UNIVERSIDAD EUROPEA DE MADRID

ESCUELA DE ARQUITECTURA, INGENIERÍA, CIENCIA Y COMPUTACIÓN (STEAM)

ÁREA INGENIERÍA INDUSTRIAL

Grado en ingeniería en sistemas industriales

TRABAJO FIN DE GRADO

SISTEMA DE RECONOCIMIENTO DE PATRONES EN UN VEHÍCULO SUBMARINO AUTÓNOMO

Alumno: Ana Martín Rodriguez

Director: Dr. Héctor Eloy Sánchez Sardi

JUNIO 2025

TÍTULO: SISTEMA DE RECONOCIMIENTO DE PATRONES EN UN VEHÍCULO SUBMARINO AUTÓNOMO

AUTOR: Ana Martin Rodriguez

DIRECTOR DEL PROYECTO: Dr. Héctor Eloy Sánchez Sardi

FECHA: 8 de junio de 2025

RESUMEN

En el desafiante y complejo entorno submarino, la capacidad de un vehículo para interpretar su medio es fundamental. Este Trabajo Fin de Grado (TFG) aborda la integración de un sistema de reconocimiento de patrones en un Vehículo Submarino Autónomo (AUV), centrándose especialmente en la detección de personas en entornos acuáticos. Todo ello con un objetivo muy claro: potenciar las misiones de búsqueda y rescate.

El proyecto reconoce los retos que presenta el medio submarino para el procesamiento visual; hablamos de problemas tan complejos como la baja visibilidad, la iluminación limitada y las distorsiones ópticas que afectan gravemente la calidad de las imágenes. Para superar estos obstáculos, se ha adoptado con determinación un enfoque basado en el aprendizaje profundo, seleccionando para la detección de objetos la potente arquitectura YOLOv11n. Además, empleamos la técnica de *transfer learning* y un *dataset* específico, el "Drowning Detection", para entrenar y ajustar el modelo.

El sistema de visión, implementado en Python, se integró de forma fluida con un modelo de simulación de AUV desarrollado en MATLAB, estableciendo una comunicación bidireccional muy efectiva para el intercambio y análisis de imágenes en tiempo real simulado. Las pruebas exhaustivas realizadas en este entorno simulado demostraron la viabilidad del sistema. Los resultados fueron muy satisfactorios, con una Precisión del 90% y un Recall del 87%, confirmando su capacidad para detectar personas en diversas secuencias de vídeo. Este proyecto sienta unas bases sólidas para futuras mejoras y, lo que es más importante, para su potencial implementación en entornos reales.

Palabras clave: Aprendizaje profundo, Búsqueda y rescate, Reconocimiento de patrones, Vehículo Submarino Autónomo (AUV), Visión artificial, YOLO.

ABSTRACT

In the challenging and complex underwater environment, a vehicle's ability to interpret its surroundings is paramount. This bachelor's Thesis (TFG) addresses the integration of a pattern recognition system into an Autonomous Underwater Vehicle (AUV), specifically focusing on the detection of individuals in aquatic settings. This endeavor's overarching objective is to significantly enhance search and rescue missions.

The project acknowledges the inherent challenges that the underwater medium presents for visual processing; these challenges encompass complex issues such as low visibility, limited illumination, and optical distortions that severely impact image quality. To overcome these obstacles, a deep learning-based approach has been decisively adopted, selecting the powerful YOLOv11n architecture for object detection. Furthermore, the transfer learning technique and a specific dataset, "Drowning Detection," were employed to train and fine-tune the model.

The vision system, meticulously implemented in Python, was seamlessly integrated with an AUV simulation model developed in MATLAB, establishing an efficient bidirectional communication for image exchange and analysis in a simulated real-time environment. The exhaustive tests

conducted in this simulated environment undeniably demonstrated the system's viability. The results proved highly satisfactory, with a Precision of 90% and a Recall of 87%, confirming its capability to detect individuals in various video sequences. This project lays a solid foundation for future improvements and, more importantly, for its potential implementation in real-world environments.

Key words: Autonomous Underwater Vehicle (AUV), Computer Vision, Deep Learning, Pattern Recognition, Search and Rescue, YOLO.

Índice

Índice de Figuras

Capítulo 1. Introducción

1.1 Contexto y motivación del proyecto

La exploración y comprensión del mundo submarino sigue siendo un desafío fascinante, a pesar de los notables avances tecnológicos. De hecho, gran parte de los océanos permanece aún inexplorada. El reconocimiento de patrones en estos entornos representa un área de investigación de gran relevancia en el ámbito de la robótica autónoma y la exploración oceánica. Sus aplicaciones son variadas, incluyendo la búsqueda y rescate, la recuperación de naufragios y la exploración de la flora y fauna submarina. Los progresos en esta disciplina han permitido mejorar considerablemente la capacidad de los vehículos submarinos autónomos (AUV) para operar en misiones de exploración, monitoreo ambiental, seguridad y rescate. No obstante, a pesar de estos avances, la detección y clasificación de objetos en entornos acuáticos continúa siendo compleja debido a las condiciones adversas del medio, como la baja visibilidad, la variabilidad lumínica y la presencia de partículas en suspensión.

El desarrollo de algoritmos robustos de reconocimiento de patrones, que permitan la identificación precisa de objetos submarinos, es fundamental para aumentar la autonomía y eficiencia de los AUVs y mejorar sus potencialidades para diversas aplicaciones, como la búsqueda y rescate, la exploración submarina y la inspección de instalaciones en alta mar. Este Trabajo Fin de Grado (TFG) se enmarca en la necesidad de perfeccionar las capacidades de estos sistemas mediante la integración de un módulo de procesamiento de imágenes, capaz de detectar y clasificar distintos elementos del entorno submarino. La implementación de estas tecnologías no solo potenciará la exploración oceánica y la investigación científica, sino que también tendrá aplicaciones directas en la preservación del medio ambiente y en la ejecución de operaciones de búsqueda y rescate.

En este contexto, el proyecto también se alinea con la dimensión social de la sostenibilidad, ya que el desarrollo de herramientas automáticas para la localización de personas en entornos hostiles puede reducir el tiempo de respuesta ante emergencias, aumentar la seguridad de los equipos de rescate y, en última instancia, salvar vidas humanas. De este modo, se refuerza el valor social del trabajo, al contribuir a sistemas más eficaces y autónomos en situaciones críticas.

A continuación, se presentan los argumentos principales que fundamentan la relevancia de esta investigación:

- **Aumento de la autonomía en vehículos submarinos:** La incorporación de un sistema de reconocimiento de patrones permitirá que los AUV tomen decisiones más precisas sin necesidad de intervención humana, mejorando su capacidad operativa en exploraciones prolongadas.

- **Mejora en la precisión de detección y clasificación de objetos:** El uso de algoritmos avanzados de inteligencia artificial, optimizados para entornos acuáticos, permitirá reducir los errores en la identificación de objetos, incrementando la fiabilidad de estos sistemas.

- **Aplicaciones en sectores estratégicos:** Este desarrollo puede beneficiar a múltiples sectores, incluyendo la exploración oceánica, la arqueología submarina, la conservación del medio marino y la industria de extracción de recursos.

- **Contribución al conocimiento científico y tecnológico:** La investigación generará nuevas metodologías y herramientas para el procesamiento de imágenes submarinas, lo que podría servir como base para futuros desarrollos en robótica marina.

- **Posibilidad de integración con sistemas existentes:** El módulo de reconocimiento de patrones podrá ser integrado en AUVs ya operativos, optimizando su funcionalidad sin requerir modificaciones estructurales significativas.

Este TFG aborda un desafío tecnológico relevante en el campo del reconocimiento de patrones en entornos submarinos. Su implementación en un AUV permitirá mejorar la autonomía, precisión y eficiencia de estos sistemas, favoreciendo su aplicación en diversas áreas de interés científico e industrial. Además, representa una oportunidad para profundizar en el estudio de algoritmos de inteligencia artificial y su adaptación a escenarios complejos, contribuyendo así al desarrollo de la robótica submarina y la exploración de los océanos.

Además, este proyecto también se alinea con la dimensión social de la sostenibilidad, ya que el desarrollo de sistemas automatizados para la detección de personas en entornos acuáticos puede tener un impacto directo en la reducción de riesgos durante misiones de búsqueda y rescate. Al permitir intervenciones más rápidas y seguras, estos sistemas contribuyen a salvar vidas humanas y a mejorar la seguridad en operaciones críticas. Por ello, más allá de su valor tecnológico, esta investigación presenta un claro potencial de aplicación en beneficio de la sociedad.

1.2 Objetivos del proyecto

1.2.1 Objetivo Global

Todo proyecto surge de una necesidad, y en este caso, el deseo de potenciar la autonomía y la capacidad de percepción de los vehículos submarinos ha guiado nuestra labor. El camino se ha centrado en establecer una serie de metas claras y ambiciosas.

1.2.2 Objetivos Específicos

1. Investigar y analizar los métodos actuales de reconocimiento de patrones en entornos submarinos.
    - Revisar literatura científica y estudios recientes sobre procesamiento de imágenes submarinas.
    - Evaluar algoritmos empleados en aplicaciones submarinas, considerando ventajas y limitaciones.
2. Entrenar y evaluar un modelo de detección basado en YOLO u otros enfoques alternativos.

- Seleccionar y procesar un conjunto de datos adecuado para la detección de objetos en entornos submarinos.
- Entrenar modelos pre-entrenados y realizar ajustes para mejorar su rendimiento en imágenes submarinas.
- Comparar diferentes métodos y seleccionar el más adecuado para la aplicación específica del AUV.

3. Integrar el modelo de reconocimiento con el entorno de simulación en MATLAB.
    - Desarrollar un flujo de trabajo que permita la comunicación entre MATLAB y Python para el análisis de imágenes.
    - Simular el comportamiento del AUV en MATLAB, incorporando la detección visual para detener el recorrido cuando se identifique presencia de personas.
4. Evaluar el desempeño del sistema integrado en simulaciones de entornos submarinos.
    - Ejecutar pruebas con diferentes condiciones de visibilidad y ruido en imágenes.
    - Identificar limitaciones y proponer mejoras en la detección y clasificación de objetos.
5. Documentar y analizar los resultados obtenidos para su aplicación en futuros desarrollos.
    - Elaborar un informe detallado con la metodología, resultados y conclusiones del estudio.
    - Comparar los resultados con investigaciones previas y evaluar su aplicabilidad en escenarios reales.
    - Proponer posibles líneas de mejora para futuras iteraciones del sistema.

Capítulo 2. Marco teórico

2.1 Vehículos submarinos autónomos

2.1.1 Tipos de vehículos y aplicaciones actuales

El campo de la robótica submarina ha experimentado un avance significativo, dando lugar a diversos tipos de vehículos diseñados para operar bajo el agua. Entre ellos, los más destacados son los Vehículos Operados Remotamente (ROVs, por sus siglas en inglés, *Remotely Operated Vehicles*) y los Vehículos Submarinos Autónomos (AUVs, por sus siglas en inglés, *Autonomous Underwater Vehicles*). Además, existen los AUVs semi-sumergibles, que combinan características de vehículos de superficie y sumergibles autónomos.

Los ROVs son vehículos submarinos conectados a un buque de superficie mediante un cable (*tether*), el cual les proporciona energía y posibilita la comunicación y el control en tiempo real por parte de un operador humano. Tradicionalmente, los ROVs con manipuladores han sido el estándar para tareas de intervención en el entorno submarino. Sin embargo, dependen de una conexión física constante, lo que puede limitar su alcance y maniobrabilidad.

Por otro lado, los AUVs son vehículos no tripulados que operan de manera autónoma, sin necesidad de un cable de conexión. Están equipados con sus propias fuentes de energía, normalmente baterías, y son capaces de ejecutar misiones preprogramadas sin intervención directa. La comunicación con estos vehículos se realiza principalmente a través de enlaces acústicos, aunque se están explorando opciones ópticas para futuras implementaciones. Esta autonomía les permite superar las limitaciones de los cables de los ROVs para ciertas tareas, especialmente en la realización de levantamientos y exploración.

Los AUVs semi-sumergibles representan una categoría intermedia. Inicialmente fueron desarrollados para mejorar la eficiencia de cobertura en levantamientos hidrográficos. Estos vehículos combinan atributos deseables de los buques de superficie con características de los AUVs, como la capacidad de operar bajo el agua y, a menudo, la autonomía. Disponen de un mástil que sobresale de la superficie, lo que facilita la comunicación y la navegación por satélite.

En cuanto a sus aplicaciones, los vehículos submarinos desempeñan un papel crucial en diversos sectores:

- **Investigación Científica Oceánica:** Los AUVs se han convertido en plataformas excelentes para la recopilación de datos cuantitativos en el océano. Se utilizan para estudiar fenómenos como la turbulencia subsuperficial, las poblaciones de plancton y krill, la variación espacial de la turbulencia sobre bancos de arena, y para realizar mediciones acústicas. Programas como el *Autosub Science Missions Programme* han demostrado la utilidad de los AUVs para abordar cuestiones que sólo pueden responderse utilizando las capacidades únicas de estos vehículos. Los AUVs también se emplean para el despliegue de sensores especializados en la investigación marina en biología, química, física, óptica y cartografía del fondo marino.

- **Defensa:** Los AUVs tienen aplicaciones tanto ofensivas como defensivas en el ámbito naval. Se están desarrollando para tareas como la contramedida de minas, el reconocimiento y la vigilancia submarina. La tendencia es a desarrollar vehículos con una mayor capacidad de detección y toma de decisiones autónoma para eliminar a los humanos de áreas de peligro. Algunos ejemplos incluyen el desarrollo de semi-sumergibles para la caza de minas y programas militares de UUVs (por sus siglas en inglés, *Unmanned Underwater Vehicles*) como el programa Marlin del Reino Unido.
- **Industria Offshore:** En la industria del petróleo y el gas, los AUVs están comenzando a tener un impacto significativo, especialmente en levantamientos de carácter comercial. Se utilizan para inspección de tuberías, levantamientos de riesgos en emplazamientos y bloques, y levantamientos de cables submarinos. La fiabilidad técnica y las soluciones rentables son prioritarias en este sector. Se están explorando aplicaciones de AUVs para la intervención en instalaciones submarinas, con la posibilidad de acoplarse para recibir energía y control desde la superficie.
- **Levantamientos Detallados del Fondo Marino:** Los AUVs equipados con sónar multihaz, perfilador de subfondo y sónar de barrido lateral permiten generar mapas digitales detallados del terreno submarino. Estos mapas tienen diversas aplicaciones, como el estudio de rutas de oleoductos, la evaluación ambiental de arrecifes de coral de aguas frías, la evaluación de poblaciones de peces, la demostración de contramedidas de minas y los levantamientos batimétricos de campos de petróleo y gas.
- **Aplicaciones Emergentes:** Se están explorando nuevas aplicaciones para los AUVs en áreas como la arqueología submarina, la monitorización ambiental o el transporte de carga. La versatilidad de los AUVs los posiciona como una tecnología clave en la exploración de entornos extremos.

Actualmente, la línea entre ROVs y AUVs se está volviendo cada vez más difusa, con el desarrollo de sistemas híbridos que buscan combinar las mejores propiedades de ambos. Además, se está investigando activamente la manipulación autónoma con AUVs. Proyectos como SAUVIM y Girona 500 I-AUV han demostrado avances significativos en esta dirección, lo que podría ampliar considerablemente las capacidades de los vehículos autónomos para tareas de intervención submarina.

2.1.2 Arquitectura general de un AUV

La arquitectura general de un vehículo submarino autónomo (AUV) es un sistema complejo que integra múltiples tecnologías para permitir su operación independiente bajo el agua. A diferencia de los vehículos operados remotamente (ROVs), los AUVs no requieren conexión física con un buque de superficie, lo que impone requisitos específicos en su diseño.

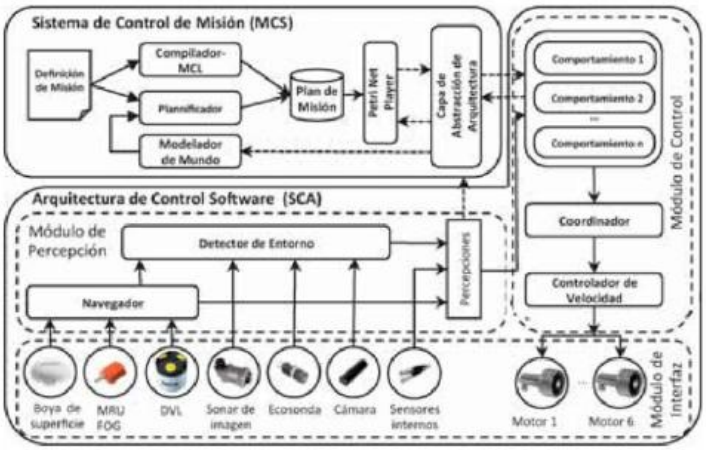

Figura 1 - Arquitectura de control de un AUV.

Fuente: Carreras, M., Ridao, P., Garcia, R., Romagós, D., & Palomeras, N. (2012). Inspección visual subacuática mediante robótica submarina. Revista Iberoamericana de Automática e Informática Industrial RIAI, 9, 34-45.

2.1.2.1 Diseño estructural y materiales

Uno de los elementos fundamentales en la arquitectura de un AUV es su diseño estructural. Este debe proporcionar alta resistencia y rigidez, manteniendo al mismo tiempo un peso reducido y un coste razonable. Dado que muchos AUVs presentan flotabilidad neutra, el peso total —incluyendo estructura, fuente de energía y carga útil— debe igualar al desplazamiento del vehículo.

Los materiales utilizados incluyen compuestos de fibra, por su baja densidad, aunque presentan propiedades anisotrópicas y limitaciones en la fabricación de secciones gruesas. También se emplean acero inoxidable y aluminio por su resistencia a la corrosión.

El diseño del casco a presión es clave, ya que actúa como fuente principal de flotabilidad. Se consideran configuraciones como el monocasco sin refuerzo, cilindros con refuerzo anular y materiales cerámicos. La facilidad de fabricación, ensamblaje, modificación o reparación también influyen en el diseño estructural. Algunos AUVs incorporan secciones de aluminio ranuradas para facilitar la integración de nuevas cargas útiles.

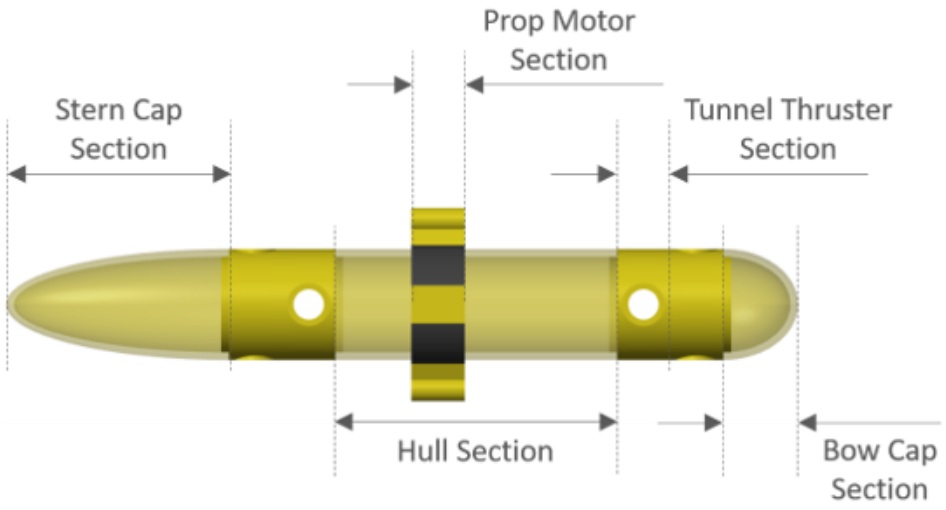

Figura 2 - Estructura de un casco de un AUV

Fuente: Mathworks-robotics/modeling-and-simulation-of-an-AUV-in-Simulink. (2025). [MATLAB]. MathWorks Robotics.

2.1.2.2 Fuente de energía

La **fuente de energía** determina la autonomía del AUV y condiciona el funcionamiento de sensores y sistemas de propulsión. Las baterías secundarias más comunes son las de ion de litio, polímero de litio, níquel-metal hidruro y plomo-ácido de gel. Las baterías de polímero de litio destacan por su alta densidad energética y baja masa.

También se utilizan baterías primarias, como las de manganeso alcalino, especialmente en misiones que requieren mayor duración. Se investigan soluciones como las pilas de combustible, aunque presentan desafíos técnicos y de seguridad. Los volantes de inercia ofrecen alta potencia en intervalos breves, pero no son viables para misiones prolongadas. La energía solar puede emplearse para recargar baterías si el vehículo emerge a la superficie regularmente.

La gestión energética es crítica. Los sistemas de control pueden modificar los parámetros de la misión en función del nivel de energía disponible.

2.1.2.3 Sistema de propulsión

El **sistema de propulsión** permite el movimiento del AUV en el medio acuático. Habitualmente, los vehículos cuentan con propulsores combinados con superficies de control, como alerones, timones y elevadores, para lograr una maniobrabilidad eficiente.

2.1.2.4 Sensores

Los **sensores** proporcionan al AUV información sobre su entorno y sobre su propio estado. Se clasifican en sensores *exteroceptivos* (percepción del entorno) y *propioceptivos* (estado interno del vehículo).

En aplicaciones científicas, se utilizan sensores para estudiar procesos físicos, químicos, geológicos y biológicos. Su integración en el AUV requiere considerar la disponibilidad de espacio, la necesidad de estanqueidad y la neutralidad en flotabilidad. Asimismo, deben

cuidarse los aspectos eléctricos, como la alimentación y el apantallamiento de cables, para evitar interferencias.

2.1.2.5 Sistema de control y arquitectura informática

El **sistema de control y arquitectura informática** actúan como el "cerebro" del AUV. Estos sistemas gestionan su comportamiento autónomo y su interacción con el entorno.

Se emplean plataformas como VMEbus o CompactPCI, aunque también se adoptan arquitecturas distribuidas mediante buses como el CAN bus, que ofrece alta fiabilidad y flexibilidad. En el plano del software, se utilizan arquitecturas jerárquicas, heterárquicas, de subsunción o híbridas.

Se han desarrollado ejecutivos como T-REX y Huxley, orientados a la gestión de misiones y al control del vehículo. El middleware facilita la interoperabilidad entre componentes y la flexibilidad en el desarrollo de sistemas robóticos submarinos.

2.1.2.6 Navegación

La **navegación** permite que el AUV siga una trayectoria planificada y localice puntos de interés o zonas de estudio. Se utilizan sensores inerciales (giroscopios, acelerómetros) combinados con sistemas acústicos de posicionamiento, como USBL (por sus siglas en inglés, *Ultra-Short Baseline*), SBL (por sus siglas en inglés, *Short Baseline*) y LBL (por sus siglas en inglés, *Long Baseline*).

Cuando el vehículo se encuentra en la superficie, puede emplear DGPS (por sus siglas en inglés, *Differential Global Positioning System*) para mejorar la precisión. Técnicas como SLAM (por sus siglas en inglés, *Simultaneous Localization and Mapping*, localización y mapeo simultáneos) permiten construir un mapa del entorno mientras se localiza la posición del AUV.

2.1.2.7 Comunicación

La **comunicación** submarina se realiza principalmente mediante enlaces acústicos, debido a la alta atenuación de las ondas electromagnéticas en el agua. No obstante, el ancho de banda acústico es limitado.

Cuando el AUV emerge o despliega una antena, puede establecer comunicaciones por radio o satélite para la transmisión de datos y recepción de nuevas instrucciones.

2.1.2.8 Sistemas de acoplamiento

En algunas misiones, los AUVs incorporan **sistemas de acoplamiento** que permiten recargar baterías, descargar datos o recibir nuevos planes sin necesidad de extraer el vehículo del agua.

Estos sistemas incluyen mecanismos de orientación, acoplamiento mecánico y eléctrico, alojamiento (*garaging*) y sistemas de lanzamiento y recuperación.

2.1.2.9 Lanzamiento y recuperación

Los **sistemas de lanzamiento y recuperación** son esenciales para la operación segura del AUV. En el caso de vehículos de gran tamaño, se requieren mecanismos específicos para su despliegue y extracción del medio acuático.

2.2 Reconocimiento de patrones

2.2.1 Principios básicos

El reconocimiento de patrones es un campo de la inteligencia artificial que se centra en la identificación y clasificación de patrones en los datos. Los principios básicos de este campo implican la extracción de características relevantes de los datos brutos, la selección de un modelo apropiado para representar los patrones, el entrenamiento de ese modelo utilizando datos etiquetados (en el aprendizaje supervisado) o no etiquetados (en el aprendizaje no supervisado), y finalmente, la evaluación del rendimiento del modelo en datos nuevos.

En entornos submarinos, el reconocimiento de patrones se enfrenta a desafíos únicos que complican la aplicación de los principios básicos. Las imágenes submarinas, una fuente común de datos en este dominio, se ven afectadas por diversos factores:

- **Visibilidad limitada y baja iluminación:** La luz se atenúa y dispersa rápidamente en el agua, lo que resulta en imágenes con bajo contraste, colores desvanecidos y visibilidad reducida.
- **Artefactos visuales y distorsiones:** La refracción de la luz, las partículas en suspensión y el uso de iluminación artificial provocan distorsiones y ruido.
- **Variabilidad en la calidad de la imagen:** Las condiciones del agua, la profundidad y el equipo de captura afectan significativamente la calidad de las imágenes submarinas.
- **Dinamismo de las escenas:** La fauna marina y otros elementos móviles dificultan la extracción de características.
- **Oclusión y densidad de objetos:** Los objetos pueden estar densamente agrupados o parcialmente ocluidos, lo que dificulta su detección y reconocimiento individual.
- **Diferencias de escala:** La detección debe adaptarse a objetos de tamaños muy diversos.
- **Escasez de datos etiquetados:** La obtención y anotación de grandes volúmenes de imágenes submarinas es costosa y compleja, lo que limita el entrenamiento de modelos supervisados.

Para abordar estos desafíos, se emplean estrategias especializadas en cada fase del proceso de reconocimiento:

- **Preprocesamiento y mejora de la imagen:** Se aplican técnicas de mejora de imagen para aumentar el contraste, corregir el color y reducir el ruido.
- **Extracción de características robustas:** Se utilizan descriptores insensibles a variaciones en iluminación y ruido, o se recurre a redes neuronales profundas que aprenden características jerárquicas directamente de los datos. Las convoluciones deformables permiten adaptar los filtros a la geometría de los objetos, mejorando la detección de formas irregulares.
- **Modelado y clasificación:** Se han adaptado modelos clásicos y redes avanzadas como YOLO (*You Only Look Once)*, SSD (por sus siglas en inglés, *Single Shot Detector*) o Faster R-CNN (por sus siglas en inglés, *Region-based Convolutional Neural Network*). Arquitecturas recientes, como Underwater-YOLO, incorporan mecanismos de atención

para gestionar la oclusión y detectar objetos de pequeño tamaño. El uso de atención dual permite combinar características globales y locales, facilitando la detección de objetos parcialmente visibles.

- **Aprendizaje por refuerzo:** Algunos enfoques integran el aprendizaje por refuerzo profundo (DRL, por sus siglas en inglés, *Deep Reinforcement Learning*) para optimizar parámetros de algoritmos de procesamiento de imágenes y obtener mapas de profundidad más precisos en entornos submarinos.
- **Fusión sensorial:** Además de las imágenes ópticas, se pueden utilizar otros sensores como el sonar para obtener información complementaria sobre el entorno submarino, y las técnicas de reconocimiento de patrones pueden aplicarse a estos datos o a la fusión de datos multisensoriales.

2.2.2 Aplicaciones en entornos marinos

El reconocimiento de patrones posee una amplia variedad de aplicaciones en entornos marinos, donde su integración con vehículos submarinos y sistemas de observación resulta crucial para el avance tecnológico y científico.

- **Navegación y Control de Vehículos Submarinos Autónomos (AUVs):** El reconocimiento de patrones es esencial para tareas como el seguimiento de trayectorias, la evitación de obstáculos, el acoplamiento submarino o la implementación de sistemas SLAM (localización y mapeo simultáneos). La detección de objetos proporciona información útil para el control dinámico de la trayectoria de los AUVs.
- **Monitoreo ecológico e investigación de la biodiversidad marina:** El análisis de imágenes submarinas permite identificar organismos marinos y evaluar la salud del ecosistema. Arquitecturas como Underwater-YOLO mejoran la detección de especies pequeñas u ocluidas. La disponibilidad de conjuntos de datos específicos, como CLfish-V1, permite adaptar los modelos al reconocimiento de fauna marina.
- **Inspección de infraestructuras submarinas:** A través de vehículos operados remotamente (ROVs) o AUVs, se inspeccionan tuberías, cables o estructuras sumergidas. El reconocimiento de patrones puede detectar anomalías estructurales, contribuyendo a la planificación del mantenimiento y la prevención de fallos.
- **Seguimiento de la vida marina:** Las técnicas de reconocimiento se aplican al análisis de vídeos para estudiar el comportamiento y desplazamiento de organismos marinos. Esta información es relevante en estudios de comportamiento animal y en la conservación de especies.
- **Seguridad marítima y detección de objetos:** Estas técnicas permiten identificar objetos sumergidos potencialmente peligrosos o llevar a cabo tareas de vigilancia en zonas costeras y portuarias. Si bien no se detallan casos concretos en las fuentes analizadas, las aplicaciones en este ámbito están en expansión.

El reconocimiento de patrones en entornos marinos permite mejorar la autonomía de los vehículos submarinos, optimizar las tareas de inspección y vigilancia, y potenciar el

conocimiento sobre la biodiversidad marina. El desarrollo de arquitecturas adaptadas y el uso de datos multisensoriales continúan ampliando el potencial de estas aplicaciones.

2.3 Redes neuronales y aprendizaje profundo

Las redes neuronales artificiales son modelos computacionales inspirados en el funcionamiento del cerebro humano. Estas estructuras, formadas por capas de nodos interconectados, permiten procesar información compleja y aprender patrones a partir de datos. El aprendizaje profundo (*deep learning*) representa una subdisciplina del aprendizaje automático que utiliza redes neuronales con múltiples capas para aprender representaciones jerárquicas de los datos.

Este enfoque ha demostrado una capacidad notable para resolver problemas tradicionalmente difíciles de formalizar, como el reconocimiento de objetos, rostros o palabras habladas. A diferencia de los métodos que requieren especificar manualmente características relevantes, el aprendizaje profundo permite a las máquinas descubrir por sí mismas las representaciones óptimas para una tarea determinada.

Uno de los avances clave del aprendizaje profundo es su capacidad para construir conceptos complejos a partir de conceptos más simples, a través de una estructura en capas que transforma progresivamente los datos. Esta jerarquía de representaciones mejora significativamente el rendimiento en tareas de clasificación, segmentación y detección de objetos.

Históricamente, la inteligencia artificial logró avances en dominios formales y estructurados, como el ajedrez. Sin embargo, tareas intuitivas para los humanos, como identificar una figura en una imagen, resultaban más desafiantes para las máquinas. El aprendizaje profundo ha cambiado este paradigma, permitiendo que algoritmos superen el rendimiento humano en tareas como clasificación de imágenes, detección de peatones y segmentación semántica.

El éxito reciente del aprendizaje profundo se debe al aumento en la disponibilidad de grandes volúmenes de datos, al avance de los recursos computacionales (especialmente el uso de GPUs, por sus siglas en inglés, *Graphics Processing Units*) y al desarrollo de arquitecturas eficientes y escalables. Además, técnicas como la retropropagación y la representación distribuida han permitido entrenar modelos más profundos y complejos con mayor eficacia.

Actualmente, el aprendizaje profundo tiene aplicaciones en múltiples disciplinas, incluyendo visión por computador, procesamiento del lenguaje natural, diagnóstico médico, robótica y control autónomo. En el contexto marino, su capacidad para extraer características robustas a partir de imágenes degradadas lo convierte en una herramienta idónea para el reconocimiento de patrones en entornos submarinos.

2.3.1 Arquitecturas comunes para visión artificial

La visión artificial, como rama de la inteligencia artificial, se apoya en múltiples arquitecturas de redes neuronales para llevar a cabo tareas como la clasificación de imágenes, la detección de

objetos o la segmentación semántica. Estas arquitecturas han evolucionado significativamente en los últimos años, permitiendo abordar problemas visuales complejos con alta precisión.

Una de las arquitecturas más relevantes en este campo es la red neuronal convolucional (CNN, por sus siglas en inglés *Convolutional Neural Network*). Este tipo de red está diseñado específicamente para procesar datos con estructura espacial, como imágenes, y se inspira en el funcionamiento de la corteza visual del cerebro. Las CNNs resultan particularmente eficaces para analizar información visual donde los patrones pueden encontrarse distribuidos en distintas ubicaciones de la imagen.

Las CNNs se componen principalmente de capas convolucionales y capas de *pooling* (submuestreo). Las capas convolucionales aplican filtros sobre la imagen de entrada para extraer características visuales relevantes, como bordes, texturas o formas. Posteriormente, las capas de *pooling* reducen la resolución espacial de los mapas de características, disminuyendo así el consumo computacional y facilitando la generalización del modelo. Tras varias repeticiones de estas capas, se incorporan capas densas para realizar la clasificación final.

Entre las arquitecturas más utilizadas dentro de las CNNs se encuentran:

- **VGG16:** Es una red profunda con 16 capas que utiliza filtros pequeños (3x3) y operaciones de *pooling* (2x2). A pesar de haber sido superada por modelos más recientes en rendimiento, su diseño sencillo la convierte en una opción útil para tareas base o como punto de partida para redes personalizadas.
- **ResNet50:** Introduce conexiones residuales que permiten saltos entre capas, facilitando el entrenamiento de redes profundas al mitigar el problema del desvanecimiento del gradiente. Esta arquitectura ha demostrado alta eficacia en clasificación y detección de objetos.

En tareas de detección de objetos, la familia YOLO (*You Only Look Once*) destaca por su enfoque de regresión de pasada única, lo que permite realizar detección en tiempo real. YOLO predice simultáneamente múltiples cuadros delimitadores (*bounding boxes*) y las probabilidades asociadas a las clases detectadas. Versiones como YOLOv3, YOLO9000 y recientemente YOLOv11 han mejorado la precisión y la capacidad para detectar múltiples clases, incluyendo aplicaciones en entornos submarinos.

Otras arquitecturas destacadas para detección de objetos incluyen:

- **Single Shot Detector (SSD):** Utiliza una arquitectura en forma de pirámide para identificar objetos a diferentes escalas.
- **R-CNN (por sus siglas en inglés, *Region-based Convolutional Neural Network*) y variantes:** Estas redes generan primero regiones propuestas en la imagen, que luego se analizan con una CNN para clasificar y refinar los cuadros detectados.
- **RetinaNet:** Implementa dos ramas separadas para clasificación y regresión de cuadros delimitadores, mejorando la precisión en objetos de tamaño reducido.

Para la segmentación semántica, donde cada píxel debe clasificarse en una categoría, una de las arquitecturas más eficaces es Mask R-CNN, que combina detección de objetos y segmentación precisa.

Si bien las CNNs dominan muchas aplicaciones de visión, otras arquitecturas también son relevantes:

- **Perceptrón Multicapa (MLP, por sus siglas en inglés, *Multi-Layer Perceptron*):** Aunque más limitado en visión artificial, puede utilizarse en tareas de clasificación de imágenes simples.
- **Transformers:** Originados en el procesamiento de lenguaje natural, los Transformers han demostrado eficacia en tareas visuales gracias a su mecanismo de autoatención, como ocurre en modelos generativos como DALL·E.
- **Redes Generativas Adversarias (GANs, por sus siglas en inglés, *Generative Adversarial Networks*):** Compuestas por un generador y un discriminador, estas redes permiten crear imágenes sintéticas realistas y tienen aplicaciones en síntesis de datos, restauración de imágenes y generación de contenido visual.

En conjunto, la elección de la arquitectura depende de la naturaleza de la tarea, las condiciones del entorno de trabajo y los recursos computacionales disponibles. Las CNNs, junto con arquitecturas especializadas como YOLO y GANs, constituyen la base del desarrollo de sistemas de visión artificial robustos, especialmente en escenarios complejos como los entornos submarinos.

2.3.2 La familia YOLO (You Only Look Once)

La familia de arquitecturas YOLO (*You Only Look Once*) representa un conjunto de modelos revolucionarios en el campo de la visión artificial, diseñados específicamente para la detección y segmentación de objetos en tiempo real. Desarrollada por Ultralytics como proyecto de código abierto, esta tecnología se caracteriza por su enfoque de predicción en un solo paso, diferenciándose de los métodos tradicionales basados en múltiples etapas.

YOLO divide la imagen de entrada en una cuadrícula y asigna a cada celda la tarea de predecir un conjunto fijo de cuadros delimitadores (*bounding boxes*) junto con sus puntuaciones de confianza. Durante el entrenamiento, un único predictor de cuadro es responsable de cada objeto, según la mayor intersección sobre unión (IoU, por sus siglas en inglés, *Intersection over Union*) con la verdad de terreno. Posteriormente, se aplica un proceso de supresión no máxima (NMS, por sus siglas en inglés, *Non-Maximum Suppression*) para eliminar detecciones redundantes, mejorando así la precisión.

Las principales ventajas de YOLO son:

- Alta velocidad de procesamiento, adecuada para aplicaciones en tiempo real.
- Buena precisión en la detección de objetos de diferentes tamaños.

- Facilidad de uso y entrenamiento, gracias a la implementación simplificada por Ultralytics.
- Soporte activo y mejoras constantes por parte de la comunidad de desarrollo.
- Reducción de falsos positivos en comparación con métodos basados en propuestas de regiones.

Entre las limitaciones identificadas se encuentran:

- Dificultad en la detección de objetos solapados.
- Rendimiento ligeramente inferior en tareas específicas en comparación con detectores como Faster R-CNN.

Desde su introducción en 2015, YOLO ha evolucionado significativamente a través de diversas versiones que han mejorado su velocidad, precisión y capacidad de detección

- **YOLOv2 (YOLO9000):** Mejoró la precisión y la velocidad, incorporando una nueva arquitectura Darknet-19 y aumentando el rango de clases detectables.
- **YOLOv3:** Introdujo Darknet-53 como arquitectura base, predicción en múltiples escalas y clasificadores logísticos independientes.
- **YOLOv4:** Optimizó el entrenamiento en paralelo con la estructura Backbone (CSPDarknet53), Neck (PANet) y Head (YOLOv3), e incorporó técnicas como *Bag of Specials* y *Bag of Freebies*.
- **YOLOv6:** Introdujo el uso de "dense anchor boxes" para mejorar la precisión en la generación de cuadros de anclaje.
- **YOLOv7:** Mejoró la detección de objetos de diferentes formas y tamaños mediante la utilización de nueve cajas de anclaje.
- **YOLOv8:** Incorporó una nueva API para entrenamiento e inferencia optimizados en CPU y GPU, manteniendo compatibilidad con versiones anteriores.
- **YOLOv11:** Introdujo mejoras adicionales en métricas como mAP (por sus siglas en inglés, *mean Average Precision*), puntuación F1, precisión e IoU, aplicadas en escenarios de detección en imágenes y vídeos.

La versatilidad de YOLO permite su aplicación en numerosos sectores:

- **Vehículos autónomos:** Detección de peatones, señales y otros vehículos.
- **Seguridad:** Monitoreo de cámaras de vigilancia para identificación de comportamientos anómalos.
- **Sanidad:** Análisis de imágenes médicas y detección de patologías.
- **Retail:** Control de inventarios y análisis de comportamiento de clientes.
- **Análisis de grandes volúmenes de datos:** Aplicaciones en redes sociales y minería de datos visuales.

El uso de técnicas de *transfer learning* permite adaptar modelos preentrenados a conjuntos de datos específicos, reduciendo los tiempos de entrenamiento y mejorando la precisión en tareas especializadas.

En conjunto, la familia YOLO ha establecido un estándar de referencia en la detección de objetos gracias a su combinación de velocidad, eficiencia y precisión. Su evolución continua y la participación de la comunidad aseguran su relevancia en aplicaciones tecnológicas cada vez más exigentes.

2.3.3 Transfer learning y sus beneficios

El *transfer learning* (aprendizaje por transferencia) constituye una de las técnicas más relevantes del aprendizaje profundo para el desarrollo de sistemas de inteligencia artificial. Su principio básico consiste en aprovechar el conocimiento adquirido por un modelo en una tarea inicial y aplicarlo en una tarea diferente, pero relacionada, ajustando únicamente ciertas capas de la red. Esta metodología ha demostrado ser especialmente eficaz en campos como la visión artificial y el procesamiento del lenguaje natural.

El origen del *transfer learning* se encuentra en algoritmos de aprendizaje automático diseñados para abordar tareas múltiples de forma simultánea. Posteriormente, su formalización estadística permitió una aplicación más generalizada, considerando dominios de características, tareas definidas numéricamente y funciones de probabilidad asociadas.

En el contexto del aprendizaje profundo, el *transfer learning* se implementa descubriendo arquitecturas de redes neuronales que facilitan el aprendizaje y reutilizando los parámetros preentrenados como valores iniciales para nuevas tareas. Esto permite economizar tiempo de entrenamiento y reducir los requerimientos computacionales.

El procedimiento general de transferencia de aprendizaje consiste en seleccionar un modelo previamente entrenado sobre un conjunto de datos extenso y generalista. Posteriormente, este modelo se ajusta utilizando un conjunto de datos específico al nuevo dominio, adaptando las capas finales para reconocer las nuevas clases de interés. Durante el ajuste, es habitual "congelar" las capas iniciales de la red para conservar los pesos ya aprendidos, mientras que solo las capas superiores son entrenadas de nuevo.

Entre los principales beneficios del *transfer learning* se destacan:

- Reducción significativa del tiempo y coste computacional en el entrenamiento de nuevos modelos.
- Disminución de la cantidad de datos necesarios para entrenar el modelo en la nueva tarea.
- Mejora en la capacidad de generalización del modelo para tareas específicas.
- Facilidad para adaptar arquitecturas complejas, previamente entrenadas en grandes conjuntos de datos, a dominios concretos.

En el ámbito de la visión artificial, el *transfer learning* se aplica habitualmente mediante el uso de arquitecturas preentrenadas como ResNet50 o VGG16 sobre bases de datos extensas como ImageNet. Estas redes se adaptan para nuevas tareas de clasificación, detección o segmentación

de objetos, ajustando únicamente sus capas superiores para acomodarse a las particularidades del nuevo conjunto de datos.

El *transfer learning* representa un avance fundamental dentro del aprendizaje automático, facilitando el desarrollo de aplicaciones prácticas en numerosos campos mediante el aprovechamiento eficiente de conocimientos previos y reduciendo las barreras de entrada para la creación de modelos especializados.

2.4 Entornos submarinos: retos para el procesamiento visual

El procesamiento visual en entornos submarinos presenta desafíos técnicos únicos que no se encuentran en otros escenarios terrestres o aéreos. Estos desafíos derivan de las propiedades físicas del medio acuático y afectan directamente la adquisición, interpretación y análisis de imágenes.

2.4.1 Propiedades físicas del medio submarino

La luz sufre una atenuación significativa al penetrar en el agua. La absorción y dispersión provocan una pérdida progresiva de intensidad y un cambio en las características cromáticas de la imagen. Las longitudes de onda más largas (como el rojo) se absorben rápidamente, dejando predominancia de tonalidades verdes y azules a medida que aumenta la profundidad.

La turbidez del agua, causada por partículas en suspensión, incrementa la dispersión de la luz y genera ruido visual. Asimismo, los reflejos y refracciones pueden distorsionar la geometría de los objetos captados.

2.4.2 Factores que afectan la captura de imágenes

- **Visibilidad reducida:** La distancia a la que se pueden distinguir objetos disminuye considerablemente, afectando la capacidad de obtener imágenes claras y definidas.
- **Iluminación limitada:** La ausencia de luz solar a grandes profundidades exige el uso de fuentes de iluminación artificial, que pueden introducir artefactos como sombras duras, brillos y distorsiones.
- **Movimiento del entorno:** Las corrientes marinas, las partículas flotantes y los organismos en movimiento dificultan la estabilidad de las capturas de imagen.
- **Oclusión parcial de objetos:** La acumulación de sedimentos y la disposición tridimensional del entorno submarino dificultan la detección completa de estructuras y organismos.

2.4.3 Implicaciones para el procesamiento visual

Estos factores exigen el uso de técnicas de preprocesamiento especializadas antes de aplicar algoritmos de detección o clasificación. Entre ellas se incluyen:

- Corrección de color para restaurar el equilibrio cromático.

- Mejora de contraste para resaltar los contornos de los objetos.
- Eliminación de ruido mediante filtros adaptativos.
- Uso de métodos robustos a la distorsión, como redes neuronales convolucionales deformables.

Además, las arquitecturas de visión artificial deben adaptarse para reconocer objetos con formas irregulares, tamaños variables y baja diferenciación respecto al fondo.

2.4.4 Avances recientes y estrategias específicas

La aplicación de técnicas de *transfer learning*, el entrenamiento de modelos en conjuntos de datos sintéticos y la fusión de datos ópticos con sensores acústicos (como sonar) constituyen estrategias recientes para mejorar la percepción visual submarina.

Modelos como Underwater-YOLO han adaptado arquitecturas clásicas de detección para incluir mecanismos de atención que compensan la baja visibilidad y la oclusión, mejorando la detección de objetos pequeños y parcialmente ocultos.

El desarrollo de redes específicas para condiciones submarinas continúa siendo un área activa de investigación, con el objetivo de optimizar el procesamiento visual en misiones de búsqueda, monitoreo ambiental, inspección de infraestructuras y arqueología subacuática.

Capítulo 3. Análisis del problema y requisitos del sistema

3.1 Problemas específicos del entorno submarino

El entorno submarino presenta una serie de desafíos inherentes que dificultan la operación y navegación de los vehículos autónomos submarinos (AUVs). Uno de los principales problemas es la ausencia de propagación de señales electromagnéticas bajo el agua, lo que impide el uso de sistemas de localización basados en redes de balizas de largo alcance, como el Sistema de Posicionamiento Global (GPS) utilizado en superficie o en tierra firme.

Aunque existen equivalentes acústicos, estos sistemas presentan limitaciones significativas tanto en alcance como en precisión. Además, su funcionamiento requiere el despliegue previo de balizas cuidadosamente ubicadas, restringiendo el rango operativo del vehículo al área definida entre ellas. El equipo de sonar, aunque útil para proporcionar datos de rango y correspondencia topográfica, presenta una resolución insuficiente para lograr una navegación precisa a escala submétrica.

La visión artificial ofrece un enfoque potencialmente más preciso para el posicionamiento submarino, siempre que exista una representación adecuada del entorno. Sin embargo, su aplicabilidad se ve limitada a distancias cortas debido a las condiciones adversas de visibilidad e iluminación. La dinámica del fondo marino, como la presencia de algas en movimiento, introduce variabilidad adicional que complica tanto el procesamiento de imágenes como la creación de mapas estáticos fiables.

La obtención de una posición precisa del AUV constituye, por tanto, un desafío significativo en estos entornos. Los sistemas orientados a liberar la operación humana de tareas de bajo nivel, como la planificación de rutas y la evitación de obstáculos, deben afrontar estas limitaciones físicas y ambientales. En este sentido, el desarrollo de sistemas de navegación inteligentes que integren la adquisición de datos y el proceso de navegación representa un área de creciente interés, con el objetivo de reducir los costos operativos y ampliar el acceso a esta tecnología.

La naturaleza no estructurada del entorno submarino, junto con la presencia de elementos en movimiento, complica la tarea de construir representaciones ambientales precisas, como mosaicos de imágenes del fondo marino. Además, los AUVs suelen estar sujetos a restricciones de movimiento no holonómicas, impuestas por la disposición de sus propulsores, lo que limita su capacidad de maniobra.

Finalmente, las distorsiones ópticas propias de las lentes submarinas, especialmente en los bordes de las imágenes, deben ser consideradas tanto en la creación de mosaicos como en los algoritmos de navegación basados en visión. Asimismo, las diferencias entre las imágenes adquiridas en tiempo real y los mosaicos preconstruidos, derivadas de la falta de planitud del entorno y de las variaciones en la iluminación, afectan negativamente la precisión del emparejamiento visual.

3.2 Requisitos funcionales y no funcionales

3.2.1 Requisitos funcionales

El sistema deberá permitir la detección y clasificación en tiempo real de patrones visuales en entornos submarinos, con especial énfasis en la identificación de personas. Asimismo, se realizará la simulación completa del sistema en MATLAB, integrando el proceso de adquisición de imágenes, procesamiento de datos y generación de señales de notificación basadas en los resultados de la detección.

- Detección y clasificación en tiempo real de objetos submarinos (personas, peces, objetos no vivos).
- Utilización del sistema de reconocimiento con modelo de simulación de un AUV en Simulink.
- Capacidad de procesar imágenes capturadas por cámaras embarcadas en el vehículo.
- Generación de salidas de datos compatibles con los sistemas de navegación y decisión del AUV.

3.2.2 Requisitos no funcionales

El sistema simulado deberá ser capaz de procesar y reconocer correctamente las imágenes capturadas en condiciones submarinas adversas. Además, deberá garantizar la compatibilidad e interoperabilidad entre el entorno de procesamiento desarrollado en Python y la plataforma de simulación en MATLAB/Simulink. El modelo base utilizado, proporcionado por MATLAB, será adaptado para incluir el módulo de reconocimiento de patrones, asegurando su modularidad y su capacidad de ampliación futura.

- El sistema debe mantener un tiempo de respuesta inferior a X segundos para cada imagen.
- El consumo de recursos debe ser compatible con la capacidad de procesamiento embarcada.
- La red de detección debe ser robusta frente a condiciones de baja visibilidad y ruido visual.
- El sistema debe permitir futuras extensiones (por ejemplo, añadir nuevas clases de objetos sin necesidad de reentrenar toda la red).
- Compatibilidad de formatos de salida con MATLAB/Simulink.

3.3 Especificaciones del sistema integrado

El sistema desarrollado en este proyecto está compuesto por tres módulos principales: el modelo del AUV implementado en MATLAB/Simulink, el sistema de procesamiento visual desarrollado en Python y el mecanismo de interacción entre ambos entornos.

El modelo de simulación se basa en una plataforma proporcionada por MATLAB, adaptada para incluir entradas externas correspondientes a la detección de patrones visuales. Este modelo permite simular el comportamiento del vehículo autónomo en escenarios submarinos variados.

El sistema de procesamiento visual se ha implementado en Python utilizando la arquitectura YOLOv11n, entrenada previamente mediante el *framework* de Ultralytics. El modelo final, almacenado en formato .pt, se utiliza para la detección de personas en imágenes submarinas. La detección se realiza mediante un *script* de Python que analiza imágenes individuales capturadas durante la simulación.

La interacción entre MATLAB y Python se establece a través de un proceso de intercambio controlado: MATLAB simula la obtención de imágenes submarinas y guarda los archivos correspondientes. Posteriormente, MATLAB invoca a la función de detección en Python, pasando la ruta de la imagen como argumento. Python procesa la imagen utilizando el modelo YOLOv11n y, en caso de detección de personas, devuelve a MATLAB un mensaje que contiene el identificador del objeto detectado y su posición en la imagen (coordenadas del cuadro delimitador). Estos resultados son empleados en la simulación para activar comportamientos del AUV, como cambios de trayectoria o la simulación de misiones de búsqueda y rescate.

El sistema se ha diseñado siguiendo un enfoque modular, que permite la futura incorporación de nuevas clases de objetos o la integración de datos provenientes de otros sensores. Además, se ha considerado la eficiencia computacional para asegurar que el procesamiento visual en Python pueda ejecutarse de manera compatible con los tiempos de la simulación en Simulink.

Capítulo 4. Metodología

4.1 Flujo de trabajo general del proyecto

El desarrollo de este proyecto se ha estructurado siguiendo una secuencia de fases que abarcan desde la definición inicial de los objetivos hasta la validación del sistema integrado en el entorno de simulación. A continuación, se describe de manera general el flujo de trabajo seguido:

1. **Definición de objetivos:** Se identificó la necesidad de integrar un sistema de reconocimiento de patrones en un vehículo submarino autónomo, centrándose en la detección de personas en entornos submarinos.
2. **Investigación teórica y selección de tecnologías:** Se realizó un estudio de las tecnologías disponibles para la detección de patrones submarinos, seleccionándose la arquitectura YOLOv11n como base para el desarrollo del sistema de procesamiento visual.
3. **Entrenamiento del modelo de detección:** Se llevó a cabo el entrenamiento de un modelo YOLO utilizando *datasets* (conjuntos de datos) específicos para la detección de personas en entornos acuáticos. El modelo fue entrenado y validado en Python.
4. **Desarrollo del sistema de comunicación MATLAB-Python:** Se implementó un mecanismo de intercambio de datos que permite a MATLAB enviar imágenes a Python, donde son procesadas por el modelo de detección, y recibir de vuelta los resultados de clasificación.
5. **Integración en el modelo de simulación:** El sistema de detección se integró en un modelo base de Simulink, permitiendo 1ue los resultados de la clasificación de patrones se incorporasen en la trayectoria del AUV simulada en el modelo de MATLAB/Simulink
6. **Validación y pruebas del sistema:** Se llevaron a cabo simulaciones de misiones de búsqueda y rescate para validar el comportamiento del sistema integrado y evaluar su rendimiento en condiciones submarinas simuladas.
7. **Documentación y análisis de resultados:** Finalmente, se elaboró la documentación del trabajo, incluyendo el análisis de los resultados obtenidos y la identificación de posibles mejoras futuras.

4.2 Preparación y análisis del dataset

4.2.1 Selección del dataset

4.2.1.1 Resumen y Características del Dataset:

El conjunto de datos seleccionado para este proyecto es el denominado "Drowning Detection", disponible en la plataforma Roboflow. Este *dataset* está específicamente enfocado en la detección de personas en riesgo de ahogamiento en entornos acuáticos.

Se seleccionó la versión v15, caracterizada por una distribución de imágenes del 80% para entrenamiento (2551 imágenes), 10% para validación (317 imágenes) y 10% para prueba (327 imágenes), alcanzando un total de 3195 imágenes. Esta organización facilita el entrenamiento, validación y prueba del modelo de manera estructurada.

El *dataset* ofrece una amplia variedad de formatos de anotación y archivos de configuración, permitiendo su uso con diversos modelos de detección de objetos populares. Entre los principales formatos disponibles se encuentran:

- **YOLO:** Admite versiones como YOLOv11, YOLOv9, YOLOv8, YOLOv5 y YOLOv7, con anotaciones en formato TXT y archivos YAML de configuración. También incluye el formato Darknet TXT para YOLOv3 y YOLOv4.
- **COCO JSON:** Compatible con *frameworks* como EfficientDet PyTorch y Detectron2.
- **Pascal VOC XML:** Un formato XML ampliamente utilizado para la gestión local de datos.
- **TFRecord:** Formato binario empleado en TensorFlow 1.5 y TensorFlow 2.0.
- **PaliGemma JSONL:** Formato orientado al ajuste de modelos multimodales como PaliGemma de Google.
- **CreateML JSON:** Compatible con las herramientas CreateML y Turi Create de Apple.

En cuanto al preprocesamiento aplicado, las imágenes fueron auto-orientadas y redimensionadas a un tamaño de 640x640 píxeles, ajustándolas dentro del marco con la adición de bordes negros cuando fue necesario. No se aplicaron técnicas de aumento de datos adicionales en esta versión del conjunto.

4.2.1.2 Motivos para su selección

El *dataset* "Drowning Detection" se consideró adecuado para este proyecto por las siguientes razones:

1. **Foco en la detección de personas en entornos acuáticos:** Dado su objetivo principal de detección de ahogamientos, el *dataset* proporciona una cantidad significativa de imágenes de personas en el agua, lo cual es fundamental para el propósito de este proyecto.
2. **Variedad de formatos de anotación:** La disponibilidad de múltiples formatos facilita la integración con diferentes arquitecturas de modelos de detección de objetos, como la familia YOLO.
3. **División estructurada de datos:** La separación clara en conjuntos de entrenamiento, validación y prueba permite un desarrollo y evaluación sistemática del modelo.
4. **Preprocesamiento básico implementado:** La existencia de un preprocesamiento inicial facilita su utilización directa, aunque podrían ser necesarios preprocesamientos adicionales específicos para mejorar la calidad de imágenes submarinas (como la corrección de color o la eliminación de neblina).

4.2.2 Anotación y formato

Al seleccionar el modelo YOLOv11 para este proyecto, el conjunto de datos se utilizó en formato de anotación TXT, acompañado de un archivo de configuración en formato YAML.

El formato de anotación TXT utilizado por YOLO representa cada objeto detectado mediante una línea en un archivo de texto, cuyo nombre coincide con el de la imagen correspondiente, pero con extensión .txt. Cada línea del archivo contiene cinco valores separados por espacios: el

índice numérico de la clase, las coordenadas del centro del cuadro delimitador (*centro_x* y *centro_y*), y el ancho y la altura del cuadro (*ancho* y *alto*). Todos los valores están normalizados entre 0 y 1, con relación al tamaño total de la imagen.

El archivo de configuración YAML proporciona la información necesaria para utilizar correctamente el conjunto de datos en el entrenamiento o la inferencia de modelos YOLOv11. Este archivo especifica las rutas de los directorios de imágenes de entrenamiento y validación, el archivo que contiene los nombres de las clases, y el número total de clases presentes en el conjunto de datos. En algunos casos, puede incluir información adicional relacionada con los parámetros de entrenamiento, aunque aspectos como la arquitectura del modelo suelen definirse en archivos independientes.

El *dataset* "Drowning Detection" proporciona para cada imagen su correspondiente archivo .txt de anotaciones, con las coordenadas normalizadas y el identificador de clase. Asimismo, incluye un archivo .yaml que facilita su integración con modelos basados en la familia YOLO, permitiendo así su uso directo en el flujo de entrenamiento establecido para este proyecto.

4.3 Entrenamiento del modelo de reconocimiento

4.3.1 Arquitectura seleccionada

Para este proyecto se ha seleccionado la arquitectura YOLOv11, la última iteración de la serie YOLO, caracterizada por importantes mejoras en su diseño estructural y en los métodos de entrenamiento respecto a versiones anteriores.

La arquitectura de YOLOv11 mantiene la organización típica de los modelos YOLO, estructurada en tres bloques principales: *backbone*, *neck* y *head*. Cada uno de estos bloques cumple funciones específicas dentro del proceso de detección.

El *backbone* se encarga de extraer características de la imagen de entrada a diferentes escalas mediante capas convolucionales especializadas. YOLOv11 introduce una arquitectura mejorada del *backbone* para optimizar la extracción de características, aumentando la precisión en la detección de objetos complejos y mejorando la eficiencia del modelo.

El *neck* actúa como puente entre el *backbone* y la *head*, integrando características de distintas escalas. En YOLOv11, se utilizan estructuras refinadas que combinan velocidad y rendimiento, destacando la incorporación de un bloque SPPF (*Spatial Pyramid Pooling Fast*) y el bloque C2PSA (*Convolutional Block with Parallel Spatial Attention*). Estos módulos permiten mejorar la representación multiescala de los datos y potenciar la atención espacial sin afectar significativamente el tiempo de inferencia.

La *head* es la responsable de generar las predicciones finales. Utiliza convoluciones *depthwise* para reducir la complejidad computacional, similar a lo implementado en versiones anteriores como YOLOv10. La *head* produce directamente las cajas delimitadoras y las probabilidades de clase para cada objeto detectado.

Entre las principales novedades arquitectónicas de YOLOv11 destacan los siguientes bloques:

- **C3k2 Block:** Introducido como sustitución del bloque C2f de YOLOv8, este módulo permite capturar características más complejas gracias a una estructura interna de tres bloques convolucionales combinados con múltiples *bottlenecks*.
- **C2PSA Block:** Un avanzado módulo de auto-atención que mejora la capacidad de modelado global, facilitando la detección precisa incluso en entornos complejos como el medio submarino.
- **DWConv (*Depthwise Convolution*):** Utilizado para aligerar las operaciones convolucionales en la *head* de detección, mejorando así la velocidad de procesamiento.

La arquitectura de YOLOv11 consta de ocho etapas, utilizando el bloque C3k2 como componente principal. El proceso de *downsampling* (reducción del tamaño de los datos de la imagen para facilitar su procesamiento) se realiza mediante convoluciones con kernel 3×3 y stride 2, y se mantiene el uso del módulo SPPF en el neck para potenciar el aprendizaje de características a múltiples escalas.

En términos de eficiencia, YOLOv11m logra una mayor precisión media (mAP) en el conjunto de datos COCO utilizando un 22% menos de parámetros que YOLOv8m, reflejando la eficacia de los avances introducidos en el diseño de la red.

YOLOv11 está diseñado para ser adaptable a diferentes entornos de despliegue, incluyendo dispositivos embebidos, plataformas en la nube y sistemas equipados con GPU. Además, soporta tareas como la detección de objetos, la segmentación de instancias, la clasificación de imágenes, la estimación de pose y la detección de objetos orientados (OBB, por sus siglas en inglés, *Oriented Bounding Box*).

Cabe destacar que, debido a la reciente aparición de YOLOv11, aún no se dispone de una publicación académica formal que documente exhaustivamente su arquitectura.

En resumen, YOLOv11 representa una evolución significativa de las versiones anteriores, enfocándose en una extracción de características más eficaz, una mayor eficiencia de procesamiento y una amplia versatilidad, aspectos especialmente relevantes para su aplicación en entornos submarinos, donde la precisión y la rapidez en la detección son factores críticos.

4.3.2 Proceso de entrenamiento y validación

El proceso de entrenamiento y validación del modelo YOLOv11n se llevó a cabo utilizando el entorno Python junto con la librería Ultralytics. El objetivo principal fue adaptar un modelo preentrenado a las características específicas del conjunto de datos "Drowning Detection v15", centrado en la detección de personas en entornos acuáticos.

4.3.2.1 Entrenamiento

Se utilizó como punto de partida el modelo preentrenado YOLOv11n, cargado mediante el módulo Ultralytics. El entrenamiento se configuró para ejecutarse durante 10 épocas, utilizando imágenes de tamaño 640x640 píxeles. El optimizador empleado fue SGD (por sus siglas en inglés,

Stochastic Gradient Descent, Descenso de Gradiente Estocástico), siguiendo la configuración predeterminada de Ultralytics, y el tamaño del lote (*batch size*) se mantuvo en los valores automáticos ajustados al *hardware* disponible.

El proceso de entrenamiento se basó en el archivo de configuración *data.yaml*, que definía la localización de los conjuntos de datos de entrenamiento, validación y prueba. La división de los datos siguió el esquema 80% para entrenamiento, 10% para validación y 10% para prueba, tal como estaba establecido en el conjunto original.

Durante el entrenamiento, se aplicaron técnicas automáticas de augmentación de datos como rotaciones, cambios de escala y ajustes de brillo, destinadas a mejorar la capacidad de generalización del modelo.

4.3.2.2 Validación

La validación del modelo YOLOv11n se llevó a cabo de forma automática, al concluir cada época durante el proceso de entrenamiento. Esta fase de evaluación se realizó sobre el subconjunto de validación del *dataset* "Drowning Detection v15", que representa el 10% del total de imágenes (317 en este caso), y se mantuvo completamente separado del conjunto de prueba final para asegurar una evaluación imparcial.

El archivo *data.yaml*, esencial para el entrenamiento, contenía las rutas de los conjuntos de datos de entrenamiento, validación y prueba, además de especificar el número de clases. En cada ciclo de entrenamiento, el rendimiento del modelo fue evaluado minuciosamente con las métricas estándar de detección de objetos, tal como define Ultralytics:

- **Precisión:** Mide la proporción de verdaderos positivos respecto al total de predicciones positivas realizadas. Nos indica, en esencia, cuántas de las detecciones del modelo fueron realmente correctas.
- **Recall:** Refleja la capacidad del modelo para identificar todos los objetos relevantes presentes en la imagen, es decir, cuántos de los objetos que *debía* detectar fueron efectivamente encontrados.
- **mAP@0.5 (*mean Average Precision* con solapamiento del 50%):** Representa el promedio de la precisión cuando el solapamiento entre el cuadro predicho y el objeto real es de al menos el 50%. Es una métrica crucial que nos da una idea general de la calidad de la detección.
- **mAP@0.5:0.95 (*mean Average Precision* con solapamiento variable):** Esta métrica es más estricta y ofrece una visión más completa, ya que calcula la mAP promediando los resultados a través de diferentes umbrales de solapamiento, que van desde el 50% hasta el 95% en incrementos de 0.05.

El propósito principal de este proceso de validación continua fue monitorizar la evolución del modelo a lo largo de las épocas de entrenamiento y detectar cualquier indicio de sobreajuste o estancamiento en el aprendizaje. De hecho, se observaron mejoras consistentes en las métricas durante las primeras épocas, seguidas de una fase de estabilización en los valores. Este patrón es un buen indicio de que el modelo alcanzó una convergencia adecuada y un ajuste razonable a los datos, sin memorizar excesivamente el conjunto de entrenamiento.

En paralelo, se generaron automáticamente gráficos de las curvas de pérdida (tanto para la caja delimitadora, o *box loss*, como para la clasificación, o *classification loss*), además de las curvas de precisión y *recall*. Estas representaciones visuales fueron fundamentales para evaluar de manera intuitiva la progresión del aprendizaje y confirmar que el modelo estaba generalizando de forma adecuada, sin desviaciones anómalas que pudieran comprometer su rendimiento.

Este proceso de validación continua resultó ser un pilar fundamental para garantizar que el modelo final no solo fuera capaz de aprender a detectar patrones relevantes en el conjunto de entrenamiento, sino que también demostrara una robusta capacidad para generalizar y ofrecer un rendimiento fiable en imágenes que no había visto previamente.

4.3.2.3 Observaciones durante el entrenamiento

Durante las primeras épocas se observó una tendencia a la mejora progresiva tanto en la reducción de las pérdidas (*box loss*, *classification loss*) como en el incremento de las métricas de precisión y *recall*. Al finalizar las 10 épocas de entrenamiento, los valores de mAP y *recall* se estabilizaron, indicando la convergencia del modelo.

No obstante, se detectaron ciertos errores de clasificación en imágenes con condiciones de baja visibilidad o con oclusiones parciales, situaciones típicas en entornos acuáticos reales. Estos resultados abren la posibilidad de futuras mejoras mediante técnicas de augmentación específicas o ajustes adicionales en la arquitectura del modelo.

El modelo final obtenido fue considerado adecuado para su integración en el sistema de simulación, logrando un equilibrio entre precisión, *recall* y eficiencia computacional, factores fundamentales para su implementación práctica en misiones de búsqueda y rescate.

4.4 Evaluación del modelo

4.4.1 Métricas utilizadas

Para evaluar el desempeño del modelo de detección de objetos en entornos submarinos, se emplearon métricas estándar ampliamente reconocidas en el ámbito de la visión artificial y el aprendizaje profundo:

- **Precisión:** La precisión mide la proporción de predicciones correctas positivas sobre el total de predicciones positivas realizadas. Es decir, de todas las veces que el modelo indicó la presencia de una persona, cuántas veces acertó realmente. Una alta precisión implica una baja tasa de falsos positivos.
- **Recall:** El *recall* (o sensibilidad) mide la capacidad del modelo para detectar correctamente todos los objetos relevantes presentes en las imágenes. Representa la proporción de verdaderos positivos detectados respecto al total de verdaderos positivos existentes. Un alto *recall* indica que se pierden pocas detecciones reales.
- **mAP@0.5 (*mean Average Precision* at IoU 0.5):** El mAP@0.5 es el promedio de precisión obtenido cuando se considera una predicción correcta si el solapamiento (IoU,

Intersection over Union) entre la caja predicha y la real es de al menos un 50%. Es una métrica comúnmente utilizada para evaluar la calidad general de la detección de objetos.

- **mAP@0.5-0.95:** Esta métrica extiende la evaluación de mAP calculándola sobre múltiples umbrales de solapamiento, que varían de 0.5 a 0.95 en incrementos de 0.05. Proporciona una evaluación más rigurosa de la precisión del modelo en diferentes niveles de exigencia, reflejando su capacidad para localizar objetos de manera precisa y consistente.

4.4.2 Resultados preliminares

Concluido el proceso de entrenamiento y validación, se procedió a un análisis detallado de los resultados obtenidos, empleando diversas métricas de rendimiento y herramientas gráficas. Una de las visualizaciones más reveladoras fue, sin duda, la matriz de confusión normalizada. Esta matriz permite observar con gran claridad la precisión del modelo a la hora de clasificar correctamente las distintas clases presentes en el conjunto de validación.

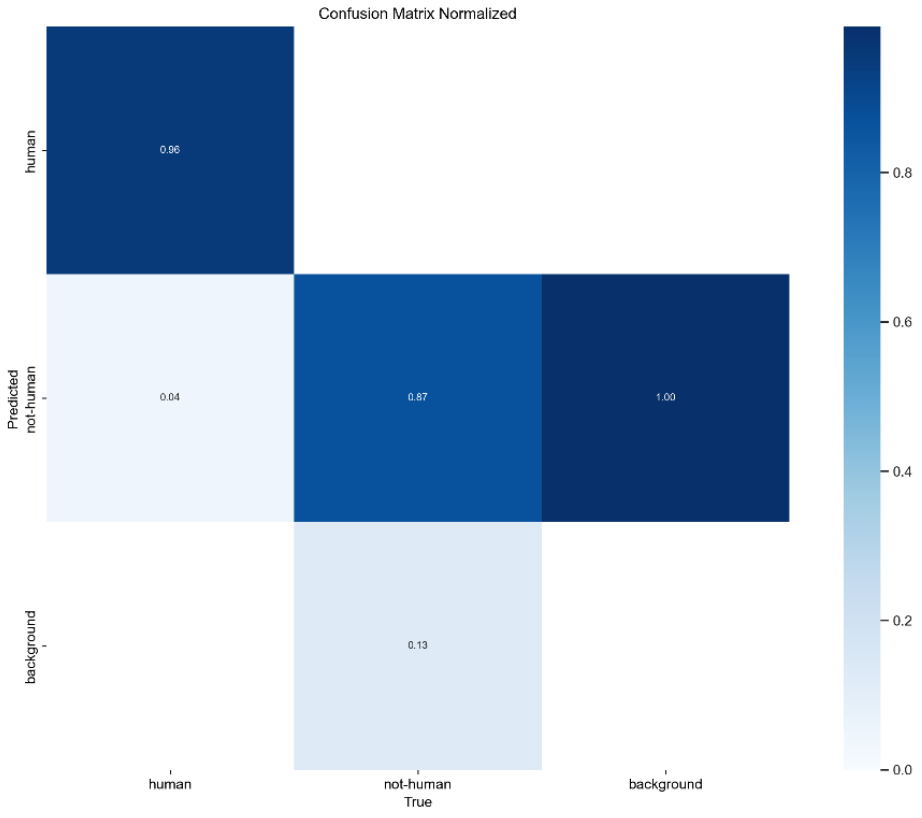

Figura 3 - Matriz de confusión normalizada para el modelo YOLOv11n.
Representa la proporción de aciertos y errores en la clasificación de las diferentes clases.

Al observar esta matriz, se aprecia cómo el modelo distingue de forma efectiva entre las clases "human" y "not-human", mostrando un porcentaje de aciertos realmente elevado en ambas. De hecho, la clase "human" fue identificada correctamente en un impresionante 96% de los casos. Por su parte, la clase "not-human" también obtuvo una tasa de acierto muy sólida, un 87%, lo

que es un rendimiento notable si consideramos las condiciones tan particulares de los entornos acuáticos, que suelen ser un verdadero desafío para la visión artificial.

A continuación, se resumen los principales resultados cuantitativos obtenidos por el modelo:

- **Precisión (90%):** De todas las ocasiones en las que el modelo predijo la presencia de una persona, el 90% fueron aciertos. Esto es un indicador muy positivo, pues significa que el sistema generó una tasa muy baja de falsos positivos.

- **Recall (87%):** El modelo fue capaz de detectar el 87% de todas las personas que realmente estaban presentes en las imágenes. Aunque es cierto que no identificó el 100% de los casos, logró detectar una gran mayoría de ellos.

- **mAP@0.5 (*mean Average Precision* con solapamiento del 50%) (85%):** Esta métrica, que evalúa el promedio de precisión exigiendo al menos un 50% de solapamiento entre la predicción y la anotación real, alcanzó un 85%. Un valor así de elevado nos indica una excelente capacidad para localizar personas de manera adecuada.

- **mAP@0.5–0.95 (*mean Average Precision* con solapamiento variable) (65%):** Esta es una métrica más exigente, ya que promedia distintos niveles de solapamiento. El 65% obtenido demuestra un rendimiento aceptable, si bien nos muestra que aún hay margen de mejora en la detección precisa bajo criterios más estrictos de localización.

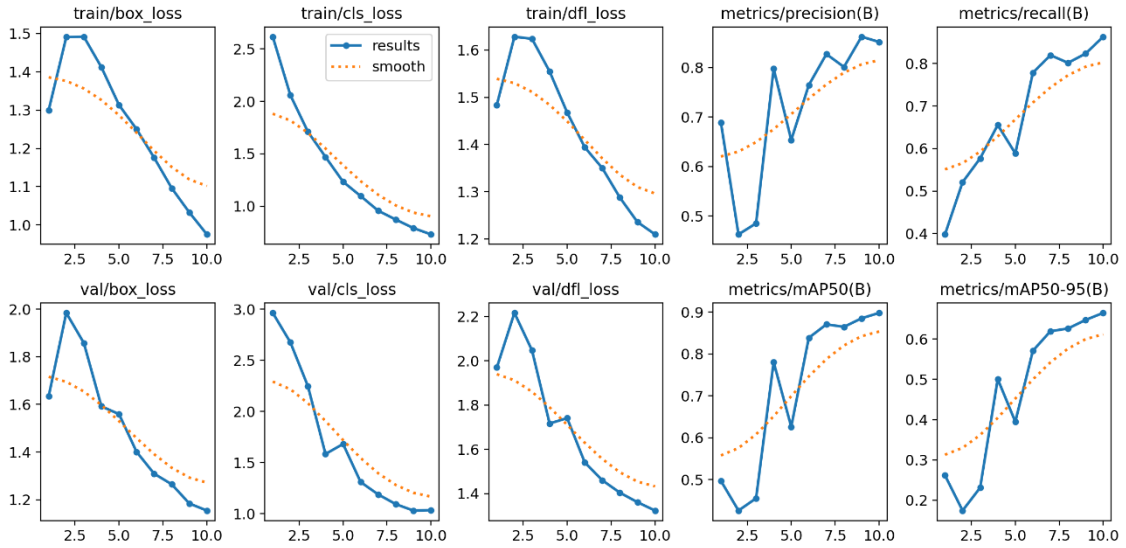

Figura 4 - Evolución de las métricas durante el entrenamiento del modelo.
Se muestra la evolución de las pérdidas, precisión, recall y mAP en las distintas épocas de entrenamiento y validación.

En definitiva, estos resultados confirman la eficacia general del modelo YOLOv11n que hemos entrenado para la detección de personas en entornos submarinos, teniendo en cuenta las dificultades inherentes al medio. No obstante, existen situaciones particulares —como una visibilidad extremadamente baja u oclusiones importantes— en las que el rendimiento puede disminuir. Esto, lejos de ser un impedimento, abre la puerta a futuras optimizaciones y mejoras para el sistema.

Capítulo 5. Modelo del sistema del AUV

5.1 Modelo de Simulink del vehículo

Para la simulación del comportamiento del vehículo submarino autónomo (AUV), se ha utilizado una trayectoria tridimensional extraída de un modelo de ejemplo proporcionado por MATLAB. La información del recorrido del AUV extraída del modelo utilizado para la simulación en MATLAB/Simulink ha sido utilizada para incorporar el sistema de reconocimiento de patrones.

La trayectoria extraída define una serie de puntos en el espacio tridimensional por los que el AUV virtual debe desplazarse. Esta información se ha integrado cuidadosamente en un *script* propio, que simula la navegación del vehículo y la adquisición de imágenes de su entorno. Durante este trayecto, se han establecido dos puntos específicos donde el AUV realiza una detención simulada, con el propósito de analizar imágenes. Esto, sin duda, reproduce un comportamiento típico en tareas de búsqueda y rescate, donde la pausa para la observación es crucial.

Este enfoque permite una implementación más sencilla. Además, facilita enormemente la integración del módulo de visión, que ha sido desarrollado en Python, mediante una comunicación fluida entre ambos entornos. Esta interacción se detalla con mayor profundidad en el apartado siguiente.

5.2 Interfaz entre Python y Matlab

La comunicación entre MATLAB y Python es, sin duda, un componente fundamental en el sistema que hemos propuesto. Es lo que permite integrar el modelo de detección de personas desarrollado en Python con la simulación del recorrido del vehículo submarino autónomo (AUV) en MATLAB.

El sistema se articula mediante tres scripts principales:

- Un script de MATLAB denominado trayectoria.m, que simula el desplazamiento del AUV a lo largo de una trayectoria predefinida.
- Una función auxiliar en MATLAB, denominada analizar_imagen.m, que gestiona la comunicación entre MATLAB y Python.
- Un script de Python, que ejecuta el modelo de detección basado en YOLOv11 y devuelve los resultados de la predicción.

El flujo de ejecución se organiza del siguiente modo:

1. **Ejecución del recorrido simulado:** El script trayectoria.m inicia la simulación del movimiento del AUV en un entorno 3D. A lo largo de su trayecto, se definen dos puntos concretos donde el vehículo "se detiene" para simular la adquisición de una imagen.

2. **Análisis de la imagen capturada:** En los puntos definidos, trayectoria.m llama a la función analizar_imagen.m, encargada de enviar la imagen capturada al entorno Python. Esta imagen representa el entorno del AUV en ese momento.

3. **Procesamiento en Python:** El script de Python procesa la imagen utilizando el modelo entrenado con YOLOv11. Si el modelo detecta una persona, genera un archivo de salida con la información relevante: identificador del objeto, coordenadas de la *bounding box* y nivel de confianza.

4. **Recepción del resultado y decisión en MATLAB:** La función analizar_imagen.m recupera el resultado generado por Python, lo interpreta y lo convierte a un formato comprensible por MATLAB. En función de este resultado, trayectoria.m decide si el AUV continúa su recorrido o si finaliza la simulación, emulando así una posible situación de detección de una persona y parada del vehículo.

Este enfoque modular permite mantener separados los componentes de visión artificial y simulación, favoreciendo la escalabilidad del sistema. Asimismo, garantiza la interoperabilidad entre MATLAB y Python mediante una estructura clara, en la que cada componente tiene una responsabilidad bien definida.

5.3 Sistema de visión

El sistema de visión implementado en este proyecto se basa en un enfoque modular y robusto para la detección de personas en entornos acuáticos, utilizando el modelo YOLOv11n combinado con un algoritmo de seguimiento de objetos (SORT). Todo el proceso se desarrolla en un script de Python diseñado para recibir imágenes desde MATLAB, procesarlas, y devolver los resultados de detección.

El componente central es una clase denominada **DetectorTracker**, la cual encapsula tanto la carga del modelo como el procesamiento visual. Esta clase incluye los siguientes elementos clave:

- **Carga del modelo YOLOv11n:** A través del paquete Ultralytics, se carga el modelo previamente entrenado para la detección de personas. Este modelo se ha ajustado mediante *transfer learning* sobre un conjunto de datos específico de personas en entornos acuáticos.

- **Redimensionamiento inteligente de imágenes:** Antes del análisis, se verifica si la imagen excede un tamaño máximo predefinido. En tal caso, se redimensiona proporcionalmente para optimizar el rendimiento del modelo sin perder relación de

aspecto. Esta etapa mejora la eficiencia en equipos con capacidad computacional limitada.

- **Procesamiento de detecciones:** Se aplica el modelo YOLOv11n a la imagen procesada. Para cada detección se extraen las coordenadas de la **bounding box**, la confianza de la predicción y la clase del objeto. Solo se consideran válidas aquellas detecciones con confianza superior a un umbral predefinido y cuya clase sea 'person' o 'human'.

- **Seguimiento de objetos (SORT):** Las detecciones válidas se introducen en un sistema de seguimiento basado en el algoritmo SORT, que asigna identificadores únicos a cada objeto detectado. Esto permite mantener la coherencia entre imágenes consecutivas y facilita el seguimiento temporal.

- **Visualización:** Como paso opcional, el sistema muestra visualmente la imagen analizada con las bounding boxes superpuestas, junto con la etiqueta de clase y la puntuación de confianza, permitiendo verificar la detección de manera interactiva.

- **Devolución de resultados:** El sistema devuelve a MATLAB una lista con los objetos detectados, incluyendo su identificador, posición (coordenadas de la caja) y nivel de confianza. Esta información es posteriormente utilizada en la simulación para decidir la evolución del comportamiento del AUV.

Este sistema de visión constituye un bloque fundamental dentro del modelo general, ya que permite dotar al vehículo submarino simulado de la capacidad de percibir su entorno y reaccionar en función de la presencia de personas, contribuyendo directamente a la simulación de misiones de rescate.

Además, para el seguimiento de objetos a lo largo de secuencias de vídeo, se ha integrado el algoritmo SORT (*Simple Online and Realtime Tracking*), desarrollado por Bewley (2016). Este algoritmo utiliza un modelo de movimiento basado en filtro de Kalman y la asociación de detecciones mediante la métrica IoU (Intersection over Union). En este proyecto ha sido adaptado para trabajar juntamente con el sistema de detección basado en YOLOv11n. SORT ha sido seleccionado por su eficiencia en aplicaciones en tiempo real y se encuentra distribuido bajo la licencia GPLv3.

5.4 Pruebas en entorno simulado

Con el objetivo de evaluar el funcionamiento del sistema propuesto, se ha desarrollado un entorno simulado en MATLAB que representa la trayectoria de un vehículo submarino autónomo (AUV) y su interacción con un sistema externo de visión artificial implementado en Python. Las pruebas realizadas buscan verificar la integración funcional del sistema, así como la capacidad del AUV para responder adecuadamente ante la detección de personas en imágenes capturadas durante su recorrido.

El entorno simulado está compuesto por tres bloques principales:

- **Generación de trayectoria (MATLAB):** El archivo trayectoria.m contiene el código encargado de simular el movimiento del AUV a lo largo de una ruta tridimensional predefinida. Esta trayectoria fue extraída y adaptada a partir de un modelo de Simulink de referencia proporcionado por MathWorks. A lo largo del recorrido, se han definido dos puntos de análisis en los cuales el AUV "captura" imágenes ficticias representadas por archivos de imagen previamente seleccionados.
- **Módulo de análisis de imagen (MATLAB):** En cada uno de los puntos de análisis, el script principal llama a la función analizar_imagen.m Esta función es la que se encarga de gestionar toda la comunicación con el módulo de visión en Python. Recibe la imagen simulada, la transfiere al sistema de detección y espera a la respuesta.
- **Procesamiento de visión (Python):** Una vez recibida la imagen, el script de Python basado en la clase DetectorTracker realiza la detección de personas utilizando el modelo YOLOv11n. Si se identifica al menos una persona, el sistema devuelve a MATLAB la posición y el identificador (ID) del objeto detectado. Esta respuesta es interpretada por la función analizar_imagen.m y utilizada por el script principal para decidir si el AUV debe continuar su recorrido o detenerse simulando una actuación de emergencia o rescate.

Durante las pruebas, se simularon distintas condiciones visuales utilizando imágenes con variaciones en la iluminación y en la visibilidad. El sistema fue capaz de detectar correctamente personas, activando de manera adecuada la respuesta del AUV en el entorno MATLAB.

Este enfoque ha permitido validar el flujo de comunicación entre MATLAB y Python, así como comprobar la correcta integración del sistema de visión artificial con el modelo de simulación. Los resultados obtenidos son prometedores y confirman la viabilidad de emplear técnicas de visión por computador para apoyar misiones autónomas en entornos submarinos simulados.

Capítulo 6. Resultados

6.1 Resultados de la detección en videos de prueba

Una vez entrenado el modelo YOLOv11n con el conjunto de datos *Drowning Detection*, se procedió a evaluar su rendimiento sobre vídeos externos no utilizados durante el entrenamiento. El objetivo fue comprobar la capacidad del modelo para detectar personas en entornos acuáticos, simulando escenarios reales de búsqueda y rescate.

Durante las pruebas, el modelo demostró una capacidad robusta para identificar correctamente la presencia de personas en diferentes condiciones visuales. La red fue capaz de generar *bounding boxes* bien posicionadas alrededor de los individuos, incluso en presencia de oleaje, iluminación variable o ángulos de cámara no ideales. En la Figura 5 se muestran algunas capturas de vídeo con las detecciones realizadas por el modelo, evidenciando la funcionalidad del sistema.

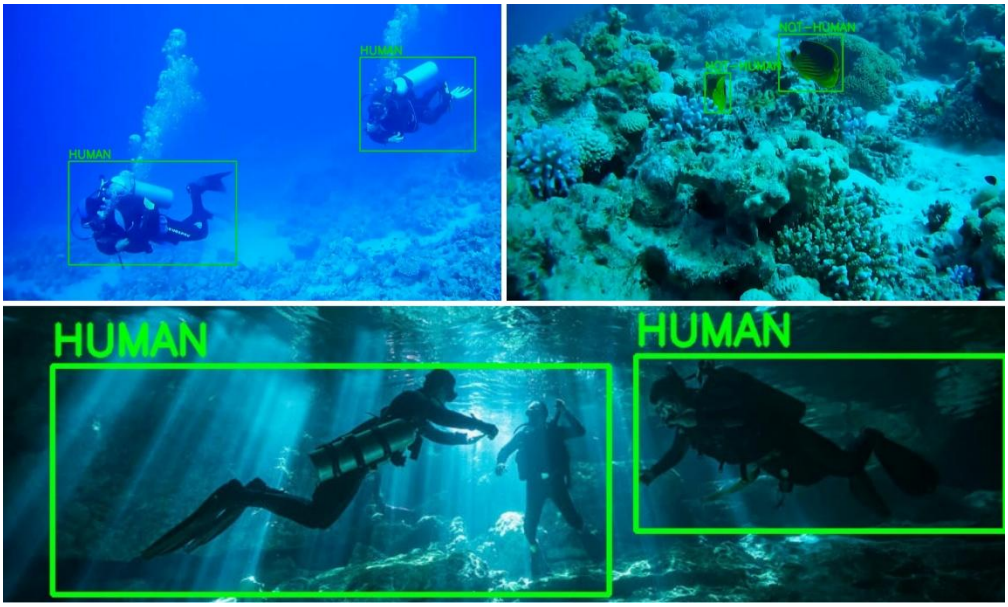

Figura 5 - Capturas de detección del modelo YOLOv11n en vídeos de prueba

6.2 Evaluación de la implementación en Matlab

Durante la simulación realizada en MATLAB, el sistema integrado mostró un comportamiento estable y coherente en la detección de personas a partir de imágenes obtenidas en puntos específicos del recorrido del AUV. En esta fase de prueba, se ejecutó el flujo completo de análisis: captura de imagen, envío a Python para su procesamiento mediante YOLOv11n, y recepción de resultados por parte de MATLAB para decidir la continuidad o interrupción del trayecto simulado.

En la Figura 6 se presenta el estado inicial de la simulación, con el AUV avanzando por la trayectoria sin haber alcanzado aún los puntos de parada.

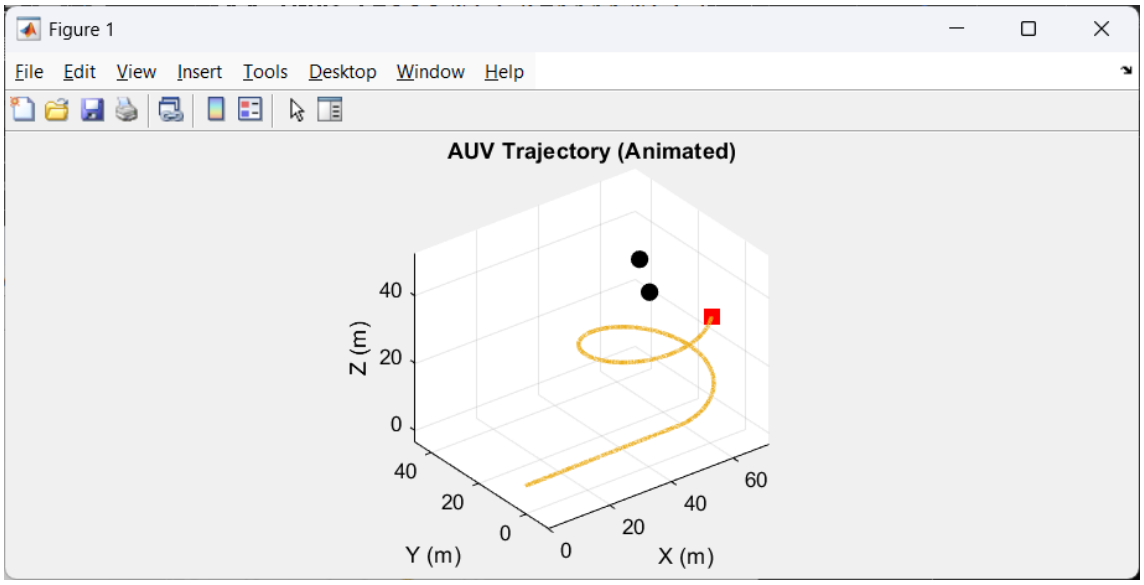

Figura 6 - Recorrido inicial del AUV antes de alcanzar los puntos de parada.

Una vez alcanzado el primer punto de análisis, el sistema detectó que NO era una persona. Mostrando que se reconoce este caso.Tal como se muestra en la Figura 7, el sistema detuvo el recorrido, se visualizó la imagen analizada con su correspondiente cuadro delimitador, y se notificó el resultado en el Command Window de MATLAB con los datos de identificación y posición del objeto detectado en caso de que se detecte un humano.

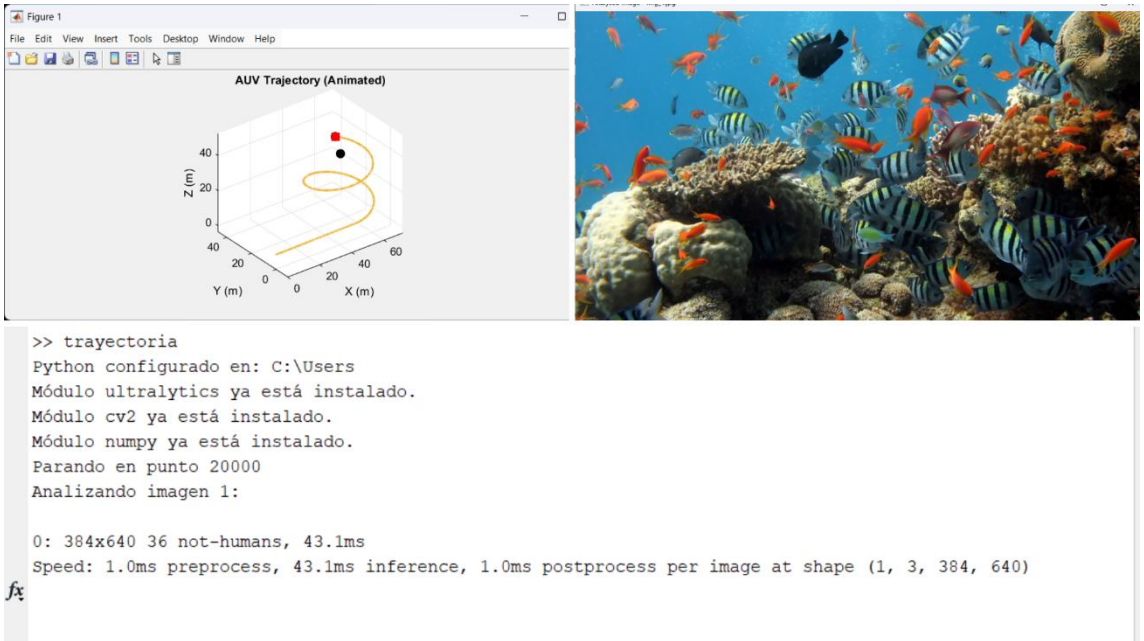

Figura 7 - Imagen analizada en el primer punto de parada. No se detecta una persona.

El mismo proceso se repitió en el segundo punto de parada, detectando una persona en este caso.La Figura 8 muestra la segunda imagen procesada, junto con la respuesta del sistema en MATLAB.

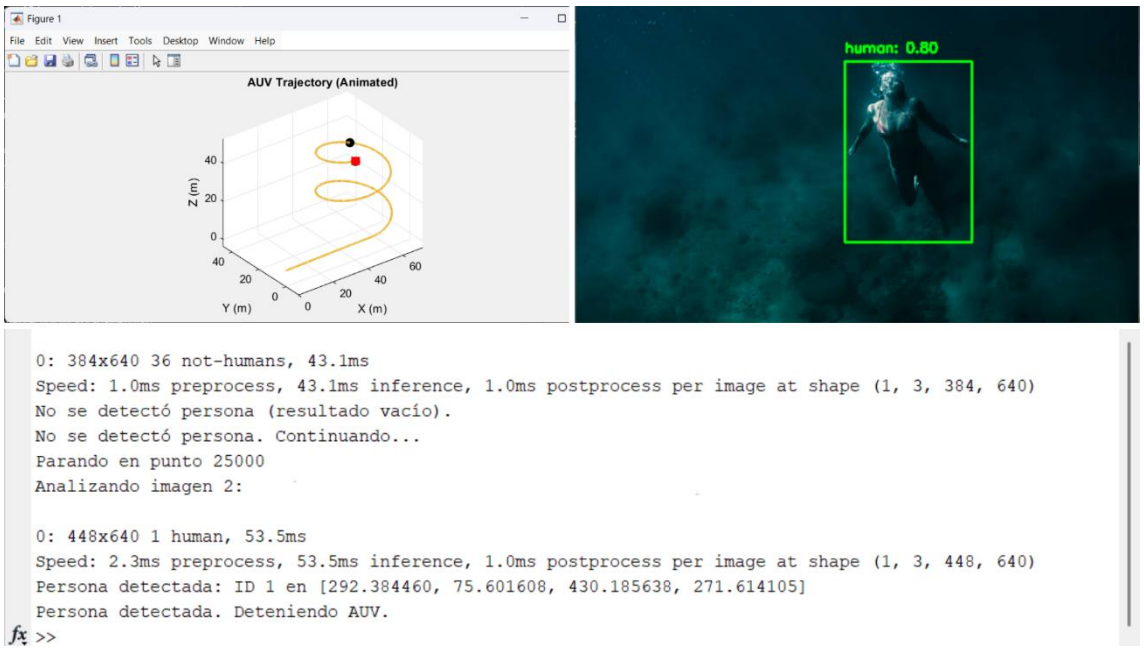

Figura 8 - Segunda imagen capturada y analizada. Se detecta una persona.

Cabe destacar que, aunque el modelo fue entrenado para distinguir entre las clases *human* y *not-human*, en esta implementación se decidió mostrar únicamente las detecciones correspondientes a personas. Esta elección se debe a razones de claridad visual en el entorno gráfico, ya que mostrar todas las clases simultáneamente generaba una gran cantidad de cuadros que dificultaban la interpretación de las imágenes durante la simulación.

6.3 Observación de las clases detectadas

Durante la implementación del sistema de visión en la simulación MATLAB, se ha optado por mostrar exclusivamente la detección de la clase *"human"*, a pesar de que el modelo entrenado está capacitado para reconocer también la clase *"not-human"*. Esta decisión se ha tomado de forma intencionada con el objetivo de simplificar la visualización y facilitar la interpretación de los resultados durante la simulación.

En pruebas previas, se observó que permitir la visualización de todas las clases disponibles en el modelo generaba una cantidad excesiva de cajas delimitadoras (*bounding boxes*), lo que provocaba una imagen sobrecargada y dificultaba el seguimiento visual de los objetos realmente relevantes para la misión, en este caso, las personas.

Por tanto, aunque el sistema mantiene internamente la capacidad de detectar distintas clases, solo se presentan aquellas que corresponden a seres humanos, alineándose así con el propósito principal del proyecto: la detección de personas en entornos acuáticos con fines de búsqueda y rescate.

Capítulo 7. Conclusiones y trabajos futuros

7.1 Conclusiones del proyecto

Este Trabajo de Fin de Grado ha abordado la integración de un sistema de reconocimiento de patrones basado en visión artificial en un vehículo submarino autónomo (AUV), con el objetivo de detectar personas en entornos acuáticos como parte de misiones de búsqueda y rescate. Para ello, se ha empleado un modelo YOLOv11n, previamente entrenado mediante transfer learning sobre un dataset específico, y se ha implementado su funcionamiento dentro de un entorno simulado desarrollado en MATLAB.

El sistema ha demostrado una capacidad adecuada para detectar personas en imágenes submarinas, obteniendo métricas satisfactorias de precisión, recall y mAP. La conexión entre MATLAB y Python ha permitido establecer un flujo de trabajo funcional, en el que el AUV simulado genera imágenes, las transmite para su análisis y responde en función de los resultados obtenidos.

En resumen, se ha logrado demostrar la viabilidad de integrar un sistema de visión artificial moderno en un entorno de simulación para AUVs, sentando las bases para desarrollos más avanzados en entornos reales o semirrealistas.

7.2 Posibles mejoras

Durante el desarrollo del proyecto se han identificado diversas áreas susceptibles de mejora:

- Aumento de la robustez del modelo ante condiciones adversas, tales como baja visibilidad, oclusión parcial o presencia de reflejos. Para ello, sería recomendable aplicar técnicas de aumento de datos más específicas para imágenes submarinas o entrenar el modelo con conjuntos de datos más variados.
- Optimización del rendimiento en tiempo real, adaptando la arquitectura del modelo para entornos con recursos computacionales limitados, como AUVs reales operando con hardware embebido.
- Automatización de la comunicación entre MATLAB y Python, incluyendo la detección continua sin necesidad de intervención manual y el almacenamiento de resultados de forma estructurada para su análisis posterior.
- Mejor gestión de la trayectoria del AUV en función de los resultados de detección, incorporando lógica de decisión más compleja que permita replanificación de la misión o activación de acciones específicas según el número o la posición de las personas detectadas.

7.3 Extensiones futuras del sistema

El sistema desarrollado puede ser ampliado y adaptado en futuras líneas de trabajo. Algunas de las más relevantes incluyen:

- Integración de sensores adicionales como cámaras térmicas o multiespectrales, que podrían mejorar la detección en condiciones extremas o en aguas turbias.
- Incorporación de sistemas de sonar, que permitirían detectar y ubicar personas o cuerpos en situaciones donde la visión artificial no sea suficiente. Esta extensión abriría nuevas posibilidades en entornos con nula visibilidad o a grandes profundidades, permitiendo la fusión de información visual y acústica para una percepción más completa del entorno.
- Implementación en un AUV físico, adaptando el modelo y el flujo de comunicación a los sistemas de control y procesamiento de un vehículo real, validando el sistema en pruebas de campo.
- Uso de redes neuronales más avanzadas o especializadas, como modelos orientados a segmentación de personas, reconocimiento de poses o incluso análisis de comportamiento en el agua.
- Ampliación del modelo para la detección de residuos en el entorno marino, con el objetivo de dotar al sistema de un propósito dual: búsqueda de personas y vigilancia ambiental. Esta mejora permitiría identificar objetos flotantes no deseados o basura, contribuyendo a tareas de preservación del ecosistema marino mediante la integración de criterios de sostenibilidad ambiental.

Estas futuras ampliaciones podrían aumentar significativamente la capacidad de los AUVs para llevar a cabo misiones críticas de búsqueda y rescate en condiciones reales, consolidando el uso de la inteligencia artificial como herramienta de apoyo en robótica submarina.

BIBLIOGRAFÍA

ardian. (2024). Drowning detection Dataset. En *Roboflow Universe*. Roboflow. https://universe.roboflow.com/ardian-yyb0l/drowning-detection-b3lye

Bewley, A., Ge, Z., Ott, L., Ramos, F., & Upcroft, B. (2016). Simple online and realtime tracking. *2016 IEEE International Conference on Image Processing (ICIP)*, 3464-3468. https://doi.org/10.1109/ICIP.2016.7533003

CANO, I. R. (2020, agosto 19). *El transfer learning y las redes convolucionales*. https://www.viewnext.com/transfer-learning-y-redes-convolucionales/

Carreras, M., Ridao, P., Garcia, R., Romagós, D., & Palomeras, N. (2012). Inspección visual subacuática mediante robótica submarina. *Revista Iberoamericana de Automática e Informática Industrial RIAI*, *9*, 34-45. https://doi.org/10.1016/j.riai.2011.11.011

Deep learning. (s. f.). http://www.deeplearningbook.org

Design, Modeling, and Simulation of Autonomous Underwater Vehicles. (s. f.). Recuperado 27 de mayo de 2025, de https://es.mathworks.com/videos/design-modeling-and-simulation-of-autonomous-underwater-vehicles-1619636864529.html

Gracias, N. R., van der Zwaan, S., Bernardino, A., & Santos-Victor, J. (2003). Mosaic-based navigation for autonomous underwater vehicles. *IEEE Journal of Oceanic Engineering*, *28*(4), 609-624.

Griffiths, G. (Ed.). (2002). *Technology and Applications of Autonomous Underwater Vehicles* (0 ed.). CRC Press. https://doi.org/10.1201/9780203522301

Guinea, I. (2025, abril 11). *Crea tu primera Inteligencia Artificial de detección: Guía con YOLO - Blog SEO, Diseño Web & Gráfico*. https://carontestudio.com/blog/crea-tu-primera-ia-de-deteccion-guia-practica-yolo/

Hidayatullah, P., Syakrani, N., Sholahuddin, M. R., Gelar, T., & Tubagus, R. (s. f.). *YOLOv8 to YOLO11: A Comprehensive Architecture In-depth Comparative Review*. https://arxiv.org/pdf/2501.13400

Juan Manuel, C. C., & José Guillermo, G. M. in. (2020). *Desarrollo de un algoritmo de navegación autónoma para uavs basado en objetivos dados usando técnicas de aprendizaje por refuerzo profundo*. Universidad Santo Tomás.

Li, Z., Zheng, B., Chao, D., Zhu, W., Li, H., Duan, J., Zhang, X., Zhang, Z., Fu, W., & Zhang, Y. (2024). Underwater-Yolo: Underwater Object Detection Network with Dilated Deformable Convolutions and Dual-Branch Occlusion Attention Mechanism. *Journal of Marine Science and Engineering*, *12*(12), 2291. https://doi.org/10.3390/jmse12122291

Mathworks-robotics/modeling-and-simulation-of-an-AUV-in-Simulink. (2025). [MATLAB]. MathWorks Robotics. https://github.com/mathworks-robotics/modeling-and-simulation-of-an-AUV-in-Simulink (Obra original publicada en 2021)

Rozada Raneros, S. (2021). *Estudio de la arquitectura YOLO para la detección de objetos mediante deep learning* [Info:eu-repo/semantics/masterThesis]. http://uvadoc.uva.es/handle/10324/45359

Siciliano, B., & Khatib, O. (Eds.). (2016). *Springer Handbook of Robotics*. Springer International Publishing. https://doi.org/10.1007/978-3-319-32552-1

Wadoo, S., & Kachroo, P. (2017). *Autonomous underwater vehicles: Modeling, control design and simulation*. CRC Press.

ANEXOS

trayectoria.m

```
% Configurar entorno Python

configurar_python();



% Obtener la ruta del script actual para cargar archivos relativos

current_script_folder = fileparts(mfilename('fullpath'));



% Inicializar contador de imágenes analizadas

contador_img = 1;



% Cargar datos

% Usamos fullfile para construir la ruta a sim_data.mat

sim_data_path = fullfile(current_script_folder, 'sim_data.mat');

if ~exist(sim_data_path, 'file')

    error('No se encontró el archivo sim_data.mat en: %s', sim_data_path);

end

load(sim_data_path); % Carga sim_data.mat



if exist('sim_data', 'var')

    Xdata = sim_data.DataX;

    Ydata = sim_data.DataY;

    Zdata = sim_data.DataZ;

else

    Xdata = DataX;

    Ydata = DataY;

    Zdata = DataZ;
```

```
end


% Índices donde se detiene a analizar imagen

stop_idx = [20000, 25000];


% Crear figura

figure;

hold on;

grid on;

axis equal;

view(3);  % Vista 3D

title('AUV Trajectory (Animated)', 'Color', 'k');

xlabel('X (m)', 'Color', 'k');

ylabel('Y (m)', 'Color', 'k');

zlabel('Z (m)', 'Color', 'k');

set(gca, 'XColor', 'k', 'YColor', 'k', 'ZColor', 'k', 'GridColor', [0.4, 0.4, 0.4]);


% Marcar puntos de parada

plot3(DataX(stop_idx), DataY(stop_idx), DataZ(stop_idx), 'ko', 'MarkerSize', 8,
'MarkerFaceColor', 'k');


% Inicializar el AUV y la línea recorrida

auv = plot3(NaN, NaN, NaN, 'rs', 'MarkerSize', 10, 'MarkerFaceColor', 'r');

recorrido = plot3(NaN, NaN, NaN, 'Color', [0.93, 0.69, 0.13], 'LineWidth', 2);


% Inicializar trayectoria recorrida

X_tray = [];
```

```
Y_tray = [];

Z_tray = [];


for i = 1:length(DataX)

   % Añadir punto a la trayectoria

   X_tray(end+1) = DataX(i);

   Y_tray(end+1) = DataY(i);

   Z_tray(end+1) = Zdata(i);


   % Actualizar la visualización

   set(recorrido, 'XData', X_tray, 'YData', Y_tray, 'ZData', Z_tray);

   set(auv, 'XData', Xdata(i), 'YData', Ydata(i), 'ZData', Zdata(i));

   drawnow;


   % Parada para análisis

   if any(i == stop_idx)

      fprintf('Parando en punto %d\n', i);


      % Crear nombre con contador

      imagen_nombre = sprintf('img_%d.jpg', contador_img);

      % Construir la ruta completa a la imagen de forma relativa

      imagen_path = fullfile(current_script_folder, imagen_nombre);


      fprintf('Analizando imagen %d: %s\n', contador_img, imagen_path);


      % Aumentar el contador después de preparar la ruta de la imagen

      contador_img = contador_img + 1;
```

```
    % Llamar a analizar_imagen con la ruta completa y relativa
    result = analizar_imagen(imagen_path);



    if result
        disp('Persona detectada. Deteniendo AUV.');
        break;
    else
        disp('No se detectó persona. Continuando...');
    end
  end
  pause(0.00000000000001);  % Velocidad del movimiento
end
```

configurar_python.m

```
function configurar_python()

% CONFIGURAR_PYTHON - Configura el entorno Python y asegura que los paquetes necesarios estén instalados.

% Ruta al ejecutable de Python

 % Asumimos que el ejecutable de Python estará en una subcarpeta 'PythonEnv' dentro

 % de la carpeta del proyecto, o que se instalará allí si se crea un instalador.

 % Es una buena práctica que tu entorno Python se encuentre dentro de tu proyecto.

 % Aquí usaremos una ruta relativa más genérica. Si el instalador de Python

 % lo pone en un lugar específico, deberás ajustarlo.

 % Para un ejecutable creado con PyInstaller, el Python estará "dentro" del exe.

 % Para el Application Compiler de MATLAB, es posible que no necesites especificarlo

 % de esta manera, ya que el MCR no se basa en un Python externo preinstalado.

 % Pero si es para ejecutar durante el desarrollo o si el Python es externo,

 % una opción sería usar la ruta por defecto de Anaconda si está instalada en el sistema,

 % o que el usuario la especifique.

 % Por ahora, para ser más robustos con un instalador, esta función puede ser menos crítica

 % si el Python está 'empaquetado'. Si es para desarrollo, la ruta absoluta de Anaconda está bien.



 % **Opción 1: Mantener la ruta de Anaconda si se asume que estará instalada**

 % pyPath = 'C:\Users\tu_ruta\python.exe';

 % **Opción 2: Usar el python por defecto del sistema o el encontrado por MATLAB (más flexible para empaquetado)**

 % MATLAB intentará encontrar una instalación de Python. Si estás creando un standalone,

 % esto es menos relevante porque el entorno Python en tiempo de ejecución se gestiona

 % de forma diferente por el empaquetador (MATLAB o PyInstaller).

 % Para que sea más robusto al empaquetar, es mejor NO intentar fijar una ruta absoluta aquí.
```

```
% La gestión de la versión de Python para la compilación de MATLAB es diferente a la

% de la ejecución de PyInstaller.


% Para el contexto de empaquetado, si usas `py.` dentro de MATLAB,

% MATLAB ya habrá configurado su puente con Python.

% Lo importante es que los módulos se puedan importar.

% Esta función es más crítica para la **configuración inicial de desarrollo**

% y para asegurar que los **módulos estén instalados**.

% En un entorno empaquetado, los módulos ya deberían estar presentes.


% Si el usuario no tiene Python o Anaconda, esta función NO lo instalará,

% solo los módulos si la ruta de Python es válida.

%En este ejemplo se usa la opción 2



  % Obtener la ruta del script actual (aunque aquí no la usaremos directamente para pyPath)

  current_script_folder = fileparts(mfilename('fullpath'));


  current = pyenv;

  if isempty(current.Executable) % Si MATLAB no ha encontrado un entorno Python

      disp("MATLAB no ha detectado un entorno Python. Por favor, asegúrese de tener Python instalado y configurado en el PATH o use 'pyenv' para especificarlo.");

      % En un ejecutable compilado, esto significa que el MATLAB Runtime no encontró un Python compatible.

      % Para desarrollo, necesitarías configurar tu entorno Python manualmente con pyenv('Version', 'ruta/a/python.exe').

      return; % No podemos continuar si no hay Python

  else
```

```
    pyPath = current.Executable; % Usar el Python que MATLAB ya ha encontrado
    fprintf("Python configurado en: %s\n", pyPath);
  end

  % Lista de módulos necesarios
  requiredModules = {'ultralytics', 'cv2', 'numpy'}; % Añadido 'cv2' explícitamente

  % Instalar los que falten
  for i = 1:length(requiredModules)
    module = requiredModules{i};
    if ~pyModuleExists(module)
      fprintf("Instalando módulo de Python: %s...\n", module);
      % Asegurarse de que `pyenv().Executable` esté entre comillas dobles
      % para manejar rutas con espacios.
      [status, cmdout] = system(sprintf('"%s" -m pip install %s', pyenv().Executable, module));
      if status ~= 0
        fprintf(2, 'ERROR al instalar %s: %s\n', module, cmdout);
        % Puedes lanzar un error o advertir al usuario aquí
      else
        disp(cmdout); % Muestra la salida de pip
        fprintf("Módulo %s instalado correctamente.\n", module);
      end
    else
      fprintf("Módulo %s ya está instalado.\n", module);
    end
  end
end
```

```
function exists = pyModuleExists(mod)
% Verifica si un módulo de Python está instalado
    try
        py.importlib.import_module(mod);
        exists = true;
    catch ME
        % Ignorar el error si es ModuleNotFoundError
        if contains(ME.message, 'ModuleNotFoundError')
            exists = false;
        else
            rethrow(ME); % Volver a lanzar otros tipos de errores
        end
    end
end
```

analizar_imagen.m

```
function hayPersona = analizar_imagen(imagen_path)

% ANALIZAR_IMAGEN - Llama al modelo YOLO en Python para analizar si hay personas en una imagen.

% Entrada:

%   imagen_path - Ruta a la imagen a analizar

%  Se puede añadir otra entrada si tienes otros modelos:

%   model_path  - Ruta al modelo YOLOv11 (por ejemplo, 'yolov8n.pt')

% Salida:

%   hayPersona  - true si hay al menos una persona detectada, false en caso contrario

% También muestra la posición y el ID si hay detección.


current_script_folder = fileparts(mfilename('fullpath'));

if count(py.sys.path, current_script_folder) == 0

   insert(py.sys.path, int32(0), current_script_folder);

end


mod = py.importlib.import_module('video_pred');

py.importlib.reload(mod);


persistent detector

if isempty(detector)

   model_relative_path = fullfile(current_script_folder, 'dataset.yolo11', 'runs', 'detect', 'train', 'weights', 'last.pt');

   %Si el modelo no está en la ruta relativa esperada lanza un error

   if ~exist(model_relative_path, 'file')

      error('El archivo del modelo YOLO no se encontró en: %s', model_relative_path);

   end
```

```
    detector = mod.DetectorTracker(model_relative_path);
end

resultado = detector.process_frame(imagen_path);

if isa(resultado, 'py.str') || isa(resultado, 'char')
    disp(char(resultado));
    hayPersona = false;
elseif isa(resultado, 'py.list') || isa(resultado, 'py.numpy.ndarray')
    % Convertir a cell array de MATLAB
    resultado_m = cell(resultado);

    if isempty(resultado_m)
        disp('No se detectó persona (resultado vacío).');
        hayPersona = false;
    else
        hayPersona = true;
        for i = 1:length(resultado_m)
            fila = cell(resultado_m{i});  % Cada fila debería ser una lista de 5 elementos
            if length(fila) >= 5
                x1 = double(fila{1});
                y1 = double(fila{2});
                x2 = double(fila{3});
                y2 = double(fila{4});
                track_id = double(fila{5});
                fprintf('Persona detectada: ID %d en [%f, %f, %f, %f]\n', track_id, x1, y1, x2, y2);
```

```
        else
            warning('Fila con formato inesperado en la detección.');
        end
    end
  end
else
  warning('Tipo de resultado desconocido: %s', class(resultado));
  hayPersona = false;
end


end
```

video_pred.py

```
import os

from ultralytics import YOLO

import numpy as np

import cv2

from sort import Sort

import sys


class DetectorTracker:


  def __init__(self, model_path):

    #print(f"DEBUG: DetectorTracker inicializado. Python Executable: {sys.executable}")

    self.model = YOLO(model_path)

    self.tracker = Sort()

    self.threshold = 0.4

    self.max_display_size = 1000 # Define el tamaño máximo del lado más largo (en píxeles)


  def process_frame(self, image_path):

    #print(f"DEBUG: process_frame llamado con imagen: {image_path}")

    frame = cv2.imread(image_path)


    if frame is None:

      print(f"ERROR: No se pudo cargar la imagen desde {image_path}")

      return "Error: Imagen no encontrada."


    #print(f"DEBUG: Imagen '{os.path.basename(image_path)}' cargada correctamente. Tamaño original: {frame.shape[1]}x{frame.shape[0]}")
```

```
        # - Redimensionar imagen -
        # Obtener las dimensiones actuales de la imagen
        h, w = frame.shape[:2]

        # Calcular el factor de escala manteniendo la relación de aspecto
        if max(h, w) > self.max_display_size:
            scale_factor = self.max_display_size / max(h, w)
            new_w = int(w * scale_factor)
            new_h = int(h * scale_factor)
            frame_resized = cv2.resize(frame, (new_w, new_h), interpolation=cv2.INTER_AREA)
            #print(f"DEBUG: Imagen redimensionada a: {new_w}x{new_h}")
        else:
            # Si la imagen ya es más pequeña que el tamaño máximo no la redimensionamos
            frame_resized = frame
            #print("DEBUG: Imagen no redimensionada (ya es menor que el tamaño máximo).")

        # -Redimensionar imagen -


        results = self.model(frame_resized)[0] # Importante: usar frame_resized para la detección
        #print(f"DEBUG: YOLO model processed. Detections found: {len(results.boxes.data.tolist())}")

        detections = []
        person_detections_for_display = []
```

```
    for result in results.boxes.data.tolist():
        x1, y1, x2, y2, score, class_id = result
        if score > self.threshold and results.names[int(class_id)].lower() in ['person', 'human']:
            detections.append([x1, y1, x2, y2, score])
            person_detections_for_display.append({
                'x1': int(x1), 'y1': int(y1), 'x2': int(x2), 'y2': int(y2),
                'score': score, 'class_name': results.names[int(class_id)]
            })
    #print(f"DEBUG: Person detections for display: {len(person_detections_for_display)}")

    if len(detections) > 0:
        tracked_objects = self.tracker.update(np.array(detections))
        output = tracked_objects.tolist()
    else:
        output = []

    # Dibuja y muestra la imagen (usando frame_resized)
    if len(person_detections_for_display) > 0:
        for det in person_detections_for_display:
            x1, y1, x2, y2 = det['x1'], det['y1'], det['x2'], det['y2']
            score = det['score']
            class_name = det['class_name']
            cv2.rectangle(frame_resized, (x1, y1), (x2, y2), (0, 255, 0), 2)
            label = f"{class_name}: {score:.2f}"
            cv2.putText(frame_resized, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0,
255, 0), 2)
```

```
        #print("DEBUG: Bounding boxes dibujadas en el frame redimensionado.")
    #else:
        #print("DEBUG: No se detectaron personas, mostrando frame redimensionado sin BBoxes.")



    window_name = f"Analysed Image - {os.path.basename(image_path)}"
    #print(f"DEBUG: Intentando mostrar ventana '{window_name}'...")
    cv2.imshow(window_name, frame_resized) # Mostrar la imagen redimensionada
    #print("DEBUG: cv2.imshow llamado. Esperando tecla...")
    cv2.waitKey(0)
    #print("DEBUG: Tecla presionada o ventana cerrada. Llamando a destroyAllWindows...")
    cv2.destroyAllWindows()
    #print("DEBUG: Ventanas destruidas. Finalizando process_frame.")


    return output
```