapada del círculo de alcance como la mitad del área solapada. En cambio, en las diferencias de recompensas, ambos agentes no reciben recompensas por el área solapada. Sin embargo, estos casos son tan poco frecuentes que no tienen un impacto significativo en los resultados, como se puede observar en las similitudes de las métricas en la Tabla 5.5. Aunque en general, los resultados son ligeramente mejores en las diferencias de recompensas, la diferencia es despreciable. Una vez más, es importante destacar que las diferencias de rendimiento entre las distintas arquitecturas y recompensas son mínimas e incluso en algunos casos insignificantes. Esto indica que el algoritmo es lo suficientemente robusto como para manejar estos cambios sin que tengan un impacto significativo en su desempeño general. El mecanismo propuesto para pasar de una fase de exploración a otro de intensificación funciona bien independientemente del fracaso de las políticas a la hora de la realización de la tarea. Por tanto, se ha conseguido que una sola red neuronal aprenda a realizar dos tareas de forma

secuencial en un entorno multiagente.

5.3 Generalización

A pesar de que todos los entrenamientos se realizaron utilizando los mismos valores para ν-intervals, se han realizado experimentos adicionales al variar estos valores. El objetivo es comprobar si el algoritmo ha aprendido verdaderamente a realizar las dos tareas de forma independiente y si puede tomar decisiones óptimas incluso en situaciones que nunca había encontrado durante el entrenamiento. Al variar los valores de ν-intervals, se busca evaluar la robustez del algoritmo y su capacidad para generalizar el conocimiento adquirido durante el entrenamiento. Si el algoritmo es capaz de adaptarse bien a diferentes valores de ν-intervals y tener un desempeño efectivo en ambas tareas, esto indicaría que ha aprendido patrones y estrategias generales en lugar de depender únicamente de los valores específicos de ν-intervals utilizados durante el entrenamiento.

Se seleccionó la arquitectura "Net0-Arch1" entrenada con diferencias de recompensas debido a su destacado rendimiento en los resultados obtenidos. En la figura Figura 5.14 se muestran las 10 curvas de evolución de ν utilizadas en el experimento. Cada curva representa un conjunto específico de valores de $\nu - interval$ y se registraron las métricas correspondientes a 200 episodios para cada configuración. Las diferentes curvas de evolución de ν representan diferentes estrategias de transición entre las fases de exploración y explotación durante un episodio. Estas estrategias se pueden resumir de la siguiente manera:

1. **Sólo exploración**: El agente se dedica únicamente a la exploración y no realiza transiciones a la fase de intensificación. En este caso, el agente continúa explorando durante todo el episodio.
2. **80-90, ..., 10-20**: Existen transiciones definidas en momentos específicos del episodio. Por ejemplo, en el intervalo "80-90", el agente explora hasta el 80 % de la duración del episodio, luego transita a la fase de intensificación hasta el 90 % del episodio y finalmente se concentra únicamente en la explotación.
3. **Sólo explotación**: No se realiza ninguna fase de exploración y el agente comienza a intensificar desde el primer instante del episodio, sin tener información previa sobre la contaminación del mapa.

(a) Sólo exploración

(b) 80-90

(c) 70-80

(d) 60-70

(e) 50-60

(f) 40-50

(g) 30-40

(h) 20-30

(i) 10-20

(j) Sólo explotación

Figura 5.14 Valores de ν.

La Figura 5.15 presenta diagramas de cajas que muestran la distribución de los porcentajes del mapa visitado en los 200 episodios para cada $\nu - interval$ utilizado. Se observa una tendencia en la cual a medida que la fase de exploración disminuye, también disminuye la media del porcentaje del mapa visitado. Cuando se realiza únicamente la fase de exploración, se obtiene una mediana del porcentaje del mapa visitado de aproximadamente el 99.63 %, con un intervalo de confianza del 2 %. Esto indica que durante esta fase, el agente logra explorar casi la totalidad del mapa. Incluso cuando la fase de exploración constituye solo el 10 % del episodio, se alcanza una mediana del porcentaje del mapa visitado de alrededor del 54.83 %. Esto demuestra que, a pesar de una exploración limitada en términos de duración, el agente aún logra visitar una proporción significativa del mapa. Por otro lado, cuando no hay fase de exploración, no se registra ningún porcentaje del mapa visitado, ya que el agente pasa directamente a la fase de intensificación sin explorar previamente. Por último, se destaca que con un intervalo de $\nu - interval$ de 40-50, donde la fase de exploración abarca la mitad del episodio, se logra visitar más del 90 % del mapa. Esto demuestra la robustez de la fase de exploración.

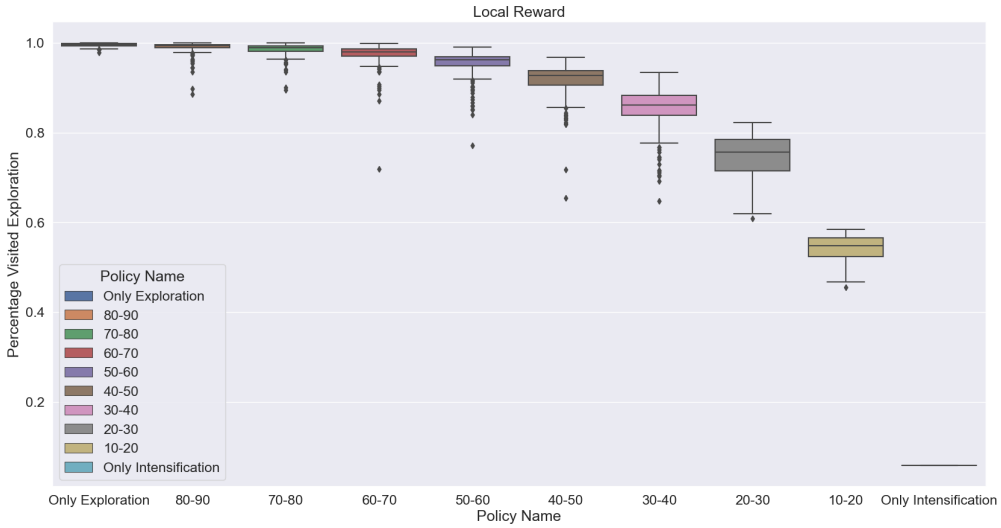


Figura 5.15 Porcentaje del mapa visitado al final de la fase de exploración.

El porcentaje del mapa visitado al final del episodio también sigue una tendencia similar al de la fase de exploración, como se ve en la Figura 5.15. Es importante destacar que, incluso cuando solo se realiza la fase de explotación, se logra visitar aproximadamente el 65 % del mapa, lo cual indica que el agente es capaz de cubrir una parte significativa del entorno sin necesidad de exploración previa ya que es necesario para encontrar picos de interés. Incluso en casos donde la fase de exploración es considerablemente más corta en comparación con la fase de explotación, la política basada en la explotación logra cubrir hasta un 30 % adicional del mapa en el peor de los casos ($nu - intervals$ 10-20). Esto se debe a la alta *idleness* de las zonas no descubiertas.

La Figura 5.17 presenta datos interesantes relacionados con la suma del interés ponderado recolectado por la flota en diferentes escenarios. Se observa que los casos en los que se dedica más del 50 % del episodio a la fase de exploración muestran una mayor suma de interés ponderado.

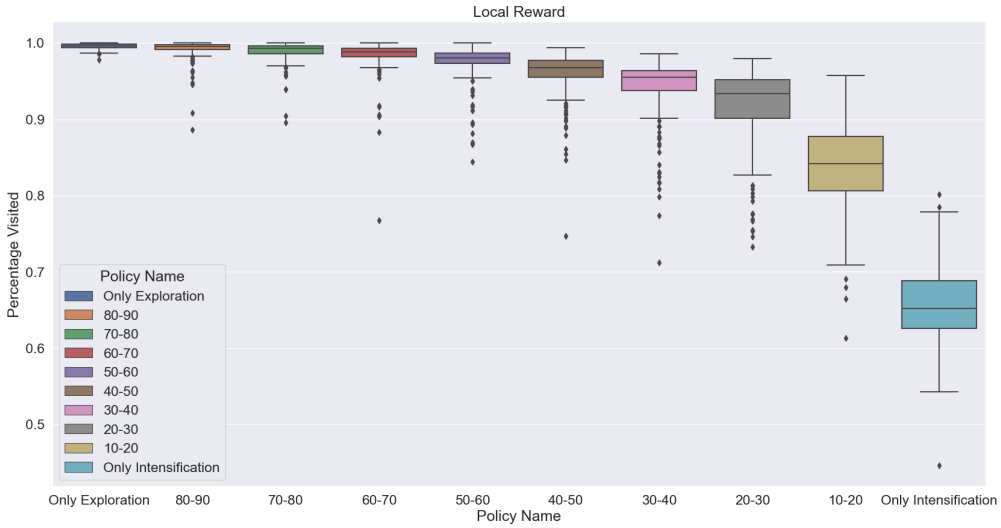


Figura 5.16 Porcentaje del mapa visitado al final del episodio.

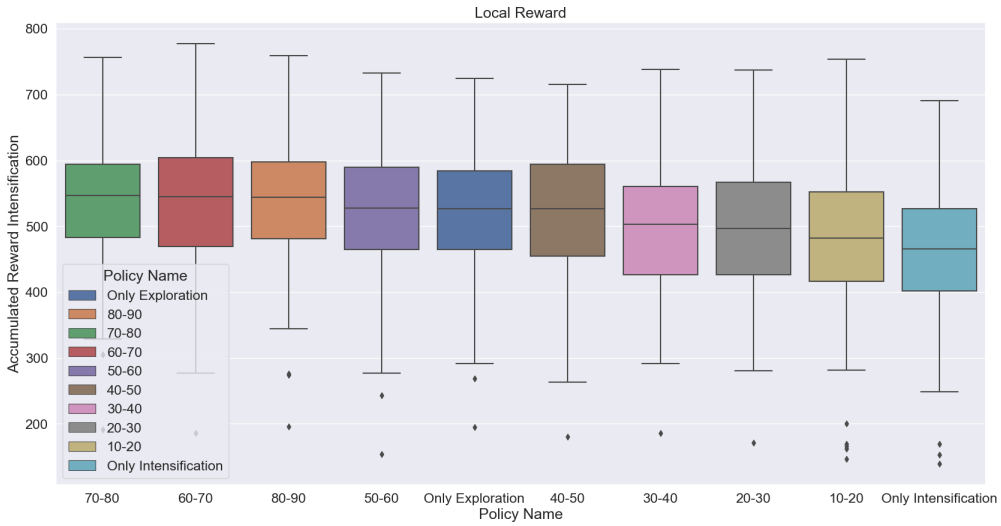


Figura 5.17 Suma del interés ponderado recolectado por toda la flota.

Esto se debe a dos factores clave:

1. En esos casos, el porcentaje del mapa visitado es cercano al 99 %, lo que indica que el agente ha explorado prácticamente la totalidad del entorno y, por lo tanto, tiene un conocimiento más completo para realizar una intensificación efectiva. La información recopilada durante la fase de exploración proporciona una base sólida para la toma de decisiones durante la fase de explotación.
2. Aunque la fase de exploración no se centra en la intensificación, el agente aún recopila interés durante esta etapa. Aunque esta recolección de interés puede no ser tan eficiente

como en la fase de explotación, sigue siendo beneficioso ya que permite obtener información sobre las zonas contaminadas, incluso de manera incidental. Esta información adicional puede tener un impacto positivo en el rendimiento general del agente.

En conclusión, la exploración inicial permite al agente descubrir y explorar nuevas áreas, mientras que la fase de explotación aprovecha la información recopilada para cubrir un porcentaje aún mayor del mapa. Aunque la fase de exploración puede ser breve en comparación con la fase de explotación, sigue siendo crucial para el rendimiento general del agente en términos de la cobertura del mapa.

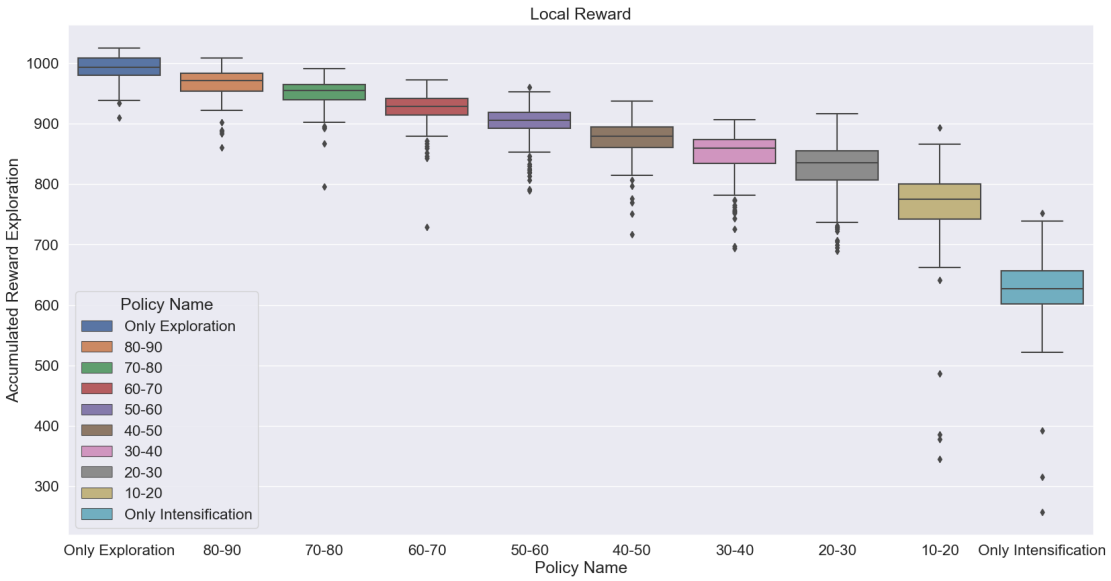
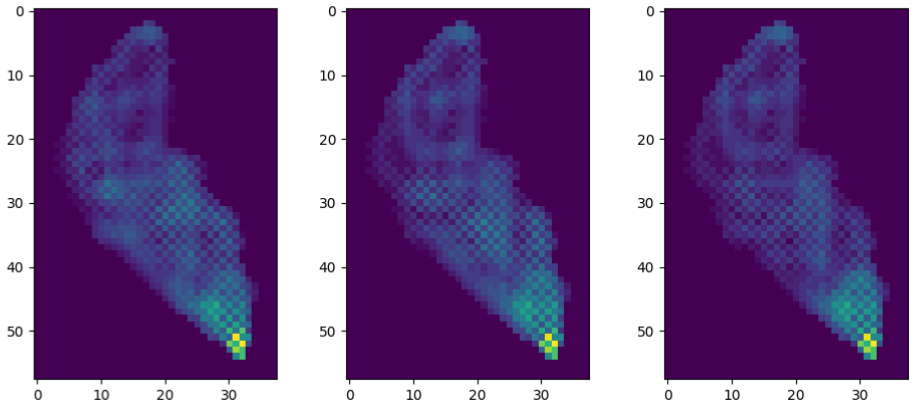


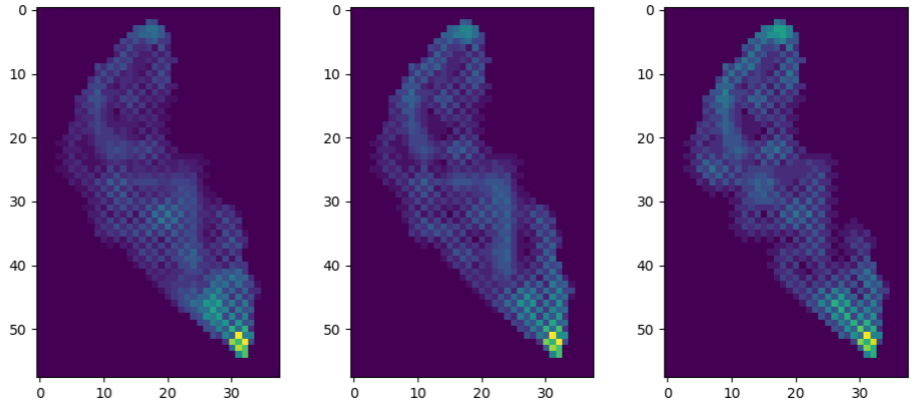
Figura 5.18 Suma del *idleness* recolectado por toda la flota.

En cuanto al *idleness* recogido, se observa que su valor disminuye a medida que se reduce la duración de la fase de exploración. Esta tendencia era esperada, ya que durante la exploración se visitan menos casillas en comparación con la fase de explotación. Es importante destacar que en la fase explotativa no solo se considera el *idleness*, sino también la importancia de la zona. En la fase de explotación, el agente toma decisiones basadas en la importancia de la zona, lo que implica que puede dirigirse a áreas que tienen mayor relevancia y posiblemente menor *idleness* que otras.

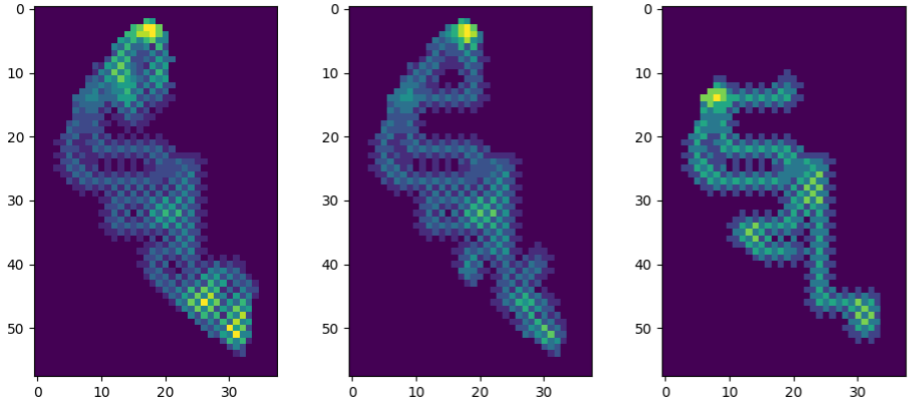
Por último las figuras 5.19 y 5.20 muestran como va cambiando la matriz de visitas de las fases a medida que se va aumentando la duración de la fase explotativo.



(a) Sólo exploración (b) 80-90 (c) 70-80

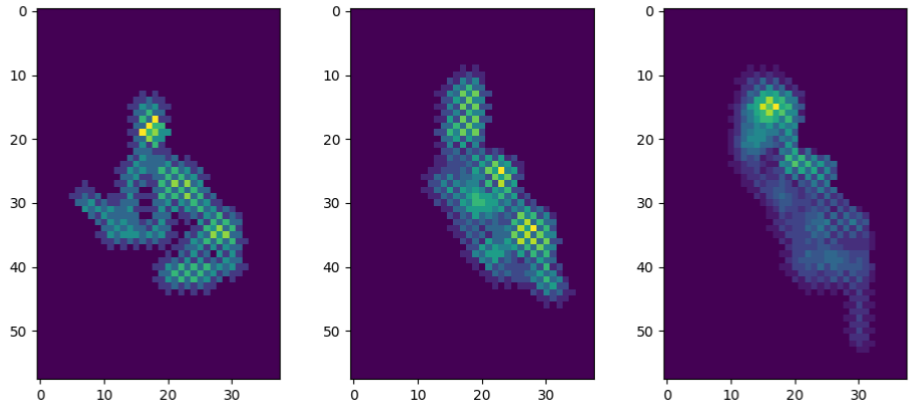


(d) 60-70 (e) 50-60 (f) 40-50

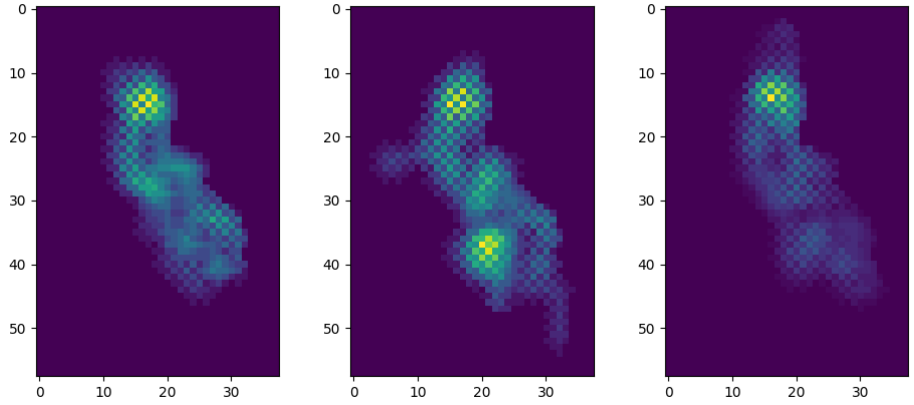


(g) 30-40 (h) 20-30 (i) 10-20

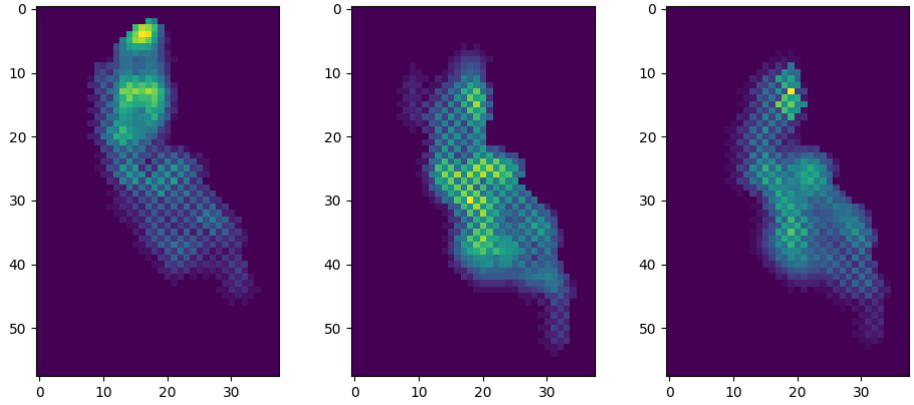
Figura 5.19 Mapa de visitas de la fase de exploración en los distintos $v - intervals$.



(a) 80-90 (b) 70-80 (c) 60-70



(d) 50-60 (e) 40-50 (f) 30-40



(g) 20-30 (h) 10-20 (i) Sólo explotación

Figura 5.20 Mapa de visitas de la fase de explotación en los distintos $v - intervals$.

6 Discusión y conclusiones

En los últimos tiempos, el uso de Aprendizaje por Refuerzo Profundo (DRL) en escenarios estocásticos y complejos ha ganado popularidad debido a su capacidad de generalizar y operar en situaciones no vistas previamente. La tarea de vigilancia del lago Ypacaraí representa un desafío significativo debido a su gran tamaño y la necesidad de una cobertura eficiente y no homogénea. En este sentido, el DRL se presenta como una técnica prometedora para abordar esta tarea. Específicamente, técnicas como Deep Q-Learning y variantes como Dueling Deep Q Learning permiten aprender la tarea sin necesidad de tener un modelo explícito del entorno, lo que facilita la adaptación a diferentes dinámicas e interacciones. En los últimos años, ha habido un aumento en el uso de múltiples Vehículos Autónomos de Superficie (ASV) para la monitorización de entornos contaminados. Esta tendencia se debe a que el uso de ASV reduce la necesidad de exponer a las personas a ambientes peligrosos para su salud. Sin embargo, para que los ASV puedan realizar eficazmente su labor, es necesario que sean capaces de generar y seguir waypoints que determinen la trayectoria óptima a seguir. En este sentido, es beneficioso utilizar el enfoque de Aprendizaje por Refuerzo Profundo en el caso multiagente, donde se deben abordar no solo la planificación reactiva de trayectorias, sino también la coordinación y prevención de colisiones entre los diferentes agentes. Mediante el uso de DRL en un entorno multiagente, los ASVs pueden aprender a tomar decisiones de manera autónoma y coordinada, evitando así la descoordinación y las colisiones entre ellos. Esto permite una monitorización eficiente y segura de los entornos contaminados.

6.1 Conclusiones

Para resolver el problema multiagente, se ha desarrollado una red neuronal que recibe como entrada: el estado actual del entorno percibido por todos los agentes y la posición de un agente y la de los demás agentes. Utilizando esta información, la red neuronal es capaz de determinar qué acción debe tomar ese agente. Como las experiencias entre agentes es equivalente, al final la red neuronal aprende de forma centralizada con las experiencias recolectadas por todos los agentes y adopta un comporta mejor que si los agentes hubieran aprendido de forma independiente [5].

En el caso del lago Ypacaraí, el escenario es Parcialmente Observable ya que la contaminación

es desconocida al inicio del episodio y en consecuencia los vehículos deben tomar medidas y planificar las trayectorias a la misma vez. Es por ello que se ha dividido la tarea de patrullaje en dos fases: la fase de exploración donde se busca una cobertura total del mapa y la fase de explotación donde se busca intensificar en las zonas más contaminadas. Para ello se han implementado dos técnicas:

- La red neuronal tiene dos cabezas, una estima la función Q que sirve para la fase exploratoria y la otra para la fase explotativa.
- Una nueva contribución científica que sirve para transicionar suavemente entre las dos fases. Consiste en crear una variable ν que sirva de probabilidad de selección de una acción exploratoria y explotativa y configurar una curva de evolución que permita determinar la duración de las fases y del tiempo de transición.

Se ha llevado a cabo un estudio sobre diferentes arquitecturas de red neuronal y se han comparado dos modelos de recompensa para evaluar el rendimiento de un algoritmo encargado de patrullar y coordinar agentes en la recopilación de información. Los resultados han demostrado que las diferencias de rendimiento entre las distintas arquitecturas y recompensas son mínimas, e incluso en algunos casos, insignificantes. Además, se ha observado que la diferencia entre utilizar la diferencia de recompensas y la recompensa local solo tiene un efecto significativo cuando dos agentes toman medidas en la misma área de detección. En general, estos hallazgos sugieren que el algoritmo es lo suficientemente robusto como para manejar estos cambios sin que afecten significativamente su desempeño general.

A pesar de que todos los entrenamientos se realizaron utilizando los mismos valores para, se han realizado experimentos adicionales al variar estos valores. Se observa una tendencia en la cual a medida que la fase de exploración disminuye, también disminuye la media del porcentaje del mapa visitado. Se destaca que con un intervalo de $\nu - interval$ de 40-50, donde la fase de exploración abarca la mitad del episodio, se logra visitar más del 90 % del mapa. Se observa que los casos en los que se dedica más del 50 % del episodio a la fase de exploración muestran una mayor suma de interés ponderado. El porcentaje del mapa visitado es cercano al 99 %, lo que indica que el agente ha explorado prácticamente la totalidad del entorno y, por lo tanto, tiene un conocimiento más completo para realizar una intensificación efectiva. La exploración inicial permite al agente descubrir y explorar nuevas áreas, mientras que la fase de explotación aprovecha la información recopilada para cubrir un porcentaje aún mayor del mapa. En conclusión, el algoritmo ha aprendido verdaderamente a realizar las dos tareas de forma independiente y puede tomar decisiones óptimas incluso en situaciones que nunca había encontrado durante el entrenamiento. Aunque todos los entrenamientos se realizaron utilizando los mismos valores para $\nu - intervals$, se han realizado experimentos adicionales variando estos valores. Se observa que a medida que disminuye la fase de exploración, disminuye también el porcentaje medio del mapa visitado. Además, se observa que con un intervalo $nu - intervals$ de 40-50, en el que la fase de exploración abarca la mitad del episodio, se visita más del 90 % del mapa. Asimismo, se observa que los casos en los que más del 50 % del episodio se dedica a la fase de exploración muestran una mayor suma de interés ponderado. Esto es porque el porcentaje del mapa visitado es cercano al 99 % en estos casos, lo que indica que el agente ha explorado prácticamente todo el entorno y ha adquirido un conocimiento más completo para realizar una intensificación efectiva.

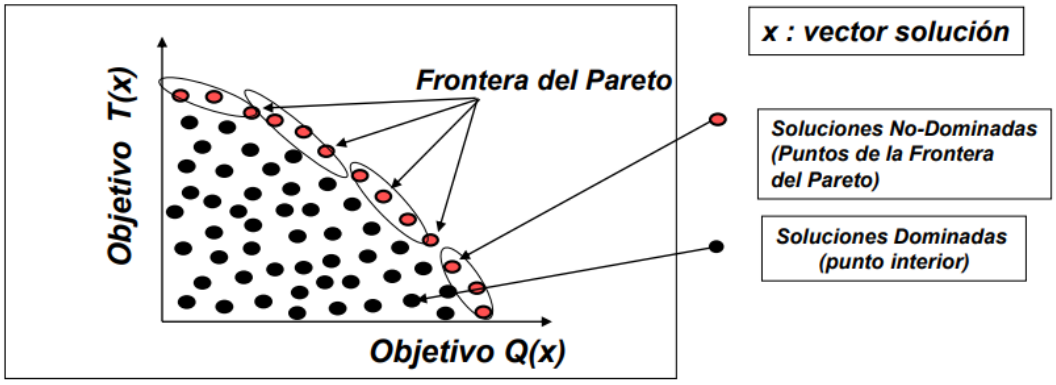


Figura 6.1 Ejemplo de frente (frontera) de Pareto donde se busca maximizar dos objetivos.

En conclusión, el algoritmo ha aprendido a realizar ambas tareas de forma independiente y puede tomar decisiones óptimas incluso en situaciones no encontradas durante el entrenamiento.

6.2 Líneas futuras

En este trabajo nos hemos enfocado en la monitorización de un lago llevado a cabo por varios Vehículos Autónomos de Superficie (ASV)s. Sin embargo, hemos encontrado dificultades para optimizar la coordinación entre los ASV debido a las colisiones que ocurren entre ellos. Por lo tanto, sería beneficioso desarrollar una técnica de negociación o coordinación que pueda prevenir las colisiones entre los agentes cuando se encuentren en situaciones de riesgo de colisión.

En nuestro estudio, hemos entrenado una red neuronal utilizando un valor fijo para $\nu - intervals$ y hemos observado que la red puede generalizar eficazmente a diferentes valores de $\nu - intervals$. Sin embargo, sería interesante volver a entrenar la red con diferentes valores de $\nu - intervals$ para determinar cuál configuración proporciona el mejor rendimiento. Dado que hay dos objetivos a optimizar, exploración y explotación, sería apropiado buscar valores de $\nu - intervals$ sin tener ninguna información previa sobre las preferencias del usuario y considerar igual importancia para ambos objetivos. Nos encontramos ante un problema de optimización multiobjetivo, donde la solución óptima se encuentra en el frente de Pareto. El frente de Pareto consiste en un conjunto de soluciones no dominadas, lo que significa que ninguna solución puede mejorar un objetivo sin empeorar otro. Una solución domina a otra si es mejor o igual en todos los objetivos y mejor en al menos uno de ellos. Para determinar la solución óptima, debemos identificar el conjunto de soluciones de Pareto, que cumplen con la propiedad de Optimalidad de Pareto. Una solución es Pareto-óptima si no es dominada por ninguna otra solución en el espacio de soluciones. Por lo tanto, debemos buscar un frente de Pareto que muestre las políticas exploratorias y explotativas entrenadas con diferentes valores de $\nu - intervals$ y que muestre las políticas no dominadas, es decir, aquellas soluciones que no pueden ser mejoradas en ningún objetivo sin empeorar otro. En la Figura 6.1 se muestra un ejemplo de frente de Pareto (también llamado frontera de Pareto) donde se maximizan dos objetivos.

Por último, los algoritmos genéticos son excelentes en la optimización multiobjetivo. Estos

métodos se basan en el proceso genético de los organismos vivos y utilizan una analogía directa con el comportamiento natural. En los algoritmos genéticos, se genera una población de individuos, donde cada individuo representa una solución y tiene una puntuación que refleja la calidad de esa solución. Los individuos con puntuaciones más altas tienen una mayor probabilidad de ser seleccionados y sobrevivir en el proceso. 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.

Las principales ventajas de los algoritmos genéticos son que:

- Son algoritmos poblaciones que son capaces de obtener múltiples soluciones en una sola ejecución.
- Pueden explorar diferentes áreas del espacio de soluciones.
- No son sensibles a la forma específica del frente de Pareto.

Sin embargo, es importante tener en cuenta que el entrenamiento con algoritmos genéticos puede llevar tiempo. Aunque su convergencia es estable, el proceso de entrenamiento puede ser prolongado. A pesar de esto, los algoritmos genéticos son eficaces en encontrar soluciones óptimas para problemas de optimización multiobjetivo.

Índice de Figuras

Índice de Tablas

Bibliografía

[1] S. Yanes Luis, D. Gutiérrez-Reina, and S. Toral Marín, "A dimensional comparison between evolutionary algorithm and deep reinforcement learning methodologies for autonomous surface vehicles with water quality sensors," *Sensors*, vol. 21, no. 8, 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/8/2862

[2] M. Arzamendia, D. Gutierrez, S. Toral, D. Gregor, E. Asimakopoulou, and N. Bessis, "Intelligent online learning strategy for an autonomous surface vehicle in lake environments using evolutionary computation," *IEEE Intelligent Transportation Systems Magazine*, vol. 11, no. 4, pp. 110–125, 2019.

[3] S. Y. Luis, D. G. Reina, and S. L. T. Marín, "A deep reinforcement learning approach for the patrolling problem of water resources through autonomous surface vehicles: The ypacarai lake case," *IEEE Access*, vol. 8, pp. 204 076–204 093, 2020.

[4] F. P. Samaniego, D. G. Reina, S. L. T. Marín, M. Arzamendia, and D. O. Gregor, "A bayesian optimization approach for water resources monitoring through an autonomous surface vehicle: The ypacarai lake case study," *IEEE Access*, vol. 9, pp. 9163–9179, 2021.

[5] S. Yanes Luis, D. Gutiérrez, and S. Toral, "A multiagent deep reinforcement learning approach for path planning in autonomous surface vehicles: The ypacarai-lake patrolling case." *IEEE Access*, vol. PP, 01 2021.

[6] R. S. Sutton, A. G. Barto *et al.*, "Introduction to reinforcement learning," 1998.

[7] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, no. 1, 2016.

[8] M. Hessel, J. Modayil, H. van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, "Rainbow: Combining improvements in deep reinforcement learning," 2017.

[9] M. Theile, H. Bayerlein, R. Nai, D. Gesbert, and M. Caccamo, "Uav coverage path planning under varying power constraints using deep reinforcement learning," in *2020 IEEE/RSJ*

International Conference on Intelligent Robots and Systems (IROS). IEEE, 2020, pp. 1444–1449.

[10] C. Piciarelli and G. L. Foresti, "Drone patrolling with reinforcement learning," in *Proceedings of the 13th International Conference on Distributed Smart Cameras*, 2019, pp. 1–6.

[11] R. Shah, Y. Jiang, J. Hart, and P. Stone, "Deep r-learning for continual area sweeping," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5542–5547.

[12] S. Gronauer and K. Diepold, "Multi-agent deep reinforcement learning: A survey," *Artif. Intell. Rev.*, vol. 55, no. 2, p. 895–943, feb 2022. [Online]. Available: https://doi.org/10.1007/s10462-021-09996-w

[13] X. Lyu and C. Amato, "Likelihood quantile networks for coordinating multi-agent reinforcement learning," 2020.

[14] G. Palmer, R. Savani, and K. Tuyls, "Negative update intervals in deep multi-agent reinforcement learning," 2019.

[15] G. Palmer, K. Tuyls, D. Bloembergen, and R. Savani, "Lenient multi-agent deep reinforcement learning," in *Proceedings of the 17th International Conference on Autonomous Agents and MultiAgent Systems*, ser. AAMAS '18. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2018, p. 443–451.

[16] A. Tampuu, T. Matiisen, D. Kodelja, I. Kuzovkin, K. Korjus, J. Aru, J. Aru, and R. Vicente, "Multiagent cooperation and competition with deep reinforcement learning," *PloS one*, vol. 12, no. 4, p. e0172395, 2017.

[17] J. K. Gupta, M. Egorov, and M. Kochenderfer, "Cooperative multi-agent control using deep reinforcement learning," in *Autonomous Agents and Multiagent Systems: AAMAS 2017 Workshops, Best Papers, São Paulo, Brazil, May 8-12, 2017, Revised Selected Papers 16*. Springer, 2017, pp. 66–83.

[18] X. Zhou, P. Wu, H. Zhang, W. Guo, and Y. Liu, "Learn to navigate: Cooperative path planning for unmanned surface vehicles using deep reinforcement learning," *IEEE Access*, vol. 7, pp. 165 262–165 278, 2019.

[19] S. Yanes Luis, D. Gutiérrez-Reina, and S. Toral Marín, "Censored deep reinforcement patrolling with information criterion for monitoring large water resources using autonomous surface vehicles," *Applied Soft Computing*, vol. 132, p. 109874, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1568494622009231

[20] J. N. Foerster, Y. M. Assael, N. de Freitas, and S. Whiteson, "Learning to communicate with deep multi-agent reinforcement learning," *CoRR*, vol. abs/1605.06676, 2016. [Online]. Available: http://arxiv.org/abs/1605.06676

[21] W. Shi, J. Li, H. Wu, C. Zhou, N. Cheng, and X. Shen, "Drone-cell trajectory planning and resource allocation for highly mobile networks: A hierarchical drl approach," *IEEE Internet of Things Journal*, vol. PP, pp. 1–1, 08 2020.

[22] K. W. Kim, "Vector criterion markov decision processes." 1981.

[23] K. Van Moffaert, M. M. Drugan, and A. Nowé, "Scalarized multi-objective reinforcement learning: Novel design techniques," in *2013 IEEE Symposium on Adaptive Dynamic Programming and Reinforcement Learning (ADPRL)*, 2013, pp. 191–199.

[24] Y. Guo, A. Zeman, and R. Li, "A reinforcement learning approach to setting multi-objective goals for energy demand management," *Int. J. Agent Technol. Syst.*, vol. 1, no. 2, p. 55–70, apr 2009. [Online]. Available: https://doi.org/10.4018/jats.2009040104

[25] T. T. Nguyen, "A multi-objective deep reinforcement learning framework," *CoRR*, vol. abs/1803.02965, 2018. [Online]. Available: http://arxiv.org/abs/1803.02965

[26] A. Abdolmaleki, S. H. Huang, G. Vezzani, B. Shahriari, J. T. Springenberg, S. Mishra, D. TB, A. Byravan, K. Bousmalis, A. György, C. Szepesvári, R. Hadsell, N. Heess, and M. A. Riedmiller, "On multi-objective policy optimization as a tool for reinforcement learning," *CoRR*, vol. abs/2106.08199, 2021. [Online]. Available: https://arxiv.org/abs/2106.08199

[27] A. Abdolmaleki, S. H. Huang, L. Hasenclever, M. Neunert, H. F. Song, M. Zambelli, M. F. Martins, N. Heess, R. Hadsell, and M. A. Riedmiller, "A distributional view on multi-objective policy optimization," *CoRR*, vol. abs/2005.07513, 2020. [Online]. Available: https://arxiv.org/abs/2005.07513

[28] H. Mossalam, Y. M. Assael, D. M. Roijers, and S. Whiteson, "Multi-objective deep reinforcement learning," *CoRR*, vol. abs/1610.02707, 2016. [Online]. Available: http://arxiv.org/abs/1610.02707

[29] D. M. Roijers, S. Whiteson, and F. A. Oliehoek, "Linear support for multi-objective coordination graphs," in *Proceedings of the 2014 International Conference on Autonomous Agents and Multi-Agent Systems*, ser. AAMAS '14. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2014, p. 1297–1304.

[30] M. Zuluaga, A. Krause, and M. Püschel, "e-pal: An active learning approach to the multi-objective optimization problem," *Journal of Machine Learning Research*, vol. 17, no. 104, pp. 1–32, 2016. [Online]. Available: http://jmlr.org/papers/v17/15-047.html

[31] K. V. Moffaert and A. Nowé, "Multi-objective reinforcement learning using sets of pareto dominating policies," *Journal of Machine Learning Research*, vol. 15, no. 107, pp. 3663–3692, 2014. [Online]. Available: http://jmlr.org/papers/v15/vanmoffaert14a.html

[32] R. Yang, X. Sun, and K. Narasimhan, "A generalized algorithm for multi-objective reinforcement learning and policy adaptation," in *Advances in Neural Information Processing Systems*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett,

Eds., vol. 32. Curran Associates, Inc., 2019. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2019/file/4a46fbfca3f1465a27b210f4bdfe6ab3-Paper.pdf

[33] M. Reymond and A. Nowe, "Pareto-dqn: Approximating the pareto front in complex multi-objective decision problems," in *Proceedings of the Adaptive and Learning Agents Workshop 2019 (ALA-19) at AAMAS*, May 2019, null ; Conference date: 13-05-2019 Through 14-05-2019. [Online]. Available: https://ala2019.vub.ac.be

[34] S. Parisi, M. Pirotta, and J. Peters, "Manifold-based multi-objective policy search with sample reuse," *Neurocomputing*, vol. 263, pp. 3–14, 2017, multiobjective Reinforcement Learning: Theory and Applications. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0925231217310986

[35] S. Parisi, M. Pirotta, and M. Restelli, "Multi-objective reinforcement learning through continuous pareto manifold approximation," *J. Artif. Intell. Res.*, vol. 57, pp. 187–227, 2016.

[36] J. Xu, Y. Tian, P. Ma, D. Rus, S. Sueda, and W. Matusik, "Prediction-guided multi-objective reinforcement learning for continuous robot control," in *Proceedings of the 37th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, H. D. III and A. Singh, Eds., vol. 119. PMLR, 13–18 Jul 2020, pp. 10 607–10 616. [Online]. Available: https://proceedings.mlr.press/v119/xu20h.html

[37] Y. Zhang and Q. Yang, "A survey on multi-task learning," *CoRR*, vol. abs/1707.08114, 2017. [Online]. Available: http://arxiv.org/abs/1707.08114

[38] A. Wilson, A. Fern, S. Ray, and P. Tadepalli, "Multi-task reinforcement learning: A hierarchical bayesian approach," ser. ICML '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 1015–1022. [Online]. Available: https://doi.org/10.1145/1273496.1273624

[39] H. Li, X. Liao, and L. Carin, "Multi-task reinforcement learning in partially observable stochastic environments." *Journal of Machine Learning Research*, vol. 10, no. 5, 2009.

[40] J. Andreas, D. Klein, and S. Levine, "Modular multitask reinforcement learning with policy sketches," in *Proceedings of the 34th International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, D. Precup and Y. W. Teh, Eds., vol. 70. PMLR, 06–11 Aug 2017, pp. 166–175. [Online]. Available: https://proceedings.mlr.press/v70/andreas17a.html

[41] A. A. Deshmukh, U. Dogan, and C. Scott, "Multi-task learning for contextual bandits," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, Eds., vol. 30. Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2017/file/b06f50d1f89bd8b2a0fb771c1a69c2b0-Paper.pdf

[42] T. Bräm, G. Brunner, O. Richter, and R. Wattenhofer, "Attentive multi-task deep reinforcement learning," in *Machine Learning and Knowledge Discovery in Databases -*

European Conference, ECML PKDD 2019, Würzburg, Germany, September 16-20, 2019, Proceedings, Part III, ser. Lecture Notes in Computer Science, U. Brefeld, É. Fromont, A. Hotho, A. J. Knobbe, M. H. Maathuis, and C. Robardet, Eds., vol. 11908. Springer, 2019, pp. 134–149. [Online]. Available: https://doi.org/10.1007/978-3-030-46133-1_9

[43] T.-L. Vuong, D.-V. Nguyen, T.-L. Nguyen, C.-M. Bui, H.-D. Kieu, V.-C. Ta, Q.-L. Tran, and T.-H. Le, "Sharing experience in multitask reinforcement learning," in *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*. International Joint Conferences on Artificial Intelligence Organization, 7 2019, pp. 3642–3648. [Online]. Available: https://doi.org/10.24963/ijcai.2019/505

[44] T. Shu, C. Xiong, and R. Socher, "Hierarchical and interpretable skill acquisition in multi-task reinforcement learning," 12 2017.

[45] D. Guo, B. A. Pires, B. Piot, J. bastien Grill, F. Altché, R. Munos, and M. G. Azar, "Bootstrap latent-predictive representations for multitask reinforcement learning," 2020.

[46] M. Hessel, H. Soyer, L. Espeholt, W. Czarnecki, S. Schmitt, and H. van Hasselt, "Multi-task deep reinforcement learning with popart," *CoRR*, vol. abs/1809.04474, 2018. [Online]. Available: http://arxiv.org/abs/1809.04474

[47] H. van Hasselt, A. Guez, M. Hessel, and D. Silver, "Learning functions across many orders of magnitudes," *CoRR*, vol. abs/1602.07714, 2016. [Online]. Available: http://arxiv.org/abs/1602.07714

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

[49] A. Machado, G. Ramalho, J.-D. Zucker, and A. Drogoul, "Multi-agent patrolling: An empirical analysis of alternative architectures," in *International workshop on multi-agent systems and agent-based simulation*. Springer, 2002, pp. 155–170.

[50] A. Almeida, G. Ramalho, H. Santana, P. Tedesco, T. Menezes, V. Corruble, and Y. Chevaleyre, *Recent Advances on Multi-agent Patrolling*, ser. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2004, vol. 3171, p. 474–483. [Online]. Available: http://link.springer.com/10.1007/978-3-540-28645-5_48

[51] J. T. i Viñals, *Introducción al aprendizaje por refuerzo profundo: Teoría y práctica en Python*, 1st ed. Independently published, 14 abril 2021.

[52] R. S. Sutton, "Learning to predict by the methods of temporal differences," *Machine learning*, vol. 3, no. 1, pp. 9–44, 1988.

[53] M. L. Littman, "Markov games as a framework for multi-agent reinforcement learning," in *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ser. ICML'94. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994, p. 157–163.

[54] L. Matignon, G. J. Laurent, and N. Le Fort-Piat, "Independent reinforcement learners in cooperative markov games: a survey regarding coordination problems," *The Knowledge Engineering Review*, vol. 27, no. 1, p. 1–31, 2012.

[55] N. Fulda and D. Ventura, "Predicting and preventing coordination problems in cooperative q-learning systems," in *Proceedings of the 20th International Joint Conference on Artifical Intelligence*, ser. IJCAI'07. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2007, p. 780–785.

[56] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0893608089900208

[57] R. E. Bellman and S. E. Dreyfus, *Applied Dynamic Programming*. Santa Monica, CA: RAND Corporation, 1962.

[58] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *nature*, vol. 518, no. 7540, pp. 529–533, 2015.

[59] Z. Wang, T. Schaul, M. Hessel, H. Hasselt, M. Lanctot, and N. Freitas, "Dueling network architectures for deep reinforcement learning," in *International conference on machine learning*. PMLR, 2016, pp. 1995–2003.

[60] C. Claus and C. Boutilier, "The dynamics of reinforcement learning in cooperative multiagent systems," ser. AAAI '98/IAAI '98. USA: American Association for Artificial Intelligence, 1998, p. 746–752.

[61] A. K. Agogino and K. Tumer, "Analyzing and visualizing multiagent rewards in dynamic and stochastic domains," *Autonomous Agents and Multi-Agent Systems*, vol. 17, no. 2, p. 320–338, oct 2008. [Online]. Available: https://doi.org/10.1007/s10458-008-9046-9

Glosario

ASV Vehículos Autónomos de Superficie. 81

CNN Convolutional Neural Network. 36

DL Deep Learning. 9

Double-DQN Double Deep Q-Network. 9–11, 41, 42

DQN Deep Q-Network. 9

DRL Deep Reinforcement Learning. 9, 10

Dueling-DQN Dueling Deep Q-Network. 9

MADRL Multiagent DRL. 10

MDP Proceso de decisión de Markov. 12, 25, 30

ML Machine Learning. 9

MORL Multi-Objective Reinforcement Learning. 11

MTRL Multi-Task Reinforcement Learning. 12, 13

RL Reinforcement Learning. 9, 11, 23, 25, 26, 28

TD Diferencia Temporal. 28

TSP Traveling Salesman Problem. 15