UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y
SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN DE CONTROLADORES DE ROBOTS SUBMARINOS MEDIANTE REDES NEURONALES ARTIFICIALES Y APRENDIZAJE POR REFUERZO

ALONSO PÉREZ JIMÉNEZ

2023

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y
SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DISEÑO E IMPLEMENTACIÓN DE CONTROLADORES DE ROBOTS SUBMARINOS MEDIANTE REDES NEURONALES ARTIFICIALES Y APRENDIZAJE POR REFUERZO

Autor
ALONSO PÉREZ JIMÉNEZ

Tutores

RAFAEL SENDRA ARRANZ
CÉSAR ORTIZ TORO

Ponente
ÁLVARO GUTIÉRREZ MARTÍN

2023

Resumen

La comunicación con vehículos subacuáticos ha sido un reto de ingeniería desde que se inventaron los submarinos. Este problema gana importancia cuando se trata de vehículos subacuáticos no tripulados, que tradicionalmente han sido remotamente controlados, llevando al desarrollo de sistemas de cableado para transmitir instrucciones y recibir información. Estos sistemas tienen limitaciones en alcance y suponen un alto coste de desarrollo, construcción y operación. Un vehículo autónomo subacuático (AUV) es un tipo de robot submarino diseñado para operar de manera autónoma en entornos acuáticos sin la necesidad de intervención humana directa. Esto soluciona el problema de la comunicación suprimiendo la necesidad de tener una comunicación constante con el AUV. Esta solución, a cambio de simplificar las operaciones, aumenta la complejidad de los controladores de los robots.

En este Trabajo de Fin de Grado se ha explorado el uso de redes neuronales artificiales para el desarrollo de controladores de AUVs mediante aprendizaje por refuerzo. El aprendizaje por refuerzo es una familia de algoritmos dedicados a entrenar agentes mediante recompensas dependientes de las acciones que realizan mientras exploran el entorno. El algoritmo concreto que se ha utilizado es Deep Q-Learning, una evolución de Q-Learning para su uso con redes neuronales artificiales.

La implementación del proyecto se ha realizado sobre el simulador del proyecto NAUTILUS, un proyecto conjunto de varias universidades para el desarrollo de enjambres de vehículos submarinos autónomos guiados por inteligencia artificial. Este simulador está desarrollado en Python, por lo que para la implementación de las redes neuronales se ha usado la librería Pytorch.

Se ha desarrollado un controlador con el fin de que el AUV esquive los obstáculos del entorno. La red neuronal se ha entrenado en el escenario con terreno más accidentado y luego se ha probado en otros tres escenarios nunca vistos durante el entrenamiento. Después de un análisis de los resultados, se ha verificado que los robots entrenados son capaces de resolver correctamente la tarea impuesta. Se ha comprobado que el controlador desarrollado es capaz de esquivar los objetos de los distintos escenarios a pesar de las diferencias en la forma y tamaño de estos.

Palabras clave: Vehículo submarino autónomo, aprendizaje por refuerzo, Deep Q-Learning, esquivador de obstáculos, redes neuronales artificiales.

Abstract

Communication with underwater vehicles has been an engineering challenge ever since submarines were invented. This problem becomes significant when it comes to unmanned underwater vehicles, which have traditionally been remotely controlled, leading to the development of wired systems that transmit instructions and receive information. These systems have limitations in terms of range and their high costs in terms of development, construction, and operation. An autonomous underwater vehicle (AUV) is a type of underwater robot designed to operate autonomously in aquatic environments without the need for direct human intervention. This solves the communication problem by eliminating the need for constant communication with the AUV. However, this solution, while simplifying operations, increases the complexity of robot controllers.

In this BSc Thesis, the use of artificial neural networks for the development of AUV controllers using reinforcement learning has been explored. Reinforcement learning is a family of algorithms dedicated to training agents through rewards dependent on the actions they take while exploring the environment. The specific algorithm used is deep Q-learning, an evolution of Q-learning for use with artificial neural networks.

The project implementation was carried out using the simulator from the NAUTILUS project, a joint project of several universities for the development of swarms of autonomous underwater vehicles guided by artificial intelligence. Since this simulator is developed in Python, the PyTorch library was used for the implementation of the neural networks.

A controller was developed to enable the AUV to evade environmental obstacles. The neural network was trained in the most rugged terrain scenario and then tested in three other scenarios that were never seen during training. After analyzing the results, it was verified that the trained robots are capable of correctly solving the given task. It was observed that the developed controller is capable of evading objects in different scenarios despite the differences in their shape and size.

Keywords: Autonomous underwater vehicle, reinforcement learning, deep Q-learning, obstacle avoidance, artificial neural networks.

Agradecimientos

A mis tutores Rafael Sendra Arranz y César Ortiz Toro por haberme guiado a lo largo de este trabajo y haber estado dispuestos a ayudarme y pendientes de mis avances en todo momento.

A Álvaro Gutiérrez Martín por hacer que me interesase por este campo gracias a su asignatura IRIN.

A mi familia y amigos por su apoyo y a Celia por llamar *piojos* a los robots.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción y objetivos

Los vehículos no tripulados han ganado popularidad como herramienta para solucionar problemas muy variados, como llevar ayuda humanitaria a lugares golpeados por desastres naturales o monitorizar a fauna salvaje [3]. Esto es debido a la versatilidad que tienen y a la relativa facilidad que existe para comunicarnos con ellos en el medio aéreo o terrestre, ya que podemos usar tecnologías como el tradicional mando radio control u otras mas avanzadas como las redes LTE [4]. Sin embargo, en el entorno subacuático, la comunicación y el control de robots presentan desafíos y limitaciones significativas debido a la dificultad de transmitir ondas electromagnéticas a través del agua. Esto ha dado lugar al uso de cables para controlar los robots, convirtiéndolos en robots subacuáticos controlados remotamente (ROUV por sus siglas en inglés), y con ello, al desarrollo de los Tether Management System [5]. Es decir, en vez de conectar el robot directamente al barco, se desciende una plataforma con el ROUV junto a una bobina con un cable relativamente ligero, y esta plataforma se conecta al barco mediante un cable más grueso. El uso de cables permite tener un gran ancho de banda en la comunicación y aumentar el tiempo de operación mandando energía a través del mismo. A su vez, esta solución también tiene algunas limitaciones importantes. Los cables pueden ser un peligro si otros objetos colisionan con ellos, aumentan mucho el peso y ocupan mucho espacio, es fácil que se enganchen o dañen con obstáculos y pueden ser muy caros.

Sin embargo, existe una alternativa para evitar cualquier tipo de conexión, los vehículos submarinos autónomos (AUV, Autonomous Underwater Vehicles). Un AUV es un tipo de robot submarino diseñado para operar de manera autónoma en entornos acuáticos sin la necesidad de intervención humana directa [6]. En este contexto, hace algunas décadas se planteo el uso de redes neuronales artificiales para controlarlos [7], debido a la complejidad de usar enfoques de programación tradicional. Las redes neuronales artificiales presentan ventajas como su capacidad para resolver problemas no lineales, realizar tareas en paralelo y tener una gran cantidad de variables de entrada. En este Trabajo de Fin de Grado, se explorará el uso de redes neuronales artificiales para abordar este desafío.

Para ello se usará el simulador que está siendo desarrollado como parte del proyecto Enjambres de vehículos submarinos autónomos guiados por Inteligencia

Artificial: su momento ha llegado (NAUTILUS)[1]. Este proyecto tiene como objetivo desarrollar enjambres de AUVs que funcionen de forma coordinad y descentralizada sin la interacción directa de humanos. El enjambre estará diseñado para navegar de forma autónoma y tener comportamientos colaborativos entre sus miembros, ya que funcionará como un sistema único y descentralizado donde cada AUV es guiado por inteligencia artificial.

Con el fin de optimizar los parámetros de las redes neuronales artificiales que definen el comportamiento de los robots, se usarán algoritmos de aprendizaje por refuerzo (RL). RL surgió como un enfoque para que agentes adquiriesen de manera autónoma comportamientos complejos a partir recompensas entregadas en función de sus interacciones con el entorno [8]. En este modelo los agentes son capaces de aprender en función de las recompensan que se que obtienen interaccionando con el entorno.

1.1. Objetivos y contribuciones

El propósito de este Trabajo de Fin de Grado consiste en el diseño e implementación de controladores mediante redes neuronales artificiales para manejar el comportamiento de los AUVs. Para cumplir este objetivo se usarán algoritmos de aprendizaje por refuerzo en el entrenamiento de las redes neuronales artificiales. El desarrollo de estos controladores se integrará dentro de un simulador de AUVs perteneciente al proyecto NAUTILUS. Se entrenará el controlador neuronal con el objetivo de que el AUV esquive los obstáculos del entorno. Se realizarán distintas pruebas en distintos escenarios del simulador para comprobar la eficacia de las redes neuronales artificiales como solución para los robots submarinos autónomos.

Para ello primero se realizará un estudio del estado del arte sobre controladores de robots optimizados mediante algoritmos de RL, consecuentemente se elegirá el algoritmo de RL que se utilizará para esta cuestión. Actualmente el simulador no soporta el uso de redes neuronales artificiales en los controladores de los AUVs. Se implementará la lógica necesaria para su uso como selector de la acción a realizar por el robot en función de los datos de los sensores. También se modificará el simulador para añadir las funcionalidades necesarias para realizar el proceso de entrenamiento de RL, como reiniciar los robots entre episodios o hacer auto-guardados durante los entrenamientos. Posteriormente, se procederá a la definición y validación experimental de controladores de evasión de colisiones en múltiples escenarios. Para ello se diseñará una función de recompensa a emplear para que los AUVs aprendan a realizar el objetivo que les hemos planteado, no colisionar con los elementos presentes en el escenario. Por último, se realizará el análisis y la validación del comportamiento de los robots en los experimentos realizados.

[1] Grant: PID2020-112502RB/AEI/10.13039/501100011033

1.2. Planificación del proyecto

Para cumplir los objetivos descritos anteriormente, este Trabajo de Fin de Grado se ha dividido en 5 capítulos:

- `Capítulo 1:` presenta los objetivos y motivaciones detrás del proyecto. A continuación se realiza un estudio sobre los logros obtenidos por trabajos similares.
- `Capítulo 2:` explica los orígenes y las bases del funcionamiento del análisis por refuerzo. Además, describe en detalle el algoritmo usado en este Trabajo de Fin de Grado, Deep Q-Learning.
- `Capítulo 3:` muestra las herramientas y tecnologías usadas para la implementación del proyecto así como el modelo del AUV que usa el simulador.
- `Capítulo 4:` se centra en el proceso de desarrollo del proyecto. Explica las distintas decisiones tomadas para su desarrollo así como el razonamiento detrás de ellas.
- `Capítulo 5:` muestra la evolución del proceso de entrenamiento de la red neuronal artificial. Además, analiza las trayectorias seguidas por el controlador neuronal en distintas situaciones y escenarios para validar su comportamiento.
- `Capítulo 5:` este último capítulo expone las conclusiones extraídas tras el análisis del capítulo anterior y propone lineas futuras para continuar el desarrollo de este Trabajo de Fin de Grado.

1.3. Estado del arte

El uso de redes neuronales artificiales para controlar AUVs no es reciente, hace varias décadas ya se empezó a plantear en la literatura [9] [10] [11]. Esto no es sorprendente, teniendo en cuenta tanto las ventajas de su uso, como su habilidad para resolver problemas no lineales, su estructura en paralelo o su capacidad para tener muchas variables como entrada [7]. No obstante, hay que tener en cuenta que en el caso de la robótica existe una dificultad añadida, hay que transferir las políticas aprendida por el agente en el simulador al entorno real, la ausencia de determinismo y a la existencia de parámetros que escapan a nuestro control complican enormemente la tarea.. En [12], se propone el enfoque de la Aleatorización de Dinámicas, donde se entrena al robot en una variedad de simulaciones con diferentes parámetros dinámicos aleatorizados. Esto ayuda al robot a adaptarse a las variaciones del mundo real, tal y como los resultados experimentales muestran. Se mostró una mejora significativa el rendimiento y la capacidad de adaptación del agente en comparación con los enfoques tradicionales.

Para entrenar estas redes neuronales artificiales, uno de los enfoques con más potencial en el campo de la robótica es el aprendizaje por refuerzo (RL) [13], una manera eficiente de adquirir datos y aprender habilidades para un agente. RL tiene la ventaja de que, con una recompensa apropiada, el robot puede aprender una estrategia compleja sin necesitar más datos sobre el entorno que los proporcionados por sus sensores. Este enfoque realmente surgió en el campo de la psicología [14], pero con la creación de las redes neuronales artificiales y el desarrollo de algoritmos cada vez más efectivos para aplicarlo, como actor-critic [15] o DQN [16], se ha convertido en una de las principales ramas de entrenamiento para este paradigma.

En el caso concreto de los AUVs se ha usado aprendizaje por refuerzo para entrenar redes neuronales capaces de guiarles en el cumplimiento de distintas tareas. Este enfoque cobra importancia al tener en cuenta la dificultad que supone que el medio acuático no sea lineal debido a fenómenos como las corrientes marinas. Dicha dificultad puede verse en la complejidad matemática a la que tuvieron que llegar en [17] y [18] para solucionar el problema de que un AUV, desde un punto cualquiera, se acercase y siguiese una trayectoria predefinida de forma precisa usando un enfoque más tradicional. En cambio, en [19] se aprovechó RL para entrenar redes neuronales que controlasen AUVs para realizar una tarea similar, seguir trayectorias manteniendo una profundidad constante respecto al fondo marino. Comparando ambos trabajos, se puede llegar a la conclusión de que aprovechar la habilidad de las redes neuronales para solucionar problemas no lineales lleva a una simplificación en la resolución de este tipo de tareas.

En [20], se utilizó RL y redes neuronales artificiales para dotar a los robots de la capacidad de explorar y buscar en aguas desconocidas de forma autónoma. En concreto se usó una variante del algoritmo de RL actor-critic [15]. Este algoritmo utiliza dos redes neuronales, una para elegir la acción y otra para evaluar y actualizar los valores estado-acción. El modelo actor-critic también se ha utilizado para entrenar AUVs en el seguimiento de trayectorias y esquivación de obstáculos [21], usando una evolución del algoritmo DDPG [22] que permite el uso de espacios de acción continuos.

Los autores de [23] propusieron un framework de control de movimiento de AUVs de extremo a extremo basado en el algoritmo PPO [24]. Tomando directamente la información de percepción del sonar original como entrada sin considerar las características dinámicas del AUV y con una función recompensa bien diseñada permite que el AUV evite colisiones de manera segura en un entorno submarino complejo. En [25], también se utilizó directamente la imagen de un sonar, bidimensional en este caso, como entrada de la red neuronal profunda, y la señal de control de la aleta como acción. Se propuso un algoritmo de evasión de colisiones basado en RL profundo que era capaz de completar las tareas de seguimiento de trayectoria y evasión de obstáculos al mismo tiempo.

Otro algoritmo de RL que se ha empleado para entrenar redes neuronales y conseguir que AUVs esquiven obstáculos es doble DQN [26]. Este es una extensión

de DQN desarrollada para abordar el problema de la sobreestimación de valores de DQN. En [27], se estudió el uso de doble DQN para calcular la trayectoria a seguir por un AUV para evitar los obstáculos del entorno usando los datos recogidos por un sonar activo. El resultado fue superior al de un algoritmo genético [28] y al de aprendizaje profundo [29], tanto en eficiencia en los entrenamientos como en efectividad a la hora de esquivar obstáculos en movimiento. Además el uso de DQN frente a los algoritmos genéticos o aprendizaje profundo conlleva la ventaja de tener la habilidad de poder continuar el entrenamiento una vez desplegado el robot en lo que se conoce como aprendizaje online.

En cuanto a los avances más actuales, en 2022 [30] se propuso combinar dos sistemas de activación de eventos junto aprendizaje por refuerzo, creando así un nuevo algoritmo ET-SAC (Event-Triggered Soft Actor–Critic), para crear un controlador de AUV que esquive obstáculos desconocidos (tanto estáticos como dinámicos). El primer sistema tiene unos estados que se activan en función de si el robot detecta obstáculos. El segundo sistema se usa para recompensar la correcta esquivación de obstáculos donde la recompensa se proporciona según si el AUV entra a una zona peligrosa cerca de los obstáculos o sale de ella. Se llegó a la conclusión de que este algoritmo es muy efectivo tras probarlo en entornos de simulación de 2 y 3 dimensiones.

Capítulo 2

Aprendizaje por refuerzo

El aprendizaje por refuerzo (RL) tiene sus bases en el área de la psicología [31]. B.F. Skinner [32], en la década de 1930, demostró que los animales pueden ser entrenados para realizar tareas complejas mediante mecanismos de refuerzo, como el suministro de comida cuando realizan la acción deseada. Desarrolló la idea del refuerzo positivo mediante el cual un animal o cualquier agente puede aprender a optimizar su comportamiento aprendiendo de sus experiencias previas.

En la década de 1950, Richard Bellman [33] formuló las ecuaciones de Bellman. Este conjunto de ecuaciones no lineales proporcionan al agente una política de comportamiento óptima. Esta política óptima la componen el conjunto de acciones que maximizan el valor esperado a lo largo de las interacciones con el entorno. En 1989 Christopher Watkins propuso Q-learning como parte de su tesis doctoral [34]. Watkins buscaba desarrollar un algoritmo de aprendizaje por refuerzo que pudiera aprender a tomar decisiones óptimas sin requerir un modelo explícito del entorno. El algoritmo se basaba en la idea de actualizar, usando las ecuaciones de Bellman, una función de valor $Q(s, a)$ para cada par estado-acción, representando la recompensa esperada al tomar una acción en un estado dado.

En 1999, Sutton [35] introduce la política de descenso del gradiente, uno de los pilares de RL. La política de descenso del gradiente se utiliza para actualizar los parámetros de una función de valor o una política con el objetivo de mejorar el desempeño del agente. La función objetivo puede ser la diferencia entre las recompensas esperadas y las recompensas obtenidas, o cualquier otra medida de rendimiento que se desee optimizar.

En esencia, el aprendizaje por refuerzo consiste en hacer que un agente aprenda qué hacer, cómo mapear situaciones en acciones, con el objetivo de maximizar una recompensa numérica. El aprendiz debe descubrir qué acciones producen la mayor recompensa mediante la interacción con el entorno. En los casos más interesantes y desafiantes, las acciones pueden afectar no sólo la recompensa inmediata, sino también la siguiente situación y, a través de ella, todas las recompensas siguientes. Estas dos características, búsqueda de prueba y error y la recompensa retrasada, son las dos características mas significativas de RL [8]. Hoy en día se aplica mediante el uso

de distitos algoritmos, como Q-Learning[34], SARSA[36], Actor-Critic[15] y Deep Q-Network (DQN)[16], que se utilizan en diferentes entornos y aplicaciones.

2.1. Elementos del aprendizaje por refuerzo

En RL, el foco principal está en un agente que realiza decisiones en un entorno. Para comprender mejor este proceso, hay tres elementos fundamentales con los que se trabaja: *estados*, *acciones* y *recompensas*. [8]

Estados: Los estados representan la situación o el contexto en el que se encuentra el agente en un momento dado. Pueden ser descripciones del entorno o información capturada mediante sensores. Por ejemplo, en un juego de ajedrez, el estado sería la posición de las piezas en el tablero y el turno del jugador.

Acciones: Las acciones son las elecciones que el agente puede realizar en un estado determinado, es decir, las decisiones que el agente puede tomar para interactuar con el entorno. Siguiendo el ejemplo del ajedrez, las acciones podrían ser los movimientos legales que el jugador puede hacer con cada una de sus piezas.

Recompensas: Las recompensas son señales numéricas que indican al agente cuán bien o mal está desempeñándose en un determinado estado y con una acción específica. Pueden ser positivas, negativas o neutrales, y sirven como guía para que el agente aprenda a tomar decisiones que maximicen las recompensas acumuladas a lo largo del tiempo y de esta forma optimizar la toma de decisiones para la tarea. En nuestro ejemplo del ajedrez podría ser 9 puntos por capturar una dama o 5 por una torre.

2.2. Q-Learning

Q-Learning es un algoritmo popular de RL que se utiliza para aprender una política óptima en entornos donde las acciones y estados están discretizados. El algoritmo se basa en estimar el valor de una acción en un estado dado, llamado el valor Q.

El algoritmo utiliza una tabla de valores Q, que se inicializa con valores arbitrarios o en ceros. A medida que el agente explora el entorno y recibe recompensas, actualiza los valores $Q(s, a)$ en función de la información recopilada. La idea principal es utilizar la ecuación de actualización de valores $Q(s, a)$, basada en las ecuaciones de Bellman[33], para mejorar progresivamente las estimaciones:

$$[\underbrace{Q_{new}(s,a)}_{\text{Nuevo Q-Value}} = Q(s,a) + \underbrace{\alpha}_{\text{Factor aprendizaje}}[\underbrace{R(s,a)}_{\text{Recompensa}} + \underbrace{\gamma}_{\text{Factor descuento}} \overbrace{\max Q'(s',a')}^{\text{Máxima recompensa predicha, dado el nuevo estado y las acciones posibles}} - Q(s,a)]$$

Q(s, a) Valor Q del par estado-acción actual. Representa la estimación del valor esperado acumulado a lo largo del tiempo al tomar la acción 'a' en el estado 's'.

Factor de aprendizaje α Factor que controla la rapidez con la que se actualizan los valores Q. Un valor más alto significa que las actualizaciones son más rápidas y tienen un mayor impacto en los valores Q. Un valor más bajo hace que las actualizaciones sean más lentas y se tenga más en cuenta la experiencia pasada.

Factor de descuento γ Factor que determina la importancia relativa de las recompensas futuras en comparación con las recompensas inmediatas. Un valor cercano a 1 indica que se da mayor importancia a las recompensas futuras, obteniendo un enfoque más a largo plazo. Un valor cercano a 0 da mayor importancia a las recompensas inmediatas, obteniendo un enfoque más a corto plazo. El factor de descuento se utiliza para variar la importancia del valor acumulado de las recompensas a lo largo del tiempo.

maxQ'(s',a') Valor Q máximo del próximo estado s al tomar todas posibles acciones a. Representa la mejor estimación del valor esperado acumulado a lo largo del tiempo desde el próximo estado s.

2.3. Deep Q-Learning

En el año 2013 DeepMind Technologies [16] presenta un modelo de red neuronal convolucional que, tras ser entrenada usando una variante de Q-Learning, es capaz de jugar a 7 juegos de la Atari2000 sin hacer cambios en su arquitectura o algoritmo de entrenamiento, superando en 6 juegos a los anteriores intentos y en 3 de ellos a humanos expertos. Esto se pudo lograr gracias a la combinación de Q-Learning con redes neuronales profundas, creando lo que hoy es conocido como Deep Q-Learning (DQN), donde los pesos de la red se ajustan utilizando un algoritmo de descenso de gradiente, minimizando el error entre los valores predichos y los valores reales obtenidos durante la interacción con el entorno.

En una tarea compleja, la tabla de valores Q puede llegar a ser inabarcable computacionalmente. Para abordar este problema de escalabilidad de Q-Learning, en lugar de mantener una tabla Q explícita, DQN utiliza una red neuronal para aprender una aproximación de la función Q. La red neuronal toma un estado como entrada y produce un vector de valores Q para todas las acciones posibles. En esencia, la red neuronal aprende a estimar los valores Q en función del estado de entrada.

El uso de redes neuronales en el DQN ofrece varias ventajas. Primero, permite manejar espacios de estados continuos o de muchas dimensiones. Además, la red neuronal puede generalizar el conocimiento aprendido a partir de ejemplos limitados y realizar interpolaciones entre estados similares. Por último, DQN puede aprender directamente a partir de datos de entrada en crudo, como píxeles de imágenes, lo que lo hace adecuado para tareas de aprendizaje en entornos visuales, como videojuegos.

2.3.1. Replay Memory

En lugar de utilizar las experiencias en el orden en que se producen, se almacenan en una memoria de tamaño limitado y se muestrean aleatoriamente durante el proceso de entrenamiento. La Replay Memory es la estructura de datos utilizada para almacenar y organizar las experiencias pasadas del agente en tuplas de estado, acción, recompensa y próximo estado.

Este planteamiento se basa en la idea de que las experiencias consecutivas en el tiempo pueden estar altamente correlacionadas. Esto puede llevar a un aprendizaje ineficiente y a la sobrevaloración de ciertas acciones. La replay memory permite romper esta correlación temporal seleccionando las muestras de experiencia de manera aleatoria en vez de utilizarlas inmediatamente después de ser experimentadas. Esto aumenta la estabilidad del entrenamiento porque suaviza la distribución de las muestras. Además, permite al algoritmo ser más eficiente computacionalmente porque en lugar de realizar actualizaciones de la red neuronal en cada paso de tiempo, se realizan actualizaciones en lotes.

2.3.2. Descripción del algoritmo

En el Algoritmo 1 se muestran los principales pasos de DQN propuestos por DeepMind. Al comenzar se inicializa la red neuronal con pesos aleatorios y la replay memory. Posteriormente se realiza un bucle. En primer lugar se selecciona una acción. En función de una probabilidad ϵ esta será aleatoria o la predicha por la red neuronal. A continuación se ejecuta dicha acción en el simulador y se observa la recompensa obtenida y el estado siguiente. Este conjunto de datos se almacena como una nueva entrada en la replay memory para, por último realizar el descenso de gradiente usando muestras aleatorias de la memoria.

Algorithm 1 Deep Q-learning con Experience Replay [16]

Inicializar replay memory $\mathcal{D}$ con capacidad N
Inicializar función acción-valor $\mathcal{Q}$ con pesos aleatorios
for episodio = 1, M **do**
  Inicializar secuencia $s_1 = \{x_1\}$ y secuencia preprocesada $\phi_1 = \phi(s_1)$
  for $t = 1, T$ **do**
    Con probabilidad ϵ seleccionar una acción aleatoria a_t
    de otro modo seleccionar $a_t = \max_a \mathcal{Q}^*(\phi(s_t).a; \theta)$
    Ejecutar acción a_t en el simulador y observar la recompensa r_t y la imagen x_{t+1}
    Poner $s_{t+1} = s_t, a_t, x_{t+1}$ y preprocesar $\phi_{t+1} = \phi(s_{t+1})$
    Almacenar transición $(\phi_t, a_t, r_t, \phi_{t+1})$ en $\mathcal{D}$
    Poner $y_j = \begin{cases} r_j & \text{para} \phi_{j+1} \text{terminal} \\ r_j + \gamma \max_{a'} \mathcal{Q}(\phi_{j+1}, a'; \theta) & \text{para} \phi_{j+1} \text{no terminal} \end{cases}$
    Hacer descenso de gradiente en $(y_j - \mathcal{Q}(\phi_j, a_j; \theta))^2$
  end for
end for

Capítulo 3

Materiales y métodos

En este capítulo se exponen los métodos y herramientas usados para llevar acabo este Trabajo de Fin de Grado. Principalmente consta de tres elementos. Por un lado, el robot que se ha usado como base de la simulación debido al que es el usado por el proyecto NAUTILUS. Por otra parte, el simulador donde se ha llevado acabo la implementación del proyecto. Por último, un breve resumen sobre la librería empleada para el uso de redes neuronales y los algoritmos necesarios para su entrenamiento.

3.1. BlueROV2

El BlueROV2 [37] es el robot utilizado en el proyecto Nautilus y el cual usaremos en el simulador de este TFG. Fue diseñado por BlueRobotics con una arquitectura de código abierto que permite la fácil incorporación de nuevos sensores. Es un vehículo operado remotamente (ROV) que cuenta con la capacidad de ser controlado de forma remota a través de una conexión por cable con el fin de ser pilotado desde un barco acompañante. En la Figura 3.1 se pueden ver dos imágenes de las dos configuraciones de serie del BlueROV2.

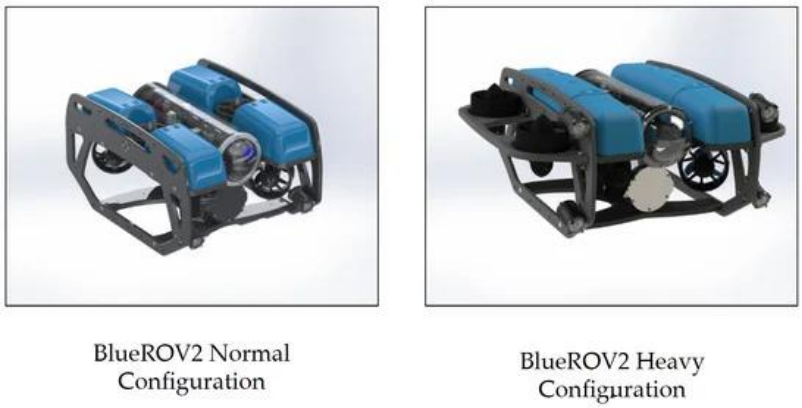


Figura 3.1: Imágenes de las dos posibles configuraciones del BlueROV2 [1].

El ROV tiene varios sensores integrados, que incluyen un transmisor de presión, sensores de detección de fugas, un módulo de detección de potencia, una unidad de

medición inercial (IMU) y un magnetómetro. La IMU consta de un acelerómetro y un giroscopio. Además, la arquitectura de software de código abierto permite al usuario agregar sensores adicionales. Su controlador de abordo es una Raspberry Pi 3 que se puede conectar por ethernet a un ordenador.

El robot, de casi 10 kilogramos, es capaz de sumergirse hasta 300 metros en su configuración de aluminio y desplazarse frontalmente hasta a 1,5 metros por segundo. Cuenta con una batería de 18Ah que le permite hacer misiones de unas dos horas con un uso normal, aparte de ser capaz de ser alimentado mediante el cable ethernet. También dispone de una cámara 1080p y 2 o 4 luces de 1500 lúmenes.

3.1.1. Propulsores

El BlueROV2 usa como propulsores el modelo T200, un propulsor submarino patentado por la misma empresa. Sus motores funcionan a 16 voltios y, a su potencia máxima de 390W, son capaces de generar 40N de empuje. La disposición de dichos propulsores depende de la versión del robot, como podemos ver en la Figura 3.2. El modelo estándar cuenta con 6 propulsores y será el que utilicemos para la simulación, no obstante existe la posibilidad de una configuración de 8 propulsores. Lo cual añade al robot la posibilidad de rotar en cabeceo.

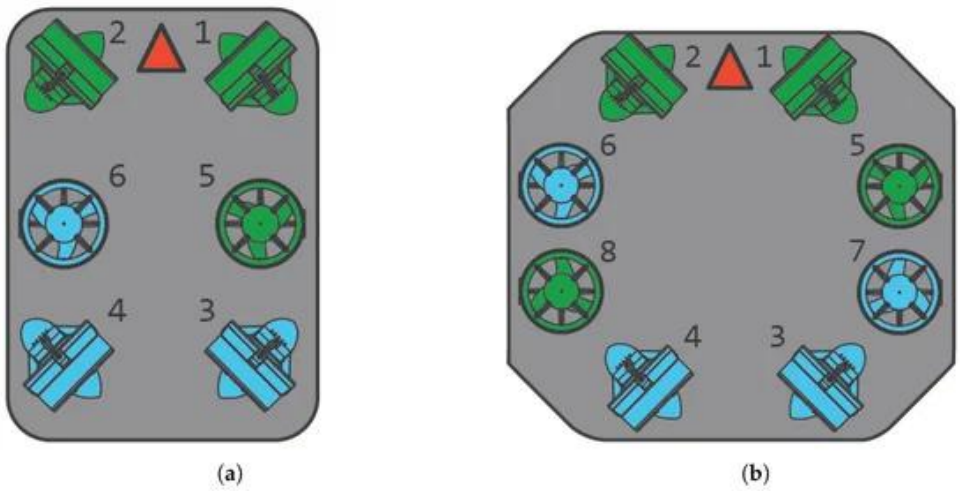


Figura 3.2: Comparación de las dos posibles configuraciones de propulsores. Los propulsores azules giran en sentido dextrógiro y los verdes en levógiro. La flecha roja es la dirección de avance frontal.(a) BlueROV2 configuración normal. (b) BlueROV2 configuración pesada. Imagen extraída de [1].

3.2. Simulador

La experimentación con AUVs en entornos reales es muy costosa por diferentes motivos. No solo se corre el riesgo de perder el robot bajo el agua o que sus componentes electrónicos se cortocircuiten por la entrada de agua por una grieta creada por algún golpe, sino que además es necesario transportar el material a un pantano o piscina profunda donde esté permitido. Por ello dentro del proyecto NAUTILUS se está desarrollando un simulador para poder llevar a cabo todos los experimentos iniciales de forma sencilla y poco costosa. El simulador implementa un entorno tridimensional para el desarrollo de las pruebas junto con la implementación de las dinámicas físicas de un cuerpo sumergido en movimiento presentadas en la Sección 3.2.1

El simulador está desarrollado en Python, un popular lenguaje de programación de alto nivel que, además, tiene dos importantes librerías para inteligencia artifical, Tensorflow [38] y PyTorch [39]. PyTorch será la que se usará para este Trabajo de Fin de Grado. El motor gráfico usado en el simulador es Panda3D, un framework de código abierto para renderizado 3D y videojuegos.

3.2.1. Modelo físico

El comportamiento del robot es descrito por su estado, el cual está definido por su dinámica, cinemática y el estado de sus motores. Para describirlo usaremos la notación de la Figura 3.3

En cuanto a la cinemática, el BlueROV2 puede moverse a lo largo de los ejes X_b, Y_b y Z_b, resultando en las velocidades lineales correspondientes u, v, w. En las rotaciones despreciaremos los giros entorno a los ejes X_b e Y_b. El movimiento en torno al eje Z se traduce en cinemática en $\dot{x}, \dot{y}$ y $\dot{z}$ como se muestra a continuación:

$$\begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T = R(\psi) \cdot \begin{bmatrix} u & v & w \end{bmatrix}^T \tag{3.1}$$

donde $R(\psi)$ es la matriz de rotación.

Para el caso de la dinámica, el objetivo es conocer el valor de $\dot{v}$, el vector de aceleraciones del robot $\begin{bmatrix} \dot{u} & \dot{v} & \dot{w} & \dot{p} & \dot{q} & \dot{r} \end{bmatrix}^T$. Despreciando las perturbaciones medioambientales y el vector de control del lastre, podemos simplificar y despejar las ecuaciones de Fossen [2] en la siguiente ecuación:

$$\dot{v} = M^{-1}(F - C(v)v) \tag{3.2}$$

donde $C(v)$ es la matriz de aceleración centrípeta y de Coriolis, y F es el vector que acumula todas las fuerzas y momentos excepto las de Coriolis. Es decir, $F = \begin{bmatrix} X, Y, Z, K, M, N \end{bmatrix}^T$, siendo X, Y y Z las fuerzas y K, M y N los momentos

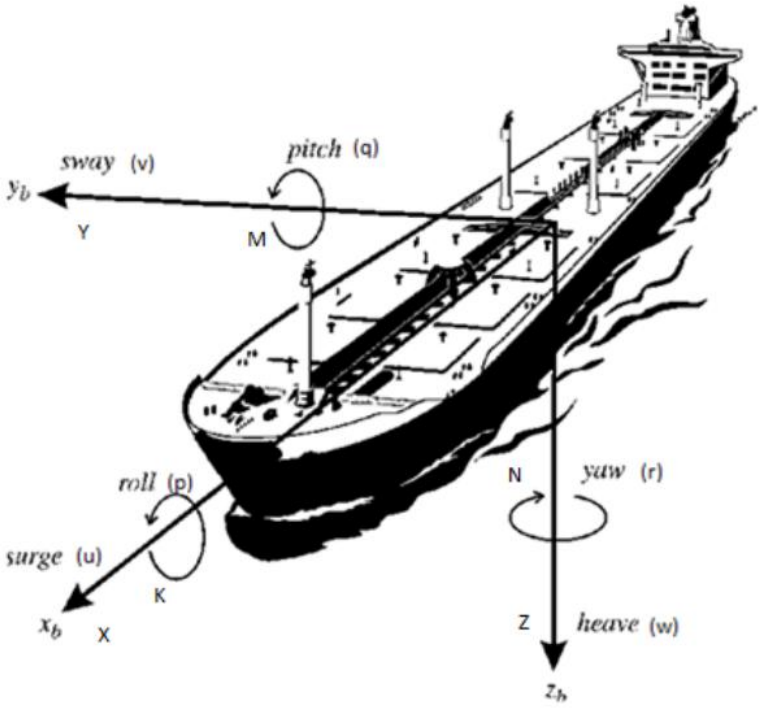


Figura 3.3: Posiciones, velocidades, fuerzas y momentos utilizados Imagen extraida de [2].

producidos en los ejes X_b, Y_b y Z_b respectivamente. A su vez, F se puede descomponer en tres fuerzas de acuerdo a la siguiente ecuación:

$$F = F_{motor} + F_g + F_D \tag{3.3}$$

F_{motor} son las fuerzas y momentos generados por la rotación de los propulsores. Para calcularlas llamaremos a F_i la fuerza generada por el motor i. Haciendo la aproximación de que la fuerza del motor coincide con su estado, usando una dinámica de primer orden, una constante de tiempo de 0,5s y despreciando la velocidad relativa del rotor con el agua, la fuerza de un motor viene dada por la ecuación:

$$F_i = estado_{motor}(i) = -2 \cdot estado_{motor}(i) + control(i) \tag{3.4}$$

Considerando las orientaciones de los propulsores en la Figura 3.2, las componentes de F_{motor} se definen de acuerdo a la siguiente ecuación:

$$\begin{aligned} X_{motor} &= \tfrac{\sqrt{2}}{2}(F_1 + F_2 + F_3 + F_4) \\ Y_{motor} &= \tfrac{\sqrt{2}}{2}(F_1 - F_2 - F_3 + F_4) \\ Z_{motor} &= F_5 + F_6 \\ K_{motor} &= 0{,}1(-F_5 + F_6) \\ M_{motor} &= 0 \\ N_{motor} &= 0{,}1\sqrt{2}(F_1 - F_2 + F_3 - F_4) \end{aligned} \tag{3.5}$$

Por otra parte, F_g engloba a las fuerzas y momentos resultantes de empuje y peso. Acorde a los datos del fabricante, la flotación total es de 2N y el peso muerto del

vehículo son 100N. Desconocemos el centro de flotación por lo que se asume el caso mas sencillo en el cual tendríamos una fuerza centrada en el eje Z_b y sendos momentos estabilizantes en los ejes X_b e Y_b. Estos últimos serán proporcionales y opuestos al cabeceo y balanceo. Las componentes resultado se agrupan en la siguiente ecuación:

$$\begin{aligned} X_g &= 0 \\ Y_g &= 0 \\ Z_g &= \begin{cases} -2 & \text{si } z \geq 0 \\ \frac{(100+2)z}{-altura_{BlueROV2}} & \text{si } 0 > z > -altura_{BlueROV2} \\ 100 & \text{si } z < -altura_{BlueROV2} \end{cases} \\ K_g &= -roll \\ M_g &= -pitch \\ N_g &= 0 \end{aligned} \tag{3.6}$$

Por último, para las fuerzas de rozamiento con el agua F_D se van a estimar todas las constantes a 1 debido a la a falta de conocimiento sobre los coeficientes hidrodinámicos del BlueROV2. En la Ecuación 3.7 se muestran las componentes de F_D .

$$\begin{aligned} X_g &= -u & K_g &= -p \\ Y_g &= -v & M_g &= -q \\ Z_g &= -w & N_g &= -r \end{aligned} \tag{3.7}$$

Se supone un modelo isótropo con masa y con masa añadida diagonal e igual a 15kg, 10kg más un 50 % extra por el efecto del agua. El resultado es la ecuación a continuación.

$$\begin{aligned} m_u &= m_v = m_w = 15 \\ m_{uv} &= m_u - m_v = 0 \end{aligned} \tag{3.8}$$

El momento de inercia también es desconocido por lo que se aproximará al de un cubo de 0.2m y 15kg, resultando la inercia mostrada a continuación:

$$\begin{aligned} I_x &= I_y = I_z = \frac{\text{masa}\cdot(0{,}2^2+0{,}2^2)}{12} \\ m_r &= I_z \end{aligned} \tag{3.9}$$

Recopilando todo esto se puede expresar la dinámica en términos de $\dot{u}$, $\dot{v}$, $\dot{w}$, $\dot{p}$, $\dot{q}$ y $\dot{r}$. No obstante, para obtener la dinámica del estado completo es necesario añadir la guiada en la rotación, asumiendo el cabeceo y balanceo despreciables. Con estas

consideraciones el resultado es el siguiente conjunto de ecuaciones:

$$\begin{aligned}
\dot{u} &= \frac{X+m_v vr}{m_u} \\
\dot{v} &= \frac{y+m_u ur}{m_v} \\
\dot{w} &= \frac{Z}{m_z} \\
\dot{r} &= \frac{N+m_{uv} uv}{m_r} \\
\dot{p} &= \frac{K}{I_x} \\
\dot{q} &= \frac{M}{I_y} \\
\dot{x} &= u \cdot cos(\psi) - v \cdot sin \\
\dot{y} &= u \cdot sin(\psi) + v \cdot cos(\psi) \\
\dot{z} &= w
\end{aligned} \tag{3.10}$$

Donde los términos $m_v vr$, $m_u ur$ y $m_{uv} uv$ corresponden a la aceleración de Coriolis.

3.2.2. Sensores y actuadores

Los sensores son la única forma que el robot tiene para recibir información del entorno. El simulador facilita la implementación de diferentes tipos de sensores, ofreciendo, en el momento de realizar este trabajo, diversos modelos de sonar, un radar, un GPS, un medidor de profundidad y un sensor virtual (sin equivalente en el AUV) de colisiones. En este Trabajo de Fin de Grado usaremos únicamente el radar para dar información al controlador, el GPS y el sensor de colisiones (usado en la función de recompensa del entrenamiento de RL). Actualmente solo hay un actuador en el simulador, dedicado a modificar la potencia, de forma individual, de cada uno de los motores.

El sensor principal que usaremos será el radar. Este sensor simula un radar de barrido mediante la toma de de lecturas de forma secuencial en 8 direcciones equiespaciadas en el plano x-y del AUV como se muestra en la Figura 3.4. El simulador comprueba a que distancia está el objeto mas cercano en esa dirección y la normaliza entre 0 y 1 si el valor de la medida es inferior a un umbral predefinido. En este Trabajo de Fin de Grado se ha definido un umbral de 8 metros, por lo que cualquier objeto a más de 8 metros se interpretará como un 0 en la lectura del radar. Los sensores están simulados de forma realista por lo que, en el caso del radar, existe un retardo en las actualizaciones de cada dirección igual al tiempo de barrido del mismo. En la Figura 3.4 se representa el funcionamiento del radar. Las líneas son las medidas que toma mientras que el área azul es la zona ciega del sensor.

Para el entrenamiento usaremos también los sensores GPS y de colisiones. El primero es la forma que tiene el controlador de acceder a la posición global del BlueROV2 en el simulador, ya que debido a a la arquitectura sensor-controlador-actuador este no tiene acceso directo a ella. El segundo comprueba si el volumen de colisión del robot interseccionа con otros objetos y, en caso de hacerlo, indica en que punto se está produciendo. De esta forma, es posible saber si el robot se ha chocado con un elemento del escenario o con otro robot.

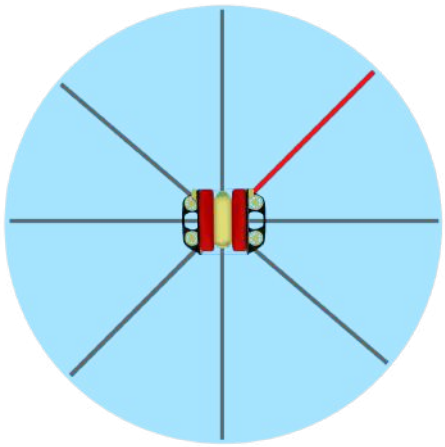


Figura 3.4: Representación del funcionamiento del radar.

3.2.3. Controlador y flujo de la simulación

Uno de los conceptos fundamentales en el simulador es la estructura sensor-controlador-actuador. Esta estructura ofrece una representación fiel de cómo funcionan los robots en la vida real, es decir, aislados completamente de su entorno. En este contexto, los robots solo reciben estímulos de entrada, que luego son evaluados para generar una respuesta de salida.

Cada robot incluye un controlador que define su funcionamiento. El controlador solo puede obtener información a través de los sensores del robot y generar respuestas a través de los actuadores que controlan los motores. El controlador es el encargado de procesar las medidas de los sensores y actualizar los actuadores. En este TFG se ha diseñado e implementado un controlador neuronal. Es decir, una red neuronal es la encargada de elegir la acción a realizar en base a la medida de los sensores. La Figura 3.5 muestra un diagrama de la arquitectura sensor-controlador-actuador.

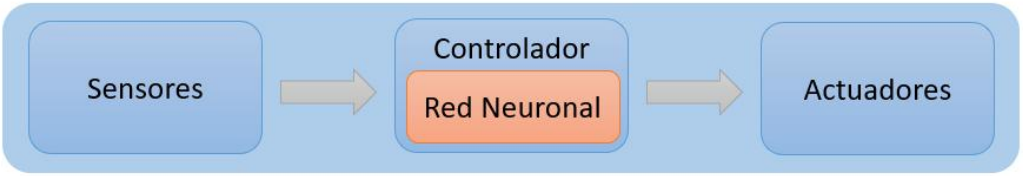
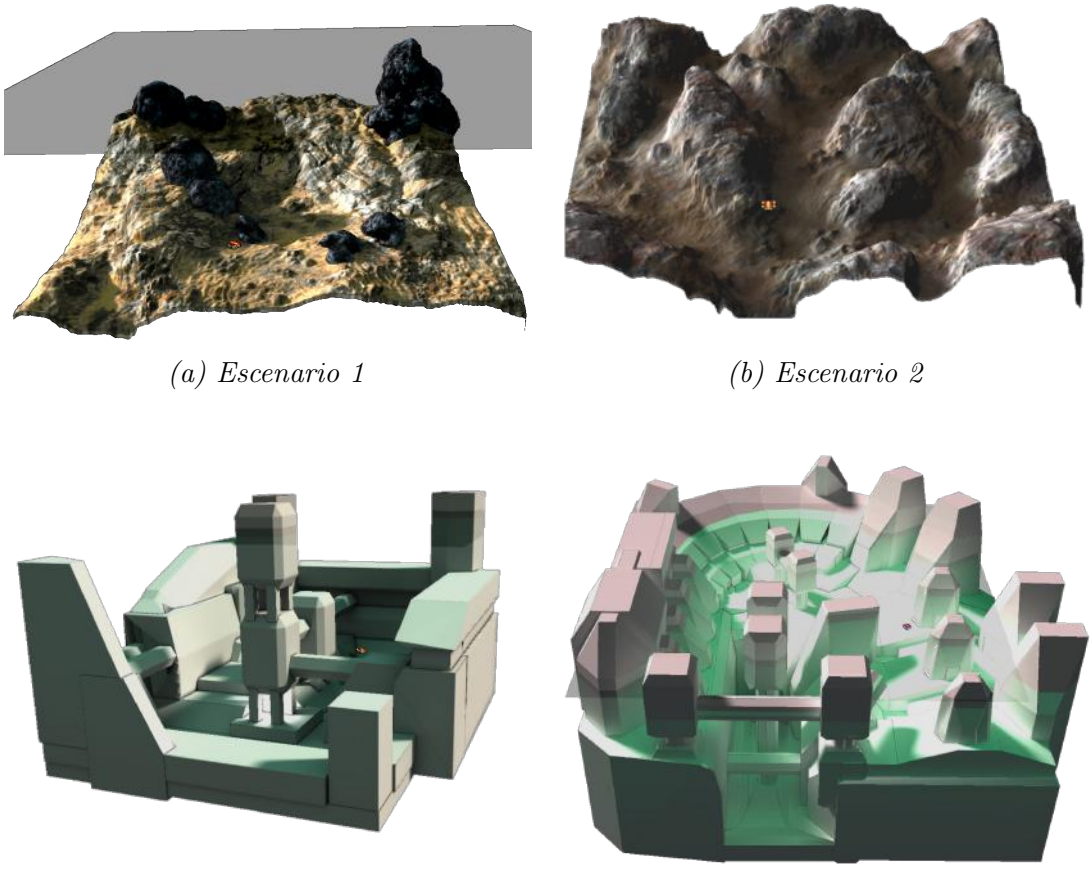


Figura 3.5: Arquitectura sensor-controlador-actuador.

En cada ciclo de la simulación, el controlador lee las medidas de los sensores y actualiza los actuadores. Tras esto, el simulador evalúa el estado actual del robot y aplica las dinámicas correspondientes, tomando en cuenta la influencia del entorno acuático. Este proceso se repite en cada iteración del simulador, con un período de 10ms (o lo que es equivalente, una tasa de actualización de 100 Hz).

3.2.4. Escenarios

Actualmente el simulador cuenta con cuatro escenarios distintos, mostrados en la Figura 3.6. El primero consiste en una plataforma irregular inclinada con rocas de formas esféricas. El segundo es una formación rocosa de suelo marino, entre las colinas de piedras se forman pasajes por las que el BlueROV2 tendrá que navegar. Este escenario, al ser el más accidentado, será el que se emplee para los entrenamientos. El tercer y cuarto son construcciones artificiales sumergidas donde los obstáculos son, principalmente, paredes lisas y esquinas de 90º.



(a) Escenario 1 *(b) Escenario 2*

(c) Escenario 3 *(d) Escenario 4*

Figura 3.6: Imágenes de los escenarios del simulador. *(a)* y *(b)* Escenarios con suelo marino rocoso. *(c)* y *(d)* Escenarios de construcciones artificiales sumergidas.

3.3. PyTorch

PyTorch [39] es una librería de tensores optimizada para deep learning. Es desarrollada por el Facebook's AI Research Lab y es de código abierto. Su distribución mas usada es la de Python pero también esta disponible para C++ y Java. Una de las características que la hace muy buena opción es su capacidad para utilizar tanto GPUs como CPUs para el procesamiento de los tensores. Además, al ser una de las librerías más populares, existe una gran cantidad de documentación y ejemplos de la comunidad lo cual facilitará la implementación en el simulador.

Pytorch se diseñó bajo la premisa de hacer la escritura de modelos y optimizadores sea lo más fácil y productiva posible para los desarrolladores. La complejidad inherente del aprendizaje automático es manejada internamente por la librería, mientras que el desarrollador emplea interfaces intuitivas. Este planteamiento, a su vez, evita problemas de rendimiento inesperados y permite que la mayoría de la implementación de Pytorch este escrita en C++[40] permitiendo así tener una alta eficiencia computacional. [39]

En el caso específico de RL, tiene una serie de módulos que permiten su sencilla implementación. El módulo *torch.nn* proporciona clases y funciones para la construcción de redes neuronales. Además, permite utilizar capas predefinidas como capas lineales, convolucionales o recurrentes. También contiene las funciones de pérdida como MSEloss o L1loss. Por otro lado, *Torch.optim* es el paquete que contiene los optimizadores, como Adam o SGD. Para crearlos es tan sencillo como construirlos con los hyperparametros de entrenamiento. Por último, el módulo *torch.nn.functional* contiene las funciones de activación para las neuronas de la red neuronal, como RELU o sigmoide.

Capítulo 4

Implementación

En este capítulo se abordará el desarrollo seguido para implementar aprendizaje por refuerzo en el simulador, con el fin desarrollar los entrenamientos de la red neuronal del controlador. Para ello, primero trataremos la integración de las redes neuronales creadas con Pytorch en los controladores de los AUVs, teniendo en cuenta que debe existir la posibilidad de ejecutar sistemas AUV multi-agente, tanto en simulación como en entrenamientos. En segundo lugar, se describirá el bucle de entrenamiento para el aprendizaje de las redes neuronales. Posteriormente, se presentará la arquitectura de la red neuronal y, por último, se describirá la función de recompensa empleada para los entrenamientos.

4.1. Integración con el controlador

Para la correcta implementación de controladores robóticos neuronales, se debe considerar la posibilidad de simular y entrenar varios robots simultáneamente. En el caso de una red neuronal estática, como un perceptrón [41], usar la misma instancia de la red para todos los AUVs no supone a ningún problema ya que se trata de un sistema estacionario. Sin embargo, en el caso de las redes neuronales recurrentes, ampliamente empleadas modelos de lenguaje [42], es necesario una red neuronal independiente para cada robot debido a que cada una tendrá su propia "memoria".
Por lo tanto, se debe crear una red neuronal sobre la que se aplicará el proceso de optimización, será sobre la que se realice el descenso de gradiente y no se usará para elegir movimientos de los AUVs. Al controlador de cada robot le asignaremos un duplicado de esta. La implementación de esta característica en Python se realiza mediante una copia profunda, es decir, un copia no solo del objeto principal, sino que también se duplican los objetos heredados. Al final de cada episodio se actualizarán las redes neuronales de todos los controladores.

4.2. Bucle de entrenamiento

El bucle de entrenamiento desarrollado es una adaptación del Algoritmo 1 al simulador NAUTILUS. Para poder ejecutarlo, primero debemos configurar el simulador para el entrenamiento. Para ello se ejecuta el entorno gráfico y se inicializa la red neuronal que usaremos como base con pesos aleatorios, así como la replay memory $\mathcal{D}$ con la asignada. A continuación, creamos los robots y les asociamos a cada uno un duplicado de la red neuronal base. Con esto ya tenemos listo el entorno para entrenar.

El bucle de entrenamiento está formado por un periodo de exploración de los robots, durante el cual son controlados por la red neuronal y almacenan sus transiciones $< \phi_t, a_t, r_t, \phi_{t+1} >$ en $\mathcal{D}$. A cada uno de estos periodos los llamaremos episodios y terminan cuando la simulación llega a 30 segundos de límite de tiempo o todos los robots se estrellan. Cuando un robot colisiona, queda desactivado hasta que termine el periodo de entrenamiento. Posteriormente, se ejecuta el algoritmo de descenso de gradiente sobre la red neuronal del entrenador. A continuación, se resetean la posiciones y orientaciones de los robots, devolviéndolos a sus condiciones iniciales. Además, se hace un duplicado nuevo de la red neuronal actualizada para cada controlador. En la Figura 4.1 se muestra un esquema de este proceso.

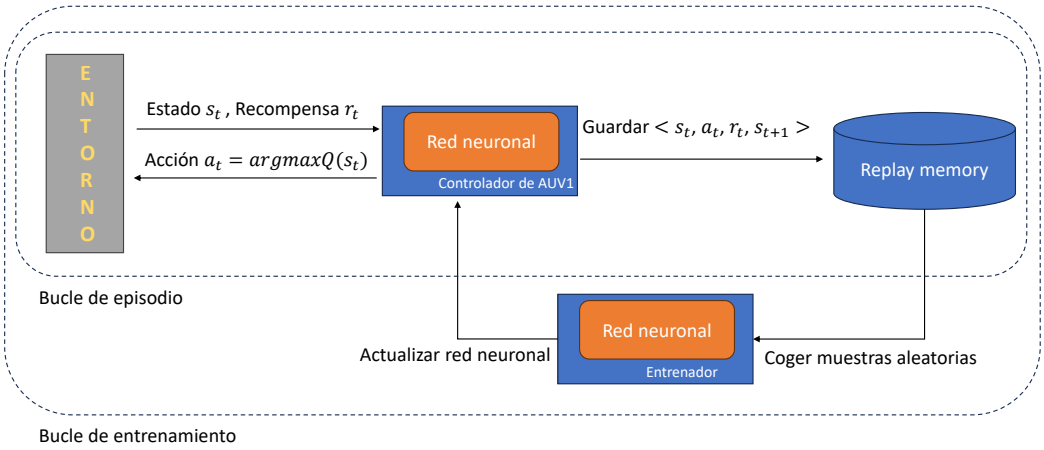


Figura 4.1: Diagrama del bucle de entrenamiento

4.3. La red neuronal artificial

La entrada de la red neuronal, la cual coincide con salida del radar, no requiere ningún preprocesamiento porque la salida del sensor está normalizada entre 0 y 1. Por tanto, la red neuronal recibe un vector de 8 dimensiones, una por cada dirección del radar representada en la Figura 3.4. La toma de las 8 medidas del radar no es inmediata, cada 10ms se actualiza una de las direcciones. El controlador solo hará predicciones cada vez que el radar complete una vuelta. Esto permite aligerar la carga computacional y que los cambios de estado sean representados de forma más precisa en la Replay Memory.

La salida de la red neuronal es un vector de 3 dimensiones en la que el valor más alto determina la acción a ejecutar. Estas acciones discretas se codifican en un conjunto de 4 voltajes, uno para cada uno de los motores de los propulsores que permiten al robot desplazarse de manera horizontal (los propulsores *1,2,3* y *4* de la Figura 3.2). Las tres acciones posibles son avanzar de frente, rotar a la izquierda y rotar a la derecha. Los motores de los propulsores verticales estarán fijados al valor necesario para que el AUV mantenga flotabilidad neutra.

Como se pude observar en la Figura 4.2, la red neuronal empleada es estática y tiene dos capas ocultas de 20 y 8 neuronas. Las funciones de activación de las neuronas de todas las capas ocultas y de salida son tangentes hiperbólicas.

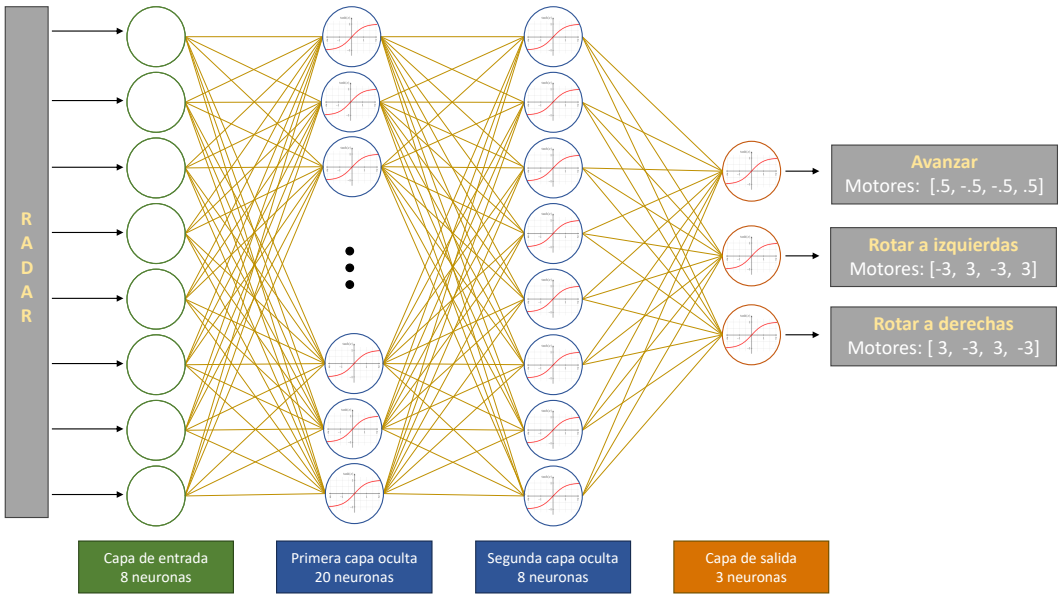


Figura 4.2: Representación esquemática de la arquitectura de la red neuronal.

4.4. Función de recompensa

La Ecuación 4.1 muestra la función de recompensa básica usada para guiar el aprendizaje por refuerzo. El primer término se corresponde con la velocidad del robot, medida como la distancia euclídea $||GPS(t) - GPS(t-1)||_2$ entre las dos últimas medidas del sensor GPS, las cuales se toman con un periodo de 10ms. Experimentalmente este valor no supera los 0.035 por lo que al multiplicarlo por 30 queda acotado entre 0 y 1. El segundo término de la ecuación es la distancia al obstáculo más cercano, calculada como el valor más alto del array de 8 posiciones que devuelve el sensor radar para cada una de las direcciones en las que mide la distancia al obstáculo más cercano, elevada al cubo y multiplicada por 3.

$$R_t = 30 \cdot \text{dist}(GPS_t, GPS_{t-1}) - 3 \cdot \text{máx}(radar)^3 \tag{4.1}$$

Como se puede ver en la Figura 4.3, elevar al cubo permite que al principio el valor sea muy pequeño, un umbral en el que el robot puede acercarse al objeto sin que la función de recompensa se vea significativamente afectada. Las medidas del radar están normalizadas entre 0 y 1, pero a partir de 0.9 el robot está colisionando. Por ello se multiplica por 3 el anterior valor, para conseguir que el valor negativo esté acotado entre 0 y -2. De esta forma la función de recompensa será negativa si el AUV se aproxima demasiado a un objeto independientemente de la velocidad a la que se este desplazando.

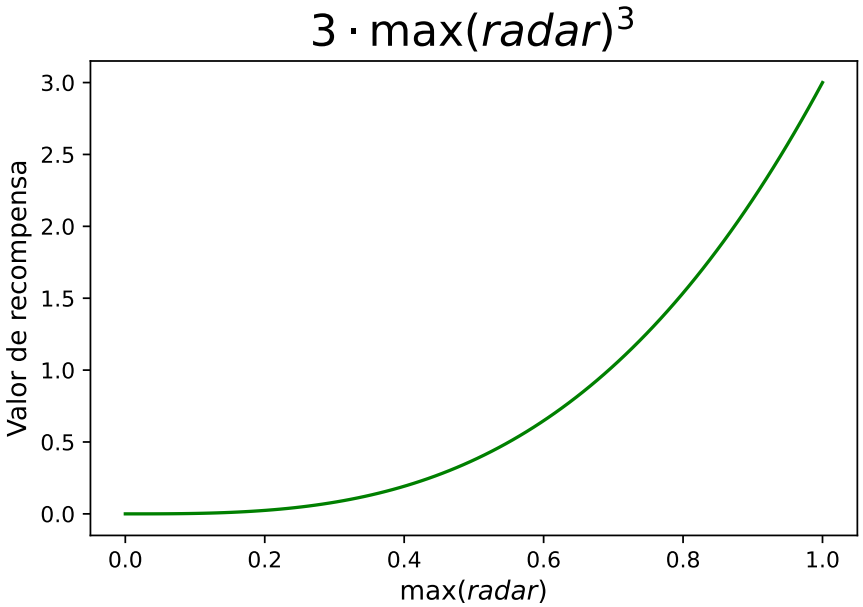


Figura 4.3: Forma de la función $3x^3$

La función de recompensa completa, descrita en el Algoritmo 2, está compuesta por tres partes. La primera parte, si el robot colisiona con algún objeto del escenario, la recompensa será -10. El segundo tramo, cuando el radar no detecta nada, es decir, cuando todas sus medidas sean 0, si el AUV no avanza la recompensa será -5. Por último, cuando no se de ninguna de las dos condiciones anteriores, la recompensa se calculará aplicando la Ecuación 4.1.

Algorithm 2 Algoritmo de la función de recompensa

$recompensa = 0$
if $colision$ **then**
  $recompensa = -10$
else if $Max(radar) = 0$ & $accion \neq avanzar$ **then**
  $recompensa = -5$
else
  $recompensa = Distancia(GPS_t, GPS_{t-1}) \cdot 30 - Max(radar)^3 \cdot 3$
end if

4.5. Hiperparámetros de entrenamiento

Para realizar un entrenamiento de aprendizaje por refuerzo, es necesario tomar una serie de decisiones relacionadas con las funciones de pérdida, optimizadores e hiperparámetros. Las funciones de pérdida cuantifican la discrepancia entre las acciones tomadas por el agente y las acciones óptimas. Los optimizadores, por otro lado, determinan cómo se actualizan los pesos del modelo durante el entrenamiento. Para este caso se ha utilizado Adam [43] como optimizador y MSE (error cuadrático medio) como función de pérdida.

En cuanto a los hiperparámetros, se ha empleado un valor de ϵ de 0.1, esta es la probabilidad de que el AUV seleccione una acción aleatoria en vez de la predicha por la red neuronal, como se ha visto en el Algoritmo 1. La tasa de aprendizaje, la rapidez con la que se actualizan los valores Q, escogida ha sido 0.0001 debido a que en trabajos similares[25] se ha observado que tasas de este orden de magnitud obtienen resultados satisfactorios. Por último, en el caso de la tasa de de descuento γ, es decir, la importancia relativa de las recompensas futuras respecto a las inmediatas, se ha escogido un valor de 0.95. Se ha elegido un valor cercano a 1 porque para que el AUV sea capaz de girar correctamente debe vencer a la inercia.

Capítulo 5

Resultados

En este capitulo se analizan los resultados obtenidos después del proceso de entrenamiento, verificando que el controlador robótico optimizado es capaz de resolver correctamente la tarea propuesta. Además, se ha evaluado el comportamiento del controlador en distintas situaciones y entornos, con el fin de comprobar sus capacidades de generalizar ante nuevos escenarios nunca vistos durante el entrenamiento.

5.1. Análisis del entrenamiento

El entrenamiento se ha realizado en el Escenario 2 debido a que es el más complejo y con mayor número de obstáculos (Figura 3.6). Debido a la alta demanda computacional durante el entrenamiento, este se ha realizado con un único robot en el entorno. La inicialización del robot en posición y orientación se realiza de forma aleatoria al comienzo de cada episodio. Un episodio termina cuando el robot colisiona con algún obstáculo o si se llega al límite de tiempo sin colisionar. En la Figura 5.1 se muestra la evolución del entrenamiento, mostrando en el eje y el tiempo que ha durado cada episodio, es decir, el tiempo que ha tardado en colisionar. Se ha empleado una media móvil para suavizar la curva ya que existe una gran varianza en la duración dependiendo del lugar donde haya empezado el episodio el AUV.

En la gráfica se pueden distinguir claramente 4 zonas. Durante las primeras decenas de episodios el robot no avanza, se limita a girar sobre si mismo por lo que no colisiona con nada y llega al limite de tiempo del episodio. A lo largo de los siguientes 3500 episodios, teniendo en cuenta la duración de los episodios, podemos deducir que el AUV se limita a ir en línea recta hasta colisionar con el primer obstáculo. Este comportamiento se muestra en la imagen a de la Figura 5.2, donde se ha simulado un AUV controlado por la red neuronal en el estado del episodio 2000.

Los 3000 episodios siguientes se corresponden con el periodo durante el cual el controlador ha aprendido a esquivar en las situaciones más sencillas. Esto se ve ejemplificado en la imagen b de la Figura 5.2, donde se muestra la simulación de un AUV controlado por la red neuronal del episodio 4000. Se observa como el robot esquiva el primer obstáculo pero colisiona con el siguiente. Esto es debido a que

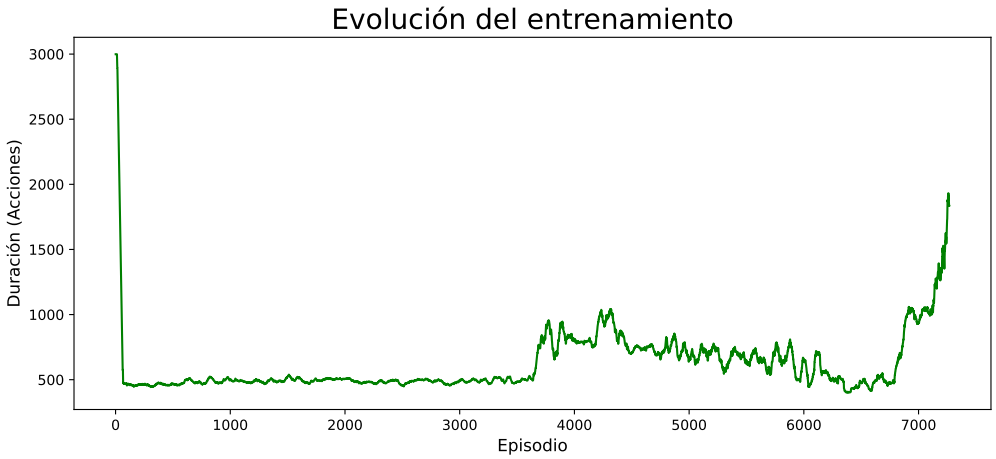
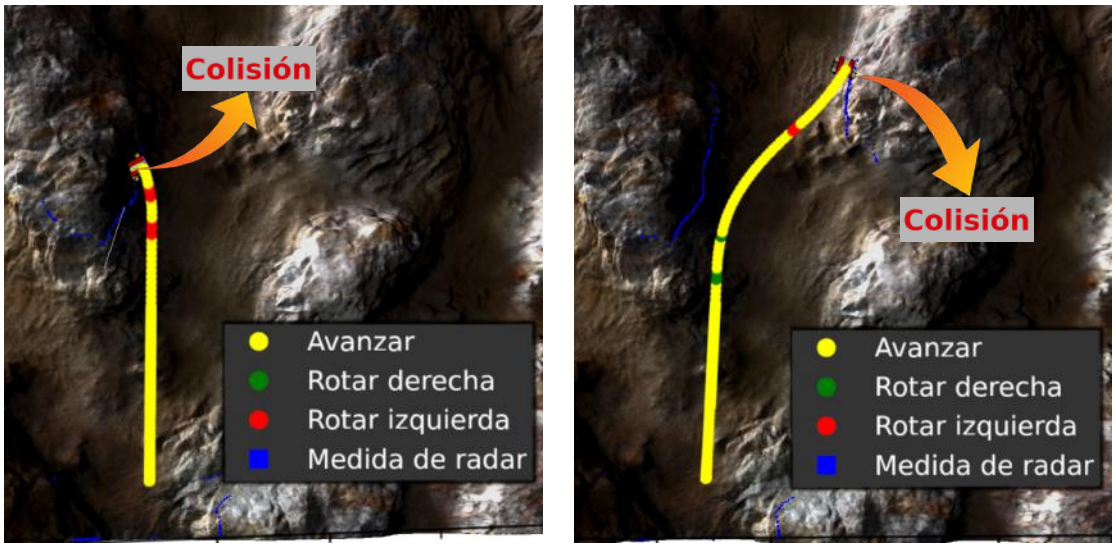


Figura 5.1: Evolución del entrenamiento del controlador.

al primer obstáculo se aproxima con un ángulo poco pronunciado. Una pequeña corrección a la derecha es suficiente para sortearlo. Sin embargo, tras este pequeño giro, el AUV se aproxima con un ángulo casi recto al siguiente obstáculo. En este caso el controlador decide hacer la misma pequeña corrección solo que a la izquierda en este caso. En este caso no es suficiente y el robot colisiona contra el obstáculo.

Por último, durante los 1000 episodios finales, el robot ya es capaz de girar a tiempo y luchar contra la inercia para esquivar objetos cuando avanza a una velocidad considerable. El incremento en la duración de los episodios refleja los progresos del proceso de entrenamiento para resolver el problema planteado. El comportamiento de esta versión se analiza en detalle en las Secciones 5.2 y 5.3.



a)Episodio 2000 *b)Episodio 4000*

Figura 5.2: Comparativa entre el comportamiento de un AUV con la red neuronal del episodio 2000 de entrenamiento frente a un AUV con la red neuronal del episodio 4000

5.2. Análisis del comportamiento de evasión de obstáculos

Para un vehículo subacuático los giros son más complicados que para uno terrestre. Los vehículos terrestres, siempre y cuando no vayan a una velocidad excesiva, pueden girar sus ruedas delanteras y su dirección de desplazamiento cambiará inmediatamente. Sin embargo, el control de un AUV es mucho más complejo. Existen principalmente dos opciones: o bien frena primero, poniendo los motores de los propulsores a girar a la inversa, y luego rota y avanza hacia la dirección que desea, o bien rota primero y luego se propulsa en contra de la inercia para frenar y, eventualmente, avanzar en esa dirección, una acción similar a la que realizaría un helicóptero en el aire. Debido a que el controlador del robot dispone de la opción de invertir los motores, la red neuronal ha aprendido a girar de la segunda manera. La Figura 5.3 muestra una secuencia de imágenes del AUV realizando dicho procedimiento.

Figura 5.3: Secuencia de movimiento del AUV para esquivar un obstáculo

Observando dicha secuencia podemos ver como para que el robot sea capaz de esquivar un objeto, este debe empezar a girar pronto, con el tiempo suficiente para vencer a la inercia. Esta necesidad de comenzar la acción pronto es un inconveniente debido a que, como se puede comprobar en la Figura 3.4, el radar tiene muchos puntos ciegos. Por lo que un obstáculo pequeño o con bordes puntiagudos puede aproximarse al AUV por estas zonas y el sensor no los detectaría hasta que estuviese muy cerca como para girar y contrarrestar la inercia. Está es la principal limitación presente en el modelo actual del AUV.

Este comportamiento de elegir girar pronto para después tener tiempo para vencer la inercia muestra una cualidad importante del aprendizaje por refuerzo, la recompensa a largo plazo. RL no tiene en cuenta únicamente las recompensas que recibe inmediatamente al ejecutar una acción. También toma las decisiones en función

de como será la recompensa en el futuro si elije esa acción, es decir, es capaz de tomar una decisión que en ese instante le proporcione una recompensa peor que otra si, a cambio, en el futuro recibirá una mejor recompensa por ello.

5.3. Análisis de trayectorias

En las Figuras 5.4, 5.5, 5.6 y 5.7 se muestra el recorrido realizado por el robot durante simulaciones en los distintos escenarios del simulador. Cada punto representa la posición del robot en un instante determinado y su color muestra la acción que el controlador decidió tomar en dicho momento. Las marcas azules son los puntos que el radar ha ido tomando como medidas a lo largo de la simulación.

Mediante estas figuras podemos analizar el comportamiento del AUV con el controlador desarrollado. Puede verse como, mediante el entrenamiento, la red neuronal ha conseguido a generalizar el problema y ha llegado a una solución mediante la cual es capaz de esquivar los obstáculos de los escenarios en los que no ha sido entrenada.

Poniendo el foco de atención en la toma de decisiones del controlador, se puede observar que la red neuronal tiene cierta tendencia a girar a la derecha. Cuando detecta un obstáculo de frente, siempre ejecuta la acción de girar a la derecha. Por el contrario, los giros a la izquierda solo se producen en el caso de que se detecte el obstáculo

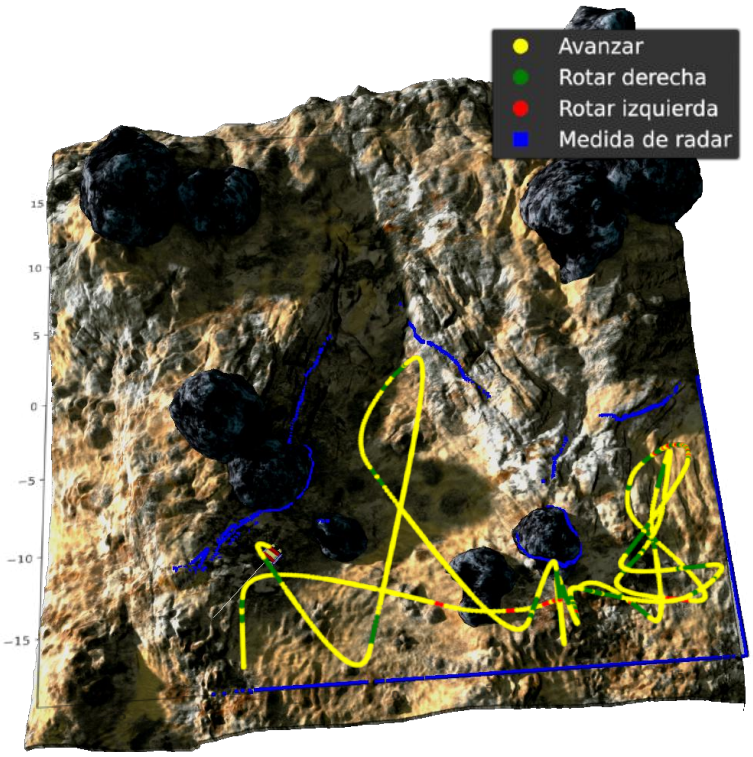


Figura 5.4: Trayectoria seguida por un AUV en el Escenario 1

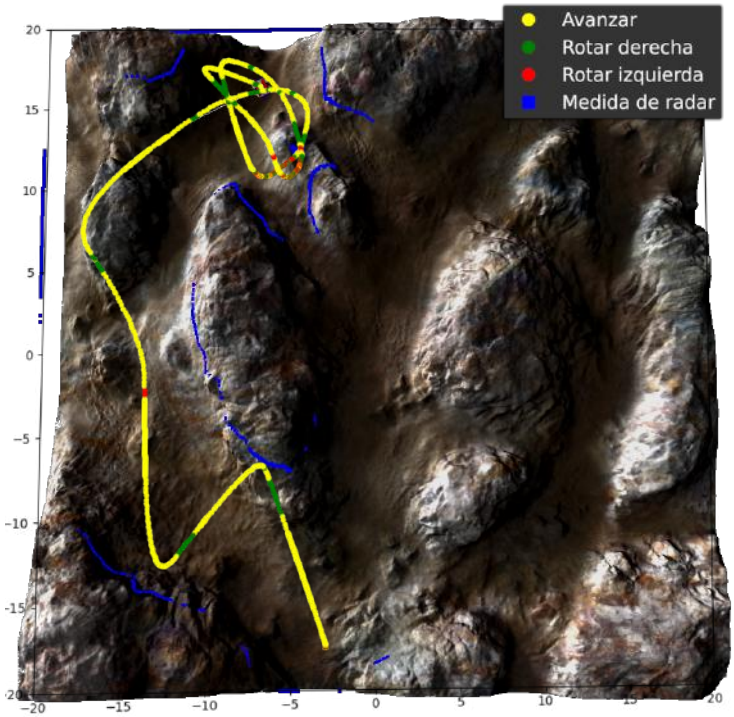


Figura 5.5: Trayectoria seguida por un AUV en el Escenario 2

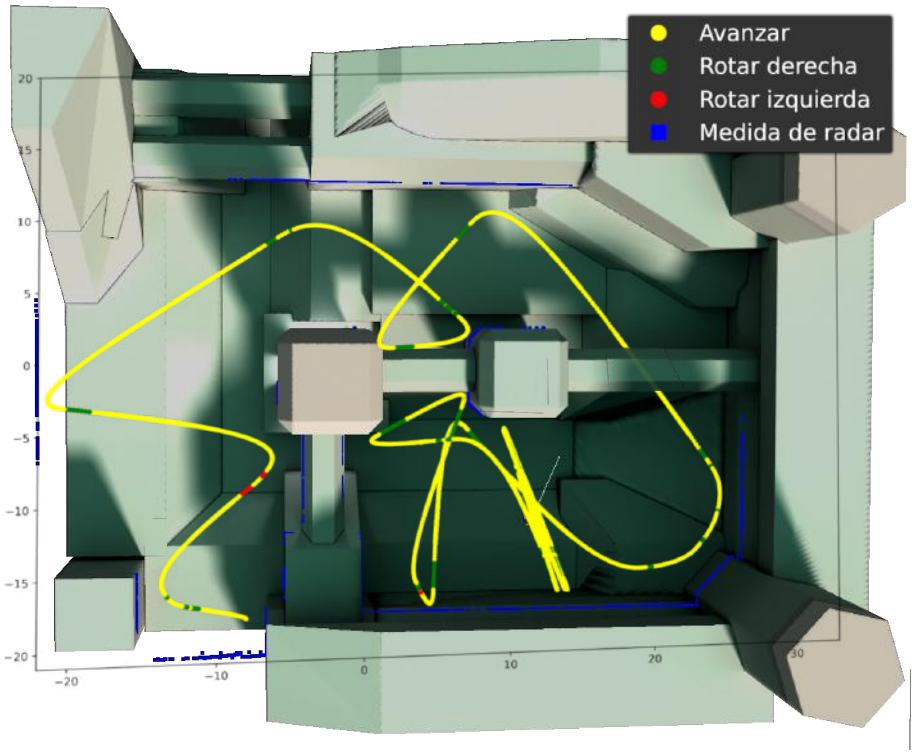


Figura 5.6: Trayectoria seguida por un AUV en el Escenario 3

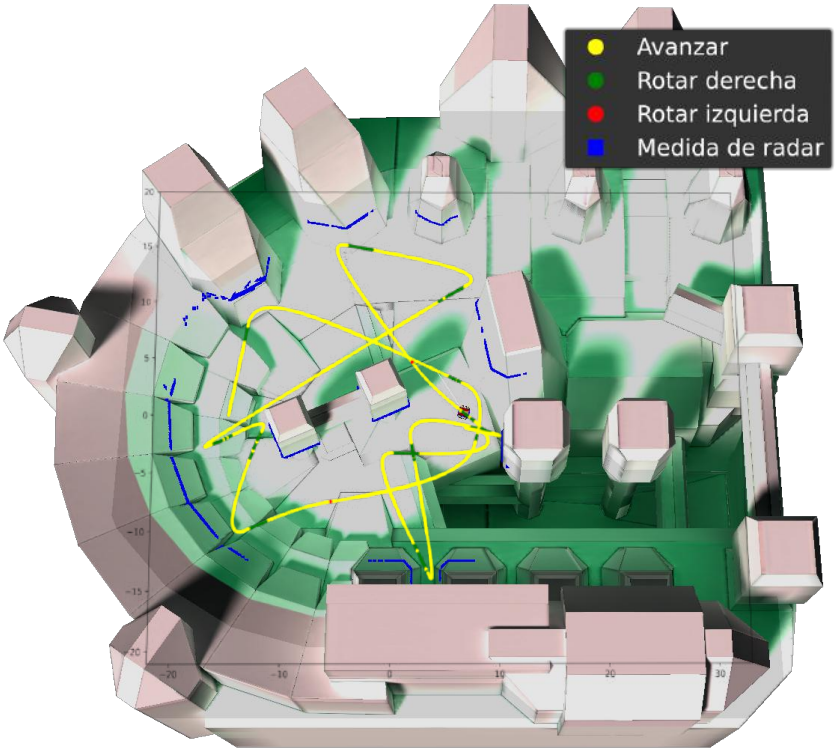


Figura 5.7: Trayectoria seguida por un AUV en el Escenario 4

exclusivamente a la izquierda.

En la primera imagen de la Figura 5.8 se puede apreciar como el robot llega a colisionar con una roca. Podemos especular que esta limitación se debe a la configuración del radar. Una separación de 45 grados entre las distintas direcciones de medida implican amplias zonas muertas que dificultan la detección del tamaño y posición de los elementos del escenario. Especulando más sobre este caso concreto se podría decir que el sensor detecta la roca únicamente con la dirección frontal del sensor radar. Esto puede limitar la información que el controlador tiene sobre el obstáculo, dando lugar una corrección de la trayectoria menor de lo necesaria, provocando que el AUV gire más tarde de lo que necesitaría y colisione con la roca.

En la segunda imagen de la Figura 5.8 se observa como el controlador está indeciso respecto a que acción tomar. Se suceden acciones de avanzar, girar a la derecha y girar a la izquierda de una forma que parece caótica. No obstante, a pesar de esto el AUV consigue no quedarse estancado en la esquina por lo que se puede afirmar que ha prevalecido la decisión correcta sobre las demás.

En la tercera imagen de la Figura 5.8 se aprecia un cambio de dirección no necesario para esquivar el obstáculo. Si el robot hubiese continuado avanzando de frente no habría colisionado. A pesar de esto, cuando el sensor detecta el obstáculo a la derecha del AUV, la red neuronal elige la acción de girar a la derecha a pesar de no tener ningún objeto de frente.

Figura 5.8: Imágenes ampliadas de algunas zonas de interés de las trayectorias simuladas

5.4. Análisis del comportamiento multiagente

Como se comentó en la Sección 4.1, la integración de Pytorch con el simulador se realizó teniendo en mente la posibilidad de realizar tanto simulaciones como entrenamientos. Esto nos permite evaluar el comportamiento del controlador neuronal cuando hay múltiples AUVs que deben evitar colisionar entre ellos.

En la Figura 5.9 se muestra el comportamiento de 5 AUVs cuando hay poca distancia entre ellos. Los puntos azules que se ven en una zona aparentemente sin obstáculos son las medidas que los radares han tomado cuando los robots estaban próximos. Se puede observar como la trayectoria que el controlador neuronal de cada uno de ellos decide seguir resulta ser, en conjunto, la solución óptima para alejarse lo más rápido posible para evitar posibles colisiones.

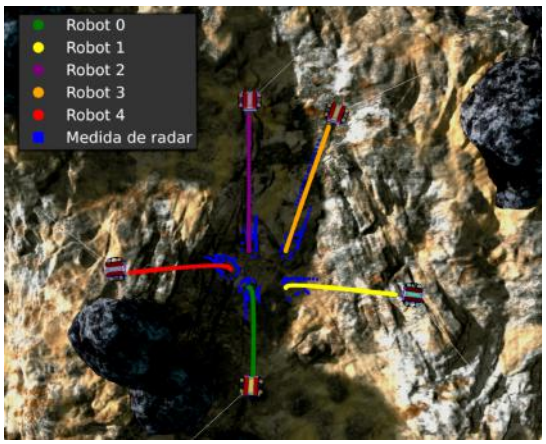


Figura 5.9: Trayectorias seguidas por varios AUVs para aumentar la distancia entre ellos y evitar posibles colisiones

5.5. Análisis estadístico del número de colisiones

Para mostrar la capacidad de generalización del modelo entrenado, en la Figura 5.10 se muestra la distribución del número de colisiones en cada uno de los cuatro escenarios mediante diagramas de caja calculados extrayendo los datos de 50 simulaciones independientes de 75 segundos en cada escenario y con un único robot.

Las medianas del número de colisiones por simulación de 75 segundos son de 2 colisiones en los escenarios 1, 2 y 4 y de 1 colisión en el Escenario 3. Estos valores similares indican que el controlador es válido para los distintos escenarios a pesar de solo haber sido entrenado en el Escenario 2. Esto da pie a afirmar que el modelo entrenado ha conseguido llegar a una solución general del problema.

Si bien es cierto que tanto las medianas como los rangos intercuartiles son similares entre los distintos entornos, no ocurre lo mismo con el caso de los valores atípicos. En los escenarios 1 y 2 hay mayor número y de mayor magnitud. Se podría especular que

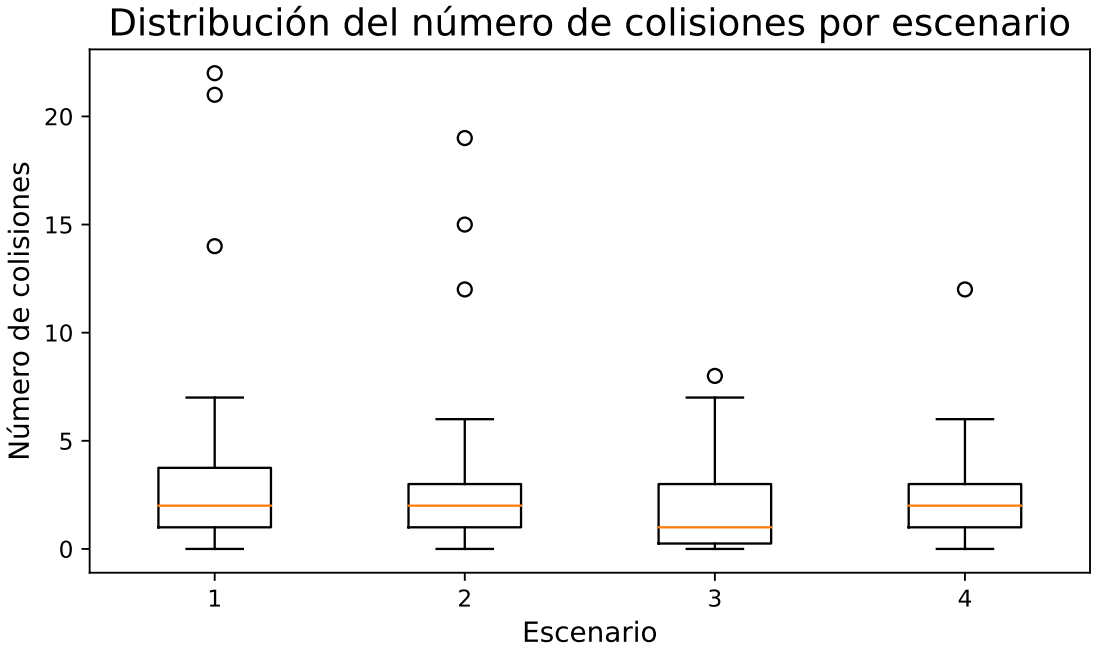


Figura 5.10: Distribución de colisiones del AUV en los distintos escenarios del simulador. Para cada escenario, la figura muestra la distribución del número de colisiones que el robot ha tenido en simulaciones independientes de 75 segundos. Se han tomado 50 muestras, correspondientes a 50 simulaciones independientes, y se han representado mediante diagramas de caja. En estos diagramas la línea naranja representa la mediana de la distribución y cada caja contiene las muestras comprendidas entre el primer y tercer cuartil, también conocido como rango intercuartil. Las líneas que salen de las cajas alcanzan hasta la muestra más alejada que se encuentre a menos de 1.5 veces el rango intercuartil. Las muestras atípicas se encuentran representadas con puntos blancos.

esto se debe a que los obstáculos de estos escenarios tienen formas más irregulares lo cual facilita que existan bordes en las paredes del terreno que el radar no detecte debido a sus puntos ciegos. Estas irregularidades también podrían provocar que el fondo del AUV colisione con ellas sin si quiera ser detectadas por el sensor radar debido a que estaría por debajo de su plano de lectura.

Capítulo 6

Conclusiones

Este capítulo está dedicado a resumir las conclusiones y resultados más relevantes de este Trabajo de Fin de Grado así como proponer lineas futuras de trabajo y mejora del proyecto.

6.1. Conclusiones

El objetivo principal de este trabajo es el diseño e implementación de controladores mediante redes neuronales artificiales para manejar el comportamiento de los AUVs mediante el uso de algoritmos de aprendizaje por refuerzo. Para verificar el cumplimiento de este objetivo se ha evaluado el controlador en distintos escenarios.

Para ello, después de estudiar el estado actual del aprendizaje por refuerzo, se ha elegido el algoritmo DQN para el entrenamiento de la red neuronal artificial usada para controlar a los robots. Para incorporar el uso de dicho algoritmo en el simulador del proyecto NAUTILUS, se han realizado modificaciones al controlador de los AUVs para que use redes neuronales para seleccionar la acción a realizar. Además, se ha implementado la red neuronal artificial y el sistema de entrenamiento usando la librería Pytorch. Se ha usado un perceptrón multicapa con dos capas ocultas, de 20 y 8 neuronas, con una capa de entrada de 8 nodos, que corresponde al vector de salida del sensor radar y tres neuronas de salida para tres posibles movimientos del AUV (avanzar, rotar a la izquierda y rotar a la derecha).

Tras más de 7000 episodios de entrenamiento se ha llegado a una versión satisfactoria de la red. Se ha mostrado la forma en la que los robots esquivan los obstáculos, llegando a la conclusión de que el principal limitante es la inercia propia del medio acuático. Por ello es importante que detecten los obstáculos del entorno con el tiempo suficiente para cambiar de dirección o sentido y contrarrestar la inercia.

Se han evaluado las trayectorias seguidas por el AUV en los distintos escenarios comprobando que el controlador es capaz de resolver la tarea correctamente en todos ellos. Para verificar esto numéricamente se ha estudiado el número de colisiones en los distintos escenarios mediante la realización de 50 simulaciones por escenario, de

75 segundos de duración y con un único AUV simultáneo. La mediana del número de colisiones en el escenario 3 ha sido 1 y en los otros tres escenarios ha sido 2. A pesar de estas similitudes en la mediana de la distribución, existe una clara diferencia entre el número y la magnitud de los valores atípicos entre los dos primeros escenarios y los otros dos. En los primeros hay más y tienen un valor mayor. Esto puede deberse al hecho de que los elementos del terreno en ambos son más irregulares que en los escenarios 3 y 4. Esto dificulta al radar la detección de los obstáculos y podría ser la causa de esta diferencia.

Para el controlador no existe diferencia entre obstáculos móviles y estacionarios, simplemente detecta la proximidad de objetos sólidos. Esto permite al AUV generalizar el comportamiento y alejarse de otros agentes para evitar colisiones. Se ha analizado el comportamiento del controlador en un entorno multi-agente, comprobando que es capaz de abordar correctamente un escenario con otros 4 robots.

En conclusión, se ha desarrollado un modelo de perceptrón multicapa que implementar en el controlador de AUVs. Se le ha entrenado usando aprendizaje por refuerzo con el objetivo de evitar colisionar con los elementos del entorno en base a las medidas de un radar de ocho posiciones. Se han analizado las trayectorias tomadas por los AUVs en los distintos escenarios y se ha validado el comportamiento del controlador neuronal en distintas simulaciones independientes y con inicializaciones aleatorias obteniendo resultados estadísticamente significativos. Esto evidencia como la red neuronal, a pesar de únicamente haber sido entrenada en le escenario 2, ha sido capaz de llegar a una solución generalizada del problema planteado por lo que se puede afirmar que se han cumplido los objetivos propuestos.

6.2. Líneas futuras

La conclusión de este trabajo abre las puertas a, un número de posibles desarrollos asociados al proyecto. Si nos centramos en el propio algoritmo de RL, se podría mejorar el controlador como esquivador de obstáculos. Por un lado se plantea la posibilidad mejorar la arquitectura de la red neuronal. Por ejemplo, añadir recurrencia podría permitir al AUV ser más eficaz venciendo a la inercia del medio. Por otro lado, es posible estudiar la aplicación de otros algoritmos de RL, como el actor-critic, el cual manejar espacios de estados continuos, para evaluar las diferencias en los resultados obtenidos y observar si hay mejoría en el desempeño del controlador.

Por otro lado, si se plantea extender el controlador desarrollado. Una posibilidad sería incorporar nuevos movimientos posibles al controlador, desplazarse lateralmente por ejemplo, para aumentar la movilidad de los AUVs. Otra opción sería la incorporación de otros objetivos al controlador, diseñando tareas y experimentos más complejos. Por ejemplo, buscar un objeto por el escenario o seguir a otro AUV por el entorno. Para esto sería necesario incorporar nuevos sensores en el simulador. También se propone la posibilidad de continuar con el proyecto abordando el problema del

movimiento coordinado de robots bajo el agua. Para ello se les entrenaría para que se desplacen en formación o realicen tareas juntos.

Referencias

[1] Malte von Benzon, Fredrik Fogh Sørensen, Esben Uth, Jerome Jouffroy, Jesper Liniger, and Simon Pedersen. An open-source benchmark simulator: Control of a bluerov2 underwater robot. *Journal of Marine Science and Engineering*, 10(12):1898, Dec 2022.

[2] Thor I Fossen. Marine control systems–guidance. navigation, and control of ships, rigs and underwater vehicles. *Marine Cybernetics, Trondheim, Norway, Org. Number NO 985 195 005 MVA, www. marinecybernetics. com, ISBN: 82 92356 00 2*, 2002.

[3] Austin Choi-Fitzpatrick, Dana Chavarria, Elizabeth Cychosz, John Paul Dingens, Michael Duffey, Katherine Koebel, Sirisack Siriphanh, Merlyn Yurika Tulen, Heath Watanabe, Tautvydas Juskauskas, et al. Up in the air: A global estimate of non-violent drone use 2009-2015. 2016.

[4] Lassi Sundqvist et al. Cellular controlled drone experiment: Evaluation of network requirements. Master's thesis, 2015.

[5] T Salgado-Jimenez, JL Gonzalez-Lopez, LF Martinez-Soto, E Olguin-Lopez, PA Resendiz-Gonzalez, and M Bandala-Sanchez. Deep water rov design for the mexican oil industry. In *OCEANS'10 IEEE SYDNEY*, pages 1–6. IEEE, 2010.

[6] D Richard Blidberg. The development of autonomous underwater vehicles (auv); a brief summary. In *Ieee Icra*, volume 4, pages 1–12, 2001.

[7] Jørgen Lorentz and Junku Yuh. A survey and experimental study of neural network auv control. In *Proceedings of Symposium on Autonomous Underwater Vehicle Technology*, pages 109–116. IEEE, 1996.

[8] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[9] MR Blackburn and HG Nguyen. An artificial neural system for autonomous undersea vehicles. Technical report, NAVAL OCEAN SYSTEMS CENTER SAN DIEGO CA, 1988.

[10] Teruo Fujii and Tamaki Ura. Neural-network-based adaptive control systems for auvs. *Engineering Applications of Artificial Intelligence*, 4(4):309–318, 1991.

[11] Gordon DeMuth and Steve Springsteen. Obstacle avoidance using neural networks. In *Symposium on Autonomous Underwater Vehicle Technology*, pages 213–215. IEEE, 1990.

[12] Xue Bin Peng, Marcin Andrychowicz, Wojciech Zaremba, and Pieter Abbeel. Sim-to-real transfer of robotic control with dynamics randomization. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 3803–3810. IEEE, 2018.

[13] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[14] Burrhus F Skinner. Reinforcement today. *American Psychologist*, 13(3):94, 1958.

[15] Vijay Konda and John Tsitsiklis. Actor-critic algorithms. *Advances in neural information processing systems*, 12, 1999.

[16] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*, 2013.

[17] Lionel Lapierre and Didik Soetanto. Nonlinear path-following control of an auv. *Ocean engineering*, 34(11-12):1734–1744, 2007.

[18] Lionel Lapierre and Bruno Jouvencel. Robust nonlinear path-following control of an auv. *IEEE Journal of Oceanic Engineering*, 33(2):89–102, 2008.

[19] Hui Wu, Shiji Song, Keyou You, and Cheng Wu. Depth control of model-free auvs via reinforcement learning. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 49(12):2499–2510, 2018.

[20] Xiang Cao, Changyin Sun, and Mingzhong Yan. Target search control of auv in underwater environment with deep reinforcement learning. *IEEE Access*, 7:96549–96559, 2019.

[21] Chenming Zhang, Peng Cheng, Bin Du, Botao Dong, and Weidong Zhang. Auv path tracking with real-time obstacle avoidance via reinforcement learning under adaptive constraints. *Ocean Engineering*, 256:111453, 2022.

[22] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. *arXiv preprint arXiv:1509.02971*, 2015.

[23] Hui Wu, Shiji Song, Yachu Hsu, Keyou You, and Cheng Wu. End-to-end sensorimotor control problems of auvs with deep reinforcement learning. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5869–5874. IEEE, 2019.

[24] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[25] Simen Theie Havenstrøm, Adil Rasheed, and Omer San. Deep reinforcement learning controller for 3d path following and collision avoidance by autonomous underwater vehicles. *Frontiers in Robotics and AI*, page 211, 2021.

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

[27] Jianya Yuan, Hongjian Wang, Honghan Zhang, Changjian Lin, Dan Yu, and Chengfeng Li. Auv obstacle avoidance planning based on deep reinforcement learning. *Journal of Marine Science and Engineering*, 9(11):1166, 2021.

[28] Mandavilli Srinivas and Lalit M Patnaik. Genetic algorithms: A survey. *computer*, 27(6):17–26, 1994.

[29] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[30] Jian Xu, Fei Huang, Di Wu, Yunfei Cui, Zheping Yan, and Xue Du. A learning method for auv collision avoidance through deep reinforcement learning. *Ocean Engineering*, 260:112038, 2022.

[31] Richard S Sutton, Andrew G Barto, et al. Reinforcement learning. *Journal of Cognitive Neuroscience*, 11(1):126–134, 1999.

[32] BF Skinner. The reinforcing effect of a differentiating stimulus. *The Journal of General Psychology*, 14(2):263–278, 1936.

[33] Richard Bellman. *Dynamic Programming*. Princeton University Press, Princeton, NJ, USA, 1 edition, 1957.

[34] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[35] Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12, 1999.

[36] Gavin A Rummery and Mahesan Niranjan. *On-line Q-learning using connectionist systems*, volume 37. University of Cambridge, Department of Engineering Cambridge, UK, 1994.

[37] Blue Robotics. BlueROV2 remotely operated vehicle. `https://https://bluerobotics.com/store/rov/bluerov2/`.

[38] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *Osdi*, volume 16, pages 265–283. Savannah, GA, USA, 2016.

[39] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

[40] Bjarne Stroustrup. *The C++ programming language*. Pearson Education, 2013.

[41] Leonardo Noriega. Multilayer perceptron tutorial. *School of Computing. Staffordshire University*, 4:5, 2005.

[42] Tomas Mikolov, Martin Karafiát, Lukas Burget, Jan Cernockỳ, and Sanjeev Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, pages 1045–1048. Makuhari, 2010.

[43] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Apéndice A

Aspectos éticos, económicos, sociales y ambientales

Este apéndice esta dedicado a la descripción y análisis del impacto de este proyecto en distintos ámbitos como el social, económico u ambiental. Se considera que no existe un impacto ético por lo que se considerarán los ámbitos social, económico y ambiental.

A.1. Introducción

Tanto el uso de robots como el de la inteligencia artificial está cada vez más extendido. Su uso está teniendo una serie de impactos en distintos ámbitos de la sociedad y economía. Este Trabajo de Fin de Grado no es la excepción, a continuación se exploraran algunos de ellos.

A.2. Descripción de impactos relevantes relacionados con el proyecto

Los impactos de este proyecto pueden dividirse en las siguientes categorías:

- Impacto social: este trabajo, al estar enmarcado dentro del proyecto NAUTILUS, ayudará a la investigación y desarrollo de nuevas soluciones para el control de robots en medios poco habituales para los humanos como lo es, en este caso, las profundidades acuáticas. También cabe mencionar que el uso de AUVs podría, en un futuro, sustituir a los buzos en las misiones más arriesgadas de buceo, evitando poner sus vidas en riesgo.
- Impacto económico: el uso de simuladores para evaluar el comportamiento de robots permite reducir costes tanto de materiales como de mano de obra. Con una simulación no hay costes de transporte al lugar de pruebas ni de construcción y puesta a punto del AUV.
- Impacto medioambiental: el uso del simulador evita el impacto medioambiental ya que no es necesario sumergir robots en el océano u otros cuerpos de agua naturales para probarlos, corriendo el riesgo de que por una avería naufraguen y contaminen dicho entorno.

A.3. Análisis detallado de alguno de los principales impactos

Las tareas de búsqueda de personas en cuevas submarinas son unas de las misiones de rescate más complicadas y arriesgadas que existen. La dificultad de navegación, la baja temperatura del agua, la poca visibilidad o la posible narcolepsia suponen un gran peligro para los buzos.

Un UAV controlado por una red neuronal, como el usado en este Trabajo de Fin de Grado, podría navegar rápidamente la cueva buscando a la víctima sin poner más vidas en peligro. Es más, en caso de ser una cueva muy grande o con muchas bifurcaciones, un enjambre de estos AUVs, como el que está desarrollando el proyecto NAUTILUS, podría recorrer la cueva exponencialmente más rápido. Los AUVs irían avanzando y eligiendo entre ellos que camino toma cada uno. Una solución así tiene un importante impacto social.

A.4. Conclusiones

Este proyecto se ha comprometido a apoyar los Objetivos de Desarrollo Sostenible (ODS) que fueron acordados por los Estados Miembros de las Naciones Unidas en 2015. Estos objetivos tienen la finalidad de cuidar nuestro planeta y lograr una sociedad pacífica y próspera para el año 2030, a través de la promoción de la salud y el bienestar de las personas.

Apéndice B

Presupuesto económico

Este proyecto se ha desarrollado a lo largo de 6 mesess en la Escuela Técnica Superior de Ingenieros de Telecomunicación. Se ha estimado un presupuesto teniendo en cuenta los recursos físicos y humanos empleados.

- **Personal**: En estos coses incluimos los recursos humanos empleados en el trabajo. Principalmente tres personas han trabajado en este proyecto, el tutor, co-tutor y el estudiante de ingeniería(ver Tabla B.1).

	Coste horario (€)	**Horas**	**Total (€)**
Tutor	60	30	1800
Co-tutor	60	30	1800
Estudiante de ingeniería	30	300	9000
TOTAL			**12600**

Tabla B.1: Costes de personal.

- **Costes de recursos materiales**: La principal herramienta usada para la implementación del trabajo ha sido el ordenador personal del estudiante, que cuenta con un i7 4790K y 16 GB de ram, al que se le ha actualizado la tarjeta gráfica a una RTX 3060. Se han calculado los costes estimando una vida útil de 5 años (ver Tabla B.2).

	Tiempo de vida (años)	**Uds.**	**Coste (€)**	**Amortización (€/mes)**	**Uso (meses)**	**Total (€)**
Ordenador	5	1	1000	16,66	6	100
RTX 3060	5	1	400	6,66	6	40
TOTAL						**140**

Tabla B.2: Costes de recursos materiales.

Con esto el coste total del proyecto, teniendo en cuenta el IVA, queda resumido en la

	Coste
Costes de personal	12600€
Costes de material	140€
Subtotal	12740€
IVA	2675,40€
Total	15415,40€

Tabla B.3: Costes totales.

TablaB.3