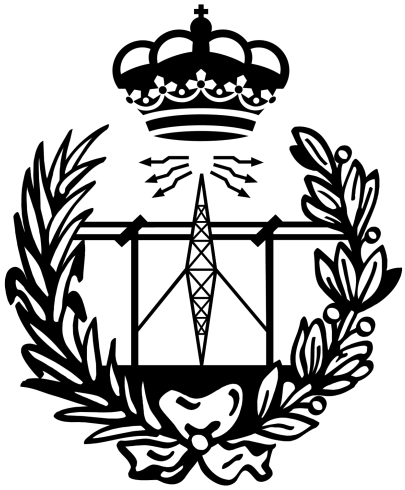
ULPGC Universidad de Las Palmas de Gran Canaria | eite ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA



TRABAJO FIN DE GRADO

INTEGRACIÓN DE SENSORES ACÚSTICOS EN UN ROBOT SUBMARINO

Titulación: Ingeniería en Tecnologías de la Telecomunicación

Mención: Sistemas de Telecomunicación

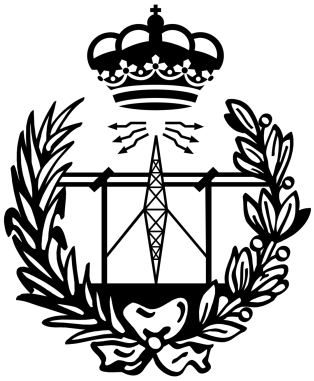
Autor: Adrian Baita Saavedra

Tutor: Eugenio Jiménez Yguácel

Fecha: Julio de 2025

ULPGC Universidad de Las Palmas de Gran Canaria | eite ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA

ESCUELA DE INGENIERÍA DE TELECOMUNICACIÓN Y ELECTRÓNICA



TRABAJO FIN DE GRADO

INTEGRACIÓN DE SENSORES ACÚSTICOS EN UN ROBOT SUBMARINO

HOJA DE EVALUACIÓN

Presidente:

Secretario:

Vocal:

Fecha: Julio de 2025

Agradecimientos

A mi mujer, Lilia, y a mi hijo, Luis:

Toda nueva aventura empieza con un primer impulso. El mío fue tu confianza, Lilia, que convirtió una idea pasajera en una decisión firme: la de volver a empezar una carrera. Hoy concluyo esta etapa con nuestro pequeño Luis en brazos, con la certeza de que todo esfuerzo valió la pena y con la ilusión de seguir caminando a tu lado. Los quiero con todo mi corazón.

A mis suegros, Cristina y Manolo, por su generosidad y presencia desde el principio. Este logro también es de ustedes.

A mis padres, Zahaida y Mauro, y a mis hermanos, Dafne y Eugenio, por asentar las bases de la persona que soy hoy.

A todos mis compañeros, por aceptarme, ayudarme, escucharme y sobre todo, conseguir que estos cuatro años hayan sido insuperables.

A mi abuelo, Pancho, que sé que me sonríe desde el cielo.

Reconocimientos

Este Trabajo de Fin de Grado se ha realizado en el contexto del proyecto NAUTILUS: Integración, Test y Validación De AUVs (PID2020-112502RB-C43) llevado a cabo por la División de Ingeniería de Comunicaciones (DIC) del Instituto para el Desarrollo Tecnológico y la Innovación en Comunicaciones (IDeTIC), al que se le reconoce su apoyo.

Índice general

Índice de figuras

Índice de tablas

Resumen

En este TFG se presenta la integración de los sensores acústicos Ping Sonar y Ping 360 y el diseño de simuladores que replican fielmente su respuesta, validados con clientes visuales (Ping Viewer) y programáticos (Ping-Python). Integrados en scripts de control mediante Pymavlink y ensayados en el entorno SITL, los simuladores posibilitaron pruebas conjuntas de percepción y maniobra. El Ping Sonar se integró con éxito en el control de profundidad, mientras que el Ping 360 lo hizo también con éxito pero revelando un problema de brechas en los desplazamientos por limitaciones de resolución angular. Los resultados confirman el potencial de los simuladores de sonar como herramienta de validación para sistemas de navegación subacuática autónoma y sientan las bases para optimizar la integración total del Ping 360 en futuros trabajos.

Abstract

This bachelor's thesis presents the integration of the Ping Sonar and Ping 360 acoustic sensors and the design of simulators that faithfully replicate their responses, validated with both visual (Ping Viewer) and programmatic (Ping-Python) clients. Integrated into control scripts via Pymavlink and tested in the SITL environment, the simulators enabled combined perception-and-maneuver trials. The Ping Sonar was successfully integrated into depth control, while the Ping 360 likewise performed well but exposed a gap issue during movements due to angular-resolution limits. The results confirm the potential of sonar simulators as a validation tool for autonomous underwater navigation systems and lay the groundwork for fully optimizing Ping 360 integration in future work.

1. Introducción

1.1. Antecedentes

En las últimas décadas, los vehículos operados de forma remota han adquirido creciente relevancia en distintos ámbitos gracias a su capacidad para ejecutar tareas complejas con gran eficacia [1]. Drones aéreos y robots terrestres han experimentado avances notables, impulsados por su facilidad de control inalámbrico y el acceso a tecnologías de localización como GPS (*Global Positioning System*). No obstante, estas prestaciones no se trasladan directamente al entorno submarino, donde las condiciones físicas y de comunicación imponen desafíos específicos. Por ejemplo, las señales de radiofrecuencia, esenciales para el control remoto en superficie, sufren una atenuación significativa en el medio acuático. Como consecuencia, los vehículos submarinos continúan dependiendo, en gran medida, de enlaces cableados o de soluciones ópticas y acústicas que aún se encuentran en proceso de optimización.

Diversas propuestas han intentado solventar el control inalámbrico de un vehículo submarino [2], incluyendo enlaces ópticos, acústicos (OFDM, PSK, FDK) y de radiofrecuencia (VLF). Sin embargo, tal y como señala [2], ninguna de estas alternativas —ni siquiera la combinación de varias, como se plantea en [3] y [4]— ha logrado resolver de manera eficaz el problema de la comunicación inalámbrica. En este contexto, la capacidad de autonomía del Vehículo Operado de Forma Remota (ROV, *Remotely Operated Vehicle*) durante la inmersión adquiere más importancia que nunca, dado que el vehículo debe tomar decisiones basadas en su entorno y cuanto mayor sea la precisión de dicho reconocimiento, más acertadas serán sus respuestas.

Ante la dificultad de mantener un control inalámbrico fiable en entornos submarinos, surgieron los Vehículos Autónomos Submarinos (AUV, *Autonomous Underwater Vehicles*) como alternativa tecnológica.

Una primera aproximación a la evolución de esta tecnología se puede apreciar en [5] y en [6]. Conviene destacar que las dos revisiones están temporalmente separadas más de nueve años, con todo lo que ello implica en términos de avance tecnológico. En [5] se enfatiza las limitaciones impuestas por la inaccesibilidad del GPS bajo el agua, la baja fiabilidad de las comunicaciones acústicas, y el alto coste de los sistemas tradicionales de navegación basados en balizas acústicas. Frente a estos métodos tradicionales surgieron alternativas innovadoras como las técnicas de localización y mapeo simultáneo (SLAM, *Simultaneous Localization and Mapping*) y la navegación cooperativa multi-agente. Buscando prescindir de aquellos métodos que requerían infraestructuras fijas o preinstaladas.

Por otro lado, [6] se centra en una revisión más integral de las tecnologías relacionadas con los vehículos autónomos submarinos adaptados a operaciones cercanas al fondo marino. No se limita solo a la navegación, sino que también aborda aspectos como la estructura física adaptada a entornos marinos, la agilidad de estas y tecnologías avanzadas de comunicación y posicionamiento acústico.

En conjunto las diferencias entre las dos revisiones responden a la necesidad emergente de desarrollar la autonomía, eficiencia y precisión de los AUV.

En el desarrollo de estas capacidades una de las opciones es la de modificar ROV añadiendo nuevo hardware o cambiando el existente y adaptando los módulos de software que fueran necesarios de tal manera que el sistema fuera capaz de realizar operaciones de forma autónoma.

La capacidad de los ROV para transformarse en AUV se hace evidente en [7], donde se demuestra que un ROV transformado en AUV es capaz de efectuar tareas complejas por una fracción del coste que supondría un AUV comercial.

En la línea de reducción de costes también se encuentra [8]. En él se expone el diseño y construcción de un AUV perfectamente funcional poniendo el foco en el apartado económico sin perder las capacidades que lo distinguen. El sistema completo costó menos de 300$.

Esto hace evidente que la integración de componentes en un sistema comercial previamente adquirido no solo puede resultar más económica, sino que además se puede llegar a conformar un sistema completo desde cero.

Una de las capacidades más importantes de estos sistemas es la de localizar al propio vehículo y a los objetos que lo rodean en el medio de exploración.

La publicación [9] realizada en 2015 explica cómo hace uso de sensores inerciales y acústicos para la resolución del problema de localización del vehículo sumergido. En concreto se utilizaron las medidas angulares de varias fuentes acústicas para corregir el dead-reckoning del sensor inercial.

Por otro lado, en [10] se centró en localizar al vehículo en el entorno que lo rodea mediante técnicas SLAM usando un sonar de escaneo mecánico, un sensor de velocidad Doppler (DVL, *Doppler Velocity Log*) y dos unidades inerciales de medida (IMU, *Inertial Measurement Unit*). Se realizó mediante formación de escaneo sintético (SSF, *Synthetic Scan Formation*) en la que las diferentes líneas de escaneo del sonar mecánico se incrustaban para formar un gráfico. Mediante técnicas de bucle se permitió la optimización de los escaneos formados.

Otro ejemplo concreto de cómo dotar a un ROV de autonomía es el expuesto en [11]. Se consigue mediante la integración de un ecosonda Ping Sonar de Blue Robotics. Implementando un sistema de control que mide continuamente la distancia al fondo marino, permitiendo que el ROV ajuste de forma precisa y automática la posición vertical del vehículo.

Una vez que hemos localizado al vehículo podremos proceder a detectar los objetos que le rodean y a localizarlos a partir de la posición de este.

En [12] y en [13] se trata la detección mediante sonar de paredes y cables submarinos respectivamente. El primero presenta un método de detección y localización de paredes en tiempo real y de forma automática. El sistema propuesto permite detectar automáticamente estructuras submarinas, permitiendo obtener

información en tiempo real sobre la posición de la pared respecto al vehículo, lo que facilita las tareas autónomas de inspección y exploración submarina.

El segundo propone un método basado en imágenes de sonar para detectar automáticamente un cable submarino durante la calibración in situ de sensores marinos. Para ello utiliza un sonar mecánico de barrido y técnicas de procesamiento de imagen, como filtrado secuencial, segmentación y comparación mediante plantillas. El método logra identificar correctamente el cable en más del 74% de los casos, también con imágenes acústicas con alta interferencia.

Si se pone el foco en el procesamiento de señales procedentes de los objetos detectados, en [14] se implementa una metodología de reconocimiento automático de objetivos (ATR, *Automatic Target Recognition*) en imágenes de sonar de visión frontal (FLS, *Forward Looking Sonar*) utilizando redes neuronales convolucionales (CNN, *Convolutional Neural Network)* en un AUV.

Los autores entrenaron dos modelos de inteligencia artificial avanzada (Mask R-CNN y SSD MobileNet V2) y evaluaron su precisión y velocidad de inferencia. Posteriormente validaron el sistema mediante pruebas de campo y confirmaron su viabilidad para la detección fiable de objetos sumergidos.

Por otro lado en [15] el objetivo era el de generar mapas binarios para la detección de obstáculos en imágenes sonar, utilizando filtros Gabor, kernels personalizados y gaussianos. En el procedimiento ajustaron los parámetros de los filtros para realzar la información de interés y minimizar ruido. Posteriormente compararon múltiples configuraciones evaluando los resultados frente a los datos de referencia. Finalmente, consiguieron más del 99% de coincidencia, evidenciando la utilidad de la binarización asistida por Gabor para detectar obstáculos.

Desde el punto de vista de la simulación en [16] se presenta una metodología para simular sonares submarinos de forma rápida haciendo uso de las capacidades de *shading* de las tarjetas gráficas modernas. En el artículo se propone integrar el simulador Gazebo y el *framework* ROCK para representar escenarios submarinos.

Se utiliza un shader personalizado para calcular la intensidad, la distancia y la distorsión angular de cada punto en la escena. Estos datos se utilizan posteriormente para la creación de imágenes de respuestas de sonar de distinto tipo permitiendo actualizar los datos de sonar en tiempo real, por debajo del tiempo requerido en simulaciones más tradicionales.

De la misma manera en [17] se presenta un método para generar imágenes de sonar que faciliten la detección de objetos submarinos utilizando vehículos autónomos (AUV). Se parte de una imagen ideal del objeto generada mediante un simulador de sonar que permite modelar la reflexión de ondas acústicas según la geometría y propiedades del blanco. Posteriormente se registra ruido de sonar real en una gran piscina vacía, empleando un sonar que capta únicamente el ruido del entorno. Luego se combinan la imagen ideal y el ruido para obtener simulaciones más parecidas a las reales. Con estas imágenes, se entrena una red neuronal basada en YOLOv3-tiny, capaz de detectar objetivos (por ejemplo, neumáticos) sin necesidad de grandes cantidades de datos de campo. En pruebas de campo, el sistema permitió distinguir correctamente los neumáticos colocados en el fondo marino de otros objetos. Esta solución permitió ahorrar tiempo y dinero al evitar tener que registrar una gran cantidad de datos reales para el entrenamiento de la red neuronal.

1.2. Objetivos

El objetivo general de este Trabajo Fin de Grado es dotar a un ROV de los componentes necesarios para reconocer su entorno y desplazarse de forma autónoma bajo el agua.

Para alcanzar este objetivo general, se han definido los siguientes objetivos operativos:

- O1. Estudio de soluciones de sensores y gobernabilidad del ROV: Consiste en el análisis del estado del arte de los sensores sonar y de los sistemas de control aplicables a un entorno submarino. Se pretende adquirir una base de

conocimiento que permita fundamentar la selección de componentes y estrategias de integración.

- O2. Seleccionar los dispositivos a integrar y el soporte lógico a utilizar: Este objetivo se centra en identificar y elegir los sensores acústicos —tanto de escaneo 360° como de distancia al fondo— y el sistema de control más adecuados, considerando su compatibilidad con el ROV BlueROV2 y las necesidades del proyecto. Asimismo, incluye la elección del software necesario de soporte para su funcionamiento.
- O3. Integrar los sensores e implementar sistemas de gobernabilidad en el ROV: Incluye tanto la conexión física de los dispositivos seleccionados como su configuración lógica, lo cual incluye el desarrollo de código para la adquisición de datos, el control del sistema y la sincronización de todos los elementos integrados en el robot.
- O4. Pruebas de funcionamiento: simulaciones y pruebas prácticas: Comprende la validación del sistema mediante simulaciones específicas para cada tipo de sensor y, posteriormente, la realización de pruebas reales en un entorno controlado. Estas pruebas permitirán evaluar la capacidad del ROV para interpretar su entorno y tomar decisiones de forma autónoma.

Cada uno de estos objetivos se articula a través de tareas planificadas que abarcan desde el análisis preliminar hasta la implementación técnica y validación experimental, asegurando así una aproximación metodológica completa y coherente con los fines del proyecto.

1.3. Estructura de la memoria

La memoria se organiza en once capítulos:

- El capítulo 1 presenta antecedentes, trabajos relacionados y objetivos.
- El capítulo 2 explica los fundamentos del sonar y los criterios de selección.
- El capítulo 3 compara diferentes sonares y justifica la elección final.

- En el capítulo 4 se describe el vehículo BlueROV2 y su arquitectura.
- El capítulo 5 detalla la integración de los sensores (hardware, software y uso de Ping Viewer).
- El capítulo 6 introduce los simuladores de ambos sonares y sus pruebas.
- El capítulo 7 lleva esos simuladores a la plataforma SITL del ROV y evalúa su rendimiento conjunto.
- El capítulo 8 cierra con las conclusiones y posibles líneas futuras.

Finalmente, se incluyen la Bibliografía, el Presupuesto y un Anexo con todos los scripts mencionados y documentación técnica de apoyo.

2. Sensores sonar

2.1. Introducción general al sonar

Como queda recogido en [18], sonar es un término derivado del inglés "*SOund NAvigation and Ranging* (SONAR)". Se trata de un dispositivo sónico de navegación y medición usado principalmente para localizar y detectar objetos sumergidos bajo el agua.

Su primera aparición se remonta a 1914 [19], cuando el canadiense Reginald Fessenden desarrolla el Oscilador de Fessenden. El primer dispositivo funcional, considerado hoy en día como el antecedente directo del sonar.

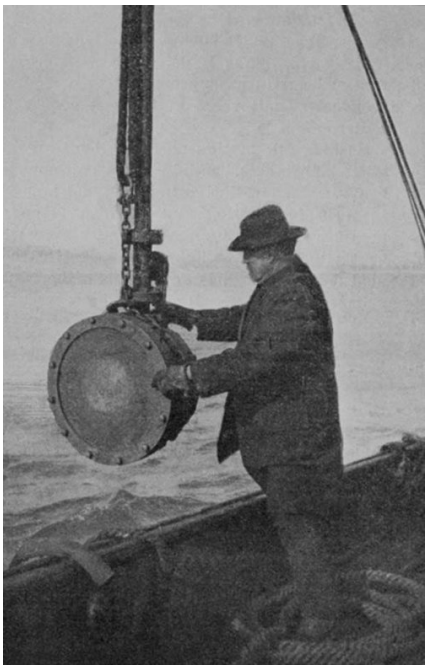


Figura 2.1: Reginald Fessenden y su oscilador eléctrico [20].

Se trataba de un sistema que incorporaba un diafragma metálico accionado electromagnéticamente para transmitir y recibir ondas de sonido en el agua. Se usó para comunicaciones submarinas y experimentos de detección de obstáculos en el mar (principalmente icebergs), suponiendo un paso pionero para la tecnología de detección submarina. Pero su rango y precisión para localizar objetos no eran tan

altos como el de las tecnologías sonar que se presentaron más tarde, la frecuencia de operación era más baja y el equipo era grande y pesado, como se muestra en la figura 2.1.

No fue hasta 1916 cuando el francés Paul Langevin y el ruso Constantin Chilowsky, registraron una patente [21] del principio del método utilizado y del aparato que se construiría como anticipo de la tecnología que después haría realidad el sonar.

Dos años después, y tras la salida de Chilowsky del proyecto como consecuencia de varios desacuerdos, Langevin presentó en solitario una nueva patente [22] dejando evidencia de su interés en usar los piezoeléctricos de cuarzo como receptores sónicos. Este hito supuso la primera realización de un sistema de detección de ecos ultrasónicos [23].

Desde entonces el desarrollo de tecnologías relacionadas con el sonar no ha cesado. Más allá de su evidente uso militar, los beneficios de esta tecnología han variado desde detección de bloques de hielo, batimetría, estudio de cetáceos, inspección de infraestructuras submarinas, navegación y localización de vehículos submarinos, detección de animales subacuáticos y muchos más.

2.2. Principios de funcionamiento de un sonar

Como hemos visto, el sonar basa su funcionamiento en la transmisión de señales acústicas y en la recepción de los reflejos de estas sobre blancos sumergidos. La diferencia de tiempos entre la señal transmitida y reflejada permite dar cuenta de la distancia a la que se encuentra el blanco. Y la dirección de apuntamiento determina su posición angular.

- Distancia al blanco*:* La distancia al blanco viene dada por la ecuación (2.1) y se define como el producto de la velocidad a la que viaja la onda acústica (c) por el tiempo que tarda en alcanzar al blanco. Este tiempo será igual a la mitad del tiempo que requiere la señal acústica para volver desde que fue transmitida.

$$D = c\left(\frac{t}{2}\right) \tag{2.1}$$

De la misma forma se deduce que las variaciones en la velocidad de propagación del sonido en el agua pueden afectar a la detección precisa de un blanco.

Según [24] la siguiente aproximación (2.2) es lo suficientemente precisa para ser válida en la mayoría de los casos. Siendo $T\ (ºC)$ la temperatura del agua, $S\ (ppt)$ su salinidad y $D\ (m)$ la profundidad a la que se está transmitiendo.

$$c = 1448.6 + 4.618T - 0.0523T^2 + 1.25(S - 35) + 0.017D \tag{2.2}$$

- Reflectividad del blanco: Los materiales que son iluminados por la señal acústica de un sonar responden reflejando la energía con distintos niveles de eficiencia.[25]. Cuanto más próxima al agua sea la densidad de un material, menor será su índice de reflectividad. Como consecuencia, los ecos de materiales como lodo, fango, arena o algas serán más débiles que los de materiales como roca, acero o madera.

  Esta propiedad de los materiales es la que posibilita discernir el blanco del agua que lo rodea. En base a esto se realiza una asignación de colores (figura 2.2).

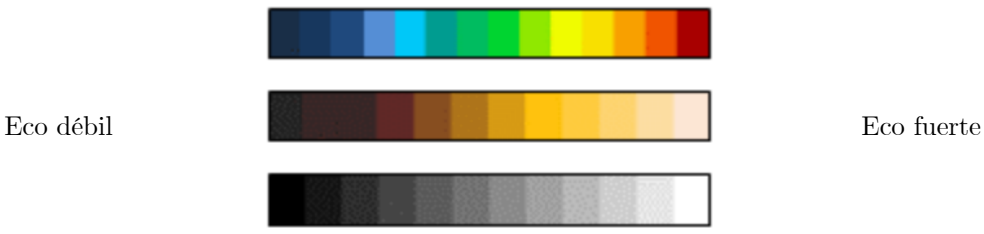


Figura 2.2. Paleta de colores típica de un display sonar [26].

- Patrón del haz del sonar*:* Los sonares de imagen de escaneo mecánico son sonares de un solo haz construido con forma de abanico (figura 2.3). Esto permite iluminar únicamente una parte muy concreta del entorno. Suele estar montado sobre un rotor, lo que le permite realizar barridos de escaneo.

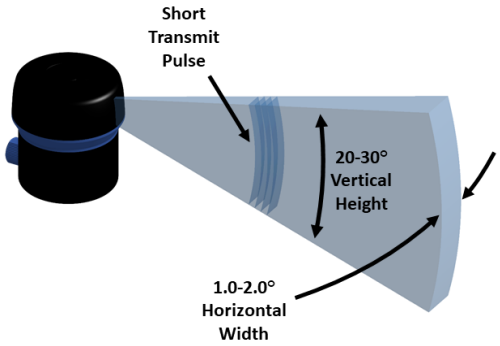


Figura 2.3: Haz en forma de abanico de los sonares de escaneo mecánico [27].

Como se muestra en la figura 2.2, el haz suele tener una apertura de entre 20^{o} y 30^{o} verticalmente y de 1^{o} a 2^{o} horizontalmente.

- Visibilidad del blanco: Aquellos blancos que se encuentren dentro de la zona iluminada por el haz serán susceptibles de ser detectados (figura 2.4).

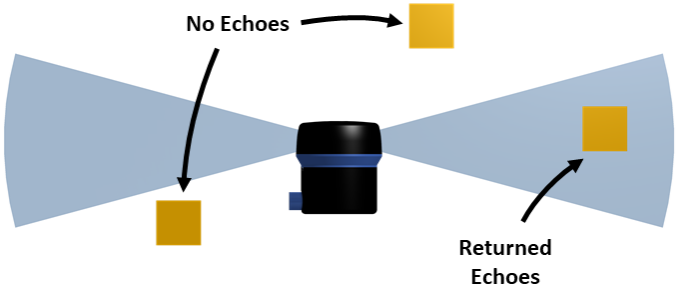


Figura 2.4: Cobertura vertical del scanning imaging sonar [28].

Además, estos sonares son incapaces de diferenciar dos objetos, dentro del haz de visibilidad, que se encuentren superpuestos en la columna de agua (con el mismo ángulo de llegada vertical), esto es conocido como "*slant range*" (figuras 2.5 y 2.6).

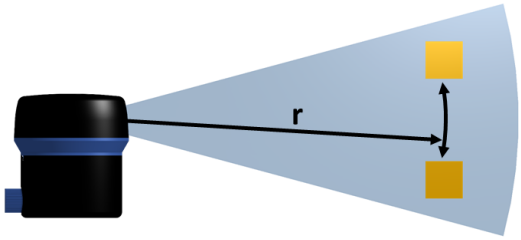


Figura 2.5: Sección vertical (Slant Range) [29].

Figura 2.6: Sección horizontal (Slant Range) [30].

- Efecto de sombra: Los blancos iluminados proyectarán una sombra (figura 2.7) como consecuencia del bloqueo de las señales acústicas. Como resultado podría haber blancos no detectados al no ser iluminados por el haz.

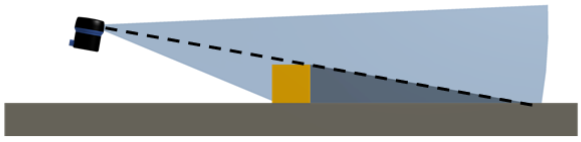


Figura 2.7: Efecto de sombra [31].

En la figura 2.8 se muestra la sombra proyectada por un neumático.

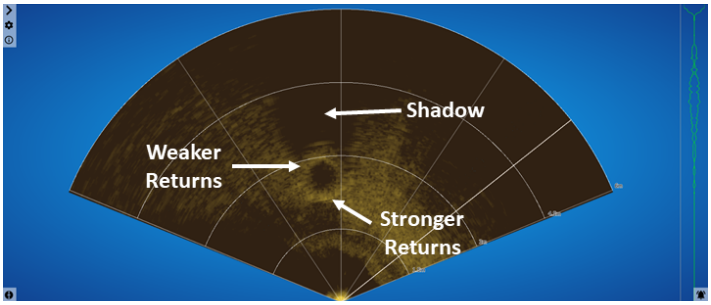


Figura 2.8: Ejemplo real del efecto sombra [32].

2.3. Clasificación de los sonares

Como se recoge en [5], los dispositivos sonar se pueden clasificar en dos categorías: tipo imagen y tipo medición de distancias.

Dentro de los dispositivos sonar de tipo imagen se destacan el sonar de barrido lateral, vista frontal, imagen de escaneo mecánico y apertura sintética. Entre los de tipo medición de distancias se destaca el sonar multi-haz. Los distintos tipos de sonares se presentan de forma resumida en las tablas 2.1 y 2.2.

Tabla 2.1: Dispositivos sonar tipo imagen.

Sonar	Descripción	Ventajas	Desventajas
Sonar de barrido lateral (figura 2.9.a)	Hace uso de múltiples haces, dirigidos perpendicularmente a la dirección de desplazamiento del vehículo, que permiten generar una imagen 2D del fondo	Se puede usar a velocidades relativamente altas (10 nudos) y permiten cubrir amplias superficies.	Resolución inversamente proporcional al alcance. Por ejemplo, 1,8 MHz proporciona un alcance de 40m.
Sonar de vista frontal (tipo imagen) (figura 2.9.c)	Como el sonar de barrido lateral pero la orientación es frontal.	Útil para cubrir el ángulo muerto bajo el vehículo.	Limitado por la relación distancia-profundidad (máximo 6:1); solo un ángulo de visión.
Sonar de apertura sintética (figura 2.9.e)	Fusiona ecos durante el desplazamiento para crear imágenes de alta resolución.	Resolución independiente del alcance.	Óptimo solo a velocidades bajas y en aguas profundas.

Sonar de imagen de escaneo mecánico (figura 2.9.d)	Un único haz montado sobre un rotor que permite escanear una franja del fondo.	Más económico que los sistemas multi-haz.	Lento. Su precisión depende de la actitud del vehículo (AUV).

Tabla 2.2: Dispositivos sonar tipo medición de distancias.

Sonar	Descripción	Ventajas	Desventajas
Ecosonda	Utiliza un haz estrecho para medir la distancia del transductor al fondo.	Es capaz de representar el fondo y la distancia a objetos que se encuentren entre el vehículo y el transductor	Medidas en una única dirección.
Perfilador	Es una variante del ecosonda capaz de penetrar en el fondo marino gracias al uso de la baja frecuencia.	Proporciona información sobre características del subsuelo.	La profundidad de penetración es inversamente proporcional a la resolución.
Multi-haz (figura 2.9.b)	Utiliza el tiempo de vuelo (ToF) de varios ecos para generar mapas batimétricos.	Es más eficiente en la captación de datos que el mono-haz.	Resolución inversamente proporcional a la frecuencia.

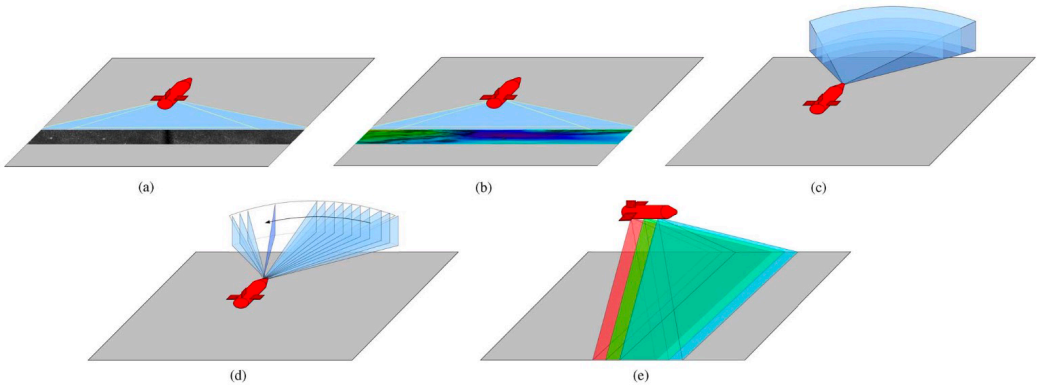


Figura 2.9: Franjas cubiertas por sensores sonar [33].

Parámetros característicos de un sonar

Los parámetros que permiten distinguir un dispositivo sonar frente a otro y facilitar la tarea de selección en función de las tareas que van a desempeñar son los que se explican a continuación:

- Tamaño y peso: El tamaño y el peso influyen directamente en la maniobrabilidad del vehículo, ya que un sensor voluminoso o demasiado pesado podría afectar negativamente a la propulsión y la estabilidad. Viene dado en gramos (g) tanto dentro como fuera del agua.
- Montura: La montura es otro punto fundamental ya que una montura incorrecta puede suponer modificaciones estructurales en el vehículo.
- Tipo: El tipo de sensor define la utilidad en la tarea prevista. Por ejemplo, un sonar de imagen de escaneo mecánico permite un campo de visión de 360^{o} útil para capturar información de todo el entorno. Por su parte, el sonar de vista frontal tiene un ángulo de visión fijo durante la exploración, útil en maniobras de evasión de obstáculos en tiempo real, ya que tiene una mayor frecuencia máxima de actualización.
- Rango (máximo y mínimo): Determinará el alcance y por ende las distancias máximas y mínimas de detección de blancos. Viene dado en metros (m).
- Resolución: En términos de capacidad para diferenciar blancos próximos entre sí. Viene dado normalmente en porcentaje del rango $(\%)$ o en milímetros (mm).
- Frecuencia de trabajo: Sonares típicos para AUV operan entre 100 kHz y 900 kHz. Frecuencias más altas ofrecen mejor resolución pero menor alcance; frecuencias bajas aumentan el alcance a costa de menor resolución.
- Conexión: Influye directamente en la facilidad de integración con la electrónica del AUV. Se destaca: USB, Ethernet (UDP), UART y RS-485.
- Protocolo de comunicación: Para controlar el dispositivo y obtener la información, los fabricantes proporcionan varias soluciones. Unos publican el protocolo que usa el sonar y lo acompañan librerías/SDK en varios lenguajes

(C++ o Python, entre otros) que implementan dicho protocolo y/o ofrecen una API sencilla al desarrollador. Otros, en cambio, no documentan el protocolo y entregan únicamente una API propietaria; el programador llama a sus métodos sin preocuparse de cómo se forman las tramas.

- Parámetros de exploración específicos: Existen ciertos dispositivos sonar que poseen parámetros clave específicos directamente relacionados con la capacidad de exploración de estos. Como por ejemplo los sonares de imagen de escaneo mecánico en los que es necesario conocer el rango de exploración, la resolución angular y la velocidad. O los sonares de vista frontal que poseen ángulo de visión y frecuencia máxima de actualización.

2.4. Criterios de selección

La selección de un dispositivo sonar depende en gran medida de la misión y de las características del vehículo que lo transporte. En reglas generales nos guiaremos del objetivo de la misión para realizar una priorización de parámetros clave. Una sugerencia de priorización para una misión cuyo objetivo es de evasión de obstáculos podría ser:

1. Tipo (FLS con alta frecuencia de actualización).
2. Rango y resolución (corto y medio alcance).
3. Capacidad de integración (física y electrónica).
4. Características físicas (tamaño y peso adecuado para la flotabilidad y maniobrabilidad del vehículo).
5. Protocolo de comunicación.
6. Fiabilidad y soporte técnico.
7. Presupuesto.

En la práctica, aunque el presupuesto se deje para el último lugar, se parte del presupuesto real y, con ese límite, se busca la opción que ofrezca el mejor resultado

posible. Si el coste se dispara, se revisan prioridades y se cederá en lo menos crítico hasta que los números encajen.

3. Selección de los sensores sonar

Se ha optado por incorporar dos dispositivos sonar para el sistema de navegación autónomo. Un dispositivo sonar tipo imagen que permita obtener información de lo que rodea al vehículo y un dispositivo sonar de tipo medición de distancias para controlar la cercanía al fondo submarino.

3.1. Análisis de dispositivos tipo imagen

Para esta finalidad la configuración propuesta es un dispositivo sonar de imagen de escaneo mecánico en la parte superior del vehículo. Lo que permite obtener información de lo que rodea al vehículo en su mismo plano horizontal en un rango de 360^{o} (figura 3.1).

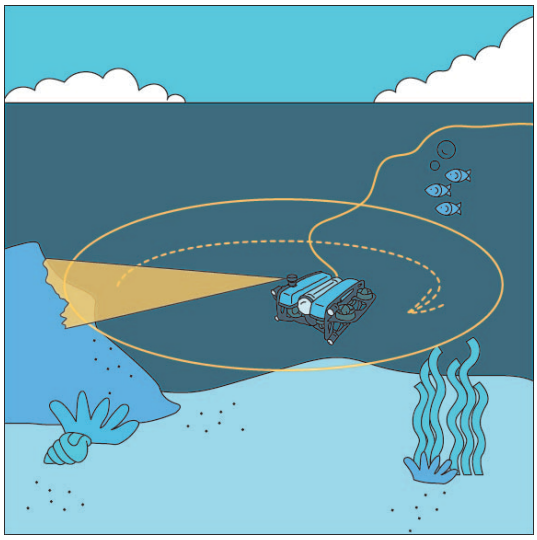


Figura 3.1: Forma de haz del sonar de escaneo mecánico [34].

Se ha realizado un análisis de distintos dispositivos sonar de diversos fabricantes y se han identificado seis posibles candidatos (tabla 3.1), exponiendo sus características más relevantes.

Tabla 3.1: Comparativa de dispositivo sonar tipo imagen.

	Seaking Hammer head	Micron Sonar MK 3	Ping360	ISS360	881L-GS	MRS900 S
Frecuencia de operación	675 kHz (LF) / 935 kHz (HF)	CHIRP ≈ 700 kHz	750 kHz	CHIRP 700 kHz (600–900 kHz banda)	310 kHz, 675 kHz o 1 MHz (programable)	900 kHz (CHIRP o CW)
Ancho/ apertura de haz	30° V × 0,9° H (LF)	35° V × 3° H	25° V × 2 ° H	23° V × 2,2° H	4° × 40° (310 kHz)	2° H × 25 ° V
					1,8° × 20° (675 kHz)	
	20° V × 0,6° H (HF)				0,9° × 10° (1 MHz)	
Resolución de rango	7,5 mm	≈ 7,5 mm	0,08 % del rango	2,5 mm (mínimo)	2 mm (1–4 m) / 10 mm (≥ 5 m)	≤ 7,5 mm
Rango de alcance	0,4 m (mínimo) 100 m (LF) / 40 m (HF)	0,3 m – 75 m	0,75 m – 50 m	0,15 m – 90 m	0,15 m (mínimo)	0,15 m – 60 m
Profundidad operativa máxima	700 m / 4000 m	750 m (estándar) / 3000 m (opcional)	300 m	4000 m / 6000 m	1000 m (estándar) / 3000 m (opcional)	2000 m
Interfaz/ protocolos de comunicación	RS-232, RS-485, ARCNET	RS-485 (2 hilos) y RS-232	USB, Ethernet (UDP), RS-485	RS-232, RS-485 y Ethernet	Ethernet 10 BASE-T (TCP/IP)	RS-232 / RS-485
Alimentación eléctrica	20 – 72 V DC 55 W	12 – 48 V DC 4 VA	11 – 25 V DC 5 W	12 – 65 V DC 95 mA en reposo / 240 mA en escaneo a 24 V	20 – 32 V DC < 7 W	12 – 60 V DC 4 W (máximo)
Peso	Aire 6,8 kg Agua 3,8 kg	Aire 0,32 kg Agua 0,18 kg	Aire 0,51 kg Agua 0,18 kg	Aire 0,38 k Agua 0,30 kg	Aire 1,8 kg (1000 m)	Aire 0,58 kg Agua 0,35 kg
Precio	8.000€	10.000€	2.750€	-	-	6.500€

3.2. Análisis de dispositivos tipo medición de distancias

Para la finalidad prevista para este dispositivo se plantea una configuración de ecosonda en la parte inferior del vehículo apuntando al fondo (figura 3.2).

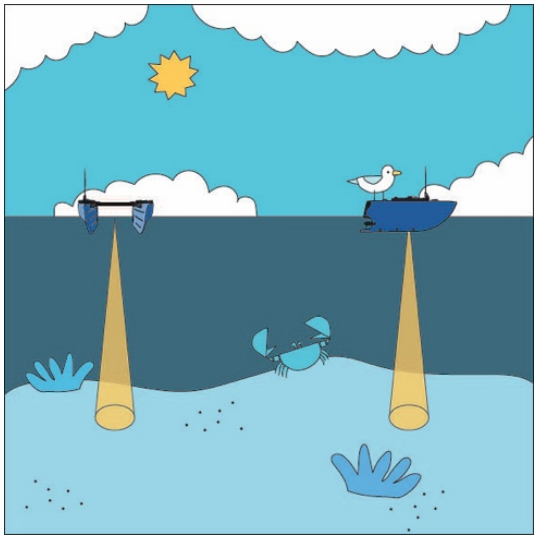


Figura 3.2: Forma de haz de una ecosonda mono-haz [35].

Se ha efectuado una búsqueda de dispositivos sonar en diversos fabricantes y se han encontrado seis posibles candidatos a dispositivo sonar tipo medición de distancias y se han representado sus características más relevantes (tabla 3.2).

Tabla 3.2: Comparativa de candidatos sonar tipo medición de distancias.

	852 Echo Sounder	S500	EU400	ISA500	Micron Echo-S	Ping Sonar
Frecuencia de operación	675 kHz	500 kHz	450 kHz	500 kHz (estándar) (400-600 kHz seleccionable)	500 kHz	115 kHz
Ancho / apertura de haz	10 ° cónico	≈ 5 ° cónico	5 ° cónico	6 ° cónico	6 ° cónico	25 ° cónico
Rango de alcance	0,50 m –100 m	0,30 m – 100 m	0,15 m – 100 m	0,10 m – 120 m	0,30 m – 50 m	0,30 m – 100 m
Resolución de rango	20 mm	3 – 24 mm	< 1 mm	1 mm	1 mm	0,5 % del rango

Profundidad operativa máxima	1000 m	300 m	5 m (limitada por cable)	1000 m	750 m	300 m
Interfaz / protocolos de comunicación	RS-485 a 115 kbps (RS-232 opcional)	Ethernet, USB, TTL-UART	USB	RS-232 y RS-485	RS-485 y RS-232	TTL-UART (Ping-Protocol)
Alimentación eléctrica	9 – 50 V DC $<$ 1,5 W	10 – 30 V DC 2,5 W en reposo/ 5 W máximo en activo	USB 5 V, 2 W máximo	9 – 36 V DC $\approx$ 1,3 W	12 – 48 V DC 1, 7 W	4,5 – 5,5 V DC $\approx$ 0,5 W
Peso	Aire 0.19 kg Agua 0.10 kg	Aire 0.094 kg Agua 0.050 kg	Aire 0.42 kg	Aire 0.30 kg Agua 0.11 kg	Aire 0.20 kg Agua 0.06 kg	Aire 0.187 kg Agua 0.100 kg
Precio	5.800€	850€	-	-	1.995€ (segunda mano)	486€

3.3. Conclusiones del análisis de dispositivos

Partiendo de la información extraída de las hojas de características de los distintos candidatos y expuesta en las tablas 3 y 4 se ha procedido a seleccionar los dispositivos que serán integrados en el ROV.

El dispositivo sonar tipo imagen elegido es Blue Robotics Ping 360 y el dispositivo sonar tipo medición de distancias elegido es Blue Robotics Ping Sonar.

Ambas son las opciones que prometen los mejores resultados para el presupuesto del que se dispone. La comunidad es la más activa de todas en foros y plataformas online. Además, el ROV sobre el que se procederá a integrar estos dispositivos es también de la empresa Blue Robotics, lo que facilitará su integración gracias a la existencia de documentación específica para esta tarea.

4. Descripción del vehículo BlueROV2

El robot submarino sobre el que se pretende realizar la integración de los sonares es el BlueROV2 de la empresa Blue Robotics, uno de los ROV comerciales más utilizados a nivel mundial.

Se trata de un ROV de código abierto y diseño modular, ampliamente utilizado en aplicaciones de inspección, investigación marina y desarrollo experimental. Su configuración de seis propulsores (4 horizontales y 2 verticales) le permite moverse con estabilidad y precisión en seis grados de libertad.

En la figura 4.1 se muestran los movimientos correspondientes a dichos grados de libertad.

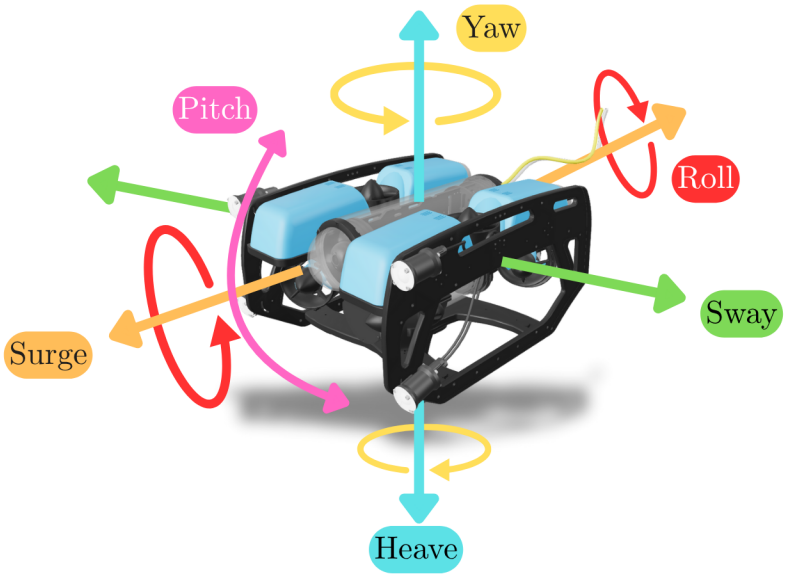


Figura 4.1: Movimientos del BlueROV2 en los seis grados de libertad.

Esta capacidad de control sobre los seis grados de libertad convierte al BlueROV2 en una plataforma ideal para el desarrollo de algoritmos de navegación, maniobra precisa y automatización de tareas subacuáticas.

En cuanto a la capacidad de expansión, el chasis abierto y los raíles laterales facilitan la instalación de *grippers*, sonares, sensores DVL (*Doppler Velocity Log*), plataforma de carga (*skid*) o módulos de batería extra.

4.1. Arquitectura física del sistema

El diagrama de bloques completo de un BlueROV2 y su sistema de control adjunto es el que se muestra en la figura 4.2. En esta figura se distinguen tres grandes bloques:

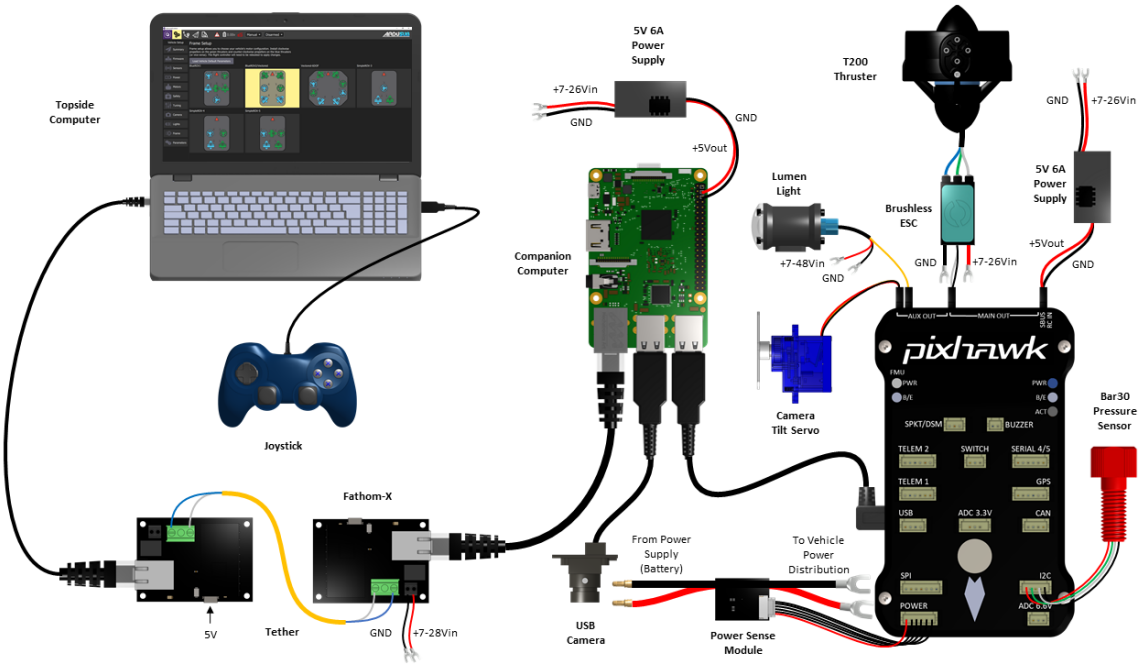


Figura 4.2: Sistema completo BlueROV2 y control de superficie.

- Autopilot: La placa del *autopilot* (*PixHawk*) procesa la señal de entrada de pilotaje y los datos de los sensores y controla los motores, luces, servos y relés del vehículo. Tiene un microcontrolador propio y ejecuta el firmware ArduSub del proyecto *Ardupilot*. Es parte del BlueROV2.
- Companion computer: El *Companion computer* transmite vídeo al Ordenador de superficie y retransmite las comunicaciones MAVLink entre el *Autopilot* y el Ordenador de superficie a través de Ethernet (el cable umbilical amarillo). Es una placa SBC (*Single Board Computer*) tipo Raspberry Pi y ejecuta una

versión personalizada de Linux. En este momento hay dos variantes de sistema operativo: el *Companion software* clásico y el nuevo BlueOS. El Companion computer es parte del BlueROV2.

- Topside computer: El ordenador de superficie es el equipo donde se reciben y visualizan el vídeo en directo y la información de telemetría. Acepta la entrada del operador desde un joystick para permitir el control del vehículo conectado. La información la recibe y transmite a través del umbilical y no forma parte del BlueROV2.

4.1.1. Autopilot

Como se indicó en la sección 4.1 el *Autopilot* (también conocido como controlador de vuelo) es el componente en el que se carga el firmware ArduSub. Aunque hay bastantes modelos diferentes disponibles estas son las características generales:

- Capacidad para cargar cualquier archivo de firmware binario ArduPilot (Copter, Plane, Rover, Boat, Sub).
- Contiene conexiones de entrada y salida para conectar múltiples periféricos. Incorpora IMU(s), brújula(s) magnética(s) y giroscopio(s) para determinar la orientación del vehículo.
- Capacidad de guardar registros del vehículo.

En el BlueROV2 del que se dispone el *Autopilot* es el modelo PixHawk1 originalmente desarrollado por la empresa 3DR. Las características principales de esta placa son:

Procesador:

- 32bit STM32F427 Cortex-M4F core con FPU
- 168 MHz
- 256 KB RAM
- 2 MB Flash
- 32 bit STM32F103 *failsafe* co-procesador

Sensores:

- Giroscopio 16 bits ST Micro L3GD20H
- Acelerómetro/magnetómetro 14 bits ST Micro LSM303D
- Acelerómetro/giroscopio de 3 ejes Invensense MPU 6000
- Barómetro MEAS MS5611

4.1.2.Companion computer

Los *Companion computer* suelen ser pequeños ordenadores monoplaca (SBC) que pueden conectarse a una tarjeta *Autopilot* y comunicarse mediante el protocolo MAVLink. El *Companion computer* tomará la telemetría del *Autopilot* usando el protocolo MAVLink y puede enrutar o procesar los datos de telemetría.

El *Companion computer* tiene dos funciones principales dentro del sistema de control ArduSub:

- Transmisión de vídeo de alta definición al ordenador de superficie.
- Transmitir comunicaciones entre el *Autopilot* y el ordenador de superficie a través de Ethernet usando el umbilical.

En el proyecto, el Companion computer de que se dispone es una placa modelo Raspberry Pi 3 que puede usar el software *Ardusub Companion* o el BlueOS.

4.1.3.Topside computer

Típicamente el ordenador de superficie ejecuta la aplicación *QGroundControl* (una potente e intuitiva estación de control terrestre *open source* para vehículos que soporten MAVLink) y es el dispositivo al que se conecta el *Companion computer* del BlueROV2. El ordenador de superficie es el que recibe la señal de vídeo en directo y la información de telemetría. Debe conectarse un joystick para el control manual.

Toda la comunicación entre el ROV2 y el ordenador de superficie se hace a través del umbilical usando Ethernet (IEEE1901).

Con el objetivo de dotar de autonomía al ROV, el subsistema del ordenador de superficie sí podría formar parte del BlueROV2 e iría sumergido con él. En estos casos no ejecutaría el *QGroundControl* sino que guiaría el BlueROV2 a base de comandos usando el protocolo MAVLink.

4.2. Arquitectura lógica del sistema

En paralelo con la arquitectura física del sistema hay una arquitectura lógica que se apoya en ella. Esta arquitectura nos muestra qué aplicaciones corren y cómo se puede acceder a la información en cada subsistema tal y como muestra la figura 4.3.

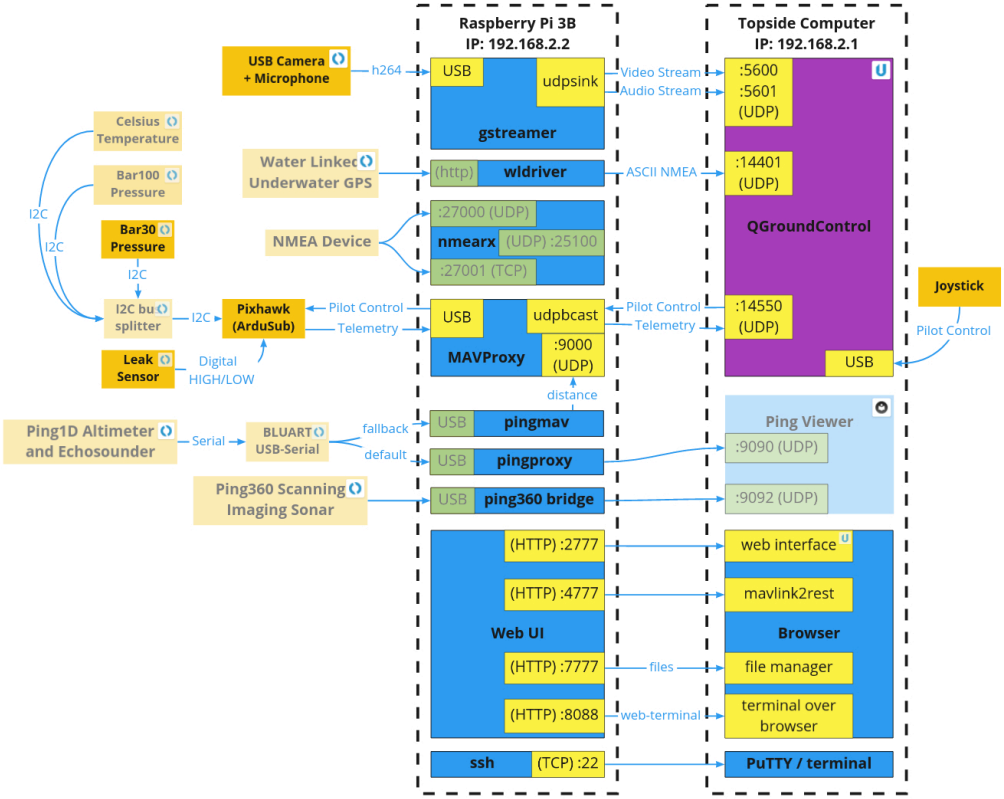


Figura 4.3: Arquitectura lógica del sistema.

En esta figura se pueden ver los tres bloques físicos principales de los que se habló antes y ahora pasaremos a explicar el software básico que se ejecuta en cada uno de ellos.

4.2.1. Autopilot

La placa de *Autopilot* funciona con el software ArduSub. El proyecto ArduSub es una solución de código abierto completa para vehículos submarinos teledirigidos (ROV) y vehículos submarinos autónomos (AUV). ArduSub forma parte del proyecto ArduPilot, derivado originalmente del código de ArduCopter. ArduSub dispone de numerosas funciones, como control de estabilidad por realimentación, mantenimiento de la profundidad y el rumbo, y navegación autónoma

El *Autopilot* se comunica con el *Companion computer* usando el protocolo MAVLink.

4.2.2. Companion computer

La placa del *Companion computer* es una Raspberry Pi 3. El software que corre es una versión modificada de Raspbian, una distribución tipo Debian orientada específicamente para las placas Raspberry Pi. Las principales funciones de este software son:

- Hace de interfaz en las comunicaciones entre el ordenador de superficie y el *Autopilot.*
- Transmite el vídeo a través del umbilical.
- Permite la interconexión de periféricos adicionales (sensores y sondas) con controladores compatibles.

De estas tres misiones la que más nos interesa es la primera. El ordenador de superficie no se comunica directamente con el *Autopilot.* En vez de eso el *Companion computer* usa una aplicación que replica la interfaz del *Autopilot.* Esta aplicación es el MAVProxy y lo que hace es permitir que el ordenador de a bordo envíe comandos MAVLink al *Autopilot* usando un puerto UDP del *Companion computer.*

4.2.3.Topside computer

En los sistemas de navegación guiados el ordenador de superficie ejecuta, habitualmente, el programa *QGroundControl*. Este programa es el encargado de comunicarse con el *Autopilot* a través del MAVProxy y de gestionar los diferentes periféricos y sensores adicionales del BlueROV2 que aparecen como distintos puertos UDP o TCP en el *Companion computer.*

5. Integración de los sensores sonar

5.1. Integración del sensor Blue Robotics Ping Sonar

En los siguientes apartados se describen la integración física (*hardware*) y lógica (*software*) del sensor Ping Sonar.

5.1.1.Conexión física

Las dimensiones del dispositivo, el montaje y el conexionado del sensor Ping Sonar se realizan como se detalla a continuación:

- Montaje en la estructura del robot: las medidas del sonar son las que se presentan en la figura 5.1, provista por el fabricante, el cual se monta bajo la estructura del vehículo haciendo uso de una montura que permite que el punto de medida sea rasante con el plano de apoyo del ROV.
- Conexión eléctrica y de interfaz de comunicación: el conexionado se hace mediante un puerto serie RS-232 con niveles TTL que se conecta al SBC mediante un conversor RS-232 a USB.

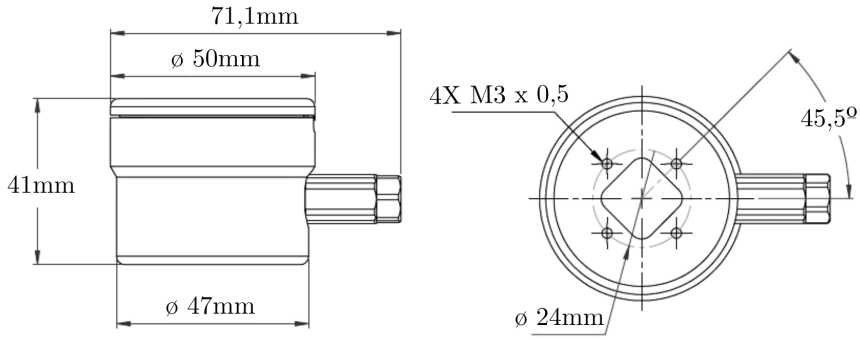


Figura 5.1: Dibujo técnico del dispositivo Ping Sonar [36].

5.1.2.Integración software

La comunicación entre la SBC y el sensor sonar se realiza a través de una dirección IP y un puerto UDP y mediante el protocolo Ping Protocol [37]. Se trata de un protocolo de comunicación binario sencillo desarrollado por Blue Robotics para el intercambio de información entre su familia de dispositivos sonar y el *Companion computer* de los vehículos sumergibles. Está pensado para un esquema maestro – esclavo, de tal manera que el *Companion computer* pide y el sonar responde.

La estructura de mensajes es la que se muestra en la tabla 5.1.

Tabla 5.1: Estructura de mensajes Ping Protocol.

Byte	Tipo	Nombre	Descripción
0	u8	*start1*	Identificador de inicio de trama, ASCII 'B'
1	u8	*start2*	Identificador de inicio de trama, ASCII 'R'
2-3	u16	*payload_length*	Número de bytes que hay en el *payload*.
4-5	u16	*message_id*	Identificador del mensaje.
6	u8	*src_device_id*	Identificador del dispositivo que envía el mensaje.
7	u8	*dst_device_id*	Identificador del dispositivo destinatario del mensaje.
8-n	u8[]	*payload*	Carga útil del mensaje.
(n+1)-(n+2)	u16	*checksum*	Suma de verificación del mensaje. Se calcula como la suma de todos los bytes del mensaje, excepto los del propio *checksum*.

Las familias de mensajes son las que siguen:

- *General*: Señalización y comunicación de propósito general.
- *Set*: Para escribir datos en el dispositivo sonar.

- *Get*: Para leer datos del dispositivo sonar.
- *Control*: Para interacciones más complejas que leer y escribir.

Los mensajes definidos para el dispositivo Ping Sonar son los listados en [38].

El más interesante en términos operativos es el mensaje *1211 distance_simple* y los campos que contiene se presentan en la tabla 5.2.

Tabla 5.2: Contenido del mensaje 1211 distance_simple.

Tipo	Nombre	Descripción	Unidades
u32	*distance*	Distancia devuelta por el sonar calculada a partir de la medida acústica más reciente.	mm
u16	*confidence*	Confianza en la medida más reciente.	%

Una vez el dispositivo está listo para su uso, el primer paso es el llamado proceso de negociación. Durante dicho proceso, el C*ompanion computer* pregunta al dispositivo sonar la versión de protocolo. Tras recibir la respuesta, el SBC ajusta la versión de protocolo y solicita la información relativa al tipo de dispositivo y a la versión de *firmware*. De esta manera el *Companion computer* ya conoce el conjunto de mensajes para comunicarse con el dispositivo sonar.

A partir de aquí las comunicaciones se realizan mediante los mensajes específicos del dispositivo.

El dispositivo sonar funciona como esclavo en una configuración maestro-esclavo síncrona. De modo que el dispositivo solo envía datos cuando lo solicite el maestro (SBC).

Este opera en bucle, transmitiendo señales y muestreándolas de forma periódica según el intervalo de ping configurado, siguiendo el ciclo mostrado en el diagrama de flujo de [39] representado en la figura 5.2 y enviará datos cuando exista una

solicitud de mensaje *profile_ data* o cuando el modo de salida continua de datos esté activado.

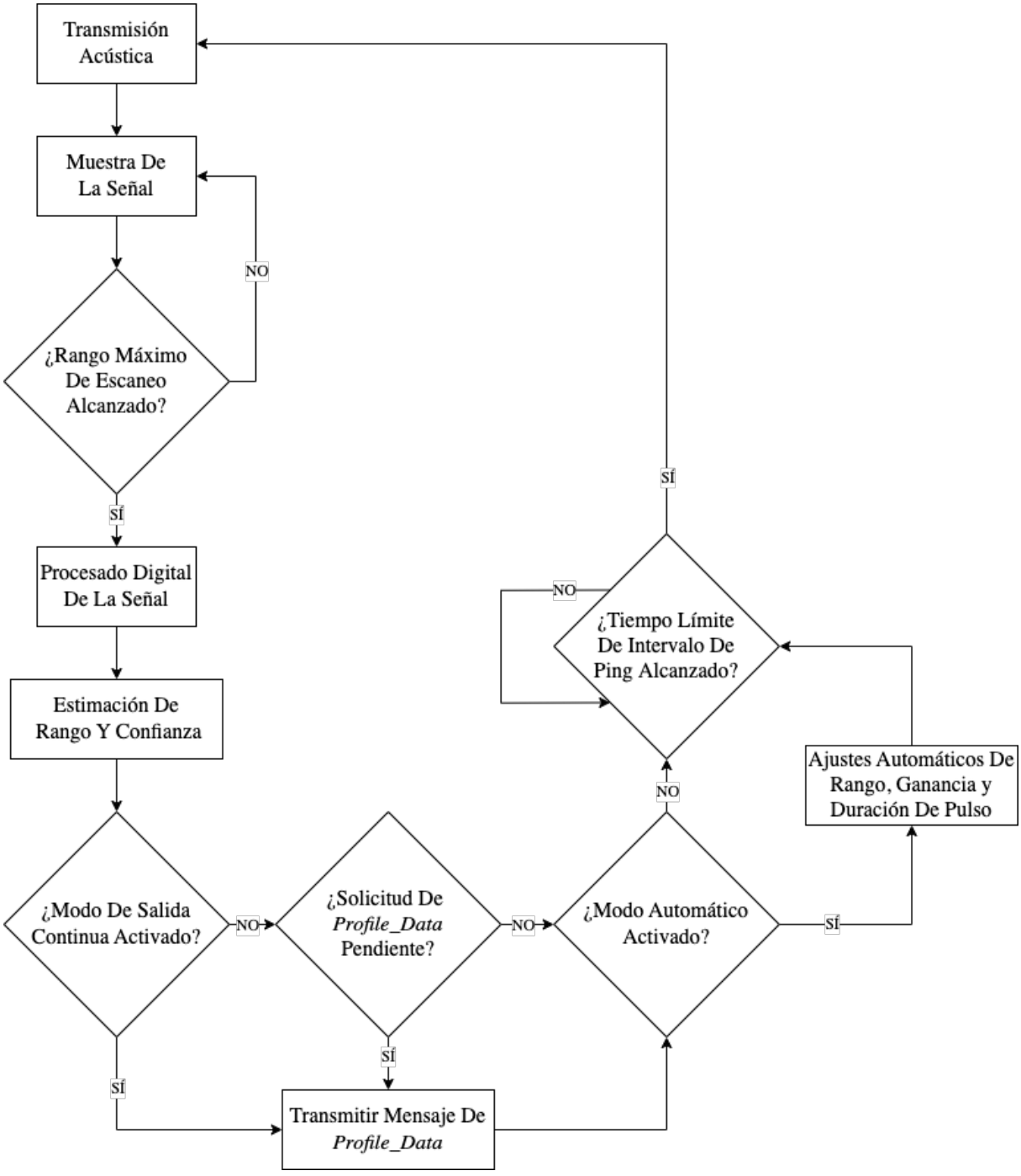


Figura 5.2: Diagrama de funcionamiento del Ping Sonar [40].

Una vez realizado el conexionado se procede a instalar Python 3 [41] y la librería Ping-Python [42].

Dicha librería nos permite crear un objeto *Ping1D* con una serie de métodos asociados a los mensajes del protocolo propios de este tipo de sonar, permitiéndonos extraer información de una manera muy cómoda.

La clase *Ping1D* hereda de la clase *PingDevice* que implementa métodos para la conexión por UDP, necesaria para la comunicación.

A la hora de operar con el dispositivo Ping Sonar, lo primero es inicializarlo. Empezamos declarando el objeto correspondiente al Ping Sonar:

```
p = Ping1D()
```

A continuación usamos el método *conect_ udp* que nos permite conectarnos al dispositivo vía IP en un puerto UDP definido:

```
p.connect_udp("0.0.0.0", int(6675))
```

Y seguidamente verificamos que todo ha ido bien mediante la respuesta *True* del método *initialize*.

```
print("Initialized: %s \n" % p.initialize())
```

El método más relevante para esta aplicación es *get_ distance_ simple* que, después de un procesado de la información desde el dispositivo sonar, nos devuelve un diccionario con una estimación y un nivel de confianza de la distancia al obstáculo.

Una de las formas de implementarlo es la siguiente:

```
distance = int(p.get_distance_simple().get("distance"))/1000
confidence = int(p.get_distance_simple().get("confidence"))
```

Hay que tener en cuenta las unidades en que vienen los diferentes campos de los mensajes. Es por esto por lo que se convierte de milímetros a metros en el ejemplo anterior.

5.2. Integración del sensor Blue Robotics Ping 360

En los siguientes apartados se describe la integración física (*hardware*) y la lógica (*software*) del sensor Ping 360.

5.2.1. Conexión física

Las dimensiones del dispositivo, el montaje y el conexionado del sensor Ping 360 se realizan como se detalla a continuación:

- Montaje en la estructura del robot: las medidas del segundo dispositivo sonar son las que se presentan en la figura 5.3, proporcionada por el fabricante.

  El sonar se monta en la parte superior de la estructura del vehículo haciendo uso de una montura en la parte central derecha del ROV.

- Conexión eléctrica y de interfaz de comunicación: se conecta la alimentación del dispositivo, asegurando que el cable rojo se conecta a tensión de 11 a 18 VDC, y el cable negro a tierra (GND) y se revisa la polaridad de la conexión. Esta revisión es importante ya que el dispositivo Ping 360 no dispone de protección contra polaridad inversa, pudiéndose llegar a dañar la placa de forma irreversible.

  La conexión de datos se realiza mediante un adaptador en línea JST-GH a JST-GH al conector JST-GH perteneciente al sensor Ping 360. El otro extremo de la placa, se conecta el cable JST-GH a USB-A.

  Para terminar, se conecta el USB-A a un ordenador para permitir la comunicación entre el sensor y el sistema de control.

  Este procedimiento garantiza tanto la alimentación adecuada del sensor como el establecimiento de una interfaz de comunicación funcional entre el Ping 360 y el entorno de procesamiento de datos.

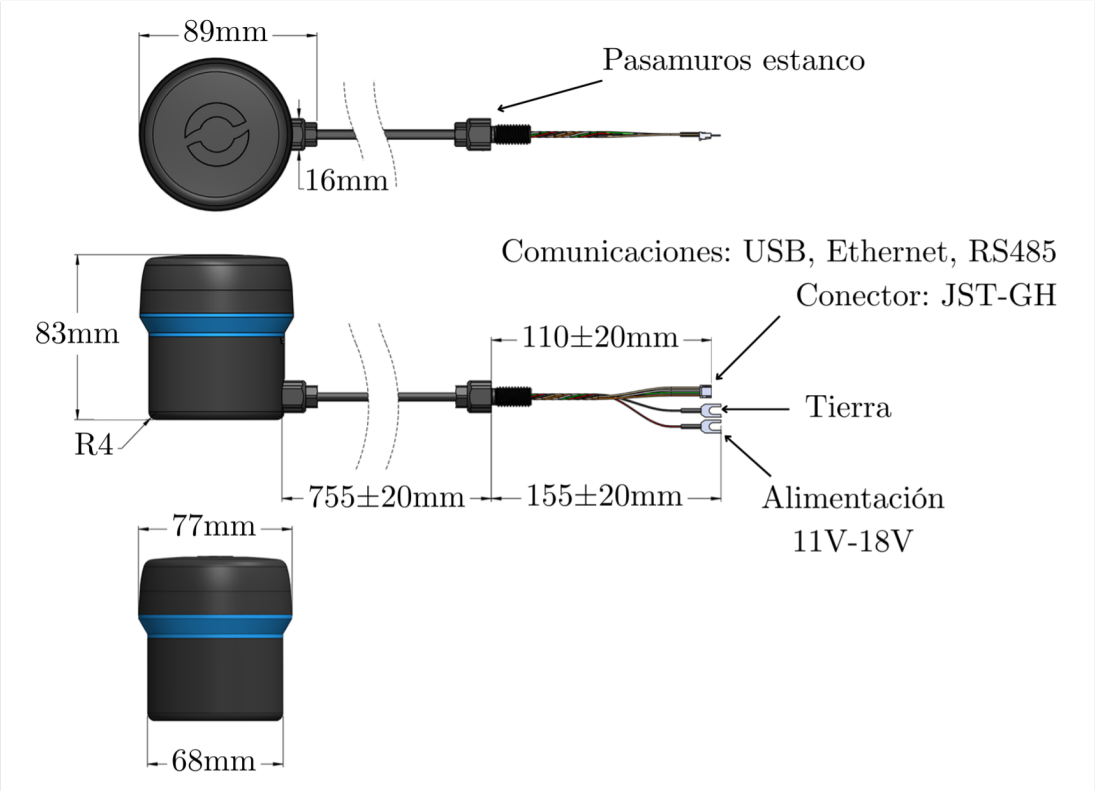


Figura 5.3: Dibujo técnico del dispositivo Ping 360 y su cableado [43].

5.2.2.Integración software

La comunicación entre la SBC y el sensor sonar se realiza a través de una IP y un puerto UDP y mediante el protocolo Ping Protocol tal y como se explicó con el dispositivo anterior. Aunque en este caso los mensajes definidos para el dispositivo Ping 360 son los listados en [44].

Los más interesantes en este caso son el mensaje *2300 device_data* y *2601 transducer* y los campos que contienen se presentan en las tablas 5.3 y 5.4.

El primero es un mensaje de obtención de información (*Get*) permitiéndonos recuperar la información relacionada con la transmisión y el segundo se trata de un mensaje de control (*Control*) que permite ordenar al sonar transmitir y registrar en el ángulo solicitado con las características de transmisión especificadas.

Tabla 5.3: Contenido del mensaje 2300 device_ data.

Tipo	Nombre	Descripción	Unidades
u8	*mode*	Modo de operación (1 para Ping360).	—
u8	*gain_ setting*	Ajuste de ganancia analógica (0 = baja, 1 = normal, 2 = alta).	—
u16	*angle*	Ángulo de la cabeza (posición del cabezal).	gradianes
u16	*transmit_ duration*	Duración de la transmisión acústica (1 – 1000 µs).	us
u16	*sample_ period*	Intervalo entre muestras de intensidad (en incrementos de 25 ns: 80 – 40000 / 2 – 1000 µs).	25 ns
u16	*transmit_ frequency*	Frecuencia de operación acústica (500 – 1000 kHz). Por el ancho de banda del receptor sólo resulta práctico usar 650 – 850 kHz.	kHz
u16	*number_ of_ samples*	Número de muestras por eco reflejado (valores admitidos: 200 – 1200).	muestras
u16	*data_ length*	Longitud del *array* que sigue.	—
u8[]	*data*	*Array* de valores de intensidades de retorno registrados a intervalos regulares dentro de la región de exploración. El primer elemento es el más cercano al sensor y el último el más lejano.	—

Tabla 5.4: Contenido del mensaje 2601 transducer.

Tipo	Nombre	Descripción	Unidades
u8	*mode*	Modo de operación (1 para el Ping360).	—
u8	*gain_ setting*	Ajuste de ganancia analógica (0 = baja, 1 = normal, 2 = alta).	—
u16	*angle*	Ángulo del cabezal.	gradianes

u16	*transmit_duration*	Duración de la transmisión acústica (1 – 1000 µs).	us
u16	*sample_period*	Intervalo entre muestras de intensidad, en incrementos de 25 ns (80 – 40000 / 2 – 1000 µs).	25 ns
u16	*transmit_frequency*	Frecuencia de funcionamiento acústico (500 – 1000 kHz). Por el ancho de banda del receptor sólo resulta práctico usar 650 – 850 kHz.	kHz
u16	*number_of_samples*	Número de muestras por eco reflejado (valores admitidos: 200 – 1200).	muestras
u8	*transmit*	0 = no transmite; 1 = transmite cuando el transductor alcanza el ángulo especificado.	—
u8	*reserved*	Reservado.	—

Habiendo instalado previamente las dependencias correspondientes procedemos de manera muy parecida que con el dispositivo anterior. En este caso el objeto a crear es un objeto *Ping360* y su inicialización es idéntica a la de los dispositivos Ping Sonar. Empezamos declarando el objeto correspondiente:

```
p = Ping360()
```

A continuación usamos el método *conect_udp* que nos permite conectarnos al dispositivo vía IP en un puerto UDP definido:

```
p.connect_udp("0.0.0.0", int(6676))
```

Y seguidamente verificamos que todo ha ido bien mediante la respuesta *True* del método *initialize*:

```
print("Initialized: %s \n" % p.initialize())
```

Los métodos asociados a los mensajes que nos interesan son *get_device_data* para el mensaje *2300 device_data* y *transmitAngle(angle)* para el mensaje *2601 transducer*.

Una de las formas de implementarlos es:

```
device_data = p.get_device_data()
```

```
transducer = p.transmitAngle(300)
```

A continuación, se procede a extraer, de cada diccionario devuelto, la información que resulte más interesante. En nuestro caso son los campos de *angle, data* y *sample_period.*

Con esto ya tenemos la posición angular, los valores de intensidad de dicha posición angular en formato hexadecimal y el periodo de muestreo.

Pero estos datos recién extraídos no pueden ser interpretados de forma directa. Habremos de acompañar estos métodos con una cierta lógica y funciones complementarias que permitan representar de forma clara ángulo y distancia al obstáculo, si es que lo hubiera.

Para ello se ha implementado una función (*max_filtered*) que permite obtener el índice del máximo de intensidad más próximo al sonar en una posición angular determinada.

Conociendo esto y la distancia entre muestras registradas por el sonar determinamos la distancia al obstáculo más próximo. El código fuente de la función *max_filtered* se puede encontrar en la sección 11.7 del Anexo.

5.3. Ping Viewer

Ping Viewer es una aplicación gráfica (GUI, *Graphical User Interface*) pensada para cualquier dispositivo que implemente Ping Protocol. Permite conectar, configurar, visualizar y registrar los datos acústicos que envía el sonar desde una única aplicación [45].

5.3.1.Instalación y ejecución

Ping Viewer se distribuye como ejecutable para los tres sistemas operativos principales (tabla 5.5):

Tabla 5.5: Distribuciones de Ping Viewer y pasos de instalación.

Sistema	Formato	Pasos de instalación
Windows	.zip	Descomprimir y ejecutar *pingviewer.exe*. Saltar el aviso de seguridad con *Más* información y hacer clic en *Ejecutar de todos modos*.
macOS	.dmg	Arrastrar la app a Aplicaciones. Si *Gatekeeper* bloquea la apertura, habilitarla en *Preferencias - Seguridad y privacidad - Abrir de todos modos*
Linux	.AppImage	Conceder permisos *chmod + x* y lanzar el archivo directamente o vía terminal

Tras instalar el software y conectar el sonar la detección suele ser automática. Aunque existe una opción en el *Device Manager* para ajustes de conexión manuales.

5.3.2.Interfaz del modo Ping1D

La ventana de sonar para el modo Ping1D incluye cuatro elementos clave:

- Lectura de distancia: muestra el último eco válido y su nivel de confianza (verde 100 %, amarillo 50 %, rojo 0 %). Se encuentra en la esquina inferior izquierda de la pantalla.
- Gráfica de retorno del eco: se presenta en el extremo derecho de la pantalla y representa la amplitud del eco frente a la distancia, duplicando la curva en simetría para facilitar la lectura.
- Eje de distancia: Se localiza en el borde izquierdo de la gráfica de retorno del eco y ajusta automáticamente la escala y señala con una flecha roja la distancia actual.
- Cascada: Ocupa la mayor parte del centro de la pantalla y representa sucesivas lecturas en tres dimensiones (distancia: eje vertical; intensidad: paleta de color; tiempo: eje horizontal). La paleta de colores es personalizable.

En la figura 5.4 se muestra un ejemplo de lectura de Ping Sonar a través de Ping Viewer.

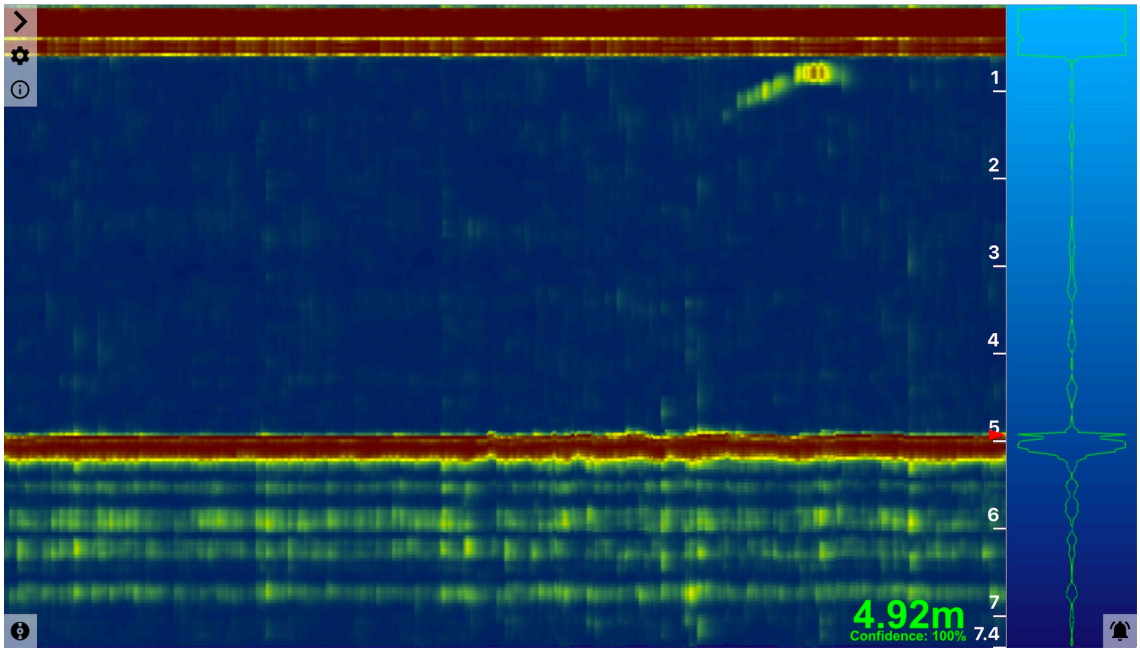


Figura 5.4: Interfaz de Ping Viewer para el modo Ping1D [46].

5.3.3.Interfaz del modo Ping360

En el modo Ping360 la cascada se "envuelve" en un diagrama polar. El eje de distancias queda en el semi-eje derecho del diagrama y la imagen resultante recuerda a una pantalla de radar clásico. Se pueden definir el rango y el ángulo de sector, así como compensar la orientación del ROV, entre otros parámetros.

Este modo también presenta gráfica de retorno, correspondiente a los ecos recibidos en la posición angular que se encuentre escaneando en ese momento. Este modo no incorpora lectura de distancia.

En la figura 5.5 se muestra un ejemplo de lectura de Ping 360 a través de Ping Viewer.

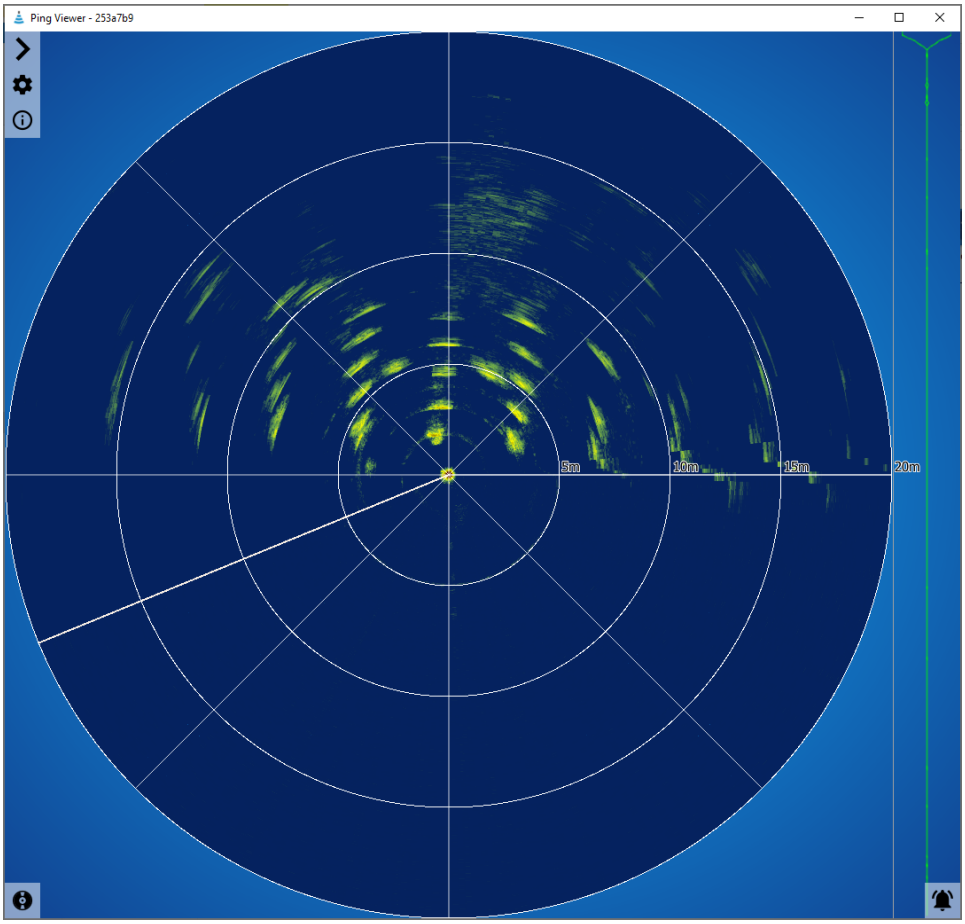


Figura 5.5: Interfaz de Ping Viewer para el modo Ping360 [47].

5.4. Pruebas de la función max_filtered

Para corroborar que la función *max_filtered* ofrecía la salida esperada se realizaron una serie de pruebas con sesiones de Ping 360 previamente grabadas para el proyecto NAUTILUS en una piscina de PLOCAN.

Ping Viewer permite reproducir estas sesiones a partir de un archivo binario y, ejecutando sobre este fichero un *script*, podemos convertirlo y formatearlo para que sea legible en formato UTF. Estos ficheros incluyen todos los mensajes de tipo *2300 device_data* enviados por el sonar durante la sesión.

Conociendo las dimensiones reales de la piscina y la posición del ROV dentro de la misma podemos discernir si la salida de la función *max_filtered* es coherente o no.

Se evaluaron las sesiones 1 y 4, con el ROV centrado en la piscina y desplazado lateralmente, como se puede observar en la figura 5.6. Y se procedió a estimar la distancia a las paredes de la piscina tanto a babor como a estribor.

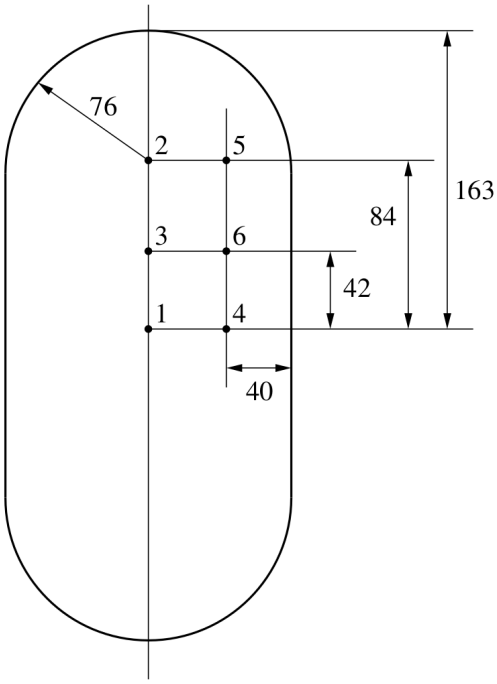


Figura 5.6: Ubicaciones del ROV y dimensiones de la piscina de PLOCAN.

Conociendo el rango de exploración y el número de muestras en dicho rango, la distancia entre muestras se calcula como (5.1):

$$\frac{\text{rango}}{\text{número de muestras}} = \text{distancia entre muestras} \tag{5.1}$$

Para una distancia determinada, conociendo la distancia entre muestras, podemos obtener el índice correspondiente de la siguiente manera (5.2):

$$\frac{\text{distancia}_A}{\text{distancia entre muestras}} = \text{índice}_A \tag{5.2}$$

Sabiendo a qué distancia y en qué índice se espera que exista un pico de intensidad, es decir, un obstáculo, podemos determinar si los datos registrados son acertados y si a la salida de la función *max_filtered* encontramos una respuesta coherente.

- ROV en la posición 1

Cabe destacar que Ping Viewer, durante la reproducción de sesiones, considera el origen de gradianes en el semieje inferior y la progresión del escaneo en sentido horario.

La imagen del registro generada para el ROV en la posición 1 es la que se muestra en la figura 5.5.

Considerando lo anterior, la posición angular correspondiente a 100 gradianes se encuentra sobre el semieje izquierdo y a 300 gradianes sobre el semieje derecho.

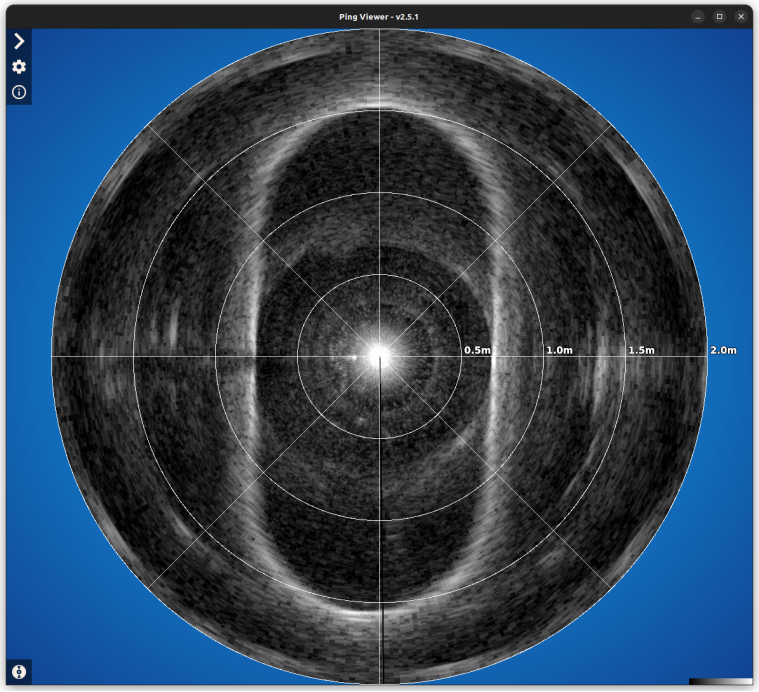


Figura 5.7: Registro del sonar para el ROV en la posición 1.

Como se observa en la figura 5.5, en la sección sonar en torno a 100 gradianes, aparentemente no hay obstáculo, posiblemente fruto de interferencias destructivas en esa zona.

Por ello se ha optado por seleccionar 105 gradianes como posición angular de comparación puesto que consideramos que la distancia variará muy levemente ya que la distancia a la pared es poca. De ser distancias más grandes habría que considerar calcular la nueva distancia real.

La entrada de la función a probar es el conjunto de valores hexadecimales correspondientes a los ángulos de 100 y 300 gradianes extraídos del archivo de texto previamente decodificado.

Para ambos conjuntos de datos las estimaciones de distancia al obstáculo son las representadas en la tabla 5.6.

Tabla 5.6: Resultados de estimación para el ROV en la posición 1.

Ángulo (Gradianes)	Distancia real(m)	Distancia estimada por la función (m)
105	≈0,76	0,76
300	0,76	0,68

- ROV en la posición 4

La imagen del registro generada para el ROV en la posición 4 es la que se muestra en la figura 5.6.

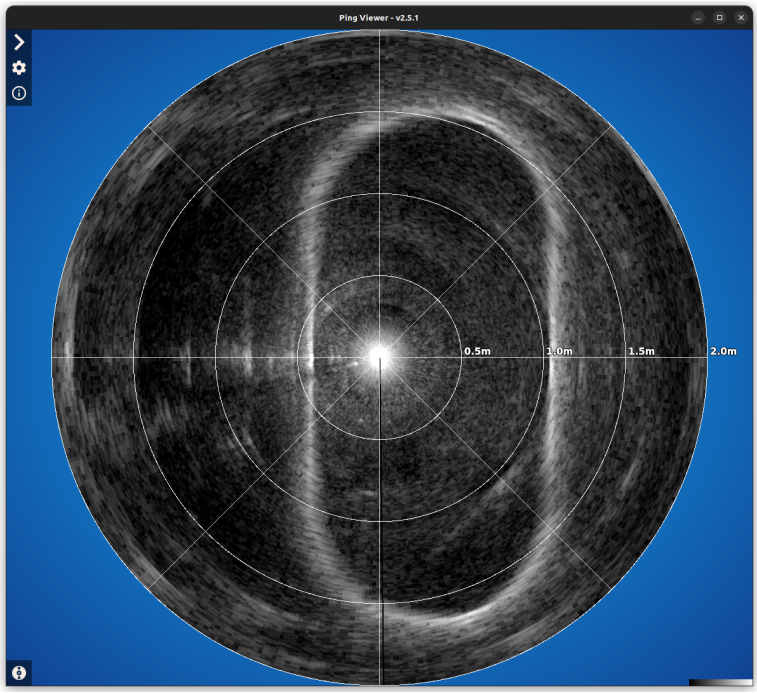


Figura 5.8: Registro del sonar para el ROV en la posición 4.

La entrada de la función son el conjunto de valores hexadecimales correspondientes a 100 y 300 gradianes extraídos desde el archivo binario previamente decodificado.

Para sendos conjuntos de datos las estimaciones del obstáculo son las representadas en la tabla 5.7.

Tabla 5.7: Resultados de estimación para el ROV en la posición 4.

Ángulo (Gradianes)	Distancia real(m)	Distancia estimada por la función (m)
100	0,40	0,40
300	1,12	1,04

6. Implementación y pruebas de funcionamiento de simuladores

6.1. Introducción

En este capítulo se detalla cómo se ha desarrollado el entorno de simulación de los sonares Ping Sonar y Ping 360 a partir de un *script* desarrollado por la empresa Blue Robotics y cómo se han implementado las funciones encargadas de generar los conjuntos de datos que representan los obstáculos durante la ejecución de cada uno de los simuladores. Además, se ha validado su funcionamiento mediante la realización de dos pruebas de funcionamiento para cada uno de los simuladores

6.2. Procedimiento de validación de los simuladores

Para corroborar que los entornos creados se generan coherentemente y que los simuladores atienden las consultas de la manera que cabría esperar de un dispositivo real, se han llevado a cabo las siguientes pruebas para cada uno de los simuladores:

- *Simulador y Ping* Viewer: esta prueba está enfocada a verificar que la generación de los obstáculos implementados es la esperada, según lo previamente configurado. Nos apoyamos en el software Ping Viewer para poder representar visualmente los obstáculos en una pantalla de sonar reglada. De esta manera corroboramos que el rango, tamaño de los obstáculos y distancia a la que se han generado son coherentes con lo configurado.
- *Simulador y Cliente*: busca verificar que el simulador se comporta de la forma esperada tras ser interrogado de la misma manera que se haría con un sonar real. Nos apoyamos en los módulos correspondientes a los dispositivos Ping Sonar y Ping 360 de la librería Ping-Python para crear un objeto de la clase

que corresponda y así usar los métodos disponibles en cada clase para interrogar al simulador.

6.3. Punto de partida

Comenzamos tomando como base un *script* (*ping1d-simulation.py) [48]* desarrollado por Blue Robotics que permite simular un Ping Sonar, conectarte a él desde Ping Viewer y observar variaciones de profundidad basadas en una respuesta sinusoidal (no real). La simulación consiste en un servicio que, gracias a un conjunto de métodos, es capaz de gestionar mensajes de Ping-Protocol, es decir, recibir, desempaquetar, empaquetar y enviar mensajes.

Esto lo consigue apoyándose en los módulos *PingParser.py*, *PingMessage.py* y *definitions*.py de la librearía *Ping-Python*.

El primero se trata de una máquina de estados que recibe los bytes que van llegando por el puerto serie o UDP y los agrupa para componer un mensaje ping completo. Una vez recibe cabecera, longitud, *payload* y CRC válidos, le da salida. *Definitions.py* es una tabla de referencia que describe todo el Ping-Protocol en forma de código. Y permite conocer qué ID corresponde a cada mensaje, cómo se llaman los mensajes en forma humana, cómo se empaquetan y extraen los campos de cada mensaje y construir de forma dinámica las sub-clases *PingMessage*. Y por último *PingMessage.py* representa el mensaje ya decodificado, siendo capaz de serializar y deserializar un mensaje y de representar los campos de este como atributos de clase, haciendo uso de *definitions.py*

6.4. Implementación de un simulador de Ping Sonar

Partimos de la base de que necesitaremos, no solo métodos que nos permitan enviar y recibir los mensajes como si de un Ping Sonar se tratara, sino que debe existir un conjunto de datos que representen el entorno que el sonar estaría registrando. De tal

manera que, para cuando el simulador de sonar reciba una solicitud, este conjunto de datos esté a disposición para atenderla.

Con este fin se ha implementado una función en un *script* (*entornos_ping1d.py*) cuyo objetivo es el de generar los datos correspondientes a un fondo plano a una distancia determinada.

Una función crea un array de 200 valores de intensidades en el que se hace coincidir el índice correspondiente a la distancia deseada con el pico de intensidad que representa al obstáculo.

Cabe destacar que para el caso del simulador de Ping Sonar no era estrictamente necesario generar un conjunto de datos para que fuera funcional, ya que con tener valores de distancia y certeza a disposición cuando se reciba una solicitud *1211 distance_simple* sería suficiente. Pero para que el simulador sea capaz de atender otro tipo de solicitudes como *1300 profile* sí es necesario.

Los scripts *simulador_ping1d.py* y *entornos_ping1d.py* pueden consultarse en la sección 11.5 y 11.3 del Anexo.

6.4.1.Prueba: Simulador de Ping Sonar y Ping Viewer

La primera prueba de funcionamiento se ha realizado ejecutando el simulador contra el Ping Viewer, es decir como si se operara el ROV conectado mediante umbilical y recibiera la información del sonar en el programa de ordenador.

El entorno a generar es un fondo plano a 20 m con una confianza fija de 80%.

Procedemos ejecutando el simulador desde terminal y luego ejecutando el programa Ping Viewer.

La conexión se ha realizado en local en 0.0.0.0 y en el puerto UDP 6676 como se observa durante el proceso de negociación en figura 6.1.

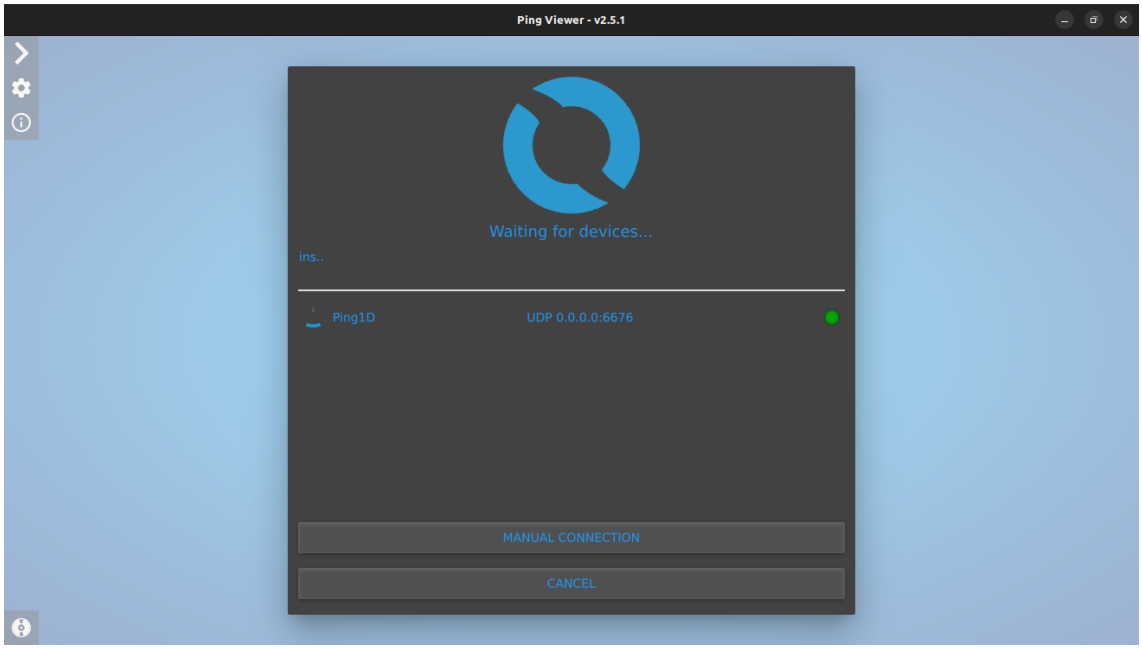


Figura 6.1: Negociación Ping Sonar.

Se ha configurado la salida continua de mensajes *1300 profile* cada segundo para no tener que hacerlo de forma manual desde la interfaz de Ping Viewer, ya que este no trae dicha funcionalidad. Es por esto por lo que, pese a no recibir solicitud de este tipo de mensaje, es igualmente enviado de forma automática.

Como la distancia al fondo no cambia, la respuesta es la que se observa en la ventana de Ping Viewer (figura 6.2) en la que aparece un obstáculo marcado como una línea blanca a una distancia de 20 m.

Figura 6.2: Respuesta Ping Viewer de fondo a 20 metros.

6.4.2.Prueba: Simulador de Ping Sonar y cliente Ping1D

La segunda prueba consiste en ejecutar el simulador contra un cliente, conformado por un objeto Ping1D (de la librería Ping-Python), que lo va a interrogar a intervalos regulares de 1 segundo usando el método *get_ distance_ simple*. Esta es precisamente la manera de operar los Ping Sonar reales desde la SBC.

El entorno a generar, en este caso, será un fondo a 7m con un 80% de confianza.

De modo que procedemos a ejecutar simulador y cliente desde terminal y, tras el proceso de negociación, se comienza a atender las solicitudes del cliente de Ping1D. Y este, a recibir respuestas (figura 6.3).

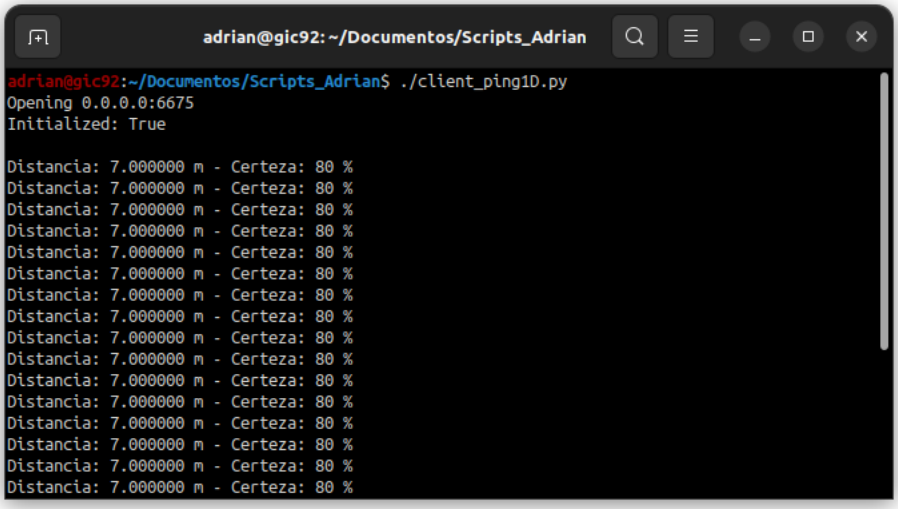


Figura 6.3: Salida por terminal del cliente de Ping1D.

6.5. Implementación de un simulador de Ping 360

Para el desarrollo del simulador Ping 360, usando como plantilla el simulador de Ping Sonar, se modifica el contenido de los campos que identifican al dispositivo y todos aquellos relacionados con el proceso de negociación, además de los campos propios de los mensajes de Ping 360.

También se definen métodos que son necesarios para la configuración del sonar a la hora de ejecutarlo contra Ping Viewer ya que este no es capaz, como cliente, de configurar los campos de número de muestras y periodo de muestreo desde su interfaz

También se han implementado funciones en un *script* (*entornos_ping360.py*) para generar: un conjunto de datos correspondientes a un segmento de pared, de un cierto tamaño, en la dirección especificada y una pared de piscina completamente circular, ambos a la distancia que se desee. Además de funciones de giro y desplazamiento que permitan modificar el entorno en respuesta del movimiento del ROV.

Para generar la pared de un cierto tamaño, determinamos trigonométricamente el ángulo del sector circular cuya cuerda es el segmento de pared. A partir de dicho ángulo hallamos las posiciones angulares que corresponden los extremos de la cuerda, sabiendo que el valor angular del punto más cercano al ROV y perpendicular a la

pared es dado por parámetro. Posteriormente, partimos de la ecuación de la recta en coordenadas polares y calculamos, para cada posición angular entre los valores extremos anteriormente determinados, la distancia a la recta. Cada uno de esos puntos es posteriormente traducido a índices de muestra con el valor de salto de muestra y usado como indicador para rellenar la posición del array correspondiente a ese índice con el valor de intensidad 255. De esta manera se va componiendo un diccionario cuyas claves son los gradianes de 0 a 399 y cuyos valores son el array de intensidades correspondiente a la posición angular.

Cabe destacar que se ha manejado la posibilidad de que el segmento de pared pase por la discontinuidad 0-400 gradianes. Se han comparado los valores correspondientes a los extremos del segmento para determinar la situación de este y para cada caso se ha especificado una lógica que permita recorrer correctamente el sector circular correspondiente.

Para generar la piscina el procedimiento es más sencillo ya que basta con calcular el índice a partir del salto de muestra una sola vez ya que para todas las posiciones angulares va a ser el mismo.

Los scripts *simulador_ping360.py* y *entornos_ping360.py* pueden consultarse en las secciones 11.6 y 11.4 del Anexo.

6.5.1.Prueba: Simulador de Ping 360 y Ping Viewer

Como se realizó con el simulador de Ping Sonar, la primera prueba se ha realizado ejecutando el simulador contra el Ping Viewer. En el caso del simulador de Ping 360, los entornos que se han puesto a prueba son la pared y la piscina.

Para todos los entornos de Ping 360 la conexión con Ping Viewer se ha realizado en local en 0.0.0.0 y en el puerto UDP 6676 como se muestra en la figura 6.4 durante el proceso de negociación. Asimismo, se ha configurado un rango de 5m y 1200 muestras por posición angular.

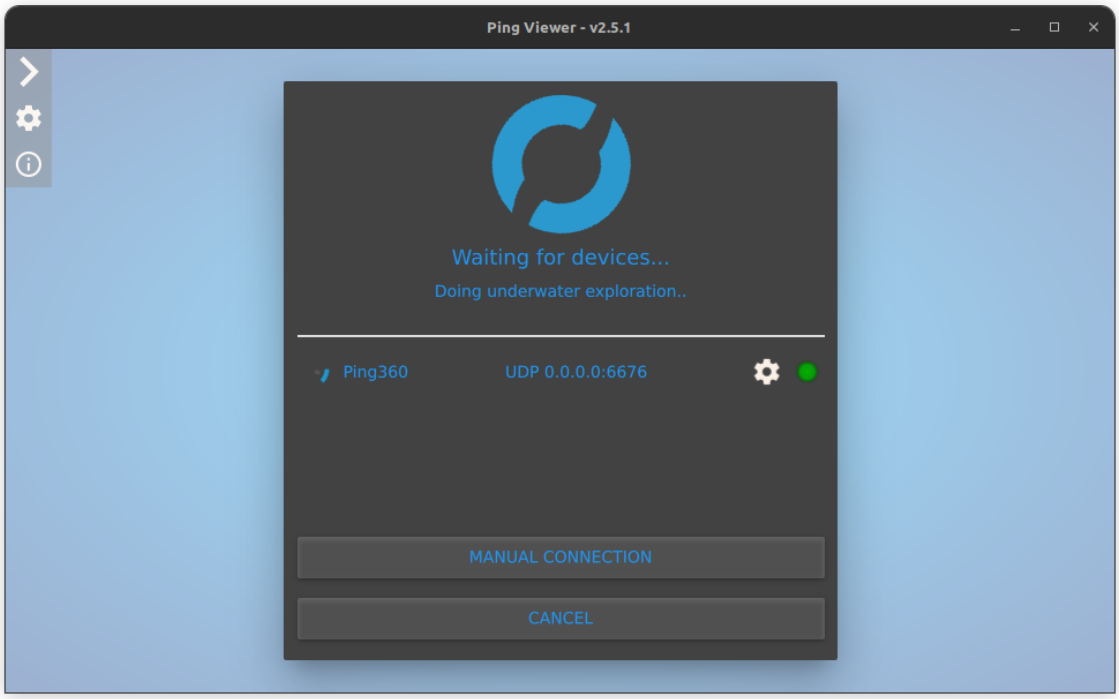


Figura 6.4: Negociación Ping 360.

6.5.1.1. Pared

En la prueba de la pared se configura una pared de 4 metros de longitud a 1 metro de distancia del ROV y a 200 gradianes.

Se observa en la figura 6.5 cómo se genera una pared del tamaño y a la distancia especificada.

También se aprecia una forma de sierra en el segmento de pared. Esto se debe a que el ancho de la pared se ha generado, por simplicidad, de forma radial con respecto al origen polar y no perpendicular a la pared.

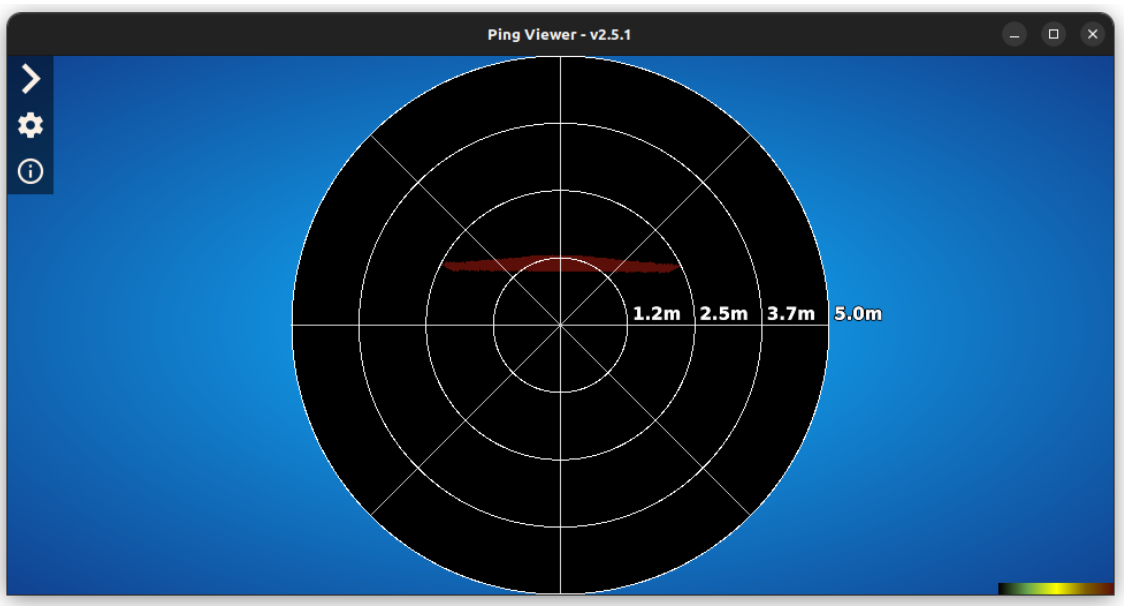


Figura 6.5: Respuesta Ping Viewer de pared a 1 metro.

6.5.1.2. Piscina circular

Para la prueba de la piscina se configura una piscina cuyo borde se encuentra a 3 metros de distancia del ROV.

El resultado se muestra en la figura 6.6, una piscina circular con el borde a la distancia especificada.

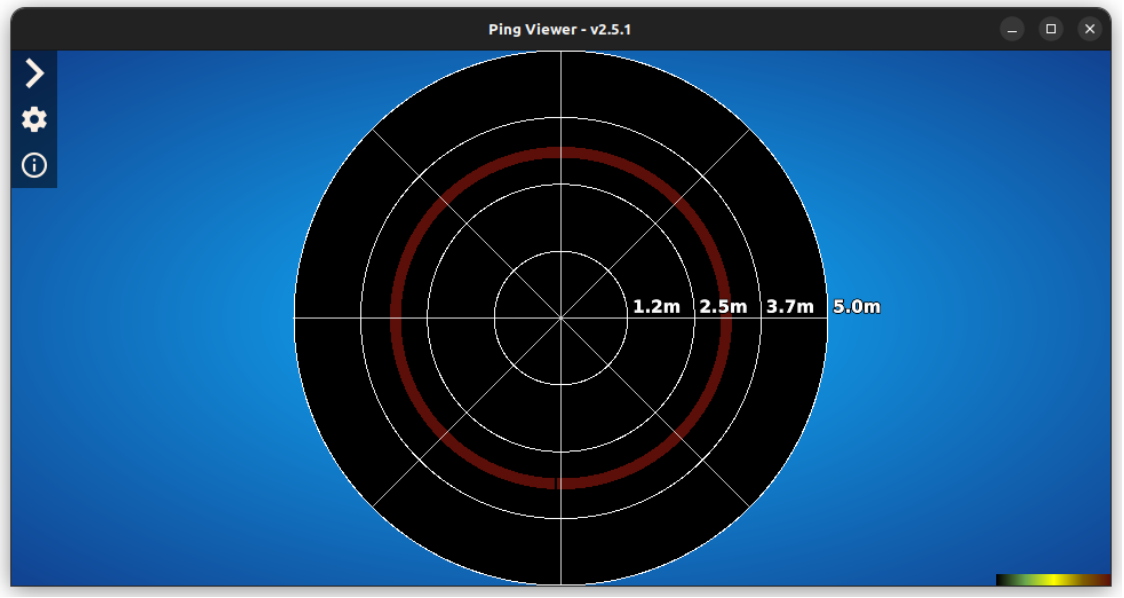


Figura 6.6: Respuesta Ping Viewer borde de piscina a 3 metros.

6.5.2.Prueba: Simulador de Ping 360 y cliente Ping360

La segunda prueba es muy similar a la del Ping Sonar, consiste en ejecutar el simulador de Ping 360 contra un cliente, conformado en este caso por un objeto Ping360 (de la librería Ping-Python), que lo va a interrogar entre 0 y 399 gradianes usando el método *transmit_angle.* Esta la forma en que se operan los dispositivos Ping 360 reales desde la SBC.

De modo que procedemos a ejecutar simulador y cliente desde terminal y, tras el proceso de negociación, se comienza a atender las solicitudes del cliente de Ping360. Y este, a recibir respuestas (figuras 6.7 y 6.8) para los casos de pared y piscina respectivamente.

Para todos los entornos de Ping 360 la conexión con el cliente Ping360 se ha realizado en local en 0.0.0.0 y en el puerto UDP 6675.

6.5.2.1.Pared

El entorno a generar, en este caso, será una pared pequeña de 0,5 metros a 4 metros de distancia en la dirección de 100 gradianes. Del mismo modo que en la prueba 2 del Ping1D se ejecutan simulador y cliente desde terminal. El resultado de detección es el que se muestra en la figura 6.7.

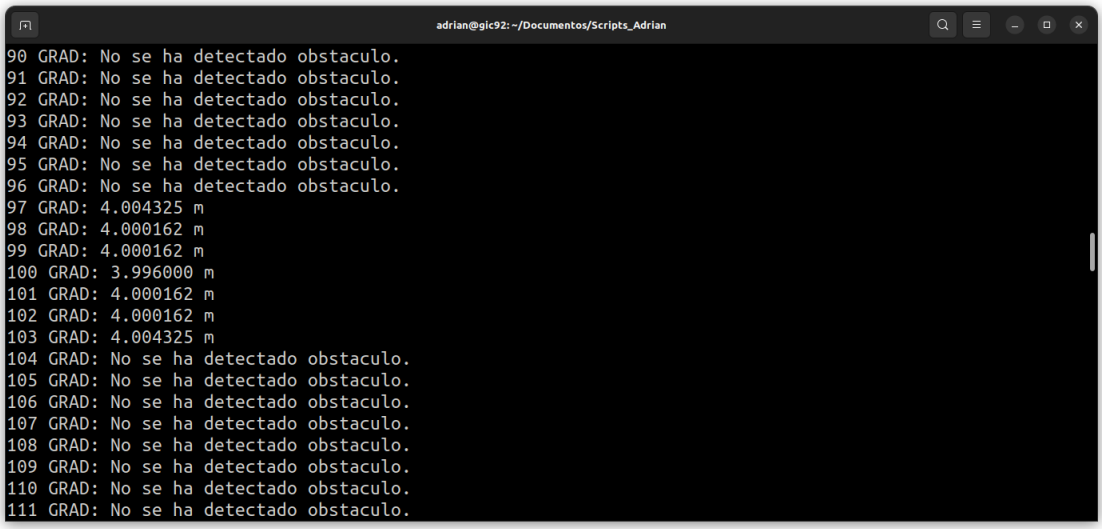


Figura 6.7: Salida por terminal del cliente de Ping360 para la pared.

Se observa que a 100 gradianes se obtiene la distancia mínima al obstáculo de 3,996 metros. La distancia al obstáculo no será exactamente la configurada a menos que dicha distancia sea múltiplo del salto de muestra, como se muestra en la figura 6.9. En cualquier caso, la diferencia es despreciable.

En cuanto a la longitud del segmento, si lo consideramos geométricamente como la cuerda del sector circular cuyo ángulo es el comprendido entre un extremo y otro de la pared generada, tenemos que dicho ángulo es 7 gradianes y que la distancia al punto medio del segmento coincide con la mínima obtenida.

A partir de la razón trigonométrica de la tangente podemos determinar la longitud del segmento. Que resulta ser 0,4398 metros. Una diferencia de 0,0602 metros con respecto a la configurada.

Si calculamos la distancia transversal entre dos puntos a una distancia R con una resolución angular fija de 1 gradián, usando la fórmula trigonométrica de la cuerda para ángulos muy pequeños (6.1):

$$d \approx R \frac{\pi}{200} \tag{6.1}$$

Tenemos que d = 0,0628 m. Lo que significa que para una distancia de 4 metros la separación entre dos puntos angularmente consecutivos es de 0,0628 metros.

Cuanto mayor sea la distancia del ROV a la pared más difícil será poder representar su longitud con exactitud. Esto es debido a que, como se observa en la figura 6.10, los puntos cercanos al origen se encuentran más próximos entre sí y los más alejados más distantes, como consecuencia de la relación anterior. Este efecto se conoce como *cross-range*.

6.5.2.2. Piscina circular

El entorno a generar, en este caso, será una piscina cuyo borde se encuentra a 1m.

Nuevamente se ejecutan simulador y cliente desde terminal. El resultado de detección es el que se muestra en la figura 6.8.

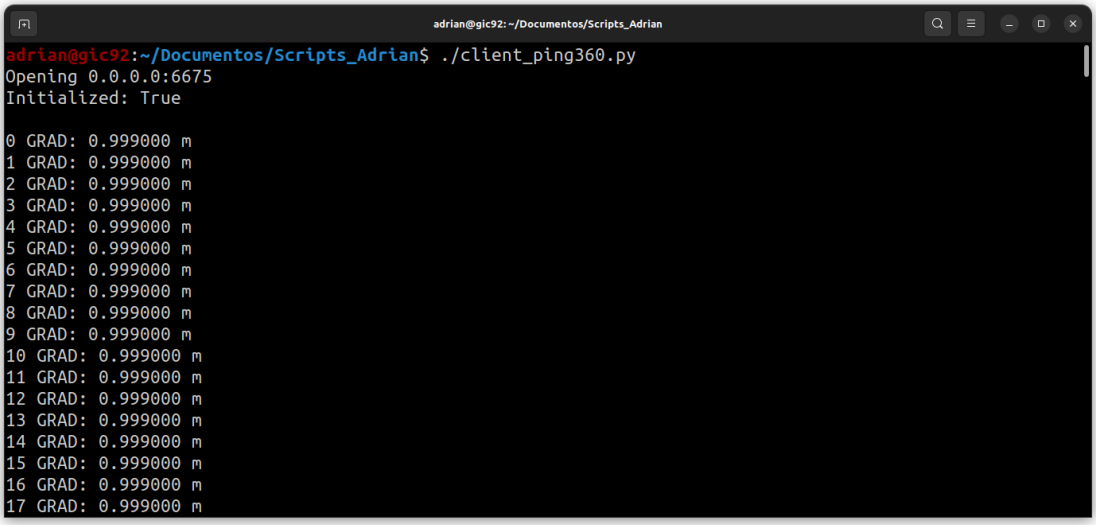


Figura 6.8: Salida por terminal del cliente de Ping360 para la piscina.

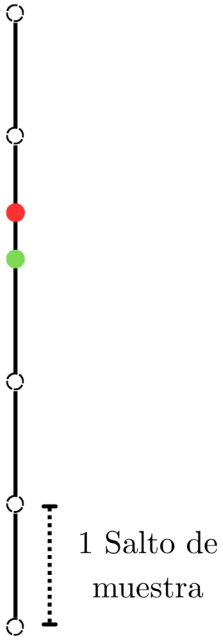


Figura 6.9: Resolución en distancia.

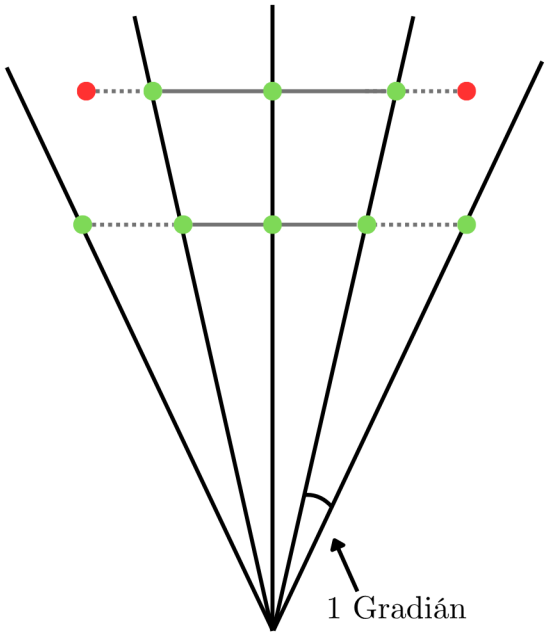


Figura 6.10: Resolución angular y efecto del cross-range.

7. Simulación del ROV y pruebas de simulación conjunta.

7.1. Introducción

En la siguiente sección nos adentramos en el banco de pruebas virtual construido para poner a prueba, de forma conjunta, al BlueROV2 y a los dos simuladores de sonar desarrollados. Primero se explica cómo se crea un ROV completamente virtual mediante el simulador SITL de ArduSub. Seguidamente se presenta Pymavlink, la biblioteca de procesamiento de mensajes MAVLink. A continuación, con estas herramientas se explica el procedimiento de validación, llevamos a cabo las pruebas de simulación conjunta y se analizan situaciones límite como las brechas que aparecen al acercarse demasiado a un obstáculo y que ponen de manifiesto los límites de la resolución angular.

7.2. SITL

Una de las funcionalidades que hacen tan interesante al proyecto ArduPilot es la posibilidad de crear vehículos virtuales. El simulador SITL (*Software In The Loop*) permite gobernar vehículos sin ningún hardware. Consiste en generar, en un ordenador, una imagen ejecutable del código de Autopilot utilizando un compilador estándar de C++. Esta compilación proporciona un ejecutable nativo que permite probar el comportamiento del código sin hardware; es decir, se genera un programa que responde a las mismas entradas y salidas que una placa Autopilot ejecutando el software ArduSub.

A este Autopilot virtual hay que añadir un módulo adicional que permita al ordenador de superficie conectarse al ROV virtual tal y como lo haría con un ROV

real. En el *Companion computer* esta labor la realiza la aplicación MAVProxy, y nada impide ejecutar este mismo software en el ordenador donde se ejecutará el ROV virtual.

A continuación, se detallan los pasos necesarios para generar un BlueROV2 virtual y los programas que se requieren para ello:

Lo primero es clonar el proyecto ArduPilot, pues ArduSub forma parte de él:

```
git clone https://github.com/ardupilot/ardupilot.git
cd ardupilot
git submodule update --init -recursive
```

A continuación, debemos asegurarnos de tener instalados los paquetes necesarios. En el directorio de instalación de ArduPilot hay un *script* que facilita la tarea si se utiliza una distribución Ubuntu:

```
Tools/environment_install/install-prereqs-ubuntu.sh -y
```

Después, actualizamos el *PATH*:

```
. ~/.profile
```

Con esto ya dispondríamos de las herramientas necesarias para crear el BlueROV2 virtual.

Si usamos otra distribución de Linux, el proceso es algo más complejo. El primer requisito es que Python sea de la versión 3.6 o superior (idealmente $\geq$ 3.10, aunque algunas distribuciones modernas, como openSUSE, aún incluyen la 3.6 por defecto). Deberíamos instalar primero algunos módulos de Python con *pip*, de forma global o para nuestro usuario.

```
sudo pip install pexpect empy future mavproxy
# Sin instalación global:
# pip install --user pexpect empy future mavproxy
```

También debemos asegurarnos de que están disponibles los siguientes *bindings* de Python: python3-devel, python3-opencv, python3-wxPython, python3-matplotlib-wx, python3-lxml y python3-pygame.

Añadimos el directorio *autotest* al *PATH*. Por ejemplo (suponiendo que ArduPilot está en el directorio raíz de su usuario):

```
export PATH=$PATH:$HOME/ardupilot/Tools/autotest
```

A continuación entramos en el directorio ArduSub y compilamos el BlueROV2 virtual:

```
cd ardupilot/ArduSub
sim_vehicle.py -L RATBeach --out=udp:0.0.0.0:14550 --map -console
```

Si todo ha ido bien, el BlueROV2 virtual se habrá compilado y estará en ejecución junto con MAVProxy. Aparecerán, como se muestra en la figura 7.1, cuatro ventanas: la consola del Autopilot, la consola de MAVProxy, la consola de SITL y un mapa con la posición inicial del BlueROV2 (RAT Beach, California). Puede activarse además un horizonte artificial desde la consola de MAVProxy. Este BlueROV2 virtual puede manejarse desde *QGroundControl* como si fuera un BlueROV2 real.

El programa *sim_vehicle.py* acepta numerosas opciones (*sim_vehicle.py --help* las lista todas). Entre ellas, puede elegirse la configuración de seis motores *(--frame=vectored)* o la de ocho *(--frame=vectored 6dof)*.

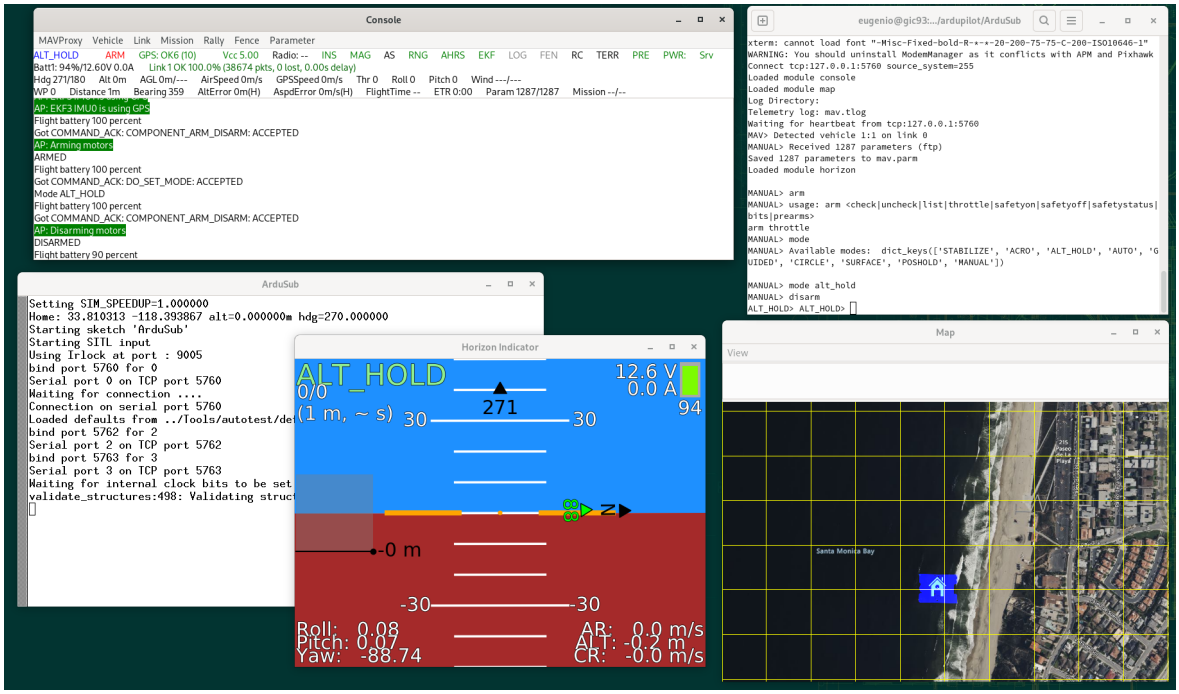


Figura 7.1: BlueROV2 virtual y MAVProxy en ejecución.

7.3. Control del BlueROV2

Con el BlueROV2 virtual en funcionamiento, es hora de elegir las herramientas necesarias para manejarlo. Como se indicó en la sección 4.2, las comunicaciones entre el ordenador de superficie y el *Companion computer* utilizan el protocolo MAVLink. Veamos qué opciones existen para trabajar con él.

7.3.1.Pymavlink

Pymavlink es una biblioteca de procesamiento de mensajes MAVLink de bajo nivel y propósito general escrita en Python. Se ha empleado para implementar comunicaciones MAVLink en múltiples sistemas, incluidos un GCS (MAVProxy), la API para desarrolladores MAVSDK y diversas aplicaciones para *Companion computer*.

Se recomienda instalarla con *pip* para asegurarse de incluir todas las dependencias:

```
sudo pip install pymavlink
# Instalación local:
```

```
# pip install --user Pymavlink
```

La documentación de ArduSub contiene un apartado dedicado a Pymavlink con numerosos ejemplos. Para ilustrar su uso con el BlueROV2 virtual, en la sección 11.14 Anexo se muestra un *script* sencillo que arma y, después, desarma el vehículo.

Al arrancar el BlueROV2 virtual y ejecutar este *script*, la consola de MAVProxy mostrará la secuencia de órdenes recibidas (figura 7.2).

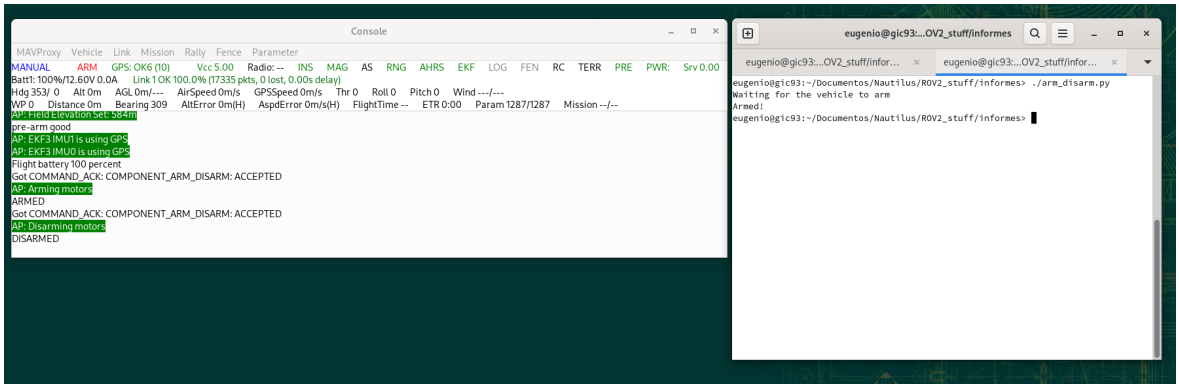


Figura 7.2: Armado y desarmado del BlueROV2 virtual usando Pymavlink.

La biblioteca Pymavlink es muy útil, pero su uso directo requiere un conocimiento profundo del funcionamiento del protocolo MAVLink y de los mensajes que acepta el vehículo sobre el que se desee operar.

7.4. Procedimiento de validación de la simulación conjunta

El objetivo de las simulaciones conjuntas del ROV y del sonar es el de verifica, por un lado, que si ejecutáramos el simulador de sonar desde un ROV real, éste va a operar con él como si estuviera efectivamente rodeado de los obstáculos que se configuren inicialmente. Y, por otro lado, que el entorno de simulación se va a adaptar en tiempo real y de forma coherente con los movimientos del ROV.

En los siguientes apartados se realizarán las integraciones de los simuladores de Ping Sonar y Ping 360 en las funciones que controlan el *heave* y el *surge* del ROV, respectivamente. Dichas funciones se encuentran declaradas en el mismo *script* encargado de controlar el ROV.

Para cada una de las integraciones, los entornos de los simuladores deberán ser capaces de adaptarse a las variaciones de *yaw* y *surge* consecuencia de maniobrar con el ROV. Para verificarlo se llevará a cabo la siguiente prueba para cada una de las integraciones:

Simulador y ROV virtual: consiste en apoyarnos del simulador SITL para realizar una simulación conjunta con cada uno de los simuladores de sonar y obstáculos. Se ejecutarán maniobras que permitan observar la respuesta del simulador a los movimientos del ROV virtual. Esto nos dará una perspectiva real del rendimiento de los simuladores cuando se pretenda poner a prueba un ROV real en un entono simulado.

7.5. Integración del Simulador de Ping Sonar en *script* de control del ROV

El *script* de control del ROV, para el caso del Ping Sonar, pasa a ser el *script* de control del *heave*.

Una vez comprobado que el simulador es capaz de representar el entorno tal y como lo haría un dispositivo Ping Sonar real se procede a su integración con el *script* de control del *heave* y con la función, dentro de esta, encargada de controlar la profundidad del ROV ya que, idealmente, cualquier movimiento en la columna de agua debería tener una respuesta en tiempo real en el conjunto de datos generado por el simulador de Ping Sonar. Durante este proceso cabe destacar que se utilizó Ping Viewer como soporte visual para corroborar que las funciones que modifican los entornos lo hicieran correctamente.

El *script* de control del *heave,* previo a la integración del simulador de Ping Sonar, consta de varios bloques que permiten su correcta ejecución:

- Configuración de frecuencia de mensajes: Está compuesto por la función request_message_interval y determina la frecuencia con la que se solicitará

un tipo de mensaje al Autopilot virtual. Toma por parámetros el *ID* del mensaje y la frecuencia de solicitud de mensaje deseada.

- Control de motores: Lo conforma la función encargada del control de motores, *set_rc_channel_pwm*, que en función del canal (1-18) y del valor de modulación del ancho de pulso (pwm, *pulse width modulation*) que se le pase por parámetro encenderá los motores correspondientes a dicho canal.
- Control del *heave* del ROV: Lo compone la función *set_target_depth_vfr* y se encarga de accionar los motores usando la función anterior hasta alcanzar la profundidad deseada dentro de una cierta tolerancia de ±0,2 metros de profundidad.
- Configuración del ROV: Este bloque tiene como objetivo crear el objeto que permite abrir una conexión MAVLink con el ROV, inicializarlo, armarlo y configurar el periodo de solicitud de mensajes que se vayan a necesitar (por ejemplo: *VFR_HUD, necesario en el bloque anterior*).
- Maniobras: contiene las llamadas a las funciones de control que de forma secuencial dan lugar a que el ROV efectué una maniobra.

Para realizar la integración se ha incluido el bloque:

- Configuración del simulador de Ping Sonar: encargado de declarar el objeto *Ping1DSimulation* cuyo bucle de servicio se ha ejecutado como un hilo, para poder atender a las consultas que se realicen desde el proceso principal.

  Además, se declara el objeto *Ping1D* y se conecta al simulador para poder interrogarlo a través de él.

También se ha modificado la función *set_target_depth_vfr*, del bloque *Control de profundidad del ROV*, para que, tras cada encendido de los propulsores encargados del *heave*, se actualicen los datos del entorno del simulador calculando la nueva distancia al fondo como (7.1):

$$P_{\text{total}} - P_{\text{ROV}} = D_{\text{fondo}} \tag{7.1}$$

Siendo P_{total} la profundidad total, es decir la distancia entre la superficie y el fondo, fijada tras declarar el simulador. P_{ROV} la profundidad del ROV en el momento en que se hace la consulta al ROV mediante el mensaje *VFR_HUD*, que contiene información sobre la altura, entre otros parámetros. Y D_{fondo} la nueva distancia del ROV al fondo marino con la que se genera el nuevo conjunto de datos.

No se ha hecho uso de una función de control de profundidad que consulte al sensor de presión del ROV ya que durante la maniobra emersión, en la segunda lectura del sensor de presión, los valores se reinician como si se encontrara en superficie. Es por esto por lo que se decide actualizar D_{fondo} a partir del valor de altitud del HUD.

El *script* de control del *heave* resultante se puede consultar en la sección 11.8 del Anexo como parte de la prueba que se realizará a continuación.

7.5.1.Prueba: Simulador de Ping Sonar y ROV virtual

Para la realización de esta prueba se ha implementado, dentro de la función que controla la profundidad, una consulta al Ping Sonar inmediatamente después de cada actualización del fondo, para poder corroborar los datos de forma continua.

En primer lugar, ejecutaremos el SITL de ArduSub vía terminal desde el directorio de ArduSub usando el comando:

```
sim_vehicle.py -L RATBeach --out=udp:0.0.0.0:9090 -map -console
```

De las ventanas que se nos presentan tras su inicialización, nos quedamos con la del HUD (*Head Up Display*) que es la que nos va a permitir ver la profundidad del ROV y si esa profundidad es coherente con la distancia al fondo obtenida a través del objeto Ping1D a partir del simulador de Ping Sonar.

Posteriormente ejecutamos el *script* de control del *heave (prueba_control_heave.py)* que contiene las llamadas a la función de control del *heave*. En él se ha declarado el objeto que permite abrir una conexión MAVLink con el ROV del SITL:

```
master = mavutil.mavlink_connection('udpin:0.0.0.0:9090')
```

Se ha configurado el heartbeat, un *thread* que se encarga de comprobar que ROV y el control de tierra o SBC siguen activos:

```
master.wait_heartbeat()
GCS_to_ROV_heartbeat = mavactive(master)
```

Se ha configurado el periodo de solicitud del mensaje que contiene la información relativa al *Head Up Display*:

```
request_message_interval(mavutil.mavlink.MAVLINK_MSG_ID_VFR_HUD, 10)
```

Y se ha procedido a armar el ROV:

```
master.arducopter_arm()
master.motors_armed_wait()
```

Además, se ha declarado el objeto *Ping1D* que permite interrogar al simulador de Ping Sonar, también declarado, y que arranca su hilo de servicio:

```
sim1D = simulador_ping1d.Ping1DSimulation()
hilo_sim1D = threading.Thread(target=sim1D.bucle_ping1D, daemon=True)
hilo_sim1D.start()

myPing1D_data_dic = {}
myPing1D=Ping1D()
myPing1D.connect_udp("0.0.0.0",6675)
```

Previa a la inmersión se verifica que el objeto Ping1D se ha conectado satisfactoriamente al simulador de Ping Sonar con la salida *True* del método *initialized*:

```
print("Ping1D initialized: %s \n" % myPing1D.initialize())
```

La posición inicial de ROV es en superficie y se ha configurado un entorno con un fondo plano a 20m de la misma:

```
sim1D._superficie_fondo = 20
sim1D._profile_data
entornos_ping1d.fondo_plano(sim1D,sim1D._superficie_fondo)
```

Como vemos en la figura 7.3, en la parte inferior derecha del HUD se muestra la información relativa a la altitud del vehículo (ALT). Los valores negativos indican que el vehículo se encuentra bajo el nivel del mar.

En el instante del inicio de la prueba, el ROV se encuentra en la superficie, a 0,0 metros de altitud.

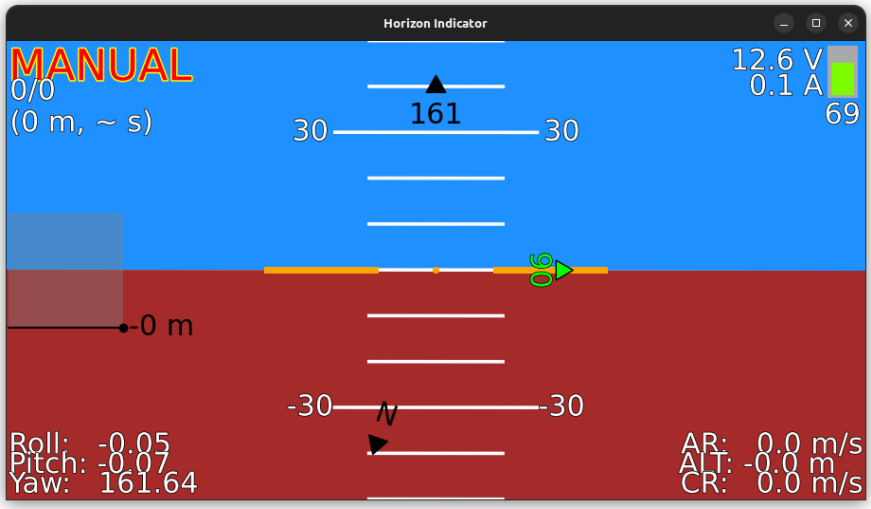


Figura 7.3: HUD antes de la inmersión.

A continuación, se inicia la inmersión programada a 4m de profundidad, primero pasando a modo de control *MANUAL* y posteriormente llamando a la función encargada del control de profundidad:

```
MODE = 'MANUAL'
CUSTOM_MODE = master.mode_mapping()[MODE]
while not master.wait_heartbeat().custom_mode == CUSTOM_MODE:
    master.set_mode(MODE)
print('Manual')
depth = 4
print('########## Bajamos a -4m ##########')
set_target_depth_vfr(depth,sim1D)
print('########## ROV a 4m de profundidad ##########')
```

Y al alcanzar dicha profundidad, la lectura del sonar de distancia al fondo, que leemos por terminal, marca 16,2 metros (figura 7.4).

```
receive message 1211    (distance_simple)
sending message 1211    (distance_simple)

Distancia al fondo: 16.189 m     Certeza: 80
########## ROV a -4m de profundidad #########
```

Figura 7.4: Salida por terminal al alcanzar la profundidad deseada

Por otro lado, en la figura 7.5, se observa que la profundidad alcanzada es de 3,8 metros (considerada aceptable por la tolerancia de 0,2m fijada en la función que controla la profundidad del ROV). Por lo que el resultado del sonar es coherente.

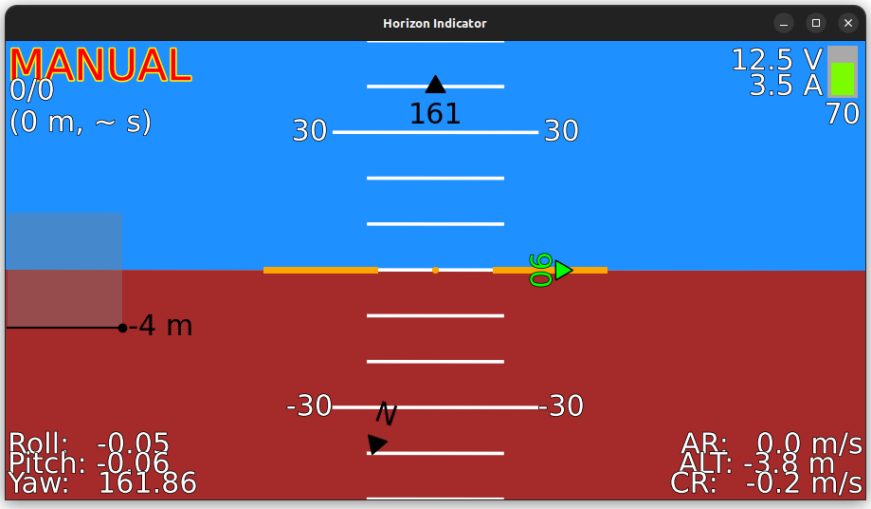


Figura 7.5: HUD al alcanzar la profundidad deseada.

Tras realizar una espera de 5 segundos en modo *ALT_HOLD* se inicia la emersión a superficie:

```
MODE = 'MANUAL'
CUSTOM_MODE = master.mode_mapping()[MODE]
while not master.wait_heartbeat().custom_mode == CUSTOM_MODE:
    master.set_mode(MODE)
print('Manual')

depth = 0
print('########## Subimos a 0m ##########')
set_target_depth_vfr(depth,sim1D)
print('########## ROV a 0m de profundidad ##########')
```

Al alcanzar la superficie, la lectura del sonar de la distancia al fondo, que leemos en el terminal, marca 19,8 metros (figura 7.6) y en la ventana del HUD (figura 7.7) se observa una altitud de -0,1 metros. Volviendo a ser coherente el resultado.

```
receive message 1211    (distance_simple)
sending message 1211    (distance_simple)

Distancia al fondo: 19.838 m     Certeza: 80
########## ROV a 0m de profundidad ##########
```

Figura 7.6: Salida por terminal al alcanzar la superficie.

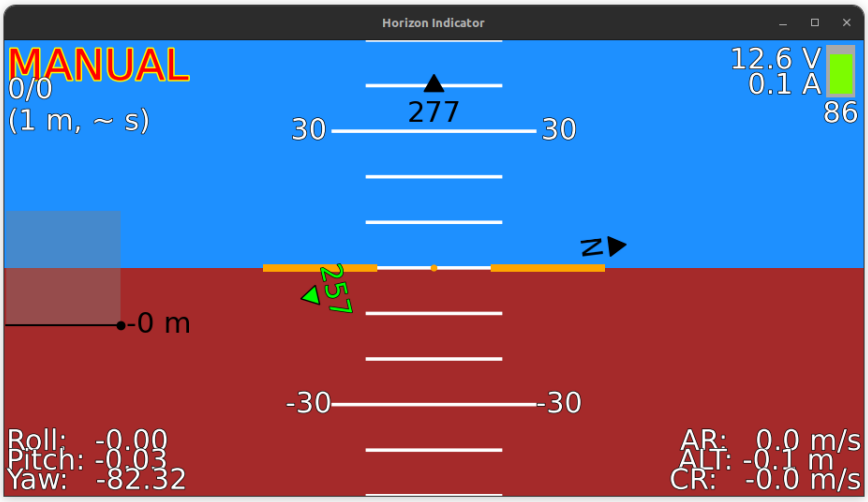


Figura 7.7: HUD al alcanzar la superficie.

7.6. Integración del Simulador de Ping 360 en *script* de control del ROV

El *script* de control del ROV, para el caso del Ping 360 pasa a ser, para el control de *yaw* y control de *surge*, el *script* de control del *yaw* y el *script* de control del *surge,* respectivamente.

Debido a que cualquier movimiento de tipo *yaw* (giro sobre sí mismo) o *surge* (avanzar/retroceder) debería tener una respuesta en tiempo real en el conjunto de datos generado por el simulador de Ping 360 se procede a la integración del simulador con los *scripts* de control del *yaw y surge* y de las funciones, que tienen como objetivo controlar dichos movimientos. La integración se ha llevado a cabo en *scripts* independientes para el *yaw* y para el *surge* por comodidad y es posible ya que no se busca realizar maniobras que involucren más de un grado de libertad en su ejecución.

Partimos de un *script* base, previo a la integración del simulador de Ping 360, que consta de los siguientes bloques: *Configuración de frecuencia de mensajes*, *control de motores, configuración del ROV* y *maniobras*. Idénticos a los del Ping Sonar. Y se incluyen adicionalmente los bloques:

- Funciones auxiliares: encargadas de realizar conversiones de radianes a grados (*a_grados*), de grados a gradianes (*a_gradianes*) y de normalizar grados entre -180º y 180º (*normaliza_180*).
- Funciones de consulta al Ping 360: cuyo objetivo es el de realizar consultas al Ping 360 en un sector sonar pasado por parámetro (get_distance_object_range) y selectivas

A continuación, se individualiza el *script* base para el control del *yaw* y el control del *surge* como se comentó al inicio de la sección.

En el *script* de control del *yaw* se contempla el bloque:

- Control del *yaw* del ROV: encargado de controlar el *yaw* del vehículo de dos formas distintas: una de manera absoluta con respecto al norte magnético (*yaw = 0*) y otra de manera relativa con respecto al *yaw* actual del vehículo. Para ello hace uso de las funciones *set_rc_yaw* y *set_rc_yaw_rel*.

Por otro lado, en el *script* de control del *surge* se contempla el bloque:

- Control del *surge* del ROV: que tiene el objetivo de, mediante la función *set_rc_advance*, controlar el surge del vehículo accionando los motores encargados de avanzar o retroceder durante un determinado tiempo dado por parámetro.

Para comenzar con la integración, se ha incluido tanto en el *script* de control del *yaw* y como el del *surge* el siguiente bloque:

- Configuración del simulador de Ping 360: cuya función es la misma que en la integración del Ping Sonar pero implementando objetos *Ping360Simulation* y *Ping360* en su lugar.

A continuación, en las siguientes secciones, se detalla cómo se han modificado las funciones de control del *yaw* y del *surge*, previamente implementadas en sus respectivos *scripts*, para que actualicen el entorno mientras sean usadas.

7.6.1.1. Yaw

Cabe destacar que el *yaw* del vehículo se maneja dentro de los mensajes MAVLink como un valor de tipo *float* que varía entre -π y π radianes, siendo 0 radianes el norte magnético. Esto es debido a que el estimador de trayectoria (EKF, Filtro de Kalman Extendido) de ArduSub se basa fundamentalmente en el magnetómetro, que mide el vector del campo magnético terrestre y fija el origen en esa dirección.

De aquí en adelante se hará uso del término guiñada para hacer referencia al giro que realiza el propio vehículo, sobre sí mismo y relativo al *yaw*. Es decir, que el *yaw* se maneja en términos absolutos, y la guiñada en términos relativos.

Dentro del *script* de control *yaw* se han implementado dos funciones de giro.

Por un lado, *set_rc_yaw* que orienta el vehículo en la dirección del *yaw* que se pasa por parámetro haciendo uso de la función *set_rc_channel_pwm* para encender los motores encargados del *yaw*. Esta función no se ha modificado ya que en su lógica no se contempla la alteración del entorno. Su función dentro de las pruebas es la de fijar orientaciones iniciales.

Por otro lado, para facilitar la operatividad del ROV al resultar más cómodo manejarlo en términos relativos que en términos absolutos, se ha implementado la función *set_rc_yaw_rel,* que orienta el vehículo de forma relativa al valor actual de *yaw*. Esta función sí contempla en su lógica la modificación del entorno, es por ello por lo que acepta por parámetro un objeto de tipo *ping360_simulation.*

Dicha lógica consiste en fijar un *yaw* objetivo y, para cada encendido de motores, comparar el *yaw* anterior al encendido con el *yaw* actual, posterior al encendido, hasta alcanzar el *yaw* objetivo. Usando como criterio una cierta tolerancia inicialmente configurada. Como resultado de dicha comparación se conoce el sentido de la guiñada del ROV (tabla 7.1) y por ende el sentido del desplazamiento del conjunto de datos del sonar. El cual solo será modificado si el encendido de motores supone una variación de *yaw* superior a 1 gradián.

Tabla 7.1: Sentido del desplazamiento de datos en función del yaw.

yaw actual – yaw anterior	Signo	Sentido de la guiñada	Sentido del desplazamiento de datos
Yaw actual > Yaw anterior	+	Estribor	Antihorario
Yaw actual < Yaw anterior	–	Babor	Horario

La función que modifica el entorno a consecuencia de la variación del *yaw* del vehículo es la función *giro*. Que toma un valor de gradianes por parámetro y lo suma o resta a todos los ángulos del conjunto de datos del sonar para que el entorno se mantenga coherente con la actitud del ROV. Teniendo en cuenta el sentido de desplazamiento de los datos de la tabla 7.1, la representación del impacto de las guiñadas en el conjunto de datos del sonar son las que se muestran en las figuras 7.8 y 7.9.

Figura 7.8: Guiñada 45º a estribor.

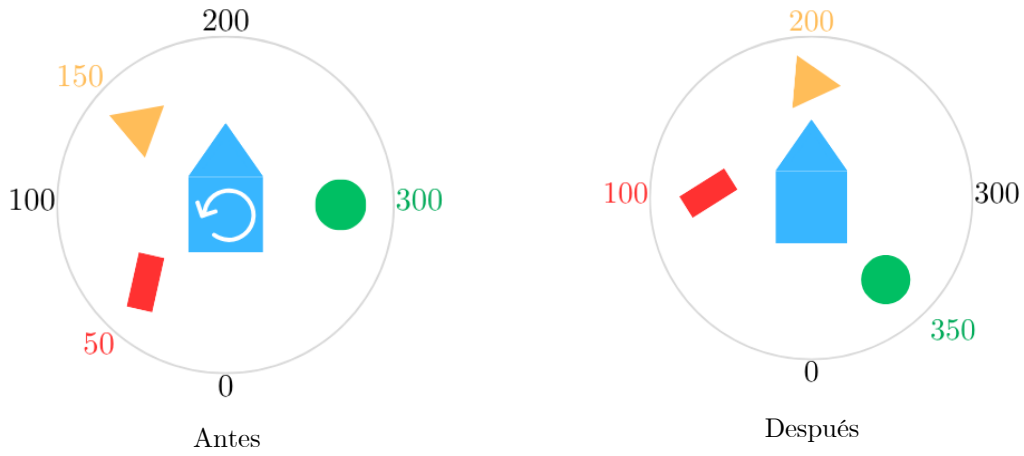


Figura 7.9: Guiñada de 45º a babor.

Cabe destacar que la función de control del *yaw set_rc_yaw_rel* también acepta por parámetro ángulos fuera del rango [-180º, 180º].

El *script* de control del *yaw* resultante se pueden consultar en la sección 11.9 del Anexo como parte de la prueba que se realizará a continuación.

7.6.2.Prueba: Simulador de Ping 360 y ROV virtual – *Yaw*

En los siguientes apartados se explican las pruebas realizadas con el simulador de Ping 360 y el ROV virtual realizando maniobras de guiñada para los obstáculos: pared y piscina circular.

7.6.2.1.1. Pared

Se ha evaluado el comportamiento de una pared generada por el simulador de Ping 360 frente a los movimientos de guiñada del ROV para diversas operaciones, variando la posición inicial con respecto al norte magnético, y el sentido de la guiñada del ROV.

Como en la prueba *Simulador de Ping Sonar y ROV virtual*, lanzamos el SITL, nos quedamos con la ventana del HUD y ejecutamos el *script* de control del *yaw (prueba _control_yaw_pared.py)* donde se crea el objeto que permite abrir una conexión MAVLink con el ROV del SITL.

Se ha configurado el *heartbeat*, como en casos anteriores, el periodo de solicitud del mensaje que contiene la información relativa al *Head Up Display* y también a armar el ROV.

A continuación, se ha declarado el objeto *Ping360* que permite interrogar al simulador de Ping 360, también declarado, y que arranca su hilo de servicio:

```
sim360 = simulador_ping360.Ping360Simulation()
hilo_sim360 = threading.Thread(target=sim360.bucle_ping360, daemon=True)
hilo_sim360.start()

myPing360_data_dic = {}
myPing360 = Ping360()
myPing360.connect_udp("0.0.0.0",6676)
```

También, fijamos un rango de 5m y 1200 muestras por posición angular:

```
sim360.set_range(5,1200)
```

Previa a la inmersión se verifica que el objeto Ping1D se ha conectado satisfactoriamente al simulador de Ping Sonar con la salida *True* del método *initialized*:

```
print("Ping360 initialized: %s \n" % myPing360.initialize())
```

Todas las maniobras que se presentan a continuación son en superficie, ya que el resultado no sería diferente habiendo sumergido el ROV y entrando en modo ALT_HOLD para mantener la profundidad.

En primer lugar, las maniobras 1 y 2 han consistido en realizar un giñado de 180^{o} recorriendo los valores de *yaw de* positivos a negativos y viceversa entre 90^{o} y -90^{o}, pasando por la discontinuidad angular entre $-\pi$ y π radianes.

Se busca verificar que el entorno se actualiza correctamente, que el vehículo continúa guiñando entre $-\pi$ y π radianes y en definitiva que se maneja satisfactoriamente la discontinuidad desde la función encargada de controlar la guiñada del ROV.

En la maniobra 1 (figura 7.10) se parte de una orientación inicial del ROV hacia el Este y en todas las maniobras, de una pared a 2m en 0 gradianes (en la dirección de la popa del vehículo):

En primer lugar, generamos el obstáculo con las características anteriormente descritas:

```
sim360._sonar_data = entornos_ping360.pared(sim360,2,0)
```

Fijamos la situación inicial usando la función *set_rc_yaw*, la cual no modifica el entorno. Esto significa que la posición relativa del obstáculo con respecto al vehículo no cambia:

```
yaw = 90
set_rc_yaw(yaw)
```

De esta manera el vehículo queda orientado al Este.

Se comprueba que el obstáculo se encuentra detrás del ROV realizando una consulta al sonar, en 0 gradianes, haciendo uso de la función *get_distance_object*:

```
get_distance_object(sim360,myPing360,0)
```

Tras esto se procede a realizar la maniobra de guiñada:

```
yaw_rel = 180
set_rc_yaw_rel(sim360,yaw_rel)
```

Después de la maniobra el vehículo queda satisfactoriamente orientado de forma muy aproximada en la dirección esperada y tras realizar una consulta al simulador de Ping 360 en la dirección de 200 gradianes (en la dirección de la proa del vehículo) encontramos la pared:

```
get_distance_object(sim360,myPing360,200)
```

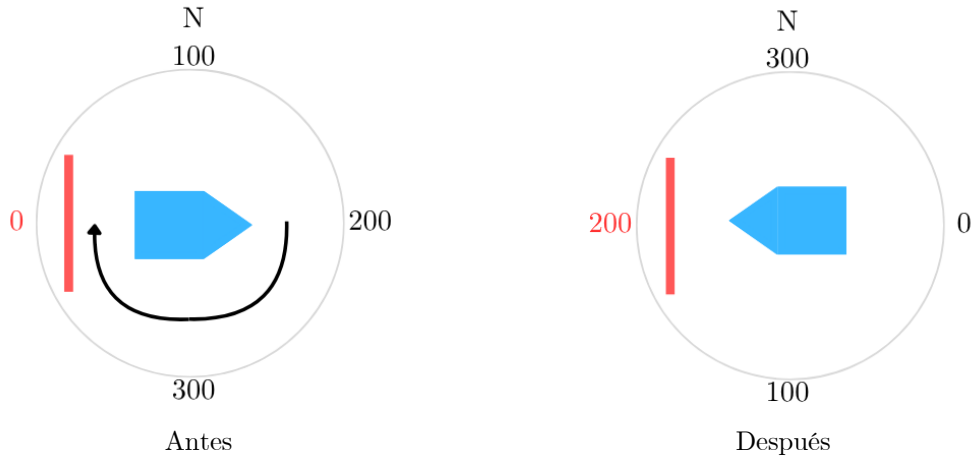


Figura 7.10: Representación de la maniobra 1.

La maniobra 2 se ha realizado de manera idéntica que la maniobra 1 salvo por la orientación inicial y el sentido de la guiñada. En este caso en vez de realizar el recorrido guiñando a estribor, se realiza guiñando a babor desde yaw = -90º.

Al igual que en la maniobra anterior, el vehículo queda orientado en la dirección correcta y tras realizar la consulta al simulador de Ping 360 en la dirección de 200 gradianes encontramos la pared.

Las maniobras 3 y 4 son de la misma índole que las dos primeras, pero con el objetivo de verificar que la actualización del entorno y la guiñada se realizan correctamente a su paso por el origen, en yaw = 0º.

Partimos de la misma orientación inicial del ROV que la maniobra 1, yaw = 90º, pero realizamos una guiñada de 180º a babor (modificando el valor de yaw_rel = *-180)* como se muestra en la figura 7.11. El vehículo queda orientado, como se esperaba y las consultas al simulador de Ping 360 (*get_distance_object*) indican que la pared se detecta en la posición y distancia adecuada.

Posteriormente, se lleva a cabo la operación 4 partiendo de yaw = -90º, guiñando 180º a estribor y el resultado es nuevamente el esperado.

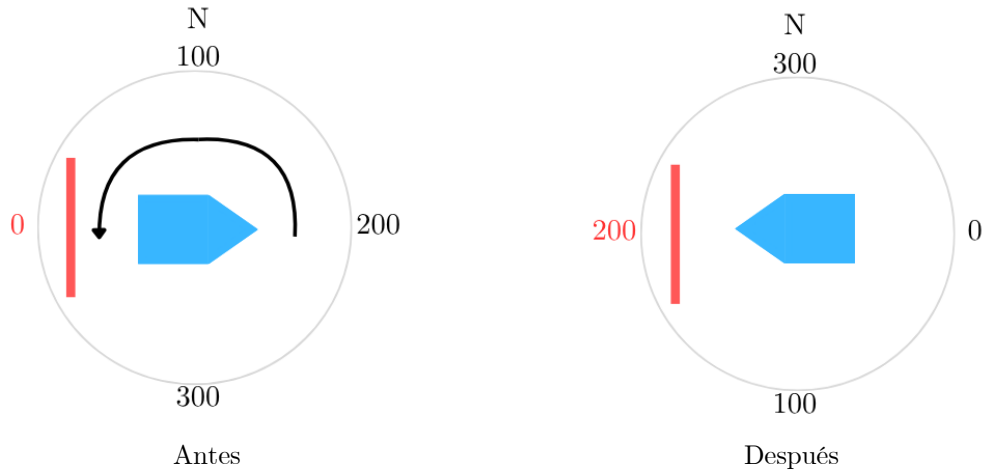


Figura 7.11: Representación de la maniobra 3.

Durante la maniobra 5 se pretende poner a prueba la entrada de valores de yaw relativos fuera del rango [-180º - 180º] a la función *set_rc_yaw_rel*.

Como se muestra en la figura 7.12 antes de guiñar, partimos de una orientación inicial norte en *yaw* = 0º. Después guiñamos 270º a estribor y la operación finaliza con los resultados esperados, ya que tras realizar la consulta al simulador de Ping 360, la pared se detecta a 100 gradianes a babor.

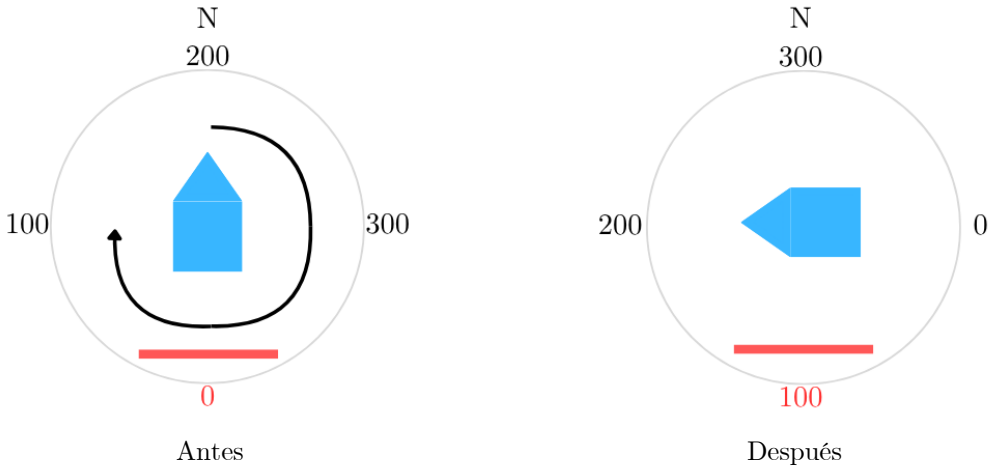


Figura 7.12: Representación de la maniobra 5.

Seguidamente se pone a prueba con una guiñada de 270º a babor, durante la maniobra 6, y el resultado es nuevamente el esperado. Se localiza la pared en 300 gradianes.

7.6.2.1.2. Piscina circular

Se ha evaluado el comportamiento de una pared de piscina circular generada por el simulador de Ping 360 frente a los movimientos de guiñada del ROV.

Tras iniciar el SITL se llama al *script* de control del *yaw* (*prueba_control_yaw_piscina.py*). En él se ha configurado una piscina circular a 3m de distancia del ROV:

```
sim360._sonar_data = entornos_ping360.piscina(sim360,3)
```

La primera maniobra de esta prueba es la maniobra 7. En la que como se muestra en la figura 7.13, se parte de una orientación ESTE, $yaw = 0$, y se realiza una guiñada a babor de 180^{o}. Tras la maniobra se realizan consultas a 100, 200, 300 y 0 gradianes y se observa que las distancias a las paredes de la piscina circular son de 3m, como era de esperar.

Figura 7.13: Representación de la maniobra 7.

Además, se realizó una maniobra 8. En ella se parte de la misma orientación que en la maniobra anterior. Pero en este caso el sentido de la guiñada es de 180^{o} a estribor. Y el resultado es el mismo que en la maniobra anterior.

Esta prueba es limitada en cuanto a conclusiones ya que, tanto antes como después de la maniobra, los resultados de consulta al simulador de Ping 360 serán idénticos, si todo ha salido bien. Es por esto por lo que solo hemos podido verificar que el

conjunto de datos no ha sido alterado de manera anómala. No podemos verificar que, tras la guiñada, el conjunto de datos haya variado de manera coherente con el movimiento del ROV. Aunque es lógico pensar que si en la prueba con la pared el conjunto de datos se ha modificado de manera coherente, en esta prueba también lo ha sido.

7.6.2.2.Surge:

Para el control del *surge* del vehículo se ha implementado la función *set_ rc_ advance.* Esta función será la encargada de hacer avanzar o retroceder el vehículo en base a dos parámetros: dirección de desplazamiento y tiempo de desplazamiento.

Dentro de ella se implementa un bucle de duración igual al tiempo de desplazamiento en el que a cada segundo que transcurre se realiza un encendido de motores.

Hemos supuesto por simplicidad que la velocidad del vehículo cuando se encuentra realizando una maniobra de *surge* es de 0,5m/s. De esta manera determinamos que con cada iteración del bucle el vehículo avanza 0,5 metros. Esto nos facilita la tarea de actualización del entorno.

Para este propósito se ha implementado la función *desplazamiento*, que recibe por parámetros el desplazamiento en x y el desplazamiento en y en metros que se pretende realizar.

Los valores positivos de y coincidirán con la dirección de la proa del vehículo y los negativos con la dirección de popa. El desplazamiento en y es el que involucra el movimiento de *surge* y el x el movimiento de *sway.*

La función *desplazamiento* busca en el conjunto de datos del simulador de Ping 360, para cada posición angular, los valores correspondientes al obstáculo, aquellos con niveles de intensidad iguales a 255, y convierte la coordenada de polar a

cartesiana. A continuación, realiza el desplazamiento y vuelve a convertir la coordenada a polar para componer el nuevo conjunto de datos desplazados.

Los *scripts* de control del *surge* resultantes se pueden consultar en las secciones 11.10 y 11.11 del Anexo como parte de las pruebas que se realizarán a continuación.

7.6.3.Prueba: Simulador de Ping 360 y ROV virtual – *Surge*

En los siguientes apartados se explican las pruebas realizadas con el simulador de Ping 360 y el ROV virtual realizando maniobras de guiñada para los obstáculos: pared y piscina circular.

7.6.3.1.1. Pared

Se ha evaluado el comportamiento de una pared generada por el simulador de Ping 360 frente a los movimientos de avance y retroceso del ROV. Se procede ejecutando el SITL y el *script prueba_control_surge_pared.py* que hace uso de Pymavlink de la misma manera que en pruebas anteriores. De hecho, todo el proceso inicial previo al inicio de las maniobras es idéntico salvo por el entorno y el rango de operación del simulador del sonar.

Partimos de una orientación norte (la inicial cuando se lanza el SITL), rango del simulador de Ping 360 a 10m:

```
sim360.set_range(5,1200)
```

Y una pared, como obstáculo, situada a 5m en 200 gradianes (situación A de la figura 7.14):

```
sim360._sonar_data = entornos_ping360.pared(sim360,5,200)
```

Se ha comprobado que existe una pared en la dirección configurada mediante la función *get_distance_object*:

```
MODE = 'MANUAL'
CUSTOM_MODE = master.mode_mapping()[MODE]
while not master.wait_heartbeat().custom_mode == CUSTOM_MODE:
```

```
   master.set_mode(MODE)
print('Manual\n')

get_distance_object(sim360,myPing360,200)
```

Se ha iniciado la maniobra de avance durante 6 segundos (3m):

```
set_rc_advance(sim360,1,6)
```

Y tras la maniobra se ha vuelto a realizar una consulta al simulador de Ping 360 en la dirección de avance.

Como se esperaba, aparece representado el obstáculo a aproximadamente 2m y 200 gradianes (situación B de la figura 7.14).

De la misma manera iniciamos la maniobra de retroceso durante 6 segundos, para volver a la posición inicial:

```
set_rc_advance(sim360,-1,6)
```

Realizamos una consulta al simulador de Ping 360 y obtenemos una lectura de aproximadamente 5m en 200 gradianes (situación A de la figura 7.14).

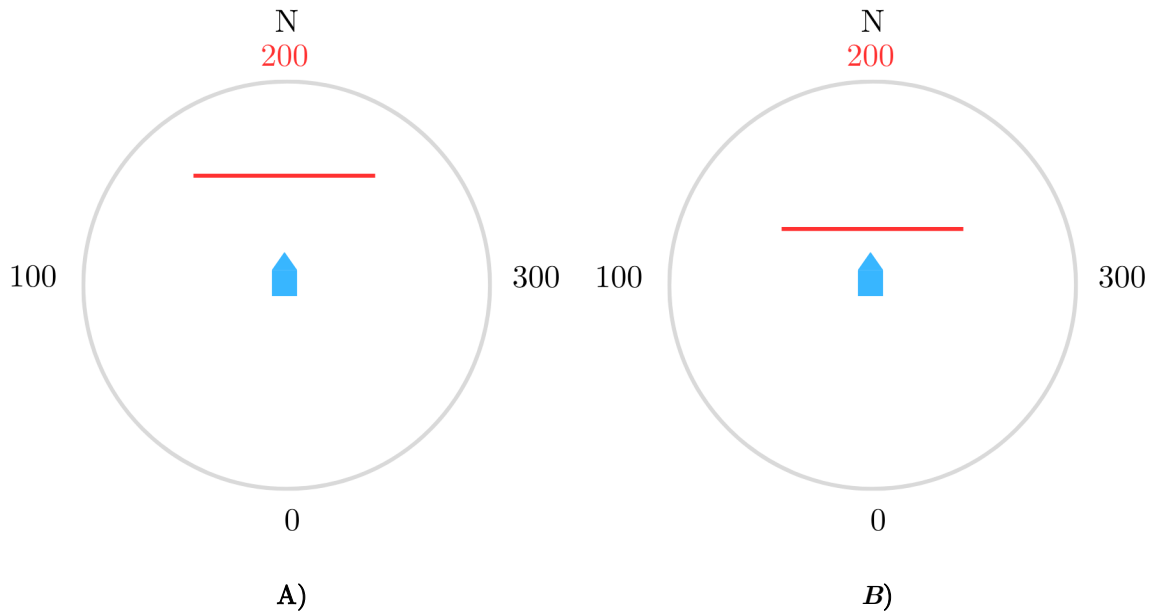


Figura 7.14: Comportamiento de una pared frente a los movimientos de surge.

7.6.3.1.2. Piscina circular

En el caso de la evaluación del comportamiento de la piscina, se procede de la misma manera que en el caso de la pared, pero cambiando el entorno.

Se procede ejecutando el SITL y el *script* de control del *surge* (*prueba_control_surge_piscina.py*) cuya configuración previa a las maniobras es idéntica a la anterior, salvo por el entorno:

```
sim360._sonar_data = entornos_ping360.piscina(sim360,5)
```

Partimos de la misma posición inicial (situación A de la figura 7.15), realizamos la maniobra de avance 3m (situación B de la figura 7.15), y concluimos con la maniobra de retroceso 3m para volver a la posición inicial (situación A de la figura 7.15).

Las lecturas realizadas después de cada desplazamiento fueron nuevamente coherentes.

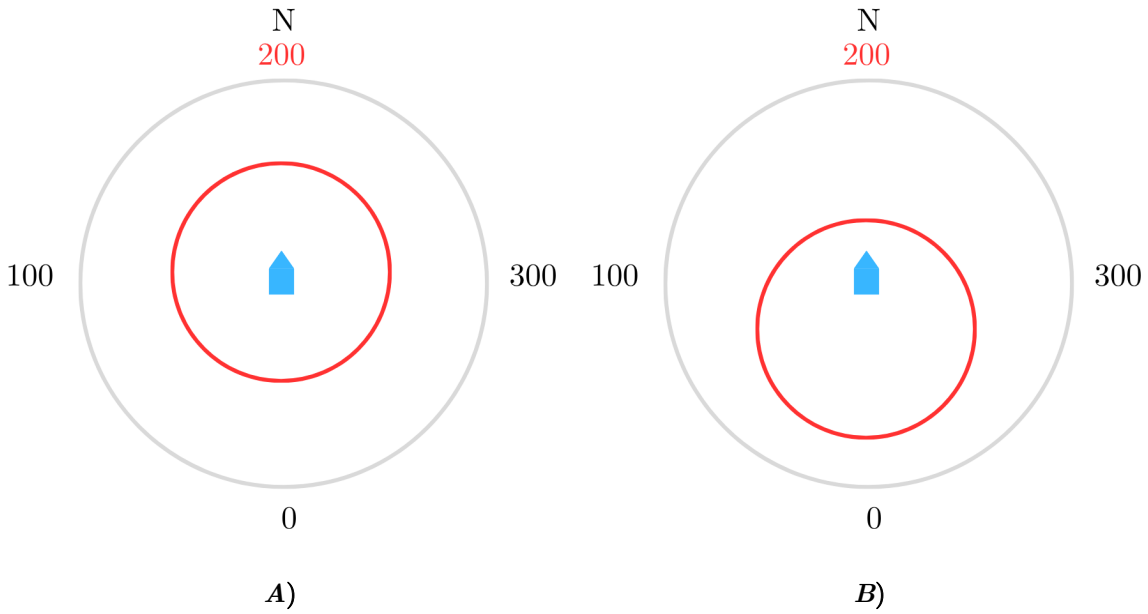


Figura 7.15: Respuesta de una piscina circular frente a movimientos de surge.

7.6.4.Problema de brechas de obstáculo

Tras realizar una consulta al simulador de Ping360 después de una maniobra de avance hacia el obstáculo (por ejemplo: una pared) haciendo uso de la función

get_ distance_ object_ range en un rango de 100 a 300 gradianes, nos encontramos con que, pese a que en 200 gradianes existe un valor de intensidad que representa un obstáculo, la detección del obstáculo no es continua. Y ciertas posiciones angulares adyacentes están vacías tras el desplazamiento. En la figura 7.16 se observa este efecto de manera simplificada, ya que se expone desde una perspectiva de 1 haz por cada 5 gradianes y no de 1 haz por gradián. El efecto real es lógicamente amplificado al haber más haces.

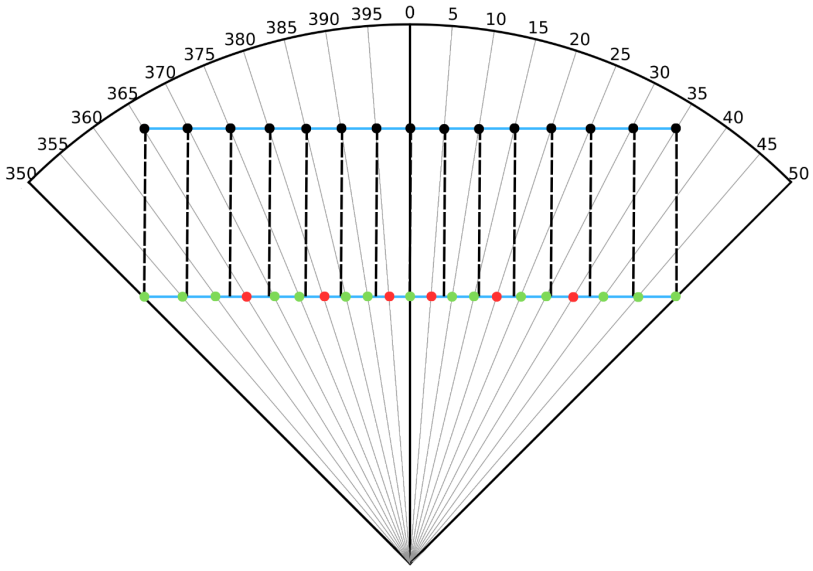


Figura 7.16 Aumento posiciones angulares tras el desplazamiento.

Además, como consecuencia de los redondeos, las posiciones de las muestras que representan el obstáculo no volverán a su posición inicial tras la maniobra de retroceso al punto inicial.

Cuando se genera un obstáculo a una cierta distancia, el generador del obstáculo rellena todas las posiciones angulares en los índices que correspondan para conformar el obstáculo. Al acercarnos, el obstáculo se debe representar de manera más próxima al origen. Y la cantidad de posiciones angulares necesarias para representar dicho obstáculo aumenta. Esto supone un aumento en el número de gradianes por muestra, un problema de resolución angular.

7.6.4.1. Prueba de brechas de obstáculo con script

Sea un obstáculo (pared) de:

- $L = 3m$ Longitud de la pared
- $d0 = 3m$ Distancia inicial al transductor
- $d1 = d0 - 2 \ = \ 1m$ Distancia tras desplazar la pared 2 m hacia el sonar

La apertura angular correspondiente al segmento que representa la pared se obtiene de la siguiente expresión (7.2):

$$\beta \ = \ 2 \ \arctan\left(\frac{L/2}{d}\right) \tag{7.2}$$

β para el caso inicial es (7.3):

$$\beta_0 \ = \ 2 \ \arctan\left(\frac{1.5}{3}\right) \ \approx \ 53.13^{\circ} \tag{7.3}$$

Y para el aso posterior al avance (7.4):

$$\beta_1 \ = \ 2 \ \arctan\left(\frac{1.5}{1}\right) \ \approx \ 101.54^{\circ} \tag{7.4}$$

El número de haces necesarios en cada caso para representar una pared de forma continua son (7.5, 7.6):

$$n_0 = \frac{\beta_0}{\Delta\theta} \approx \frac{53.13^{\circ}}{0.9^{\circ}} \simeq 59 \text{ rayos} \tag{7.5}$$

$$n_1 = \frac{\beta_1}{0.9^{\circ}} \simeq 113 \text{ rayos} \tag{7.6}$$

Tanto antes como después del desplazamiento aparecen 59 ecos (posiciones angulares con valor 255). Pero tras desplazar la pared los haces necesarios para representar la pared de forma continua son casi el doble. Por eso hay saltos entre posiciones angulares.

Esto es lo que se evidencia en *prueba_ brechas_ datos.py* en donde se hace uso de la función *pared_ con_ tamaño* del *script entornos_ ping360.py* para desplazar 2 metros hacia el origen una pared de 3 metros de longitud que se encuentra a 3 metros de distancia y observar lo que sucede con el conjunto de datos que representan el obstáculo. Para ello se hace uso de la función *localizar_ 255* encargada de recorrer el conjunto de datos del sonar y localizar los máximos de intensidad más próximos a este. Devolviendo e imprimiendo por pantalla las tuplas de posiciones angulares e índices correspondientes a los máximos.

La función es cuestión es llamada durante la prueba justo después de generar el conjunto de datos del sonar y justo después de modificarlo con la función *desplazamiento*:

```
sonar_data = pared_con_tamaño(3,3,200)
localizar_255 (sonar_data)

sonar_data_desplazado = desplazamiento(sonar_data, 0, 2)
localizar_255 (sonar_data_desplazado)
```

En las secciones 11.12 y 11.15 se pueden consultar, respectivamente, el *script prueba_ brechas_ datos.py* y la salida por terminal de la prueba.

7.6.4.2. Prueba de brechas en los obstáculos con Ping Viewer

Para visualizar el efecto de este problema efectuamos desplazamientos de diversos obstáculos desde el propio *script* del simulador de Ping 360 usando como cliente el software Ping Viewer.

Para ello cada vez que el barrido del sonar pasa por 0 gradianes se procede a utilizar la función desplazamiento del *script entornos_ ping360.py* para desplazar un obstáculo de pared a 3 m y de una pared de piscina a 3 m.

Las figuras 7.17 y 7.18 representa la situación inicial antes de las maniobras de acercamiento.

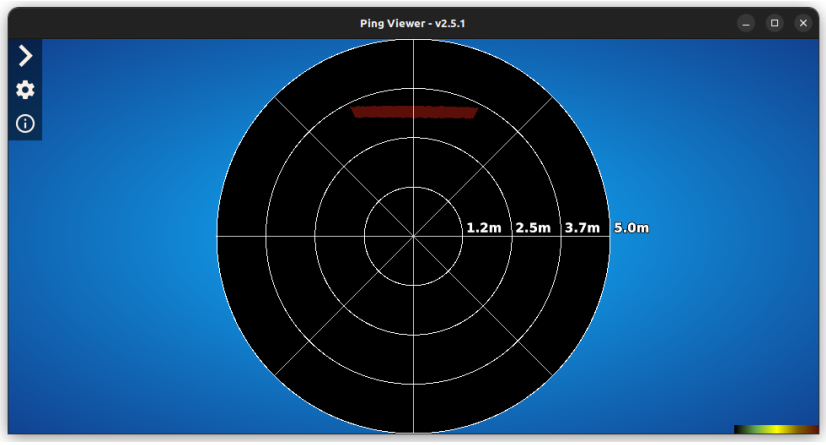


Figura 7.17: Situación previa al acercamiento de la pared.

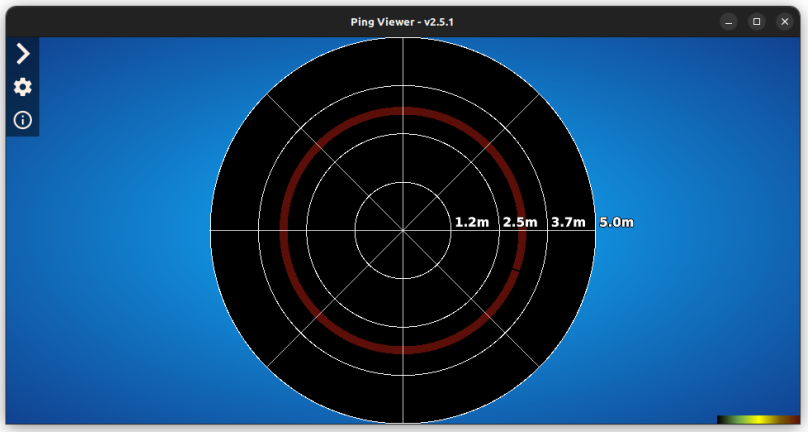


Figura 7.18: Situación previa al acercamiento de la piscina circular.

Las figuras 7.19 y 7.20, representan después de las maniobras de acercamiento. que cuando se presenta el problema introducido anteriormente.

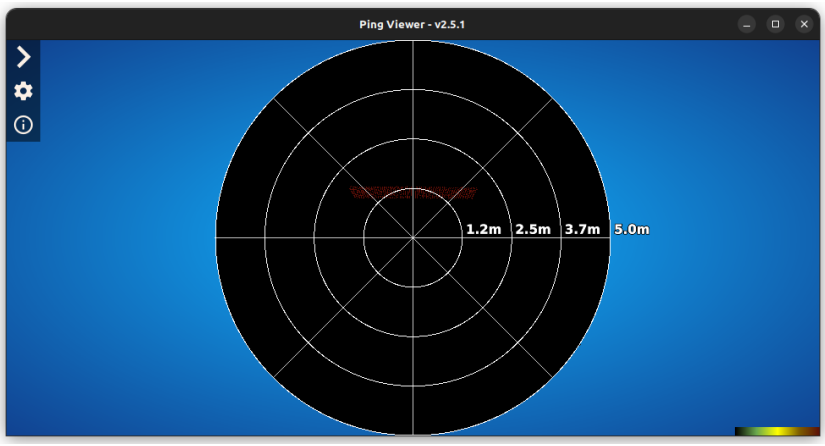


Figura 7.19: Situación posterior al acercamiento de la pared.

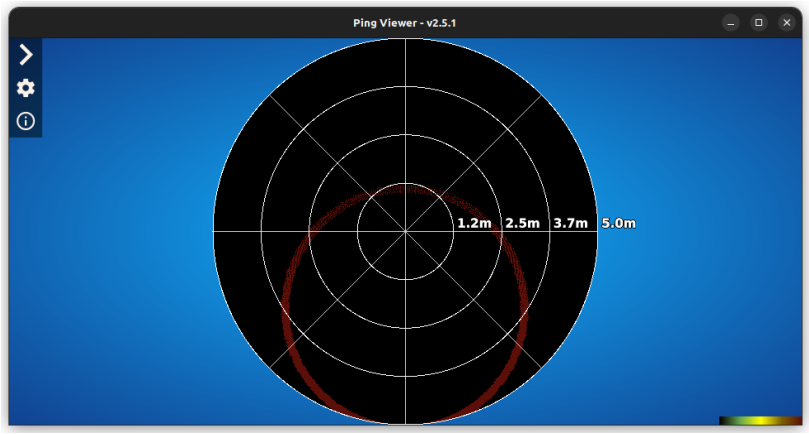


Figura 7.20: Situación posterior al acercamiento de la piscina circular.

Por último, en las figuras 7.21 y 7.22 se observa el resultado de haber vuelto a la posición inicial partiendo del conjunto de datos anteriormente modificado en el acercamiento.

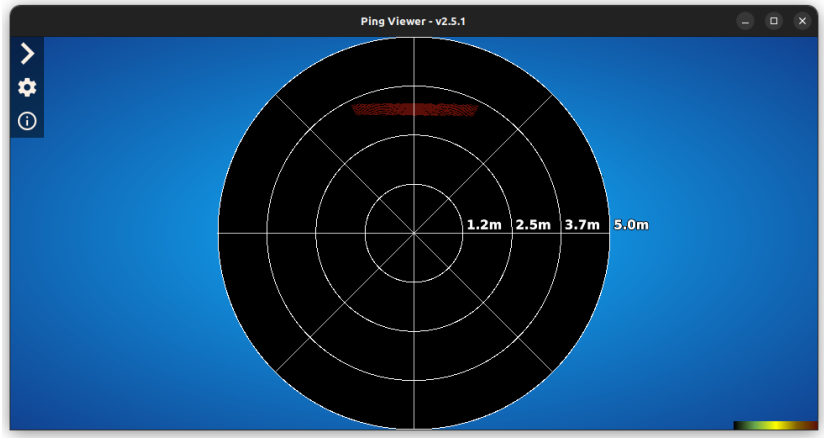


Figura 7.21: Situación posterior al alejamiento de la pared.

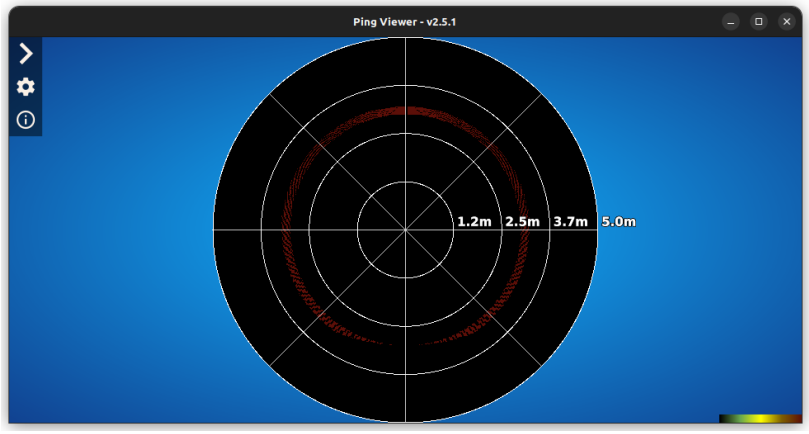


Figura 7.22: Situación posterior al alejamiento de la piscina circular.

El resultado de este problema y de los redondeos tras los cambios de coordenadas es que con cada actualización del entorno, si este supone un acercamiento, el conjunto de datos se va degradando cada vez más hasta, resultar prácticamente inservible.

8. Conclusiones

8.1. Conclusiones

El presente Trabajo de Fin de Grado ha tenido como objetivo dotar a un ROV submarino de herramientas que le permitan avanzar hacia la autonomía operativa en entornos subacuáticos.

Con este propósito, se han llevado a cabo las siguientes actividades:

- Integración de los sensores seleccionados: Ping Sonar y Ping 360.
- Desarrollo de dos simuladores capaces de replicar el comportamiento del Ping Sonar y del Ping 360.
- Pruebas de validación de los simuladores mediante clientes reales, tanto visuales (Ping Viewer) como programáticos (librería Ping-Python).
- Integración de los simuladores en scripts de control de ROV reales utilizando la librería Pymavlink.
- Simulaciones conjuntas de sonar y ROV virtual mediante SITL.
- Estudio del problema de aparición de brechas en los obstáculos durante desplazamientos.

La integración del sensor Ping Sonar se ha realizado con éxito, siendo suficiente el uso de los métodos de la clase *Ping1D* de la librería Ping-Python para su operación. En el caso del Ping 360, los métodos disponibles en la clase *Ping360* no permiten obtener directamente información sobre la presencia ni la distancia de obstáculos. Por tanto, se ha desarrollado un método alternativo que, tras su validación con datos reales y simulados, ha demostrado ser funcional.

La implementación del simulador de Ping Sonar se ha desarrollado sin incidencias, al igual que la función encargada de generar los conjuntos de datos que representan el fondo marino. Las pruebas realizadas con este simulador, tanto visuales como

funcionales, han arrojado resultados satisfactorios, evidenciando una correcta replicación del comportamiento del sensor.

Para el simulador de Ping 360, se han adaptado los scripts del Ping Sonar para generar los mensajes específicos de este sensor y se han desarrollado funciones que permiten simular entornos con obstáculos de pared y piscinas circulares. Las pruebas de este simulador también se han completado satisfactoriamente, evidenciando además el efecto conocido como *cross-range*, relacionado con la resolución angular del sonar.

Posteriormente, se integraron ambos simuladores en scripts de control de un ROV virtual mediante la librería Pymavlink. En el caso del Ping Sonar, la integración en el control de profundidad (*heave*) fue exitosa, permitiendo una simulación conjunta precisa mediante SITL.

En el caso del Ping 360, la integración fue satisfactoria para las maniobras de guiñada (*yaw*), pero no completamente para los desplazamientos longitudinales (*surge*). Durante estos movimientos, aunque el comportamiento espacial del ROV era coherente, se identificó un problema de brechas en la representación de obstáculos, asociado a errores de redondeo y al incremento de posiciones angulares necesarias para representar adecuadamente un objeto a distancias cortas.

En conclusión, la integración de los dispositivos y la implementación de sus simuladores ha sido globalmente satisfactoria. El simulador de Ping Sonar ha demostrado su funcionalidad tanto de forma aislada como integrada en el control de un ROV. En el caso del Ping 360, la integración también ha resultado adecuada para maniobras de yaw, aunque debe revisarse para movimientos de surge a fin de resolver el problema de representación continua de obstáculos.

Estos resultados evidencian el potencial del uso de simuladores de sonar en entornos virtuales como herramienta de validación y prueba para sistemas de navegación subacuática autónoma.

8.2. Líneas futuras

Los siguientes apartados recogen las principales líneas de continuidad del presente Trabajo de Fin de Grado. Se plantean como oportunidades para profundizar y enriquecer la labor de investigación y desarrollo realizada, al tiempo que ponen de relieve algunas de las limitaciones detectadas. A partir de las conclusiones alcanzadas, se proponen las siguientes recomendaciones:

- *Mejora del conjunto de datos del sonar*: Se propone implementar un segundo conjunto de datos de mayor resolución angular, mayor número de muestras y mayor rango que el actualmente configurado en el simulador. Tras cada desplazamiento, este conjunto de datos mejorado sería el que se modifique en función de los movimientos del ROV. Posteriormente, el conjunto de datos original se actualizaría a partir del conjunto mejorado antes de atender cualquier consulta. Esta estrategia permitiría mitigar el problema de aparición de brechas en los obstáculos y, además, posibilitaría que un obstáculo que hubiera salido del rango configurado pudiera volver a detectarse en caso de un nuevo acercamiento.
- *Actualización en segundo plano de los entornos (yaw y heave):* Se recomienda implementar un proceso paralelo (*thread*) encargado de realizar lecturas recurrentes de los valores de yaw y altitud del ROV. Esto permitiría actualizar continuamente el entorno del simulador, de forma independiente a los eventos de encendido de los motores. Dado que el movimiento del vehículo genera cambios en estos parámetros incluso después del apagado de los propulsores (debido a su inercia), este enfoque garantizaría una representación más precisa y actualizada del entorno simulado.

9. Bibliografía

[1] M. Stein, «How the Micro ROV Class Will Change the Maritime Sector: An Introductory Analysis on ROV,Big Data and AI», *Autonomous Vehicles - Applications and Perspectives*, jul. 2023, doi: 10.5772/INTECHOPEN.1002223.

[2] Y. Lou y N. Ahmed, «Underwater Communications and Networks», 2022, doi: 10.1007/978-3-030-86649-5.

[3] T. Le Xuan, T. Phan Anh, D. Tran Khanh, D. Nguyen, y T. Pham Xuan, «Communication and Control for Remotely Operated Underwater Vehicles», *Lecture Notes in Civil Engineering*, vol. 208, pp. 216-221, 2022, doi: 10.1007/978-981-16-7735-9_22.

[4] D. Centelles, A. Soriano, R. Marín, y P. J. Sanz, «Arquitectura para Teleoperación Inalámbrica con Realimentación Visual de ROVs basados en ArduSub», doi: 10.17979/spudc.9788497497565.0408.

[5] L. Paull, S. Saeedi, M. Seto, y H. Li, «AUV navigation and localization: A review», *IEEE Journal of Oceanic Engineering*, vol. 39, n.o 1, pp. 131-149, ene. 2014, doi: 10.1109/JOE.2013.2278891.

[6] J. Zhou, Y. Si, y Y. Chen, «A Review of Subsea AUV Technology», *Journal of Marine Science and Engineering 2023, Vol. 11, Page 1119*, vol. 11, n.o 6, p. 1119, may 2023, doi: 10.3390/JMSE11061119.

[7] J. S. Willners *et al.*, «From market-ready ROVs to low-cost AUVs», *Oceans Conference Record (IEEE)*, vol. 2021-September, 2021, doi: 10.23919/OCEANS44145.2021.9705798.

[8] A. M. Fuad *et al.*, «Implementation of the AQUALUNG: A new form of Autonomous Underwater Vehicle», *2022 IEEE/OES Autonomous*

Underwater Vehicles Symposium, AUV 2022, 2022, doi: 10.1109/AUV53081.2022.9965924.

[9] J. Choi y H. T. Choi, «Underwater vehicle localization using angular measurements of underwater acoustic sources», *2015 12th International Conference on Ubiquitous Robots and Ambient Intelligence, URAI 2015*, pp. 235-238, dic. 2015, doi: 10.1109/URAI.2015.7358944.

[10] T. Hansen y A. Birk, «An Open-Source Solution for Fast and Accurate Underwater Mapping with a Low-Cost Mechanical Scanning Sonar», *Proc IEEE Int Conf Robot Autom*, pp. 9968-9975, 2024, doi: 10.1109/ICRA57147.2024.10609976.

[11] S. Siregar, R. Febriansyah, M. I. Sani, y U. Ikbal, «Implementation of Depth Control using a Blue robotic Ping Sonar Altimeter and Echo sounder in Explorer Class ROV», *Proceedings of the International Conference on Electrical Engineering and Informatics*, vol. 2020-October, oct. 2020, doi: 10.1109/ICELTICS50595.2020.9315409.

[12] M. Aubard, A. Madureira, L. Madureira, y J. Pinto, «Real-Time Automatic Wall Detection and Localization based on Side Scan Sonar Images», *2022 IEEE/OES Autonomous Underwater Vehicles Symposium, AUV 2022*, 2022, doi: 10.1109/AUV53081.2022.9965813.

[13] A. J. Oliveira, B. M. Ferreira, R. Diamant, y N. A. Cruz, «Sonar-based Cable Detection for in-situ Calibration of Marine Sensors», *2022 IEEE/OES Autonomous Underwater Vehicles Symposium, AUV 2022*, 2022, doi: 10.1109/AUV53081.2022.9965846.

[14] L. Zacchini *et al.*, «Forward-Looking Sonar CNN-based Automatic Target Recognition: An experimental campaign with FeelHippo AUV», *2020 IEEE/OES Autonomous Underwater Vehicles Symposium, AUV 2020*, sep. 2020, doi: 10.1109/AUV50043.2020.9267902.

[15] S. Paik y S. Lee, «Preliminary Study of Binarization Method for Obstacle Detection in Underwater Sonar Image via Gabor Filter Parameter Design», *2020 IEEE/OES Autonomous Underwater Vehicles Symposium, AUV 2020*, sep. 2020, doi: 10.1109/AUV50043.2020.9267939.

[16] R. Cerqueira, T. Trocoli, G. Neves, L. Oliveira, S. Joyeux, y J. Albiez, «Custom Shader and 3 D Rendering for computationally efficient Sonar Simulation», 2016.

[17] M. Sung, Y. W. Song, y S. C. Yu, «Underwater Object Detection of AUV based on Sonar Simulator utilizing Noise Addition», *2022 IEEE/OES Autonomous Underwater Vehicles Symposium, AUV 2022*, 2022, doi: 10.1109/AUV53081.2022.9965853.

[18] United States. Department of the Air Force, *Glossary of Standardized Terms: Administrative Practices (Air Force Manual 11-1)*, 1a ed., vol. AFM 11-1. Washington, D.C.: U.S. Government Printing Office , 1961. Accedido: 3 de julio de 2025. [En línea]. Disponible en: https://www.google.es/books/edition/Administrative_Practices_Glossary_of_Sta/WAkoAQAAMAAJ?hl=es&gbpv=1&dq=Glossary+of+standardized+terms.+Administrative+practices&pg=PP5&printsec=frontcover

[19] R. F. Blake, «Submarine signaling: The protection of shipping by a wall of sound and other uses of the submarine telegraph oscillator», *Proceedings of the American Institute of Electrical Engineers*, vol. 33, n.o 10, pp. 1569-1581, dic. 2013, doi: 10.1109/PAIEE.1914.6661262.

[20] NOAA Photo Library, «Reginald Fessenden and his electric oscillator», Scientific American Supplement, No. 2071. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://oceanexplorer.noaa.gov/history/docs/media/lowering-500.jpg

[21] C. M. Chilowsky y M. P. Langevin, «Procédés et appareils pour la production de signaux sous-marins dirigés et pour la localisation à distance

d'obstacles sous-marins», 502 913, 4 de marzo de 1920 Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://ieeemilestones.ethw.org/w/images/0/00/Patents%2C_1916-17_Chilowsky_and_Langevin.pdf

[22] M. P. Langevin, «Procédé et appareils d'émission et de réception des ondes élastiques sous-marines à l'aide des propriétés piézo-électriques du quartz», 505 703, 17 de noviembre de 1918 Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://ieeemilestones.ethw.org/w/images/8/84/Brevet_505.703_1918.pdf

[23] P. Langevin y S. Cittornegs, «Piezoelectric signaling apparatus», vol. 248, p. 870, jun. 1920.

[24] J. M. Hovem, «Underwater acoustics: Propagation, devices and systems», *J Electroceram*, vol. 19, n.o 4, pp. 339-347, dic. 2007, doi: 10.1007/S10832-007-9059-9.

[25] R. D. Christ y R. L. Sr. Wernli, *The ROV Manual: A User Guide for Observation-Class Remotely Operated Vehicles*, 2nd ed. Oxford: Butterworth-Heinemann, 2013. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://www.sciencedirect.com/book/9780080982885/the-rov-manual

[26] Blue Robotics, «Typical scanning sonar color palettes», Understanding and Using Scanning Sonars. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://bluerobotics.com/wp-content/uploads/2019/10/Color-Palettes-R5.png

[27] Blue Robotics, «Typical fan shaped beam used on scanning sonars», Understanding and Using Scanning Sonars. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://bluerobotics.com/wp-content/uploads/2019/06/SS_Guide-Beam_Patterns-curved-R3.png

[28] Blue Robotics, «Target Visibility », Understanding and Using Scanning Sonars. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://bluerobotics.com/wp-content/uploads/2019/06/SS_Guide-Vertical-Coverage-R3.png

[29] Blue Robotics, «Vertical Arrival Angle and Slant Range », Understanding and Using Scanning Sonars. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://bluerobotics.com/wp-content/uploads/2019/06/SS_Guide-Slant_Range-side-R3.png

[30] Blue Robotics, «Vertical Arrival Angle and Slant Range Top View», Understanding and Using Scanning Sonars. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://bluerobotics.com/wp-content/uploads/2019/06/SS_Guide-Slant_Range-top-R3.png

[31] Blue Robotics, «Acoustic Shadows», Understanding and Using Scanning Sonars. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://bluerobotics.com/wp-content/uploads/2019/10/SS_Guide-Long-Shadow-r2.png

[32] Blue Robotics, «Echoes from Targets», Understanding and Using Scanning Sonars. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://bluerobotics.com/wp-content/uploads/2019/06/SS_Guide-Echoes-from-Targets.png

[33] C. Mai, S. Pedersen, L. Hansen, K. L. Jepsen, y Z. Yang, «Subsea infrastructure inspection: A review study», *USYS 2016 - 2016 IEEE 6th International Conference on Underwater System Technology: Theory and Applications*, pp. 71-76, abr. 2017, doi: 10.1109/USYS.2016.7893928.

[34] Blue Robotics, «Scanning imaging sonar beam shape», A Smooth Operator's Guide to Underwater Sonars and Acoustic Devices. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://bluerobotics.com/wp-content/uploads/2023/07/SCANNING-IMAGING-SONAR-VIEW-A-1.jpg

[35] Blue Robotics, «Single beam echosounder beam shape», A Smooth Operator's Guide to Underwater Sonars and Acoustic Devices. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://bluerobotics.com/wp-content/uploads/2023/07/SINGLE-BEAM-ECHOSOUNDER-1.jpg

[36] Blue Robotics, «Ping Sonar», Ping Sonar Altimeter and Echosounder. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://bluerobotics.com/wp-content/uploads/2019/01/PING_BR-100725_RevA_PUBLIC.png

[37] «Ping Protocol». Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://docs.bluerobotics.com/ping-protocol/

[38] «ping1d - Ping Protocol». Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://docs.bluerobotics.com/ping-protocol/pingmessage-ping1d/

[39] «Ping Sonar Technical Manual». Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://bluerobotics.com/learn/ping-sonar-technical-guide/

[40] Blue Robotics, «Ping internal logic diagram», Ping Sonar Technical Manual. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://bluerobotics.com/wp-content/uploads/2019/09/Ping-EchoSounder-Device-Behavior-1024x341.png

[41] «Download Python | Python.org». Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://www.python.org/downloads/

[42] Blue Robotics, «GitHub - bluerobotics/ping-python: Python scripts and examples for the Ping sonar.», bluerobotics/ping-python. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://github.com/bluerobotics/ping-python

[43] Blue Robotics, «Ping360 Sonar», Ping360 Scanning Imaging Sonar. Accedido: 2 de julio de 2025. [En línea]. Disponible en:

https://bluerobotics.com/wp-content/uploads/2019/09/PING360-SONAR-R2-2D-DRAWING-2.png

[44] Blue Robotics, «ping360 - Ping Protocol», Ping Protocol. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://docs.bluerobotics.com/ping-protocol/pingmessage-ping360/

[45] Blue Robotics, «Ping Viewer Documentation», Ping Viewer. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://docs.bluerobotics.com/ping-viewer/

[46] Blue Robotics, «Ping Viewer Interface Ping1D». Accedido: 4 de julio de 2025. [En línea]. Disponible en: https://bluerobotics.com/wp-content/uploads/2019/01/ping-viewer-1.jpg

[47] Blue Robotics, «Ping Viewer Interface Ping360». Accedido: 4 de julio de 2025. [En línea]. Disponible en: https://bluerobotics.com/wp-content/uploads/2022/11/Ping360_Pilings.png

[48] Blue Robotics, «ping1d-simulation.py», bluerobotics / ping-python. Accedido: 2 de julio de 2025. [En línea]. Disponible en: https://github.com/bluerobotics/ping-python/blob/master/tools/ping1d-simulation.py

10. Presupuesto

En este capítulo se presenta el presupuesto detallado del Trabajo Fin de Grado (TFG). El objetivo es cuantificar los costes asociados a:

- Recursos materiales (software y hardware).
- Trabajo propio remunerado por horas dedicadas.
- Impuestos aplicables y cálculo del coste total.

10.1. Recursos materiales

Los recursos materiales se han clasificado en software, hardware inventariable (sujeto a amortización) y hardware fungible (gasto directo). Para la amortización se emplea el método lineal, que reparte de forma uniforme la pérdida de valor de un activo a lo largo de toda su vida útil.

Así, el coste de amortización C_a de cada elemento se determina mediante la expresión (10.1):

$$C_a = \frac{V_{adq} - V_{res}}{V_u} \tag{10.1}$$

donde V_{adq} representa el precio de adquisición, V_{res} el valor residual (fijado en el 10 % de V_{adq}) y V_u los años de vida útil del recurso.

10.1.1.Recursos software

Los recursos software implicados en el desarrollo de este trabajo son los que se detallan junto a sus costes en la tabla 10.1. Para el estudio se considera que todos los recursos software tienen una vida útil de 5 años.

Tabla 10.1: Costes de amortización de software.

Software	Valor adquisición (€)	Amortización anual (€)	Tiempo de uso	Coste imputado (€)
SITL	0,00	0,00	3 meses	0,00
Visual Studio Code	0,00	0,00	3 meses	0,00
Canva	420	75,6	3 meses	18,9
Ping Viewer	0,00	0,00	3 meses	0,00
Ardusub	0,00	0,00	3 meses	0,00
Pymavlink	0,00	0,00	3 meses	0,00
Ping-Python	0,00	0,00	3 meses	0,00
Microsoft Excel	0,00	0,00	3 meses	0,00
Microsoft Word	0,00	0,00	3 meses	0,00
		Coste amortización software:		**18,90**

10.1.2.Recursos hardware

Los recursos hardware y sus costes se especifican en la tabla 10.2 para aquellos de tipo fungible y en la tabla 10.3 para los inventariables. Para el estudio se considera que todos los recursos materiales tienen una vida útil de 5 años.

Tabla 10.2: Costes de materiales hardware fungible.

Material	Coste imputado (€)
Cable de conexión de alta potencia	1.867,26
Fathom-X Tether Interface (FXTI)	195,21

Cargador H6 PRO	148,51
Mando de Xbox	19,79
Conectores, electrónica básica y tornillería	68,00
Coste hardware fungible:	**2.298,77**

Tabla 10.3: Costes de amortización hardware inventariable.

Hardware	Valor adquisición (€)	Amortización anual (€)	Tiempo de uso	Coste imputado (€)
ROV BlueROV2	3.904,25	702,77	3 meses	175,7
Sonar Ping Sonar	364,96	65,7	3 meses	16,42
Sonar Ping 360	2.334,12	420,14	3 meses	105,04
		Coste amortización hardware inventariable:		**297,16**

10.2. Trabajo tarifado por tiempo empleado

Para valorar el tiempo dedicado por el autor se toma como referencia la tabla de retribuciones del personal técnico publicada en el BOULPGC el 9 de julio 2024 (figura 10.1).

Ya que el TFG tiene una duración estimada de 300 horas, a repartir en 15 semanas, la dedicación del autor es de 20 horas a la semana. Enmarcándose en la categoría de *TÉCNICO MECES 2* a tiempo parcial.

	TIPO PERSONAL	TITULACIÓN MÍNIMA EXIGIDA	CATEGORÍA	DEDICACION (horas semanales)	RETRIBUCIÓN MENSUAL	SEGURIDAD SOCIAL MENSUAL	SEGURIDAD SOCIAL ANUAL	RETRIBUCIÓN BRUTA ANUAL	COSTE TOTAL ANUAL (H+I+L)	COSTE MENSUAL
PERSONAL INVESTIGADOR	INVESTIGADOR	Master o equivalente (MECES 3)	ICP2 (MECES 3)	TC 37,5	1.886,85	612,81	7.353,72	22.642,16	29.995,88	2.499,66
				TP 20	1.006,31	435,07	5.220,84	12.075,74	17.296,58	1.441,38
	INVESTIGADOR DOCTOR	Doctor (MECES 4)	ICP1 (DOCTOR)	TC 37,5	2.156,50	700,31	8.403,72	25.878,05	34.281,77	2.856,81
				TP 20	1.150,13	435,07	5.220,84	13.801,58	19.022,42	1.585,20
PERSONAL TÉCNICO Y DE APOYO	PERSONAL DE APOYO	Técnico superior FP o equivalente (MECES 1)	PACP3 (MECES 1)	TC 37,5	1.323,00	483,00	5.796,00	15.876,00	21.672,00	1.806,00
				TP 20	705,60	360,78	4.329,36	8.467,20	12.796,56	1.066,38
		Grado o equivalente (MECES 2)	PACP2 (MECES 2)	TC 37,5	1.344,50	582,39	6.988,68	16.134,03	23.122,71	1.926,89
				TP 20	717,08	435,07	5.220,84	8.604,96	13.825,80	1.152,15
		Master o equivalente (MECES 3)	PACP1 (MECES 3)	TC 37,5	1.886,85	612,81	7.353,72	22.642,16	29.995,88	2.499,66
				TP 20	1.006,31	435,07	5.220,84	12.075,74	17.296,58	1.441,38
	TÉCNICO	Técnico superior FP o equivalente (MECES 1)	TCP5 (MECES 1)	TC 37,5	1.323,00	483,00	5.796,00	15.876,00	21.672,00	1.806,00
				TP 20	705,60	360,78	4.329,36	8.467,20	12.796,56	1.066,38
		Grado o equivalente (MECES 2)	TCP4 (MECES 2)	TC 37,5	1.425,12	582,39	6.988,68	17.101,48	24.090,16	2.007,51
				TP 20	760,05	435,07	5.220,84	9.120,64	14.341,48	1.195,12
		Master o equivalente (MECES 3)	TCP3 (MECES 3)	TC 37,5	1.633,21	582,39	6.988,68	19.598,57	26.587,25	2.215,60
				TP 20	871,04	435,07	5.220,84	10.452,47	15.673,31	1.306,11

Figura 10.1: Contrataciones de personal técnico e investigador de la ULPGC.

En la tabla 10.4 se especifica el sueldo bruto que debería cobrar el autor de este trabajo para un régimen de dedicación de 20 horas semanales.

Tabla 10.4: Trabajo tarifado por tiempo empleado.

Categoría	Retribución mensual	Semanas	Total (€)
Grado (Meces 2)	760,05 €	15 semanas	2.660,18 €

Por tanto, el salario que percibiría el autor de este TFG es de **2.660,18 €**

10.3. Aplicación de impuestos y coste final

En la Comunidad Autónoma de Canarias se aplica el I.G.I.C. (Impuesto General Indirecto Canario) del 7 % a la compra de bienes y servicios. Teniendo en cuenta que los salarios no están sujetos a I.G.I.C., en la tabla 10.5 se detallan los costes totales con el impuesto y sin el mismo.

Tabla 10.5: Costes totales del Trabajo de Fin de Grado.

Concepto	Coste
Costes amortización software	18,90 €
Costes hardware fungible	2.298,77 €
Costes amortización hardware inventariable	297,16 €
Trabajo tarifado por tiempo empleado	2.660,18 €
Costes totales sin I.G.I.C.	**5.275,01 €**
Costes totales con I.G.I.C.	**5.458,05 €**

El coste final de la realización de este Trabajo de Fin de Grado titulado *Integración de sensores acústicos en un robot submarino* es de **5.458,05 €.**

11. Anexo

11.1. Script cliente_ping1d.py

```
# Permite ejecutar el script con `./archivo.py` usando Python 3
#!/usr/bin/env python3

# Librería del sonar Ping 1D
from brping import Ping1D
import time

if __name__ == "__main__":
    # Crea el objeto sonar
    p = Ping1D()

    # Conexión UDP en todas las interfaces, puerto 6675
    p.connect_udp("0.0.0.0", 6675)

    # Inicializa el sonar e imprime el estado devuelto
    print("Initialized: %s\n" % p.initialize())

    while True:
        # Solicita una medida de distancia básica
        result = p.get_distance_simple()

        # Convierte la distancia de mm a metros
        distance = int(result["distance"]) / 1000
        confidence = result["confidence"]

        # Muestra distancia y porcentaje de certeza
        print(f"Distancia: {distance:.3f} m - Certeza: {confidence} %")

        # Espera 1 s antes de la siguiente lectura
        time.sleep(1)
```

11.2. Script cliente_ping360.py

```
# Permite ejecutar directamente el fichero con Python 3
#!/usr/bin/env python3

# Importa la clase Ping360 (sonar de 360 °) y las definiciones de mensajes
from brping import Ping360, definitions
# Módulo propio con rutinas de filtrado y detección de picos
import filtrado

# Diccionario con todos los tipos de 'payload' definidos por la librería
payload_dict = definitions.payload_dict_all

# Estructura donde se guardarán los parámetros de configuración del sonar
data_dic = {}

if __name__ == "__main__":
    import argparse               # Preparado para añadir CLI más adelante
(actualmente sin uso)

    # Crea la instancia del sonar
    sonar = Ping360()
    # Conecta vía UDP en la IP local (todas las interfaces) y puerto 6675 — modo
depuración
    sonar.connect_udp("0.0.0.0", 6675)

    # Inicializa el dispositivo y muestra el resultado
    print("Initialized: %s \n" % sonar.initialize())

    # Lee una vez la configuración del sonar (p. ej. 'sample_period')
    sample_period = sonar.get_device_data()["sample_period"]

    # Bucle de escaneo continuo en 400 pasos (0 - 399 grados)
    while True:
        for angle in range(0, 400):
            # Envía un ping al ángulo actual y recibe un             mensaje
device_data
            device_data_message = sonar.transmitAngle(angle)
            data_array = device_data_message.data

            # Pasa los bytes/enteros de intensidades a lista Python
            intensity_list = list(data_array)

            # Calcula el incremento espacial de cada muestra (metros)
            sample_step = (1500 * sample_period * 25e-9) / 2
            # Filtra la señal y localiza la posición del eco de mayor energía
            peak_pos = filtrado.max_filtered(intensity_list, sample_step)

            if peak_pos:
                # Convierte la posición del pico a distancia real (m)
                distance = peak_pos * sample_step
                # Informa ángulo y distancia del obstáculo detectado
                print("%i GRAD: %f m" % (angle, distance))
            else:
                # Sin detección fiable en este ángulo
                print("%i GRAD: No se ha detectado obstaculo." % angle)
```

11.3. Script entornos_ping1d.py

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

############### GENERADOR DE FONDOS ###############

def fondo_plano(sim, distancia_fondo):
    #Guardamos el nuevo valor de distancia en el campo de distancia del simulador
    sim._distance = int(distancia_fondo*1000)
    profile_data = []
    #Calculamos el índice a partir el cual debe encontrarse el fondo en el
conjunto de datos
    index_pared = distancia_fondo / (sim._sample_jump)
    #Iteramos el conjunto de datos hasta que demos con el índice que nos interesa
    for index in range(sim._profile_data_length):
        if index_pared <= index <= index_pared + int(0.1 / sim._sample_jump):
            #Añadimos el valor de intensidad más alto en los índides
correspondientes
            profile_data.append(255)
        else:
            profile_data.append(0)
    return bytearray(profile_data)
```

11.4. Script entornos_ping360.py

```
#!/usr/bin/env python3
import numpy as np
import math

############### FUNCIONES DE DESPLAZAMIENTO ###############

def giro(sim, diff_yaw = 0):
    new_sonar_data_polar = {}

    if diff_yaw != 0:
        for angle,array in sim._sonar_data.items():
            #Yaw positivo supone virar a estribor por lo que el entorno gira en
sentido antihorario
            #Yaw negativo supone virar a babor por lo que el entorno gira en
sentido horario
            nuevo_angulo = normalizar_gradianes(angle - round(diff_yaw))
            new_sonar_data_polar[nuevo_angulo] = array

        return new_sonar_data_polar

def desplazamiento(sim, desplazamiento_x=0, desplazamiento_y=0):
    new_sonar_data = {clave: [0] * len(valor) for clave, valor in
sim._sonar_data.items()}

    if desplazamiento_x != 0 or desplazamiento_y != 0:
        for angle, array in sim._sonar_data.items():
            for index, value in enumerate(array):
                # Desplazamos solo los valores que sean 255 para evitar que los
valores que sean 0 los machaquen
                if value == 255:
                    # Convertimos a distancia el índice
                    distance = index * sim._sample_jump
                    # Convertir coordenadas polares a cartesianas
                    x, y = polares_a_cartesianas(distance, angle)

                    # Aplicar el desplazamiento
                    nueva_x = x + desplazamiento_x
                    nueva_y = y + desplazamiento_y

                    # Convertir de vuelta a polares
                    new_distance, theta = cartesianas_a_polares(nueva_x, nueva_y)
                    r = new_distance / sim._sample_jump
                    r = round(r)

                    # Validar los índices radial y angular
                    if 0 <= r < sim._data_length:
                        theta = round(theta) % 400  # Asegurar índice angular
válido

                        # Asegurar que no haya claves repetidas (sin sobrescribir
valores)
                        if new_sonar_data[theta][r] == 0:  # Solo asignar si no
ha sido asignado antes
                            new_sonar_data[theta][r] = value

    return new_sonar_data


############### GENERADORES DE OBSTACULOS ###############

def pared(sim, distancia_pared, angulo_pared=0):
    number_of_samples = sim._number_of_samples
    sample_jump = sim._sample_jump
```

```
    # Convertir el ángulo a radianes
    alpha = np.radians(angulo_pared * 0.9)  # Ángulo de la pared respecto al
sonar

    # Crear un diccionario para almacenar los valores para cada ángulo
    sonar_data = {}

    # Rango de ángulos en el que se aplicará la pared, con ajuste circular
    angle_range_min = (angulo_pared - 50) % 400
    angle_range_max = (angulo_pared + 50) % 400

    # Recorrer los ángulos de 0 a 400 gradianes
    for angle in range(400):
        data = []
        # Convertir el ángulo a radianes
        theta = np.radians(angle * 0.9)

        # Calcular r usando la ecuación de la recta
        r_value = distancia_pared / np.cos(theta - alpha)

        # Calcular el índice correspondiente a r_value
        n_value = int(r_value / sample_jump)

        # Verificar si el ángulo está dentro del rango, considerando el cambio
0/400
        if angle_range_min < angle_range_max:
            # Rango normal
            in_angle_range = angle_range_min <= angle <= angle_range_max
        else:
            # Rango que cruza el límite de 400 grados
            in_angle_range = angle >= angle_range_min or angle <= angle_range_max

        # Llenar los datos para cada muestra `n`
        for n in range(number_of_samples + 1):
            if n_value <= n <= n_value + int(0.1 / sample_jump) and
in_angle_range:
                data.append(255)
            else:
                data.append(0)

        # Guardar el resultado en el diccionario, usando el ángulo como clave
        sonar_data[angle] = data

    return sonar_data

def pared_con_tamaño(sim,tamaño_pared,distancia_pared,angulo_pared_gradianes =
0):

    number_of_samples = sim._number_of_samples
    sample_jump = sim._sample_jump

    # Ángulo de la pared respecto al sonar
    angulo_pared_grados = angulo_pared_gradianes * 0.9
    # Convertir el ángulo a radianes
    angulo_pared_radianes = np.radians(angulo_pared_grados)

    # Crear un diccionario paraS almacenar los valores para cada ángulo
    sonar_data = {}
    # Rango de angulos en radianes a cubir para conseguir el tamaño de la pared
    angulo_tamaño_radianes = np.arctan((tamaño_pared/2)/distancia_pared)
    angulo_tamaño_gradianes = angulo_tamaño_radianes*(200/math.pi) # Pasamos a
gradianes para hacer el barrido

    # Rango de ángulos en el que se generará la pared, con ajuste circular
    angle_range_min = (angulo_pared_gradianes - angulo_tamaño_gradianes) % 400
    angle_range_max = (angulo_pared_gradianes + angulo_tamaño_gradianes) % 400

    # Recorrer los ángulos de 0 a 400 gradianes
```

```
    for angle_gradianes in range(400):
        data = []
        # Convertir el ángulo a radianes
        angle_grados = angle_gradianes * 0.9
        angle_radianes = angle_grados * (np.pi/180)

        # Calcular r usando la ecuación de la recta
        r_value = distancia_pared / np.cos(angle_radianes -
angulo_pared_radianes)

        # Calcular el índice correspondiente a r_value
        n_value_float = r_value / sample_jump
        n_value_int = int(n_value_float)

        # Vemos si estamos en caso de salto de 400-0
        if angle_range_min < angle_range_max:
            # Caso normal
            in_angle_range = angle_range_min <= angle_gradianes <=
angle_range_max
        else:
            # Caso que cruza el límite de 400 grados
            in_angle_range = angle_gradianes >= angle_range_min or
angle_gradianes <= angle_range_max

        # Llenar los datos para cada muestra `n`
        for n in range(number_of_samples):
            if n_value_int <= n <= n_value_int + int(0.3 / sample_jump) and
in_angle_range:
            # if n_value_int == n and in_angle_range:
                data.append(255)
            else:
                data.append(0)

        # Guardar el resultado en el diccionario, usando el ángulo como clave
        sonar_data[angle_gradianes] = data

    return sonar_data

def piscina(sim,distancia_borde):

    number_of_samples = sim._number_of_samples
    sample_jump = sim._sample_jump

    # Crear un diccionario para almacenar los valores de r para cada ángulo
    sonar_data = {}

    # Recorrer los ángulos de 0 a 400 gradianes
    for angle in range(400):
        data = []
        #Calculo el indice correspondiente a r_value
        n_value = int(distancia_borde / sample_jump)
        for n in range(number_of_samples+1):
            if n_value <= n <= n_value + (0.2 /sample_jump):
                data.append(255)
            else:
                data.append(0)

        # Guardar el resultado en el diccionario, usando el ángulo como clave
        sonar_data[angle] = data

    return sonar_data


############### FUNCIONES DE CONVERSIÓN ###############

def normalizar_gradianes(theta_grad):
    """
    Normaliza un ángulo en gradianes para que esté en el rango [0, 400).
```

```
    :param theta_grad: ángulo en gradianes
    :return: ángulo normalizado en el rango [0, 400)
    """
    return theta_grad % 400

def polares_a_cartesianas(r, theta_grad):
    """
    Convierte coordenadas polares (r, theta) en gradianes a cartesianas (x, y).
    :param r: radio en coordenadas polares
    :param theta_grad: ángulo en gradianes
    :return: coordenadas cartesianas (x, y) con todos los decimales.
    """
    theta_rad = theta_grad * math.pi / 200  # Conversión de gradianes a radianes
    # OJO Las funciones representan angulos en sentido horario
    x = r * math.sin(theta_rad)
    y = r * math.cos(theta_rad)
    return x, y

def cartesianas_a_polares(x, y):
    """
    Convierte coordenadas cartesianas (x, y) a polares (r, theta) en gradianes.
    :param x: coordenada x en cartesianas
    :param y: coordenada y en cartesianas
    :return: coordenadas polares (r, theta en gradianes)
    """
    r = math.sqrt(x**2 + y**2)
    # OJO Las funciones representan angulos en sentido horario
    theta_rad = math.atan2(x, y)
    # Como la funcion atan2 devuelve valores entre [0,pi] y  [0,-pi]:
    if theta_rad < 0:
        theta_rad = theta_rad + 2*math.pi
    theta_grad = theta_rad * 200 / math.pi  # Conversión de radianes a gradianes
    return r, theta_grad
```

11.5. Script simulador_ping1d.py

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

# This script simulates a Blue Robotics Ping Echosounder device
# A client may connect to the device simulation on local UDP port 6676

from brping import definitions, PingMessage, PingParser
import entornos_ping1d
import filtrado

import socket
import time
import errno
import numpy as np

payload_dict = definitions.payload_dict_all

class Ping1DSimulation(object):
    def __init__(self):
        self.client = None # (ip address, port) of connected client (if any)
        self.parser = PingParser() # used to parse incoming client comunications

        # Socket to serve on
        self.sockit = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sockit.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.sockit.setblocking(True)
        #  Elegir puerto 6676 para conectar con PingViewer y 6675 para conectar
con cliente ROV
        self.sockit.bind(('0.0.0.0', 6675))

        self._mode_auto = True # automatic gain and range selection 0: manual
mode, 1: auto mode
        self._mode_continuous = True # automatic continuous output of profile
messages NO sirve para nada

        self._device_type = 1 # device type
        self._device_model = 1 # device model
        self._device_revision = 1 # device revision

        # Set of the device_information (4) - Common messages - fields

        #Ping1D (object) pide los atributos SIN "_firmware + [...]"
        self._version_major = 1
        self._version_minor = 1
        self._version_patch = 1

        #Pingviewer pide los atributos CON "_firmware + [...]"
        self._firmware_version_major = 1
        self._firmware_version_minor = 1
        self._firmware_version_patch = 1

        self._reserved = 0

        self._device_id = 0 #(0-255) 255 reservado broadcast

        self._voltage_5 = 5000 #5V rail voltage (mV)

        self._speed_of_sound = 1500000 # en mm/s

        self._distance = 5000 #The current return distance determined for the
most recent acoustic measurement. (mm)
        self._confidence = 80 #Confidence in the most recent range measurement.
(%)
        self._transmit_duration = 100 #Acoustic duration in microseconds (us)
        self._ping_number = 0 # count the total measurements taken since boot
```

```
        self._scan_start = 0 #The beginning of the scan range in mm from the
transducer.(mm)
        self._scan_length = 50000 #The length of the scan range. Minimum 1000mm
        self._gain_setting = 0 #The current gain setting. 0: 0.6, 1: 1.8, 2: 5.5,
3: 12.9, 4: 30.2, 5: 66.1, 6: 144
        self._profile_data_length = 200 # number of data points in profile
messages (constant for now)
        self._profile_data = bytearray(np.random.randint(0, 256, 200)) #An array
of return strength measurements taken at regular intervals across the scan
region. The first element is the closest measurement to the sensor, and the last
element is the farthest measurement in the scanned range.

        self._processor_temperature = 0 #The temperature in centi-degrees
Centigrade (100 * degrees C).
        self._pcb_temperature = 0 #The temperature in centi-degrees Centigrade
(100 * degrees C).

        self._ping_enabled = 0 #The state of the acoustic output. 0: disabled,
1:enabled


        #self._pulse_duration = 100 # length of acoustic pulse (constant for now)
        self._ping_interval = 100 # milliseconds between measurements The minimum
interval between acoustic measurements. The actual interval may be longer.
        self._sample_period = 0
        self._sample_jump = 0
        self._superficie_fondo = 0
        self._range = 0


    # read incoming client data
    def read(self):
        try:
            data, self.client = self.sockit.recvfrom(4096)

            # digest data coming in from client
            for byte in data:
                if self.parser.parse_byte(byte) == PingParser.NEW_MESSAGE:
                    # we decoded a message from the client
                    self.handleMessage(self.parser.rx_msg)

        except EnvironmentError as e:
            if e.errno == errno.EAGAIN:
                pass # waiting for data
            else:
                print("Error reading data", e)

        except KeyError as e:
            print("skipping unrecognized message id: %d" %
self.parser.rx_msg.message_id)
            print("contents: %s" % self.parser.rx_msg.msg_data)
            pass

    # write data to client
    def write(self, data):
        if self.client is not None:
            self.sockit.sendto(data, self.client)

    # Send a message to the client, the message fields are populated by the
    # attributes of this object (either variable or method) with names matching
    # the message field names
    def sendMessage(self, message_id):
        msg = PingMessage(message_id)
        print("sending message %d\t(%s)" % (msg.message_id, msg.name))

        # pull attributes of this class into the message fields (they are named
the same)
        for attr in payload_dict[message_id]["field_names"]:
```

```
            try:
                # see if we have a function for this attribute (dynamic data)
                # if so, call it and put the result in the message field
                setattr(msg, attr, getattr(self, attr)())
            except AttributeError as e:
                try:
                    # if we don't have a function for this attribute, check for a
_<field_name> member
                    # these are static values (or no function implemented yet)
                    #print(attr + " : " + str(getattr(self, "_" + attr)))
                    setattr(msg, attr, getattr(self, "_" + attr))
                except AttributeError as e:
                    # anything else we haven't implemented yet, just send a sine
wave
                    setattr(msg, attr, self.periodicFnInt(20, 120))

        # send the message to the client
        msg.pack_msg_data()
        self.write(msg.msg_data)

    def set_segment_range(self, start,stop): #Pendiente de entender como muestrea
em el segmento
        self._scan_start = start
        self._scan_length = stop-start

    def update_sample_jump_period(self):
        self._sample_jump = ((self._scan_length - self._scan_start) /
self._profile_data_length)/1000 # m
        self._sample_period =
(self._sample_jump/(self._speed_of_sound/1000))*1000 #sample-jump en m;
speed_of_sound en mm/ -> m/s; sample_period en s -> ms

    # handle an incoming client message
    def handleMessage(self, message):
        print("receive message %d\t(%s)" % (message.message_id, message.name))
        if message.message_id == definitions.COMMON_GENERAL_REQUEST:
            # the client is requesting a message from us
            self.sendMessage(message.requested_id)
        # hack for legacy requests
        elif message.payload_length == 0:
            self.sendMessage(message.message_id)
        else:
            # the client is controlling some parameter of the device
            self.setParameters(message)

    # Extract message fields into attribute values
    # This should only be performed with the 'set' category of messages
    # TODO: mechanism to filter by "set"
    def setParameters(self, message):
        for attr in payload_dict[message.message_id]["field_names"]:
            setattr(self, "_" + attr, getattr(message, attr))

    def print_distance_confidence(self):
        pos_max = filtrado.max_filtered(self._profile_data,self._sample_jump)
        if pos_max:
            self._distance = int((pos_max * self._sample_jump) * 1000)
            print("Distancia: %f m - Certeza: %d %%" % (self._distance / 1000,
self._confidence))

    def bucle_ping1D(self):

        # Last measurement time
        lastUpdate = 0

        self.update_sample_jump_period()
        self._profile_data = bytearray(entornos_ping1d.fondo_plano(self,7))

        while True:
```

```
            last_range = self._scan_length - self._scan_start
            # read any incoming client communications
            self.read()
            print(" ")

            #Si cambia el rango actualizamos el salto de muestras y generamos el
entorno actualizado
            if last_range != (self._scan_length - self._scan_start):
                self.update_sample_jump_period()

            # COMENTAR CUANDO SE USE  CLIENT_PING1D.PY
            # # Update background ping count and continuous output
            # if time.time() > lastUpdate + self._ping_interval / 1000.0:
            #     lastUpdate = time.time()
            #     self._ping_number += 1
            #     if self._mode_continuous:
            #         self.sendMessage(definitions.PING1D_PROFILE)

            #         self.print_distance_confidence()
            #don't max cpu
            time.sleep(0.01)


################ COMENTAR CUANDO SE USEN HILOS EN CODIGO DE CONTROL
################################
#The simulation to use
# sim = Ping1DSimulation()
# sim.bucle_ping1D()
```

11.6. Script simulador_ping360.py

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

# This script simulates a Blue Robotics Ping 360  device
# A client may connect to the device simulation on local UDP port 6676

from brping import definitions, PingMessage, PingParser
import filtrado
import entornos_ping360
import socket
import time
import errno
import sys

payload_dict = definitions.payload_dict_all

class Ping360Simulation(object):
    def __init__(self):
        self.client = None # (ip address, port) of connected client (if any)
        self.parser = PingParser() # used to parse incoming client comunications

        # Socket to serve on
        self.sockit = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.sockit.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
        self.sockit.setblocking(True)
        self.sockit.bind(('0.0.0.0', 6676))

        self._mode_auto = False # automatic gain and range selection
        self._mode_continuous = True # automatic continuous output of profile
messages
        self._speed_of_sound = 1500 # default 1500m/s

        #Set of the ping360 device_data(2300) - Ping360 messages - information
        self._mode = 1
        #gain_setting_match
        self._gain_setting = 1 # default 1 (normal)
        self._angle = 0
        #transmit_duration_match
        self._transmit_duration = 11
        #sample_period_match
        self._sample_period = 88
        #trasmit_frequency_match
        self._transmit_frequency = 750 #default 750Hz
        #number_of_samples_match
        self._number_of_samples = 1200
        self._data_length = 1200

        self._range = None
        self._sample_jump = None
        self._sonar_data = {}

        # Set of the device_information (4) - Common messages - fields
        #Ping360 (object) pide los atributos SIN "_firmware + [...]"
        self._device_revision = 0
        self._device_type = 2
        self._version_major = 1
        self._version_minor = 1
        self._version_patch = 1

        #Pingviewer pide los atributos CON "_firmware + [...]"
        self._firmware_version_major = 1
        self._firmware_version_minor = 1
        self._firmware_version_patch = 1

        self._reserved = 1
```

```
        # Set of the transmit field used in transducer (2601) - Ping360 messages
- for communication
        self._transmit = 0

        # # Set of the sample jump and range bnased in the configuration
parameters in the file
        # self._sample_jump =
rango.sample_jump(self._sample_period,self._speed_of_sound)
        # self._range = self._sample_jump * self._number_of_samples

    def print_config(self):
        print("\n")
        print("VALORES DE CONFIGURACION:")
        print("Mode:", self._mode)
        print("Gain Setting:", self._gain_setting)
        print("Transmit Duration:", self._transmit_duration)
        print("Sample Period:", self._sample_period)
        print("Transmit Frequency:", self._transmit_frequency)
        print("Number of Samples:", self._number_of_samples)
        print("Speed of Sound:", self._speed_of_sound)
        print("Range:", self._range)
        print("Sample Jump: ", self._sample_jump)
        print("\n")
        return

    # read incoming client data
    def read(self):
        try:
            data, self.client = self.sockit.recvfrom(4096)

            # digest data coming in from client
            for byte in data:
                if self.parser.parse_byte(byte) == PingParser.NEW_MESSAGE:
                    # we decoded a message from the client
                    self.handleMessage(self.parser.rx_msg)

        except EnvironmentError as e:
            if e.errno == errno.EAGAIN:
                pass # waiting for data
            else:
                print("Error reading data", e)

        except KeyError as e:
            print("skipping unrecognized message id: %d" %
self.parser.rx_msg.message_id)
            print("contents: %s" % self.parser.rx_msg.msg_data)
            pass

    # write data to client
    def write(self, data):
        if self.client is not None:
            self.sockit.sendto(data, self.client)

    # Send a message to the client, the message fields are populated by the
    # attributes of this object (either variable or method) with names matching
    # the message field names
    def sendMessage(self, message_id):
        self._data = bytearray(self._sonar_data[self._angle])
        msg = PingMessage(message_id)
        print("sending message %d\t(%s)" % (msg.message_id, msg.name))

        # pull attributes of this class into the message fields (they are named
the same)
        for attr in payload_dict[message_id]["field_names"]:
            try:
                # see if we have a function for this attribute (dynamic data)
```

```
                # if so, call it and put the result in the message field
                setattr(msg, attr, getattr(self, attr)())
            except AttributeError as e:
                try:
                    # if we don't have a function for this attribute, check for a
_<field_name> member
                    # these are static values (or no function implemented yet)
                    #print(attr + " : " + str(getattr(self, "_" + attr)))
                    setattr(msg, attr, getattr(self, "_" + attr))
                except AttributeError as e:
                    # anything else we haven't implemented yet, just send a sine
wave
                    setattr(msg, attr, self.periodicFnInt(20, 120))

        # send the message to the client
        msg.pack_msg_data()
        self.write(msg.msg_data)

    # handle an incoming client message
    def handleMessage(self, message):
        print("receive message %d\t(%s)" % (message.message_id, message.name))
        if message.message_id == definitions.COMMON_GENERAL_REQUEST:
            # the client is requesting a message from us
            self.sendMessage(message.requested_id)
        # hack for legacy requests
        elif message.payload_length == 0:
            self.sendMessage(message.message_id)
        else:
            # the client is controlling some parameter of the device
            self.setParameters(message)

    # Extract message fields into attribute values
    # This should only be performed with the 'set' category of messages
    # TODO: mechanism to filter by "set"
    def setParameters(self, message):
        # for attr in payload_dict[message.message_id]["field_names"]:
        #     setattr(self, "_" + attr, getattr(message, attr))

        # Modificación para que el mensaje TRANSDUCER NO MODIFIQUE LA
CONFIGURACION
        for attr in payload_dict[message.message_id]["field_names"]:
            if attr == "angle":
                setattr(self, "_" + "angle", getattr(message, attr))
            elif attr == "transmit":
                setattr(self, "_" + "transmit", getattr(message, attr))

        #Actualizacion del salto de muestra
        # self._sample_jump =
configuration.sample_jump(self._sample_period,self._speed_of_sound)
        # self._range = int(round(self._sample_jump * self._number_of_samples))

    def set_range(self,range, number_of_samples = 1200,):
        int_range = int(range)
        t_lectura = 2 * int_range/self._speed_of_sound
        t_muestreo_sin_ajustar = t_lectura/number_of_samples

        self._sample_period = int(t_muestreo_sin_ajustar/25e-9) # Porque se
define en saltos de 25ns

        # Verificar si number_of_samples está en el rango 200 - 1200
        if not (200 <= number_of_samples <= 1200):
            print("Error: 'number_of_samples' = %d no se encuentra en el rango de
200 a 1200." % number_of_samples)
            return  sys.exit()# Salir de la función si number_of_samples está
fuera del rango

        # Verificar si t_muestreo está en el rango 80 - 40000
        if not (80 <= self._sample_period <= 40000):
```

```
            print("Error: 'sample_period' = %d no se encuentra en el rango de 80
a 40000. Cambie rango o número de muestras." % self._sample_period)
            return sys.exit() # Salir de la función si sample_period está fuera
del rango

        self._number_of_samples = int(number_of_samples)
        self._data_length = self._number_of_samples
        self._range = int_range
        self._transmit_duration = int(adjustTransmitDuration(int_range,
self._sample_period, self._speed_of_sound))
        self._sample_jump = sample_jump(self._sample_period,
self._speed_of_sound)
        return

    def set_transmit_frequency(self,transmit_frequency):
        self._transmit_frequency = transmit_frequency
        return

    def set_speed_of_sound(self,speed_of_sound):
        self._speed_of_sound = speed_of_sound
        return

    def print_distance_angle(self):
        #Parte de interpretación de distancias
        pos_max =
filtrado.max_filtered(self._sonar_data[self._angle],self._sample_jump)

        if pos_max:

            distancia = pos_max * self._sample_jump
            print("%i GRAD: %f m" % (self._angle, distancia))
        else:
            print("%i GRAD: No se ha detectado obstaculo." % self._angle)

    def bucle_ping360(self):

        self.set_range(5,1200) #Ojo con llamar a set range antes de llamar a
set_speed_of_sound

        #Selector de entorno
        self._sonar_data = entornos_ping360.pared_con_tamaño(self,0.5,4,100)
        # self._sonar_data = environments_ping360.piscina(self,1)

        while True:

            # read any incoming client communications
            self.read()

            if self._transmit == 1:
                # Force to transmit only if there is a request to do so
                self._transmit = 0
            self.sendMessage(definitions.PING360_DEVICE_DATA)

                #sim.print_distance_angle()

                # don't max cpu
                time.sleep(0.05)
                #time.sleep(2)
        #----------------------------

def getSamplePeriod(sample_period, _samplePeriodTickDuration=25e-9):

    # type: (float, float) -> float

    """  Sample period in ns """

    return sample_period * _samplePeriodTickDuration
```

```
def sample_jump(sample_period,speed_of_sound):
    sample_jump = (speed_of_sound * sample_period * 25e-9)/2
    return sample_jump



def adjustTransmitDuration(range, sample_period, speed_of_sound,
_firmwareMinTransmitDuration=5):

    # type: (float, float, int, int) -> float

    """

    @brief Adjust the transmit duration for a specific range

    Per firmware engineer:

    1. Starting point is TxPulse in usec = ((one-way range in metres) * 8000) /
(Velocity of sound in metres

    per second)

    2. Then check that TxPulse is wide enough for currently selected sample
interval in usec, i.e.,

        if TxPulse < (2.5 * sample interval) then TxPulse = (2.5 * sample
interval)

        (transmit duration is microseconds, samplePeriod() is nanoseconds)

    3. Perform limit checking

    Returns:

        float: Transmit duration

    """

    duration = 8000 * range / speed_of_sound

    transmit_duration = max(

        2.5 * getSamplePeriod(sample_period) / 1000, duration)

    return max(_firmwareMinTransmitDuration,
min(transmitDurationMax(sample_period), transmit_duration))

def transmitDurationMax(sample_period, _firmwareMaxTransmitDuration=500):

    # type: (float, int) -> float

    """

    @brief The maximum transmit duration that will be applied is limited
internally by the

    firmware to prevent damage to the hardware

    The maximum transmit duration is equal to 64 * the sample period in
microseconds

    Returns:

        float: The maximum transmit duration possible

    """
```

```
    return min(_firmwareMaxTransmitDuration, getSamplePeriod(sample_period) *
64e6)


################ COMENTAR CUANDO SE USEN HILOS EN CODIGO DE CONTROL
################################
# sim_ping360 = Ping360Simulation()
# sim_ping360.bucle_ping360()
```

11.7. Script filtrado.py

```
#!/usr/bin/env python3
import numpy as np

def max_filtered (data_int,sample_jump):

    #Calculamos el índice correspondiente a 40cm a partir del cual ya
consideramos que no corresponda a reflexiones del propio ROV
    sample_number_filter = int(0.4/sample_jump)
    # Filtro valores por percentil 90 y por valor de muestra >
number_sample_filter para filtrar el ROV
    percentile_90 = np.percentile(data_int, 90)
    filtered_percentile = [(i, val) for i, val in enumerate(data_int) if val >
percentile_90 and i > sample_number_filter]

    #Determinamos la prosición del máximo de intensidad
    # Verificar si filtered_percentile está vacío
    if not filtered_percentile:
        first_max_position = False
        #print("No hay objetos que destaquen.")
    else:
        # Separar posiciones e intensidades
        positions = [pos for pos, intensity in filtered_percentile]
        intensities = [intensity for pos, intensity in filtered_percentile]
        # Encontrar el valor máximo de intensidad
        max_intensity = max(intensities)
        # Encontrar la posición de la primera aparición de este valor máximo
        first_max_position = positions[intensities.index(max_intensity)]

        return first_max_position
```

11.8. Script prueba_control_heave.py

```
#! /usr/bin/python3
import time
import threading
from mavactive import mavactive, mavutil

from brping import Ping1D

import entornos_ping1d
import simulador_ping1d

# ------------------------------------------------
# Función de configuracion de frecuencia de solicitud de mensajes
# ------------------------------------------------
def request_message_interval(message_id: int, frequency_hz: float):
    master.mav.command_long_send(
        master.target_system, master.target_component,
        mavutil.mavlink.MAV_CMD_SET_MESSAGE_INTERVAL, 0,
        message_id, # The MAVLink message ID
        1e6 / frequency_hz, # The interval between two messages in microseconds.
Set to -1 to disable and 0 to request default rate.
        0, 0, 0, 0, # Unused parameters
        0, # Target address of message stream (if message has target address
fields). 0: Flight-stack default (recommended), 1: address of requestor, 2:
broadcast.
    )

# ------------------------------------------------
# Función de control de motores
# ------------------------------------------------
def set_rc_channel_pwm(channel_id, pwm=1500):
    if channel_id < 1 or channel_id > 18:
        print("Channel does not exist.")
        return
    rc_channel_values = [65535 for _ in range(18)]
    rc_channel_values[channel_id - 1] = pwm
    master.mav.rc_channels_override_send(
        master.target_system,
        master.target_component,
        *rc_channel_values)

# ------------------------------------------------
# Función que controla la profundidad del ROV
# ------------------------------------------------
def set_target_depth_vfr (depth, sim1D):
    # En esta versión, las profundidades positivas indican mayor profundidad
    # Por ejemplo: set_target_depth (1.5) lleva al ROV a 1.5 m de profundidad

    # Leemos la profundidad actual
    message = master.recv_match(type='VFR_HUD', blocking=True)
    depth_read = -message.alt  # Altitud negativa = profundidad positiva

    # Precisión en la profundidad deseada: ±0.2 m
    depth_delta = 0.2
    pwm = 1500
    delta_pwm = 60

    depth_diff = depth_read - depth  # Diferencia actual-objetivo

    while abs(depth_diff) > depth_delta:
        if depth_diff > 0:
            # Estamos más profundos de lo deseado: subir (menos empuje hacia
abajo)
            set_rc_channel_pwm(3, pwm + delta_pwm)
        else:
```

```
            # Estamos más altos de lo deseado: bajar (más empuje hacia abajo)
            set_rc_channel_pwm(3, pwm - delta_pwm)

        # Actualizamos profundidad
        message = master.recv_match(type='VFR_HUD', blocking=True)
        depth_read = -message.alt
        depth_diff = depth_read - depth

        print('VFR', 'depth_read', depth_read, 'depth_diff', depth_diff)

        # Actualización del perfil para el simulador
        sim1D._profile_data = entornos_ping1d.fondo_plano(sim1D,
sim1D._superficie_fondo - depth_read)
        #Ping1D para consultar la distancia
        get_distance_dict = {}
        get_distance_dict = myPing1D.get_distance_simple()
        distance = int(get_distance_dict.get('distance'))/1000
        confidence = int(get_distance_dict.get('confidence'))
        print("Distancia al fondo: %s m\t Certeza: %s" % (distance,confidence))

    time.sleep(5)

# ------------------------------------------------
# Configuración del ROV
# ------------------------------------------------

# # Create the connection
master = mavutil.mavlink_connection('udpin:0.0.0.0:9090')
boot_time = time.time()
# Wait a heartbeat before sending commands
master.wait_heartbeat()
# Set up a thread to send heartbeat pings TO the the ROV
GCS_to_ROV_heartbeat = mavactive(master)
# Messages we are going to request
# Configure VFR_HUD message to be sent at 10 Hz
request_message_interval(mavutil.mavlink.MAVLINK_MSG_ID_VFR_HUD, 10)
# Configure ATTITUDE message to be sent at 10 Hz
request_message_interval(mavutil.mavlink.MAVLINK_MSG_ID_ATTITUDE, 10)
# Configure SCALED_PRESSURE2 message to be sent at 10 Hz
request_message_interval(mavutil.mavlink.MAVLINK_MSG_ID_SCALED_PRESSURE2, 10)

# arm ArduSub autopilot and wait until confirmed
master.arducopter_arm()
master.motors_armed_wait()

# ------------------------------------------------
# Declaración de objeto Ping1D() y configuración de simulador Ping1DSimulation()
# ------------------------------------------------
sim1D = simulador_ping1d.Ping1DSimulation()
hilo_sim1D = threading.Thread(target=sim1D.bucle_ping1D, daemon=True)
hilo_sim1D.start()

myPing1D_data_dic = {}
myPing1D=Ping1D()
myPing1D.connect_udp("0.0.0.0",6675)
print("Ping1D initialized: %s \n" % myPing1D.initialize())

myPing1D.set_range(0,25000)
sim1D.update_sample_jump_period() #Debe ejecutarse despues de variar el rango

# ------------------------------------------------
# Inicio de maniobras
# ------------------------------------------------

#Partimos de un escenario de fondo a 20m
sim1D._superficie_fondo = 20
sim1D._profile_data = entornos_ping1d.fondo_plano(sim1D,sim1D._superficie_fondo)
```

```
time.sleep(10)


#set the desired operating mode
MODE = 'MANUAL'
CUSTOM_MODE = master.mode_mapping()[MODE]
while not master.wait_heartbeat().custom_mode == CUSTOM_MODE:
    master.set_mode(MODE)
print('Manual')

# Bajamos
print('########## Bajamos a -4m ##########')
depth = 4
min_distance_to_floor = 2

set_target_depth_vfr(depth,sim1D)
print('########## ROV a -4m de profundidad ##########')

#Pasamos a ALT_HOLD para mantener la profundidad
MODE = 'ALT_HOLD'
CUSTOM_MODE = master.mode_mapping()[MODE]
while not master.wait_heartbeat().custom_mode == CUSTOM_MODE:
    master.set_mode(MODE)
print('Alt Hold')

time.sleep(10)

MODE = 'MANUAL'
CUSTOM_MODE = master.mode_mapping()[MODE]
while not master.wait_heartbeat().custom_mode == CUSTOM_MODE:
    master.set_mode(MODE)
print('Manual')

# Subimos
print('########## Subimos a 0m ##########')
depth = 0
min_distance_to_floor = 2

set_target_depth_vfr(depth,sim1D)
print('########## ROV a 0m de profundidad ##########')
```

11.9. Script prueba_control_yaw.py

```
#! /usr/bin/python3
import time
import threading
from mavactive import mavactive, mavutil

from brping import Ping1D

import entornos_ping1d
import simulador_ping1d

# ------------------------------------------------
# Función de configuracion de frecuencia de solicitud de mensajes
# ------------------------------------------------
def request_message_interval(message_id: int, frequency_hz: float):
    master.mav.command_long_send(
        master.target_system, master.target_component,
        mavutil.mavlink.MAV_CMD_SET_MESSAGE_INTERVAL, 0,
        message_id, # The MAVLink message ID
        1e6 / frequency_hz, # The interval between two messages in microseconds.
Set to -1 to disable and 0 to request default rate.
        0, 0, 0, 0, # Unused parameters
        0, # Target address of message stream (if message has target address
fields). 0: Flight-stack default (recommended), 1: address of requestor, 2:
broadcast.
    )

# ------------------------------------------------
# Función de control de motores
# ------------------------------------------------
def set_rc_channel_pwm(channel_id, pwm=1500):
    if channel_id < 1 or channel_id > 18:
        print("Channel does not exist.")
        return
    rc_channel_values = [65535 for _ in range(18)]
    rc_channel_values[channel_id - 1] = pwm
    master.mav.rc_channels_override_send(
        master.target_system,
        master.target_component,
        *rc_channel_values)

# ------------------------------------------------
# Función que controla la profundidad del ROV
# ------------------------------------------------
def set_target_depth_vfr(depth, sim1D):
    # En esta versión, las profundidades positivas indican mayor profundidad
    # Por ejemplo: set_target_depth(1.5) lleva al ROV a 1.5 m de profundidad

    # Leemos la profundidad actual
    message = master.recv_match(type='VFR_HUD', blocking=True)
    depth_read = -message.alt  # Altitud negativa = profundidad positiva

    # Precisión en la profundidad deseada: ±0.2 m
    depth_delta = 0.2
    pwm = 1500
    delta_pwm = 60

    depth_diff = depth_read - depth  # Diferencia actual-objetivo

    while abs(depth_diff) > depth_delta:
        if depth_diff > 0:
            # Estamos más profundos de lo deseado: subir (menos empuje hacia
abajo)
            set_rc_channel_pwm(3, pwm + delta_pwm)
        else:
```

```
            # Estamos más altos de lo deseado: bajar (más empuje hacia abajo)
            set_rc_channel_pwm(3, pwm - delta_pwm)

        # Actualizamos profundidad
        message = master.recv_match(type='VFR_HUD', blocking=True)
        depth_read = -message.alt
        depth_diff = depth_read - depth

        print('VFR', 'depth_read', depth_read, 'depth_diff', depth_diff)

        # Actualización del perfil para el simulador
        sim1D._profile_data = entornos_ping1d.fondo_plano(sim1D,
sim1D._superficie_fondo - depth_read)
        #Ping1D para consultar la distancia
        get_distance_dict = {}
        get_distance_dict = myPing1D.get_distance_simple()
        distance = int(get_distance_dict.get('distance'))/1000
        confidence = int(get_distance_dict.get('confidence'))
        print("Distancia al fondo: %s m\t Certeza: %s" % (distance,confidence))

    time.sleep(5)

# ------------------------------------------------
# Configuración del ROV
# ------------------------------------------------

# # Create the connection
master = mavutil.mavlink_connection('udpin:0.0.0.0:9090')
boot_time = time.time()
# Wait a heartbeat before sending commands
master.wait_heartbeat()
# Set up a thread to send heartbeat pings TO the the ROV
GCS_to_ROV_heartbeat = mavactive(master)
# Messages we are going to request
# Configure VFR_HUD message to be sent at 10 Hz
request_message_interval(mavutil.mavlink.MAVLINK_MSG_ID_VFR_HUD, 10)
# Configure ATTITUDE message to be sent at 10 Hz
request_message_interval(mavutil.mavlink.MAVLINK_MSG_ID_ATTITUDE, 10)
# Configure SCALED_PRESSURE2 message to be sent at 10 Hz
request_message_interval(mavutil.mavlink.MAVLINK_MSG_ID_SCALED_PRESSURE2, 10)

# arm ArduSub autopilot and wait until confirmed
master.arducopter_arm()
master.motors_armed_wait()

# ------------------------------------------------
# Declaración de objeto Ping1D() y configuración de simulador Ping1DSimulation()
# ------------------------------------------------
sim1D = simulador_ping1d.Ping1DSimulation()
hilo_sim1D = threading.Thread(target=sim1D.bucle_ping1D, daemon=True)
hilo_sim1D.start()

myPing1D_data_dic = {}
myPing1D=Ping1D()
myPing1D.connect_udp("0.0.0.0",6675)
print("Ping1D initialized: %s \n" % myPing1D.initialize())

myPing1D.set_range(0,25000)
sim1D.update_sample_jump_period() #Debe ejecutarse despues de variar el rango

# ------------------------------------------------
# Inicio de maniobras
# ------------------------------------------------

#Partimos de un escenario de fondo a 20m
sim1D._superficie_fondo = 20
sim1D._profile_data = entornos_ping1d.fondo_plano(sim1D,sim1D._superficie_fondo)
```

```
time.sleep(10)


#set the desired operating mode
MODE = 'MANUAL'
CUSTOM_MODE = master.mode_mapping()[MODE]
while not master.wait_heartbeat().custom_mode == CUSTOM_MODE:
    master.set_mode(MODE)
print('Manual')

# Bajamos
print('########## Bajamos a -4m ##########')
depth = 4
min_distance_to_floor = 2

set_target_depth_vfr(depth,sim1D)
print('########## ROV a -4m de profundidad ##########')

#Pasamos a ALT_HOLD para mantener la profundidad
MODE = 'ALT_HOLD'
CUSTOM_MODE = master.mode_mapping()[MODE]
while not master.wait_heartbeat().custom_mode == CUSTOM_MODE:
    master.set_mode(MODE)
print('Alt Hold')

time.sleep(10)

MODE = 'MANUAL'
CUSTOM_MODE = master.mode_mapping()[MODE]
while not master.wait_heartbeat().custom_mode == CUSTOM_MODE:
    master.set_mode(MODE)
print('Manual')

# Subimos
print('########## Subimos a 0m ##########')
depth = 0
min_distance_to_floor = 2

set_target_depth_vfr(depth,sim1D)
print('########## ROV a 0m de profundidad ##########')
```

11.10. Script prueba_control_surge_pared.py

```
#! /usr/bin/python3
import time
import math
import threading
from mavactive import mavactive, mavutil
from brping import  Ping360
import entornos_ping360
import simulador_ping360, filtrado
import math

# ——————————————————————————————————————————
# Función de control de motores
# ——————————————————————————————————————————
def set_rc_channel_pwm(channel_id, pwm=1500):
    if channel_id < 1 or channel_id > 18:
        print("Channel does not exist.")
        return
    rc_channel_values = [65535 for _ in range(18)]
    rc_channel_values[channel_id - 1] = pwm
    master.mav.rc_channels_override_send(
        master.target_system,
        master.target_component,
        *rc_channel_values)

# ——————————————————————————————————————————
# Funciones auxiliares
# ——————————————————————————————————————————
def a_grados(rad):
    """Radianes → grados."""
    return rad * 180 / math.pi

def a_gradianes(rad):
    """Radianes → gradianes (200 g = π rad)."""
    return rad * 200 / math.pi

def normaliza_180(ang_deg):
    """Lleva cualquier ángulo a la franja [-180, 180)."""
    return (ang_deg + 180) % 360 - 180

# ——————————————————————————————————————————
# Función que controla el avance y retroceso del ROV
# ——————————————————————————————————————————
def set_rc_advance(sim360,direction,seconds):
    # Controlamos el avance o el retroceso
    pwm_neutral = 1500
    delta_pwm = 100
    if direction == 1 :
    # avanzamos
        pwm = pwm_neutral+delta_pwm

    elif direction == -1 :
    # retrocedemos
        pwm = pwm_neutral-delta_pwm

    # Nos movemos durante seconds segundos :-)
    for i in range(0,seconds):
        set_rc_channel_pwm(5, pwm)
        # Opcion 1: actualizar el entorno con cada encendido de motores
        sim360._sonar_data =
entornos_ping360.desplazamiento(sim360,0,direction*0.5)
        time.sleep(1)
    # Opcion 2: actualizar el entorno al finalizar la maniobra.
    # desplazamiento_en_y=direction*0.5*seconds
```

```
    # sim360._sonar_data =
environments_ping360.desplazamiento(sim360,0,desplazamiento_en_y)
    # Neutral
    set_rc_channel_pwm(5, pwm_neutral)
    time.sleep(2)

# ------------------------------------------------
# Función de configuracion de frecuencia de solicitud de mensajes
# ------------------------------------------------
def request_message_interval(message_id: int, frequency_hz: float):
    master.mav.command_long_send(
        master.target_system, master.target_component,
        mavutil.mavlink.MAV_CMD_SET_MESSAGE_INTERVAL, 0,
        message_id, # The MAVLink message ID
        1e6 / frequency_hz, # The interval between two messages in microseconds.
Set to -1 to disable and 0 to request default rate.
        0, 0, 0, 0, # Unused parameters
        0, # Target address of message stream (if message has target address
fields). 0: Flight-stack default (recommended), 1: address of requestor, 2:
broadcast.
    )

# ------------------------------------------------
# Funciones de consulta al Ping 360
# ------------------------------------------------
def get_distance_object_range(sim360,myPing360,start = 0,stop = 400):
    for angle in range(start,stop):
            msg = myPing360.transmitAngle(angle) # De auí sacamos los valores de
intensidades del sonar
            #print("Angulo: %d" % angle)
            data_array = msg.data

            data_lista = [k for k in data_array]

            sample_jump = sim360._sample_jump
            pos_max = filtrado.max_filtered(data_lista,sample_jump)

            if pos_max:

                distancia = pos_max * sample_jump
                print("%i GRAD: %f m" % (angle, distancia))
            else:
                print("%i GRAD: No se ha detectado obstaculo." % angle)

def get_distance_object(sim360,myPing360,angle):
    msg = myPing360.transmitAngle(angle) # De auí sacamos los valores de
intensidades del sonar
    #print("Angulo: %d" % angle)
    data_array = msg.data

    data_lista = [k for k in data_array]

    sample_jump = sim360._sample_jump
    pos_max = filtrado.max_filtered(data_lista,sample_jump)

    if pos_max:

        distancia = pos_max * sample_jump
        print("%i GRAD: %f m" % (angle, distancia))
    else:
        print("%i GRAD: No se ha detectado obstaculo." % angle)

# ------------------------------------------------
# Configuración del ROV
# ------------------------------------------------

# Create the connection
master = mavutil.mavlink_connection('udpin:0.0.0.0:9090')
```

```
boot_time = time.time()
# Wait a heartbeat before sending commands
master.wait_heartbeat()
# Set up a thread to send heartbeat pings TO the the ROV
GCS_to_ROV_heartbeat = mavactive(master)
# Messages we are going to request
# Configure VFR_HUD message to be sent at 10 Hz
request_message_interval(mavutil.mavlink.MAVLINK_MSG_ID_VFR_HUD, 10)
# Configure ATTITUDE message to be sent at 10 Hz
request_message_interval(mavutil.mavlink.MAVLINK_MSG_ID_ATTITUDE, 10)
# Configure SCALED_PRESSURE2 message to be sent at 10 Hz
request_message_interval(mavutil.mavlink.MAVLINK_MSG_ID_SCALED_PRESSURE2, 10)

# arm ArduSub autopilot and wait until confirmed
master.arducopter_arm()
master.motors_armed_wait()

# ------------------------------------------------
# Declaración de objeto Ping360() y configuración de simulador
Ping360Simulation()
# ------------------------------------------------
sim360 = simulador_ping360.Ping360Simulation()
hilo_sim360 = threading.Thread(target=sim360.bucle_ping360, daemon=True)
hilo_sim360.start()

myPing360_data_dic = {}
sim360.set_range(10,1200)

sim360._sonar_data = entornos_ping360.pared(sim360,2,100)

myPing360 = Ping360()
myPing360.connect_udp("0.0.0.0",6676)
print("Ping360 initialized: %s \n" % myPing360.initialize())

# ------------------------------------------------
# Inicio de maniobras
# ------------------------------------------------

#Se repite esta línea porque la función connect_udp modifica el diccionario
_sonar_data
sim360._sonar_data = entornos_ping360.pared(sim360,5,200)

time.sleep(2)

#set the desired operating mode
MODE = 'MANUAL'
CUSTOM_MODE = master.mode_mapping()[MODE]
while not master.wait_heartbeat().custom_mode == CUSTOM_MODE:
    master.set_mode(MODE)
print('Manual\n')

print('Comprobamos que hay una pared a 5m en la dirección de proa (200
gradianes)')
get_distance_object(sim360,myPing360,200)
# get_distance_object_range(sim360,myPing360,100,300)
time.sleep(2)

# Avanzamos 6 segundos (3m)
print('\n Avanzamos 6 segundos (3m)')
set_rc_advance(sim360,1,6)

print('\n Comprobamos que hay una pared a 2m en la dirección de proa (200
gradianes)')
get_distance_object(sim360,myPing360,200)
# get_distance_object_range(sim360,myPing360,100,300)
time.sleep(2)

# Retrocedemos 6 segundos (3m)
```

```
print('\n Retrocedemos 6 segundos (3m)')
set_rc_advance(sim360,-1,6)

print('\n Comprobamos que hay una pared a 5m en la dirección de proa (200
gradianes)')
get_distance_object(sim360,myPing360,200)
# get_distance_object_range(sim360,myPing360,100,300)
```

11.11. Script prueba_control_surge_piscina.py

```
#! /usr/bin/python3
import time
import math
import threading
from mavactive import mavactive, mavutil
from brping import  Ping360
import entornos_ping360
import simulador_ping360, filtrado
import math

# ------------------------------------------------
# Función de control de motores
# ------------------------------------------------
def set_rc_channel_pwm(channel_id, pwm=1500):
    if channel_id < 1 or channel_id > 18:
        print("Channel does not exist.")
        return
    rc_channel_values = [65535 for _ in range(18)]
    rc_channel_values[channel_id - 1] = pwm
    master.mav.rc_channels_override_send(
        master.target_system,
        master.target_component,
        *rc_channel_values)

# ------------------------------------------------
# Funciones auxiliares
# ------------------------------------------------
def a_grados(rad):
    """Radianes → grados."""
    return rad * 180 / math.pi

def a_gradianes(rad):
    """Radianes → gradianes (200 g = π rad)."""
    return rad * 200 / math.pi

def normaliza_180(ang_deg):
    """Lleva cualquier ángulo a la franja [-180, 180)."""
    return (ang_deg + 180) % 360 - 180

# ------------------------------------------------
# Función que controla el avance y retroceso del ROV
# ------------------------------------------------
def set_rc_advance(sim360,direction,seconds):
    # Controlamos el avance o el retroceso
    pwm_neutral = 1500
    delta_pwm = 100
    if direction == 1 :
    # avanzamos
        pwm = pwm_neutral+delta_pwm

    elif direction == -1 :
    # retrocedemos
        pwm = pwm_neutral-delta_pwm

    # Nos movemos durante seconds segundos :-)
    for i in range(0,seconds):
        set_rc_channel_pwm(5, pwm)
        # Opcion 1: actualizar el entorno con cada encendido de motores
        sim360._sonar_data =
entornos_ping360.desplazamiento(sim360,0,direction*0.5)
        time.sleep(1)
    # Opcion 2: actualizar el entorno al finalizar la maniobra.
    # desplazamiento_en_y=direction*0.5*seconds
```

```
    # sim360._sonar_data =
environments_ping360.desplazamiento(sim360,0,desplazamiento_en_y)
    # Neutral
    set_rc_channel_pwm(5, pwm_neutral)
    time.sleep(2)

# ------------------------------------------------
# Función de configuracion de frecuencia de solicitud de mensajes
# ------------------------------------------------
def request_message_interval(message_id: int, frequency_hz: float):
    master.mav.command_long_send(
        master.target_system, master.target_component,
        mavutil.mavlink.MAV_CMD_SET_MESSAGE_INTERVAL, 0,
        message_id, # The MAVLink message ID
        1e6 / frequency_hz, # The interval between two messages in microseconds.
Set to -1 to disable and 0 to request default rate.
        0, 0, 0, 0, # Unused parameters
        0, # Target address of message stream (if message has target address
fields). 0: Flight-stack default (recommended), 1: address of requestor, 2:
broadcast.
    )

# ------------------------------------------------
# Funciones de consulta al Ping 360
# ------------------------------------------------
def get_distance_object_range(sim360,myPing360,start = 0,stop = 400):
    for angle in range(start,stop):
            msg = myPing360.transmitAngle(angle) # De auí sacamos los valores de
intensidades del sonar
            #print("Angulo: %d" % angle)
            data_array = msg.data

            data_lista = [k for k in data_array]

            sample_jump = sim360._sample_jump
            pos_max = filtrado.max_filtered(data_lista,sample_jump)

            if pos_max:

                distancia = pos_max * sample_jump
                print("%i GRAD: %f m" % (angle, distancia))
            else:
                print("%i GRAD: No se ha detectado obstaculo." % angle)

def get_distance_object(sim360,myPing360,angle):
    msg = myPing360.transmitAngle(angle) # De auí sacamos los valores de
intensidades del sonar
    #print("Angulo: %d" % angle)
    data_array = msg.data

    data_lista = [k for k in data_array]

    sample_jump = sim360._sample_jump
    pos_max = filtrado.max_filtered(data_lista,sample_jump)

    if pos_max:

        distancia = pos_max * sample_jump
        print("%i GRAD: %f m" % (angle, distancia))
    else:
        print("%i GRAD: No se ha detectado obstaculo." % angle)

# ------------------------------------------------
# Configuración del ROV
# ------------------------------------------------
# Create the connection
master = mavutil.mavlink_connection('udpin:0.0.0.0:9090')
boot_time = time.time()
```

```
# Wait a heartbeat before sending commands
master.wait_heartbeat()
# Set up a thread to send heartbeat pings TO the the ROV
GCS_to_ROV_heartbeat = mavactive(master)
# Messages we are going to request
# Configure VFR_HUD message to be sent at 10 Hz
request_message_interval(mavutil.mavlink.MAVLINK_MSG_ID_VFR_HUD, 10)
# Configure ATTITUDE message to be sent at 10 Hz
request_message_interval(mavutil.mavlink.MAVLINK_MSG_ID_ATTITUDE, 10)
# Configure SCALED_PRESSURE2 message to be sent at 10 Hz
request_message_interval(mavutil.mavlink.MAVLINK_MSG_ID_SCALED_PRESSURE2, 10)

# arm ArduSub autopilot and wait until confirmed
master.arducopter_arm()
master.motors_armed_wait()

# ------------------------------------------------
# Declaración de objeto Ping360() y configuración de simulador
Ping360Simulation()
# ------------------------------------------------
sim360 = simulador_ping360.Ping360Simulation()
hilo_sim360 = threading.Thread(target=sim360.bucle_ping360, daemon=True)
hilo_sim360.start()

myPing360_data_dic = {}
sim360.set_range(10,1200)

sim360._sonar_data = entornos_ping360.pared(sim360,2,100)

myPing360 = Ping360()
myPing360.connect_udp("0.0.0.0",6676)
print("Ping360 initialized: %s \n" % myPing360.initialize())

# ------------------------------------------------
# Inicio de maniobras
# ------------------------------------------------

#Se repite esta línea porque la función connect_udp modifica el diccionario
_sonar_data
sim360._sonar_data = entornos_ping360.piscina(sim360,5)

time.sleep(5)

#set the desired operating mode
MODE = 'MANUAL'
CUSTOM_MODE = master.mode_mapping()[MODE]
while not master.wait_heartbeat().custom_mode == CUSTOM_MODE:
    master.set_mode(MODE)
print('Manual\n')

print('Comprobamos que hay una pared de piscina a 5m en todas las direcciones')
get_distance_object(sim360,myPing360,200)
get_distance_object(sim360,myPing360,0)
get_distance_object(sim360,myPing360,100)
get_distance_object(sim360,myPing360,300)
# get_distance_object_range(sim360,myPing360,100,300)
time.sleep(2)

# Avanzamos 6 segundos (3m)
print('\n Avanzamos 6 segundos (3m)')
set_rc_advance(sim360,1,6)

print('\n Comprobamos que hay una pared de piscina a 2m en la dirección de proa
(200 gradianes)' \
'\n 8m en la dirección de popa (0 gradianes)' \
'\n y aproximadamente 5m a babor y estribor)')
get_distance_object(sim360,myPing360,200)
get_distance_object(sim360,myPing360,0)
```

```
get_distance_object(sim360,myPing360,100)
get_distance_object(sim360,myPing360,300)
# get_distance_object_range(sim360,myPing360,100,300)
time.sleep(2)

# Retrocedemos 6 segundos (3m)
print('\n Retrocedemos 6 segundos (3m)')
set_rc_advance(sim360,-1,6)

print('Comprobamos que hay una pared de piscina a 5m en todas las direcciones')
get_distance_object(sim360,myPing360,200)
get_distance_object(sim360,myPing360,0)
get_distance_object(sim360,myPing360,100)
get_distance_object(sim360,myPing360,300)
# get_distance_object_range(sim360,myPing360,100,300)
```

11.12. Script prueba_brechas_datos.py

```
#!/usr/bin/env python3
import numpy as np
import entornos_ping360
import math

number_of_samples = 1200 # Número de índices por ángulo
sample_jump = 0.0041 # Distancia entre muestras en metro (m/muestra)
distancia_pared = 3 # Distancia inicial en metros hasta la pared
tamaño_de_pared = 3 # Tamaño de la pared de extremo a extremo en metros
angulo_pared_gradianes = 200
data_length = 1200

# Inicializar el diccionario para almacenar los datos del sonar para cada ángulo
sonar_data = {}

def pared_con_tamaño(tamaño_pared,distancia_pared,angulo_pared_gradianes = 0):
    # Ángulo de la pared respecto al sonar
    angulo_pared_grados = angulo_pared_gradianes * 0.9
    # Convertir el ángulo a radianes
    angulo_pared_radianes = np.radians(angulo_pared_grados)

    # Crear un diccionario paraS almacenar los valores para cada ángulo
    sonar_data = {}
    # Rango de angulos en radianes a cubir para conseguir el tamaño de la pared
    angulo_tamaño_radianes = np.arctan((tamaño_pared/2)/distancia_pared)
    angulo_tamaño_gradianes = angulo_tamaño_radianes*(200/math.pi) # Pasamos a
gradianes para hacer el barrido

    # Rango de ángulos en el que se generará la pared, con ajuste circular
    angle_range_min = (angulo_pared_gradianes - angulo_tamaño_gradianes) % 400
    angle_range_max = (angulo_pared_gradianes + angulo_tamaño_gradianes) % 400

    # Recorrer los ángulos de 0 a 400 gradianes
    for angle_gradianes in range(400):
        data = []
        # Convertir el ángulo a radianes
        angle_grados = angle_gradianes * 0.9
        angle_radianes = angle_grados * (np.pi/180)

        # Calcular r usando la ecuación de la recta
        r_value = distancia_pared / np.cos(angle_radianes -
angulo_pared_radianes)

        # Calcular el índice correspondiente a r_value
        n_value_float = r_value / sample_jump
        n_value_int = int(n_value_float)

        # Vemos si estamos en caso de salto de 400-0
        if angle_range_min < angle_range_max:
            # Caso normal
            in_angle_range = angle_range_min <= angle_gradianes <=
angle_range_max
        else:
            # Caso que cruza el límite de 400 grados
            in_angle_range = angle_gradianes >= angle_range_min or
angle_gradianes <= angle_range_max

        # Llenar los datos para cada muestra `n`
        for n in range(number_of_samples):
            # if n_value <= n <= n_value + int(0.1 / sample_jump) and
in_angle_range:
            if n_value_int == n and in_angle_range:
                data.append(255)
            else:
```

```
                data.append(0)

        # Guardar el resultado en el diccionario, usando el ángulo como clave
        sonar_data[angle_gradianes] = data

    return sonar_data

def localizar_255(sonar_data):
    lista_255 = []
    for angle, array in sonar_data.items():
        for index, value in enumerate(array):
            if value == 255:
                tupla = (angle,index)
                lista_255.append(tupla)
    for tupla in lista_255:
        print(tupla)
    return lista_255

def desplazamiento(sonar_data,desplazamiento_x=0, desplazamiento_y=0):
    new_sonar_data = {clave: [0] * len(valor) for clave, valor in
sonar_data.items()}
    if desplazamiento_x != 0 or desplazamiento_y != 0:
        for angle, array in sonar_data.items():
            for index, value in enumerate(array):
                #DESPLAZAR SOLO LOS VALORES DE 255 PORQUE EL VALOR NULO ESTÁ
MACHACANDO EL 255
                if value == 255:
                    #Convertimos a distancia el indice
                    distance = index * sample_jump
                    # Convertir coordenadas polares a cartesianas
                    x, y = entornos_ping360.polares_a_cartesianas(distance,
angle)

                    # Aplicar el desplazamiento
                    nueva_x = x + desplazamiento_x
                    nueva_y = y + desplazamiento_y

                    # Convertir de vuelta a polares
                    new_distance, theta =
entornos_ping360.cartesianas_a_polares(nueva_x, nueva_y)
                    r = new_distance / sample_jump
                    r = round(r)
                    # Validar los índices radial y angular
                    if 0 <= r < data_length:
                        theta = round(theta) % 400  # Asegurar índice angular
válido
                        new_sonar_data[theta][r] = value
    return new_sonar_data

sonar_data =
pared_con_tamaño(tamaño_de_pared,distancia_pared,angulo_pared_gradianes)
print("ANTES DE DESPLAZAR")
localizar_255 (sonar_data)
sonar_data_desplazado = desplazamiento(sonar_data, 0, 2)
print("DESPUES DE DESPLAZAR")
localizar_255 (sonar_data_desplazado)
```

11.13. Script prueba_brechas_pingviewer.py

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

# This script simulates a Blue Robotics Ping 360  device
# A client may connect to the device simulation on local UDP port 6676

from brping import definitions
import entornos_ping360, simulador_ping360
import time

payload_dict = definitions.payload_dict_all
##################### PRUEBA PROBLEMAS DE BRECHAS DE OBSTÁCULO
############################

#The simulation to use and config
sim = simulador_ping360.Ping360Simulation()

#Configuración del sonar  NO LE SIRVE AL COMPANION YA QUE ESTO LO DEBERÍA HACER
ENVIANDO UN MENSAJE TRANSDUCER
sim.set_transmit_frequency(750)
sim.set_speed_of_sound(1500)
sim.set_range(5,1200) #Ojo con llamar a set range antes de llamar a
set_speed_of_sound

#Selector de entorno
sim._sonar_data = entornos_ping360.pared_con_tamaño(sim,3,3,200)
# sim._sonar_data = environments_ping360.piscina(sim,3)
# sim._sonar_data = environments_ping360.pared_con_tamaño(sim,3,3,300)
nprueba = 1

# DESPLAZAMIENTO INTERMITENTE
if nprueba == 1:
    n=1
    distancia_desplazada = 2
    while True:

        if sim._angle == 100:

            # Desplazamiento alternado
            if n == 1:
                sim._sonar_data =
entornos_ping360.desplazamiento(sim,0,distancia_desplazada)
                n = 0
            else:
                sim._sonar_data = entornos_ping360.desplazamiento(sim,0,-
distancia_desplazada)
                n = 1

            # Desplazamiento unico
            # sim._sonar_data =
environments_ping360.desplazamiento(sim,0,distancia_desplazada)

        # read any incoming client communications
        sim.read()

        if sim._transmit == 1:
            # Force to transmit only if there is a request to do so
            sim._transmit = 0
            sim.sendMessage(definitions.PING360_DEVICE_DATA)

            #sim.print_distance_angle()

            # don't max cpu
            time.sleep(0.010)
            #time.sleep(2)
```

```
# ACERCAMIENTO 1M
elif nprueba == 2:
    n=0
    distancia_desplazada = 1
    while True:

        if sim._angle == 1:

            # Desplazamiento alternado
            if n == 1:
                sim._sonar_data =
entornos_ping360.desplazamiento(sim,0,distancia_desplazada)
                n = n+1
            else:
                n = n+1

            # Desplazamiento unico
            # sim._sonar_data =
environments_ping360.desplazamiento(sim,0,distancia_desplazada)

        # read any incoming client communications
        sim.read()

        if sim._transmit == 1:
            # Force to transmit only if there is a request to do so
            sim._transmit = 0
            sim.sendMessage(definitions.PING360_DEVICE_DATA)

            #sim.print_distance_angle()

            # don't max cpu
            time.sleep(0.010)
            #time.sleep(2)

# ACERCAMIENTO 2M
elif nprueba == 3:
    n=0
    distancia_desplazada = 2
    while True:

        if sim._angle == 1:

            # Desplazamiento alternado
            if n == 1:
                sim._sonar_data =
entornos_ping360.desplazamiento(sim,0,distancia_desplazada)
                n = n+1
            else:
                n = n+1

            # Desplazamiento unico
            # sim._sonar_data =
environments_ping360.desplazamiento(sim,0,distancia_desplazada)

        # read any incoming client communications
        sim.read()

        if sim._transmit == 1:
            # Force to transmit only if there is a request to do so
            sim._transmit = 0
            sim.sendMessage(definitions.PING360_DEVICE_DATA)

            #sim.print_distance_angle()

            # don't max cpu
            time.sleep(0.010)
            #time.sleep(2)
```

```
# ACERCAMIENTO 3M
elif nprueba == 4:
    n=0
    distancia_desplazada = 3
    while True:

        if sim._angle == 1:

            # Desplazamiento alternado
            if n == 1:
                sim._sonar_data =
entornos_ping360.desplazamiento(sim,0,distancia_desplazada)
                n = n+1
            else:
                n = n+1

            # Desplazamiento unico
            # sim._sonar_data =
environments_ping360.desplazamiento(sim,0,distancia_desplazada)

        # read any incoming client communications
        sim.read()

        if sim._transmit == 1:
            # Force to transmit only if there is a request to do so
            sim._transmit = 0
            sim.sendMessage(definitions.PING360_DEVICE_DATA)

            #sim.print_distance_angle()

            # don't max cpu
            time.sleep(0.010)
            #time.sleep(2)
```

11.14. *Script* de ejemplo Pymavlink

```
# Import mavutil
print("Importing mavutil")
from pymavlink import mavutil

# Create the connection
print("Connecting to vehicle on 'udpin:0.0.0.0:14550'")
master = mavutil.mavlink_connection('udpin:0.0.0.0:14550')

# Wait a heartbeat before sending commands
print("Waiting for first heartbeat")
master.wait_heartbeat()
print("Heartbeat received")

#https://mavlink.io/en/messages/common.html#MAV_CMD_COMPONENT_ARM_DISARM
# Arm
print("Sending ARM command")
# master.arducopter_arm() or:
master.mav.command_long_send(
master.target_system,
master.target_component,
mavutil.mavlink.MAV_CMD_COMPONENT_ARM_DISARM,
0,
1, 0, 0, 0, 0, 0, 0)

# wait until arming confirmed (can manually check with master.motors_armed())
print("Waiting for the vehicle to arm")
master.motors_armed_wait()
print('Armed!')

# Disarm
print("Sending DISARM command")
# master.arducopter_disarm() or:
master.mav.command_long_send(
master.target_system,
master.target_component,
mavutil.mavlink.MAV_CMD_COMPONENT_ARM_DISARM,
0,
0, 0, 0, 0, 0, 0, 0)

# wait until disarming confirmed
print("Waiting for the vehicle to disarm")
master.motors_disarmed_wait()
print("Disarmed, operation complete")
```

11.15. Salida por terminal del script *prueba_ brecha_ datos.py*

```
adrian@gic92:~/Documentos/Scripts_Adrian$  cd
/home/adrian/Documentos/Scripts_Adrian ; /usr/bin/env /bin/python
/home/adrian/.vscode/extensions/ms-python.debugpy-2025.8.0-linux-
x64/bundled/libs/debugpy/adapter/../../debugpy/launcher 40045 --
/home/adrian/Documentos/Scripts_Adrian/prueba_brecha_datos.py
ANTES DE DESPLAZAR
(171, 814)
(172, 808)
(173, 802)
(174, 797)
(175, 791)
(176, 786)
(177, 782)
(178, 777)
(179, 773)
(180, 769)
(181, 765)
(182, 761)
(183, 758)
(184, 755)
(185, 752)
(186, 749)
(187, 747)
(188, 744)
(189, 742)
(190, 740)
(191, 739)
(192, 737)
(193, 736)
(194, 734)
(195, 733)
(196, 733)
(197, 732)
(198, 732)
(199, 731)
(200, 731)
(201, 731)
(202, 732)
(203, 732)
(204, 733)
(205, 733)
(206, 734)
(207, 736)
(208, 737)
(209, 739)
(210, 740)
(211, 742)
(212, 744)
(213, 747)
(214, 749)
(215, 752)
(216, 755)
(217, 758)
(218, 761)
(219, 765)
(220, 769)
(221, 773)
(222, 777)
(223, 782)
(224, 786)
(225, 791)
(226, 797)
(227, 802)
(228, 808)
```

```
(229, 814)
DESPUES DE DESPLAZAR
(138, 433)
(139, 421)
(140, 410)
(142, 399)
(143, 388)
(144, 378)
(146, 369)
(147, 358)
(149, 349)
(151, 340)
(153, 331)
(154, 323)
(156, 315)
(158, 307)
(160, 300)
(162, 293)
(165, 287)
(167, 280)
(169, 275)
(172, 269)
(174, 265)
(177, 260)
(180, 257)
(182, 253)
(185, 250)
(188, 248)
(191, 246)
(194, 245)
(197, 243)
(200, 243)
(203, 243)
(206, 245)
(209, 246)
(212, 248)
(215, 250)
(218, 253)
(220, 257)
(223, 260)
(226, 265)
(228, 269)
(231, 275)
(233, 280)
(235, 287)
(238, 293)
(240, 300)
(242, 307)
(244, 315)
(246, 323)
(247, 331)
(249, 340)
(251, 349)
(253, 358)
(254, 369)
(256, 378)
(257, 388)
(258, 399)
(260, 410)
(261, 421)
(262, 433)
```