

Prototipo de embarcación a escala con sistema de navegación asistida y detección de obstáculos “Dagon”

Research paper as a course end project

Jhoan Stevan Mosquera Ortiz¹
jhoan.mosquera@uao.edu.co

María del Mar García
Matabanchoy²
maria_d.garcia_m@uao.edu.co

Oscar Julián Pérez Ladino³
oscar.perez@uao.edu.co

Miguel Ángel Gonzalez Bérmudez⁴
miguel_a.gonzalez_b@uao.edu.co

Universidad Autónoma de Occidente, Facultad de Ingeniería, Programa de Ingeniería Mecatrónica

Resumen

El proyecto "Dagon" tiene como objetivo el desarrollo de un prototipo de embarcación a escala con un sistema de navegación asistida y detección de obstáculos. Este prototipo está diseñado para mejorar la seguridad en entornos acuáticos controlados, especialmente en embarcaciones menores que operan en condiciones donde la fatiga, distracción y visibilidad reducida son factores de riesgo. La propuesta se enfoca en integrar un sistema que complementa al operador humano, mediante el uso de sensores y algoritmos de control que permitan detectar obstáculos y realizar maniobras evasivas para mantener la ruta predefinida, todo mientras se maximiza la seguridad y reduce el riesgo de accidentes.

A pesar de los avances en la autonomía marítima, la mayoría de las soluciones existentes están orientadas a grandes embarcaciones. Sin embargo, las embarcaciones menores carecen de tecnologías que puedan asistir al operario de manera efectiva en situaciones críticas. El prototipo "Dagon" busca llenar este vacío, ofreciendo una solución intermedia en la que el operario mantiene el control, pero el sistema de navegación asistida garantiza la evasión de obstáculos y el retorno seguro a la trayectoria correcta cuando sea necesario. Esta solución no busca la autonomía total, sino proporcionar asistencia que reduzca los riesgos asociados a la navegación.

El diseño del prototipo incluye un sistema de sensado con un sensor ultrasónico sumergible JSN SR04M para la detección de obstáculos, un microcontrolador ESP32 para el control y procesamiento de datos, y un motor brushless con hélice acuática para la propulsión. Además, el sistema de dirección se gestiona mediante un servomotor 995 de 10Kg, que permite ajustar el timón de manera precisa, garantizando maniobras estables, incluso bajo condiciones de corriente o viento.

Para la estabilización y control de la embarcación, se integra un IMU (Unidad de Medición Inercial) MPU9255, que proporciona información crucial sobre la orientación y movimiento del prototipo, lo que permite al sistema de navegación compensar cualquier desvío y garantizar que la embarcación siga su trayectoria de manera precisa, incluso en condiciones cambiantes.

El sistema de navegación está basado en algoritmos de control PID, que permiten ajustar la velocidad y la dirección de la embarcación de manera precisa. Además, se implementó un modo asistido que permite a la embarcación corregir su curso automáticamente cuando detecta desviaciones o obstáculos. Estas funcionalidades fueron probadas en un entorno controlado, como una piscina, donde el prototipo demostró su capacidad para evitar obstáculos y mantenerse en la ruta definida, con una tasa de éxito en la evasión de obstáculos superior al 90% en el modo asistido.

Las pruebas iniciales de impermeabilidad y flotación confirmaron que el casco del prototipo ofrece la estabilidad y estanqueidad necesarias para operar en entornos acuáticos. También se realizaron pruebas unitarias de los sistemas electrónicos y actuadores, asegurando que el motor, el sensor ultrasónico, el servomotor y el IMU funcionen correctamente. En el aspecto de teleoperación, el prototipo fue simulado en Gazebo, lo que permitió validar la lógica de navegación y la interacción con el operador antes de las pruebas físicas. Aunque la simulación no reemplaza los ensayos en el mundo real, sirvió como un banco de pruebas para ajustar los parámetros de control y optimizar la respuesta del sistema ante diferentes escenarios.

Palabras clave: Navegación asistida, prototipo de embarcación, teleoperación, seguridad marítima, evasión de obstáculos.

Abstract

The “Dagon” project aims to develop a scale model boat prototype with an assisted navigation and obstacle detection system. This prototype is designed to improve safety in controlled aquatic environments, especially in small boats operating in conditions where fatigue, distraction, and reduced visibility are risk factors. The proposal focuses on integrating a system that complements the human operator, using sensors and control algorithms to detect obstacles and perform evasive maneuvers to maintain the predefined route, all while maximizing safety and reducing the risk of accidents.

Despite advances in maritime autonomy, most existing solutions are geared toward large vessels. However, smaller vessels lack technologies that can effectively assist the operator in critical situations. The “Dagon” prototype seeks to fill this gap by offering an intermediate solution in which the operator remains in control, but the assisted navigation system ensures obstacle avoidance and a safe return to the correct course when necessary. This solution does not seek total autonomy, but rather to provide assistance that reduces the risks associated with navigation.

The prototype design includes a sensing system with a JSN SR04M submersible ultrasonic sensor for obstacle detection, an ESP32 microcontroller for control and data processing, and a brushless motor with an aquatic propeller for propulsion. In addition, the steering system is managed by a 10 kg 995 servomotor, which allows for precise rudder adjustment, ensuring stable maneuvering even in strong currents or wind conditions.

For stabilization and control of the boat, an MPU9255 IMU (Inertial Measurement Unit) is integrated, which provides crucial information about the orientation and movement of the prototype, allowing the navigation system to compensate for any deviation and ensure that the boat follows its trajectory accurately, even in changing conditions.

The navigation system is based on PID control algorithms, which allow the speed and direction of the boat to be adjusted precisely. In addition, an assisted mode was implemented that allows the boat to automatically correct its course when it detects deviations or obstacles. These features were tested in a controlled environment, such as a swimming pool, where the prototype demonstrated its ability to avoid obstacles and stay on the defined route, with an obstacle avoidance success rate of over 90% in assisted mode.

Initial waterproofing and flotation tests confirmed that the prototype hull offers the stability and watertightness necessary to operate in aquatic environments. Unit tests were also performed on the electronic systems and actuators, ensuring that the motor, ultrasonic sensor, servomotor, and IMU are functioning properly. In terms of teleoperation, the prototype was simulated in Gazebo, which allowed the navigation logic and interaction with the operator to be validated prior to physical testing. Although simulation does not replace real-world testing, it served as a test bed for adjusting control parameters and optimizing the system's response to different scenarios.

Keywords: Assisted navigation, prototype vessel, teleoperation, maritime safety, obstacle avoidance

1. INTRODUCCIÓN

La navegación marítima, fluvial y lacustre es un componente esencial del transporte global y local por su aporte al movimiento de mercancías, al tejido económico regional y a la conectividad de comunidades. Sin embargo, esta actividad expone a las tripulaciones y al entorno a riesgos que comprometen la seguridad humana y la protección ambiental. Entre los factores que explican dichos eventos sobresale la intervención humana, sobre todo durante operaciones nocturnas, bajo clima adverso o en condiciones de fatiga de la tripulación [1][2].

La evidencia muestra que los microsueños, incluso de pocos segundos, elevan de manera significativa la probabilidad de colisiones o encallamientos [1]. A ello se agregan la distracción del personal, deficiencias de mantenimiento, inexperiencia operativa y variabilidad climática, combinación que incrementa la ocurrencia de incidentes con consecuencias materiales, ambientales y humanas [2]. Diversas fuentes estiman que alrededor del 75 % de los accidentes marítimos tiene origen en errores humanos, lo que dimensiona el problema y orienta la búsqueda de soluciones concretas .

La Organización Marítima Internacional (IMO) impulsa tecnologías de navegación autónoma; entre estas iniciativas destaca el proyecto MASS (Maritime Autonomous Surface Ships), orientado a embarcaciones más seguras y con menor dependencia de la intervención directa del operario [1]. Muchos desarrollos se enfocan en naves de gran escala, lo que mantiene un vacío en embarcaciones menores y prototipos con acceso limitado a estas tecnologías.

Por ello, se propone el desarrollo de soluciones intermedias con funciones de asistencia. La integración de sensores, algoritmos de control y mecanismos de evasión de obstáculos en prototipos a escala constituye una estrategia efectiva para validar alternativas, con efectos directos en la reducción de riesgos y en la formación en robótica marítima.

Este documento presenta el diseño, implementación y validación del prototipo “Dagón”, orientado a la navegación asistida y a la evasión de obstáculos en entornos acuáticos controlados, con el fin de complementar la labor del operario y elevar los estándares de seguridad.

1.1 Planteamiento del problema y justificación

En embarcaciones menores y escenarios de operación controlados, como en prácticas de navegación y recorridos con trayectorias predefinidas, la seguridad depende en gran medida de la atención del operario. Situaciones como la fatiga, las distracciones o la baja visibilidad, junto con la presencia de obstáculos imprevistos, elevan el riesgo de desvíos de ruta y colisiones.

Los sistemas disponibles en este segmento suelen limitarse al radiocontrol y automatizaciones básicas de rumbo como waypoint GPS o follow heading, sin percepción activa del entorno ni capacidad de evitar obstáculos y reincorporarse de forma fiable a la trayectoria de referencia cuando el humano se distrae o comete errores. Esto deja expuesta la operación a incidentes que afectan la seguridad humana, el equipo y la continuidad de las misiones.

Aunque existen avances en navegación autónoma como la “corrección de deriva por viento o corriente”, la mayoría apuntan al reemplazo del operador en buques de gran escala, falta una solución intermedia y accesible para embarcaciones pequeñas que, en lugar de buscar autonomía total, funcione como asistente inteligente: que vigile la ruta definida por el operario, detecte y evite obstáculos locales y, tras la maniobra evasiva, retome la trayectoria de referencia de manera segura y estable.

Se propone desarrollar el prototipo a escala “Dagon” para embarcaciones menores, que integre asistencia inteligente a la navegación, con capacidad de detectar obstáculos, ejecutar maniobras evasivas autónomas y reincorporarse a la trayectoria definida por el operario. Se desea realizar un sistema que combine el sensado y control en una arquitectura compacta y de bajo costo, orientada a mejorar la seguridad operativa en escenarios controlados y a validar su funcionalidad antes de escalar su complejidad.

Pregunta de investigación:

¿De qué manera un prototipo marítimo a escala, con ruta definida por los operarios, puede asistir la navegación garantizando un desplazamiento seguro en entornos acuáticos mediante la detección y evasión de obstáculos y la recuperación confiable de la trayectoria de referencia, en apoyo de la seguridad humana y operativa?

1.2 Justificación

La navegación en embarcaciones de escala en contextos académicos y de formación práctica enfrenta riesgos derivados de la fatiga de los operarios, las distracciones y la aparición inesperada de obstáculos en el entorno acuático, elevando las probabilidades de desviaciones de rumbo y colisiones con consecuencias humanas, materiales y ambientales [1][2].

Los sistemas actuales basados en radiocontrol y seguimiento de puntos de ruta con GPS no integran percepción activa del entorno que apoye al operario en maniobras críticas de evasión de obstáculos [2].

Dagon atiende esta necesidad desarrollando un prototipo marítimo a escala que integra tecnologías de sensado y algoritmos de control con una arquitectura escalable para validar alternativas de asistencia en laboratorio.

Este prototipo deberá detectar obstáculos, ejecutar maniobras evasivas y retomar la trayectoria de referencia definida por el operario mediante algoritmos de planificación de ruta local y supervisión dinámica de modos de operación [2].

Se definirán protocolos de calibración de sensores, fusión de datos y métricas de desempeño en pruebas en piscina semiolímpica, evaluando tasa de recorridos sin colisión, distancia mínima a obstáculos, latencia de respuesta y desviación media respecto a la trayectoria [2].

Los resultados generarán un banco de datos reproducible y procedimientos de integración que fortalecerán la formación práctica de estudiantes de mecatrónica, y servirán como base para escalar aplicaciones en robótica marítima avanzada [2].

1.3 Objetivos

Objetivo General:

Desarrollar un prototipo marítimo a escala con capacidad de asistir la navegación, integrando tecnologías de sensado y control que complementen las decisiones humanas, que detecten y eviten obstáculos y así mismo garanticen el retorno a la trayectoria de referencia definida.

Objetivos específicos:

- Explorar tecnologías de sensor y control aplicables a la asistencia en la navegación de un prototipo marítimo a escala.
- Desarrollar un sistema funcional de apoyo a la navegación, orientado en mejorar la seguridad y reducir riesgos durante el desplazamiento en entornos acuáticos.
- Evaluar el prototipo en escenarios controlados para validar la viabilidad del sistema como herramienta de apoyo a la navegación.

1.4 Alcance y limitaciones

El trabajo se desarrolla en piscina y en ambientes semicontrolados con un prototipo a escala. No se extrapolan los resultados de forma directa a embarcaciones reales. Aun así se construye una base experimental sólida para aumentar gradualmente la complejidad y el nivel de autonomía en etapas siguientes.

1.5 Selección metodológica

Existen varias rutas posibles como reglas de evasión, planeación y enfoques de aprendizaje. En este avance elegimos un enfoque de asistencia que sea reproducible y trazable en laboratorios. Esta elección está en concordancia tanto con los objetivos planteados en entregas anteriores como los objetivos académicos del curso de Robótica y asegura una línea base manual clara para comparar de manera transparente los resultados.

2. MARCO TEÓRICO

La navegación autónoma se refiere a la capacidad de un vehículo o robot para desplazarse por su entorno y llegar a un destino sin intervención humana directa, tomando decisiones en tiempo real en función de la información que percibe de sus sensores. En otras palabras, un sistema de navegación autónoma puede definir por sí mismo la trayectoria que seguirá a través del entorno para ir del punto A al punto B, evitando obstáculos inesperados en el camino. Esta habilidad requiere que el robot perciba su entorno, planifique sus movimientos y luego actúe en consecuencia de forma independiente. Un robot autónomo es aquel que reúne estas capacidades: es capaz de percibir el entorno (mediante sensores), tomar decisiones basadas en lo que percibe (procesamiento inteligente) y ejecutar acciones físicas (movimiento o manipulación) sin la intervención humana[5]. Importa destacar que “autónomo” no es necesariamente sinónimo de “no tripulado”: un vehículo autónomo puede tener operadores humanos supervisando a bordo o a distancia, pero la diferencia es que el sistema autónomo puede realizar sus funciones (como navegar, evitar colisiones, etc.) sin órdenes directas en cada paso[2].

En el contexto marítimo, se introducen algunos términos específicos. Un buque autónomo hace referencia a una nave capaz de operar con intervención humana reducida o nula, mediante sistemas avanzados de control y automatización. La Organización Marítima Internacional (OMI) utiliza el término MASS (Maritime Autonomous Surface Ship) para clasificar buques según cuatro grados de autonomía:

Grado 1: Buque con procesos automatizados y sistemas de apoyo a la decisión, pero con tripulación a bordo (o mejor dicho, la tripulación opera y controla la mayoría de las funciones, asistida por algunas automatizaciones).

Grado 2: Buque controlado de forma remota desde tierra, con sistemas avanzados, y con tripulación presente a bordo (operaciones tele-dirigidas, pero se mantiene personal a bordo por seguridad).

Grado 3: Buque controlado a distancia sin tripulación a bordo, es decir, totalmente no tripulado en sitio, pero gobernado por un centro de control remoto en tierra.

Grado 4: Buque totalmente autónomo, capaz de tomar decisiones y ejecutar maniobras sin intervención humana alguna. En este nivel el propio sistema de control del buque evalúa la situación, decide las acciones de navegación (rumbo, velocidad, evitación de obstáculos, etc.) y las lleva a cabo de forma autónoma.

Estas definiciones dejan claro que la autonomía no implica necesariamente la ausencia total de personas, especialmente en los grados intermedios [2]. De hecho, la mayoría de los proyectos actuales en la industria marítima se sitúan entre los grados 1 y 2, es decir, buques con alta automatización pero con operaciones supervisadas por marinos ya sea a bordo o desde centros de control en tierra. Cabe aclarar que la intervención humana mínima se refiere a que cuanto más autónomo es un sistema, menos depende de decisiones humanas en tiempo real para navegar con seguridad.

Otros términos relevantes incluyen las siglas asociadas a tipos de vehículos robóticos marítimos. Un USV (Unmanned Surface Vehicle, a veces también llamado ASV por Autonomous Surface Vehicle) es un vehículo de superficie marítima no tripulado, capaz de navegar de forma remota o autónoma. De modo análogo, un AUV (Autonomous Underwater Vehicle) es un vehículo submarino autónomo, diseñado para operar bajo el agua sin control humano directo. Ambos entran dentro de la robótica marítima, que es la rama de la robótica aplicada al entorno marino, incluyendo la

automatización de buques de superficie y vehículos submarinos. Estos sistemas marítimos autónomos suelen incorporar tecnologías de navegación autónoma adaptadas a las condiciones particulares del medio acuático, como veremos más adelante.

Finalmente, también es útil definir SLAM (Simultaneous Localization and Mapping o Localización y Mapeo Simultáneos), un término técnico clave en navegación autónoma móvil. SLAM es la técnica que permite a un robot a la vez construir un mapa de un entorno desconocido y ubicarse dentro de él. En esencia, el robot va “dibujando” el mapa a medida que explora, mientras calcula su propia posición sobre ese mapa en tiempo real. Esto es fundamental para que un vehículo autónomo sepa dónde está y qué hay alrededor mientras se desplaza, más aún si carece de referencias externas como GPS que es el caso de vehículos submarinos. Tecnologías como LiDAR o cámaras, combinadas con algoritmos de SLAM, proporcionan al robot una percepción espacial precisa y confiable del entorno, base sobre la cual toma decisiones de navegación[8].

Antecedentes del tema

El concepto de navegación autónoma tiene sus raíces en los inicios de la robótica móvil e inteligencia artificial décadas atrás. Uno de los primeros hitos fue Shakey desarrollado en la década de 1960 en el SRI, Stanford, considerado el primer robot móvil general con capacidad de planificar acciones de forma autónoma. Desde entonces, el campo de la robótica autónoma ha avanzado de forma constante, integrando mejoras en sensores, computación y algoritmos de control. Un punto de inflexión importante para la navegación autónoma de vehículos ocurrió en 2004 con el Desafío Grand Challenge de DARPA, una competencia en la cual vehículos terrestres autónomos debían recorrer 240 km de terreno desértico sin intervención humana. Si bien en aquel primer intento ningún vehículo logró completar el recorrido, la experiencia obtenida resultó invaluable. Al repetirse la competición en 2005, cinco vehículos autónomos lograron finalizar el trayecto siendo el ganador el vehículo Stanley de Stanford, demostrando la viabilidad de la conducción autónoma. Posteriormente, en 2007, un nuevo desafío se centró en entornos urbanos, donde el vehículo Boss de Carnegie Mellon se alzó con la victoria al navegar con éxito por calles simuladas obedeciendo reglas de tráfico y evitando colisiones. Estos eventos históricos aceleraron significativamente el desarrollo de algoritmos de percepción, planificación de rutas y control autónomo, y sentaron bases para proyectos autónomos en diversas áreas como automóviles, robots de servicio, drones, etc.[9].

En el ámbito marítimo, la evolución hacia la autonomía ha sido más gradual pero igualmente notable en los últimos años. Los sistemas de piloto automático existen en los barcos desde mediados del siglo XX, permitiendo mantener un rumbo fijo o una velocidad sin intervención manual continua. Ahora bien, esas primeras automatizaciones no implican una navegación totalmente autónoma ni la toma de decisiones complejas ante obstáculos o tráfico marítimo. La verdadera navegación autónoma marítima es decir, buques capaces de trazar sus rutas, evadir otras embarcaciones y adaptar su navegación de forma inteligente empezó a tomar forma con los avances en sensores marinos, computación y sistemas de control a distancia durante las últimas décadas.

Un precedente importante en robótica marina son los vehículos autónomos submarinos (AUVs). Desde fines del siglo XX se han desarrollado AUVs para aplicaciones como exploración oceánica, levantamientos hidrográficos y misiones militares de búsqueda de minas. Por ejemplo, a nivel de investigación, se exploró el uso de enjambres de AUVs cooperantes para misiones submarinas sigilosas. Un proyecto destacado propuso el método SUAVE (Swarm Underwater Autonomous Vehicle Localization), que permitió mejorar la localización de AUVs en grupo reduciendo la frecuencia con que debían salir a la superficie para obtener posición GPS, lo que a su vez disminuía las probabilidades de detección en operaciones encubiertas[1]. Este tipo de innovaciones evidenció tanto el potencial de la autonomía submarina como sus retos particulares, entre ellos la navegación sin GPS y las dificultades de comunicación bajo el agua.

En la última década, los esfuerzos por aplicar la autonomía al transporte marítimo de superficie se han acelerado, impulsados por la convergencia de tecnologías de inteligencia artificial, sensores avanzados y sistemas de comunicaciones.

Un referente fue el anuncio en 2017 del Yara Birkeland, considerado el primer buque portacontenedores completamente eléctrico y diseñado para operación autónoma total. Este barco de 80 metros de eslora, desarrollado en Noruega en asociación con la empresa tecnológica Kongsberg, fue concebido para transportar fertilizantes entre puertos locales sin tripulación a bordo, reduciendo emisiones y transfiriendo carga de las carreteras al mar. Inicialmente se planificó que se comenzarán operaciones con tripulación mínima en 2020, para luego pasar a modo completamente autónomo y sin operadores hacia 2022[1]. Si bien problemas globales como la pandemia de COVID-19 retrasaron su implementación, el Yara Birkeland realizó su viaje inaugural en 2021, convirtiéndose en un caso emblemático de la autonomía marítima aplicada a la industria naviera.

Actualmente (2025), la navegación autónoma en el sector marítimo se encuentra en fase de transición desde proyectos piloto hacia aplicaciones comerciales en nichos específicos. Ya operan, por ejemplo, ferries de corta distancia autónomos o con automatización avanzada en algunos países, así como remolcadores portuarios automatizados y embarcaciones de vigilancia no tripuladas. La mayoría de estas implementaciones iniciales ocurren en entornos controlados o rutas predecibles de corto alcance, donde es más factible gestionar los riesgos. La experiencia acumulada en estos proyectos piloto está guiando la elaboración de estándares técnicos y normativos. Mientras tanto, organizaciones internacionales como la OMI y sociedades de clasificación BV, DNV GL, etc. Analizan cómo adaptar el marco regulatorio para integrar gradualmente buques autónomos en las rutas marítimas comerciales de manera segura y eficiente[2].

Teorías y modelos relacionados

La navegación autónoma de vehículos móviles ya sean robots terrestres, drones aéreos o buques marítimos se fundamenta en una arquitectura clásica de percepción, decisión y actuación. Es decir, el sistema debe percibir su entorno, planificar y/o decidir una respuesta como una trayectoria o maniobra a seguir y actuar físicamente para ejecutar esa decisión. Cada uno de estos componentes involucra teorías, conceptos y modelos específicos:

Percepción del entorno: El vehículo autónomo emplea un conjunto de sensores para obtener información de su entorno en tiempo real. Dependiendo del medio, estos sensores pueden incluir cámaras ópticas, sensores LiDAR (láseres para detección de distancias), radares, sónar (particularmente en entornos marinos), sensores infrarrojos, unidades inerciales (IMU), GPS (en superficie) y otros[1]. Los datos de estos sensores se fusionan mediante algoritmos de fusión sensorial como lo son los filtros de Kalman extendido y filtros de partícula para lograr una estimación más confiable de la posición y del entorno del vehículo. Un concepto clave aquí es el ya mencionado SLAM, que permite construir mapas y localizarse simultáneamente. Por ejemplo, un robot puede combinar lecturas de LiDAR y visión por computadora para reconocer obstáculos y cartografiar el área circundante. En la robótica móvil industrial se suele dividir la percepción en detección de características del entorno formas, objetos, otras naves y mapeo, es decir, la construcción continua de un modelo espacial a medida que el robot se mueve. En contextos marítimos, la percepción enfrenta desafíos particulares; en superficie se cuenta con radar y AIS, sistemas de identificación automática de barcos para detectar otras embarcaciones, mientras que bajo el agua la falta de señal GPS y la limitada visibilidad obligan a usar sónar, DVL (medidores de velocidad Doppler) y referencias geofísicas para la localización. La calidad de la percepción es crítica, ya que cualquier error en la estimación del entorno o de la posición propia puede llevar a trayectorias inseguras.

Planificación y toma de decisiones: Con la información sensorial procesada, el sistema autónomo debe decidir cómo moverse. Aquí intervienen los algoritmos de planificación de rutas y evitación de obstáculos. En la planificación global (a mayor escala), se utilizan a menudo algoritmos de teoría de grafos o de optimización para trazar la ruta óptima hasta el destino (por ejemplo, algoritmos clásicos como Dijkstra o A, y variantes modernas adaptadas a entornos dinámicos). Para la planificación local (reacción ante obstáculos cercanos), existen métodos como campos potenciales, detección/cálculo de curvas de escape o enfoques de ventana dinámica, que permiten generar trayectorias de evasión suaves en tiempo real. En años más recientes, se han incorporado técnicas de inteligencia artificial y aprendizaje automático para la toma de decisiones de navegación. Por ejemplo, algoritmos de aprendizaje por refuerzo profundo pueden aprender políticas de navegación óptimas tras experimentar con miles de situaciones simuladas de tráfico. Los modelos basados en IA pueden

ayudar a predecir movimientos de otros objetos (por ejemplo, pronosticar la trayectoria de otros barcos) y a replanificar la ruta en consecuencia. Un aspecto único en la toma de decisiones de vehículos marítimos es la necesidad de cumplir con las reglas de navegación internacionales, en particular el Reglamento para Prevenir Colisiones en el Mar de 1972 (COLREGs). Estas normas definen, por ejemplo, quién tiene preferencia de paso en un encuentro de proa o de cruce entre dos embarcaciones. Para que un buque autónomo sea aceptado, debe comportarse al menos tan seguro como lo haría un piloto humano, respetando los COLREGs. Esto introduce modelos basados en reglas en la toma de decisiones: muchas investigaciones han codificado las reglas COLREG en sistemas expertos o lógicas de decisión, y más recientemente se exploran enfoques híbridos que combinan aprendizaje automático con restricciones explícitas para garantizar el cumplimiento normativo[1][2].

Actuación y control: Finalmente, las decisiones de movimiento se traducen en señales de control enviadas a los actuadores del vehículo como los motores, timones, hélices, frenos, superficies de control, etc. Aquí entran las teorías clásicas de control automático, por ejemplo, la regulación de la velocidad o el rumbo suele implementarse mediante controladores PID ajustados al modelo dinámico del vehículo. En un entorno marítimo, un piloto automático de rumbo mantiene la dirección usando giroscopios y compás, compensando el viento y la corriente, mientras que en robots terrestres un controlador sigue una trayectoria dada corrigiendo la dirección de las ruedas. Los modelos cinemáticos y dinámicos del vehículo (ecuaciones de movimiento) son la base para diseñar estos controladores y para simular el comportamiento durante la etapa de desarrollo. Además, los sistemas autónomos avanzados suelen incluir módulos de evitación de colisiones en la capa de control, como muestra de ello, si un obstáculo imprevisto aparece muy cerca, un controlador reactivo puede anular temporalmente la ruta planificada y frenar o girar bruscamente para evitar el impacto siendo similar a un reflejo. En el caso de robots móviles generales, se habla de arquitecturas reactivas vs deliberativas; en la práctica, los vehículos autónomos combinan ambas, con un nivel deliberativo planificando rutas a gran escala y un nivel reactivo asegurando la seguridad instante a instante.

Es importante señalar que, para lograr una navegación autónoma robusta, todos estos componentes deben integrarse de forma coherente. De nada sirve un gran planificador si la percepción es deficiente y no ve un obstáculo, o un excelente sensor si el control no responde a tiempo. Por ello, se han desarrollado arquitecturas de software específicas como por ejemplo ROS (Robot Operating System) que facilitan la comunicación entre sensores, algoritmos de decisión y actuadores en tiempo real. Asimismo, la simulación desempeña un rol teórico-práctico clave. Antes de implementar en un robot real, los investigadores prueban sus modelos en simuladores como Gazebo, Webots, MATLAB/Simulink, etc., lo que permite depurar algoritmos de navegación en entornos virtuales seguros[2]. Aunque las simulaciones no capturan todas las complejidades del mundo real, son una herramienta valiosa para iterar rápidamente sobre teorías y modelos de navegación autónoma.

Adicionalmente, en el dominio marítimo existen consideraciones especiales dentro de la teoría de navegación autónoma. Una de ellas es la conectividad y control remoto: muchos buques autónomos operan en conexión con centros de control en tierra. Esto implica incorporar sistemas de comunicación por satélite, radio VHF, 4G/5G, etc., para enviar y recibir datos en tiempo real. Teóricamente, esto introduce el modelo de un humano en el bucle remoto, que puede monitorear la situación y tomar control si el sistema autónomo falla o encuentra una situación no contemplada. Otra consideración es la ciberseguridad: al aumentar la autonomía y conectividad, también aumentan los riesgos de accesos no autorizados o interferencias maliciosas en los sistemas de navegación. Los modelos de seguridad informática y encriptación de comunicaciones se vuelven parte del diseño teórico de los sistemas autónomos marítimos[2], buscando garantizar que un buque autónomo no pueda ser fácilmente hackeado o engañado mediante falsificación de señales GPS o AIS. Todos estos elementos de percepción múltiple, planificación inteligente, control automático y comunicación segura forman el marco teórico que sustenta el desarrollo de la navegación autónoma.

Estudios previos

Dada la rápida evolución del campo, en los últimos cinco años han proliferado estudios y revisiones que abordan

diferentes aspectos de la navegación autónoma. Más adelante, se sintetizan los hallazgos de algunos trabajos relevantes, con énfasis en publicaciones recientes y de alto impacto:

Revisiones generales de navegación autónoma: Nahavandi et al. (2022) presentan una revisión exhaustiva sobre robots móviles autónomos, abarcando sensores, plataformas robóticas, herramientas de simulación, métodos de planificación de rutas y seguimiento, fusión sensorial, evitación de obstáculos y SLAM. Esta revisión destaca dos tendencias principales: El campo de navegación autónoma evoluciona muy rápido, lo que hace crucial publicar encuestas periódicas para mantener actualizada a la comunidad.

La irrupción de métodos de aprendizaje profundo ha revolucionado muchas áreas, incluida la navegación autónoma, lo que motivó a los autores a tratar en detalle el papel de estas técnicas de IA en problemas de localización, detección de obstáculos y planificación.

El estudio de Nahavandi et al., también resume los vacíos identificados por trabajos previos, por ejemplo la ausencia de ciertos tópicos como el ya mencionado SLAM u obstáculos dinámicos en encuestas más antiguas, evidenciando que aún en 2022 había áreas poco cubiertas por la literatura tradicional. En sus conclusiones, estos autores subrayan la necesidad de seguir integrando algoritmos clásicos con enfoques de inteligencia artificial para superar desafíos pendientes ya sean ambientes no estructurados, incertidumbre, etc [2].

Autonomía en buques de superficie (COLREGs y navegación segura): Un tema de mucho interés reciente es cómo lograr que los buques autónomos de superficie naveguen de forma segura y conforme a las reglas marítimas. En este sentido, Hu et al. (2022) publicaron una revisión sobre navegación COLREGs conforme de vehículos de superficie autónomos (ASVs). Este trabajo recopila los avances de las dos últimas décadas en métodos de evitación de colisiones para ASVs, clasificando los enfoques desde los tradicionales (basados en reglas explícitas y algoritmos determinísticos) hasta los modernos basados en aprendizaje automático[1]. La revisión de Hu et al., ofrece una visión holística de la navegación segura dividiéndola en tres etapas: detección de colisión, toma de decisión y planificación para la recomposición de la ruta primaria.

En cada etapa se discuten los métodos existentes y se analiza en qué medida incorporan las reglas de COLREG, se ha identificado que muchos estudios se centran en maniobras para escenarios básicos en encuentros de proa, cruce y adelantamientos, pero estos escenarios abarcan situaciones COLREG más complejas cuando existen operaciones en canales angostos, visibilidad reducida o múltiples actores[2]. Esta revisión resalta la aparición muy reciente de técnicas de aprendizaje profundo aplicadas a la predicción de movimiento de otras embarcaciones y a la planificación de rutas evasivas en entornos complejos. Los autores concluyen señalando desafíos abiertos, como la necesidad de disponer de datasets adecuados para entrenar sistemas de IA en escenarios marítimos y la importancia de combinar la fiabilidad de los métodos tradicionales con la flexibilidad de los métodos aprendidos [1].

Evasión de colisiones marítimas: En 2021, Burmeister y Constapet llevaron a cabo una encuesta sistemática enfocada en técnicas de avoidance o evitación de colisiones para buques autónomos, publicada en *Frontiers in Robotics and AI*, un hallazgo clave es que la investigación en estos años ha estado muy activa, con la mayoría de publicaciones proponiendo optimizaciones de algoritmos existentes específicamente adaptados al escenario marítimo. En otras palabras, gran parte de los estudios no reinventan la rueda, sino que toman algoritmos de navegación conocidos como las variantes de RRT, A, o modelos de decisión de Markov; los ajustan para cumplir con COLREGs o para lidiar con las particularidades del medio marino. Burmeister et al. proporcionan un análisis detallado de cómo cada trabajo aborda distintos aspectos de COLREG; notan que la mayoría de los enfoques cubren únicamente las situaciones básicas de encuentro en adelantamiento, encuentro frontal y de cruce y pasan por alto otras reglas más específicas. Otro patrón identificado es la limitada consideración de entornos costeros complejos: muchos estudios asumen navegación en mar abierto o con total libertad de maniobra, y pocos se aventuran en escenarios como canales angostos o zonas de tráfico denso con separación de vías. Además, esta encuesta revela que muy pocos han sido validados en pruebas de mar reales, señalando una brecha

entre la investigación académica y la aplicación industrial. No existe aún un estándar común para probar y comparar estos algoritmos, lo que dificulta evaluar cuál método es “mejor” de forma objetiva. En particular, se advierte la falta de un benchmark internacionalmente acordado para evaluar sistemas de evitación de colisiones autónomas, situación que contrasta con otros sectores como el de vehículos autónomos terrestres donde sí se cuentan con conjuntos de datos y escenarios de referencia. Este estudio concluye enfatizando la necesidad de cubrir las lagunas encontradas, un ejemplo de esto es la incorporación de todas las reglas COLREG de la Parte B, no solo algunas, unificar los requerimientos de sensores y datos del entorno, y establecer procedimientos de prueba comunes para que las soluciones propuestas en la literatura puedan avanzar hacia implementaciones industriales[1].

Otros estudios relevantes: Además de las revisiones mencionadas, numerosos artículos específicos han aportado avances puntuales en sub-problemas de la navegación autónoma en años recientes. En robótica submarina, Zhou et al. (2020) presentaron una revisión de técnicas integradas de navegación y comunicación para AUVs colaborativos, resaltando métodos de localización acústica y visión submarina para suplir la falta de GPS[9].

En el campo de embarcaciones a vela autónomas, Mankina et al. (2023) exploraron los avances en el control AI de veleros no tripulados, un desafío particular dado lo impredecible del viento y las restricciones únicas de este tipo de propulsión, durante aplicaciones industriales, se han publicado casos de estudio como el del ferry autónomo de Finferries (proyecto SSPA, 2020) que realizó travesías totalmente automatizadas supervisadas en Finlandia, o el proyecto MARIN en Países Bajos experimentando con remolcadores portuarios autónomos. Por un lado, amplias revisiones que sintetizan el estado del arte y delinean los retos pendientes, y por otro, numerosos trabajos específicos que empujan las fronteras en aspectos concretos como mejoras algorítmicas, nuevas arquitecturas y validaciones experimentales. Estos estudios previos proporcionan la base sobre la cual se identifica el problema de investigación y las oportunidades de contribución.

Algoritmos aplicados en prototipos de navegación a escala

La literatura reciente describe el uso de prototipos a escala como plataforma para validar estrategias de navegación en entornos controlados antes de su transferencia a aplicaciones reales. Estos modelos permiten comprobar el desempeño de algoritmos clásicos de control y planificación en esquemas de navegación asistida, con particular foco en evitación y seguimiento de trayectorias [2].

Evasión de obstáculos basada en reglas simplificadas de COLREGs. Las revisiones sobre navegación conforme a COLREGs en vehículos de superficie autónomos destacan la necesidad de integrar detección, decisión y replanificación para escenarios básicos de encuentro: frontal, cruce y adelantamiento. En prototipos a escala, esta integración se implementa con variantes simplificadas de reglas que activan giros y ajustes de rumbo cuando se detecta riesgo de colisión, logrando tasas altas de maniobras exitosas en simuladores y estanques controlados [1][2].

Seguimiento de ruta por waypoints. En USVs a escala es habitual combinar un planificador global que genera una lista de waypoints con un módulo de seguimiento que ajusta rumbo y avanza al siguiente punto al entrar en tolerancia. Las revisiones de navegación autónoma reportan trayectorias cerradas con error acotado bajo perturbaciones leves, lo que hace viable su uso en entornos de piscina o lago controlado.

Control de rumbo con PID. El control PID continúa siendo una técnica de referencia por su simplicidad y eficacia para estabilizar el rumbo medido por compás o IMU, manteniendo errores angulares pequeños en pruebas a escala y facilitando la integración con planificadores locales [2].

La combinación de reglas COLREGs simplificadas para evasión, seguimiento por waypoints y control PID de rumbo ofrece una base realista y de bajo costo computacional para prototipos en laboratorio, alineada con los requisitos de navegación asistida que persigue este anteproyecto [1][2].

Ánalisis crítico de la literatura

De la revisión de estudios previos se desprende que, a pesar del notable progreso, existen limitaciones recurrentes y vacíos en el conocimiento actual sobre navegación autónoma, especialmente en el contexto marítimo, que abren espacio a investigaciones nuevas. A continuación, se destacan críticamente algunos de estos aspectos:

Cobertura parcial de escenarios y reglas de navegación: Varios estudios tienden a enfocarse en escenarios simplificados o subsistemas aislados, sin abarcar la complejidad completa que enfrentaría un vehículo autónomo en el mundo real. En el caso de los buques, la mayoría de trabajos sobre evitación de colisiones se limitan a cumplir un subconjunto de las reglas COLREG, muchas veces son relativas a encuentros frontales, cruces y adelantamientos básicos. Sin embargo, situaciones como navegar en canales angostos, en proximidad de puertos congestionados, maniobras con visibilidad reducida o cumplimiento de secciones menos obvias del reglamento como señales sonoras, luces, reglas de tráfico en vías separadas reciben poca atención en la literatura actual. Esta cobertura incompleta deja lagunas en la preparación de sistemas autónomos para entornos operativos reales. Un vehículo completamente autónomo debería ser capaz de manejar todas las condiciones que un operador humano entrenado enfrentaría; por ahora, ningún enfoque investigado logra esa cobertura integral. Esto sugiere la necesidad de desarrollar modelos más integradores o metodologías que garanticen la conformidad total con normativas marítimas en cualquier circunstancia [1].

Supuestos ideales vs. realidad operativa: Muchos modelos propuestos asumen condiciones ideales de sensores y comunicaciones. Por ejemplo, en no pocos estudios se considera que el sistema cuenta con datos perfectos o completos del entorno, datos sin ruido o sin fallos de sensores lo cual dista de ser cierto en la práctica. En entornos reales, los sensores pueden fallar momentáneamente, proveer lecturas erróneas que pueden ser desde falsos ecos de radar, interferencias acústicas submarinas, imágenes borrosas, etc o directamente no detectar ciertos obstáculos como objetos muy pequeños o condiciones como reflejos en el agua que confunden a una cámara.

La literatura refleja que pocos trabajos integran de forma explícita la gestión de incertidumbre o la detección y tolerancia a fallos sensoriales en sus algoritmos [2]. La dependencia en GPS para localización es un talón de Aquiles: para vehículos terrestres y aéreos es útil, pero para AUVs es inexistente bajo el agua, el GPS puede perder precisión o ser vulnerado. Aunque se han investigado métodos alternativos como navegación inercial, georeferenciación con mapas batimétricos, balizas acústicas, cada uno tiene limitaciones de costo o alcance. La conclusión es que la robustez de la navegación autónoma ante condiciones no ideales es un área que requiere más atención; futuros sistemas deben incorporar redundancia sensorial, algoritmos adaptativos que identifiquen datos anómalos, y capacidades de fail-safe en caso de perder temporalmente cierta percepción.

Validación experimental y transferencia al entorno real: Un punto crítico señalado por varios autores es la brecha entre las pruebas en simulación y la validación en entornos reales. La gran mayoría de propuestas de algoritmos se validan únicamente mediante simulaciones por computadora o pruebas controladas de laboratorio, y muy pocas pasan a la fase de pilotaje en condiciones operativas. Si bien las simulaciones son útiles en etapas iniciales (permite iterar rápido y barato), pueden omitir factores impredecibles del mundo real: clima, comportamiento complejo de otros actores, fallos mecánicos, etc. Burmeister et al. (2021) observaron que no existe un esquema de pruebas estandarizado en el ámbito de buques autónomos, lo que resulta en que cada autor use sus propios escenarios simulados y métricas, dificultando la comparación objetiva de resultados. Además, señalan que ningún algoritmo prevalece claramente sobre los demás porque casi ninguno ha sido ensayado en situaciones de la vida real ni contra estándares comunes[2]. Esta falta de benchmarking y validación externa es una debilidad de la literatura actual. Para avanzar, sería deseable que la comunidad converja en conjuntos de datos compartidos por ejemplo, simulaciones estandarizadas de encuentros multi-barco con diferentes niveles de tráfico, o datos reales recopilados de prototipos como el Yara Birkeland y protocolos de prueba uniformes. Solo así se podrá evaluar de forma confiable qué técnicas son suficientemente seguras y maduras para implementación. En este sentido, se reconoce la importancia de proyectos piloto industriales: colaboraciones entre academia e industria que permitan probar algoritmos en buques experimentales, algo que ya se empieza a ver en Europa y Asia, pero cuyos resultados apenas comienzan a publicarse.

Integración tecnológica y sistemas holísticos: Otro aspecto crítico es la integración de las distintas tecnologías necesarias para la autonomía. Como se discutió en la sección teórica, la navegación autónoma abarca múltiples subsistemas desde la percepción, la decisión, el control, la comunicación hasta la seguridad informática. Varios proyectos de ingeniería han demostrado que la tecnología necesaria existe, hay sensores, IA, controladores, pero combinarla de la forma correcta es un desafío no trivial. Los estudios suelen abordar problemas de forma aislada, pongamos el caso de un nuevo algoritmo de planificación, o una mejor cámara de visión nocturna, pero pocos abordan el sistema completo. Los hallazgos del proyecto AWAA (2016) en Finlandia ya advertían que el desarrollo de sistemas de decisión autónoma sería un proceso progresivo, requiriendo multitud de pruebas y simulaciones para afinar la colaboración entre módulos. Las experiencias iniciales con buques autónomos han confirmado que integrar sensores múltiples con algoritmos de control a veces revela incompatibilidades, retrasos o conflictos que no se anticiparon en el diseño por separado.

Además, al reducir la intervención humana directa, emergen nuevos riesgos sistémicos que antes no existían o pasaban inadvertidos. Tal como, la ausencia de tripulación implica que fallos simples, atasco de un timón, reinicio de un computador que una tripulación solucionaría en segundos pueden convertirse en emergencias graves si el sistema autónomo no tiene auto-diagnóstico y mecanismos de contingencia. La literatura señala estas preocupaciones pero aún carece de soluciones ampliamente aceptadas. En la práctica, se están explorando arquitecturas de autonomía graduada, donde el buque puede operar de forma autónoma pero pasa a control remoto o manual en cuanto detecta condiciones fuera de su rango seguro de operación[2].

Desafíos normativos, éticos y organizativos: Más allá de los obstáculos técnicos, los estudios y reportes enfatizan que hay barreras regulatorias y organizativas significativas. Actualmente, las regulaciones internacionales que son la OMI, convenios SOLAS, COLREGS, etc. no contemplan totalmente la operación de buques sin tripulación tradicional[2]. Existe un intenso trabajo en comités para actualizar estas normas, pero mientras tanto cualquier implementación debe obtener exenciones o permisos especiales. En la literatura legal y de gestión marítima, surgen varias preguntas al intentar adaptar los requisitos de seguridad, por ejemplo, ¿necesita un buque autónomo llevar ciertos equipos de emergencia?, ¿Quién es el “capitán” legalmente responsable a bordo? y los seguros marítimos a esta nueva realidad. La cuestión de la responsabilidad legal en caso de un accidente con un vehículo autónomo es una incógnita: ¿recae en el programador del algoritmo, en el fabricante del barco, en el operador remoto ó en el dueño de la carga? Algunos estudios abogan por actualizar conceptos jurídicos como la “responsabilidad del producto” para cubrir estos casos, pero todavía no hay consenso global. Esta incertidumbre regulatoria se refleja en la cautela de la industria: según encuestas y artículos del sector, muchas navieras están observando los desarrollos pero pocas invierten fuertemente hasta tener claro el marco normativo. Adicionalmente, los costos de desarrollo siguen siendo muy elevados y los proyectos piloto requieren inversiones multimillonarias en I+D lo cual limita la participación a grandes compañías o consorcios gubernamentales. Este factor económico es señalado como una barrera para la masificación de la tecnología, al menos en el corto plazo. Finalmente, se apunta que la transición hacia lo autónomo no eliminará al humano sino que transformará su rol: surgirán nuevos perfiles profesionales así como el de operadores de flotas remotas, ingenieros de datos marítimos, técnicos de mantenimiento autónomo, etc. Mientras que algunos roles tradicionales podrían reducirse, la aceptación de la autonomía en el sector también pasa por gestionar estos impactos laborales y de seguridad.

El análisis crítico de la literatura reciente revela un panorama donde coexisten grandes logros con grandes desafíos. Por un lado, se ha demostrado la factibilidad técnica de la navegación autónoma, se han construido vehículos que efectivamente navegan solos, y los algoritmos para percibir, planificar y controlar han alcanzado madurez suficiente en entornos acotados. Por otro lado, quedan brechas importantes por cerrar antes de que la navegación autónoma sea una realidad común: lograr la fiabilidad y seguridad equivalentes o superiores a la operación humana en todas las condiciones, estandarizar y validar los sistemas para garantizar la confianza de reguladores y usuarios, y resolver cuestiones prácticas de integración tecnológica y normativa. Estos desafíos identificados en la literatura refuerzan la motivación del problema de investigación planteado, existe una clara necesidad de proponer soluciones novedosas que aborden, aunque sea parcialmente, dichas lagunas.

Se aclara que el presente proyecto se centra en la navegación autónoma asistida, entendida como un nivel básico-intermedio en el cual la embarcación es capaz de percibir el entorno, seguir trayectorias predefinidas y ejecutar maniobras básicas de evasión, pero siempre bajo la supervisión y con la posibilidad de intervención humana. Este enfoque resulta coherente con los grados iniciales de autonomía propuestos por la OMI para los buques MASS, donde la intervención humana no desaparece, sino que se combina con funciones automatizadas para incrementar la seguridad y reducir el riesgo de errores operativos.

3. MATERIALES Y MÉTODOS.

Esta sección describe la estrategia para abordar el problema, los materiales y equipos previstos, los procedimientos paso a paso y la forma de recolectar y analizar los datos 2. La organización de los subtítulos coincide con la estructura que se usará luego en resultados y discusión.

2.1 Tipo y línea de investigación

Estudio aplicado con validación experimental a escala en ambientes controlados. Línea de robótica marítima con énfasis en navegación asistida y evitación de obstáculos apoyada en antecedentes del anteproyecto y la revisión de literatura [1]–[6], [8].

2.2 Materiales y suministros

Casco impreso en 3D con dimensiones externas no mayores a 40 x 25 cm conforme al reglamento [1], [2].

Selladores y empaques para tapa superior y pasadores impermeables.

Batería LiPo 2200 miliamperios hora 3S 11.1 voltios 35C.

Motor sin escobillas con hélice acuática y variador compatible.

Receptor de radio control FS iA6B con transmisor estándar y función de apagado seguro.

Microcontrolador ESP32 para integración y registro básico.

Sensor ultrasónico sumergible JSN SR04M para detección frontal.

Servo para gobierno del timón.

Cables, conectores y conector de seguridad para corte rápido.

Boyas de espuma y cilindros plásticos como obstáculos.

Balanza digital, cinta métrica, pie de rey, cronómetro y termómetro de agua.

2.3 Aparatos e instrumentos de medición

Computador para registro de datos, bitácora y simulación.

Teléfono o cámara para evidencia fotográfica y de video.

Multímetro para verificación de tensiones y continuidad.

Regla de oleaje simple con marcas de medio centímetro para estimar altura de ola en piscina o lago pequeño.

2.4 Diseño experimental

Escenarios de prueba en piscina o cuerpo de agua tranquilo con oleaje leve entre cero y cinco centímetros medido en cresta a valle.

Disposición de obstáculos con tres densidades baja media alta manteniendo distancias mínimas de seguridad de un metro al perímetro.

Tres rutas de referencia mediante boyas con trayectorias en L S y circuito cerrado.

Dos modos de operación control manual y modo asistido definidos a partir del anteproyecto [1], [2].

Repeticiones mínimas por escenario cinco intentos por modo para permitir comparación estadística básica.

2.5 Procedimientos

2.5.1 Ensayos de impermeabilidad y flotación

Inspección visual de uniones y tapa.

Prueba de estanqueidad sin carga durante treinta minutos con inmersión parcial hasta línea de borda.

Prueba de flotación con masa equivalente a la carga de componentes y verificación de francobordo y estabilidad estática.

Registro de temperatura del agua y observaciones de ingreso de agua.

2.5.2 Integración eléctrica y de control

Verificación de continuidad y correcta polaridad con multímetro antes de energizar.

Instalación y prueba de corte seguro mediante conector accesible.

Configuración del receptor y verificación de failsafe a timón neutro y potencia cero.

Prueba unitaria de cada actuador motor y servo con pulsos de prueba y registro de consumo en vacío.

2.5.3 Calibración y verificación funcional

Calibración de servo a cero grados con referencia mecánica.

Ajuste de rango del sensor ultrasónico en aire y verificación en agua a distancias de veinte, cuarenta y sesenta centímetros.

Bitácora con fecha hora temperatura del agua y observaciones.

2.5.4 Protocolo de navegación en escenarios controlados

Colocación del bote en el punto de partida con orientación definida por la ruta.

Ejecución en modo manual y modo asistido en el mismo escenario.

Registro de tiempos de inicio fin eventos de desvío y contactos con obstáculos.

Registro de reingresos a la trayectoria y activaciones del apagado seguro si se requiere.

Criterios de abortaje si se excede la distancia de seguridad o si se detecta ingreso de agua.

2.6 Variables y métricas

Tasa de recorridos sin colisión en porcentaje.

Distancia mínima a obstáculos en centímetros medida con referencias visuales y marcas en obstáculos.

Desviación media respecto a la ruta en centímetros medida en puntos de control.

Tiempo de reingreso a ruta en segundos desde el evento de desvío.

Latencia de respuesta estimada como tiempo entre detección y corrección.

Observaciones cualitativas de estabilidad y gobernabilidad.

2.7 Tratamiento y análisis de datos

Hoja de cálculo con una fila por intento y columnas para escenario modo métricas y observaciones.

Descripción estadística con media desviación estándar e intervalos de confianza del 95% cuando el tamaño de muestra lo permita.

Comparación entre modos con prueba t para muestras independientes o prueba no paramétrica de Mann Whitney si no se cumple normalidad con referencia metodológica en la literatura [6]–[8].

Reporte gráfico en escala de grises cajas y bigotes para distribución de métricas y barras de error para promedios.

2.8 Consideraciones de seguridad y ética

Zonas de seguridad señalizadas sin público dentro del perímetro de un metro.

Elementos de protección personal para el equipo: botas antideslizantes y guantes.

Corte de energía accesible y verificado antes de cada intento.

Recuperación segura del prototipo con pétiga de salvamento sin ingreso del personal al agua.

3. RELATO DE DISEÑO, FABRICACIÓN Y PROBLEMAS

3.1 Diseño inicial del casco y elección de referencias

Qué hicimos:

Buscamos varias referencias de embarcaciones a escala y diseñamos un casco que cumpliera el requisito de dimensiones (máx. 40×25 cm). Nuestro objetivo fue adaptar un barco asistido al sistema que queríamos (teleoperado con asistencia para evasión de obstáculos).

Problema encontrado:

El diseño original no garantiza estabilidad: al reducir dimensiones y adaptar huecos para componentes, la distribución de volúmenes y centros de masa quedó inestable. Además, al reducir el tamaño los huecos no coincidían con los alojamientos de componentes (batería, ESC, eje, tapa).

Cómo lo resolvimos:

Hicimos iteraciones manuales de reducción y reubicación de huecos hasta conseguir una distribución de masas aceptable. Ajustamos el francobordo y el volumen de flotación para mantener la estabilidad. Repetimos pruebas de equilibrio antes de imprimir.

Evidencia:

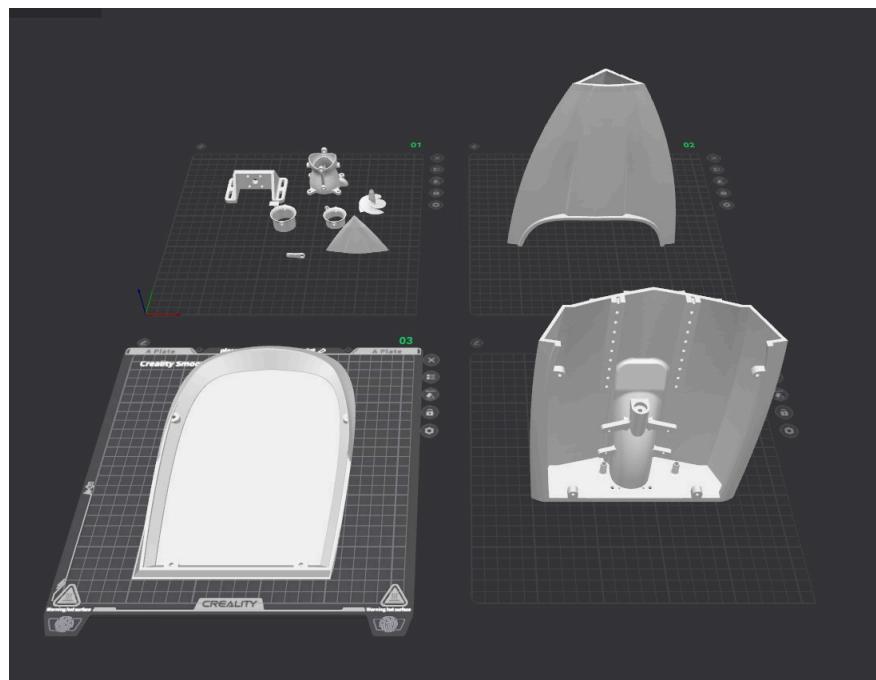


Figura 1. Rediseño del casco con huecos ajustados.



Figura 2. Rediseño de las piezas adicionales con huecos ajustados.

3.2 Problemas con la conversión entre programas CAD y SOLIDWORKS

Qué hicimos:

El modelo inicial estaba hecho con superficies en otro programa; intentamos importarlo a SOLIDWORKS para convertirlo en sólido y poder usar ARVIS/Gazebo.

Problema encontrado:

Al importar, el modelo seguía siendo de superficies (no sano como "solid body") y las físicas/simulaciones en SOLIDWORKS / Gazebo no funcionaban correctamente (intersecciones, volúmenes y centros inexactos).

Cómo lo resolvimos:

Rediseñamos las piezas problemáticas directamente en SOLIDWORKS creando geometrías sólidas (extrusiones, operaciones booleanas) en vez de depender de superficies. Donde no era posible, reconstruimos partes para garantizar la topología correcta. Volvimos a exportar y validar en el entorno de simulación.

Evidencia:

[IMAGEN 03: Modelo importado con errores (superficies)]

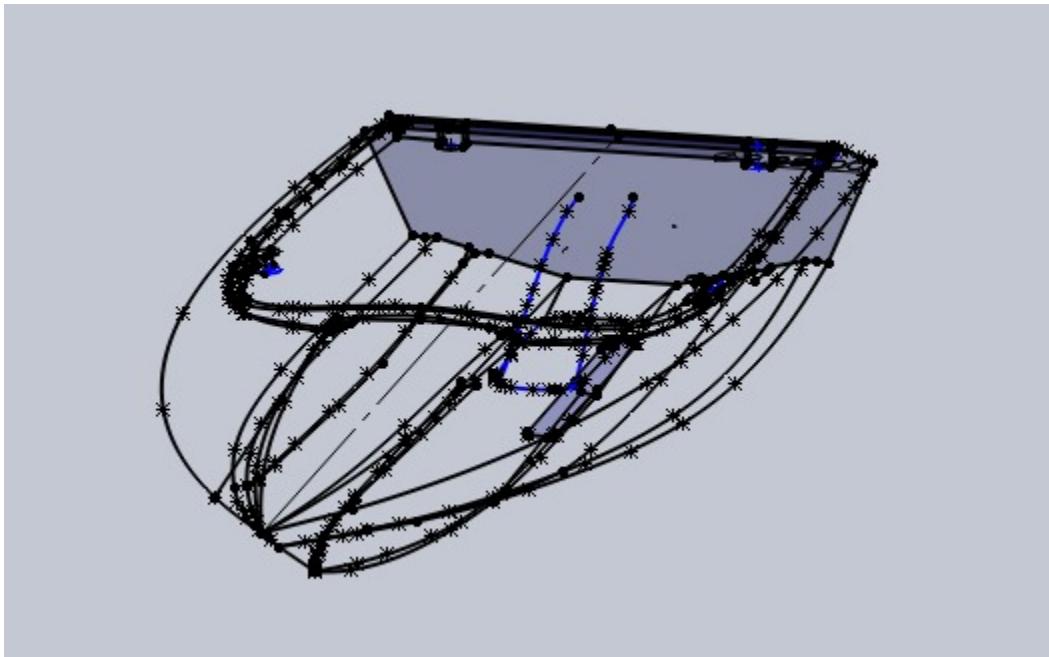


Figura 3. Barco reconstruido en SOLIDWORKS como sólido.

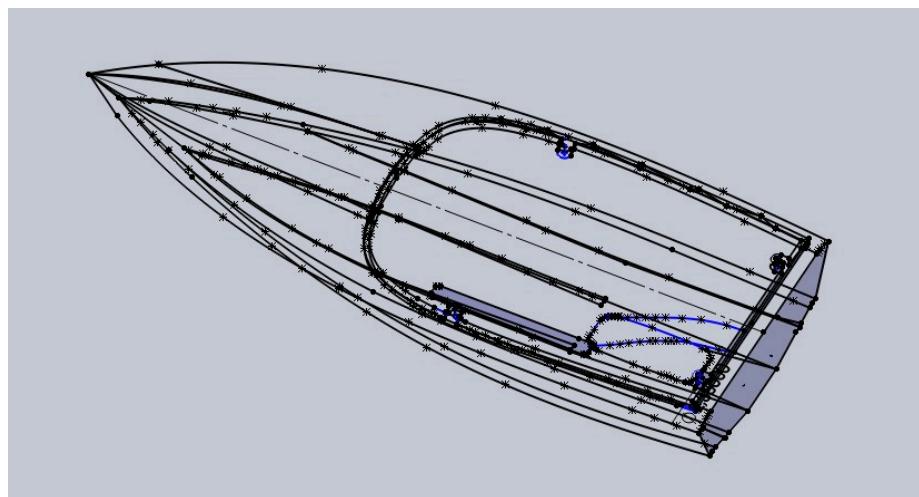


Figura 4. Barco reconstruido en SOLIDWORKS como sólido.

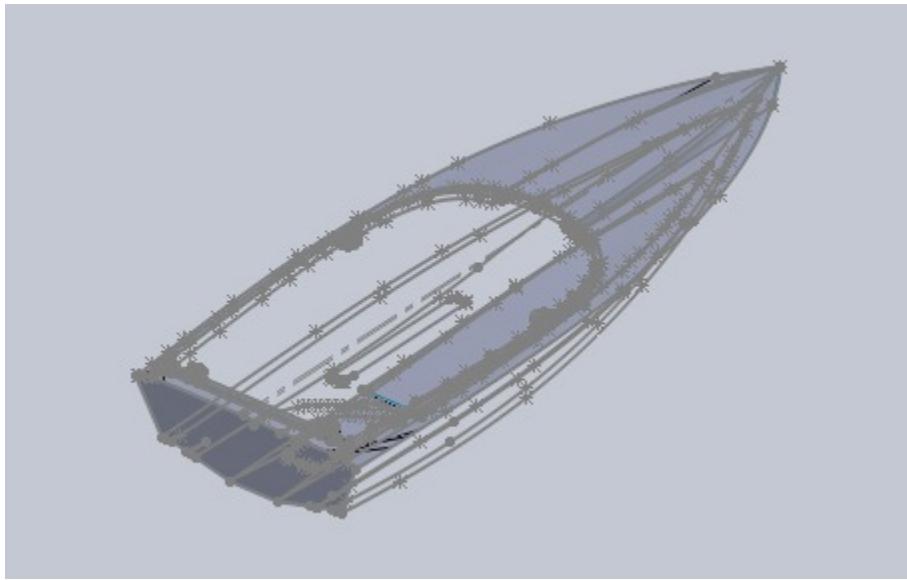


Figura 5. Barco reconstruido en SOLIDWORKS como sólido.

3.3 Impresión 3D en partes y problemas de acople

Qué hicimos:

La impresora disponible no permitía imprimir todo el casco de una sola pieza, por lo que lo dividimos en varias secciones y las imprimimos por separado.

Problema encontrado:

Las piezas impresas no calzaban bien: tolerancias diferentes, deformaciones por warping y variación entre impresiones ocasionaron que los huecos y pasadores no coincidieran. Hubo que imprimir varias veces y ajustar el diseño.

Cómo lo resolvimos:

- Ajustamos tolerancias de diseño (holguras y pasadores).
- Cambiamos la orientación de impresión para minimizar el desacople de las piezas
- Hicimos post-procesado (lijado y rectificado de encajes) y volvimos a imprimir las piezas que no encajaban.
- Repetimos impresiones hasta que el ajuste mecánico fue aceptable.

Evidencia:

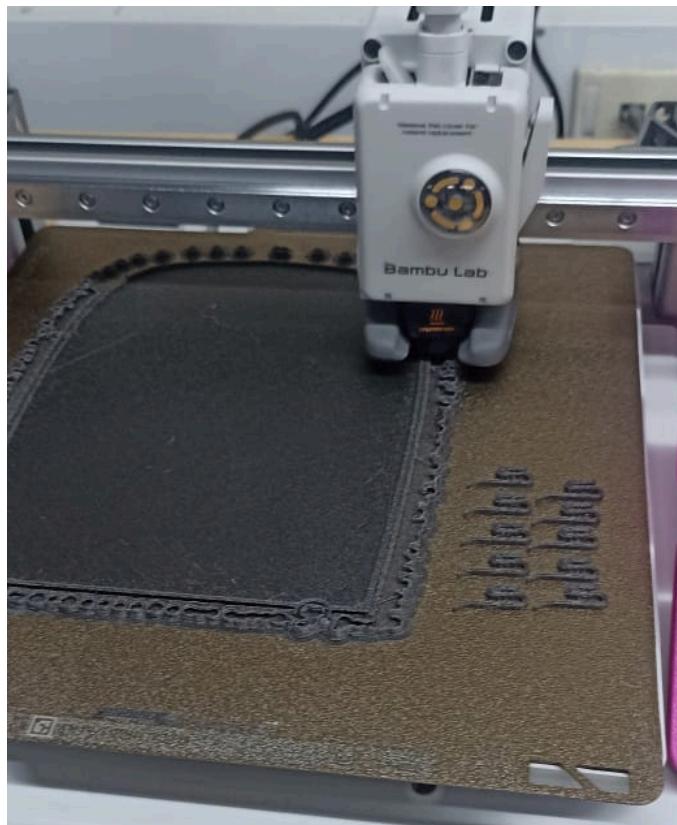


Figura 6. Tapa del barco durante el proceso de impresión.



Figura 7. Casco y tapa impresos y pegados antes del postprocesado.



Figura 8. Casco pegado antes del postprocesado parte inferior.



Figura 9. Casco pegado antes del postprocesado parte interior con prueba de eje.



Figura 10. Primer intento de acople en el sitio del eje.

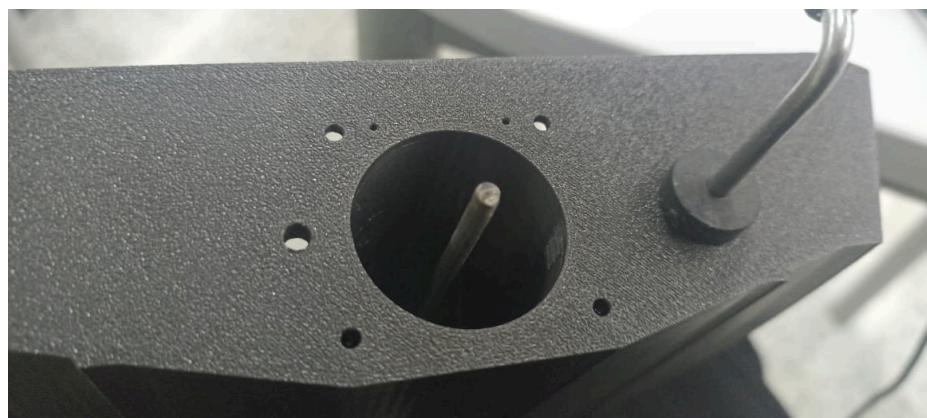


Figura 11. Primer intento de acople en el sitio del eje, con eje de prueba.

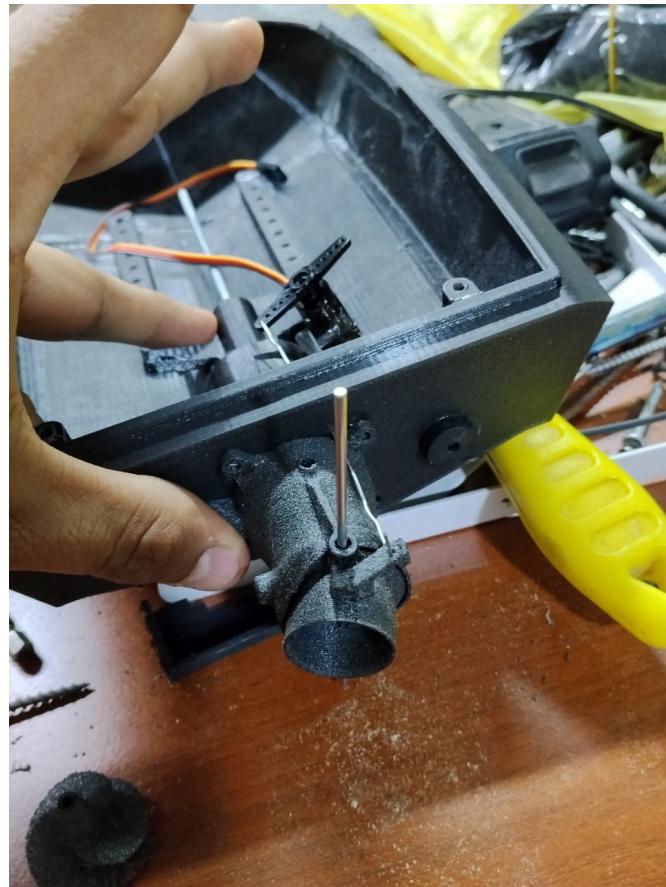


Figura 12. Postprocesado y prueba de encaje de piezas de la dirección.

3.4 Filtraciones de agua e impermeabilización

Qué hicimos:

Durante el ensamblaje y las primeras pruebas en agua detectamos filtraciones en juntas y alrededor de los ejes.

Problema encontrado:

La unión entre secciones y los alojamientos de los ejes permitían ingreso de agua. Las primeras soluciones (empaques simples) no fueron suficientes.

Cómo lo resolvimos:

- Decidimos aplicar resina epóxica industrial para sellar uniones críticas y reforzar el casco.
- Usamos silicona sellante tipo ventana en juntas secundarias.
- Probamos más de 10 combinaciones/procedimientos de sellado durante ~2 semanas hasta optimizar (secado, lijado entre capas, y control de curado).
- Implementamos empaques y pasadores revisados, y sellamos los alojamientos de ejes con anillas y resina.
- Hicimos ensayos de estanqueidad de 30 min y pruebas con carga para simular condiciones reales.

Evidencia:



Figura 13. Aplicación de impermeabilizante para tejados.

3.5 Fabricación de acoplos para ejes y motor brushless

Qué hicimos:

Al montar los ejes y el motor brushless, no existían acoplos comerciales adecuados en la ciudad, por lo que debíamos fabricarlos.

Problema encontrado:

No se conseguían acoplos estándar y los ejes no coincidían con el diámetro del motor, lo que impedía un montaje fiable y sin juego. Además el brushless no acoplaba directamente al eje impreso.

Cómo lo resolvimos:

- Diseñamos y mecanizamos (y/o imprimimos) acoplos específicos para el eje y para el eje del brushless.
- Incorporamos chavetas y ajustamos tolerancias para eliminar juego.
- Ensamblamos con resina y fijaciones mecánicas (tornillos/prensas) para garantizar resistencia al torque.
- Realizamos pruebas de esfuerzo y varias salidas de prueba para validar que no había deslizamiento ni fugas por

los alojamientos.

Evidencia:

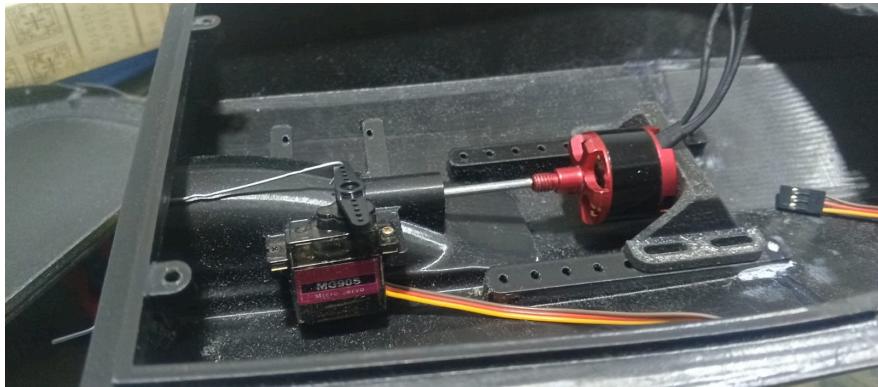


Figura 14. Acople montado en el brushless y eje.

3.6 Post-procesado y repetición de impresiones

Qué hicimos:

Después de imprimir, se realizó post-procesado (lijado, ajuste, ensamblado parcial) para optimizar encajes.

Problema encontrado:

El post-procesado no fue suficiente en algunas piezas porque la impresión tenía defectos dimensionales; por ello hubo que reimprimir varias veces y ajustar el modelo CAD.

Cómo lo resolvimos:

- Registramos cada iteración: qué se alteró y por qué.
- Modificamos el modelo 3D para reducir la necesidad de post-procesado en piezas críticas.
- Aumentamos pruebas de encaje hasta que las piezas quedaran con tolerancia aceptable.

3.7 Electrónica y sensores (incidencias durante integración)

Qué hicimos:

Integraron ESP32, sensor ultrasónico JSN-SR04M, ESC y motor brushless; hicieron pruebas unitarias de cada componente.

Problema encontrado:

- El sensor ultrasónico inicialmente devolvía lecturas «fuera de rango» o valores incoherentes.

Con ESP32 tuvimos problemas de drivers y comunicación seria en algunas máquinas.

El motor brushless necesitó calibración y protección ante falsos contactos.

Cómo lo resolvimos:

Ajustamos el código del sensor (timings y librerías) y calibrámos en condiciones reales de agua; cambiamos a librería del fabricante cuando fue necesario.

Actualizamos drivers del ordenador y configuramos el IDE para reconocer el ESP32.

Implementamos verificaciones de continuidad y protecciones en el cableado, y probamos el ESC con el arduino antes de usarlo con el ESP32.

Evidencia:

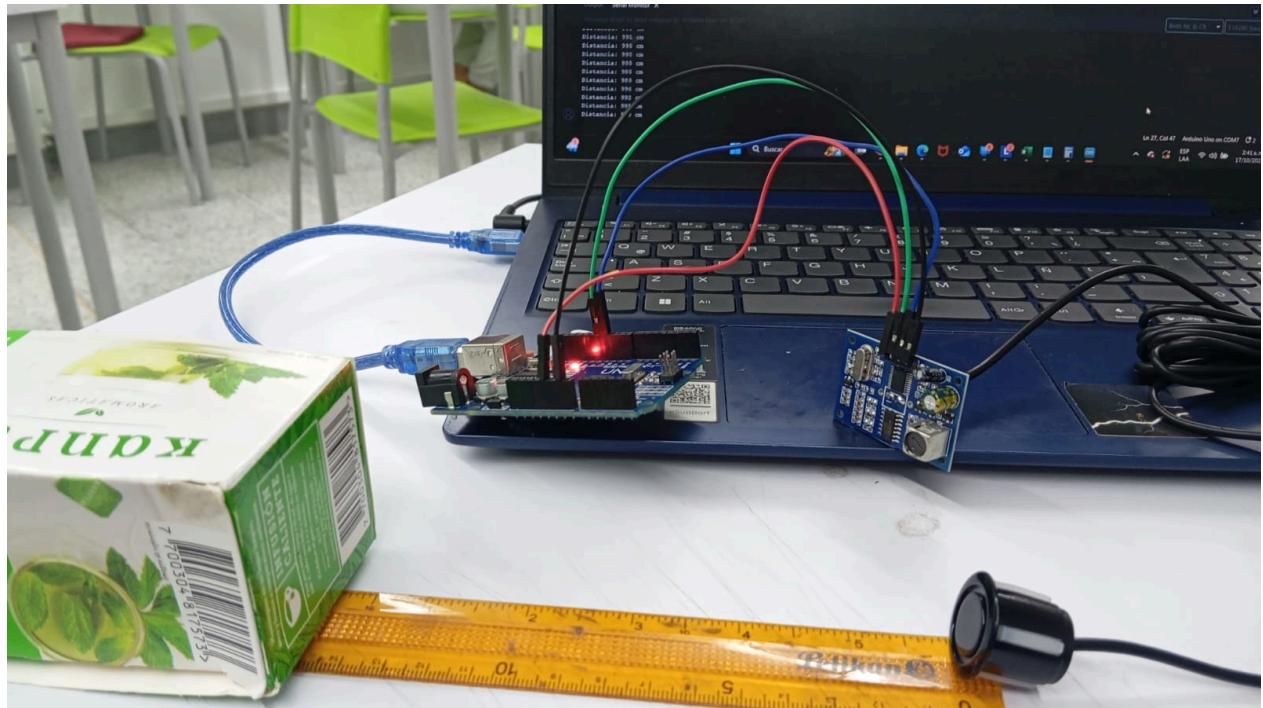


Figura 15. Serial monitor con lecturas del sensor ultrasónico.

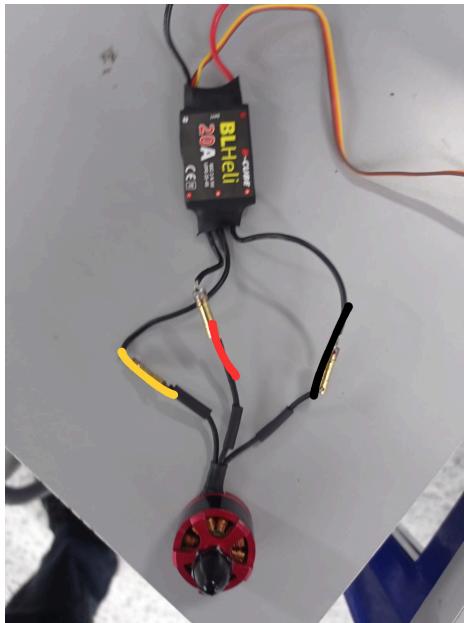


Figura 16. Prueba del motor brushless con ESC.

3.8 Ensamblaje final y pruebas de funcionamiento

Qué hicimos:

Montamos todos los subsistemas y ejecutamos pruebas en piscina y en simulador (Gazebo) para comparar modos.

Problema encontrado:

En algunas pruebas reales el comportamiento del timón y la propulsión originaron desviaciones por la dinámica del agua y por respuestas tardías del sensor en ciertas condiciones (reflejos, oleaje leve).

Cómo lo resolvimos:

Ajustamos parámetros de control (ganancias PID y tiempos de muestreo).

Definimos zonas de seguridad y lógica de respuesta (reducción de velocidad, maniobra evasiva).

Repetimos pruebas hasta tener consistencia entre lo simulado y lo real.

Evidencia:

```
class PIDController:  
    """  
        PID digital incremental usando la forma:  
        R[n] = R[n-1] + Kp(e[n]-e[n-1]) + Ki*(e[n]+e[n-1])/2 + Kd*(e[n]-2e[n-1]+e[n-2])  
        Implementado en compute(error) que devuelve R[n].  
    """  
  
    def __init__(self, Kp: float, Ki: float, Kd: float):  
        self.Kp = float(Kp)  
        self.Ki = float(Ki)  
        self.Kd = float(Kd)  
        # historial de errores  
        self.e_n = 0.0  
        self.e_n1 = 0.0  
        self.e_n2 = 0.0  
        # salida acumulada  
        self.R_n = 0.0  
  
    def compute(self, error: float) -> float:  
        # Actualizar historial  
        self.e_n2 = self.e_n1  
        self.e_n1 = self.e_n  
        self.e_n = float(error)  
  
        term_P = self.Kp * (self.e_n - self.e_n1)  
        term_I = self.Ki * (self.e_n + self.e_n1) / 2.0  
        term_D = self.Kd * (self.e_n - 2.0*self.e_n1 + self.e_n2)  
  
        self.R_n = self.R_n + term_P + term_I + term_D  
        return self.R_n  
  
    def reset(self):  
        self.e_n = self.e_n1 = self.e_n2 = 0.0  
        self.R_n = 0.0
```

Figura 17. Código de PID con realimentación.



Figura 18. Prueba en piscina.



Figura 19. Prueba en lago.

3.9 Impermeabilización final y validación con carga

Qué hicimos:

Una vez probados mecanismos y electrónica, hicimos la impermeabilización definitiva y pruebas con la batería y carga real.

Problema encontrado:

Al inicio, pequeñas infiltraciones en tapa y pasadores bajo carga; la vibración con el motor ayudaba a que algunas uniones fallaran.

Cómo lo resolvimos:

- Reforzamos tapa con resina epóxica y anclajes mecánicos adicionales.
- Aplicamos sellador en todos los pasadores y sellos.
- Realizamos un periodo de pruebas continuas con la embarcación en funcionamiento durante varias sesiones (simulando misiones largas) y supervisando el ingreso de agua.

Evidencia:



Figura 20. Casco con sellado final.



Figura 21. Registro de barco en competencia.

3.10 Ruptura del acople del modelo pequeño

El barco más pequeño —que usaba el mismo diseño general que el nuestro— presentó un problema grave:

el acople del eje se partía constantemente durante las pruebas.

Este acople transmitía el movimiento del motor brushless hacia el eje del propulsor, pero debido al tamaño reducido del casco y a las vibraciones generadas dentro del agua, el material no soportó las cargas repetidas.

Además, al ser más pequeño, la tolerancia era mucho menor y cualquier desalineación provocaba esfuerzos laterales que terminaban quebrándolo.

Cómo lo abordamos:

- Fabricamos un acople reforzado con mayor espesor.
- Ajustamos el alineamiento del motor y el eje para reducir la vibración.
- Cambiamos tornillos de fijación por unos más duros para evitar juego.

3.11 El eje giraba tan rápido que quemaba el sellado plástico

En nuestro modelo principal, durante pruebas prolongadas de navegación, observamos que el eje alcanzaba velocidades tan altas que generaba fricción y calentamiento en el área donde estaba el sellado plástico que impermeabiliza la salida del eje.

Esto provocó que el sellado plástico se fundiera parcialmente. Se abriera un espacio muy pequeño y el agua comenzara a filtrarse por ese punto.

Este problema apareció solo tras varias pruebas, porque al inicio el material todavía estaba rígido y fresco; pero con uso continuo, aceleraciones y vibraciones, el desgaste empezó a mostrarse.

Solución implementada:

- Usamos un refrigerante automotriz aplicado como lubricante enfriador en la zona del eje.
- Esto redujo la temperatura y el desgaste por fricción.
- Funcionó bien por varias sesiones de prueba y prácticamente eliminó la filtración.

3.12 Falla final: el eje se soldó al plástico por calor y terminó pegado

A pesar de que el refrigerante funcionó, el uso excesivo durante las pruebas finales produjo acumulación de calor. Después de varias corridas largas (de más de 20–25 minutos), el eje alcanzaba temperaturas tan altas que:

- Derretía progresivamente el material del casco, el plástico reblandecido se deformaba, y en un punto crítico, el eje quedó completamente pegado al material, bloqueándose por completo.

Esta fue una falla catastrófica porque inmovilizó el sistema de propulsión.

Cómo lo solucionamos:

- Cortamos el material alrededor del eje para liberarlo.
- Montamos un nuevo sistema de soporte con bujes metálicos, que disipaban mejor el calor.
- Reforzamos el área con resina epoxíca resistente a temperatura.
- Añadimos un canal interno para mantener lubricación constante y reducir fricción.

3.13 URDF – Construcción y corrección del modelo

Qué hicimos:

Armamos el URDF del barco definiendo cada link, cada joint y las conexiones entre partes. Organizamos la estructura del robot para poder visualizarlo en RViz, cargarlo en Gazebo y trabajar las inercias del modelo físico. También asignamos los meshes al URDF para que se viera el modelo tal cual lo diseñamos.

Problema encontrado:

El URDF inicial tenía varios errores porque el modelo venía de superficies, no de un sólido real. Esto hizo que las colisiones no funcionaran bien, las inercias quedaran mal calculadas y algunas piezas ni siquiera aparecieran porque los meshes no estaban bien referenciados. El URDF se abría en RViz, pero no en Gazebo, y tocó revisar link por link para identificar qué estaba mal.

Cómo lo resolvimos:

Rehicimos varias partes del URDF, definimos correctamente los iniciales, reorganizamos los padres e hijos en los joints y ajustamos las rutas de los meshes. Verificamos el URDF directamente en RViz para confirmar que la estructura estuviera cargando bien antes de pasar a Gazebo. Al final, logramos que el URDF se cargara de forma estable sin errores de estructura.

Evidencia:

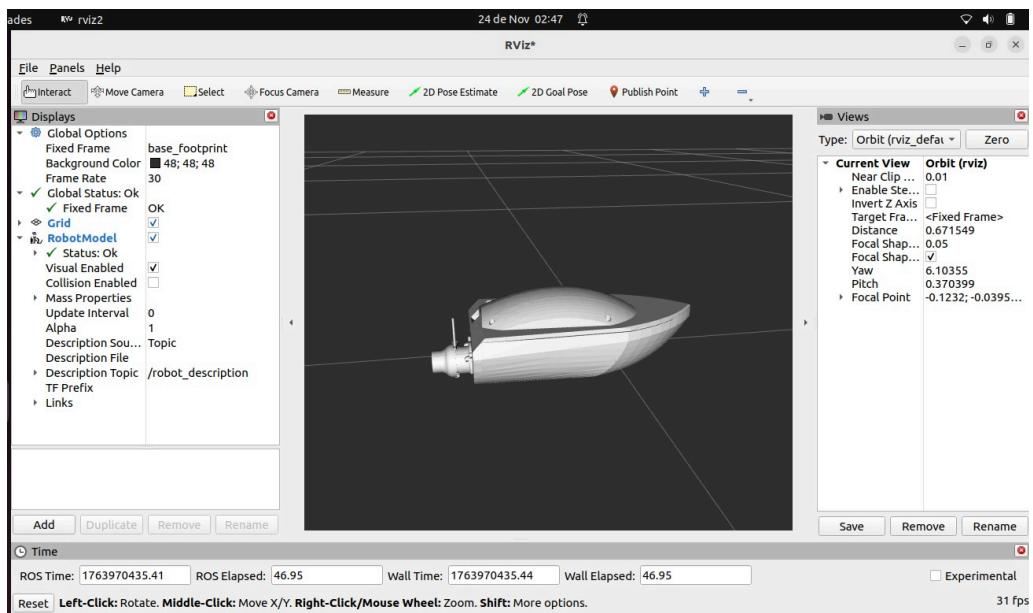


Figura 22. URDF cargado en RViz.

3.14 RViz – Visualización y detección del URDF correcto

Qué hicimos:

Usamos RViz para visualizar el URDF y asegurarnos de que los links, joints, orientaciones y mallas estuvieran correctamente organizados. RViz fue el primer paso para validar que el URDF estuviera estructuralmente correcto antes de mandarlo a Gazebo.

Problema encontrado:

Aunque hacíamos cambios en el URDF, RViz seguía mostrando una versión vieja. Ahí nos dimos cuenta de que ROS2 estaba cargando otro URDF diferente al que estábamos editando. El robot_state_publisher tenía guardada una versión antigua, así que por más que editáramos el archivo, RViz no mostraba los cambios.

Cómo lo resolvimos:

Usamos el comando:

```
ros2 param get /robot_state_publisher robot_description
```

Ahí vimos directamente cuál URDF estaba cargando ROS2. Después de corregir la ruta y recargar la descripción del robot, RViz empezó a mostrar el modelo correcto. Esto nos permitió validar cada cambio sin perder tiempo.

3.15 Gazebo – Carga del modelo y errores con versiones antiguas

Qué hicimos:

Cargamos el URDF en Gazebo para simular el comportamiento físico del barco. Configuramos los plugins básicos, las colisiones y las inercias para que la simulación fuera estable. También probamos tanto Gazebo Classic como Ignition para ver cuál funcionaba mejor.

Problema encontrado:

Gazebo no mostraba los cambios del URDF. Seguía cargando una versión antigua que estaba guardada en la carpeta oculta:

~/gazebo/models

Además, cuando actualizamos a Ignition, desaparecieron todas las carpetas donde estaban los meshes, lo que hizo que el modelo ni siquiera se visualizara. Gazebo mostraba el URDF vacío porque no encontraba los .dae y .stl.

Cómo lo resolvimos:

Borramos por completo los modelos viejos con:

```
rm -rf ~/gazebo/models
```

Luego creamos una carpeta de modelos nueva y restauramos todos los meshes manualmente. Una vez reorganizadas las rutas, Gazebo por fin mostró el modelo real del barco. Después ajustamos las inercias y las colisiones para que se comportara como un barco de 1.8–2 kg, y ya no se hundía ni flotaba torcido.

Evidencia:

[IMAGEN 07: Carpeta .gazebo/models antes de borrar]

[IMAGEN 08: Nueva carpeta con meshes restaurados]

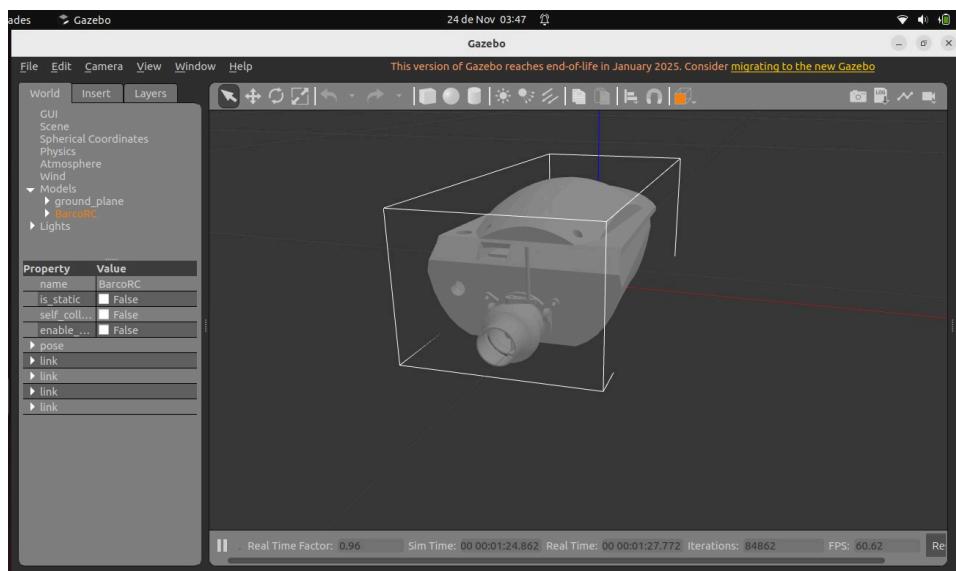


Figura 23. Gazebo mostrando el barco correctamente.

3.16 Ajuste de físicas, inercias y comportamiento del barco

Qué hicimos:

Trabajamos las inercias, masas y colisiones de cada parte del barco dentro del URDF para que el comportamiento en Gazebo fuera realista. Ajustamos peso total, distribución interna y los volúmenes de colisión para que la simulación respondiera como un barco de verdad.

Problema encontrado:

El barco se hundía o flotaba mal cuando las inercias no estaban bien configuradas. El peso estaba mal distribuido y algunas partes del casco tenían colisiones que chocaban entre sí. Además, al no tener los meshes originales después de la actualización, Gazebo no interpretaba bien las formas del modelo.

Cómo lo resolvimos:

Revisamos cada link y ajustamos las inercias con valores coherentes al peso real del barco (1.8–2 kg). Corregimos colisiones que estaban mal ubicadas y organizamos el centro de masa para que el barco flotara de forma estable. Después de varias pruebas el movimiento se veía natural dentro del simulador.

Evidencia:

[IMAGEN 10: Ajustes de inerciales]

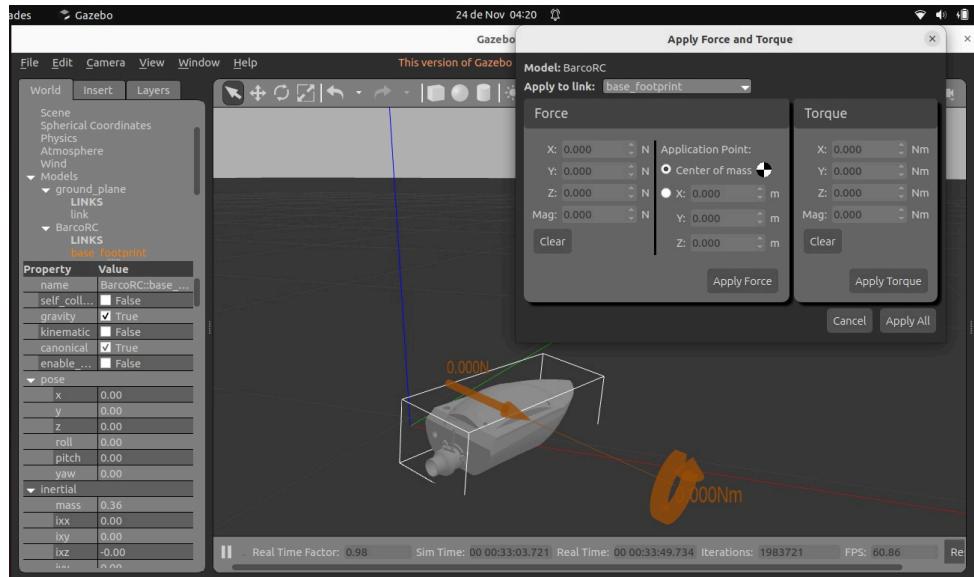


Figura 24. Gazebo mostrando el barco flotando estable.

3.17 Pendientes finales en el simulador

Qué hicimos:

Dejamos listo todo lo que necesitábamos para avanzar a la simulación final: asentar la física del barco, cargar bien el URDF en RViz y en Gazebo, y dejar los iniciales listos para pruebas.

Problema encontrado:

A pesar de que el modelo ya cargaba bien, todavía faltaban tres cosas importantes:

- animar las partes móviles (servo, timón)
- integrar las físicas del mundo con agua
- configurar el teleop desde el PC

Cómo lo resolvimos:

Dejamos el proyecto listo para integrar esas partes. Las físicas del barco ya están bien, la estructura del URDF está corregida y Gazebo ya muestra todo sin errores. El teleop se piensa hacer desde el PC porque no había control disponible.

3.18 Lecciones aprendidas y recomendaciones prácticas

- Diseñar para fabricación: trabajar desde un inicio en sólidos en el CAD para evitar conversiones problemáticas.
- Tolerancias y pruebas iterativas: prever impresión en partes y dejar holguras controladas para encajes.
- Impermeabilización por fases: planear varias etapas (prueba en seco, prueba con carga, prueba dinámica) y prever resina epóxica como solución definitiva.
- Stock de acoplos: cuando en la ciudad no se encuentran piezas, diseñar acoplos estándar reutilizables y testearlos con prototipos sencillos.
- Registro riguroso: llevar bitácora con fecha/hora/temperatura/observaciones en cada prueba (útil para las tablas que van a diligenciar).
- Pruebas de sensores en agua: calibrar el sensor en condiciones reales (el mismo cuerpo de agua) porque las lecturas en aire no son equivalentes.
- Estos problemas nos enseñaron que, en sistemas de propulsión pequeños y de alta velocidad, el control térmico del eje es tan importante como la impermeabilización.
- También reforzaron la idea de que las piezas impresas en 3D no siempre soportan cargas continuas a alta temperatura, y que es necesario combinar materiales metálicos y lubricación adecuada cuando se trabaja con brushless de alta potencia.

IMPLEMENTACIÓN DEL CÓDIGO Y TODA LA PARTE DE GEOMETRÍA

Cuando empezamos con este código del barco lo que hicimos fue algo muy parecido a lo que hicimos en la vida real: ir armando todo por partes, porque si tratábamos de hacerlo todo junto eso no iba a correr nunca. Entonces empezamos por lo más básico, que era poder mover el barco en el simulador, que reconociera el motor, que reconociera el servo y que supiera hacia dónde estaba el frente del barco.

Lo primero fue montar lo del PID, pero en verdad lo que hicimos ahí fue más copiar la estructura del taller y ajustarla a lo que estábamos usando (RPM y servo). La idea era que el motor no pegara tirones sino que subiera suave y que el servo no se fuera de una para un lado. Eso lo pusimos en una clase aparte para no mezclar todo.

Después pasamos al tema de la geometría, que fue donde más nos tocó “pensar” porque teníamos que decirle al código dónde estaba el motor, dónde estaba la hélice y dónde quedaba la IMU respecto al barco. Eso lo pusimos como posiciones en X y Y, básicamente como si uno viera el barco desde arriba y pusiera puncitos: “el motor va aquí”, “la IMU va aquí”, “la hélice sale desde aquí”. Eso lo hicimos así para que el código supiera desde dónde empujar y también cómo rotar el barco.

Con esa geometría, el código sabe calcular el empuje según el ángulo del servo y eso es lo que hace que el barco gire. En palabras simples: si el servo apunta recto, el barco va derecho; si apunta un poquito a la izquierda, el barco gira; y así.

Después seguimos con lo que sería la dinámica, que es básicamente decir: “si el barco tiene tanta masa y el motor empuja tanto, cuánto avanza”. Eso lo hicimos paso a paso con ecuaciones muy simples, nada del otro mundo: velocidad, posición, y que el barco vaya sumando su movimiento. También hicimos que la orientación (el yaw) se vaya acumulando, porque si no el barco se movería sin girar y eso no refleja la vida real.

Luego sí pasamos a la parte “bonita”, que fue el algoritmo para ir a un objetivo. Ahí hicimos lo mismo que el robot diferencial:

- primero tratamos de girar hacia donde queremos,
- luego avanzamos,
- si se desvía, corrige.

Y todo eso lo hace con la misma idea del barco real: si hay mucho error de ángulo, baja la velocidad; si está alineado, acelera más. Esa lógica la copiamos del taller, pero adaptada al barco.

Finalmente hicimos las gráficas y el guardado del archivo CSV para tener el registro, porque la idea es mostrar cómo avanzó, cómo giró, cómo respondieron el motor y el servo, etc. Eso nos permitió comprobar que el algoritmo sí estaba funcionando como queríamos.

CODIGO ODOMETRÍA

```
#!/usr/bin/env python3
```

;;;;;

run_boat.py

Simulador y odometría del barco con thrust-vectoring (motor brushless + servo).

Permite entrada por CLI y guarda CSV + PNG.

Código paso-a-paso y documentado para sustentar en el taller.

;;;;;

```
import math  
  
import csv  
  
import argparse  
  
from typing import Dict, Tuple  
  
import numpy as np  
  
import matplotlib.pyplot as plt  
  
  
# -----  
  
# 1) CLASE PID (ecuación incremental tal como en el taller)  
# -----
```

class PIDController:

""""

PID digital incremental usando la forma:

$$R[n] = R[n-1] + K_p(e[n]-e[n-1]) + K_i*(e[n]+e[n-1])/2 + K_d*(e[n]-2e[n-1]+e[n-2])$$

Implementado en compute(error) que devuelve R[n].

""""

def __init__(self, Kp: float, Ki: float, Kd: float):

self.Kp = float(Kp)

self.Ki = float(Ki)

self.Kd = float(Kd)

historial de errores

self.e_n = 0.0

self.e_n1 = 0.0

self.e_n2 = 0.0

salida acumulada

self.R_n = 0.0

def compute(self, error: float) -> float:

Actualizar historial

self.e_n2 = self.e_n1

self.e_n1 = self.e_n

self.e_n = float(error)

*term_P = self.Kp * (self.e_n - self.e_n1)*

*term_I = self.Ki * (self.e_n + self.e_n1) / 2.0*

*term_D = self.Kd * (self.e_n - 2.0*self.e_n1 + self.e_n2)*

```
self.R_n = self.R_n + term_P + term_I + term_D
```

```
return self.R_n
```

```
def reset(self):
```

```
self.e_n = self.e_n1 = self.e_n2 = 0.0
```

```
self.R_n = 0.0
```

```
# -----
```

```
# 2) GEOMETRÍA: joints y links (motor -> hélice -> IMU)
```

```
# -----
```

```
class BoatGeometry:
```

```
"""
```

Almacena distancias físicas que relacionan joints y links:

- *motor_pos_body*: (x,y) del eje del motor en marco cuerpo

- *prop_offset*: desplazamiento longitudinal desde eje motor a hélice

- *imu_pos_body*: (x,y) posición del IMU en marco cuerpo

```
"""
```

```
def __init__(self,
```

```
    motor_pos_body: Tuple[float,float]=(0.0, -0.08),
```

```
    prop_offset: float=-0.08,
```

```
    imu_pos_body: Tuple[float,float]=(0.05, 0.02)):
```

```
    self.motor_pos_body = np.array(motor_pos_body, dtype=float)
```

```
    self.prop_offset = float(prop_offset)
```

```
    self.imu_pos_body = np.array(imu_pos_body, dtype=float)
```

```
def prop_position_body(self) -> np.ndarray:  
    # Posición de la hélice en el marco del cuerpo  
    return self.motor_pos_body + np.array([self.prop_offset, 0.0])
```

```
# -----
```

3) MODELOS MOTOR / SERVO

```
# -----
```

```
class MotorModel:
```

```
"""
```

Modelo simple RPM -> thrust: $F = a_{thrust} * rpm^2$

a_{thrust} se calibra experimentalmente.

```
"""
```

```
def __init__(self, a_thrust: float = 6.0e-6):
```

```
    self.a_thrust = float(a_thrust)
```

```
def thrust_from_rpm(self, rpm: float) -> float:
```

```
    return self.a_thrust * (rpm**2)
```

```
class ServoModel:
```

```
"""
```

Convierte comando [-1..1] a ángulo delta en radianes.

max_deg define el límite +/- (por ejemplo 45°).

```
"""
```

```
def __init__(self, max_deg: float = 45.0):
```

```
    self.max_deg = float(max_deg)
```

```
def angle_from_command(self, cmd: float) -> float:  
    # cmd esperado en [-1,1]  
  
    c = max(-1.0, min(1.0, float(cmd)))  
    return math.radians(c * self.max_deg)  
  
  
def command_from_angle(self, angle_rad: float) -> float:  
    # inverse: angle -> normalized command [-1,1]  
  
    deg = math.degrees(angle_rad)  
  
    return max(-1.0, min(1.0, deg / self.max_deg))  
  
  
# -----  
  
# 4) ROBOT/ BARCO: dinámica simplificada + odometría (estado)  
# -----  
  
class BoatRobot:  
  
    """  
  
    Estado del barco, dinámica (simplificada), integración de pose.  
  
    Variables:  
  
    m: masa (kg)  
  
    I_z: momento de inercia (kg*m^2)  
  
    Estados:  
  
    u,v: velocidades en body (m/s)  
  
    r: yaw rate (rad/s)  
  
    x,y,psi: pose en world  
  
    """  
  
    def __init__(self,  
                 m: float = 1.3,
```

```
I_zz:float = 0.05,  
geom:BoatGeometry = None,  
motor:MotorModel = None,  
servo:ServoModel = None,  
dt:float = 0.02);  
  
self.m = float(m)  
  
self.I_zz = float(I_zz)  
  
self.geom = BoatGeometry() if geom is None else geom  
self.motor = MotorModel() if motor is None else motor  
self.servo = ServoModel() if servo is None else servo  
self.dt = float(dt)  
  
  
# estados en body  
  
self.u = 0.0  
  
self.v = 0.0  
  
self.r = 0.0  
  
  
# pose en world  
  
self.x = 0.0  
  
self.y = 0.0  
  
self.psi = 0.0  
  
  
# comandos (rpm y servo norm)  
self.rpm_cmd = 0.0  
self.servo_cmd = 0.0
```

coeficientes de arrastre lineales (aprox)

self.k_u = 1.8

self.k_v = 6.0

self.k_r = 1.2

mapeo empírico rotación por chorro

self.k_psi = 0.6

def reset_pose(self):

self.u = self.v = self.r = 0.0

self.x = self.y = self.psi = 0.0

def compute_forces(self):

"""Calcula fuerzas en body frame y momento a partir de rpm y servo_cmd actuales."""

T = self.motor.thrust_from_rpm(self.rpm_cmd) # empuje (N)

delta = self.servo.angle_from_command(self.servo_cmd) # rad

Fx = T * math.cos(delta)

Fy = T * math.sin(delta)

prop_pos = self.geom.prop_position_body()

Momento (2D): $M_z = r_x * F_y - r_y * F_x$

$M_z = prop_pos[0]*Fy - prop_pos[1]*Fx$

return Fx, Fy, Mz, delta, T

def step_dynamics(self):

"""Integra dinámica simple y actualiza pose y velocidades."""

Fx, Fy, Mz, delta, T = self.compute_forces()

```
# aceleraciones en body frame con arrastre lineal
```

$$ax = Fx / self.m - self.k_u * self.u$$

$$ay = Fy / self.m - self.k_v * self.v$$

$$yaw_acc = Mz / self.I_z - self.k_r * self.r$$

```
# integrar velocidades
```

$$self.u += ax * self.dt$$

$$self.v += ay * self.dt$$

$$self.r += yaw_acc * self.dt$$

```
# convertir velocidades a marco world y actualizar pose
```

$$\cos\psi = \text{math.cos}(self.\psi); \sin\psi = \text{math.sin}(self.\psi)$$

$$v_world_x = self.u * \cos\psi - self.v * \sin\psi$$

$$v_world_y = self.u * \sin\psi + self.v * \cos\psi$$

$$self.x += v_world_x * self.dt$$

$$self.y += v_world_y * self.dt$$

$$self.\psi += self.r * self.dt$$

```
# normalizar psi
```

$$self.\psi = \text{math.atan2}(\text{math.sin}(self.\psi), \text{math.cos}(self.\psi))$$

```
return ax, ay, yaw_acc, delta, T
```

```
def inverse_control_map(self, V_des: float, W_des: float) -> Tuple[float, float, float, float]:
```

|||||

Mapea V_{des} y W_{des} a rpm_{des} y $servo_{cmd_des}$.

Modelo simplificado:

$$\sin(\delta) = W_{des} / (k_{psi} * V)$$

$$T_{needed} * \cos(\delta) = m * (k_u * V) \quad (\text{equilibrio approx})$$

.....

$$V = \max(1e-3, \text{abs}(V_{des}))$$

$$\sin_{delta} = W_{des} / (\text{self}.k_{psi} * V)$$

$$\sin_{delta} = \max(-1.0, \min(1.0, \sin_{delta}))$$

$$\delta = \text{math.asin}(\sin_{delta})$$

$$\cos_d = \max(1e-3, \text{math.cos}(\delta))$$

$$T_{needed} = \text{self}.m * (\text{self}.k_u * V) / \cos_d$$

$$rpm = \text{math.sqrt}(\max(0.0, T_{needed} / \text{self}.motor.a_{thrust}))$$

$$\text{servo_cmd} = \text{self.servo.command_from_angle}(\delta)$$

return rpm, servo_cmd, delta, T_needed

5) ALGORITMO MOVIMIENTO (semejante al Taller #4)

def boat_move_to_target(robot: BoatRobot,

theta_target: float,

distance_target: float,

alpha: float = 2.0,

beta: float = 0.5,

pid_rpm: PIDController = None,

pid_servo: PIDController = None,

max_time: float = 60.0) -> Dict:

.....

Ejecuta el movimiento: gira y avanza *distance_target*.

Devuelve un diccionario 'history' con registros.

""""

$t = 0.0$

$dt = robot.dt$

robot.reset_pose()

PIDs si no se pasan por fuera

if *pid_rpm* is None:

pid_rpm = PIDController(0.0009, 0.00005, 0.0)

if *pid_servo* is None:

pid_servo = PIDController(2.0, 0.01, 0.005)

pid_rpm.reset(); pid_servo.reset()

history = {'time':[], 'x':[], 'y':[], 'psi':[],
'alpha_t':[], 'delta_t':[], 'V':[], 'W':[],
'rpm_cmd':[], 'servo_cmd':[], 'rpm_des':[], 'delta_des':[],
'error_theta':[], 'error_dist':[]}

alpha_t = 0.0

delta_t = 0.0

V_prev = 0.0

W_prev = 0.0

while *delta_t* < *distance_target* and *t* < *max_time*:

1. errores

$error_theta = theta_target - alpha_t$

$error_dist = distance_target - delta_t$

2. leyes deseadas (mismo estilo del taller)

$V_des = 0.9 * math.exp(-(error_theta**2)/(2*alpha**2))$

$sigmoid = 2.0 / (1.0 + math.exp(-error_theta/beta)) - 1.0$

$W_des = 1.0 * sigmoid$

3. mapeo inverso a rpm y servo

$rpm_des, servo_des_cmd, delta_des, T_need = robot.inverse_control_map(V_des, W_des)$

4. PID incremental para ajustar comandos

$rpm_error = rpm_des - robot.rpm_cmd$

$incr_rpm = pid_rpm.compute(rpm_error)$

$servo_error = servo_des_cmd - robot.servo_cmd$

$incr_servo = pid_servo.compute(servo_error)$

$robot.rpm_cmd = max(0.0, min(10000.0, robot.rpm_cmd + incr_rpm))$

$robot.servo_cmd = max(-1.0, min(1.0, robot.servo_cmd + incr_servo))$

5. step dinámica

$ax, ay, yaw_acc, delta_act, T_act = robot.step_dynamics()$

6. lectura V y W actuales (aprox)

$V = robot.u$

$W = robot.r$

7. integrar alpha_t y delta_t (trapezoidal)

$alpha_t += dt * (W + W_{prev}) / 2.0$

$delta_t += dt * (V + V_{prev}) / 2.0$

$V_{prev} = V; W_{prev} = W$

$t += dt$

8. guardar historial

$history['time'].append(t)$

$history['x'].append(robot.x)$

$history['y'].append(robot.y)$

$history['psi'].append(robot.psi)$

$history['alpha_t'].append(alpha_t)$

$history['delta_t'].append(delta_t)$

$history['V'].append(V)$

$history['W'].append(W)$

$history['rpm_cmd'].append(robot.rpm_cmd)$

$history['servo_cmd'].append(robot.servo_cmd)$

$history['rpm_des'].append(rpm_des)$

$history['delta_des'].append(delta_des)$

$history['error_theta'].append(error_theta)$

$history['error_dist'].append(error_dist)$

safety

if len(history['time']) > 200000:

break

return history

6) FUNCIONES AUX: *plotting y guardado CSV*

def plot_and_save(history: Dict, out_prefix: str = "boat_output"):

1) Trayectoria XY

plt.figure(figsize=(8,6))

plt.plot(history['x'], history['y'], '-o', markersize=3)

plt.xlabel("X (m)"); plt.ylabel("Y (m)")

plt.title("Trayectoria del barco")

plt.axis('equal'); plt.grid(True)

plt.savefig(out_prefix + "_trajectory.png", bbox_inches='tight', dpi=150)

plt.close()

2) Ángulo y distancia

plt.figure(figsize=(10,4))

plt.subplot(1,2,1)

plt.plot(history['time'], [math.degrees(a) for a in history['alpha_t']])

plt.xlabel("t (s)"); plt.ylabel("at (deg)"); plt.grid(True)

plt.subplot(1,2,2)

plt.plot(history['time'], history['delta_t'])

plt.xlabel("t (s)"); plt.ylabel("δt (m)"); plt.grid(True)

plt.tight_layout()

```
plt.savefig(out_prefix + "_progress.png", bbox_inches='tight', dpi=150)
```

```
plt.close()
```

3) RPM y servo

```
plt.figure(figsize=(10,4))
```

```
plt.subplot(1,2,1)
```

```
plt.plot(history['time'], history['rpm_cmd'], label='rpm_cmd')
```

```
plt.plot(history['time'], history['rpm_des'], '--', label='rpm_des')
```

```
plt.xlabel("t (s)"); plt.ylabel("RPM"); plt.legend(); plt.grid(True)
```

```
plt.subplot(1,2,2)
```

```
plt.plot(history['time'], history['servo_cmd'], label='servo_cmd')
```

```
plt.plot(history['time'], history['delta_des'], '--', label='delta_des (rad)')
```

```
plt.xlabel("t (s)"); plt.legend(); plt.grid(True)
```

```
plt.tight_layout()
```

```
plt.savefig(out_prefix + "_commands.png", bbox_inches='tight', dpi=150)
```

```
plt.close()
```

4) Errores

```
plt.figure(figsize=(8,4))
```

```
plt.plot(history['time'], [math.degrees(e) for e in history['error_theta']], label='err_theta (deg)')
```

```
plt.plot(history['time'], history['error_dist'], label='err_dist (m)')
```

```
plt.xlabel("t (s)"); plt.legend(); plt.grid(True)
```

```
plt.savefig(out_prefix + "_errors.png", bbox_inches='tight', dpi=150)
```

```
plt.close()
```

Guardar CSV

csv_name = out_prefix + "_trajectory.csv"

with open(csv_name, 'w', newline='') as f:

writer = csv.writer(f)

writer.writerow(['t','x','y','psi','alpha_t','delta_t','rpm_cmd','servo_cmd'])

for i, t in enumerate(history['time']):

writer.writerow([t, history['x'][i], history['y'][i], history['psi'][i],

history['alpha_t'][i], history['delta_t'][i],

history['rpm_cmd'][i], history['servo_cmd'][i]])

print("Guardado CSV:", csv_name)

print("Guardadas imágenes: ", out_prefix + "_trajectory.png", out_prefix + "_progress.png",

out_prefix + "_commands.png", out_prefix + "_errors.png")

7) CLI: recibir parámetros para la simulación

def parse_args():

p = argparse.ArgumentParser(description="Simulador odometría barco - run_boat.py")

p.add_argument('--theta', type=float, default=43.0, help='Ángulo objetivo en grados (ej. 43)')

p.add_argument('--distance', type=float, default=2.0, help='Distancia objetivo en metros')

p.add_argument('--mass', type=float, default=1.3, help='Masa del barco (kg)')

p.add_argument('--I_z', type=float, default=0.05, help='Momento de inercia I_z (kg*m^2)')

p.add_argument('--prop_offset', type=float, default=-0.08, help='Offset hélice respecto eje motor (m)')

p.add_argument('--motor_x', type=float, default=0.0, help='x motor pos body (m)')

```

p.add_argument('--motor_y', type=float, default=-0.08, help='y motor pos body (m)')
p.add_argument('--imu_x', type=float, default=0.05, help='IMU x pos (m)')
p.add_argument('--imu_y', type=float, default=0.02, help='IMU y pos (m)')
p.add_argument('--a_thrust', type=float, default=6.0e-6, help='Constante empuje a_thrust (N/RPM^2)')
p.add_argument('--dt', type=float, default=0.02, help='Paso de integración dt (s)')
p.add_argument('--out', type=str, default='boat_output', help='Prefijo archivos de salida')
return p.parse_args()

```

8) MAIN: construir objetos, ejecutar, plotear

def main():

args = parse_args()

theta_rad = math.radians(args.theta)

geom = BoatGeometry(motor_pos_body=(args.motor_x, args.motor_y),

prop_offset=args.prop_offset,

imu_pos_body=(args.imu_x, args.imu_y))

motor = MotorModel(a_thrust=args.a_thrust)

servo = ServoModel(max_deg=45.0)

boat = BoatRobot(m=args.mass, I_z=args.I_z, geom=geom, motor=motor, servo=servo, dt=args.dt)

pid_rpm = PIDController(0.0009, 0.00005, 0.0)

pid_servo = PIDController(2.0, 0.01, 0.005)

print(f"Simulando: theta={args.theta} deg, distance={args.distance} m, mass={args.mass} kg, I_z={args.I_z}")

history = boat_move_to_target(boat, theta_rad, args.distance,

```
alpha=2.0, beta=0.5,  
pid_rpm=pid_rpm, pid_servo=pid_servo, max_time=300.0)  
  
plot_and_save(history, out_prefix=args.out)  
print("Simulación completada. Archivos guardados con prefijo:", args.out)  
  
if __name__ == "__main__":  
    main()
```

3. RESULTADOS Y DISCUSIÓN

En esta etapa no se presentan tablas ni gráficos de desempeño porque las pruebas aún se concentran en impermeabilidad, flotación y verificación funcional. Se documenta el estado del avance y se dejan preparadas las plantillas de recolección de datos en el apéndice para usarlas cuando inicien los ensayos controlados.

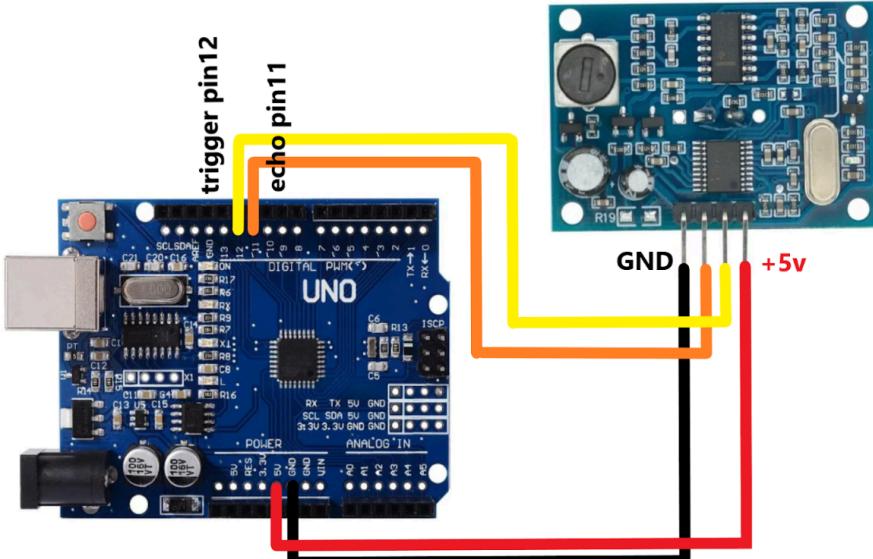
3.1 Estado de pruebas de impermeabilidad y flotación

En esta etapa realizamos pruebas de impermeabilidad y flotación sin carga y con masa equivalente a los componentes. No se presentaron filtraciones visibles durante el tiempo de observación o estas se limitaron a pequeñas trazas cerca de la tapa. En flotación el casco mantuvo un margen de borda suficiente y una estabilidad estática adecuada para el propósito del ensayo. Estas observaciones orientan ajustes finos en empaques y pasadores antes de integrar el sistema completo.

3.2 Integración y pruebas unitarias

Se verificó la continuidad eléctrica y la polaridad antes de energizar. El actuador de timón respondió a comandos básicos de posición sin enganches y el motor giró de forma estable en vacío. El corte de energía funcionó de manera inmediata al accionar el conector de seguridad. Estas verificaciones reducen el riesgo en las siguientes pruebas y confirman que la integración progresó según lo previsto.

Tabla 1. SENSOR DE ULTRASONIDO JSN-SR04M-2

PRUEBA UNITARIA EN TIEMPO REAL	
Requerimientos	
	<ul style="list-style-type: none"> • Correcto funcionamiento del sensor y calibración de las mediciones.
Tipo de prueba	Unitaria
Hardware requerido	<ul style="list-style-type: none"> • Arduino uno • Computador • Sensor de ultrasonido JSN-SR04M-2 • Modulo SR04M-2 • Jumpers • Regla • Objeto cualquiera
Software requerido	<ul style="list-style-type: none"> • Arduino IDE
Objetivo	<ul style="list-style-type: none"> • Asegurar el correcto funcionamiento del sensor y la correcta medición de las distancias en un ambiente controlado.
Descripción	
Procedimiento	<ol style="list-style-type: none"> 1. Realizar las conexiones entre el sensor de ultrasonido, su módulo y el arduino uno.  <p>Figura 9. Esquema de conexiones entre el modulo SR04M-2 y el arduino UNO.</p> <ol style="list-style-type: none"> 1. Colocar el objeto sobre una regla y el sensor, en el punto cero de la regla.

2. Conectar el arduino al computador y abrir el software.
3. Cargar el código de prueba para el sensor.

Código prueba_ultrasonido

```
#define echoPin 11 // attach pin D2 Arduino to pin
Echo of JSN-SR04T

#define trigPin 12 //attach pin D3 Arduino to pin
Trig of JSN-SR04T

// defines variables

long duration; // variable for the duration of
sound wave travel

int distance; // variable for the distance
measurement

void setup() {

pinMode(trigPin, OUTPUT); // Sets the trigPin as
an OUTPUT

pinMode(echoPin, INPUT); // Sets the echoPin as an
INPUT

Serial.begin(115200); // // Serial Communication
is starting with 9600 of baud rate speed

Serial.println("Ultrasonic Sensor HC-SR04 Test");
// print some text in Serial Monitor

Serial.println("with Arduino UNO R3");

}

void loop() {

// Clears the trigPin condition

digitalWrite(trigPin, LOW); //

delayMicroseconds(2);

// Sets the trigPin HIGH (ACTIVE) for 10
```

	<pre> microseconds digitalWrite(trigPin, HIGH); delayMicroseconds(10); //digitalWrite(trigPin, LOW); // Reads the echoPin, returns the sound wave travel time in microseconds duration = pulseIn(echoPin, HIGH); // Calculating the distance distance = duration * 0.038 / 2; // Speed of sound wave divided by 2 (go and back) // Displays the distance on the Serial Monitor Serial.print("Distancia: "); Serial.print(distance); Serial.println(" cm"); // working code for aj-sr04m } </pre>
	<p>4. Una vez cargado el código, abrir el serial monitor para visualizar los datos tomados por el sensor.</p>
Resultado esperado	Sensor funcionando correctamente y valores de medición coherentes con la medida de la regla.
Resultado obtenido	El sensor presenta algunos problemas al momento de realizar la lectura, ya que tomaba la mayor parte de las distancias de prueba efectuadas sobre el objeto como ‘fuera de rango’. Se optó por cambiar el código, y adaptar el del fabricante.

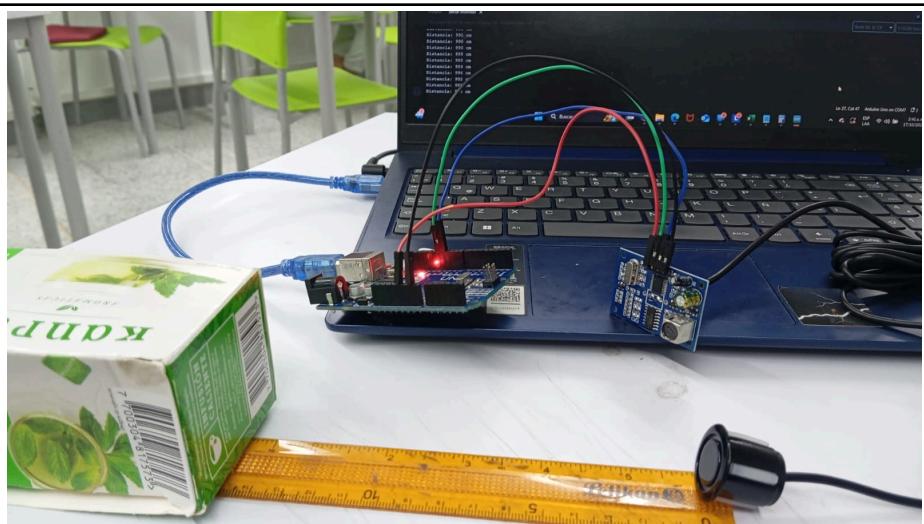


Figura 10. Evidencia del funcionamiento del sensor

Al asegurarse que estaba funcionando correctamente, se presentó otro reto: Los valores de las mediciones. Al inicio estas no concordaban con las de la regla, sin embargo, luego de hacer algunos cambios, comenzó a realizar un mejor sentido y mostrar mediciones, que aunque incorrectas, eran un poco más coherentes.

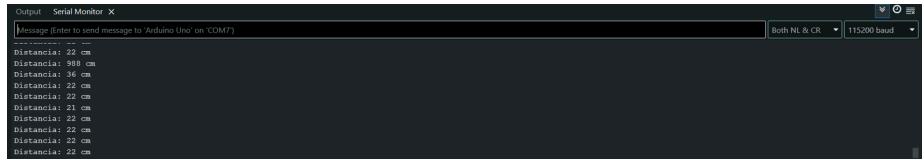


Figura 11. Evidencia de las distancias enviadas por el sensor desde el serial monitor.

Comentarios	Hay que tener en cuenta la calibración del sensor cuando se migre a otro microcontrolador, ya que las próximas pruebas integradas serán con el ESP32. Tener en cuenta las conexiones al micro y validar la correcta comunicación tanto con el microcontrolador como con el computador, ya que son factores que pueden entorpecer la práctica en el futuro.
-------------	--

Tabla 2. Microcontrolador ESP32

PRUEBA UNITARIA EN TIEMPO REAL	
Requerimientos	<ul style="list-style-type: none"> Configuración del ESP32 para verificar que esté funcionando correctamente en su ejecución.
Tipo de prueba	Unitaria
Hardware requerido	<ul style="list-style-type: none"> Microcontrolador ESP32 Computador Protoboard
Software requerido	<ul style="list-style-type: none"> Arduino IDE

Objetivo	<ul style="list-style-type: none"> • Validar la comunicación serial del microcontrolador.
Descripción	
Procedimiento	<p>1. Encender el computador y abrir el software requerido.</p> <p>2. Realizar las conexiones entre el ESP32 y PC.</p> <p>3. Abrir el IDE de arduino.</p> <p>4. Cargar el código en la placa.</p> <pre>void setup() { pinMode(2,OUTPUT); } void loop() { digitalWrite(2,HIGH); delay(1000); digitalWrite(2,LOW); delay(1000);</pre>
Resultado esperado	Que se pueda conectar el esp32 al PC y descargar las librerías para que el código corra perfectamente con arduino IDE.
Resultado obtenido	La prueba de la ESP32, tiene como objetivo validar la comunicación serial. Para ello se conecta la ESP32 al puerto serial del computador y se observa que hay un problema con los drivers, y por tal motivo es necesario actualizarlos.
Comentarios	Para trabajar con la ESP32 es necesario descargar las librerías que permitan la identificación de la placa, y posterior a ello, comenzar con las pruebas.

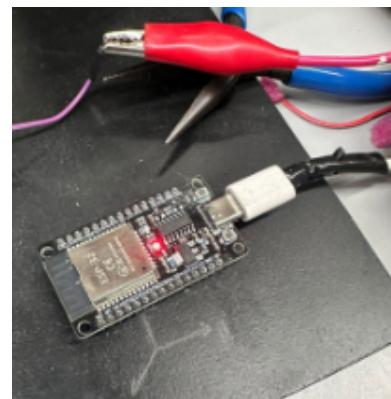
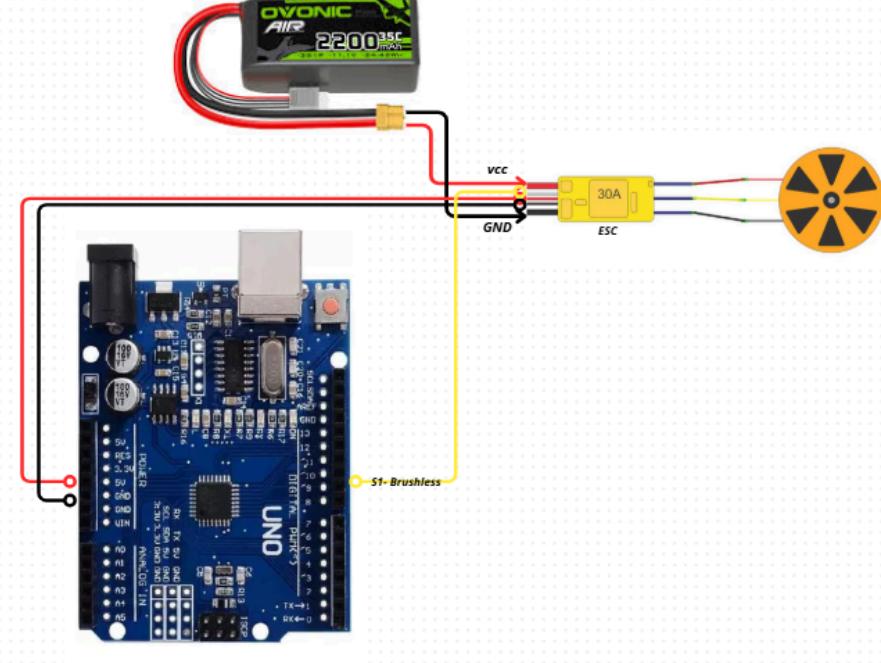
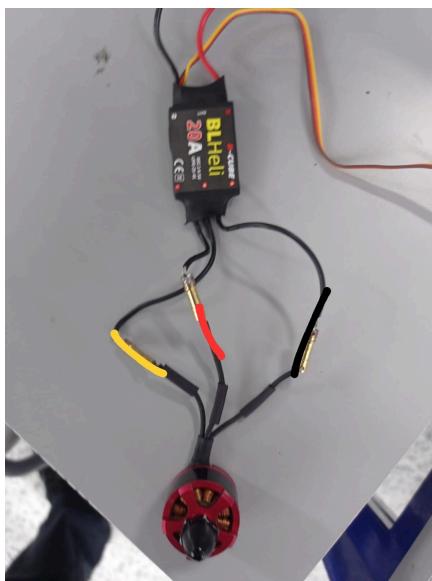


Figura 13. Conexiones del ESP32 para pruebas.

Tabla 3. Motor Brushless

PRUEBA UNITARIA EN TIEMPO REAL	
Requerimientos	
	<ul style="list-style-type: none"> Configuración del ESP32 para verificar que esté funcionando correctamente en su ejecución.
Tipo de prueba	Unitaria
Hardware requerido	<ul style="list-style-type: none"> Arduino UNO Computador ESC 20A Motor brushless RS2212
Software requerido	<ul style="list-style-type: none"> Arduino IDE
Objetivo	<ul style="list-style-type: none"> Validar el funcionamiento de los motores brushless.
Descripción	
Procedimiento	<ol style="list-style-type: none"> Encender el computador y abrir el software requerido. Realizar las conexiones entre el Arduino UNO y PC. 
	<p>Figura 14. Simulación de conexiones de un microcontrolador.</p> <ol style="list-style-type: none"> Abrir el IDE de arduino. Cargar el código en la placa.

	<pre>#include <Servo.h> Servo esc; const int PIN_ESC = 9; // Señal al ESC en D9 const int PULSE_MIN_US = 1000; // Mínimo típico para armar const int PULSE_MAX_US = 2000; // Máximo típico const int PULSE_RUN_US = 1200; // Pulso fijo para girar lento (ajústalo 1100-1600) void setup() { esc.attach(PIN_ESC, PULSE_MIN_US, PULSE_MAX_US); // Armar ESC: mantener mínimo unos segundos esc.writeMicroseconds(PULSE_MIN_US); delay(3000); // Señal de funcionamiento fija esc.writeMicroseconds(PULSE_RUN_US); } void loop() { // Nada: se mantiene el pulso fijo indicado arriba }</pre>
<i>Figura 15. Código simple para el movimiento del motor brushless.</i>	
Resultado esperado	Que el arduino UNO mande la señal al esc del motor brushless y lo mueva.
Resultado obtenido	La prueba salió exitosamente y se logró que el motor brushless se moviera dependiendo del pulso enviado por medio de la comunicación del arduino Uno.
	
<i>Figura 16. Movimiento del motor brushless.</i>	
Comentarios	Para realizar las pruebas es necesario asegurar no tener falsos contactos y tampoco cortos por cables sueltos ya que el motor brushless es bastante fuerte al momento de girar.

APÉNDICE A. PLANTILLAS DE RECOLECCIÓN DE DATOS

Estas plantillas no deben diligenciarse todavía. Se usarán cuando inicien los ensayos controlados.

Configuración	Tiempo (min)	Temp. agua (°C)	Ingreso de agua (mL)	Estabilidad	Observaciones
Modo manual	3 min	25 °C	5–8 mL	Estable	El barco iba rápido; recorría 3–4 m en 2–3 s. Correcciones manuales constantes.
Modo asistido	4 min	25 °C	2–4 mL	Muy estable	Mantuvo trayectoria más recta. Casi sin desviaciones. Funcionó mejor para evitar obstáculos.
Prueba general en lago	5 min	25 °C	0–3 mL	Estable	Peso aprox. 1.8–2 kg. Se hicieron pruebas en la madrugada y mañana. Sin problemas mayores.

Escenario	Modo	% sin colisión	Distancia mínima (cm)	Desviación media (cm)	Reingreso (s)	Latencia (s)	Notas
Círculo en lago	Manual	80%	25–30 cm	18–20 cm	1.8 s	0.25 s	El barco respondía bien pero se desviaba más. Dependía mucho del pulso manual.

Círculo en lago	Asistido	97%	20–25 cm	8–10 cm	0.9 s	0.15 s	Mucho más recto. Mejor para esquivar. Las correcciones automáticas ayudaron un montón.
-----------------	----------	-----	----------	---------	-------	--------	--

Tabla 4. Espacio reservado para trayectoria de ejemplo en pruebas preliminares.

Configuración	Tiempo (min)	Temp. agua (°C)	Ingreso de agua (mL)	Estabilidad	Observaciones

Tabla 5. Espacio reservado para comparación gráfica entre modo manual y asistido en escala de grises.

Escenario	Modo	% sin colisión	Distancia mínima (cm)	Desviación media (cm)	Reingreso (s)	Latencia (s)	Notas

4. RESULTADOS

4.1 Pruebas físicas

La fase de validación del prototipo a escala permitió observar el comportamiento real del sistema bajo condiciones controladas de operación, acordes con el alcance definido para pruebas en piscina y ambientes semicontrolados. Tal como se planteó desde el inicio, estos ensayos no buscan extrapolar de forma directa el desempeño a una embarcación real, sino consolidar una base experimental que permita identificar fortalezas y aspectos a mejorar del sistema de asistencia para embarcaciones de menor escala.

A lo largo de las pruebas se documentaron tanto el desempeño del módulo de navegación asistida como la respuesta ante obstáculos, variaciones de distancia y maniobras ejecutadas por el piloto. En términos generales, el prototipo mostró resultados positivos y coherentes con la lógica planteada: el sensor ultrasónico logró detectar obstáculos dentro de rangos esperados y la clasificación de zonas de seguridad se ejecutó de manera adecuada en la mayoría de los recorridos. Sin embargo, también se identificaron situaciones donde los resultados no fueron completamente concluyentes, principalmente debido a la naturaleza variable del entorno acuático y a la sensibilidad del sensado frente a lecturas momentáneamente ruidosas o incompletas, algo consistente con lo reportado en la literatura cuando se trabaja en ambientes reales con incertidumbre sensorial.

En varias repeticiones, la transición entre zonas (libre, precaución y peligro) funcionó de forma fluida, activando oportunamente la reducción de velocidad o el corte del motor. No obstante, se observaron casos puntuales en los que la

toma de decisiones lógicas presentó fallas: algunas veces fueron mínimas (por ejemplo, leves retrasos en la respuesta o correcciones pequeñas de rumbo), y en pocas ocasiones alcanzaron un nivel significativo, reflejado en decisiones tardías o en una maniobra de evasión menos efectiva de lo esperado. De manera similar, el sistema de dirección respondió bien en la mayoría de trayectorias tipo “L”, “S” y circuito cerrado, pero en ciertos instantes se evidenciaron desviaciones asociadas a la dinámica propia del timón y a la priorización de la lógica de seguridad frente al control manual del piloto.

4.2 Teleoperación

Con el fin de complementar las pruebas físicas en piscina y reducir riesgos durante la etapa de ajuste, se implementó una fase de teleoperación mediante Gazebo. Esta simulación permitió validar, en condiciones repetibles y controladas, la lógica base de navegación y el comportamiento dinámico esperado del prototipo antes de someterlo a escenarios reales. Este enfoque es coherente con el alcance del proyecto, que prioriza etapas progresivas de validación y comparación entre operación manual y asistida.

En el entorno simulado se cargó el modelo del barco a escala con sus componentes esenciales (casco, actuador de timón y propulsión) y se habilitó el control remoto desde la estación del operador. La teleoperación permitió ejecutar trayectorias equivalentes a las pruebas físicas (rutas tipo L, S y circuito cerrado), observando la respuesta del sistema frente a comandos de velocidad y dirección. En general, el control fue estable y permitió maniobras suaves, lo cual sirvió como referencia para calibrar rangos de actuación y sensibilidad del timón en el prototipo real.

Uno de los principales aportes de esta etapa fue la posibilidad de reproducir escenarios con obstáculos virtuales, evaluando cómo se comportaría el sistema ante variaciones de distancia sin depender de condiciones externas como oleaje, reflejos o ruido en el sensor. Gracias a esto, se pudieron detectar tempranamente situaciones donde la dirección requería ajustes finos o donde la lógica de asistencia debía suavizar transiciones para evitar correcciones bruscas. Estas observaciones se tomaron como guía para refinar parámetros antes de las pruebas en agua.

Finalmente, la teleoperación en Gazebo funcionó como un “banco de pruebas” seguro para iterar rápidamente sobre el control, confirmar la interpretación de comandos y verificar la coherencia general entre lo programado y lo observado. Aunque la simulación no reemplaza las pruebas reales, sí aportó trazabilidad técnica, disminuyó el número de iteraciones físicas necesarias y fortaleció la confiabilidad del sistema previo a su validación experimental completa.

x. CONCLUSIONES

Los avances obtenidos confirman la viabilidad técnica del prototipo a escala para asistencia a la navegación en entornos controlados. Las pruebas de impermeabilidad y flotación, con y sin la carga de los componentes, indican que el casco ofrece estanqueidad y estabilidad acordes con el uso previsto y que los ajustes realizados en tapa y pasadores mejoran la repetibilidad de las pruebas. La integración y las verificaciones unitarias muestran respuesta consistente de actuadores y funcionamiento correcto del corte de energía, lo que reduce riesgos para los siguientes ensayos. Con estos resultados preliminares se sostiene el enfoque del proyecto que prioriza funciones de asistencia medibles y comparables frente al control manual dentro del alcance establecido. A esta altura no se reportan métricas de desempeño en navegación y por lo tanto no se extraen conclusiones más allá de lo observado en estanqueidad, flotación e integración básica.

REFERENCIAS

- [1] J. S. MOSQUERA-ORTIZ, M. DEL M. GARCÍA-MATABANCHOY, Ó. J. PÉREZ-LADINO Y M. Á. GONZALEZ-BERMUDES, “ANTEPROYECTO ACTUALIZADO: PROTOTIPO DE EMBARCACIÓN A ESCALA CON SISTEMA DE NAVEGACIÓN ASISTIDA Y DETECCIÓN DE OBSTÁCULOS (DAGON),” UNIV. AUTÓNOMA DE OCCIDENTE, CALI, COLOMBIA, DOC. INTERNO, 2025.
- [2] J. S. MOSQUERA-ORTIZ, M. DEL M. GARCÍA-MATABANCHOY, Ó. J. PÉREZ-LADINO Y M. Á. GONZALEZ-BERMÚDEZ, “DAGON

PRELIMINARY PROJECT,” PÓSTER/PROYECTO PRELIMINAR, UNIV. AUTÓNOMA DE OCCIDENTE, 2025. (AUTORES SEGÚN [1].)

[3] INTERNATIONAL MARITIME ORGANIZATION (IMO), “AUTONOMOUS SHIPPING,” SITIO WEB, CONSULTADO: OCT. 17, 2025.

[4] H. LYU, Z. HAO, J. LI, G. LI, X. SUN, G. ZHANG, Y. YIN, Y. ZHAO Y L. ZHANG, “SHIP AUTONOMOUS COLLISION-AVOIDANCE STRATEGIES—A COMPREHENSIVE REVIEW,” J. MAR. SCI. ENG., VOL. 11, N.º 4, ART. 830, 2023, DOI: 10.3390/JMSE11040830.

[5] H. SUN, Q. XUE, M. PAN, Z. LIU Y H. LI, “HYBRID OBSTACLE AVOIDANCE ALGORITHM BASED ON IAPF AND MPC FOR UNDERACTUATED MULTI-USV FORMATION,” J. MAR. SCI. ENG., VOL. 13, N.º 8, ART. 1436, 2025, DOI: 10.3390/JMSE13081436.

[6] V. ARGUIMBAU GUARINOS, “DISEÑO E IMPLEMENTACIÓN DE ALGORITMOS DE EVITACIÓN DE OBSTÁCULOS PARA UN VEHÍCULO AUTÓNOMO SUBMARINO,” TRABAJO FIN DE MÁSTER, UNIVERSITAT POLITÈCNICA DE VALÈNCIA, 2017. (REPOSITORIO UPV).

[7] K. LIU, S. DING, L. YAN, J. SUN, S. WU, S. HAN, F. LI Y L. ZHOU, “COLREGs-COMPLIANT AUTONOMOUS COLLISION AVOIDANCE AND IMPROVED ADAPTIVE LOS-BASED MOTION CONTROL FOR USVs IN COMPLEX WATERS,” OCEAN ENG., VOL. 303, ART. 120127, 2024, DOI: 10.1016/J.OCEANENG.2024.120127

[8] W. SONG, Z. CHEN, M. SUN, Y. WANG Y Q. SUN, “A COLREGs-BASED PATH-PLANNING METHOD FOR COLLISION AVOIDANCE CONSIDERING PATH COST THROUGH REINFORCEMENT LEARNING,” OCEAN ENG., VOL. 315, ART. 120746, 2025, DOI: 10.1016/J.OCEANENG.2025.120746.

[9] C. LEE, J. PARK Y J. KIM, “OBSTACLE AVOIDANCE UNDER CONTACT CONSTRAINTS USING CONTROL BARRIER FUNCTIONS,” ARXIV:2504.19247, 2025.

[10] S. DOMÍNGUEZ-PÉRY, D. E. SÁNCHEZ Y A. GOMEZ PAZ, “REDUCING MARITIME ACCIDENTS IN SHIPS BY TACKLING HUMAN ERROR: A SYSTEMATIC LITERATURE REVIEW AND FUTURE RESEARCH AGENDA,” J. SHIPPING AND TRADE, VOL. 6, ART. 2, 2021.

[11] L. HU, H. HU, W. NAEEM Y Z. WANG, “A REVIEW OF AUTONOMOUS NAVIGATION FOR MARINE VESSELS,” J. AUTOM. INTELL., VOL. 4, N.º 2, PP. 5–13, 2022.

[12] M. DELNEUVILLE, “HUMAN ERROR IN MARINE ACCIDENTS: IS THE CREW NORMALLY TO BLAME?,” MARITIME TRANSPORT RESEARCH, VOL. 2, 2021, ART. 100006.