industriales
etsii

Escuela Técnica Superior de Ingeniería Industrial

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Escuela Técnica Superior de Ingeniería Industrial

Modelado, simulación y control de pequeños vehículos submarinos no tripulados

TRABAJO FIN DE GRADO

GRADO EN INGENIERIA ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Autor: **Pablo Aguirre Cárcel**
Director: Gregorio Munuera Saura

Cartagena, 19 de abril de 2020

Universidad Politécnica de Cartagena

Índice

Índice de ilustraciones

Índice de tablas

Capítulo 1. Introducción

El desarrollo tecnológico de vehículos autónomos está teniendo en los últimos años una gran importancia. Esto es debido en gran medida a la reducción de coste y de tamaño de los componentes principales, como son los sensores y las unidades de computación de pequeñas dimensiones. Este proyecto tiene como objeto de estudio los UUV (*Unmanned underwater vehicles*) los cuales tienen una gran cantidad de aplicaciones, desde la realización de trabajos peligrosos para una persona hasta la investigación de los fondos marinos.

El campo del modelado y la simulación está también en auge gracias al incremento del poder de computación del que nos beneficiamos tanto los usuarios como los investigadores. Por ello cada vez resulta más sencillo el desarrollo y uso de sistemas de simulación que imiten un entorno submarino donde probar en segundos: miles de diseños ROV diferentes, modos de funcionamiento, respuesta ante situaciones poco comunes…, para los cuales serían necesarias cientos horas de preparación para realizar cada uno de los experimentos de manera real.

1.1. Antecedentes

Este proyecto se puede entender como la continuación lógica en la línea de investigación del Área de Ingeniería Mecánica en vehículos submarinos no tripulados. Esta línea ha sido desarrollada en gran medida mediante la elaboración de trabajos de fin de estudios de diferentes alumnos. En concreto se puede establecer como claro precedente el proyecto del alumno Ignacio Cotera [1] “Construcción, modelización y simulación de un pequeño vehículo submarino comercial”. En este se montó el ROV comercial OpenROV y se obtuvieron mediante simulaciones con la herramienta informática Flowsimulation los coeficientes hidrodinámicos con los que modelar su comportamiento bajo el agua. Este trabajo nos permitirá tener disponible un ROV real tanto para realizar mejoras y modificaciones en su control como para comprobar la validez del modelo y los parámetros obtenidos comparando los resultados que obtengamos en el simulador con los que resulten al realizar experimentos con el ROV.

Para la obtención y uso de estos coeficientes hidrodinámicos se han utilizado las bases sentadas en otros proyectos anteriores relacionados con esta línea de investigación como: el elaborado por Antonio Garrido Pellicer [2] dónde se estudian los distintos métodos para obtener los valores y se explica cómo obtener estos mediante los métodos de mecánica de fluidos computacional (en inglés, Computational Fluid Dynamics, CFD) (con este método se obtienen, como ya hemos

comentado, los parámetros para el vehículo OpenROV), pero también en el trabajo de Jose Antonio Ruiz Ruiz [3] se obtienen los parámetros de los vehículos CUBO y REMUS. En la misma dirección va el trabajo de José García García[4], pero usando un software alternativo.

Para obtener este programa simulador nos hemos apoyado en el programa anterior generado en parte por las aportaciones de trabajos como el de Jorge Juan García García [5] donde se expone el modelo matemático del simulador y los métodos computacionales para resolver las ecuaciones dinámicas del mismo. En este trabajo el modelo es implementado en Matlab, sin embargo, este es traspasado a C++ por Javier de la Red en su TFG, un lenguaje de nivel de abstracción menor, de esta forma se puede obtener una mayor eficiencia computacional.

No podemos dejar de mencionar los esfuerzos de investigación en el diseño de diferentes ROVs funcionales con el objetivo de ser construidos y usados para la investigación gracias a los cuales se ha avanzado en el conocimiento del comportamiento y prestaciones de los modelos. Entre estos trabajos podemos destacar los de Antonio Resina[6], Álvaro García Foz y José García García[4].

En cuanto a los métodos de control para vehículos submarinos, tema el cual trataremos en este proyecto, tenemos como precedente el trabajo de Eloy Yagüe Martínez, donde desarrolla un método de control para el vehículo CUBO. También tocan este campo los trabajos de Javier de la Red y Federico Sánchez Durán.

1.2. Objetivos

Los objetivos planteados para este proyecto son los siguientes:

- Continuar el desarrollo del programa de simulación en C++ de vehículos submarinos.
- Mejorar el modelo actual para que su comportamiento sea más parecido al real.
- Estructurar la programación del modelo para que pueda ser compatible con el mayor número de vehículos submarinos posible.
- Plantear y desarrollar una forma sencilla y amigable para el usuario que permita cambiar los diferentes parámetros del modelo y de la simulación de manera ordenada y clara.
- Proponer y desarrollar sistemas de control básicos para los vehículos modelados.
- Configurar la herramienta desarrollada para varios vehículos: REMUS100, OpenROV y CUBO.
- Comprobar el comportamiento del simulador realizado para diferentes situaciones con el modelo del OpenROV.

- Reprogramación del software del vehículo OpenROV real para poder controlarlo de forma manual o mediante un método de control asistido.
- Contrastar los resultados obtenidos de la simulación, con pruebas realizadas utilizando el vehículo OpenROV real.

1.3. Fases

1. Documentación y estudio de información sobre diferentes tipos de UUVs y ROVs, aplicaciones, últimos avances…
2. Estudio del modelo matemático mediante el cual simulamos el comportamiento del vehículo.
3. Estudio de buenas prácticas y métodos estándar de programación en C++, orientación a objetos (herencia, composición, polimorfismo…), estructuración de programa, uso de librerías…
4. Análisis sobre la mejor estructuración del programa de simulación a desarrollar para que se pueda adaptar a la gama más amplia de vehículos y a sus diferentes configuraciones.
5. Implementación del método de entrada de datos y mejoras al programa simulador para aumentar su universalidad y precisión.
6. Introducción de parámetros de los diferentes vehículos de los cuales disponemos de coeficientes hidrodinámicos (REMUS, CUBO y OpenROV).
7. Desarrollo en el simulador de un método de control de posición general para cualquier tipo de ROV, independientemente del numero y situación de sus hélices.
8. Pruebas y parametrización de los controladores para los diferentes vehículos modelados.
9. Desarrollo en el simulador de un sistema de control específico para el vehículo OpenROV.
10. Estudio de JavaScript y NodeJS, necesario para entender y modificar la programación de los diferentes sistemas de control y comunicación del OpenROV.
11. Pruebas y parametrización del sistema de control para el OpenROV.
12. Implementación de mejoras en la programación del OpenROV.
13. Experimentación con el vehículo tanto para comprobar el funcionamiento de las mejoras como para comprobar la similitud entre el modelo simulado y el comportamiento real.

Capítulo 2. Estado del arte

En la actualidad existe una gran variedad de sistemas con los que simular el comportamiento y control de vehículos submarinos. Algunos más generalista y accesibles para cualquier usuario mientras que otros se desarrollan de manera cerrada para dar solución a problemas concretos de una investigación o relacionados con una actividad empresarial determinada. Nosotros nos centraremos en explorar los diferentes métodos de modelado y simulación a los cuales se puede acceder de forma pública cuyas posibilidades de uso abarcan la mayor cantidad de escenarios posibles.

2.1. Simuladores

2.1.1. Simuladores basados en ROS

Si hablamos de software accesible de desarrollo, modelado y simulación de robots móviles (entendiendo que los vehículos submarinos no tripulados son en realidad robots móviles) no podemos dejar de comentar las diferentes aplicaciones basadas en ROS

ROS, del inglés Robot Operating System (Sistema Operativo Robótico) es un framework para el desarrollo de software para robots que provee la funcionalidad de un sistema operativo en un clúster heterogéneo[7]. Se podría decir que ROS es sistema operativo de código abierto el cual nos proporciona un entorno de desarrollo sobre el cual se pueden construir diferentes aplicaciones para robots, tanto para su programación como para su simulación.

Ilustración 1 Logotipo de Ros

ROS nos facilita en gran medida el desarrollo de aplicaciones para robots ya que nos ofrece una gran cantidad de funcionalidades, herramientas y atajos para comenzar a crear nuestro programa desde un nivel superior. Es decir, al comenzar el desarrollo de una aplicación robótica no tendremos que crear gran parte de las funcionalidades desde cero, como por ejemplo la

comunicación entre las diferentes partes, el entorno de simulación tanto 3d como 2d etc. Ya que están incorporadas dentro del propio sistema. Además, te da acceso a muchas otras características más avanzadas y complejas que ya han sido desarrolladas por otros usuarios y son fácilmente implementables a tu aplicación, y de forma gratuita ya que es un software de código abierto y dispone de una comunidad muy activa. Algunas de estas funcionalidades pueden ser el reconocimiento de objetos, el cálculo de rutas de navegación, la planificación del movimiento… [8]

Además, es usado comúnmente junto al programa *Gazebo*, un entorno grafico 3D simulador de solidos rígidos con el que visualizar e interactuar con la simulación de los robots. Está integrado con ROS mediante el uso de mensajes, servicios y dinámicas ROS.

En la actualidad podemos encontrar una gran cantidad de proyectos desarrollados sobre ROS que nos permiten simular el funcionamiento de vehículos submarinos. A continuación, analizaremos algunos de ellos:

2.1.1.1. UUV Simulator

[9]

UUV_Simulator es un set de paquetes el cual incluye plugins y aplicaciones ROS que permiten entre otras cosas la simulación de vehículos submarinos en Gazebo. Todos ellos son de código abierto y compatibles entre sí, por lo que mediante el uso de estas herramientas las cuales han sido desarrolladas por la comunidad puedes disponer de un entorno de simulación muy completo donde comenzar nuevos proyectos, realizar pruebas de diseño o probar diferentes métodos de control.

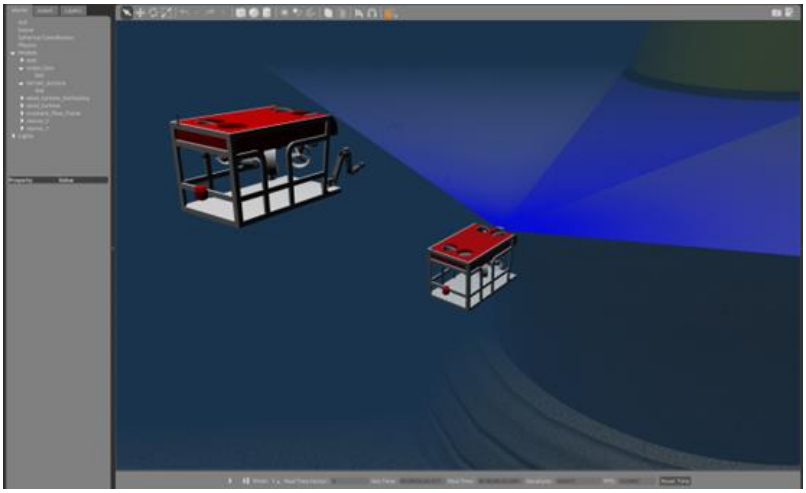


Ilustración 2 UUV Simulator

Podemos enumerar algunas de las funcionalidades que incluye este conjunto de paquetes:

- Implementación de las ecuaciones del movimiento en vehículos submarinos de Fossen [10]
- Modelos de propulsores que relacionan la velocidad angular de la hélice con la fuerza ejercida, mediante el uso de diferentes modelos matemáticos, por ejemplo el modelo de Yoerger (1990) [11] o el de Bessa (2006) [12]
- Modelado de fuerzas resultantes por acción de los timones.
- Simulación de corrientes de agua, pudiendo ser o de velocidad constante o siguiendo el proceso Gauss-Markov[13]
- Simulación de diferentes sensores
- Varios métodos de control (Controladores PID no lineales [10], controladores PID de los 6 grados de libertad, controladores PD con restauración de fuerzas de compensación…)
- Entorno visual con texturas adecuadas para simulación de entornos submarinos
- Escenarios de demostración donde probar las simulaciones
- Modelos completos de diferentes vehículos: *ECA A9 AUV* [14], *Light Autonomous Underwater Vehicle (LAUV)* [15]…

Alguna de estas funcionalidades, como los diferentes métodos de control, pueden ser no solo usados para la simulación, sino que pueden ser programados y utilizados por el robot real, teniendo acceso así a sistemas de control de alto nivel y complejidad sin tener que desarrollarlos desde cero.

2.1.1.2. UWSim

UWSim es un simulador de vehículos submarinos pensado para el desarrollo e investigación de robots marinos. Este funciona sobre ROS es un sustituto para el simulador ROS por defecto Gazebo. El escenario submarino puede ser configurado usando software de modelado y se pueden añadir vehículos submarinos, vehículos superficiales y vehículos con brazos robóticos manipuladores, además de sensores simulados accesibles mediante interfaces ROS. Esto permite integrar fácilmente la herramienta de visualización con las arquitecturas de control. [16]

Ilustración 3 UWSim

2.1.2. Simuladores de investigación

También podemos encontrar simuladores no basados en el sistema ROS. Estos no suelen seguir la misma filosofía que los estudiados anteriormente como es el software libre o la fácil disponibilidad para su modificación. Encontramos simuladores desarrollados por universidades y centros de investigación elaborados para resolver problemas concretos y avanzar en el conocimiento de estos aparatos, pero no suelen estar disponibles para su uso por personas ajenas a la investigación. A continuación, citaremos algunos artículos publicados recientemente que tratan el campo de la simulación de vehículos submarinos:

2.1.2.1. Simulación dinámica eficiente de un vehículo submarino con un manipulador robótico

En este artículo sus autores S. McMillan, D.E. Orin y R.B. McGhee buscan conseguir un simulador dinámico eficiente para un vehículo el cual dispone de un manipulador robótico. Se basan en trabajos anteriores en eficiencia de algoritmos, especialmente en el cálculo de movimientos del manipulador. También son incluidas las fuerzas hidrodinámicas que aparecen en los sistemas submarinos (masa añadida, arrastre viscoso, aceleración del fluido y fuerzas de flotabilidad). Los autores del articulo afirman haber implementado un algoritmo con el doble de eficiencia computacional que los simuladores anteriores. [17]

2.1.2.2. Modelado y simulación de vehículos submarinos autónomos: diseño e implementación.

En este *artículo* se expone el trabajo de investigación en un modelo simulador en tiempo real basado en físicas. Se incluye el modelado de las dinámicas del vehículo, el entorno y las características de los sensores. La simulación de componentes consiste en la implementación de un simulador autónomo y de un sistema hardware-in-the-loop (HIL). [18]

Un sistema HIL es una técnica en la que las señales reales de un controlador son conectadas a un sistema de pruebas que simula la realidad, engañando al controlador para que piense que está en el producto ensamblado. La prueba y la iteración del diseño se realiza como si se estuviera utilizando el sistema del mundo real. Así se pueden ejecutar fácilmente miles de escenarios posibles para poner en práctica a su controlador sin el costo y el tiempo asociados con las pruebas reales. [19]

2.1.3. Simuladores comerciales

Suelen ser soluciones profesionales desarrolladas por empresas dedicadas a la simulación industrial que dan soporte a las necesidades de otras empresas las cuales hacen uso de vehículos no tripulados.

2.1.3.1. VMAX Simulator

El simulador de proyectos Vmax es una solución de simulación integral para la industria marina, utilizada para desarrollar diversas aplicaciones de simulación.

Al proporcionar una funcionalidad estándar para cualquier equipo de ROV, VMAX permite una simulación precisa del equipo en un entorno submarino y es utilizado por operadores, empresas de servicios y firmas de ingeniería para ejecutar simulaciones de situaciones complejas en un entorno submarino.

Este software se utiliza para que los pilotos y profesionales lo usen en sus ensayos y sesiones de entrenamiento. Las empresas de ingeniería y los operadores de ROV validan los diseños de vehículos y comportamiento de estos con este tipo de simulaciones antes de realizar pruebas in situ. [20]

Ilustración 4 VMAX Simulator

Como ventaja podemos observar la gran profesionalidad del equipo y su gran precisión y ajuste con el comportamiento real. Sin embargo, tiene el coste es muy alto y la empresa tan solo proporciona la simulación de los modelos que vende ella misma, por lo que no tiene mucho potencial de universalización, a diferencia de los simuladores de código abierto.

2.1.3.2. Unicom

Unicom es una empresa escocesa dedicada a la simulación y animación 3D en los campos de extracción de petróleo, defensa e ingeniería manufacturera. Entre las diferentes herramientas que ofrece podemos encontrar un simulador de vehículos submarinos, el cual puede ser tanto personalizable para un modelo específico o usar uno de los estandarizados. Pese a que es un programa de pago pensado para grandes empresas con gran presupuesto, tiene la ventaja de que podemos usar una versión de prueba directamente desde su página web para probar su funcionamiento, la cual es muy recomendable visitar para tomar referencias de un simulador profesional. [21] [22]

Ilustración 5 Unicom

2.2. Sistemas de control ROV

El diseño de sistemas de control de cualquier parámetro (velocidad, orientación, posición absoluta…) en un vehículo submarino no es un asunto trivial, si no que en muchos casos puede requerir soluciones complejas. Existen multitud de métodos que han sido usados para aproximarse a la solución de este problema. En los sucesivos apartados exploraremos algunos de ellos de manera superficial ya que su estudio en profundidad daría para un TFG dedicado.

Para poder entender los diferentes métodos de diseño de controladores antes debemos introducir brevemente el concepto de sistema de control.

Dentro de la ingeniería de sistemas, un sistema de control es un conjunto de dispositivos encargados de administrar, ordenar, dirigir o regular el comportamiento de otro sistema, con el fin de reducir las probabilidades de fallo y obtener los resultados deseados. Por lo general, se usan sistemas de control industriales en procesos de producción industriales para controlar equipos o máquinas.

Existen dos clases comunes de sistemas de control, sistemas de lazo abierto y sistemas de lazo cerrado. En los sistemas de control de lazo abierto la salida se genera dependiendo de la entrada; mientras que en los sistemas de lazo cerrado la salida depende de las consideraciones y correcciones realizadas por la retroalimentación. [23]

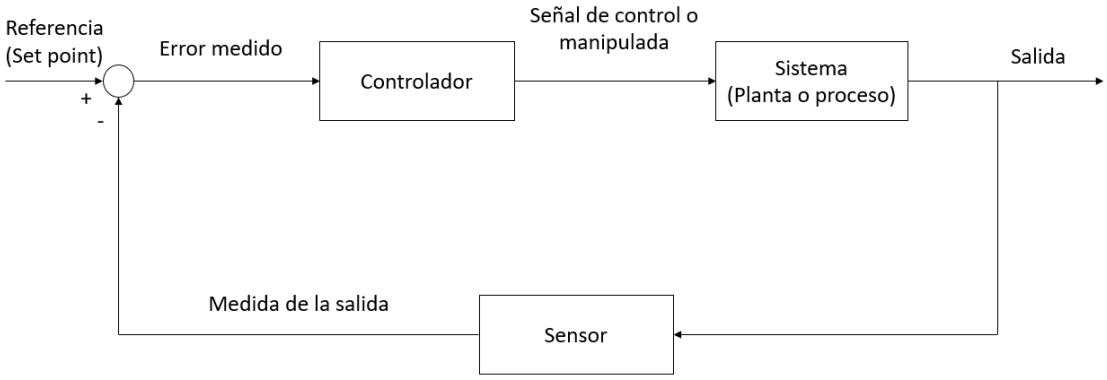


Ilustración 6 Sistema de control en lazo cerrado

2.2.1. Control mediante visión artificial

En la actualidad se está trabajando intensamente en el control de vehículos submarinos mediante la visión artificial. La visión artificial es una disciplina científica que incluye métodos para adquirir, procesar y analizar imágenes del mundo real con el fin de producir información que pueda ser tratada por una máquina. [24]

Podemos tomar como ejemplo el artículo publicado por G.L. Foresti, S. Gentili y M. Zampato: *A vision-based system for autonomous underwater vehicle navigation* [25]. Este artículo describe el trabajo para el diseño y desarrollo de un vehículo submarino autónomo (AUV), guiando su navegación mediante un sistema de visión artificial. Las tareas para las cuales se diseñó el vehículo son el estudio de fondos marinos y mantenimiento de tuberías submarinas. Se presenta un sistema basado en visión artificial para el vehículo submarino autonomo. Se realiza la

detección de bordes de tuberías submarinas, así como su eje de simetría. El método adoptado para la detección de bordes consta de dos pasos:

1) se aplica una red neuronal de retropropagación para segmentar la imagen submarina en diferentes regiones;

2) para cada región, se extraen el segmento de mejor ajuste y los parámetros relacionados.

Dado que la información sobre qué regiones de la imagen son los bordes correctos de la tubería no depende solo de las características de una sola región, sino también de las relaciones entre regiones, se analizan todos los pares de regiones posibles para determinar cuál es la correcta. Finalmente, también se obtuvieron resultados satisfactorios para tuberías parcialmente cubiertas por arena.

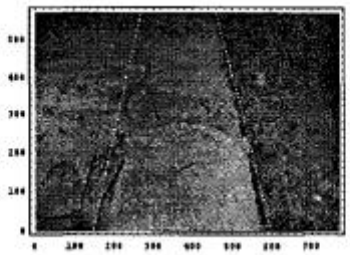


Ilustración 7 Ejemplo imagen captada por vision artificial

Otro ejemplo en este campo es la investigación llevada a cabo por S. Matsumoto e Y. Ito y plasmada en el artículo *Real-time vision-based tracking of submarine-cables for AUV/ROV*, este artículo describe un software que rastrea cables submarinos en secuencias de imágenes. Al principio, crea un conjunto de tres tipos de imágenes filtradas a partir de la original. A continuación, al aplicarles umbrales, se extrae una imagen de borde. Los umbrales se ajustan de forma dinámica y adaptativa. A partir de los resultados de la transformación de Hough de la imagen de borde, se seleccionan los candidatos de borde de cable.

Luego, se evalúa la probabilidad de que los candidatos sean una línea. Finalmente, se evalúa la distancia de los candidatos al cable en la imagen anterior. El software se probó con éxito para su rendimiento al aplicarlo a las secuencias de imágenes de cables submarinos reales. También se demostró mediante experimentos que, según la posición del cable estimada por el software, el ROV (vehículo operado de forma remota) podía rastrear un cable curvo tendido en el fondo del tanque de prueba. [26]

2.2.2. Controlador PID no lineal

La forma tradicional de aproximarse al control de un proceso no lineal es basar el diseño del controlador en un modelo de la planta linealizado sobre un punto. Es esperable que estos controladores tengan una buena respuesta mientras la planta trabaje en un rango suficientemente cercano a ese punto de linealización. Cuando la planta se encuentra operando en un rango más amplio este procedimiento puede ser repetido para distintos puntos de referencia y el controlador ajustado a los parámetros correspondientes conforme cambian las condiciones de funcionamiento.

Recientemente ha habido un creciente interés en la automatización de este procedimiento mediante el *self-tuning* o los controles adaptativos. Estas aproximaciones son particularmente valiosas en procesos los cuales el modelo no es conocido con exactitud [27].

2.2.3. Control en modo de deslizamiento

Otro método de control el cual está siendo objeto de estudio y desarrollo para su aplicación en vehículos submarinos no tripulados es el control en modo de deslizamiento.

El control en modo de deslizamiento o control en modo deslizante, en inglés *sliding mode controller* (SMC) es un método de control no lineal que altera la dinámica de un sistema no lineal mediante la aplicación de una señal de control discontinua que obliga al sistema a "deslizarse" a lo largo de una sección transversal del comportamiento normal del sistema [28].

Las técnicas de control en modo de deslizamiento fueron concebidas en un primer momento para solucionar problemas de estabilización de sistemas, en aplicaciones como control de la velocidad de motores, control de procesos químicos, servomecanismos, etc. En estas aplicaciones, el objetivo del control es que la salida del sistema adquiera un valor de referencia estable y constante. No obstante, las funciones del control en modo deslizante han ido extendiéndose hacia otros objetivos, como, por ejemplo, seguimiento y generación de señales, control óptimo, control adaptativo, etc. [29]

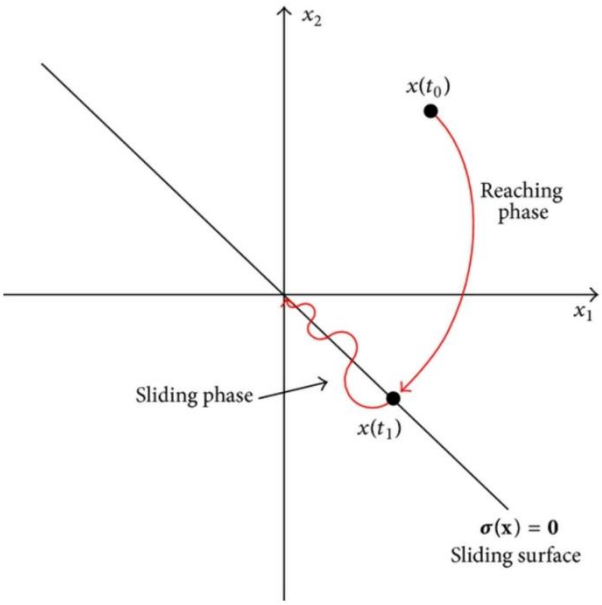


Ilustración 8 Esquema grafico del control en modo de deslizamiento

En el artículo de García Valdovinos [30] se desarrolla un simulador para el vehículo Kaxan, el cual capta la información mediante un sensor acústico y sensores inerciales. Se presenta El modelo hidrodinámico no lineal completo de seis grados de libertad con sus parámetros, la arquitectura hardware/software del ROV, simulaciones numéricas en Matlab de un control de modo deslizante de segundo orden sin modelo junto con las corrientes oceánicas como perturbaciones y con el modelado de las dinámicas de los propulsores y un entorno virtual para visualizar el movimiento del Kaxan ROV.

El artículo de Salgado Jimenez se explora también la implementación de un sistema de control en modo deslizante a un ROV, pero se sabe que estos métodos son susceptibles al ruido, que es una señal de alta frecuencia inducido por interruptores de control. Para evitar este problema, se propone un sistema de control en modo deslizante de orden superior. La característica principal de este nuevo sistema es que mantiene las principales ventajas del SMC estándar, eliminando así los efectos del ruido. [31]

2.2.4. Redes neuronales para la parametrización de PID

Durante décadas, los controladores tipo PID (Proporcional + Integral + Derivado) se han utilizado con éxito en la investigación industria para muchos tipos de plantas. Esto es gracias a su simplicidad y rendimiento adecuado en plantas lineales o linealizadas, y bajo ciertas condiciones, en plantas no lineales. Se han propuesto varios enfoques de ajuste de ganancia de controlador PID en la literatura en las últimas décadas, en la mayoría de ellos el ajuste era único sin variaciones durante el funcionamiento normal del sistema. Sin embargo, en aquellos casos en

que las plantas están sujetas a cambios paramétricos continuos o perturbaciones externas, el ajuste de ganancias continuo es una opción deseable.

Este es el caso de los ROV submarinos modulares donde los parámetros (peso, flotabilidad, masa agregada, entre otros) cambian de acuerdo con la herramienta con la que está equipado. En la práctica, se dedica una cantidad de tiempo para ajustar las ganancias PID de un ROV. Una vez que se ha logrado el mejor conjunto de ganancias, el ROV está listo para trabajar. Sin embargo, cuando el vehículo cambia su herramienta o está sujeto a las corrientes oceánicas, su rendimiento se deteriora ya que el conjunto fijo de ganancias ya no es válido para las nuevas condiciones.

Por lo tanto, se debe implementar un algoritmo de ajuste de ganancias PID para superar este problema. En el artículo [32], se propone un controlador tipo PID de autoajuste basado en redes neuronales. Las redes neuronales desempeñan el papel de estimar automáticamente el conjunto adecuado de ganancias PID que logra la estabilidad del sistema. Esta red neuronal ajusta continuamente las ganancias del controlador para lograr el error de seguimiento de posición más pequeño.

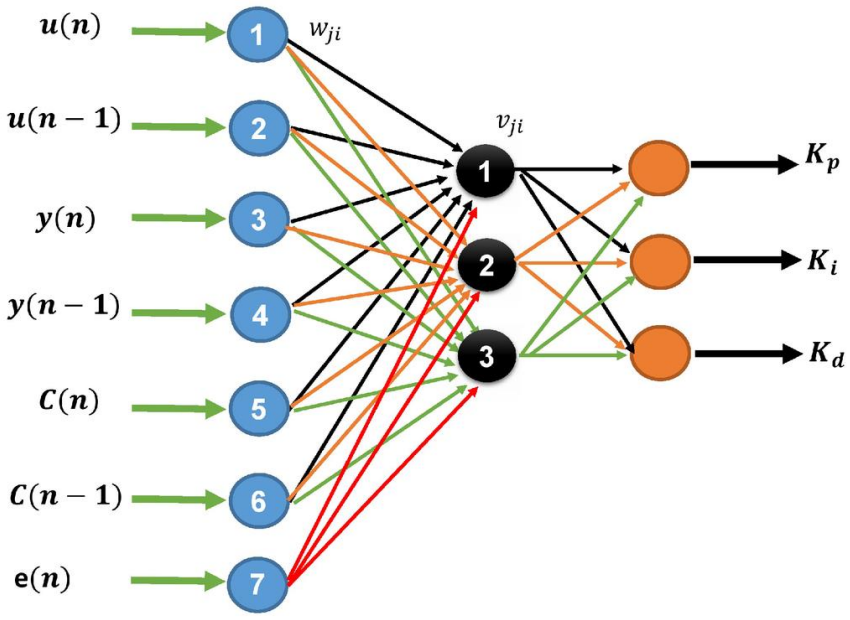


Ilustración 9 Esquema gráfico de una red neuronal

2.2.5. Control sincronizado de múltiples ROVs

Para aplicaciones específicas, por ejemplo, el muestreo oceánico, mapeo de fondos submarinos, barrido de minas, levantamiento de suelos oceánicos… es beneficioso llevar a cabo las misiones a través de la cooperación de múltiples AUV. Algunas de las ventajas son el aumento de la eficiencia, el aumento del área de servicio y la redundancia en caso de fallo.

La idea fundamental es utilizar AUVs relativamente baratos, simples y pequeños en lugar de costosos AUVs. Estos estarán especializados para resolver cooperativamente tareas subacuáticas difíciles o complejas. Se espera, por ejemplo, que los AUVs cooperativos que forman una red móvil de sensores, puedan superar a un solo vehículo grande con múltiples sensores, cuando el objetivo es escalar el gradiente de un campo ambiental [33]. Un problema fundamental en la cooperación de múltiples AUV es el control de la formación, la estructura en la cual los AUV mantienen una configuración espacial deseada y al mismo tiempo completan las tareas asignadas. [34]

Capítulo 3. Modelado de Vehículos Submarinos

3.1. Sistemas de coordenadas, posicionamiento y cinemática

Analizaremos el movimiento de un vehículo submarino con seis grados de libertad para determinar su posición y orientación en el espacio tridimensional y en el tiempo.

Las tres primeras coordenadas independientes (x,y,z) hacen referencia a la posición del vehículo, y los tres restantes (θ,ϕ,ψ) se corresponden con la orientación del mismo respecto a un sistema de referencia fijo a la tierra, llamado Global. Estas coordenadas son inerciales.

Usando la notación tradicional de usada en vehículos navales, estas variables se nombran como: *avance* (x), *deriva* (y), *arfada* (z), *balanceo* (θ), *cabeceo* (ϕ) y *guiñada* (ψ).

Por convenio se considera que la dirección "x" positiva se toma hacia adelante, la "y" se toma hacia la derecha y la "z" hacia abajo. Por la regla de la mano de derecha se obtienen la dirección positiva de los ángulos de rotación.

A continuación, para continuar con el análisis de los movimientos del UUV definiremos los sistemas de referencia. Para ello supondremos dos ejes de coordenadas, uno que estará fijo en la superficie del mar, al que llamaremos Global. Al ser la influencia de la rotación de la Tierra tan pequeña en nuestro análisis, lo consideraremos un sistema inercial a todos los efectos, por lo que podremos aplicar las leyes del movimiento de Newton. En general expresaremos la posición y orientación del submarino en el sistema de referencia Global, mientras que las velocidades lineales y angulares en el sistema relativo.

Posiciones y ángulos respecto al sistema de ref. global	Movimientos relativos al vehículo	Velocidades lineales y angulares respecto al sistema de ref. local	Fuerzas y momentos respecto al sistema de ref. local
x	Avance	u	X
y	Deriva	v	Y
z	Arfada	w	Z
Θ	Balanceo	p	K
Φ	Cabeceo	q	M
ψ	Guiñada	r	N

Tabla 1 Sistema de referencia en un UUV

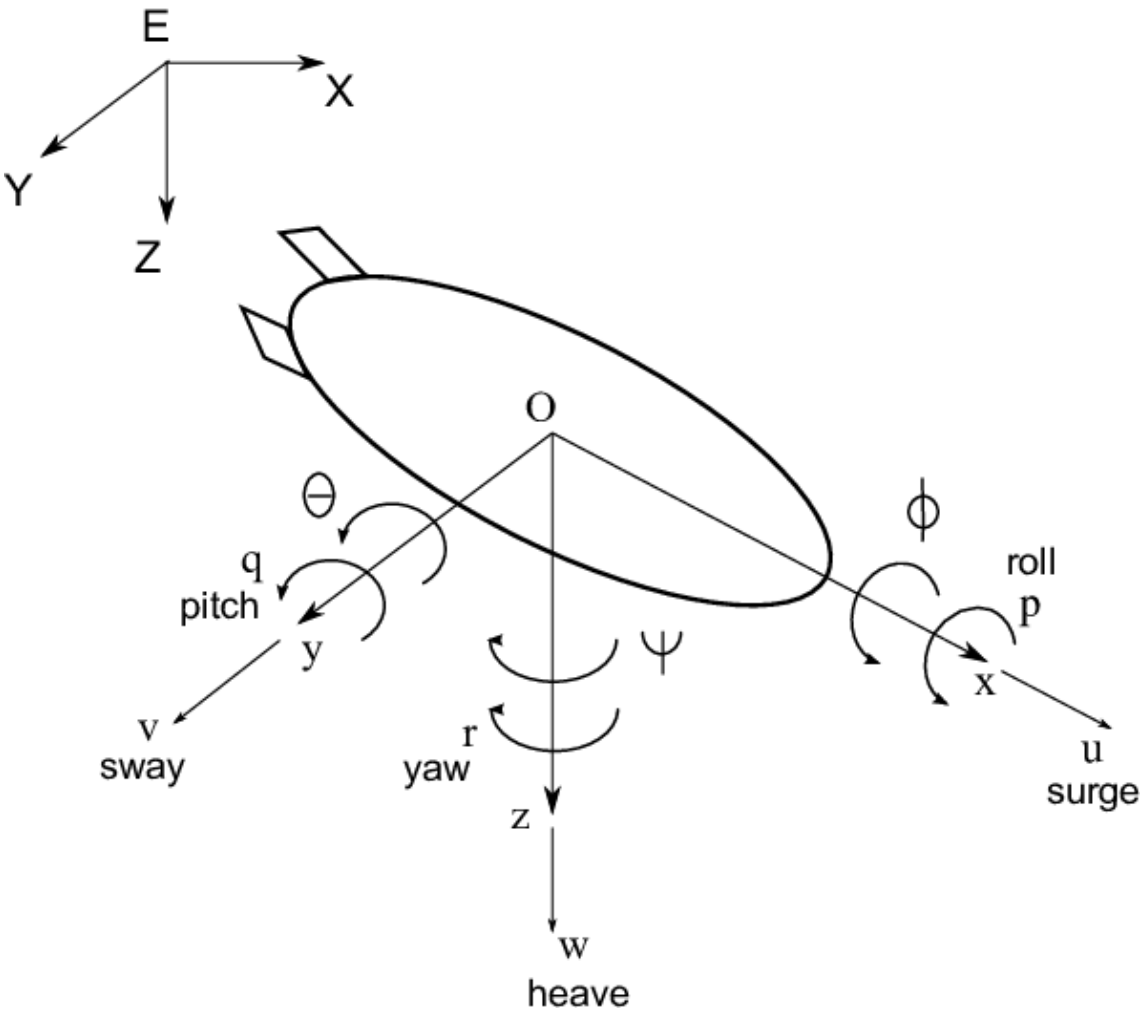


Ilustración 10 Sistema de referencia en un vehículo submarino

Definimos los vectores unitarios (î, ĵ, k̂) referenciados al eje origen del sistema de referencia Global. Por lo tanto, la posición del vehículo se puede escribir como combinación lineal de estos tres vectores unitarios.

$$r_{O`} = X\,\vec{\imath} + Y\,\vec{\jmath} + Z\vec{k}$$

El otro sistema de referencia, al que llamaremos Local será solidario al vehículo, su origen (O') se establecerá en el UUV, atendiendo a criterios de simetría, pero por lo general no tiene por qué coincidir con el centro de gravedad (c.d.g), ni con el centro de carena (c.d.c). Los ejes x', y' y z' coincidirán con los principales ejes de inercia, como se observa en la figura. Este sistema de referencia será no inercial debido a que al ser solidario al vehículo estará sometido a aceleraciones y rotaciones.

Al igual que en el caso anterior definimos los vectores unitarios (î', ĵ', k̂'), gracias a los cuales podemos expresar cualquier punto del espacio como combinación lineal de estos.

Supondremos que todas las fuerzas y momentos que actúan sobre el vehículo autónomo se aplican sobre el centro de gravedad, ya que es donde se ubican los ejes principales de inercia. Por otra parte, las fuerzas hidrodinámicas se calcularán des desde el centro de carena.

Los vectores de posición del c.d.g y c.d.c relativos al origen del sistema local los definimos como $\overrightarrow{\rho_G}$ y $\overrightarrow{\rho_B}$ respectivamente. Sus posiciones vienes definidas por:

$$\overrightarrow{\rho_G} = [x_G\vec{\imath} + y_G\vec{\jmath} + z_G\vec{k}] \quad ; \quad \overrightarrow{\rho_B} = [x_B\vec{\imath} + y_B\vec{\jmath} + z_B\vec{k}]$$

[10], [35]

3.1.1. Ángulos de Euler

La relación entre dos sistemas cartesianos cualquiera se puede ver como un desplazamiento del centro de coordenadas y una rotación de los ejes. En este caso nos fijaremos únicamente en la orientación. Según el teorema de Euler, cualquier rotación puede ser descrita únicamente mediante tres parámetros. Es decir, que para dar a un objeto una orientación tiene que ser sometido a tres rotaciones.

Cualquier vector de posición r, en un sistema de referencia viene dado por r0=x0,y0,z0. Si rotamos sobre el Angulo ϕ, sobre el eje x, las coordenadas de ese mismo vector se pueden obtener mediante trigonometría, lo que nos da las siguientes igualdades:

Esta relación puede expresarse de forma matricial mediante:

$$r_1 = [R]^{-1}_{x_0,\emptyset} r_o$$

Siendo R:

$$R_{x_0,\emptyset} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{bmatrix}$$

Donde R es una matriz ortogonal en la que su inversa es igual a su traspuesta.

$$[R]^T = [R]^{-1}$$

Con un razonamiento análogo en los dos ejes restantes obtenemos:

$$R_{y_0,\theta} = \begin{bmatrix} cos\theta & 0 & -sin\theta \\ 0 & 1 & 0 \\ sin\theta & 0 & cos\theta \end{bmatrix}$$

$$R_{z_0,\psi} = \begin{bmatrix} cos\psi & -sin\psi & 0 \\ sin\psi & cos\psi & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Por lo tanto, podemos rotar un eje de coordenadas estos tres ángulos aplicando la siguiente matriz de rotación:

$$[R] = [R]^T{}_{x_0,\emptyset} [R]^T{}_{y_0,\theta} [R]^T{}_{z_0,\psi}$$

Finalmente operando obtenemos:

$$[R] = \begin{bmatrix} cos\psi cos\theta & cos\psi sin\theta sin\phi - sin\psi cos\phi & cos\psi sin\theta cos\phi + sin\psi sin\phi \\ sin\psi cos\theta & sin\psi sin\theta sin\phi + cos\psi cos\phi & sin\psi sin\theta cos\phi - cos\psi sin\phi \\ -sin\theta & cos\theta sin\phi & cos\theta cos\phi \end{bmatrix}$$

De esta forma ponemos transformar cualquier vector de un sistema de referencia a otro gracias a esta matriz

$$r_{i'j'k'} = [R]^T r_{ijk}$$

Sabiendo esto podemos comenzar a analizar las trasformaciones que nos permitirán hacer el análisis cinemático del vehículo.

[10]

3.2. Análisis cinemático

Mediante ángulos de Euler

Para simplificar la notación esto definiremos los siguientes vectores de utilidad.

$$\eta = [\eta_1^T, \eta_2^T] \qquad \eta_1 = [x, y, z]^T \qquad \eta_2 = [\phi, \theta, \psi]^T$$

Donde η es el vector posición y orientación del vehículo referido al sistema de referencia Global

$$\upsilon = [\upsilon_1^T, \upsilon_2^T] \qquad \upsilon_1 = [u, v, w]^T \qquad \upsilon_2 = [p, q, r]^T$$

υ es el vector de velocidad lineal y angular en el sistema Local, es decir, el solidario al UUV.

$$\tau = [\tau_1^T, \tau_2^T] \qquad \tau_1 = [X, Y, Z]^T \qquad \tau_2 = [K, M, N]^T$$

Y por último τ denota las fuerzas y momentos que actúan sobre el vehículo en el sistema de referencia Local.

Para pasar desde las velocidades medidas respecto al eje inercial "ṅ", a las referidas desde el eje solidario al vehículo "υ", debemos obtener la matriz de transformación $J(\eta_2)$, que solo dependerá de la orientación del sistema de referencia local:

$$\dot{\eta}_1 = J_1(\eta_2)\upsilon_1$$

La transformación inversa quedará definida por la siguiente expresión:

$$\upsilon_1 = J_1^{-1}(\eta_2)\dot{\eta}_1$$

En el Fossen [10] se demuestra que esa matriz es equivalente a la calculada anteriormente [R], por lo tanto:

$$[J_1] = \begin{bmatrix} cos\psi cos\theta & cos\psi sin\theta sin\phi - sin\psi cos\phi & cos\psi sin\theta cos\phi + sin\psi sin\phi \\ sin\psi cos\theta & sin\psi sin\theta sin\phi + cos\psi cos\phi & sin\psi sin\theta cos\phi - cos\psi sin\phi \\ -sin\theta & cos\theta sin\phi & cos\theta cos\phi \end{bmatrix}$$

Y mantiene la propiedad de ser su inversa igual a su traspuesta:

$$[J_1]^{-1} = [J_1]^T$$

El vector de velocidades angulares medidas desde el eje Local "υ_2" y el vector de velocidades angulares de Euler "η_2" están relacionados por la matriz de transformación J_2.

$$\dot{\eta}_2 = J_2(\eta_2)\upsilon_2$$

La cual tiene es de la siguiente forma:

$$J_2^{-1}(\eta_2) = \begin{bmatrix} 1 & 0 & -sin\theta \\ 0 & cos\phi & cos\theta sin\phi \\ 0 & -sin\phi & cos\theta cos\phi \end{bmatrix} \qquad J_2(\eta_2) = \begin{bmatrix} 1 & sin\phi tan\theta & cos\phi tan\theta \\ 0 & cos\phi & -sin\phi \\ 0 & \frac{sin\phi}{cos\theta} & \frac{cos\phi}{cos\theta} \end{bmatrix}$$

Podemos observar que esta matriz de transformación tiene dos peculiaridades, la primera que es que su traspuesta no es igual a su inversa.

$$J_2^{-1}(\eta_2) \neq J_2^T(\eta_2)$$

La segunda es que para un pitch "θ" de 90° la matriz está indefinida. Para vehículos de superficie esto no es un problema, sin embargo, en vehículos submarinos como el nuestro si lo es ya que en muchas ocasiones pueden trabajar cerca de esa singularidad.

Este problema se puede solucionar de dos formas, la primera es definir con dos representaciones de los ángulos de Euler con la singularidad en diferentes puntos, y usar en cada momento la conveniente. La otra posibilidad es el uso de la representación mediante cuaterniones, la cual será la que desarrollaremos a continuación.

En resumen, las ecuaciones cinemáticas pueden ser expresadas de forma vectorial como:

$$\begin{bmatrix}\dot{\eta}_1\\ \dot{\eta}_2\end{bmatrix} = \begin{bmatrix}J_1(\eta_2) & 0_{3X3}\\ 0_{3X3} & J_2(\eta_2)\end{bmatrix}\begin{bmatrix}\upsilon_1\\ \upsilon_2\end{bmatrix} \quad \leftrightarrow \quad \dot{\eta} = J(\eta)\upsilon$$

[10]

3.2.1. Cuaterniones

Un cuaternión se define como un numero complejo formado por cuatro unidades (i,j,k,1) acompañadas de un parámetro real, donde (i,j,k) son los tres vectores ortonormales.

$$q = q_1 i + q_2 j + q_3 k + q_4$$

Por lo tanto, un cuaternión se puede ver como la combinación lineal de un vector y un escalar.

$$q = q_0 + q_4$$

Donde

$$q_0 = [q_1, q_2, q_3]^T$$

Si $q_4 = 0$, q es un numero puramente imaginario llamado vector cuaternión. De igual manera, si $q_0 = 0$ lo llamaremos cuaternión escalar. Usando cuaterniones podemos describir el movimiento del sistema de referencia no inercial (Local) respecto al inercial (Global)

Los parámetros de Euler se definen como:

$$\varepsilon = [\varepsilon_1, \varepsilon_2, \varepsilon_3]^T = \lambda \sin\frac{\beta}{2}$$

$$\eta = \cos\frac{\beta}{2}$$

Siendo lambda:

$$\lambda = \pm\frac{\varepsilon}{\sqrt{\varepsilon^T\varepsilon}}$$

$$\sqrt{\varepsilon^T\varepsilon} \neq 0$$

Consecuentemente los parámetros de Euler se pueden escribir como.

$$e = \begin{bmatrix}\varepsilon_1\\ \varepsilon_2\\ \varepsilon_3\\ \eta\end{bmatrix} = \begin{bmatrix}\lambda \sin\frac{\beta}{2}\\ \cos\frac{\beta}{2}\end{bmatrix}; \qquad 0 \leq \beta \leq 2\pi$$

Esto implica que se debe cumplir que $e^T e = 1$, es decir:

$$\varepsilon_1^2 + \varepsilon_2^2 + \varepsilon_3^2 + \eta^2 = 1$$

La transformación que relaciona la velocidad lineal entre el sistema de referencia inercial y el solidario al vehículo es la siguiente:

$$\dot{\eta}_1 = E_1(e)\upsilon_1$$

Donde E_1 (e) es:

$$E_1(e) = \begin{bmatrix} 1-2(\varepsilon_2^2+\varepsilon_3^2) & 2(\varepsilon_1\varepsilon_2-\varepsilon_3\eta) & 2(\varepsilon_1\varepsilon_3+\varepsilon_2\eta) \\ 2(\varepsilon_1\varepsilon_1+\varepsilon_1\eta) & 1-2(\varepsilon_1^2+\varepsilon_1^2) & 2(\varepsilon_2\varepsilon_3-\varepsilon_1\eta) \\ 2(\varepsilon_1\varepsilon_3-\varepsilon_2\eta) & 2(\varepsilon_2\varepsilon_3+\varepsilon_1\eta) & 1-2(\varepsilon_1^2+\varepsilon_2^2) \end{bmatrix}$$

Igual que en la representación por ángulos de Euler, la inversa de la matriz de transformación es igual a su traspuesta:

$$E_1^{-1}(e) = E_1^T(e)$$

También se puede obtener que

$$\dot{e} = E_2(e)\upsilon_2$$

Donde $\upsilon_2 = [p, q, r]^T$ y:

$$E_2(e) = \frac{1}{2}\begin{bmatrix} \eta & -\varepsilon_3 & \varepsilon_2 \\ \varepsilon_3 & \eta & -\varepsilon_1 \\ -\varepsilon_2 & \varepsilon_1 & \eta \\ -\varepsilon_1 & -\varepsilon_2 & -\varepsilon_3 \end{bmatrix}$$

$$E_2^T(e)E_2^T(e) = \frac{1}{4}I_{3X3}$$

Siendo I_{3X3} la matriz identidad de tres dimensiones.

Por lo tanto, las ecuaciones cinemáticas del movimiento se pueden expresar como:

$$\begin{bmatrix} \dot{\eta} \\ \dot{e} \end{bmatrix} = \begin{bmatrix} E_1(e) & 0_{3X3} \\ 0_{4X3} & E_2(e) \end{bmatrix} \begin{bmatrix} \upsilon_1 \\ \upsilon_2 \end{bmatrix} \leftrightarrow \dot{\eta}_E = E(\eta_E)\upsilon$$

Donde $\eta_E = [x, y, z, \varepsilon_1, \varepsilon_2, \varepsilon_3, \eta]^T$.

Los parámetros de Euler pueden ser obtenidos desde la matriz de transformación con un sencillo algoritmo el cual puede ser consultado en el libro de Fossen [10].

De igual modo es simple transformar los parámetros de Euler a los ángulos de Euler, gracias a esto en el simulador trabajaremos en ocasiones con cuaterniones y en otras con matrices de transformación, usando siempre la solución más sencilla entre los dos métodos de cálculo además los cuaterniones nos servirán para evitar posibles fuentes de problemas como pueden ser las singularidades que ocurren al tener un pitch de 90° si usamos las matrices de transformación.

3.3. Análisis dinámico

El análisis dinámico del ROV puede ser expresado en una única ecuación matricial donde están plasmadas todas las fuerzas que influyen en su comportamiento.

$$\mathrm{M}\dot{\upsilon} + \mathrm{C}(\upsilon)\upsilon + \mathrm{D}(\upsilon)\upsilon + \mathrm{g}(\eta) = \tau$$

Esta ecuación matricial consta de los siguientes términos:

-M: Matriz de masas

-$C(v)$: Matriz de Coriolis en función de la velocidad

-$D(v)$: Matriz de resistencia hidrodinámica

-$g(\eta)$: Vector de flotación función de la posición

-τ: Vector de fuerzas de propulsión

-v: Vector de velocidad

-$\dot{v}$: Vector de aceleración del vehículo

A continuación, realizaremos una descripción de cada uno de estos factores que afectan a la dinámica del vehículo. [35]

3.3.1. Matriz de masas

La matriz de masas (M) está formada por la matriz de la masa y la inercia del vehículo (MRB), así como de la matriz de la masa virtual (M_A).

$$M = M_{RB} + M$$

La matriz M_{RM} tiene en cuenta únicamente los efectos inerciales que produce el sólido rígido, pero también hay que tener en cuenta en las fuerzas de resistencia hidrodinámica dependientes de la aceleración. Se incluye en la matriz de masas ya que a la hora de calcular su efecto este se puede entender como la masa de fluido que arrastra al moverse el vehículo, aunque esta no sea la verdadera naturaleza de la fuerza. El efecto de este fenómeno suele tener el mismo orden de magnitud que el de la propia masa del vehículo, aunque algo inferior.

La matriz MRB tiene la siguiente estructura.

$$M_{RB} = \begin{bmatrix} m & 0 & 0 & 0 & mz_G & -my_G \\ 0 & m & 0 & -mz_G & 0 & mx_G \\ 0 & 0 & m & my_G & -mx_G & 0 \\ 0 & -mz_G & my_G & Ix & Ixy & Ixz \\ mz_G & 0 & -mx_G & Ixy & Iy & Iyz \\ -my_G & mx_G & 0 & Ixz & Iyz & Iz \end{bmatrix}$$

La matriz MA se compone con una serie de coeficientes que dependen de la masa de fluido que arrastra el vehículo en su movimiento debido a su forma y que, al estar el vehículo completamente sumergido, podemos considerar constantes.

La matriz de masa virtual tiene la siguiente forma.

$$M_A = \begin{bmatrix} X_{\dot{u}} & X_{\dot{v}} & \mathrm{X}_{\dot{w}} & X_{\dot{p}} & X_{\dot{q}} & X_{\dot{r}} \\ Y_{\dot{u}} & Y_{\dot{v}} & \mathrm{Y}_{\dot{w}} & Y_{\dot{p}} & Y_{\dot{q}} & Y_{\dot{r}} \\ \mathrm{Z}_{\dot{u}} & Z_{\dot{v}} & \mathrm{Z}_{\dot{w}} & Z_{\dot{p}} & Z_{\dot{q}} & Z_{\dot{r}} \\ K_{\dot{u}} & K_{\dot{v}} & \mathrm{K}_{\dot{w}} & K_{\dot{p}} & K_{\dot{q}} & K_{\dot{r}} \\ M_{\dot{u}} & M_{\dot{v}} & \mathrm{M}_{\dot{w}} & M_{\dot{p}} & M_{\dot{q}} & M_{\dot{r}} \\ N_{\dot{u}} & N_{\dot{v}} & \mathrm{N}_{\dot{w}} & N_{\dot{p}} & N_{\dot{q}} & N_{\dot{r}} \end{bmatrix}$$

Dicha matriz puede ser simplificada en función de la simetría del vehículo de estudio, de modo que muchos de los coeficientes se anulan.

Hallar las componentes de esta matriz no es sencillo, ya que se deben realizar pruebas experimentales o complejas simulaciones informáticas para obtenerlas.

3.3.2. Matriz de Coriolis

Mediante la matriz de Coriolis podemos calcular los efectos producidos por la fuerza de Coriolis y por las fuerzas centrípetas.

La matriz de Coriolis para un sólido rígido que se mueve a través de un fluido ideal tiene la siguiente forma:

$$C_A(v) = \begin{bmatrix} 0_{3x3} & -S(A_{11}v_1 + A_{12}v_2) \\ -S(A_{11}\,v_1 + A_{12}v_2) & -S(A_{21}v_1 + A_{22}v_2) \end{bmatrix}$$

Siendo 0_{3x3} la matriz nula de 3x3, y A las submatrices de la matriz de masas

$$M = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix}$$

Siendo S la transformada de un vector lineal tal que

$$a = (a_1, a_2, a_3)$$

$$S(a) = \begin{bmatrix} 0 & -a_3 & a_2 \\ a_3 & 0 & -a_1 \\ -a_2 & a_1 & 0 \end{bmatrix}$$

Y finalmente siendo los vectores v_1 y v_2 los vectores de 3x1 de velocidad lineal y angular.

$$v = \begin{bmatrix} u \\ v \\ w \\ p \\ q \\ r \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix}$$

3.3.3. Matriz de resistencia hidrodinámica

Se sabe que el amortiguamiento de un vehículo que viaja bajo el agua es acoplado, no lineal y que se forma por dos términos, siendo uno cuadrático y otro lineal. Además, los términos mayores de segundo orden tampoco se tendrán en cuenta, descartando así términos de orden superior. También, se considera que los coeficientes van a tener valores constantes para el rango de velocidades en el que el vehículo se desplaza. Por último, es importante conocer la geometría del vehículo porque debido a esto se pueden simplificar algunos coeficientes.

Dentro de este subapartado, se puede dividir además en las fuerzas y momentos debidos al movimiento axial y en las fuerzas y momentos por el movimiento de rotación.

3.3.3.1. Fuerzas y momentos debidos al movimiento axial

Para el primer caso, las fuerzas y momentos debidos al movimiento axial son cuando el vehículo se mueve en una dirección del sistema de referencia, pudiendo ser X, Y o Z.

$$F_{ArrastreAvanceX} = X_{u|u|} \cdot u|u| + X_u \cdot u$$

El arrastre viscoso siempre es contrario al movimiento del vehículo, por lo que en todas las ecuaciones se va a considerar $u|u|$ en lugar de u^2.

De manera análoga se representan las fuerzas en los ejes Y Z.

$$F_{ArrastreDerivaY} = X_{v|v|} \cdot v|v| + X_v \cdot v$$

$$F_{ArrastreArfadaZ} = X_{w|w|} \cdot w|w| + X_w \cdot w$$

Además, dependiendo la simetría o falta de ella en el vehículo, se puede producir momentos debidos a este movimiento axial:

$$M_{ArrastreAvanceM} = M_{u|u|} \cdot u|u| + M_u \cdot u$$

$$K_{ArrastreDerivaK} = K_{v|v|} \cdot v|v| + K_v \cdot v$$

$$N_{ArrastreArfadaN} = N_{w|w|} \cdot w|w| + N_w \cdot w$$

3.3.3.2. Fuerzas y momentos debidos a la rotación

En el segundo caso tenemos las fuerzas y momentos debidos a la rotación, estas se obtienen cuando el vehículo gira respecto a alguna de las direcciones con una velocidad angular determinada. De este modo, se producen movimientos de balance, cabeceo y guiñada.

$$K_{ArrastreBalanceK} = K_{p|p|} \cdot p|p| + K_p \cdot p$$

$$M_{ArrastreCabeceoM} = M_{q|q|} \cdot q|q| + M_q \cdot q$$

$$N_{ArrastreGuiñadaN} = N_{r|r|} \cdot r|r| + N_r \cdot r$$

Además, también aparecen las siguientes fuerzas:

$FArrastre\ Balance\ Y = Ypp \cdot pp$

$FArrastre\ Balance\ Z = Zpp \cdot pp$

$FArrastre\ Cabeceo\ X = Xqq \cdot qq$

$FArrastre\ Cabeceo\ Z = Zqq \cdot qq$

$FArrastre\ Guiñada\ X = Xrr \cdot rr$

$FArrastre\ Guiñada\ Y = Yrr \cdot rr$

[5]

De este modo la matriz de resistencia hidrodinámica se puede expresar como:

$$\mathrm{D}(\upsilon) = \mathrm{D}_{\mathrm{l}} + D_{nl} \cdot |\vec{\upsilon}| + D_u \cdot |u| + D_v \cdot |v| + D_w \cdot |w|$$

Siendo:

D_{l} : matriz de resistencia hidrodinámica lineal.

D_{nl}: matriz de resistencia hidrodinámica no lineal.

D_u: matriz de parámetros cruzados en el sentido del eje x.

D_v: matriz de parámetros cruzados en el sentido del eje y.

D_w: matriz de parámetros cruzados en el sentido del eje z.

Estas matrices pueden ser halladas por métodos experimentales, analíticos o mediante simulaciones informáticas con métodos CFD. La denominación de cada parámetro será la siguiente:

3.3.3.3. Matriz de resistencia hidrodinámica lineal:

$$\mathrm{D}_{\mathrm{l}} = \begin{bmatrix} Xu & Yu & Zu & Ku & Mu & Nu \\ Xv & Yv & Zv & Kv & Mv & Nv \\ Xw & Yw & Zw & Kw & Mw & Nw \\ Xp & Yp & Zp & Kp & Mp & Np \\ Xq & Yq & Zq & Kq & Mq & Nq \\ Xr & Yr & Zr & Kr & Mr & Nr \end{bmatrix}$$

Xu - Coeficiente que relaciona linealmente la fuerza en la dirección del eje X con la velocidad en esa misma dirección.

Yu - Coeficiente que relaciona linealmente la fuerza en la dirección del eje Y con la velocidad en el eje X.

Xv - Coeficiente que relaciona linealmente la fuerza en la dirección del eje X con la velocidad en el eje Y.

Con este mismo criterio se definen el resto de las componentes de la matriz de resistencia hidrodinámica lineal.

3.3.3.4. Matriz de resistencia hidrodinámica no lineal.

$$\mathrm{D_{nl}} = \begin{bmatrix} Xuu & Yuu & Zuu & Kuu & Muu & Nuu \\ Xvv & Yvv & Zvv & Kvv & Mvv & Nvv \\ Xww & Yww & Zww & Kww & Mww & Nww \\ Xpp & Ypp & Zpp & Kpp & Mpp & Npp \\ Xqq & Yqq & Zqq & Kqq & Mqq & Nqq \\ Xrr & Yrr & Zrr & Krr & Mrr & Nrr \end{bmatrix}$$

Xu - Coeficiente que relaciona cuadráticamente la fuerza en la dirección del eje X con la velocidad en esa misma dirección.

Yu - Coeficiente que relaciona cuadráticamente la fuerza en la dirección del eje Y con la velocidad en el eje X.

Xv - Coeficiente que relaciona cuadráticamente la fuerza en la dirección del eje X con la velocidad en el eje Y.

Con este mismo criterio se definen el resto de las componentes de la matriz de resistencia hidrodinámica no lineal.

3.3.3.5. Matriz de parámetros cruzados en el sentido del eje x.

$$\mathrm{D_{u}} = \begin{bmatrix} Xuv & Yuv & Zuv & Kuv & Muv & Nuv \\ Xuw & Yuw & Zuw & Kuw & Muw & Nuw \\ Xup & Yup & Zup & Kup & Mup & Nup \\ Xuq & Yuq & Zuq & Kuq & Muq & Nuq \\ Xur & Yur & Zur & Kur & Mur & Nur \end{bmatrix}$$

Xuv - Coeficiente que relaciona la fuerza en la dirección del eje X con las velocidades en los ejes X e Y.

Yuv - Coeficiente que relaciona la fuerza en la dirección del eje Y con las velocidades en los ejes X e Y.

Xuw - Coeficiente que relaciona la fuerza en la dirección del eje X con las velocidades en los ejes X y Z.

Con este mismo criterio se definen el resto de las componentes de la matriz de parámetros cruzados.

3.3.3.6. Matriz de parámetros cruzados en el sentido del eje y.

$$\mathrm{D_v} = \begin{bmatrix} Xvu & Yvu & Zvu & Kvu & Mvu & Nvu \\ Xvw & Yvw & Zvw & Kvw & Mvw & Nvw \\ Xvp & Yvp & Zvp & Kvp & Mvp & Nvp \\ Xvq & Yvq & Zvq & Kvq & Mvq & Nvq \\ Xvr & Yvr & Zvr & Kvr & Mvr & Nvr \end{bmatrix}$$

Xvu - Coeficiente que relaciona la fuerza en la dirección del eje X con las velocidades en los ejes Y y X.

Yvu - Coeficiente que relaciona la fuerza en la dirección del eje Y con las velocidades en los ejes Y y X.

Xvw - Coeficiente que relaciona la fuerza en la dirección del eje X con las velocidades en los ejes Y y Z.

Con este mismo criterio se definen el resto de las componentes de la matriz de parámetros cruzados.

3.3.3.7. Matriz de parámetros cruzados en el sentido del eje z.

$$\mathrm{D_w} = \begin{bmatrix} Xwu & Ywu & Zwu & Kwu & Mwu & Nwu \\ Xwv & Ywv & Zwv & Kwv & Mwv & Nwv \\ Xwp & Ywp & Zwp & Kwp & Mwp & Nwp \\ Xwq & Ywq & Zwq & Kwq & Mwq & Nwq \\ Xwr & Ywr & Zwr & Kwr & Mwr & Nwr \end{bmatrix}$$

Xwu - Coeficiente que relaciona la fuerza en la dirección del eje X con las velocidades en los ejes Z y X.

Ywu - Coeficiente que relaciona la fuerza en la dirección del eje Y con las velocidades en los ejes Z y X.

Xwv - Coeficiente que relaciona la fuerza en la dirección del eje X con las velocidades en los ejes Z e Y.

Con este mismo criterio se definen el resto de las componentes de la matriz de parámetros cruzados.

3.3.4. Flotación:

El fenómeno de flotación de un cuerpo sumergido en un líquido se produce cuando el empuje debido al fluido es mayor o igual al peso del sólido.

$$\vec{E} \geq -\vec{P}_{flotador}$$

Por el principio de Arquímedes sabemos que:

$$E = -\rho_{agua} * V_{sum}$$

El signo negativo viene del sentido del sistema de referencia, el cual es positivo en el sentido de la gravedad, es decir, hacia abajo.

La primera ecuación es equivalente a decir que la densidad eficaz del flotador tiene que ser menor o igual a la densidad del fluido (la densidad eficaz es el cociente de la masa del submarino entre el volumen del líquido desplazado)

$$\rho_{agua} \leq \rho_{eficaz}$$

$$\rho_{eficaz} = \frac{m_{solido}}{V_{sum}}$$

El centro de carenas de un flotador C, es decir, el punto de aplicación del empuje coincide con el centro de masas del volumen de líquido desplazado. Es importante recordar que, en general, este centro de carena no coincidirá con el centro de masas del vehículo. Para analizar la estabilidad de este es interesante estudiar la posición relativa de estos dos puntos.

Un sistema estable es aquel en el cual pequeñas perturbaciones no son seguidas por un alejamiento indefinido de la situación de equilibrio. Dado que consideramos fuerzas verticales, debemos considerar la estabilidad frente a perturbaciones que modifican la posición vertical y perturbaciones que hacen girar el cuerpo sin variar su posición. En el caso de cuerpos en

equilibrio sumergido, estos están en la frontera entre el equilibrio vertical estable e inestable, las perturbaciones no son atenuadas ni amplificadas.

Ante perturbaciones que producen escora, hay que estudiar la estabilidad rotatoria. Como por lo general el empuje y la gravedad no se aplican en el mismo punto, para que no creen momento de rotación en el sólido deben estar en la misma vertical. Cuando se produce un balanceo el peso y el empuje dejan de ser coaxiales y se produce un par. Si el par anula el balanceo, el equilibrio es estable, si por el contrario lo amplifica, es inestable.

Es sencillo comprobar que habrá estabilidad siempre que el centro de carena C, esté por encima del centro de gravedad del cuerpo G.

Los submarinos suelen variar su flotabilidad usando cámaras donde almacenan agua y aire comprimido, para así poder cambiar su peso y moverse a través del eje z.

Mientras tanto los vehículos submarinos más pequeños, que son en los que nos centramos en este proyecto suelen tener una pequeña flotabilidad positiva, para que en caso de avería el submarino flote hasta la superficie para facilitar su rescate.

$$E > P$$

Para poder sumergirse deben entonces usar la fuerza ejercida por los propulsores [36].

3.4. Modelado de los propulsores

El modelado y el control de los elementos propulsores es una parte clave para la simulación y control de los vehículos submarinos no tripulados. Debido a esto el simulador se beneficiará de las mejoras de exactitud en el modelo de los propulsores, prediciendo de forma más precisa el comportamiento real de este.

De igual manera, las fuerzas y momentos de propulsión vienen afectadas por el modelo del motor, la geometría de la hélice y los efectos hidrodinámicos, además de otros muchos factores, lo cual hace el modelado realmente complejo. Para resolver estos problemas se ha propuesto gran cantidad de modelos de propulsores, unos más complejos que otros. A mayor complejidad, más potencial tendrán para representar fielmente la realidad, pero tendrá un mayor coste. Por un lado, de tiempo de desarrollo, programación y obtención de los coeficientes de la hélice

mediante experimentos, el cual podría haber sido empleado en mejorar otros elementos más relevantes de la simulación. Por otro, también se le añade la utilización de recursos de computación para calcular la respuesta de los propulsores, lo cual puede ralentizar la simulación. Por lo tanto, hay que buscar un compromiso entres sencillez y exactitud a la hora de elegir el modelo a implementar.

3.4.1. Modelo clásico de propulsor

En el análisis clásico la fuerza de propulsión (T) bajo condiciones estacionarias es modelada simplemente como proporcional al cuadrado de la velocidad de la hélice (Ω) con su signo (Newman, 1977).

$$T = c_1 \Omega |\Omega|$$

Sin embargo, también podemos implementar modelos más completos que se asemejen en mayor medida al comportamiento real del propulsor en una mayor gama de situaciones. A continuación, desarrollaremos el modelo propuesto por Jinhyun Kim [37].

3.4.2. Modelo avanzado de propulsor

El propulsor se puede representar como un disco de actuación que crea en el plano de la hélice una discontinuidad de presión de área A_p y una velocidad de flujo U_p. La presión cae a p_a justo antes del disco de la hélice y aumenta hasta p_b después de esta. Finalmente vuelve a la presión ambiental p_∞, lejos de la zona de influencia. Esto implica que se genere una fuerza (T) en sentido contrario al flujo.

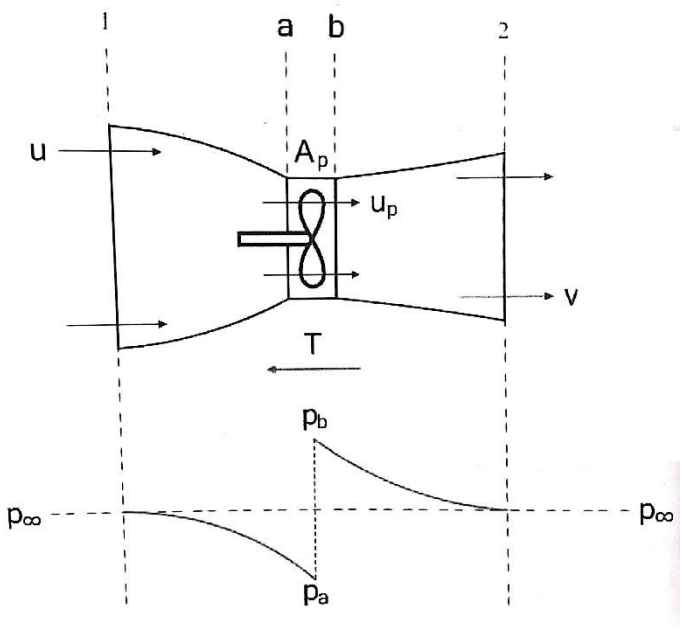


Ilustración 11 Volumen de control en un propulsor

Si usamos la conservación de momento entre las secciones 1 y 2 del volumen de control obtenemos:

$$T = \dot{m}(v - u)$$

De forma similar la relación entre la entrada y la salida de la hélice nos da que:

$$T = A_p(p_b - p_a)$$

Igualando estas dos expresiones

$$T = A_p(p_b - p_a) = \dot{m}(v - u) \quad (*)$$

Si suponemos un flujo ideal las presiones se pueden obtener aplicando la relación de Bernoulli para flujos incompresibles en los diferentes sectores.

De 1 a "a": $p_\infty + \frac{1}{2}\rho u^2 = p_a + \frac{1}{2}\rho u_p^2$

De "b" a 2: $p_\infty + \frac{1}{2}\rho v^2 = p_b + \frac{1}{2}\rho u_p^2$

Sabiendo que $\dot{m} = \rho A_p u_p$ y restando las dos expresiones obtenidas anteriormente, podemos sustituir para obtener:

$$p_b - p_a = \frac{1}{2}\rho(v^2 - u^2)$$

$$u_p = \frac{1}{2}(v + u) \rightarrow v = 2u_p - u$$

Finalmente, la fuerza de propulsión en el disco puede ser escrita en términos de u_p y u de ka siguiente forma:

$$T = 2\rho\, A_p u_p (u_p - u)$$

Por lo tanto, podemos definir la velocidad de flujo axial como

$$u_p \triangleq k_1 u + k_2 D\Omega$$

Donde k_1 y k_2 son constantes.

Para fluido cuasi-estacionario, el flujo axial solo depende del flujo ambiente y la velocidad de giro de la hélice.

Si sustituimos en la ecuaciones anteriores y operamos obtenemos el siguiente modelo, el cual usaremos para hacer el análisis adimensional

$$T = 2\rho A_p (k_1' u^2 + k'_2 u D\Omega + k'_3 D^2 \Omega^2$$

A continuación, tomaremos un enfoque de análisis dimensional.

Es común en la literatura relacionada el uso de la representación adimensional del coeficiente de propulsión, que expresa la relación entre la fuerza de propulsión, la velocidad de giro de la hélice y la velocidad del flujo ambiente:

$$K_T(J_0) = \frac{T}{\rho D\Omega|\Omega|}$$

Donde $J_0 = \frac{u}{D\Omega}$ es el ratio de avance. En algunos estudios, esta relación es representada en una tabla obtenida de manera empírica, o como una regresión lineal simple.

$$K_T = a_1 J_0 + a_2$$

Pero esta simple regresión lineal no puede estimar de forma precisa el coeficiente de propulsión, sobre todo cuando $J_0<0$. En este caso el coeficiente parece aproximarse más a una ecuación cuadrática, excepto en los puntos de discontinuidad. Es más, esta aproximación no tiene ninguna relación física con la fuerza de propulsión, es solo una aproximación linear de la siguiente gráfica:

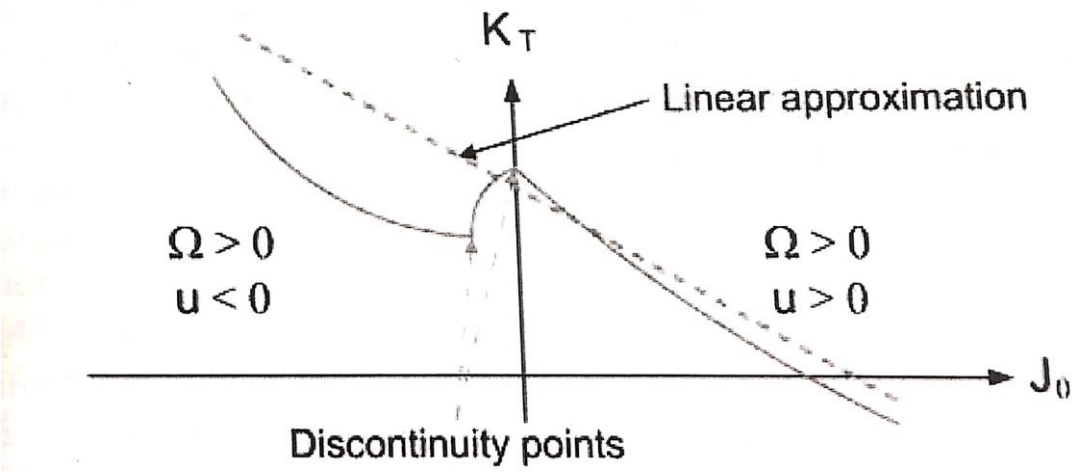


Ilustración 12 Kt frente a J0 en un propulsor

La aproximación de flujo axial estudiada anteriormente da una solución a estos problemas. Se puede expresar de la siguiente forma:

$$K_T(J_0) = \frac{\pi}{2}[k'_1 J_0^2 + k'_2 J_0 + k'_3]$$

Este, al contrario de otros modelos, sí que da una relación apropiada entre la ecuación de la fuerza de propulsión y el gráfico no lineal ya que ha sido obtenido mediante el análisis de las leyes físicas que gobiernan el sistema. Los coeficientes cuadráticos pueden ser obtenidos de manera experimental y cambian dependiendo de las características del propulsor real. Pero este modelo todavía no es capaz de aproximar las discontinuidades en el coeficiente de propulsión, pero consideramos que nos da una precisión lo suficientemente correcta para nuestro simulador, por lo tanto, este es el modelo aplicado.

[37]

3.5. Obtención de coeficientes hidrodinámicos

Conocer una estimación de la resistencia al avance del vehículo a diseñar es muy importante, ya que esta nos ayudará tanto a diseñar el vehículo y sus componentes de forma que podamos alcanzar los requisitos de funcionamiento deseados como a realizar simulaciones realistas del funcionamiento del vehículo para así diseñar métodos de control adecuados y entrenar a los pilotos de los vehículos en un entorno cerrado y seguro.

De manera tradicional estos coeficientes se han obtenido mediante ensayos experimentales, lo que conlleva grandes costes, sin embargo, en los últimos años se ha popularizado el uso de herramientas CFD, las cuales pueden ahorrar en gran medida los recursos necesarios para modelizar un vehículo.

3.5.1. CFD

En proyectos precedentes que abordan este ámbito, realizados en este mismo departamento se ha hecho uso de métodos de CFD para obtener los coeficientes hidrodinámicos de varios vehículos.

La mecánica de fluidos computacional (CFD) es una de las ramas de la mecánica de fluidos que utiliza métodos numéricos y algoritmos para resolver y analizar problemas sobre el flujo de gases y líquidos. Aun con ecuaciones simplificadas y superordenadores de alto rendimiento, solo se pueden alcanzar resultados aproximados en muchos casos. La continua investigación, sin embargo, permite la incorporación de software que reduce la velocidad de cálculo, así como el margen de error al tiempo que permite analizar situaciones cada vez más complejas como los flujos turbulentos o casos con superficie libre. La verificación de los datos obtenidos por CFD suele ser realizada en túneles de viento u otros modelos físicos a escala como tanques de experiencias hidrodinámicas. [2]

El método consiste en discretizar una región del espacio creando lo que se conoce por una malla espacial, dividiendo una región del espacio en pequeños volúmenes de control. Después se resuelve en cada uno de ellos las ecuaciones de conservación discretizadas, de forma que, en realidad, se resuelve una matriz algebraica en cada celda de forma iterativa hasta que el residuo es suficientemente pequeño. [2]

3.5.1.1. Aplicaciones CFD

Este tipo de herramientas tienen aplicación en casi todos los campos de la técnica actual desde la medicina a la ingeniería. Así mismo ocurre en la hidrodinámica, aquí aparece en los siguientes campos:

- Flujos en el interior de tuberías y conductos.
- Maniobrabilidad: El análisis mediante herramientas CFD de los flujos alrededor de apéndices del buque permiten calcular los distintos momentos producidos y así, evaluar la maniobrabilidad del buque.

- Comportamiento en el mar: En este campo los CFD constituyen una parte poco madura. Una de las cosas es estudiar las cargas de oleaje y viento sobre las estructuras que resulta ser muy importante para un buen diseño.
- Resistencia y propulsión: Las aplicaciones CFD se centran fundamentalmente en ese campo. Aquí, sobre todo, hasta finales de los 90 se solían utilizar cálculos potenciales donde se desprecian los efectos de la viscosidad y formación de olas en superficie libre. Esto proporciona cálculos simples, rápidos y más o menos acertados. Sin embargo, con el tiempo y ya a finales de los 90, empezaron a considerarse todos estos efectos siendo estos los puntos que plantean los mayores problemas en la actualidad.
- Diseño de propulsores: En campo de los CFD las técnicas utilizadas son viscosas ya que permiten un mejor cálculo del rendimiento del propulsor para una mejor aproximación a los resultados de experimentos reales. Aquí se emplean técnicas BEM ("Boundary Element Methods") u otras como superficies sustentadoras.

En los proyectos anteriores se han utilizado los CFD tanto para el análisis de la resistencia al avance del modelo como para determinar los coeficientes que determinan el comportamiento del vehículo. [3]

3.5.1.2. Ventajas e inconvenientes de los CFD

Algunas de las ventajas del uso de los CFD son las siguientes:

- Menor uso de recursos tanto temporales como materiales para generar nuevos diseños.
- Posibilidad de analizar sistemas y condiciones complejas de simular experimentalmente.
- Capacidad de estudiar sistemas bajo condiciones peligrosas o más allá de sus condiciones límites de funcionamiento, por ejemplo, accidentes con sustancias tóxicas.
- Nivel de detalle prácticamente ilimitado. Los métodos experimentales son tanto más caros cuanto mayor es el número de puntos de medida, mientras que los programas CFD pueden generar gran cantidad de información sin coste añadido y con posibilidad de hacer estudios paramétricos.

- Generación de gráficos que facilitan la comprensión del resultado y así se puede en muchos casos presentar un producto más atractivo al cliente, al comprender este mejor las cualidades del producto.

Mientras que como desventajas podemos encontrar las siguientes:

- Se precisa de un gran conocimiento de las ecuaciones que modelan ciertos fenómenos físicos, necesitando personal con grandes conocimientos en la materia.
- No siempre es posible llegar a resultados lo suficientemente precisos, dando lugar a grandes errores en cuestiones básicas.
- Simplificación del fenómeno a estudiar para que el hardware y el software puedan abordarlo. El resultado será tanto más preciso cuanto más adecuadas hayan sido las hipótesis y simplificaciones realizadas.
- La existencia de insuficientes e incompletos modelos para simular el efecto de la turbulencia, fenómenos multifásicos o la combustión, entre otros.

En resumen, los CFD son una buena herramienta de trabajo en múltiples campos de la investigación y desarrollo, pero deben ser complementados con otros métodos de análisis y experimentación como los túneles de viento, los canales hidrodinámicos… [2]

3.5.2. Obtención de parámetros del OpenROV

En el proyecto de Ignacio de la Cotera López [1] se obtuvieron mediante métodos CFD los parámetros hidrodinámicos del OpenROV.

Para ello primero se procedió a construir el modelo geométrico en 3D del vehículo, ensamblando cada pieza de manera que se asemejara lo máximo posible a la realidad. Para poder calcular los coeficientes hidrodinámicos, previamente se necesita calcular todas las fuerzas y momentos que aparecen sobre la geometría del ROV cuando se encuentra en movimiento en el seno de un fluido. Para ello, se utilizó la herramienta FluidWorks que es una extensión de simulación dentro del programa SolidWorks.

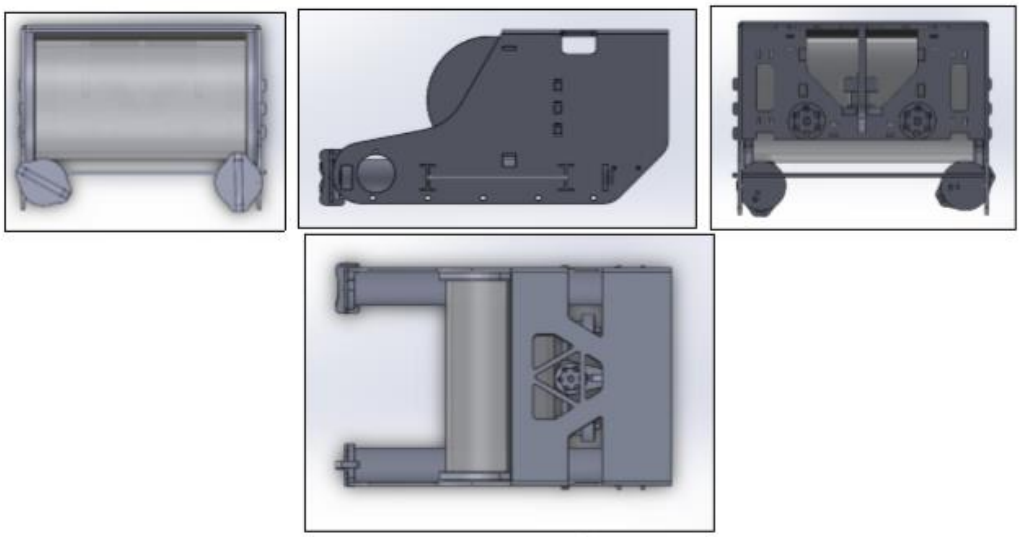


Ilustración 13 Modelo 3D de OpenROV

A continuación, se realizaron las correspondientes simulaciones, asignando un mallado adecuado para que el proceso fuera fiable y se realizara en un tiempo de simulación razonable. A continuación, a modo meramente informativo se muestra una imagen de la distribución de velocidades de un flujo circulado en la dirección de avance.

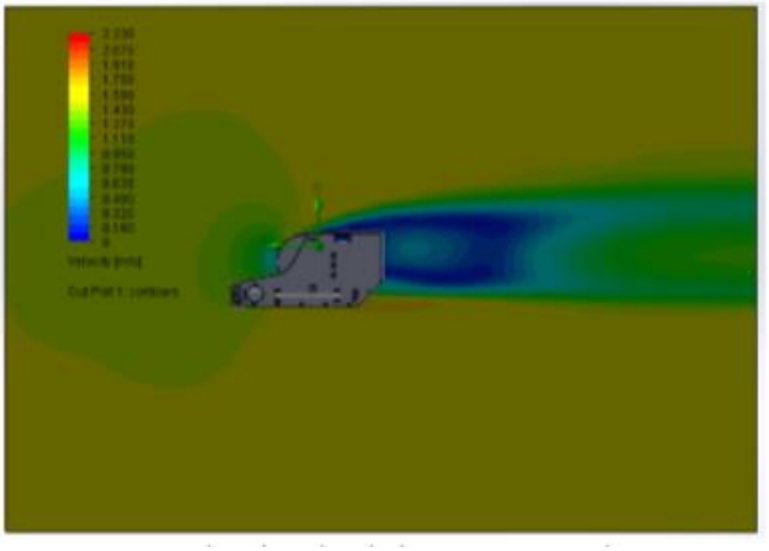


Ilustración 14 Distribución de velocidades de un flujo frente al OpenROV

De los resultados de fuerzas y momentos obtenidos por este método, con la herramienta de MatLab, se proceden a realizaron los cálculos para las regresiones y sacar los valores de los coeficientes de amortiguamiento.

De manera similar se obtuvieron estos parámetros para los vehículos REMUS100 y CUBO en el TFG de José Antonio Ruiz [3].

Estos resultados son en los cuales nos apoyaremos para realizar nuestras simulaciones.

Capítulo 4. Simulador desarrollado

El modelo matemático de la dinámica y cinemática del vehículo, así como los diferentes modelos de cada uno de los diferentes componentes de este (propulsores, timones, controladores…) han sido implementados en un programa en C++. En este capítulo explicaremos el estado actual del simulador y los cambios efectuados en este. Estos cambios estarán orientados en conseguir que el simulador sea lo más general posible, es decir, que permita simular el mayor número de vehículos y configuraciones diferentes cambiando tan solo los parámetros que los caracterizan. También se potenciará su faceta modular de manera que sea sencillo implementar mejoras y añadir extensiones.

4.1. Estructura general del programa de simulación:

La estructura del simulador se ha realizado en base a una programación orientada a objetos. Se hace uso de las diferentes propiedades de estos, como son la herencia, la composición etc.

En el Anexo 4 se refleja de manera detallada todas las clases de las que consta el simulador con sus respectivos métodos y variables.

De forma esquemática la estructura de los objetos es la siguiente:

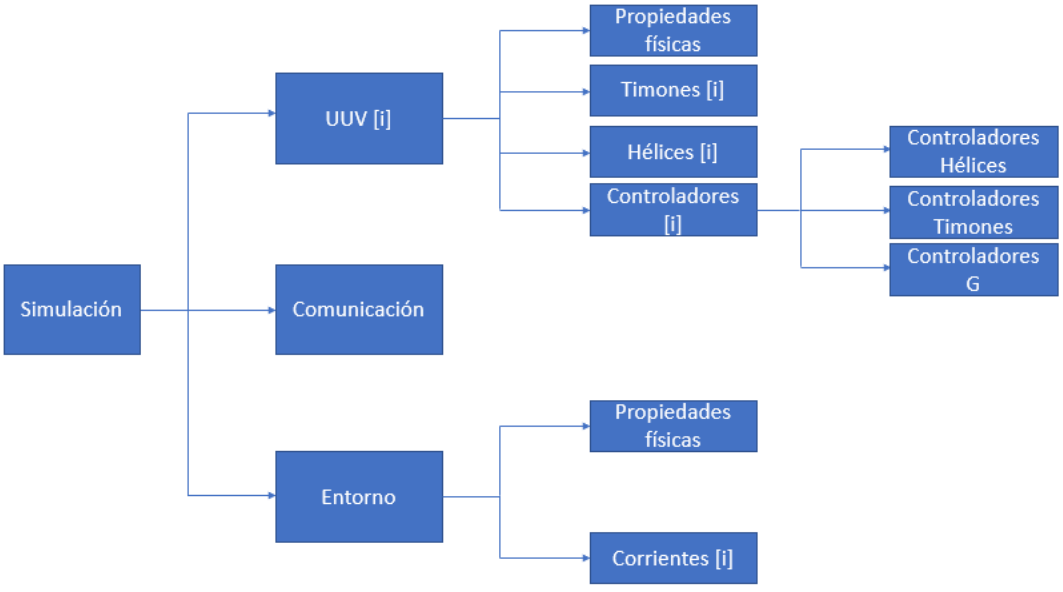


Ilustración 15 Esquema objetos simulador

- **Simulación:** Es el objeto principal se guardan las variables como el tiempo máximo de simulación, el tiempo actual de simulación, el número de vehículos que se van a simular,

el tipo de control (Automático, manual o semiautomático). Usando la composición entre objetos, el objeto simulación también almacena un objeto "Entorno" un objeto "Comunicación" y un array de objetos "UUV". En este objeto se gestiona el tiempo de simulación, se muestran las coordenadas por pantalla y se exportan a un documento de texto. También es dónde se inicializan el resto de las partes de la simulación. Por otra parte, en este objeto donde se define si la simulación se hará únicamente en el dispositivo donde lo ejecutamos o también estaremos conectados con otro ordenador interfaz gráfica y de gobierno manual utilizando la herramienta Unity.

- **UUV:** Hay uno por cada vehículo que se vaya a simular. Almacena todos los datos del vehículo (Coeficientes hidrostáticos, geometría, peso…). También almacena por composición un objeto "Hélice" por cada propulsor que tenga el vehículo, un objeto "Timón" por cada timón y un "Controlador" por cada uno de estos. Al ser inicializado carga todas matrices las propiedades necesarias para configurar el vehículo a simular, así como las condiciones iniciales en las que se encuentra al comienzo de la simulación. Es dentro de este objeto donde se simula el comportamiento dinámico del vehículo. Se resuelven las diferentes ecuaciones del movimiento con las condiciones anteriores del vehículo y la fuerza y momento resultantes al sumarse todas las diferentes fuerzas y momentos aplicadas sobre el vehículo. Las contribuciones de los timones y los propulsores son calculadas en sus objetos correspondientes.

- **Timón:** Se genera uno por cada timón tenga el vehículo. En este objeto se guardan tanto las características del timón (posición en el vehículo, ángulo máximo, velocidad máxima…) como las condiciones instantáneas del timón (ángulo, velocidad, aceleración…). Usando estos valores y la orden de control (ángulo deseado) se calculan las nuevas condiciones del timón usando las ecuaciones del modelo cinemático del timón. El valor instantáneo del ángulo es usado en el objeto "UUV" para calcular la fuerza ejercida por los timones. Una posible mejora sería que el calculo de la fuerza producida por los timones fuera calculada en este, en lugar de en el objeto UUV, siguiendo así una estructura más lógica.

- **Hélice:** Hay un objeto de este tipo por cada propulsor que hay en el vehículo. En este objeto se guardan las características de la hélice, como son la velocidad máxima de giro negativa y positiva y las constantes del modelo de la hélice. A la hora de calcular la fuerza ejercida por la hélice, el objeto recibe en cada iteración un valor de control que va desde -1 a 1, representando -1 una orden de máxima potencia en sentido negativo y 1 en sentido positivo. En base a esto y a las características de la hélice se calcula mediante el modelo la fuerza de que ejerce el propulsor. Esta fuerza será usada en el objeto UUV para calcular la resultante de todas las fuerzas ejercidas sobre el vehículo.

- **Controlador:** En este objeto se calcularán las ordenes de control que recibirán tanto las hélices (Controlador tipo H), los timones (controlador tipo T) y el centro de gravedad del vehículo (Controlador tipo G). Para facilitar la implementación de estos diferentes controles, y además poder programar varios tipos de control distinto con el objetivo de encontrar el mejor posible hacemos uso de la herencia entre objetos. Primero definimos una clase básica llamada “PID” la cual consta de 6 PIDs, en los que cada uno actúa sobre el movimiento en cada grado de libertad del submarino. Cada uno tiene sus propias constantes proporcionales, derivativas e integrales. Así esta clase madre PID guarda estos valores y tiene implementado un método mediante el cual calcula el error relativo en cada eje y mediante este obtenemos 6 valores diferentes de control. Las clases hijas reciben estos valores y los distribuyen a los diferentes actuadores, de una manera u otra dependiendo del tipo de controlador que esté implementado.

- **Entorno:** En este objeto se almacenan las condiciones del entorno en el que se desarrollará la simulación, como son la densidad, el valor de la gravedad y el número de corrientes submarinas que puedan existir.

- **Corriente:** Se creará un objeto de este tipo por cada corriente de agua que queramos en nuestro entorno. Aquí almacenaremos las características como la velocidad máxima del agua dentro de la corriente, la dirección de esta, y el ancho.

- **Comunicación:** Este objeto se utiliza para realizar la comunicación mediante ethernet entre la simulación y la interfaz gráfica y de gobierno, programada en Unity y ejecutada

en una plataforma de computación diferente. Esta comunicación se hace mediante el empleo de sockets.

4.2. Mejoras implementadas

Una vez estudiado la estructura básica del programa simulador podemos centrarnos en los cambios efectuados en el mismo para conseguir los objetivos planteados, entre los cuales están conseguir un simulador lo más general posible, con una introducción de datos sencilla y modular. También se explicarán los métodos de control implementados y las funciones añadidas para conseguir una mayor semejanza con el vehículo OpenROV.

4.2.1. Migración de compilador C++Builder a MinGW

Al inicio de este proyecto el programa se estuvo desarrollando mediante la herramienta de desarrollo de aplicaciones C++ Builder. Este entorno de desarrollo nos brinda una buena cantidad de herramientas y librerías integradas que facilitan la tarea de programación del código. Este entorno hace uso del compilador BCC32C/BCC32X C++, un compilador de alto rendimiento, de uso libre el cual se puede descargar gratuitamente desde su página web.

Sin embargo, aunque el compilador sea gratuito el resto de las herramientas que permiten trabajar de manera óptima con este compilador precisan de la compra de una licencia de C++Builder. Un objetivo importante del proyecto que estamos realizando es la universalización del programa en un sentido amplio. Tanto a la hora de la variedad de vehículos susceptibles de ser simulados, como a la hora de permitir que sea más sencillo ampliar, modificar o corregir el código fuente. Por ello tomamos la decisión de migrar el código hacia un compilador que permitiese su desarrollo en un entorno más fácilmente accesible.

Una vez tomada esta decisión debíamos escoger el compilador el cual usaríamos de ahora en adelante para continuar con el desarrollo del código. Tras barajar varios candidatos (gcc, Visula C++, CLang…) finalmente nos decidimos por MinGW.

MinGW (contracción de Minimalist GNU for Windows) nos permite disponer de gran cantidad de herramientas OpenSource para el desarrollo de aplicaciones en Windows. El motivo principal de nuestra elección es que este compilador es el que usan por defecto los entornos de desarrollo integrados (*IDEs*) más extendidos, los cuales además son gratuitos y de código abierto, como DevC++ y Code::Blocks.

Para conseguir que el código sea compilable y funcional mediante minGW tuvimos que hacer una serie de cambios menores a la parte de la comunicación mediante ethernet e incluir en *linker* una instrucción que nos permite acceder a la librería *wsock32,* necesaria para esta comunicación. En el caso de codeblocks esto se hace sencillamente accediendo a la configuración del compilador:

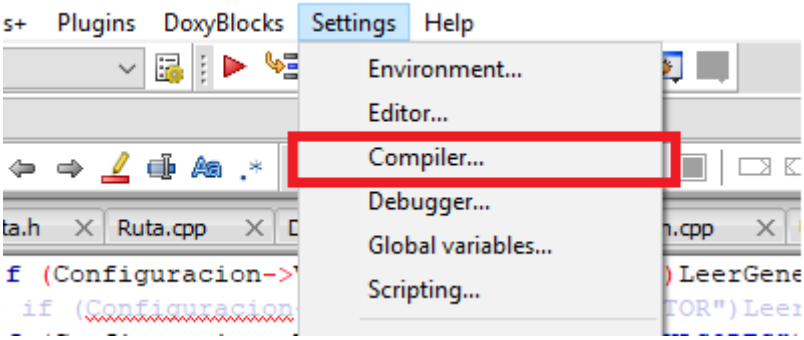


Ilustración 16 Acceso a la configuración del compilador

Y añadiendo "-lws2_32" a la configuración del *linker:*

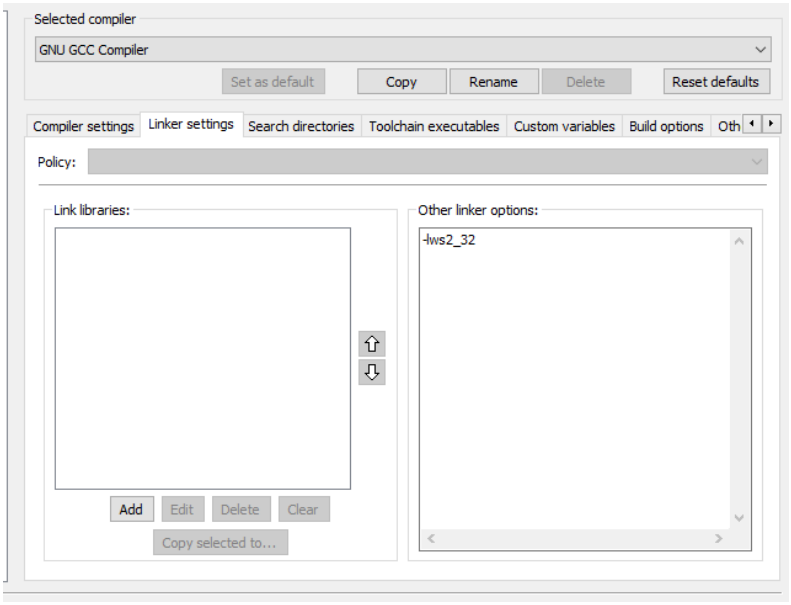


Ilustración 17 Configuración del compilador

4.2.2. Reestructuración de los archivos del programa

El código con el que comenzamos a trabajar estaba contenido en un único archivo en el que estaba la totalidad del programa. Para facilitar su modificación y tener una disposición más

adecuada de todo el código. Esto hará que sea más sencillo identificar su estructura modular y tener una visión más clara y accesible del código, así que función realiza cada módulo.

Para esto se separa cada una de las funciones y clases en dos archivos. Uno en el que están declaradas todas las variables y métodos de los que haremos uso, el cual tendrá una extensión ".h", al que llamaremos *header* (cabecera). Otro donde estarán definidas cada una de esas funciones, el cual tendrá una extensión ".cpp". Por ejemplo, para la clase "UUV" tendremos dos archivos "UUV.h" y "UUV.cpp". Los archivos ".cpp" se compilan automáticamente, mientras que las cabeceras deben ser importadas.

Para que en un archivo se puedan usar funciones o métodos declarados en archivos diferentes debemos importarlos mediante la instrucción #include. Para poder usar todas las funcionalidades de los diferentes archivos, en cada uno de ellos importaremos todas las cabeceras. Pero al hacer esto estaremos compilando repetidas veces el mismo código, lo que nos provocará un error de compilación. Para evitar esto usaremos las instrucciones de control *#ifndef*, *#define* y *#endif*.

En cada archivo colocaremos el siguiente código:

```
#ifndef ETIQUETA_UNICA

#define ETIQUETA_UNICA

(CODIGO)

#endif
```

Esto hará que todo el código contenido entre estas instrucciones se compile una sola vez, independientemente de cuantas veces lo importemos, evitando así problemas de declaraciones múltiples.

4.2.3. Implementación de sistema control mediante PIDs 6 grados de libertad

Como sistema elegido para realizar el control de posición de un vehículo cualquiera hemos optado por la implementación de seis PIDs, cada uno aplicado a un grado de libertad. Cada PID estará asignado a un eje o a un eje de giro, en el cual intentará minimizar el error existente.

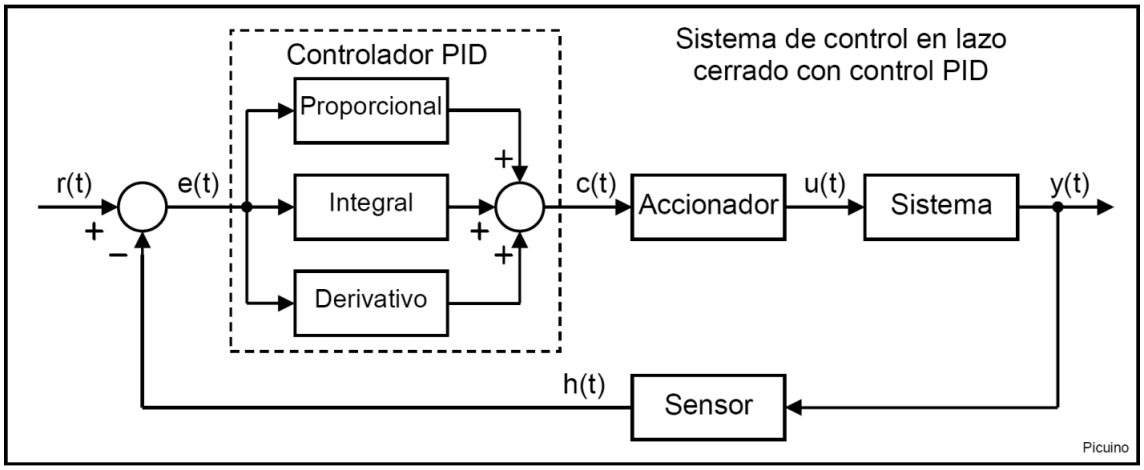


Ilustración 18 Esquema PID básico

Es decir, primero se le asigna un punto espacial y una orientación que se quiere alcanzar. Una vez establecida esta posición y orientación objetivos se calcula las coordenadas relativas de ese punto en el sistema de referencia del vehículo y los giros necesarios para alcanzar la orientación deseada. Estos parámetros serán las entradas de error de cada uno de los seis PID.

Una vez se han procesado las señales de error por los PIDs, obtenemos 6 señales de control, las cuales debemos aplicar a los actuadores para conseguir generar un movimiento en el vehículo el cual nos reduzca estos errores, lo cual no es un problema trivial en general. Más aún si intentamos desarrollar un controlador que funcione para cualquier vehículo independientemente del número y la configuración de actuadores de los cuales disponga.

El caso más sencillo sería el de un vehículo el cual tuviera control sobre los seis grados de libertad de forma independiente, es decir, que la acción sobre cada actuador tuviera solo efecto sobre una única dirección. En la práctica esto difícilmente se puede dar debido tanto a los efectos del amortiguamiento hidrodinámico, como a por otros factores, como el par que ejerce una hélice en sobre su eje de giro, o el hecho de que conseguir un momento puro mediante un actuador no es posible. Aunque el caso ideal no se pueda dar, cuanto más parecido sea nuestro sistema a este supuesto, mejor responderá a este sistema de control.

El caso más aparecido al caso ideal sería el del vehículo CUBO, el cual tiene 6 hélices mediante las cuales podemos actuar sobre cualquier grado de libertad. Cada propulsor afecta de forma predominante únicamente a un sentido de movimiento y a un momento. Por lo tanto, podemos hacer una función específica que distribuya estas señales de control proporcionadas por los PIDs de manera adecuada, definiendo de manualmente a que hélice se le aplicará cada señal y sumando estas señales para obtener la orden final enviada al actuador.

4.2.4. Hélices en cualquier posición

En este trabajo queremos aplicar el control mediante 6 PIDs de forma general evitar y el tener que implementar una función para cada vehículo. Para ello hemos diseñado un método mediante el cual esta asignación de señales se calcula automáticamente teniendo en cuenta la localización de los propulsores en el vehículo y su orientación.

De esta manera siempre dispondremos de un primer sistema de control sin necesidad de cambiar el código del programa. Este método está lejos de ser perfecto, pero hemos comprobado que da unos resultados aceptables en una gran cantidad de casos, si sumamos a esto su sencillez de implementación en nuevos vehículos podemos decir que hemos desarrollado una buena aproximación general al control.

4.2.5. PIDs avanzados

En el sistema de control mencionado anteriormente se hacía uso de 6 PIDs. Cada uno de ellos se programó inicialmente con un modelo de PID simple.

Este tipo de controlador PID funciona correctamente en una serie de condiciones concretas, pero da resultados erráticos cuando estas condiciones no son las ideales y comienzan a aparecer errores de saturación de la parte integral, windup integral, fallos en la parte derivativa… estos errores están bien documentados en la literatura sobre el tema y son resueltos con sencillos cambios en el modelo del controlador.

En este caso nosotros hemos escogido el siguiente modelo de PID avanzado:

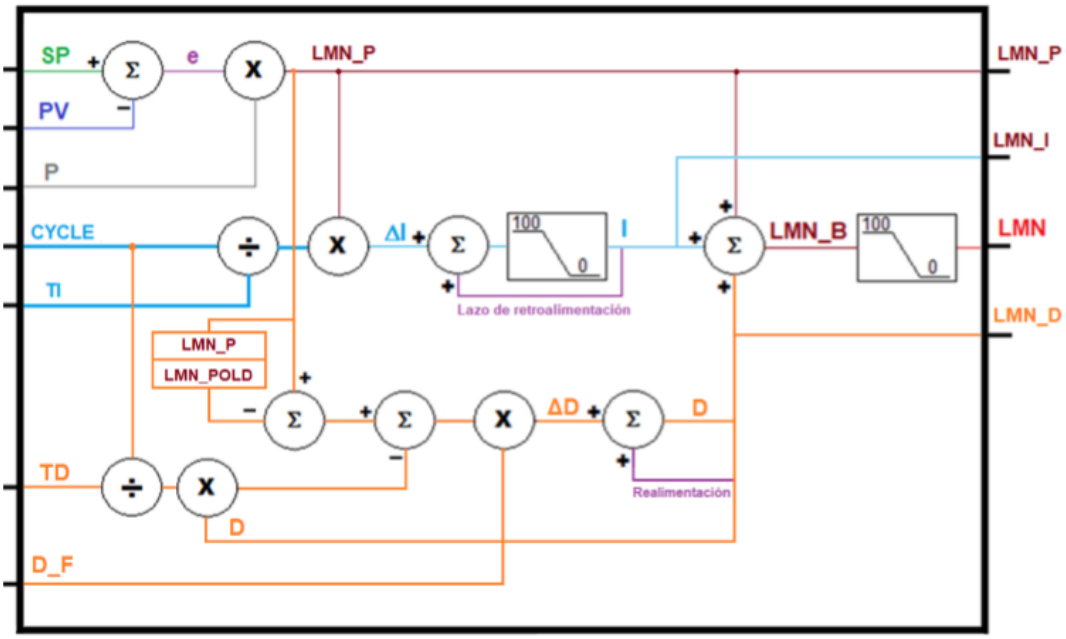


Ilustración 19 Esquema PID avanzado

Donde:

Símbolo	Significado	Símbolo	Significado
LMN	Variable de salida	Cycle	Tiempo de control
SP	Set point	Ti	Tiempo integral
PV	Valor leído por el sensor	TD	Tiempo derivativo
P	Constante proporcional	D_F	Factor Derivativo (la inversa de la constante de tiempo del sistema de amortiguamiento)

Tabla 2 Parámetros del PID avanzado

El cambio más significativo de este nuevo modelo es la introducción de un amortiguamiento para la señal derivativa. Esto nos genera una nueva variable (D_F) a la hora de configurar el PID. El resto de las variables están directamente relacionadas con las introducidas en el modelo básico mediante las siguientes expresiones.

$$P = Kp$$

$$Ki = \frac{Kp}{Ti}$$

$$Kd = Td * Kp$$

Este nuevo modelo más completo y avanzado de PID debería darnos mejores resultados, pero tiene la contrapartida de que es algo más complejo desde el punto de vista computacional y que

se añade un nuevo parámetro a configurar, D_F, por lo cual, para no aumentar la complejidad del sistema si no se requiere hemos mantenido los dos tipos de PID programados, de tal manera que se pueda seleccionar entre uno y otro.

4.2.6. Introducción de datos

Para hacer nuestro simulador lo más versátil y generalista posible, los parámetros de cada simulación deben poderse cambiar fácilmente. El número de diferentes configuraciones las cuales podemos querer simular es potencialmente infinito, por lo tanto, cuanto mayor sea el número de parámetros fácilmente variables, de mayor utilidad nos será el programa. Algunos ejemplos de parámetros a cambiar pueden ser:

El número de vehículos que queremos simular, sus características físicas, sus coeficientes hidrodinámicos, el número, posición y tipo de propulsores, el número, posición y tipo de timones, los parámetros de los controladores PID, las características de la IMU, las características del entorno de simulación, la posición inicial del vehículo, la trayectoria deseada…

Para que esta configuración no se deba hacer dentro de la programación, lo cual sería sumamente tedioso hemos decidido que todos estos valores se cambiarán en una hoja de Excel, en la que la introducción de estos sea lo más amable posible para el usuario.

Para ello, debido a la gran cantidad de parámetros a cambiar, se estructurará en diferentes hojas de cálculo.

La hoja principal será la hoja de configuración, donde en la primera fila estarán los nombres de las diferentes simulaciones, y en las filas sucesivas estarán los nombres de las configuraciones específicas de cada grupo de variables. También se indicará el número de vehículos a simular y sus diferentes características. Así si simulamos la configuración a la que hemos denominado “Petrolero” simularemos un solo vehículo, en el entorno “Entorno1”, con el tiempo de simulación “Simulacion1”, el único vehículo a simular será el “REMUS100” …

Para indicar al programa el nombre de la configuración que queremos simular, debemos escribirlo en el archivo “main” del programa, en la siguiente instrucción:

```
Simulacion sim("PETROLERO ");
```

COMPONENTE:	PETROLERO
NUMERO VEHICULOS	1
ENTORNO	Entorno1

TIEMPO SIMULACION	Simulacion1
IDENTIFICADOR	1
VEHICULO	REMUS100
TIMONES	REMUS100
COMPARADOR	CMP1
CONTROLADORES_T	P1
CONTROLADORES_H	PIDH1
IMU	IMU2
MOTOR	MOTOR1
PROPULSORES	REMUS100
TRAYECTORIA	Petrolero
CONDICIONES_INICIALES	Petrolero
CONTROLADORES_G	PIDG1
OBJETIVO	Petrolero

Tabla 3 Ejemplo de tabla de configuración

El programa lee cada uno de estos valores, y posteriormente se va a las correspondientes hojas de Excel para cargar en las variables del programa los valores ahí definidos.

Cada grupo de parámetros, como pueden ser coeficientes hidrodinámicos, configuración de las hélices, propiedades del entorno… estará en una hoja de Excel, y, por columnas se guardarán las diferentes disposiciones de valores. Por ejemplo, en la columna de Entorno1, estarán todos los parámetros que queremos cargar cuando usemos una simulación en ese entorno.

	VARIABLES	ENTORNO1	ENTORNO2
	G	9.81	9.81
	Rho	1.03E+03	1.03E+03
	Numero	1.00E+00	1.00E+00
CORRIENTE	Identificador	1	1
	Vmax	1	0
	Rumbo	90	0
	Cabeceo	0	0
	AnchoM	10	0
	AnchoT	40	0
	X0	0	0

	Y0	0	0
	Z0	0	0

Tabla 4 Ejemplo de tabla de entorno

Así pues, irá recorriendo todos los parámetros principales y accediendo a la hoja de cálculo correspondiente donde comprobará que el nombre es correcto

Debido a que al formato xls, es muy difícil acceder desde aplicaciones externas, hemos optado por, una vez que se han configurado los valores en Excel, se deban exportar a formato .csv (separado por comas), el cual nos genera un archivo mucho más fácil de leer por el programa. Esto se puede hacer muy fácilmente mediante una macro de Excel que nos exporte una detrás de otra todas las hojas que tiene el archivo.

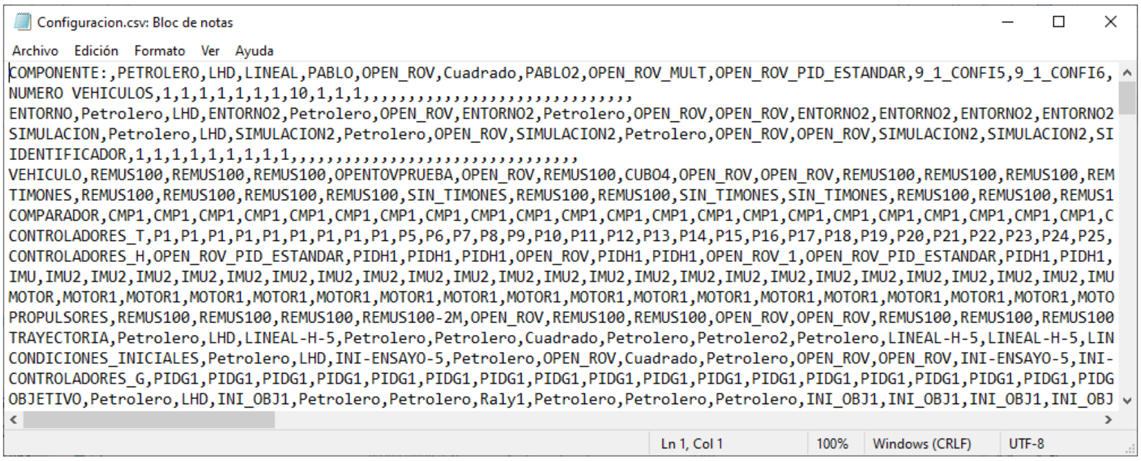
La programación de esta macro se puede hacer de diferentes formas, en el anexo 1 se adjunta la que hemos usado para este proyecto, la cual hemos intentado optimizarla para que se ejecute de la manera más rápida y automática posible.

Esta macro de Excel irá abriendo y exportando todas las hojas del documento y guardándolas en formato .csv separado por comas, poniéndole el mismo nombre a al archivo que el nombre de la hoja. Así por ejemplo el archivo configuración se guardará como "*Configuración.csv*". La hoja del archivo .xlms principal, abierta en una hoja de cálculo sería el siguiente:

	A	B	C	D	E	F	G	H
1	**COMPONENTE:**	**PETROLERO**	**LHD**	**LINEAL**	**PABLO**	**OPEN_ROV**	**Cuadrado**	**PABLO2**
2	**NUMERO VEHICULOS**	1	1	1	1	1	1	1
3	**ENTORNO**	Petrolero	LHD	ENTORNO2	Petrolero	OPEN_ROV	ENTORNO2	Petrolero
4	**SIMULACION**	Petrolero	LHD	SIMULACION2	Petrolero	OPEN_ROV	SIMULACION2	Petrolero
5	**IDENTIFICADOR**	1	1	1	1	1	1	1
6	**VEHICULO**	REMUS100	REMUS100	REMUS100	OPENTOVPRUEBA	OPEN_ROV	REMUS100	CUBO4
7	**TIMONES**	REMUS100	REMUS100	REMUS100	REMUS100	SIN_TIMONES	REMUS100	REMUS100
8	**COMPARADOR**	CMP1	CMP1	CMP1	CMP1	CMP1	CMP1	CMP1
9	**CONTROLADORES_T**	P1	P1	P1	P1	P1	P1	P1
10	**CONTROLADORES_H**	OPEN_ROV_PID_ESTANDA	PIDH1	PIDH1	PIDH1	OPEN_ROV	PIDH1	PIDH1
11	**IMU**	IMU2	IMU2	IMU2	IMU2	IMU2	IMU2	IMU2
12	**MOTOR**	MOTOR1	MOTOR1	MOTOR1	MOTOR1	MOTOR1	MOTOR1	MOTOR1
13	**PROPULSORES**	REMUS100	REMUS100	REMUS100	REMUS100-2M	OPEN_ROV	REMUS100	REMUS100
14	**TRAYECTORIA**	Petrolero	LHD	LINEAL-H-5	Petrolero	Petrolero	Cuadrado	Petrolero
15	**CONDICIONES_INICIALES**	Petrolero	LHD	INI-ENSAYO-5	Petrolero	OPEN_ROV	Cuadrado	Petrolero
16	**CONTROLADORES_G**	PIDG1	PIDG1	PIDG1	PIDG1	PIDG1	PIDG1	PIDG1
17	**OBJETIVO**	Petrolero	LHD	INI_OBJ1	Petrolero	Petrolero	Raly1	Petrolero

Ilustración 20 Ejemplo tabla de configuración en Excel

Mientras que el archivo .csv creado al ejecutar la macro de exportación sería el siguiente.



Configuracion.csv: Bloc de notas

Archivo Edición Formato Ver Ayuda

```
COMPONENTE:,PETROLERO,LHD,LINEAL,PABLO,OPEN_ROV,Cuadrado,PABLO2,OPEN_ROV_MULT,OPEN_ROV_PID_ESTANDAR,9_1_CONFI5,9_1_CONFI6,
NUMERO VEHICULOS,1,1,1,1,1,1,1,10,1,1,1,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
ENTORNO,Petrolero,LHD,ENTORNO2,Petrolero,OPEN_ROV,ENTORNO2,Petrolero,OPEN_ROV,OPEN_ROV,ENTORNO2,ENTORNO2,ENTORNO2,ENTORNO2
SIMULACION,Petrolero,LHD,SIMULACION2,Petrolero,OPEN_ROV,SIMULACION2,Petrolero,OPEN_ROV,OPEN_ROV,SIMULACION2,SIMULACION2,SI
IDENTIFICADOR,1,1,1,1,1,1,1,1,1,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
VEHICULO,REMUS100,REMUS100,REMUS100,OPENTOVPRUEBA,OPEN_ROV,REMUS100,CUBO4,OPEN_ROV,OPEN_ROV,REMUS100,REMUS100,REMUS100,REM
TIMONES,REMUS100,REMUS100,REMUS100,REMUS100,SIN_TIMONES,REMUS100,REMUS100,SIN_TIMONES,SIN_TIMONES,REMUS100,REMUS100,REMUS1
COMPARADOR,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,CMP1,C
CONTROLADORES_T,P1,P1,P1,P1,P1,P1,P1,P1,P1,P5,P6,P7,P8,P9,P10,P11,P12,P13,P14,P15,P16,P17,P18,P19,P20,P21,P22,P23,P24,P25,
CONTROLADORES_H,OPEN_ROV_PID_ESTANDAR,PIDH1,PIDH1,PIDH1,OPEN_ROV,PIDH1,PIDH1,OPEN_ROV_1,OPEN_ROV_PID_ESTANDAR,PIDH1,PIDH1,
IMU,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU2,IMU
MOTOR,MOTOR1,MOTOR1,MOTOR1,MOTOR1,MOTOR1,MOTOR1,MOTOR1,MOTOR1,MOTOR1,MOTOR1,MOTOR1,MOTOR1,MOTOR1,MOTOR1,MOTOR1,MOTOR1,MOTO
PROPULSORES,REMUS100,REMUS100,REMUS100,REMUS100-2M,OPEN_ROV,REMUS100,REMUS100,OPEN_ROV,OPEN_ROV,REMUS100,REMUS100,REMUS100
TRAYECTORIA,Petrolero,LHD,LINEAL-H-5,Petrolero,Petrolero,Cuadrado,Petrolero,Petrolero2,Petrolero,LINEAL-H-5,LINEAL-H-5,LIN
CONDICIONES_INICIALES,Petrolero,LHD,INI-ENSAYO-5,Petrolero,OPEN_ROV,Cuadrado,Petrolero,OPEN_ROV,OPEN_ROV,INI-ENSAYO-5,INI-
CONTROLADORES_G,PIDG1,PIDG1,PIDG1,PIDG1,PIDG1,PIDG1,PIDG1,PIDG1,PIDG1,PIDG1,PIDG1,PIDG1,PIDG1,PIDG1,PIDG1,PIDG1,PIDG1,PIDG
OBJETIVO,Petrolero,LHD,INI_OBJ1,Petrolero,Petrolero,Raly1,Petrolero,Petrolero,Petrolero,INI_OBJ1,INI_OBJ1,INI_OBJ1,INI_OBJ
```

Ln 1, Col 1 | 100% | Windows (CRLF) | UTF-8

Ilustración 21 Ejemplo archivo de configuración .csv

Para leer estos archivos generados en .csv se han creado dos objetos. Uno llamado *Datos* y otro *LeerDeExcel.* En el objeto datos primero se abre cada archivo, se localiza la columna de valores que nos interesa y se crea una matriz dinámica de dos columnas donde en la primera se almacena el nombre de la variable y en la segunda el valor. Posteriormente, el objeto *LeerDeExcel* se encarga de almacenar estos valores de las matrices generadas anteriormente en las variables correspondientes del programa, para que puedan ser usadas por el simulador.

Todos los parámetros se introducirán en las unidades básicas del sistema internacional (m, kg, s…) excepto cuando se indique lo contrario. A continuación, se presenta cada una de las tablas en las que vienen reflejados los diferentes parámetros y propiedades de los objetos del programa de simulación:

4.2.6.1. Tabla de propiedades físicas

I	Vector de momento de inercia	Ix
		Iy
		Iz
Ip	Vector de momento de inercia polar	Ipx
		Ipy
		Ipz
L	Longitud	
cdg	Posición centro de gravedad	x
		y
		z
cc	Posición centro de carena	x
		y
		z
m	Masa	
v	Volumen	
proa	Distancia desde origen hasta la proa	
popa	Distancia desde origen hasta la popa	
PRPSauv	Coordenadas del punto que queremos que siga la trayectoria	x
		y
		z

Tabla 5 Propiedades físicas

En la siguiente tabla introduciremos los valores de la matriz de masa añadida:

MASA AÑADIDA	**Xudot**
	Xvdot
	Xwdot
	Xpdot
	Xqdot
	Xrdot

	Yudot
	Yvdot
	…

Tabla 6 Masa añadida

En esta tabla se introducen los coeficientes de la matriz de amortiguamiento lineal:

Coeficientes hidrodinámicos lineales	Dl_AVANCE	Xu
		Yu
		Zu
		Ku
		Mu
		Nu
	Dl_RETROCESO	Xu
		Yu
		Zu
		Ku
		Mu
		Nu
	Dl_BABOR	Xv
		Yv
		Zv
		Kv
		Mv
		Nv
	…	…

Tabla 7 Coeficientes hidrodinámicos lineales

En esta tabla se introducen los coeficientes de la matriz de amortiguamiento no lineal:

Coeficientes hidrodinámicos no lineales	Dnl_AVANCE	Xuu
		Yuu
		Zuu
		Kuu
		Muu
		Nuu
	Dnl_RETROCESO	Xuu
		Yuu
		Zuu
		Kuu
		Muu
		Nuu
	Dnl_BABOR	Xvv
		Yvv
		Zvv
		Kvv

		Mvv
		Nvv
	…	…

Tabla 8 Coeficientes hidrodinámicos no lineales

En esta tabla se introducen los coeficientes de las matrices de amortiguamiento cruzadas:

Coeficientes hidrodinámicos cruzados	AVANCE_BABOR	Xuv
		Yuv
		Zuv
		Kuv
		Muv
		Nuv
	AVANCE_ESTRIBOR	Xuv
		Yuv
		Zuv
		Kuv
		Muv
		Nuv
	AVANCE_DESCENSO	Xuw
		Yuw
		Zuw
		Kuw
		Muw
		Nuw
	…	…

Tabla 9 Coeficientes hidrodinámicos cruzados

4.2.6.2. Tabla de entorno

	g	Aceleración de la gravedad
	rho	Densidad del fluido
	Numero	Numero de corrientes
CORRIENTE	Identificador	Identificador de cada corriente (1,2,3…)
	Vmax	Velocidad máxima
	Rumbo	Rumbo de la corriente
	Cabeceo	Cabeceo de la corriente
	AnchoM	Ancho vertical de la corriente
	AnchoT	Ancho transversal de la corriente
	X0	Posición del centro de la corriente
	Y0	
	Z0	

Tabla 10 Propiedades del entorno

4.2.6.3. Tabla de propulsores

Dato general:		**Numero**	Número de propulsores del vehículo	
CARACTICAS HELICE		**Tipo**	Nombre del tipo de hélice	
		SG	Signo de giro	
POSICON DE LA HÉLICE	**Posición de la hélice**	**rh**	Coordenadas de la posición de la hélice respecto al sistema del vehículo	x
				y
				z
	Dirección de avance de la hélice **Vector unitario**	**uh**	Vector de dirección hacia donde están orientadas la hélice	x
				y
				z
CARACTICAS HELICE		**Tipo**	Nombre del segundo tipo de hélice	
		SG	Signo de giro de la segunda hélice	
…		…	Continua con cada uno de los propulsores que hayamos definido	

Tabla 11 Propiedades de los propulsores

En tipo debemos poner un nombre el cual hayamos definido en la tabla de hélices, la cual es la siguiente:

PARAMETROS DE LA HÉLICE	**Diametro de la hélice**	**Dh**	Diámetro de la hélice
	Masa de la hélice	**mh**	Masa de la hélice
	Momento de Inercia	**Jh**	Momento de inercia de la hélice
	Número de RPM minimo	**Nmin**	Número mínimo de RPM
	Número de RPM maximo	**Nmax**	Número máximo de RPM
	Coeficientes de laa hélice	**k**	K1
			K2
			K3
	constantes positivas	**a**	a1
			a2
			a3
	constante experimental	**Ka**	Constante experimental
	relacion PAR/FUERZA	**nu**	Relación entre la fuerza de empuje que genera en el sentido de avance y el par en el sentido de giro

Tabla 12 Propiedades de las hélices

4.2.6.4. Tabla de timones

		Numero	Numero timones
PARAMETROS DE LOS TIMONES	**RUMBO**	**nombre**	Tipo rumbo R o tipo cabeceo C
		an_min	Angulo mínimo
		an_max	Angulo máximo
		tr	Tiempo de retardo de respuesta
		Wmax	Velocidad angular máxima
		amax	Aceleración angular máxima
COEF. TIMONES	**Xuud** **Yuud** **Zuud** **Kuud** **Muud** **Nuud**	**Cuud**	Parámetros hidrodinámicos de los timones
…	**…**	**…**	Continua para el número de timones introducido

Tabla 13 Propiedades de los timones

4.2.6.5. Controlador 6 grados de libertad

Kp	x	parámetros proporcionales para los 6 grados de libertad
	y	
	z	
	k	
	m	
	n	
Kd	x	Parámetros derivativos para los 6 grados de libertad
	y	
	z	
	k	
	m	
	n	
Ki	x	Parámetros integrales para los 6 grados de libertad
	y	
	z	
	k	
	m	
	n	

Tabla 14 parámetros de los PIDs de los propulsores

4.2.6.6. Tabla tiempos de simulación

Tiempo_Total	Tiempo de duración de la simulación
dt	Incremento de tiempo en cada iteración de la simulación (si vale 0 se realiza en tiempo real)
dt_control	Tiempo de control de los PID
dt_mostrar	Periodo en el cual se muestran los datos de la simulación

Tabla 15 Tiempos de simulación

4.2.6.7. Tabla condiciones iniciales

u **v** **w** **p** **q** **r**	**X**	Velocidades iniciales
x **y** **z** **phi** **theta** **psi**		Posición inicial

Tabla 16 Condiciones iniciales

4.2.7. Coeficientes hidrodinámicos variable según el sentido de movimiento

Para que el simulador sea lo más genérico que se pueda, se deben hacer las mínimas suposiciones y simplificaciones posibles a la hora de poder definir parámetros. Una simplificación se podría hacer es que para los dos sentidos de movimiento en cada eje de movimiento y de rotación los coeficientes hidrodinámicos son iguales. Es decir, que los coeficientes hidrodinámicos son iguales cuando el vehículo avanza y cuando retrocede, cuando asciende y desciende etc.

Esta simplificación, aunque útil en vehículos simétricos (como el CUBO) o en los cuales un sentido de movimiento predomina sobre los demás (como el REMUS100) si queremos abarcar el mayor número de casos posibles no es apropiada.

Por ello de manera general, por cada sentido de movimiento y giro se deben definir 6 parámetros lineales y no lineales diferentes. De igual modo se deben definir diferentes parámetros cruzados para cada sentido.

Esto hace que el número de coeficientes a introducir aumente notablemente. Concretamente tenemos 72 diferentes coeficientes lineales, 72 no lineales y 288 cruzados. En caso de que hagamos las suposiciones que hemos expuesto anteriormente, tan solo debemos introducir los mismos valores para los dos sentidos de cada movimiento.

4.2.8. Recreación de sistemas de control de OpenROV

En este proyecto nos centraremos especialmente en realizar la simulación del vehículo OpenROV de la forma más realista que nos sea posible. Para ello se deben implementar en el sistema de control del vehículo las modificaciones que se señalan en los siguientes apartados

4.2.8.1. Implementación del algoritmo de gestión de acción de los propulsores de OpenROV

El sistema mediante el cual se controlan las ordenes de acción de cada motor en el OpenROV (el cual describiremos en el punto 6.3.1 es correcta en este vehículo, ya que tenemos una configuración concreta de la distribución de los motores, sin embargo, en nuestro simulador tenemos implementado un sistema de control más genérico, el cual está diseñado para funcionar con cualquier disposición arbitraria de hélices.

Sin embargo, para tener una representación lo más fiel posible del vehículo real, hemos modificado el código del simulador para añadir la posibilidad de emular este método de control, poniendo así a la realidad y la simulación en las mismas condiciones, para que esta comparación pueda darnos información valida sobre la idoneidad del modelo usado.

Para ello hemos creado una nueva clase de PID, llamada PID_OpenRov la cual hereda variables y métodos de la clase básica PID. Esta nueva clase nos dará una señal de control por cada eje, pero a la hora de aplicar estas señales a los motores tan solo usamos las señales de “guiñada” y “ascenso”, es decir, tanto el movimiento sobre el eje z como la rotación sobre este mismo eje. Más adelante estas señales son transformadas en órdenes a los motores de una manera análoga a la que realiza en el sistema de control del Cockpit de OpenRov.

4.2.8.2. Implementación del sistema de cálculo de error de OpenROV

Otra diferencia significativa es a la hora de calcular la orden de control en el eje z. Tal y como está programado en el simulador generalista, este se calcula a partir del error en el eje z en el sistema de referencia solidario al vehículo, es decir, que este error no es necesariamente igual al error de profundidad, ya que si el vehículo esta rotando en lo ejes “x” o “y”, este error será la proyección del vector posición del punto objetivo sobre el eje “z” del vehículo, el cual no tiene por qué coincidir con el eje z del sistema absoluto. Esto es un inconveniente a la hora de emular el vehículo OpenRov ya que el sensor que usa para este propósito el vehículo real es un manómetro que tan solo es capaz de detectar la profundidad a la cual se encuentra el

submarino, y este es el dato que usa para calcular el error en el eje z, independientemente de la orientación que tenga el vehículo en cada momento. Este sistema es adecuado para el vehículo OpenRov ya que, como ya hemos explicado anteriormente, por sus características físicas (relación entre centro de carena y centro de gravedad), tiene tendencia a mantenerse estable y no rotar sobre los ejes “x” e “y”. Esto nos lleva a que si se quiere que los dos sistemas (real y simulado) sean lo más parecidos posible debamos cambiar el modo en el que se calcula el error en el eje z de nuestro simulador. Para ello se debe sustituir como entrada de error para este eje la posición relativa por la posición absoluta.

Así cualquier error entre la profundidad de referencia y la posición real será corregido únicamente con la acción del motor de ascenso y descenso.

4.3. Interfaz gráfica Unity

Se ha usado el entorno Unity para crear una visualización de las dinámicas del vehículo en simulación.

Unity es un motor gráfico 3D para PC y Mac que viene empaquetado como una herramienta para crear juegos, aplicaciones interactivas, visualizaciones y animaciones en 3D y tiempo real.

En este entorno se ha desarrollado un programa, el cual es capaz de representar visualmente en tiempo real los resultados de la simulación, así como capturar los eventos de un mando de control y transmitírselos al simulador de C++.

4.3.1. Comunicación entre Unity y C++

El simulador ha sido programado de tal forma que es capaz de transmitir y recibir información de la simulación desde el ordenador en el que se está ejecutando a otro. Esto nos sirve para poder ejecutar en un dispositivo el programa de simulación y en otro tener en funcionamiento el entorno visual de Unity.

Para ello haremos uso de los sockets. Para que dos programas puedan comunicarse entre sí es necesario que se cumplan ciertos requisitos:

- Que un programa sea capaz de localizar al otro.
- Que ambos programas sean capaces de intercambiarse cualquier secuencia de bytes, es decir, datos relevantes a su finalidad.

Para ello son necesarios los dos recursos que originan el concepto de socket:

- Un par de direcciones el protocolo de red (dirección IP, si se utiliza el protocolo TCP/IP), que identifican la computadora de origen y la remota.
- Un par de números de puerto, que identifican a un programa dentro de cada ordenador.

Los sockets permiten implementar una arquitectura cliente-servidor. La comunicación debe ser iniciada por uno de los programas que se denomina programa "cliente". El segundo programa espera a que otro inicie la comunicación, por este motivo se denomina programa "servidor". Nuestro servidor será el programa desarrollado en C++ mientras que el cliente será el programa de Unity, ejecutado en otro ordenador.

Un socket es un proceso o hilo existente en la máquina cliente y en la máquina servidora, que sirve en última instancia para que el programa servidor y el cliente lean y escriban la información. Esta información será la transmitida por las diferentes capas de red.[38]

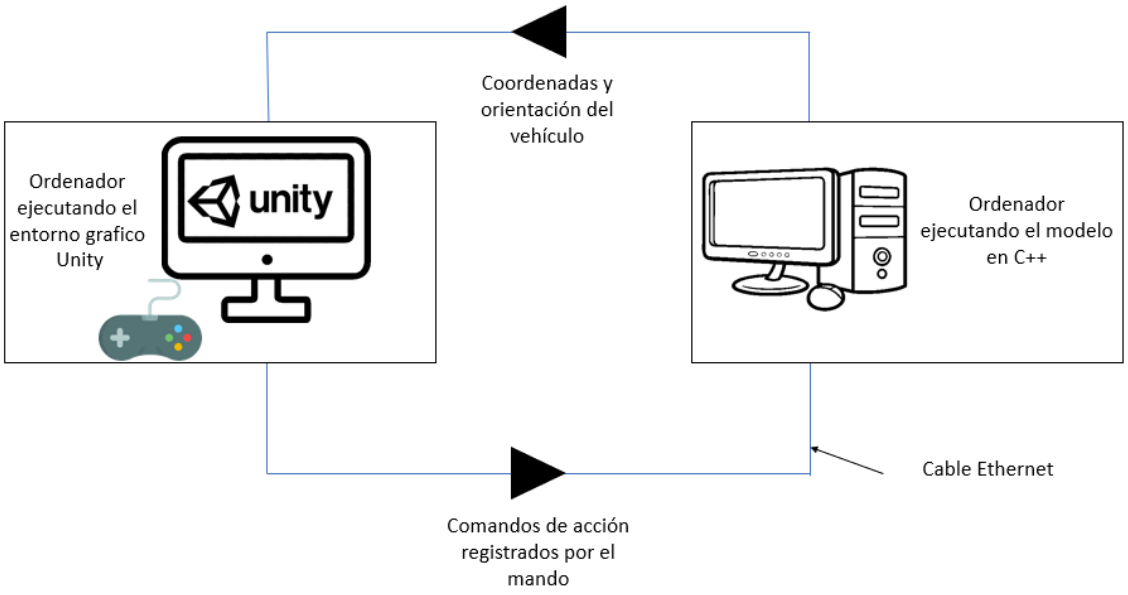


Ilustración 22 Esquema de comunicación entre simulación y entorno Unity

Nosotros usaremos este sistema de comunicación para traspasar los datos clave para el funcionamiento de la simulación. Desde el ordenador que ejecuta el programa en C++ enviamos las coordenadas del vehículo en cada instante. Esto 6 valores (las coordenadas X, Y, Z, balanceo, cabeceo y guiñada) servirán para que el sistema de simulación gráfica desarrollado en Unity sepa las coordenadas y la orientación exacta con la que debe representar al vehículo en todo instante. Por su parte el ordenador donde se ejecuta el entorno Unity tendrá conectado el mando con el que se controlará el vehículo. Este ordenador enviará al otro los datos que lea del mando, para que el simulador tenga estos datos y pueda calcular las dinámicas resultantes. El mando envía

un valor diferente por cada botón y dos por cada Joystick, lo que nos da un total de 8 valores transmitidos.

4.3.2. Utilización del simulador

Tras conectar con un cable ethernet los dos ordenadores donde se ejecutarán los dos programas iniciaremos ambos.

Primero se inicia el programa con el modelo matemático (servidor), que queda esperando a recibir una orden del programa interfaz (cliente) para comenzar la ejecución de los algoritmos del cálculo.

Cuando iniciemos la aplicación de Unity se nos mostrará el siguiente menú:

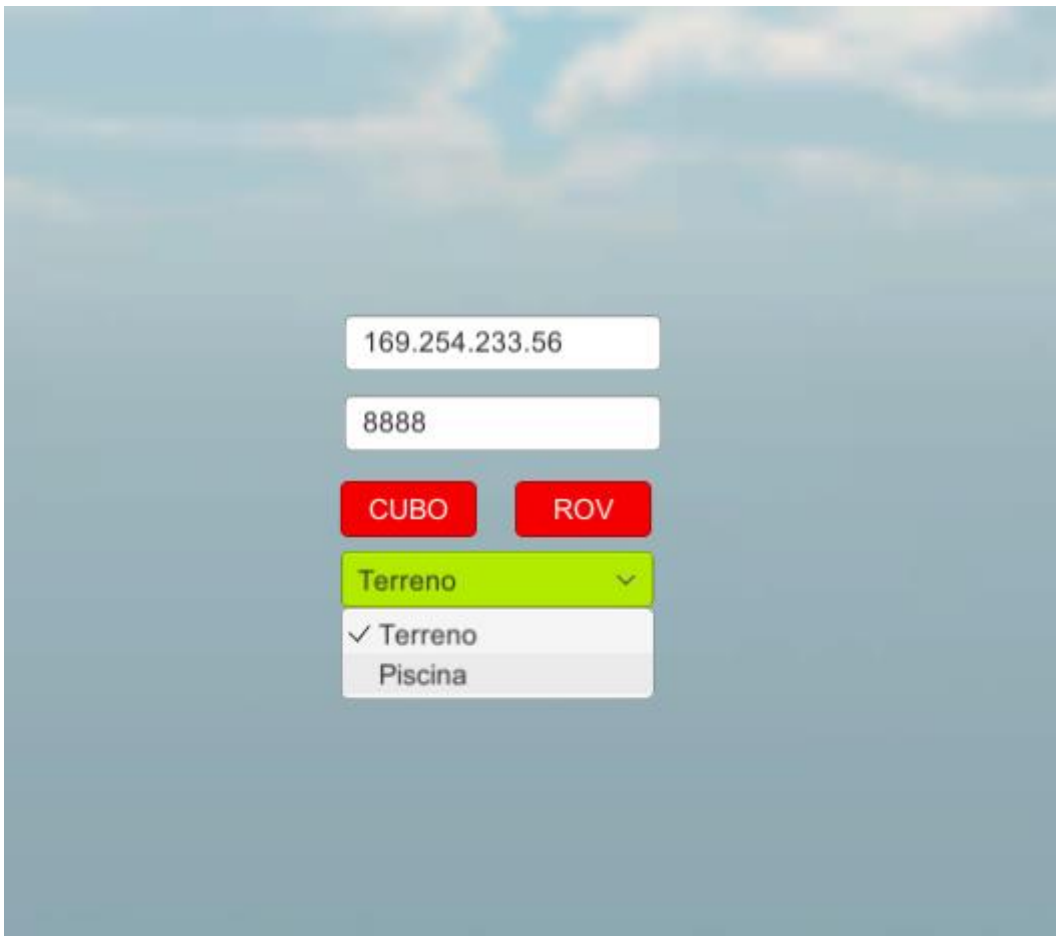


Ilustración 23 Menú de la simulación gráfica

En el cual debemos rellenar el primer campo con el numero IP del ordenador donde se ejecuta la simulación en C++ y el puerto de red, el cual viene por defecto el 8888. En la barra desplegable podremos seleccionar si queremos que se muestre el entorno “terreno”, el cual simula un fondo submarino, o el entorno “piscina”, que es una simulación gráfica de una piscina olímpica.

Una vez seleccionado el terreno pulsaremos el botón rojo con el nombre del vehículo que queremos simular. Podemos elegir entre el vehículo “CUBO”, o el OpenROV.

Cabe destacar que actualmente, esta selección de vehículo y terreno solo modificará la representación gráfica del vehículo en la simulación. Los parámetros físicos, coeficientes hidrodinámicos, numero de propulsores… deben ser definidos en la hoja de cálculo de configuración. Esto podría ser objeto de una mejora futura, haciendo que se relacione esta selección de vehículo y terreno en la interfaz gráfica con los valores que se carguen en el modelo matemático.

Si seleccionáramos “CUBO” y “Terreno” se nos mostraría la siguiente simulación:

Ilustración 24 Simulación de "CUBO" en entorno submarino en Unity

Sin embargo, en caso de que eligiéramos OpenROV y “Piscina”:

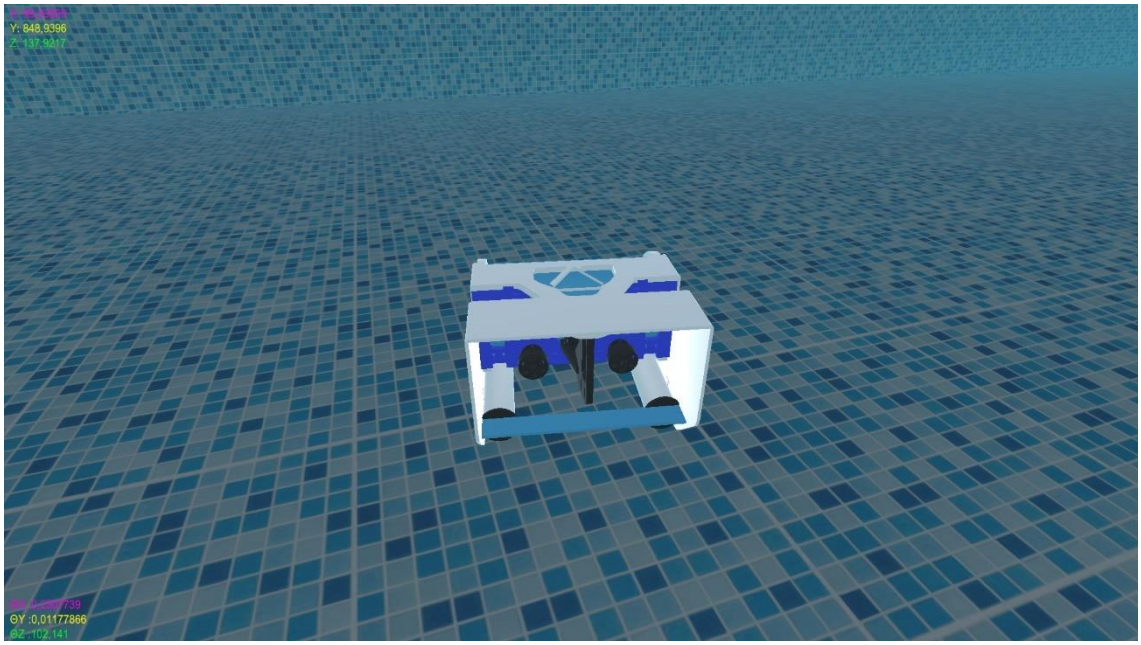


Ilustración 25 Simulación del OpenROV en piscina olímpica en Unity

También tiene implementado la posibilidad de cambiar de vista exterior a interior, para conseguir una mayor similitud con la forma real de control de los vehículos. Ver Ilustración 26.

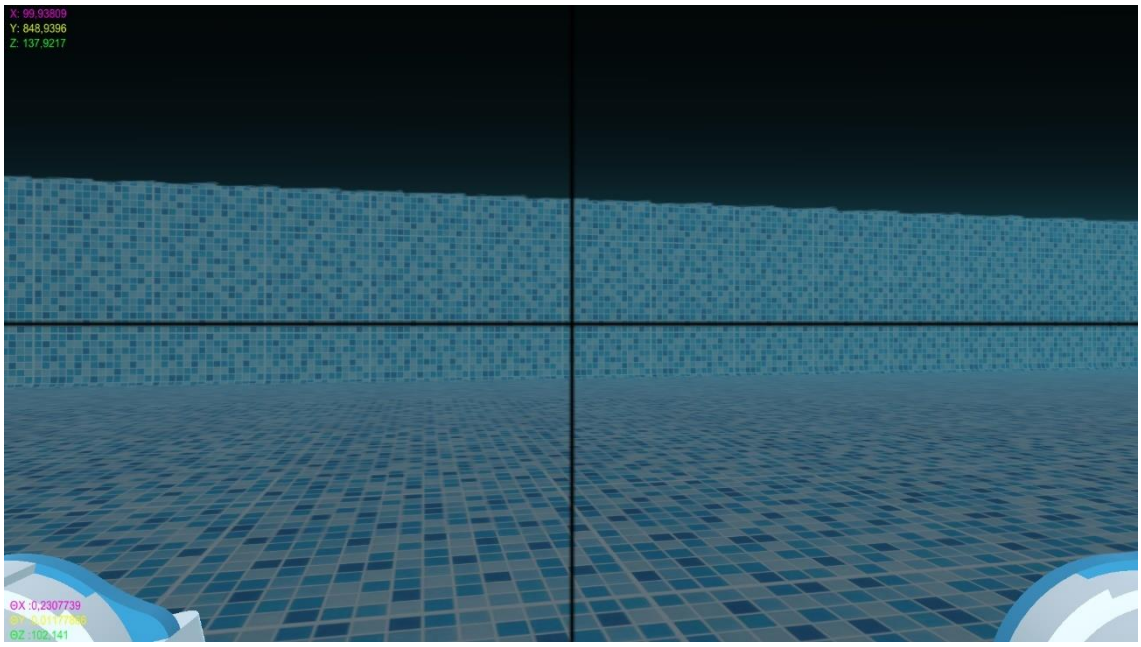


Ilustración 26 Vista interior desde el OpenROV en Unity

4.3.3. Control del simulador.

El simulador de Unity está pensado para permitir el control de la simulación del vehículo en tiempo real. Para ello se hace uso de un mando que dispone de 4 botones y dos joysticks.

Los controles asignados en el caso del vehículo OpenRov son los siguientes:

Ilustración 27 Controles de la simulación en Unity

Para activar y desactivar los PIDs hay que mantener el botón pulsado al menos un segundo. Cuando los PIDs estén activados, al dar una orden de giro o de ascenso/descenso, en lugar de mandarse la señal directamente como ocurre cuando no están activados, lo que ocurre es que se incrementa/decrementa el valor de la referencia que se está mandando a los PIDs, lo que hace que el control sea más suave.

Capítulo 5. OpenROV

5.1. Introducción

Para el desarrollo del presente proyecto, se ha usado un pequeño ROV comercial que fue comprado por la escuela de ingeniería industrial (ETSII) de la universidad politécnica de Cartagena y montado por Ignacio de la Cotera López. Tras la consulta con el antiguo director del centro, Don Antonio Guillamón, el cual fue el que adquirió el ROV, se llegó a un acuerdo con el director de la ETSINO (escuela técnica superior de ingeniería naval y oceánica) Don Gregorio Munuera, el cual era el beneficiario primero del ROV y conocedor del desarrollo que se llevaría a cabo en el proyecto de fina de estudios de Ignacio de la Cotera López. [1]

Se trata de un ROV desarrollado por una pequeña empresa de Berkeley (California) llamada OPENROV, la cual lleva años desarrollando varios modelos de ROV. El adquirido es el OPENROV 2.8., vendido por piezas y que debe ser montado por el comprador. Pensado para el desarrollo de ingeniería y estudiantes de grados universitarios en ingeniería. [1]

Este vehículo fue montado, tanto la parte electrónica como la mecánica, puesto a punto y probado por el alumno de esta misma universidad Ignacio de la Cotera en su proyecto de fin de grado, donde está documentado todo el proceso.

5.2. Sistema de referencia en OpenROV

Para referirnos a las direcciones de movimiento de movimiento y giro del vehículo es conveniente recordar los nombres y sentidos de estos referidos a este vehículo en concreto.

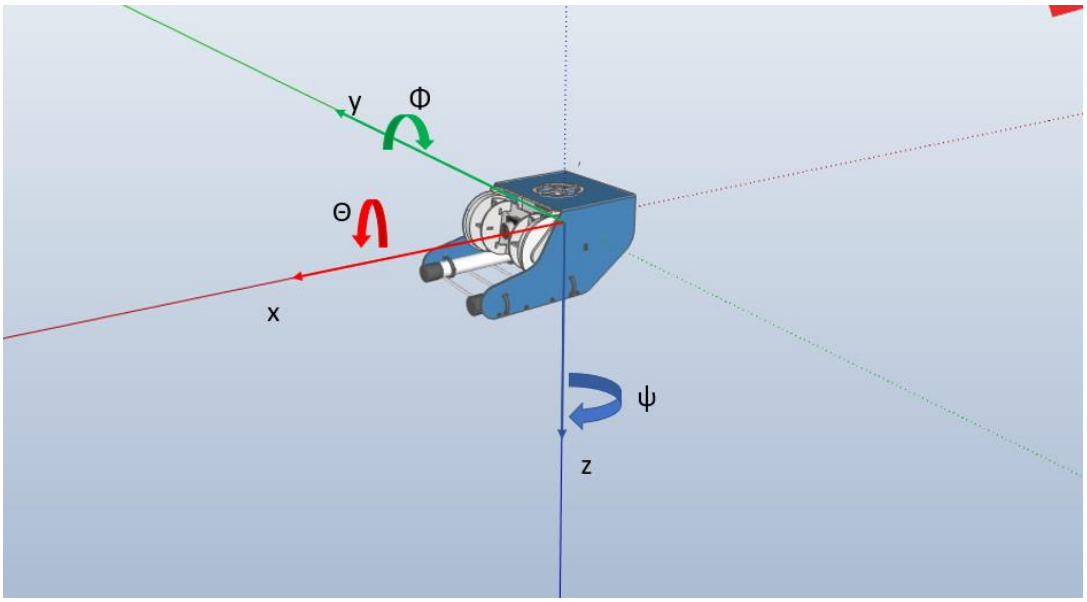


Ilustración 28 Ejes en el vehículo OpenROV

Posiciones y ángulos	Movimientos	Velocidades lineales y angulares	Fuerzas y momentos
x	Avance	u	X
y	Deriva	v	Y
z	Arfada	w	Z
Θ	Balanceo	p	K
Φ	Cabeceo	q	M
ψ	Guiñada	r	N

Tabla 17 Sistema de referencia en OpenROV

5.3. Aspectos físicos vehículo

OPENROV 2.8		California (USA)	Compañía: OpenROV
Características		**Especificaciones**	
Dimensiones	Eslora	30 cm	
	Alto	15 cm	
	Ancho	20 cm	
	Peso	2,505 kg	
Velocidad máxima	metros/segundos	1,028 (m/s)	
	nudos	2 (knt)	
Sistemas de comunicación		Comunicación por cable	
		(Ethernet y USB)	
		Beagle Board Black (Placa computadora de hardware libre)	
		PLC	
Sensores		Unidad de medición inercial (IMU)	
		HD Webcam with 120 degree FOV, 1080p	
		Scaling lasers	
Autonomía		2-3 horas	
Profundidad máxima de operación		100 metros	
Sistema de energía		Baterías de litio de tamaño 26650	
Sistema propulsivo		3 motores DST-700	
		3 álabes	
Material piezas		Metacrilato acrílico	
Adhesivo de unión		Pegamento acrílico líquido	
Sistema de construcción		Modular	

Tabla 18 Resumen propiedades del OpenROV

El vehículo en cuestión es un pequeño submarino controlado remotamente el cual dispone de las siguientes características básicas:

- Tres motores eléctricos que controlan tres hélices, dos en la parte trasera y uno en la superior. Mediante estos propulsores se puede controlar los movimientos de guiñada, avance y descenso.
- Los motores disponen de 5 niveles de velocidad, siendo 1 la más baja, y 5 la más alta.
- Una cámara dirigida en el sentido de avance del ROV que es visualizada en tiempo real desde el ordenador. También se pueden tomar fotos de la imagen capturada a través del software.
- Esta cámara también dispone de un motor eléctrico el cual hace elevar y disminuir el ángulo de dirección de enfoque respecto el eje paralelo a tierra en un rango de -60º hasta +60º de la horizontal.
- Luces LED para iluminar el entorno submarino y facilitar su control.
- Dos láseres que facilitan la percepción de las distancias y tamaños del entorno desde la vista proporcionada por la cámara.

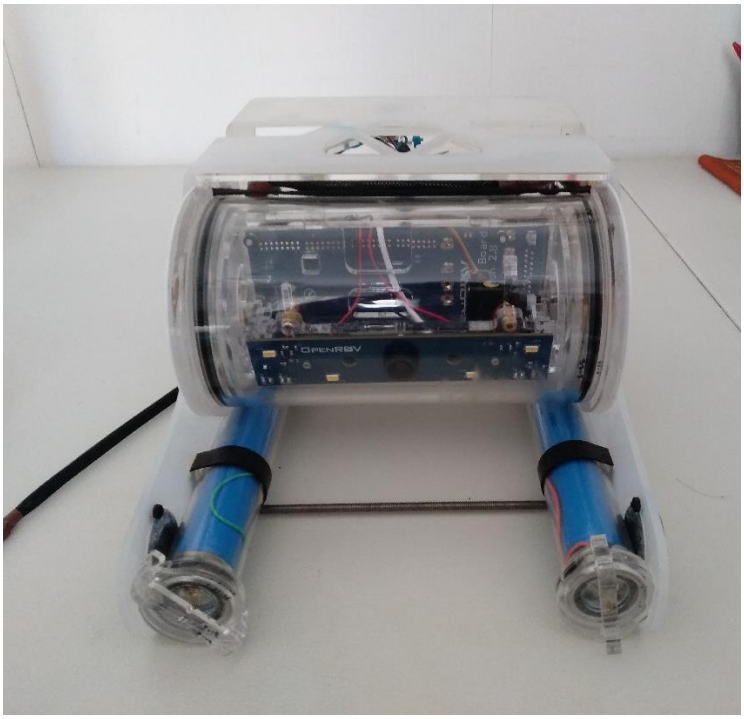


Ilustración 29 Vista frontal del OpenROV

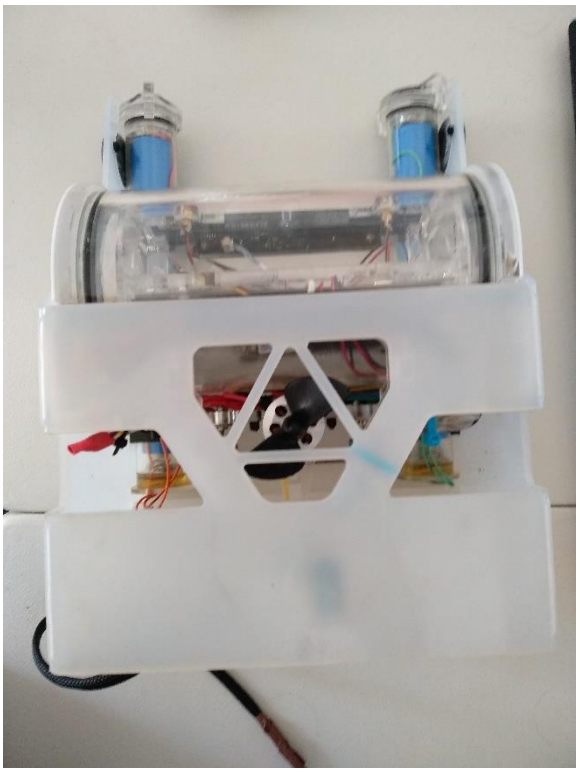


Ilustración 30 Vista superior del OpenROV

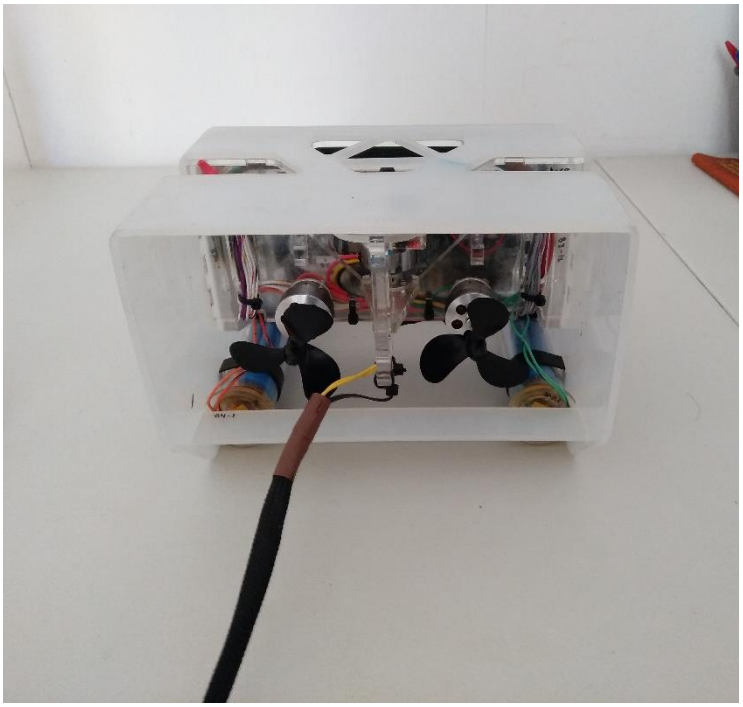


Ilustración 31 Vista trasera del OpenROV

5.4. Dispositivos de control del ROV

Desde un punto de vista más técnico, para comunicar, controlar y gestionar todas estas funcionalidades el ROV dispone de los siguientes componentes:

- Un Arduino, el cual es el encargado de realizar las tareas de más bajo nivel, es decir comunicarse directamente con los sensores y actuadores. Este se encarga de transformar en señales eléctricas las ordenes que recibe desde la BBB y de traducir las señales de la IMU en información entendible para la BBB. También se encarga de ejecutar cálculos que deben ser rápidos como las señales de actuación del PID para así evitar los tiempos de comunicación entre dispositivos.

- Un módulo IMU (Unidad de medición inercial) capaz de medir la presión, orientación, profundidad y aceleración del ROV en tiempo real, y emitir estos datos al ordenador.
- Una BeagleBoard Black (BBB) el cual es un ordenador de placa reducida, es decir un ordenador completo en un solo circuito. Este sistema es el que hace de intermediario entre el ordenador con el que actúa el usuario. Esta BBB se encarga de recibir los datos que le proporciona el Arduino y enviarlos al usuario y de recibir las ordenes enviadas por el mismo y comunicárselas al Arduino. También están programadas en la BBB la interfaz de usuario y las diferentes funcionalidades de reprogramación del Arduino.

Todo este sistema se comunica con el exterior a través de un cable que se conecta mediante un adaptador de ethernet a un ordenador estándar.

5.5. Características de los propulsores

5.5.1. Sentido de giro de las hélices

Un factor importante a tener en cuenta al analizar el comportamiento de las hélices del OPENROV es que debido a su forma y al lugar en el que están situadas, no se comportan de manera homologa girando en sentido de avance o en sentido de retroceso. La máxima potencia y eficiencia se consiguen con los propulsores funcionando en sentido de avance. Esto es debido a la propia morfología de las hélices, cuyo perfil parece estar diseñado con el objetivo de favorecer el empuje en este sentido, sacrificando el contrario. También influye la propia disposición de los propulsores, ya que se encuentran en un lugar el cual, al invertir el sentido de giro, el flujo de agua acelerado por el propulsor choca con el propio vehículo, lo cual frena el

movimiento, ya que parte del momento lineal que se genera por acción-reacción al impulsar el agua se pierde cuando esta impacta contra la carcasa del submarino.

Esto no se tiene en cuenta actualmente en el modelo propuesto y programado, lo cual es una mejora que puede ser planteada en el futuro para mejorar el simulador. Para esto habría que repetir los experimentos que hemos hecho para la hélice, pero en esta ocasión haciéndola funcionar en sentido de avance negativo. Una vez obtenidos los nuevos parámetros de modelado del propulsor deberíamos programar un sistema mediante el cual se aplique la ecuación con los parámetros correctos dependiendo del sentido de giro de la hélice.

5.5.2. Diferencias en el rendimiento de los propulsores en cada sentido de giro

Otra característica notable del sistema de propulsión de nuestro vehículo ROV es que, como es natural sobre todo al tratarse de un vehículo de coste bajo y no profesional, algunos de sus sistemas y dispositivos no son totalmente fiables y perfectos que es de la forma en la que se consideran en el modelo. Un ejemplo de esto es que los propulsores, aunque sean idénticos en teoría, en la práctica esto no es así. Si mandamos la misma orden a los dos propulsores que están orientados en la dirección de avance, estos no transmiten la misma potencia, si no que uno de ellos siempre genera una fuerza de propulsión mayor, por lo tanto, los momentos generados sobre el eje z no se anulan entre ellos. Esto causa que el momento resultante no sea nulo lo que causa que el vehículo se desvíe de la trayectoria teórica, y no ejerza una fuerza únicamente en el sentido de avance como se predice en el modelo.

Para solucionar este problema podemos seguir dos caminos. El primero sería experimentar con cada uno de los propulsores para obtener un modelo único para cada uno de ellos, consiguiendo así modelar y predecir así estas peculiaridades que diferencian a los propulsores en la práctica. Para esto necesitaríamos obtener los parámetros de los tres propulsores de los que consta y en el simulador aplicar las diferentes ecuaciones a las hélices de manera individual.

La otra aproximación para resolver esta inexactitud es intentar solucionar el problema en el vehículo real, haciendo así que su dinámica se corresponda a la teórica. Podríamos conseguir esto mediante la implementación de un sistema de control dentro del propio controlador integrado en el vehículo. Consideramos que esta solución es la más optima ya que no solo hace que nuestro simulador sea más correcto, si no que mejora el funcionamiento del vehículo en la realidad con una dedicación de tiempo y recursos relativamente baja, probablemente menor que la solución alternativa.

5.5.3. Error en el sentido de giro de las hélices

Cuando introdujimos el vehículo por primera vez al agua para comprobar su funcionamiento, este fue realmente malo. El control de la profundidad funcionaba correctamente, tanto de manera manual como usando el modo asistido por el PID, el problema aparecía en los movimientos de guiñada y avance.

Al enviarle la orden de avance el OpenRov en lugar de cumplirla giraba en círculos y ocurría lo mismo, pero en sentido contrario al enviarle la orden de retroceso. Sin embargo, al mandarle que girara hacia la derecha avanzaba, aunque con un ligero giro y con la orden de girar hacia la izquierda el vehículo retrocedía. Además, el control asistido mediante PID no era capaz de mantener la orientación estable. Al percibir estos problemas terminamos el experimento e intentamos averiguar cuál era el motivo del error.

Para ello primero inspeccionamos la programación del OpenRov Cockpit, comprobando que las teclas estaban asignadas a los movimientos correctos y que los mensajes se enviaban de la manera deseada desde el navegador a la BBB. Esta parte funcionaba sin errores, así que procedimos a inspeccionar el programa del Arduino, para comprobar que todos los movimientos se enviaban correctamente a los propulsores correspondientes. Todo parecía estar en orden también en la programación.

Al no haber encontrado la causa decidimos hacer otro experimento, esta vez metiendo el OpenRov en un recipiente pequeño con agua, en el que pudiéramos ver más fácilmente el funcionamiento de los motores. Con este método nos dimos cuenta rápidamente de cuál era el problema.

El error radicaba en que las dos hélices que actúan sobre el movimiento horizontal del submarino están dispuestas de tal manera que para ejercer fuerza en la misma dirección deben estar girando en sentido opuesto. Es decir, que el propulsor derecho debe girar en sentido antihorario para generar una fuerza de avance, mientras que el izquierdo debe hacerlo en sentido horario para generar el mismo movimiento, y esto no estaba contemplado en la programación del Arduino, sino que se consideraba que su giro debía ser igual en las dos hélices para moverse sobre el eje x sin rotar.

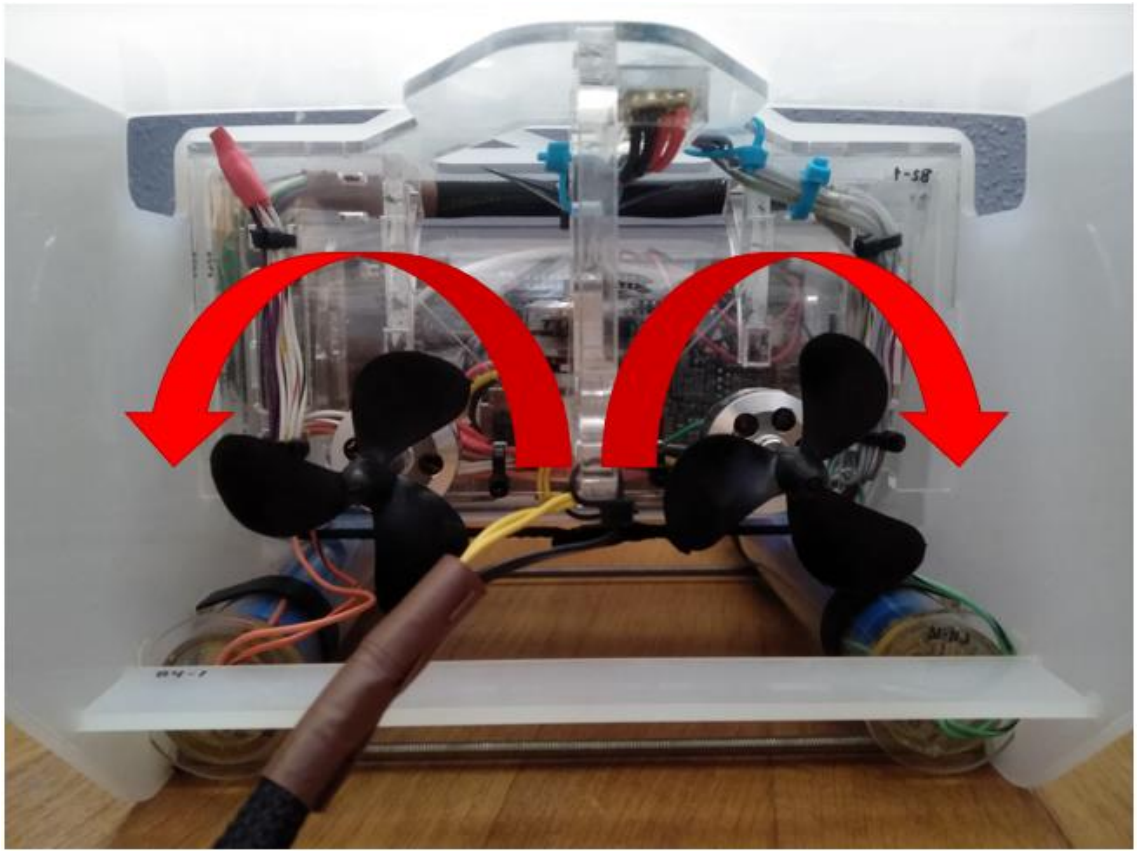


Ilustración 32 Parte posterior del OpenROV marcando el sentido de giro de las hélices para avanzar

Esto es debido a la morfología de las hélices, las cuales no son iguales, sino que están diseñadas de manera análoga para que cada una ejerza la mayor propulsión en sentidos de giro contrarios. Esto es muy conveniente ya que una hélice en funcionamiento genera un momento sobre el mismo eje sobre el que gira. Por ello, al girar las dos en sentido contrario estos momentos se cancelan.

Por lo tanto, procedimos a cambiar la programación del Arduino de manera que la señal enviada a los motores para efectuar cada movimiento sea coherente con esta distribución de las hélices.

5.5.4. Resumen de movimientos de las hélices

Finalmente, el movimiento de las hélices para cada movimiento quedaría de la siguiente forma:

- Movimiento de avance: El motor 1 gira en sentido antihorario, y el motor 2 en sentido horario. En este movimiento el motor 3 permanece inmóvil.
- Movimiento de retroceso: El motor 1 gira en sentido horario, y el 2 en sentido antihorario. En este movimiento el motor 3 permanece inmóvil.

- Movimiento de giro positivo (hacia la derecha, ya que el eje Z positivo es hacia abajo): Los motores 1 y 2 giran en el mismo sentido horario. En este movimiento el motor 3 permanece inmóvil.
- Movimiento de giro negativo (hacia la izquierda): Los motores 1 y 2 giran en el mismo sentido antihorario. En este movimiento el motor 3 permanece inmóvil.
- Movimiento de ascenso/ descenso: En este caso sólo gira el motor 3 (vertical), en sentido horario para el ascenso, y en sentido antihorario para el descenso.
- Movimientos combinados: controlando el vehículo con la introducción de comandos mediante el teclado del ordenador sólo es posible combinar los movimientos de dos direcciones, es decir, sólo puedo indicar al ROV que avance y descienda, por ejemplo, pero no puedo indicarle que avance y gire al mismo tiempo. Mientras que si se controla mediante el mando si es posible combinar todos los movimientos simultáneamente.

Capítulo 6. Programación OpenROV

El vehículo OpenROV, se puede entender como un sistema informático que consta de una parte hardware, el cual es el conjunto de elementos físicos que constituyen el sistema (Arduino, BBB, IMU, motores…) y una parte software, la cual es el soporte lógico que permite la interacción entre los componentes y ser capaz de realizar las tareas deseadas. En el capítulo anterior nos centramos especialmente en el hardware del sistema, mientras que en este estudiaremos en más profundidad la parte de software (programación, comunicación, control…), explicando su funcionamiento, y los cambios y mejoras implementadas en esta.

El sistema central mediante el cual consigue la comunicación e interacción entre los diferentes sistemas que componen el vehículo submarino es la aplicación Cockpit. Esta aplicación está integrada en el controlador y sus funcionalidades están programadas en el lenguaje JavaScript, mientras que la interfaz de usuario está desarrollada en lenguaje HTML. Además, está diseñada sobre el entorno Nodejs.

6.1. Nodejs

Node.js es un entorno en tiempo de ejecución multiplataforma, de código abierto, para la capa del servidor (pero no limitándose a ello) basado en el lenguaje de programación ECMAScript, asíncrono, con E/S de datos en una arquitectura orientada a eventos y basado en el motor V8 de Google. Fue creado con el enfoque de ser útil en la creación de programas de red altamente escalables, como, por ejemplo, servidores web.

Al contrario que la mayoría del código JavaScript, no se ejecuta en un navegador, sino en el servidor.

Para el uso que se le dará nos basta con saber que nos permite generar aplicaciones web en tiempo real, con conexiones bidireccionales, donde tanto el cliente como el servidor pueden iniciar la comunicación, lo que les permite intercambiar datos libremente.

En nuestro caso el papel de servidor lo hará la BBB y el de cliente el ordenador al que lo conectemos.

6.2. OpenRov Cockpit

Cockpit consiste en un código de JavaScript el cual se ejecuta a la vez en el navegador y en un proceso de Nodejs el cual se ejecuta en el sistema Linux del ordenador de a bordo del ROV.

El núcleo del programa Cockpit es una aplicación de una única página que tiene un conjunto estándar de objetos en el navegador, los cuales son conocidos por los plugins por lo que puede inyectar ahí sus propios elementos y protocolos.

La comunicación entre el proceso node.js y el explorador es mediante series de conexiones "socket.io" que envían eventos o mensajes en ambas direcciones. Ambos, el explorador y el proceso Node.js tienen retenedores y emisores que responden y envían mensajes entre ellos. Los plugins pueden acceder a este tráfico escuchándolo y emitiendo sus propios mensajes.

Este sistema se apoya en Node.js y Socket.io para realizar las comunicaciones tanto con el Arduino como con el ordenador de mando, y en MJPEG_Streamer para gestionar la visualización de la cámara.

Combinado estas potentes herramientas se consigue un programa sólido y con gran potencial de mejora. Además, estos programas disponen de una buena documentación los que facilita el desarrollo de mejoras y avances por la comunidad. Con Node.js y Socket.io, no solo se puede emitir video al explorador web, sino que también nos permite mostrar la información del vehículo obtenida mediante los sensores e incluso su control.

Es un programa de código abierto el cual se construye modularmente mediante una serie de "plugins" capaces de comunicarse entre sí a través de mensajes compuestos por cadenas de texto. La aplicación que está instalada en el vehículo ha sido desarrollada por la comunidad de OpenRov, y aunque actualmente esta comunidad sigue avanzando en este proyecto y lanzando nuevas versiones, hemos decidido mantener la versión instalada ya que es estable y está consolidada, por lo que es una buena base desde la que partir.

Una vez es conectado el cable de ethernet al ordenador, después de unos segundos puedes acceder desde el navegador del ordenador al servidor web que está integrado en el controlador. La interfaz que se te muestra es la siguiente:

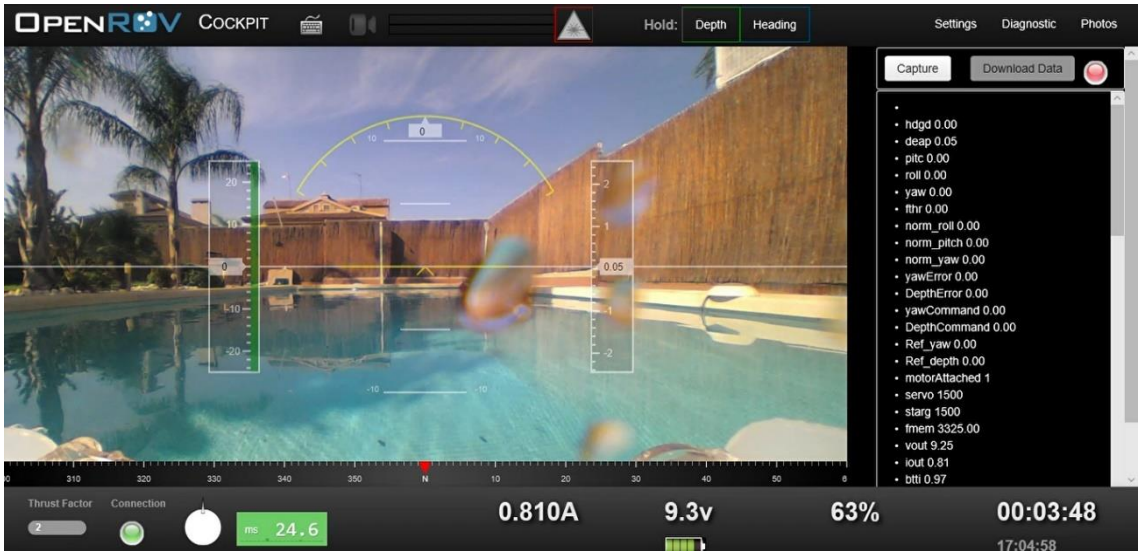


Ilustración 33 OpenROV Cockpit

6.3. Arduino

Como ya hemos comentado, el Arduino es el encargado de gestionar la parte de más bajo nivel del sistema, es decir, activar los motores, las luces, recibir la señal de los sensores, etc y trasladar la información tratada a la placa procesadora BBB.

Para realizar esta tarea de forma óptima debe tener una comunicación con la BBB, e intercambiar la información pertinente.

El Arduino está escuchando los comandos que le llegan desde el servidor, los cuales son enviados por el software Cockpit, pero también puede leer desde otros módulos de código los mensajes que el mismo envía.

6.3.1. Ordenes de acción a los motores

Las ordenes de movimiento de movimiento para los motores del vehículo se pueden calcular de diferentes formas. Por ejemplo, si estando conectados al sistema Cockpit usamos el joystick para controlar el avance del vehículo, se enviará un mensaje con el nombre “throttle” y un valor entre -100 y 100, pero a la vez se puede estar enviando la orden de que cambie el rumbo, lo que igualmente hará que se envíe el comando “yaw” con un valor entre – 100 y 100, exactamente lo mismo ocurrirá con el movimiento de ascenso y descenso, pero en este caso el mensaje comenzará con la palabra “lift”. Estos comandos pueden venir tanto del control manual o de las salidas generadas por el PID de control integrado dentro del propio Arduino.

Estos mensajes son escuchados por el Arduino dentro del objeto “CThrusters”. Este se encarga de gestionar las distintas ordenes de movimiento en cada dirección, para conseguir obtener el

valor que deberá enviar a cada uno de los diferentes motores y obtener la composición de movimientos deseada por el usuario.

El movimiento en el eje vertical es relativamente sencillo ya que es controlado únicamente por un solo motor, por lo tanto, tan solo hay que recibir el valor del comando enviado, y acotarlo a entre 1000 y 2000 ya que esta es la forma mediante la cual se envía el valor de potencia a los motores. Cuando se genera un 1000 el motor funciona a máxima potencia en sentido negativo, si recibe un 1500 el motor se mantiene apagado y con 2000 actúa con máxima potencia en sentido positivo. Para obtener el valor deseado para cada movimiento lo primero que se hace es dividir el valor que recibe entre 100, consiguiendo así un valor entre -1 y 1, después se multiplica este valor por 500 y por último se le suman 1500. Así tenemos un numero entre 1000 y 2000, que más adelante se transformará en una señal de acción hacía el motor.

El caso de los movimientos de guiñada (es decir, giro sobre el eje z) y de avance es más complejo ya que estos están relacionados entre ellos. Es decir, son efectuados por los mismos motores, los dos traseros. El movimiento de avance es proporcional a la media de las velocidades de los dos motores, mientras que el de guiñada depende de la diferencia entre las velocidades de estos dos. Por lo tanto, cuando se quieran efectuar estos dos movimientos a la vez se debe llegar a un compromiso entre ellos.

Para ello lo primero que se hace es calcular igual que en el caso del ascenso un valor entre 1000 y 2000 pero para el movimiento de avance. A continuación, se acota la entrada de giro entre -250 y 250. Este valor se suma y se resta al valor anterior. Un resultado será la salida de uno de los motores y la otra la del motor adyacente, siempre y cuando, el resultado de estas opreraciones no haya sido menor que el mínimo aceptado por el motor (1000) o mayor que el máximo (2000). Si esto ocurre se calculará por cuanto se excede y este valor será el offset, el cual será restado en los dos motores, para evitar la saturación de uno de ellos.

6.4. Comunicación entre BBB y Arduino

La comunicación entre la placa Blackbone y el Arduino se hace mediante mensajes que tiene el siguiente formato:

Los mensajes que se envían desde la blackbone al Arduino se componen de el nombre del comando seguido de unos paréntesis. Dentro de estos paréntesis están los valores numéricos asociados a este comando. Pueden tener ninguno, uno, o varios valores. Este sería un ejemplo de comando *yaw*:

"yaw(20)"

Al recibir este mensaje el Arduino lo divide en una cadena de texto donde se almacena el nombre del comando y un vector de numero enteros, donde la primera posición es el número de valores almacenados recibidos y los siguientes son estos valores recibidos. Por ejemplo, el mensaje de ejemplo anterior sería divido en *yaw* por un lado y el vector [1,20]. Así, si no se envía ningún valor junto con el comando, el vector creado sería el siguiente: [0].

Los mensajes generados por el Arduino tienen un formato ligeramente diferente, constan de una cadena de caracteres, a continuación, el carácter de *dos puntos* (":") y los valores numéricos asociados a ese mensaje. Para marcar el final de cada mensaje se cierra con *punto y coma*.

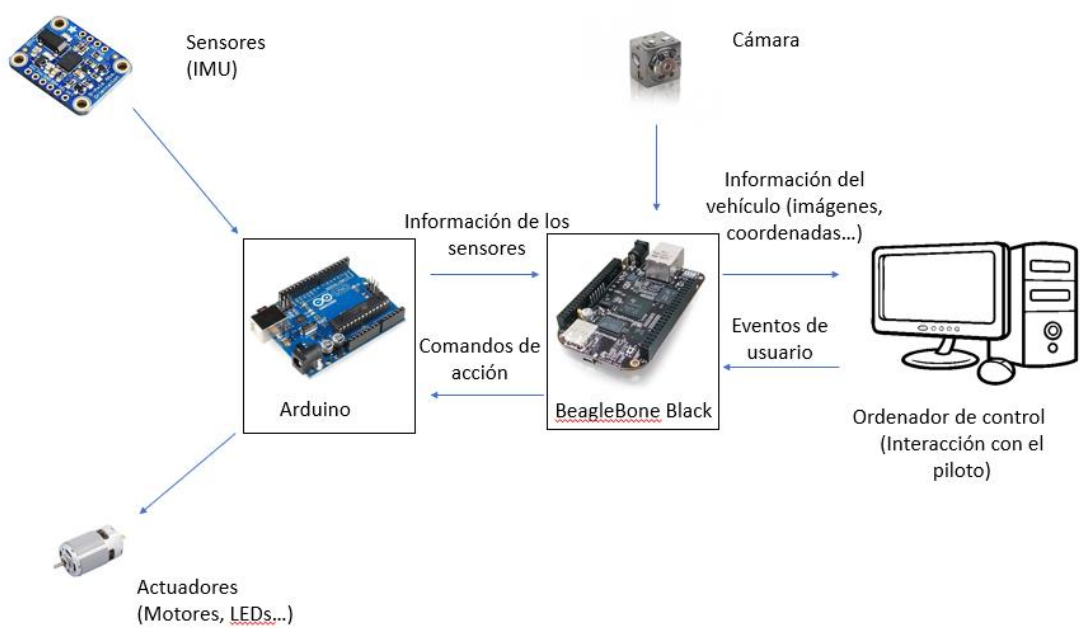


Ilustración 34 Esquema comunicación en el OpenROV

6.5. Extendiendo OpenRov Cockpit

Como sabemos, OpenRov es un sistema de código abierto, lo que nos permite acceder y modificar todo el código fuente a nuestro antojo.

La parte que realmente nos importa del sistema Cockpit está programada en JavaScript, lenguaje totalmente desconocido por mí hasta ahora. Debido a esto lo primero que hice para poder modificar el código fue acudir a bibliografía, tanto escrita como en forma de videos en la web, para familiarizarme con la estructura, funcionalidades y diferentes facetas de este lenguaje.

Una vez que adquirí unos conocimientos básicos para poder entender y escribir código en JavaScript realicé un proceso de aprendizaje similar para entender las bases de Nodejs.

6.5.1. Creación de un entorno de trabajo

Para poder tener un entorno en el que hacer pruebas y desarrollar nuestra modificación debemos ser capaces de ejecutar una versión básica del servidor en nuestro ordenador. Esto no es necesario ya que teniendo el ROV conectado al ordenador se puede modificar el código directamente, pero esto conlleva muchas limitaciones que hacen más tedioso el desarrollo, como es el tener que disponer del vehículo en todo momento, lo que impide trabajar desde casa, la biblioteca etc. También tiene la limitación del tiempo de duración de las baterías, que hace que tengas que parar a cargarlas periódicamente y, por último, la comunicación con la BBB es notablemente lenta sobre todo a la hora de subir archivos o descargarlos.

Para disponer de este entorno de trabajo lo primero que debemos hacer es instalar Nodejs, desde su página oficial:

https://nodejs.org

Una vez instalado nos descargamos desde GitHub los archivos del proyecto OpenRovCockpit en la versión 30.0.3 y los guardamos en una carpeta accesible.

https://github.com/OpenROV/openrov-cockpit/tree/30.0.3

Ahora debemos ejecutar Google Chrome, pero con la opción de acceder a archivos del sistema. Para ello ejecutamos el siguiente código llevando la precaución de no tener ninguna ventana de Chrome abierta:

```
$ Ruta/a/Chrome.exe  -allow-file-access-from-files
```

(Generalmente la ruta a Chrome es
C:\"Program Files (x86)"\Google\Chrome\Application\chrome.exe)

Cuando se abra Chrome pulsamos "Ctrl+o" y seleccionamos el archivo \openrov-cockpit-30.0.3 \src\static\testviews.html de la carpeta donde hemos descargado el programa. Se nos debe mostrar la siguiente pantalla:

Ilustración 35 Entorno de pruebas Cockpit

Si pulsamos "Ctrl+Shift+i" accederemos a la consola de comandos, donde podremos ver en tiempo real los mensajes que se están publicando en el navegador, lo cual no será de gran utilidad cuando desarrollemos nuestro plugin.

Si no interactuamos con la página tan solo se mostrará en esta consola los "pings" periódicos que hace el sistema, pero si interactuamos con ella podemos ver los diferentes comandos que se están publicando.

Por ejemplo, vemos que si pulsamos consecutivamente la tecla "i" se muestra el siguiente registro:

```
brightness_update          view index.js:11
1                          view index.js:12
brightness_update          view index.js:11
0                          view index.js:12
brightness_update          view index.js:11
1                          view index.js:12
brightness_update          view index.js:11
0                          view index.js:12
brightness_update          view index.js:11
1                          view index.js:12
```

Ilustración 36 Ejemplo mensajes navegador Cockpit

Esto quiere decir que se está enviando el comando 'brightness_update' con el valor 0 y 1 de manera alternada.

Con esta base ya podemos empezar a trabajar con el código de JavaScript. Para ello antes tenemos que descargar algún editor de código fuente, en este caso nos hemos decantado por "Visual Studio Code" un editor gratuito multiplataforma desarrollado por Microsoft.

La parte que realmente nos interesa para ampliar y mejorar el sistema se encuentra en "openrov-cockpit-30.0.3\src\plugins".

6.5.2. Creación de un nuevo plugin

Podemos comenzar a crear nuestro plugin podemos usar la herramienta grun-init, la cual nos creará los ficheros necesarios automáticamente.

Para ello comenzamos instalado grunt-init. Si ya hemos instalado nodejs anteriormente tan solo debemos ejecutar en la ventana de comandos de Windows el siguiente comando:

```
$ npm install -g grunt-init
```

Continuamos descargando la plantilla por defecto de OpenROV, para ello lo más sencillo es descargarla en la ruta C:\Users\TuUsuario\.grunt-init\openrov-plugin.

Esto se puede hacer manualmente o, si hemos descargado git, ejecutando el siguiente comando:

```
$ git clone https://github.com/openrov/openrov-grunt-init-plugin.git ~/.grunt-init/openrov-plugin
```

Ahora tan solo tenemos que movernos con la ventana de comandos a la ruta de una carpeta vacía y ejecutar lo siguiente:

```
$ grunt-init openrov-plugin
```

Una vez hecho estos nos pedirá que escribamos los detalles del plugin, como son nombre, autor etc.

Cuando hayamos rellenado este cuestionario vamos a la carpeta donde se han creado una serie de directorios y archivos. En nuestro caso lo hemos llamado "ayudacontrol", por lo que mostramos este plugin como ejemplo. La estructura de las carpetas es la siguiente:

Ilustración 37 Estructura de archivos de un plugin

En el archivo bower se almacenan los metadatos del proyecto, como son autor, versión, descripción…

En el archivo index.js es donde programaremos la parte del plugin que se ejecutará en el servidor integrado en el submarino. Aquí es donde debemos escribir el código que se encargará de recibir los mensajes enviados por el navegador y enviarlos al Arduino. Por ejemplo:

```
socket.on('holdHeading_on', function (hdg) { //Mensaje que recibe desde el navegador, seguido del valor "hdg"

      deps.rov.send('holdHeading_toggle(' + hdg + ')'); //Mensaje que envía al Arduino, con el valor "hdg" entre paréntesis

      console.log('holdHeading_toggle(' + hdg + ')'); //Lo publica también en la consola
    });
```

En este ejemplo lo que haría el programa es escuchar el comando “holdHeading_on” y su valor, y publicárselo al Arduino. El Arduino al recibir este comando ejecuta una parte de su código que activa el PID para llevar a el submarino a la posición de cabeceo que se le ha enviado.

Esta parte del código es común para todos los plugins, por lo tanto, no es necesario escribirla en el archivo index.js de nuestro plugin, sino que se puede escribir en cualquier otro plugin y actúa de la misma forma, ya que los mensajes que se publican en el navegador son accesibles desde todos los módulos.

En plugin.js es el archivo donde programaremos el núcleo del funcionamiento de nuestro plugin. En el definido un objeto con el mismo nombre que el asignado al plugin, y a continuación un

constructor, en el cual podemos definir las variables que vayamos a usar en el resto de las funciones.

```
ayudacontrol = function ayudacontrol(cockpit) {
        console.log("Loading ayudacontrol plugin in the browser.");

        // Instance variables
        this.cockpit = cockpit;
        var rov = this;      //Esta declaración nos permite usar “rov” para
referirnos a las variables y funciones del objeto



      //Variables que vamos a usar en nuestro programa
        rov.hdgPID = 0; //Aquí guardaremos la referencia de cabeceo

        rov.dptPID = 0; //Aquí guardaremos la referencia de profundidad
```

A continuación, hacemos uso de una función para registrar el uso del teclado por parte del usuario. Cuando detectamos la activación de una determinada tecla emitimos un mensaje que podrá ser leído por cualquiera de los plugins.

```
this.cockpit.emit('inputController.register', //Función para registrar el
uso del teclado
  [
    {
      name: "ayudacontrol.PIDgiroDerecha", //Nombre de la acción
      description: "Incrementa la referencia de cabeceo" //Descripción
      defaults: { keyboard: '0'}, //Tecla que la activa
      down: function() {
        rov.cockpit.emit('subirPIDhdg',10); //Mensaje que se envía junto
con                              su valor
     }
    },
```

Este mensaje será escuchado por la función “*cockpit.on*”, que recibe como parámetros el mensaje y la función que debe ejecutar cuando lo reciba. Por ejemplo:

```
rov.cockpit.on('bajarPIDhdg', function (c) {
```

```
    rov.bajarPIDhdg(c);
  });
```

Cuando reciba *bajarPIDhdg* ejecutará la función *bajarPIDhdg()* pasándole el valor recibido, *c*.

Dentro de esta función es donde realmente programaremos las acciones que queremos que se realicen al pulsar la tecla.

En este nuevo plugin "ayudaalcontrol" programaremos las nuevas características que queramos añadir al OpenROV, haciendo uso de una metodología de programación modular, lo que nos permitirá añadir nuevas funcionalidades sin tener riesgo de modificar indebidamente alguna característica programada anteriormente.

6.6. Mejoras realizadas al software del OpenROV

Una vez que ya entendemos la programación del OpenROV estamos capacitados para introducir mejoras y modificaciones en este, las cuales nos permitan facilitar el control del vehículo.

6.6.1. Implementación PID en OpenROV.

Con la versión que traía de serie la programación del OpenROV ya estaba incluido un sistema de control de tipo proporcional. Este estaba programado en el Arduino y su funcionamiento era sencillamente el multiplicar la señal de error calculada por una constante, lo que daba la señal de salida.

Este método funcionaba correctamente para, una vez conseguida una cierta guiñada y profundidad, mantenerla. Pero no daba tan buenos resultados si queríamos mantener estas dos variables mientras el vehículo avanza o cuando cambiamos las señales de referencia e intentamos seguirlas.

Por ello decidimos mejorar el sistema de control que trae programado e implementar un PID propiamente dicho.

Lo primero es cambiar en la configuración el archivo que debe compilar. Esto está hecho para que se pueda elegir fácilmente que controlador se quiere usar: si el proporcional que venía preconfigurado o el nuevo que hemos programado.

Accedemos al archivo AConfig.h y allí ponemos a 1 el tipo de controlador que queramos usar. Siendo "STD_AUTOPILOT" el únicamente proporcional y "EXP_AUTOPILOT" el PID.

```
#define HAS_STD_AUTOPILOT (0)
```

```
#define HAS_EXP_AUTOPILOT (1)
```

Ahora podemos empezar a desarrollar este controlador PID. Primero vamos a hacer que sea capaz de recibir y de actuar en consecuencia cuando reciba los comandos "holdHeading_toggle" y "holdDepth_toggle", que deben apagar el controlador cuando este esté activo y encenderlo cuando este apagado. Para ello definimos una variable que controle el estado actual de cada controlador, hacemos que dependiendo del estado de esta variable actúe de apagando o encendiendo los controladores cuando reciba estos comandos.

Una vez hemos configurado la interfaz y la comunicación ya podemos comenzar a centrarnos en el control de estas dos variables.

A continuación, programamos los dos PIDs en sí, de manera muy similar a la explicada en el apartado donde implementábamos el método de control de los 6 grados de libertad en el simulador. En este caso en lugar de tener que programar 6 PIDs tan solo necesitamos dos, ya que tan solo podemos controlar la guiñada y la profundidad.

Esto se hará mediante la creación de una clase llamada CPIDController. Está programada en el archivo CPIDController.cpp y es ahí donde se realizan los cálculos. En EXP_AUTOPILOT se crean dos objetos de esta clase, uno será el PID que controle la guiñada y otro el que controle la profundidad. Reciben como parámetros kp, kd, ki, los limites superior e inferior de la señal de salida y el tiempo de muestreo. Al inicializarlo recibe los punteros del setpoint, el valor real de la variable y la variable de salida. En esta última variable es donde guardará el valor que enviaremos a los motores.

Una vez hechos todos los cálculos por el sistema debemos mandar la orden de acción a los motores. Para ello transformamos la señal de acción que es un valor decimal entre (-1,1) a un valor entero entre (-100,100) y lo enviamos con los siguientes comandos:

```
int m_argumentsToSend[] = { 1, lift };
command.PushCommand( "lift", m_argumentsToSend );
```

6.6.2. Introducción de consigna manualmente

Tal y como estaba programado el OpenROV al inicio de este proyecto, a los PIDs solo se le podía mandar como señal de referencia la guiñada y la profundidad del vehículo en ese instante. Esto hacía que el sistema de control realmente no ayudara demasiado en el manejo del submarino, ya que su única funcionalidad era la de mantener la orientación y profundidad, pero para llegar a la posición deseada se debía hacer manualmente.

Por ello es útil la programación de un método por el cual se pueda cambiar la referencia de los PIDs a cualquier valor arbitrario. Esta funcionalidad hará que se pueda incrementar o disminuir el valor de la referencia un cierto valor pulsando un botón. Este método de interacción con el usuario nos pareció el más sencillo e intuitivo para evitar métodos más engorrosos como introducir valores numéricos, usar el ratón etc.

Para ello modificamos nuestro plugin para Cockpit que hemos creado anteriormente para que sea capaz de registrar la pulsación de las teclas asignadas a cada función, y actuar en consecuencia. Tomamos como primer valor de setpoint el valor que tenía cuando se activó el control mediante PID, y a ese valor le sumamos el incremento o decremento asignado a cada tecla.

6.6.3. Método de cambio de parámetros del PID

Como y hemos explicado, el sistema PID esta implementado en el Arduino. Para efectuar cualquier pequeño cambio en el programa este debe ser totalmente reprogramado, tardando entre 10 y 15 minutos en compilar y grabar el nuevo programa. Por ello, resulta tedioso realizar pruebas experimentales con diferentes parámetros de los PIDs para realizar un buen ajuste, ya que cada cambio lleva mucho tiempo en poder probarse, y teniendo en cuenta que los experimentos se deben realizar en una piscina o en el mar, no es una forma cómoda de trabajar debido a los muchos contratiempos que aparecen.

En los primeros experimentos nos encontramos con este problema a la hora de buscar los parámetros adecuados para el PID, ya que, al ser 6 parámetros diferentes, dependientes unos de otros, la parametrización mediante ensayo y error requiere de muchas pequeñas modificaciones progresivas en estos valores, para las cuales era necesario compilar el código del Arduino totalmente, lo que hacía que en toda una jornada de experimentación solo se pudieran hacer dos o tres inmersiones útiles.

Por ello se ha implementado un sistema mediante el cual se puede realizar la asignación de parámetros desde el sistema cockpit. Para ello tuvimos en cuenta que para la reprogramación de la parte de OpenROV Cockpit que se ejecuta en el navegador del ordenador tan solo tarda unos segundos (el tiempo que tarde en actualizarse la página de visualización). Para implementarlo hemos creado una función en el sistema Arduino para que sea capaz de recibir estos valores como comandos desde la BBB y aplicarlos al controlador.

También hemos modificado la parte del código de nuestro plugin que se ejecuta en el navegador de tal manera que al pulsar la tecla “d” se envíen los nuevos valores de los PIDs. Para ajustar

estos valores hay que acceder a la programación de Cockpit. En el archivo del plugin que hemos creado se cambian los valores, a continuación, se guarda y actualiza la página del navegador.

Así cuando pulsemos la tecla "d" se aplicarán los nuevos valores, proceso que es notablemente más rápido que reprogramar el Arduino, gracias a lo que podremos realizar más experimentos en menor tiempo.

Para cambiar estos valores debemos acceder a los archivos donde se guarda la programación del Cockpit dentro de la BBB, mediante el explorador de archivos de Windows, por ejemplo, escribiendo en la barra de navegación \\192.168.254.1.

Accedemos al archivo:

OpenROV/opt/cockpit/src/plugins/ayudaalcontrol/ayudaalcontrol.js

Ahí buscamos la siguiente parte:

```
  ayudacontrol.prototype.CambiarParametros = function CambiarParametros()
{
    this.cockpit.socket.emit('Cambiar_kp_yaw',1);
    this.cockpit.socket.emit('Cambiar_kd_yaw',0);
    this.cockpit.socket.emit('Cambiar_ki_yaw',0);
    this.cockpit.socket.emit('Cambiar_kp_depth',1);
    this.cockpit.socket.emit('Cambiar_kd_depth',0);
    this.cockpit.socket.emit('Cambiar_ki_depth',1);
  }
```

En ella podemos cambiar los 6 parámetros de los dos PIDs, siendo yaw el que controla la guiñada y Depth el que controla la profundidad.

Una vez cambiados estos parámetros seguimos estos pasos:

- Guardamos el archivo,
- Refrescamos la pestaña del navegador si tenemos abierto OpenROV Cockpit con la tecla F5
- Pulsamos la tecla "d"

Y se habrán actualizado los nuevos valores.

6.6.4. Calibración y puesta a cero

Las señales de los sensores integrados en el ROV son leídas por el Arduino periódicamente. Estas señales son interpretadas a través del programa implementado en este, siendo así transformadas finalmente en valores que representan magnitudes físicas.

Tanto las medidas de orientación dadas por la IMU como las de profundidad obtenidas mediante el manómetro son susceptibles de necesitar una puesta a 0 para establecer el punto de partida a partir del cual referenciar todas las medidas.

Además, en el proyecto de Ignacio Cotera se cometió un error a la hora de colocar el giroscopio ya que no se posicionó totalmente recto. En sus propias palabras:

"En un principio todo parece funcionar correctamente, pero al colocar el ROV sobre una superficie plana, paralela al plano de tierra, se aprecia en el monitor una ligera desviación en el balance y en el cabeceo.

Este error no puede ser de la IMU, ya que se comprueba que funciona con normalidad, por lo que es fruto de un mal montaje del acelerómetro de la IMU sobre el ROV. […] Al depositar la placa electrónica sobre la base acrílica y echar la resina epoxy, aparece un pequeño desnivel por la soldadura de los cables. […]Para corregirlo se le indicará, mediante el código de Arduino, que se ponga el nivel 0 de balance desde el valor igual a -3, y el nivel 0 de cabeceo desde el valor igual a -10." [1]

Aunque quedara recogido en la memoria, finalmente no se realizó al no encontrar la parte del código donde realizar este cambio.

Ilustración 38 Vista desde el OpenROV sin calibrar

En nuestro proyecto en lugar de simplemente corregir ese error, hemos implementado un sistema de calibración de todos los parámetros para poder ajustar a 0 todos los valores de los sensores, reparando así ese error y añadiéndole una característica útil en general.

Para realizar la calibración tan solo debemos colocar el vehículo en una superficie plana y pulsar la combinación de teclas “alt+4”. Hemos elegido esta combinación ya que si hubiéramos asignado esta función a una sola tecla podría ser pulsada accidentalmente durante el trascurso de una inmersión, lo cual puede ser problemático. Al ser más una opción de configuración que solo se debe usar una vez no es necesario que sea fácil de acceder.

Ilustración 39 Vista desde el OpenROV calibrado

6.6.5. Mostrar variables PID.

Para que la funcionalidad del cambio de los setpoints sea útil y sencilla de usar es conveniente que el piloto sea consciente de la referencia que está configurada en cada momento. Por ello es necesario mostrar esta información en pantalla, así como otras variables útiles sobre el funcionamiento actual del PID.

Con este objetivo hemos añadido a la lista de datos que se muestran en la pantalla de manejo del OpenROV las siguientes variables:

- **yawError**: Diferencia entre el valor de la referencia y el valor de guiñada.
- **DepthError:** Diferencia entre el valor de la referencia y el valor de profundidad.
- **yawCommand**: Valor de la señal de acción del PID de control de la guiñada. Comprendido entre (-1,1)
- **DepthCommand:** Valor de la señal de acción del PID de control de la profundidad. Comprendido entre (-1,1)
- **Ref_yaw:** Valor del setpoint para el PID que controla la guiñada.

- **Ref_depth:** Valor del setpoint para el PID que controla la profundidad.

Para poder mostrar estas variables debemos cambiar el archivo de la programación del Arduino NDataManager.cpp, para añadir estos valores a el resto que envía hacia la BBB. Por ejemplo, para enviar la referencia de guiñada debemos añadir dentro de la función NDataManager::OutputNavdata las siguientes líneas de código:

```
        Serial.print( F( "Ref_yaw:" ) );
        Serial.print( NDataManager::m_controllerData.yawSetpoint );
        Serial.println( ';' );
```

Así haremos de forma homologa con todas las variables que queramos enviar a la BBB y mostrar en pantalla. También nos servirá para leerlas y usarlas como valores en la programación del entorno Cockpit.

De tal modo que se mostrarían en pantalla a continuación del resto de datos de navegación del vehículo:

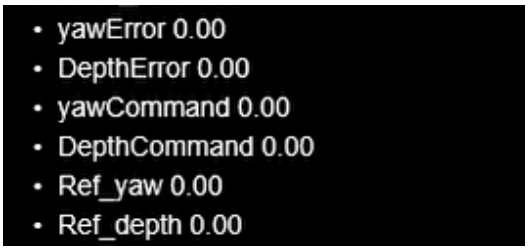


Ilustración 40 Información de estado del PID

6.6.6. Resumen funcionalidades añadidas.

Las teclas asignadas a estas nuevas funcionalidades, así como a las que trae por defecto el software instalado, son numerosas por lo que no es factible aprenderlas todas de memoria, por lo que en la pestaña de ajustes se puede acceder a cada la tecla y una pequeña etiqueta descriptiva. A modo de resumen podemos ver que estas son las nuevas características añadidas y como acceder a ellas:

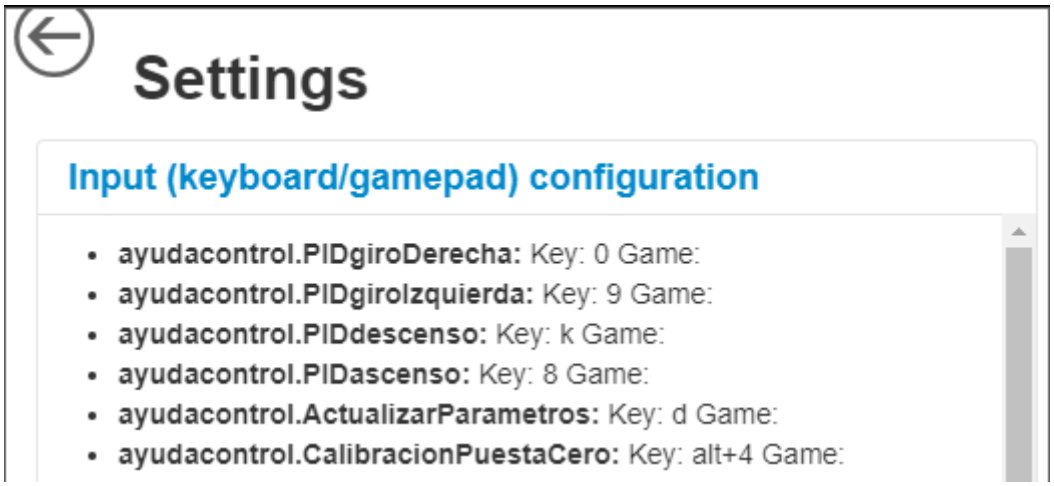


Ilustración 41 Funciones y teclas asignadas

- **PIDgiroDerecha:** Aumenta en 10º el setpoint para la guiñada, lo que corresponde a girarlo hacia la derecha.
- **PIDgiroIzquieda:** Disminuye en 10º el setpoint para la guiñada, lo que corresponde a girarlo hacia la izquierda.
- **PIDdescenso:** Aumenta en 0.1 metros el setpoint para la profundidad, lo que corresponde a descender.
- **PIDascenso:** Aumenta en 0.1 metros el setpoint para la profundidad, lo que corresponde a ascender.
- **ActualizarParametros:** Manda los parámetros de los PIDs que tenga programados en el entorno Cockpit hacia el Arduino.
- **CalibracionPuestaAcero:** Pone a cero todos los sensores. Se recomienda hacerlo antes de cada inmersión.

Capítulo 7. Ensayos OpenRov.

En este capítulo se presentarás los resultados de los diferentes experimentos realizados tanto con el vehículo real como con su simulación. Estos ensayos nos permitirán valorar la exactitud de los modelos considerados.

7.1. Ensayos vehículo real

7.1.1. Ensayos de movimientos básicos

Lo primero que debemos comprobar es la respuesta del vehículo cuando se accionan los movimientos básicos (Avance, retroceso, giro hacia babor, giro hacia estribor, descenso y ascenso)

En los experimentos de retroceso, giro hacia babor y giro hacia estribor se realizaron con el vehículo en la superficie del agua, por lo que la velocidad vertical no pudo se medida en esas condiciones, sin embargo, durante otros experimentos donde no se midieron los tiempos y distancias se ha observó que esta velocidad es prácticamente igual a la que se obtiene dejando que el vehículo flote libremente, por lo tanto se ha estimado con este valor.

Todos estos experimentos se realizaron con un nivel de potencia de los motores de 2 sobre 5, lo que corresponde a un 40% de la potencia máxima. Esto se hizo debido a que por las pequeñas dimensiones de la piscina donde se realizaron los ensayos era difícil y relativamente peligroso para el vehículo (por posibles choques con las paredes) realizarlos a máxima potencia.

7.1.1.1. Avance

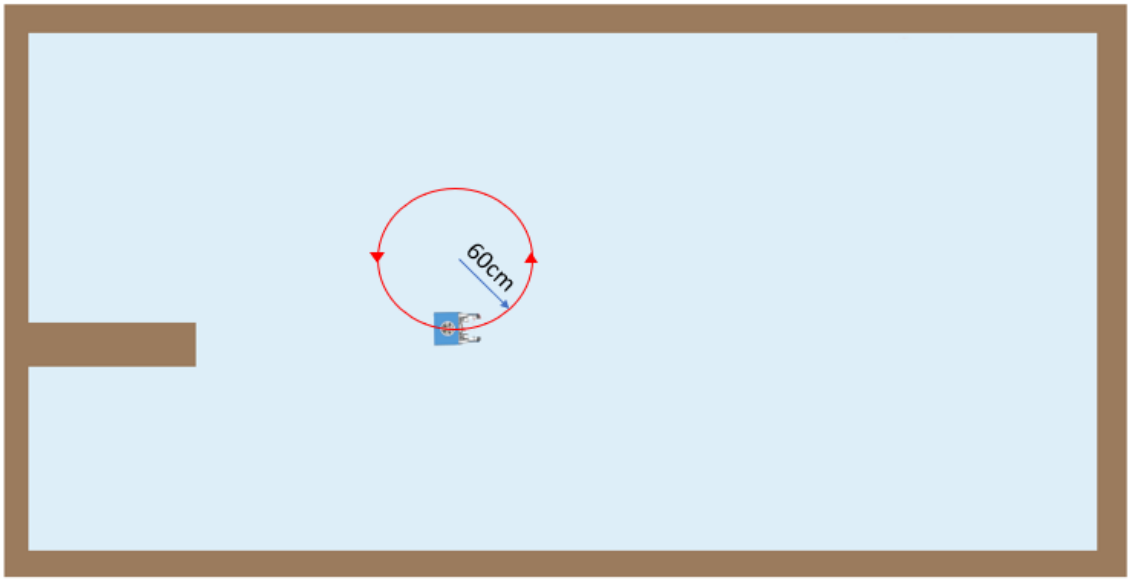


Ilustración 42 Vista superior del movimiento de avance

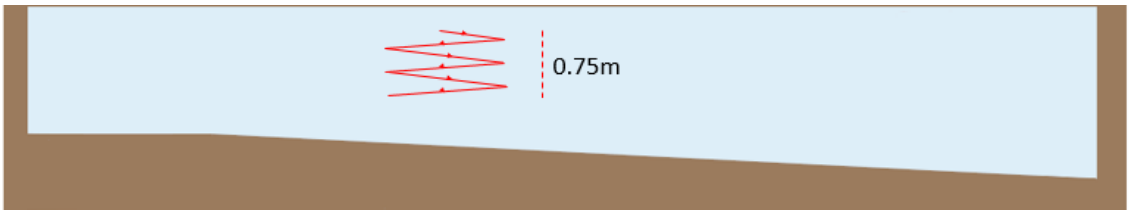


Ilustración 43 Vista lateral del movimiento de avance

Avance		Unidades
Tiempo	20	s
Distancia recorrida	9.42	m
Profundidad	0.75	m
Angulo recorrido	900	grados
Velocidad lineal	0.471	m/s
Velocidad vertical	0.0375	m/s
Velocidad angular	0.78539816	rad/s
Radio de giro	0.6	m

Tabla 19 Resultados ensayo movimiento de avance

Cuando se manda la señal de avance podemos observar como el vehículo en lugar de avanzar en línea recta como se podría esperar, describe una trayectoria circular de un radio aproximado de 60 cm. Esta trayectoria tiene un sentido de giro hacia babor, lo que parece coherente sabiendo que el motor izquierdo da una menor potencia que el derecho. Además de este movimiento de giro, también se produce una sumersión del vehículo conforme avanza.

7.1.1.2. Retroceso

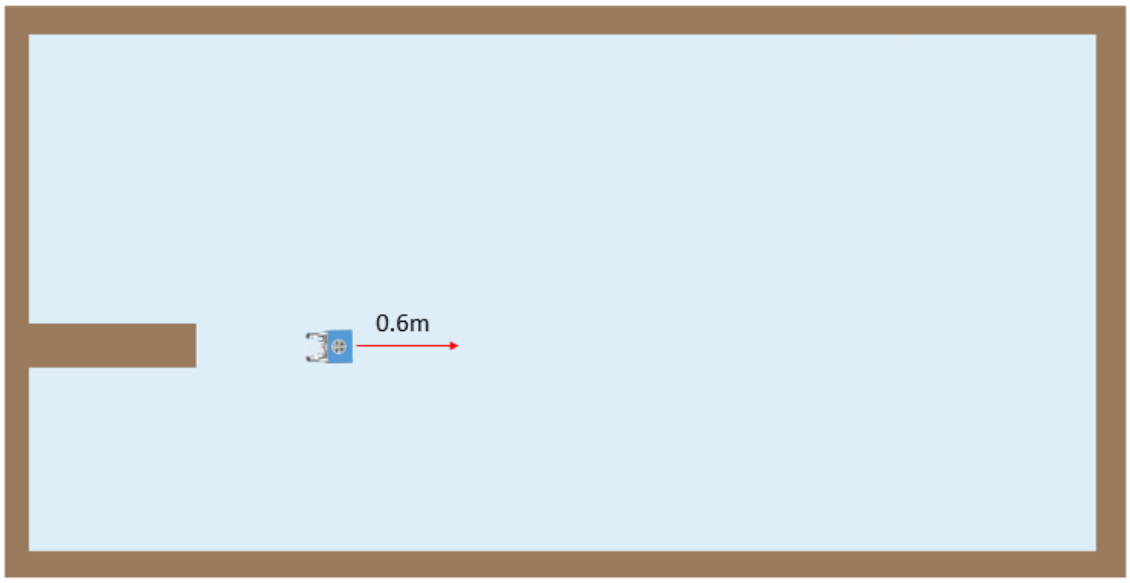


Ilustración 44 Vista superior del movimiento de retroceso

Retroceso		Unidades
Tiempo	6	s
Distancia recorrida	0.6	m
Profundidad	0	m
Velocidad lineal	0.1	m/s
Velocidad vertical	0	m/s

Tabla 20 Resultados ensayo movimiento de retroceso

El movimiento de retroceso se realiza en línea recta a una velocidad relativamente baja en comparación con la observada en el movimiento de avance. Esto es coherente sabiendo que las hélices tienen una morfología diseñada para dar máxima potencia en sentido de avance, mientras que esta se ve muy mermada cuando se hacen girar en sentido contrario. No se observa ningún movimiento de inmersión.

Es importante notar que este experimento se realizó con el vehículo en la superficie del agua, por lo que los resultados pueden ser diferentes a cuando este esté totalmente sumergido.

7.1.1.3. Giro hacia estribor

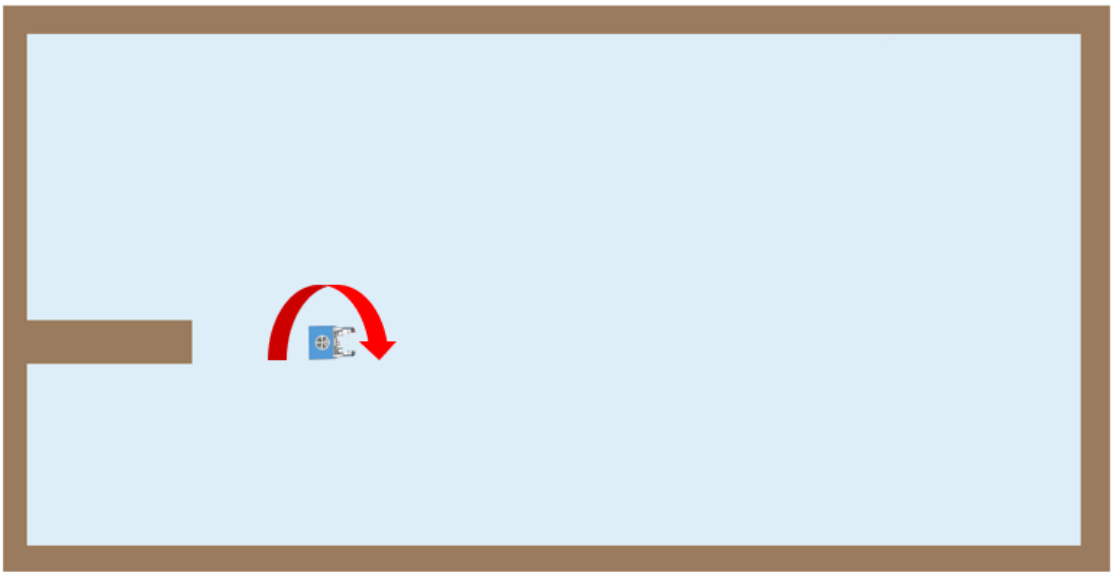


Ilustración 45 Vista superior del giro a estribor

Giro hacia estribor		Unidades
Tiempo	17	s
Distancia recorrida	0	m
Profundidad	0	m
Angulo recorrido	650	grados
Velocidad lineal	0	m/s
Velocidad vertical	0	m/s
Velocidad angular	0.66733177	rad/s
Radio de giro	0	m

Tabla 21 Resultados ensayo de giro a estribor

Cuando se le manda la orden de giro hacia estribor el vehículo comienza a girar sobre sí mismo en dicho sentido. El radio giro es prácticamente inapreciable. No se observa ningún movimiento de inmersión.

Al igual que el anterior este experimento se realizó con el vehículo en la superficie del agua, por lo que los resultados pueden ser diferentes a cuando este esté totalmente sumergido.

7.1.1.4. Giro hacia babor

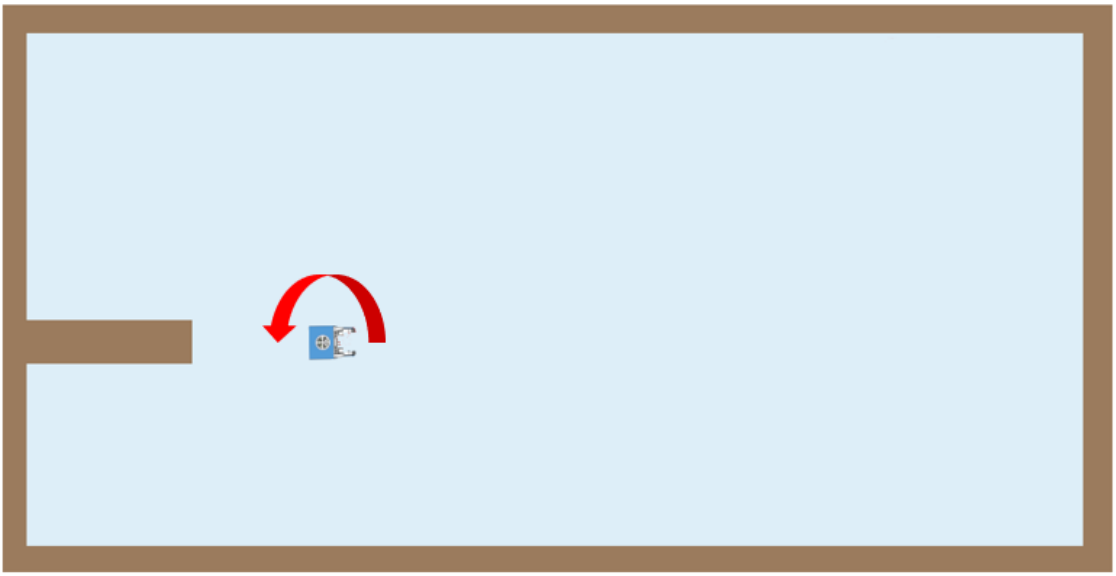


Ilustración 46 Vista superior del giro a babor

Giro hacia babor		Unidades
Tiempo	11	s
Distancia recorrida	0	m
Profundidad	0	m
Angulo recorrido	800	grados
Velocidad lineal	0	m/s
Velocidad vertical	0	m/s
Velocidad angular	1.26933037	rad/s
Radio de giro	0	m

Tabla 22 Resultados ensayo de giro a babor

Cuando se le manda la orden de giro hacia estribor el vehículo comienza a girar sobre sí mismo en dicho sentido. En este caso el giro es notablemente más rápido que en el giro hacia babor. Esto es coherente con que el motor izquierdo dé una menor potencia. El radio giro es prácticamente inapreciable. No se observa ningún movimiento de inmersión.

Al igual que los experimentos anteriores, este se realizó con el vehículo en la superficie del agua, por lo que los resultados pueden ser diferentes a cuando este esté totalmente sumergido.

7.1.1.5. Descenso

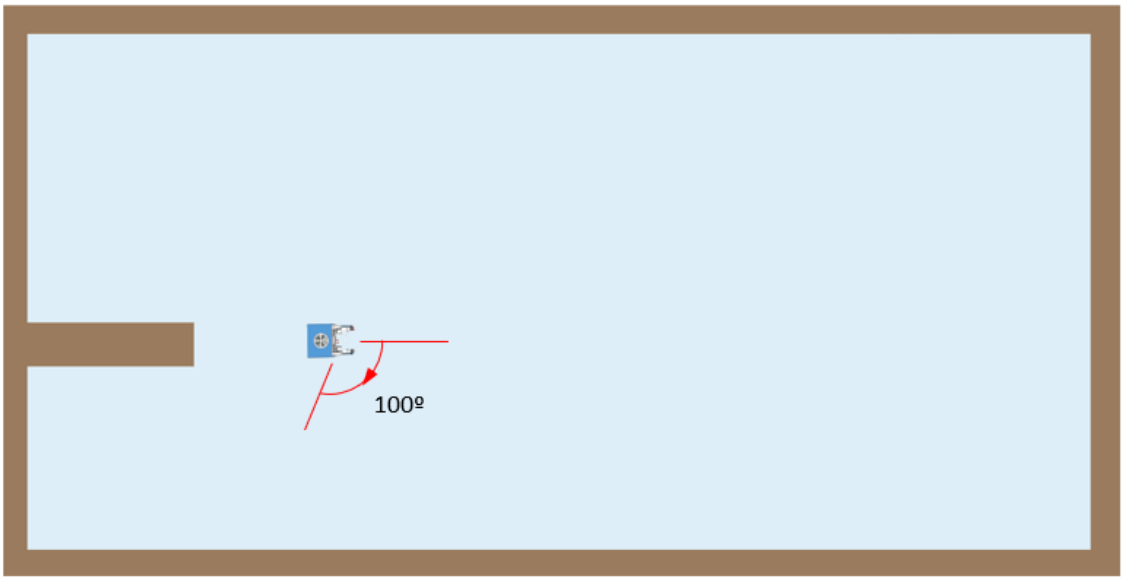


Ilustración 47 Vista superior del movimiento de descenso

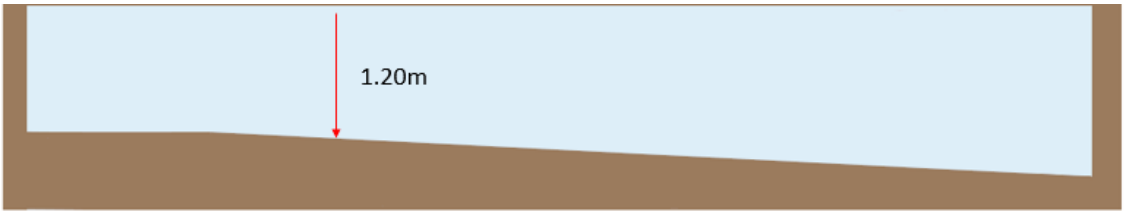


Ilustración 48 Vista lateral del movimiento de descenso

Descenso		Unidades
Tiempo	9	s
Distancia recorrida	1.2	m
Profundidad	1.2	m
Angulo recorrido	100	grados
Velocidad lineal	0.13333333	m/s
Velocidad vertical	0.13333333	m/s
Velocidad angular	0.2	rad/s
Radio de giro	0	m

Tabla 23 Resultados ensayo movimiento de descenso

Cuando mandamos la orden de descenso el vehículo comienza a descender realizando un giro sobre sí mismo relativamente lento en dirección de estribor. Este giro se debe a que la hélice superior genera un momento en su sentido de giro. Las velocidades de ascenso como de descenso so prácticamente idénticas.

7.1.1.6. Ascenso

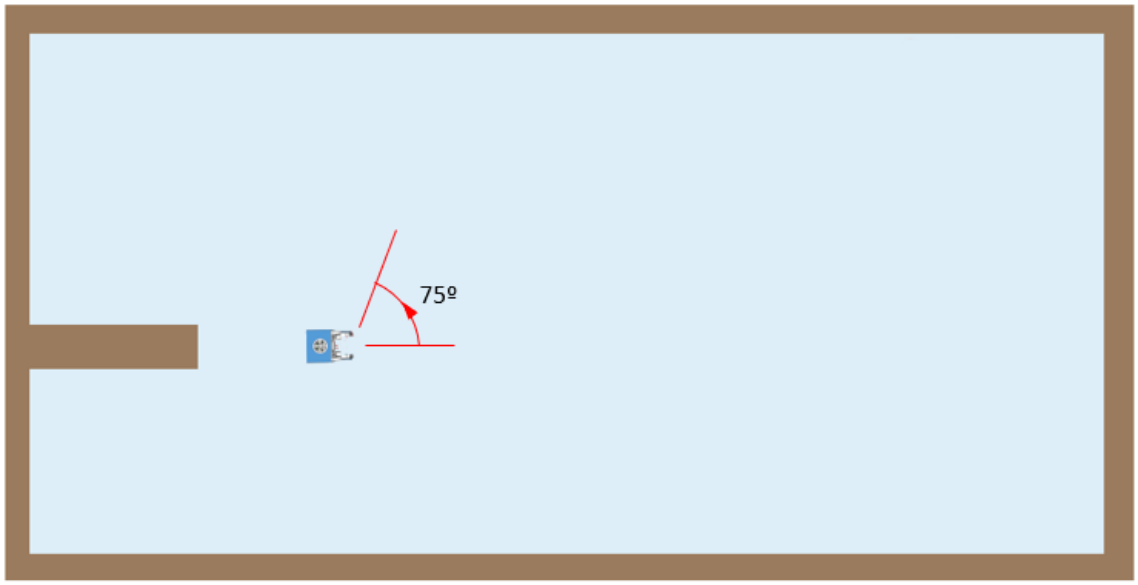


Ilustración 49 Vista superior del movimiento de ascenso

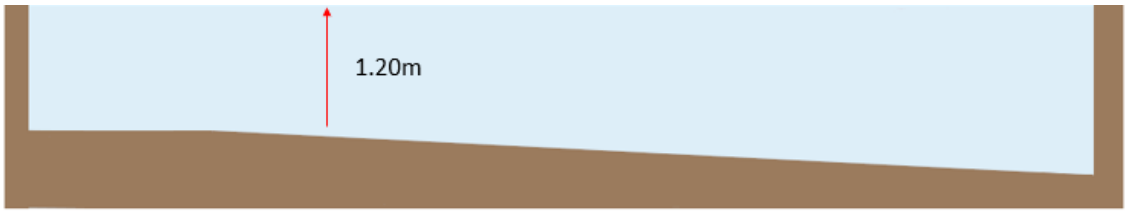


Ilustración 50 Vista lateral del movimiento de ascenso

Ascenso		Unidades
Tiempo	9	s
Distancia recorrida	1.2	m
Profundidad	1.2	m
Angulo recorrido	75	grados
Velocidad lineal	0.13333333	m/s
Velocidad vertical	0.13333333	m/s
Velocidad angular	0.1454441	rad/s
Radio de giro	0	m

Tabla 24 Resultados ensayo movimiento de ascenso

Cuando mandamos la orden de ascenso el vehículo comienza a ascender realizando un giro sobre sí mismo relativamente lento en dirección de babor. Las velocidades de ascenso como de descenso so prácticamente idénticas.

7.1.2. Ensayos PID

A continuación, realizamos un par de experimentos en los que el vehículo estará siendo controlado por los PIDs. Estos estarán parametrizados para comportarse únicamente como controladores proporcionales.

7.1.2.1. Ensayo 1

En este primer ensayo la potencia de los motores será de 2 sobre 5, es decir del 40%. Tan solo estará activo el controlador asignado a la guiñada. Este se activará para mantener la orientación paralela a las paredes laterales de la piscina y a continuación se le dará orden de avance. Los parámetros usados serán los siguientes:

Ensayo 1	Kp	Kd	Ki
PID guiñada	1	0	0
PID profundidad	Desactivado		

Tabla 25 parámetros PID ensayo 1

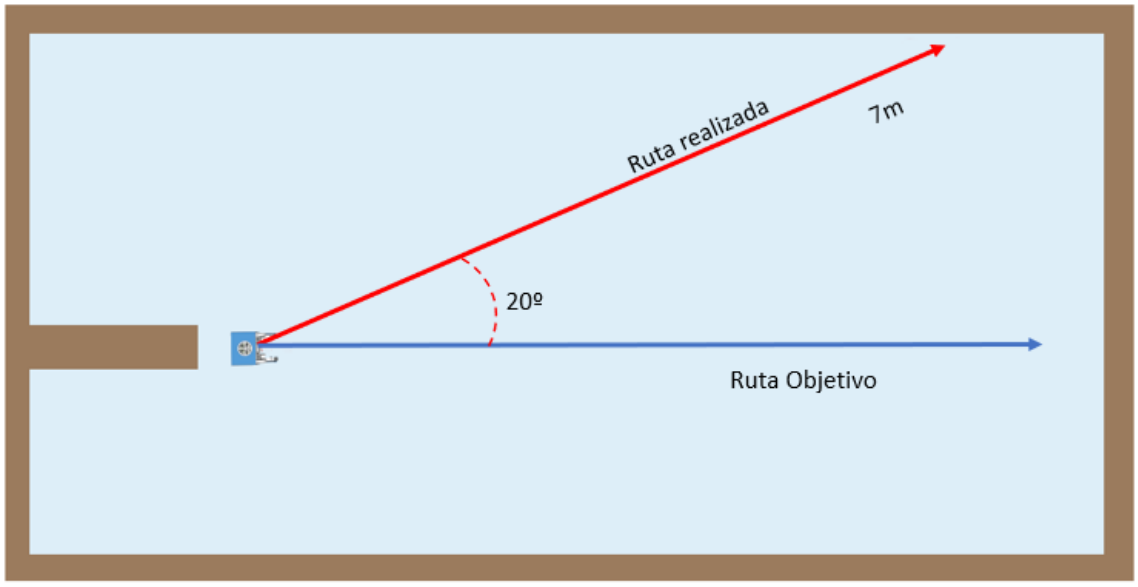


Ilustración 51 Vista superior del ensayo 1

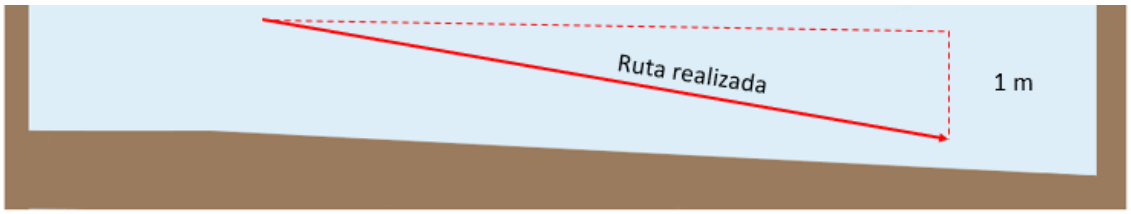


Ilustración 52 Vista lateral del movimiento del ensayo 1

Potencia de los motores: 40%

Ensayo PID 1		Unidades
Tiempo	14	s
Distancia recorrida	7	m
Profundidad	1	m
Velocidad horizontal	0.5	m/s
Velocidad vertical	0.14285714	m/s
Ángulo de desviación de trayectoria	20	grados

Tabla 26 Resultados ensayo PID 1

Al dar la señal de avance el PID no fue capaz de mantener la orientación deseada, sino que se giró unos 20 grados hacia babor, dirección a la que tiende a girar al no ser controlado. Pero aun con esta desviación sí fue capaz de mantener una trayectoria recta. Una vez dejamos de mandar la orden de avance el vehículo volvió a recuperar la orientación deseada. Este resultado es típico de un sistema de primer orden controlado con un controlador tipo P, por lo que se podría corregir esa pequeña desviación añadiendo una parte integral y derivativa. La profundidad va aumentando libremente, como es lógico, ya que no la estamos controlando.

7.1.2.2. Ensayo 2

En este segundo ensayo la potencia de los motores será de 5 sobre 5, es decir del 100%. Estará activados los controladores de guiñada y profundidad. Estos se activarán para mantener la orientación y profundidad que tenía el vehículo al comenzar el experimento y a continuación se le dará orden de avance. Los parámetros usados serán los siguientes:

Ensayo 2	Kp	Kd	Ki
PID guiñada	1	0	0
PID profundidad	1	0	0

Tabla 27 Parámetros PID ensayo 2

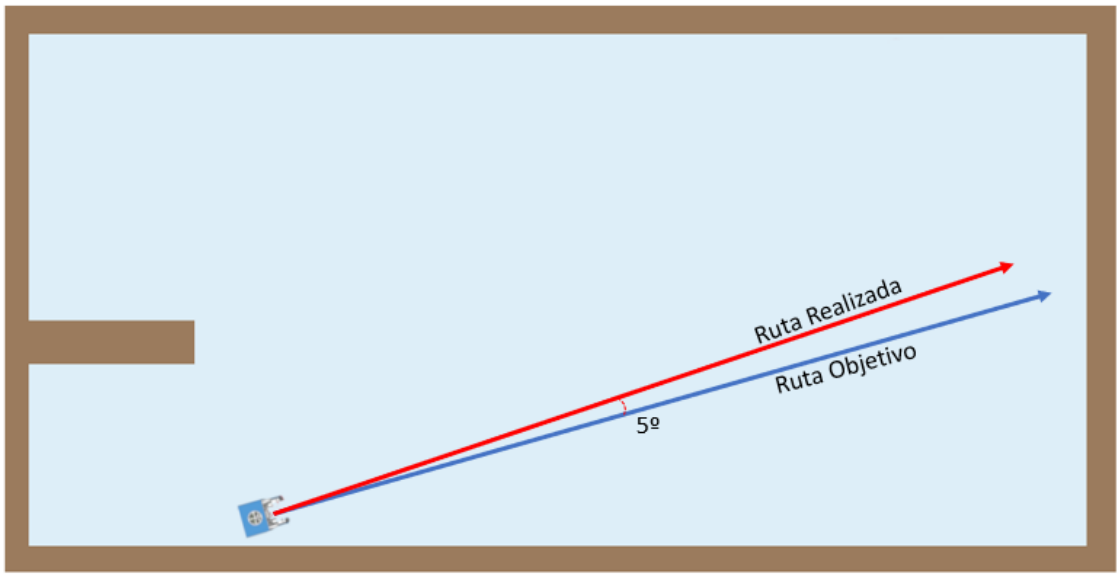


Ilustración 53 Vista superior del ensayo 2

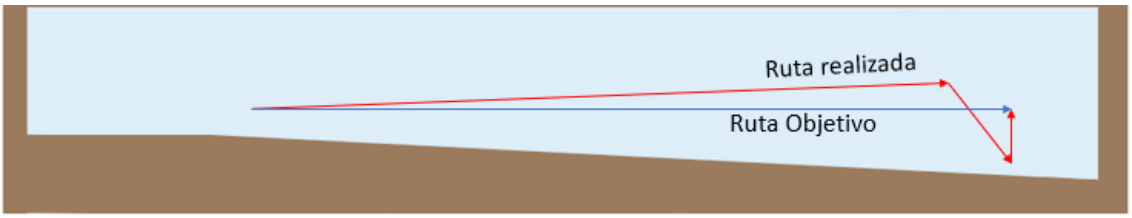


Ilustración 54 Vista lateral del ensayo 2

Potencia de los motores:100%

Ensayo PID 2		Unidades
Tiempo	6	s
Distancia recorrida	6	m
Profundidad	0	m
Velocidad horizontal	1	m/s
Velocidad vertical	0	m/s
Ángulo de desviación de trayectoria	5	grados

Tabla 28 Resultados ensayo PID 2

Vemos como en este caso, al tener los motores una mayor potencia, le es más sencillo mantener la orientación, por ello solo con el controlador tipo P, es capaz de desviarse tan solo 5º y de mantener una trayectoria recta. En cuanto a la profundidad, vemos que no es capaz de mantenerla exactamente como al inicio, pero sí que reduce el error considerablemente. Una vez dejamos de aplicar la orden de avance el controlador lleva al vehículo a la profundidad requerida.

Cuando este ensayo se realiza sin tener activado el controlador de profundidad, el vehículo asciende y desciende con gran velocidad sin control.

7.2. Ensayos en el simulador con el modelo 1

Igual que en hemos hecho con el OpenROV en la realidad, comprobaremos la respuesta del vehículo cuando se accionan los movimientos básicos (Avance, retroceso, giro hacia babor, giro hacia estribor, descenso y ascenso) en el modelo de OpenROV simulado.

Todos estos experimentos se realizaron con un nivel de potencia de los motores de 2 sobre 5, lo que corresponde a un 40% de la potencia máxima. Esto se hará para que sea más sencillo comparar los resultados con los obtenidos en los experimentos realizados al vehículo real.

Los parámetros hidrodinámicos utilizados en la simulación son los obtenidos en el TFG de Ignacio de la Cotera López [1] con ciertas correcciones que permitan realizar una simulación coherente.

Estas correcciones son básicamente el cambio de signo de la diagonal de coeficientes cuadráticos, ya que por error en el proyecto citado se consideraron positivos, lo que no es coherente con el sentido físico de esos coeficientes ya que, de ser positivos, en velocidades altas, el agua en lugar de generar una fuerza contraria al movimiento del vehículo la generaría a favor de este, lo que causa que la velocidad del vehículo tienda a infinito.

Corrigiendo este error todavía el funcionamiento no es coherente, ya que recibe una fuerza hidrodinámica exagerada en el movimiento de balanceo, lo que hace girar sobre sí mismo al vehículo a gran velocidad. Para corregir esto hemos disminuido en una orden de magnitud algunos de los parámetros referidos a este movimiento los cuales tenían valores exageradamente grandes.

En el anexo 2 se encuentran los parámetros exactos usados para las simulaciones.

Se muestran las gráficas de la trayectoria seguida en cada simulación, tanto una representación en 3D como las proyecciones en los ejes (X, Y) y (X, Z). El punto de partida se representa con un círculo rojo.

7.2.1. Movimientos básicos

7.2.1.1. Avance

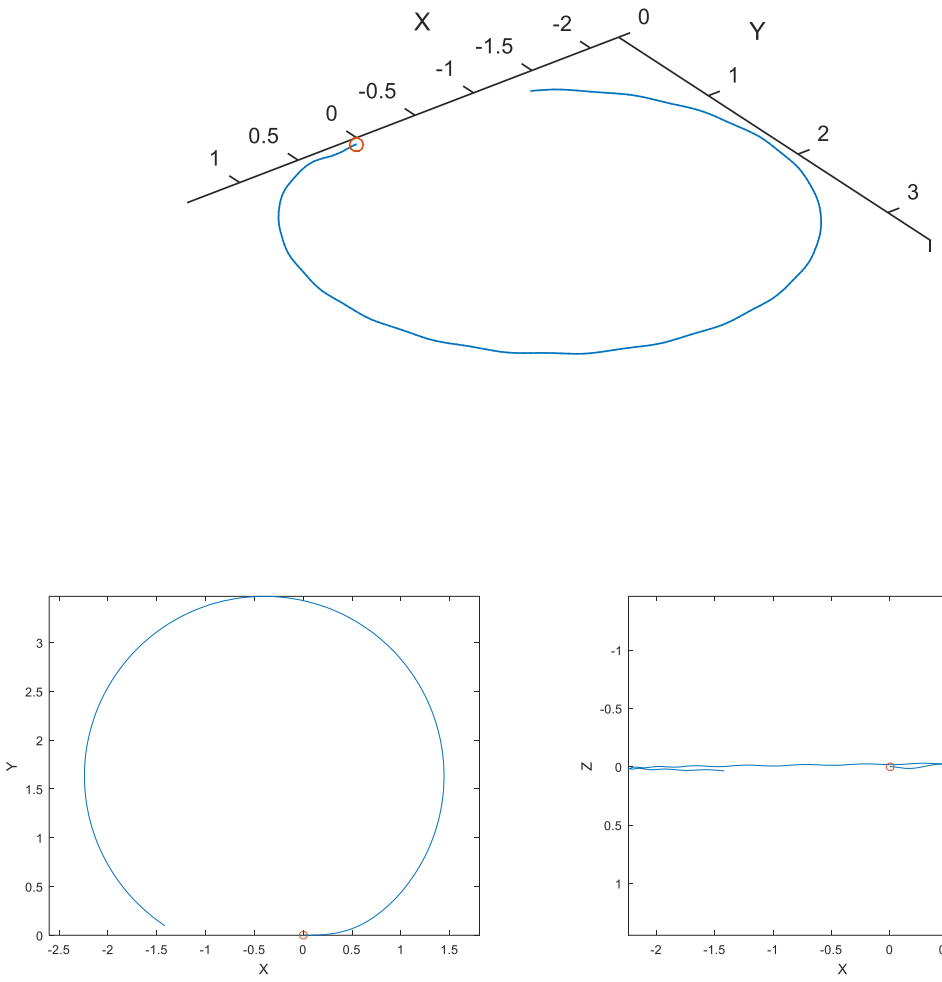


Ilustración 55 Simulación de movimiento de avance (Modelo 1)

	Avance	Unidades
Distancia total	10.05	m
Distancia vertical	0.05	m
Angulo rotado	-230	grados
Velocidad total	0.5025	m/s
Velocidad vertical	0.0025	m/s
Velocidad angular	-0.2	rad/s
Radio	1.75	m

Tabla 29 Resultados simulación de movimiento de avance (Modelo 1)

En esta simulación podemos ver que cuando se le manda la orden de avance, el vehículo realizar una trayectoria circular girando hacia babor, con un radio de 1.75m. Simultáneamente desciende, pero a una velocidad casi inapreciable.

7.2.1.2. Retroceso

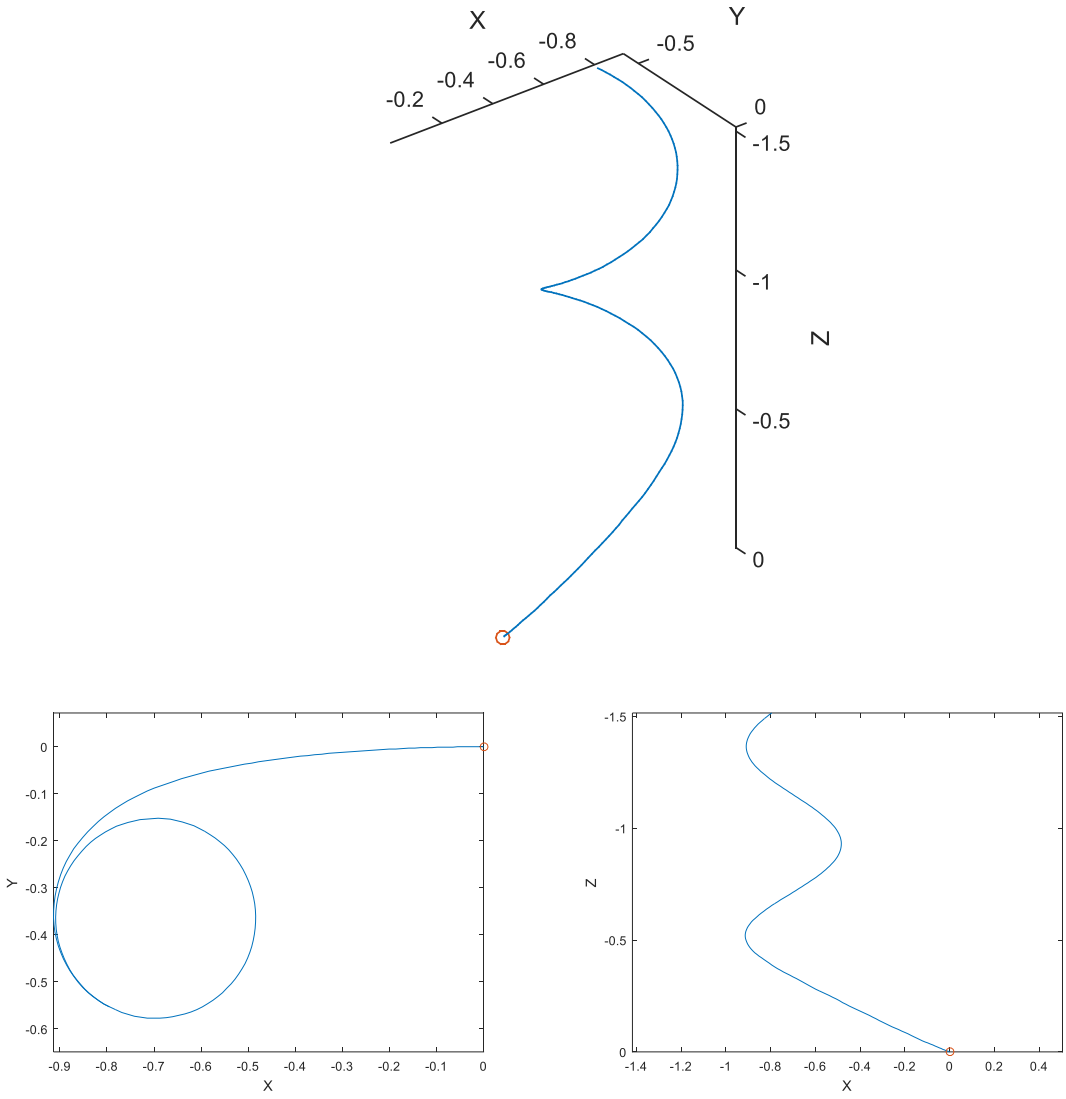


Ilustración 56 Simulación de movimiento de retroceso (Modelo 1)

	Retroceso	Unidades
Distancia total	3	m
Distancia vertical	-1.51	m
Angulo rotado	460	grados
Velocidad total	0.15	m/s
Velocidad vertical	-0.0755	m/s
Velocidad angular	0.401	rad/s
Radio	0.2	m

Tabla 30 Resultados simulación de movimiento de retroceso (Modelo 1)

Cuando mandamos la orden de retroceso el vehículo primero comienza a retroceder en línea recta hasta que los pocos segundos empieza a describir trayectorias circulares. Este movimiento lo acompaña con uno de ascenso.

7.2.1.3. Estribor

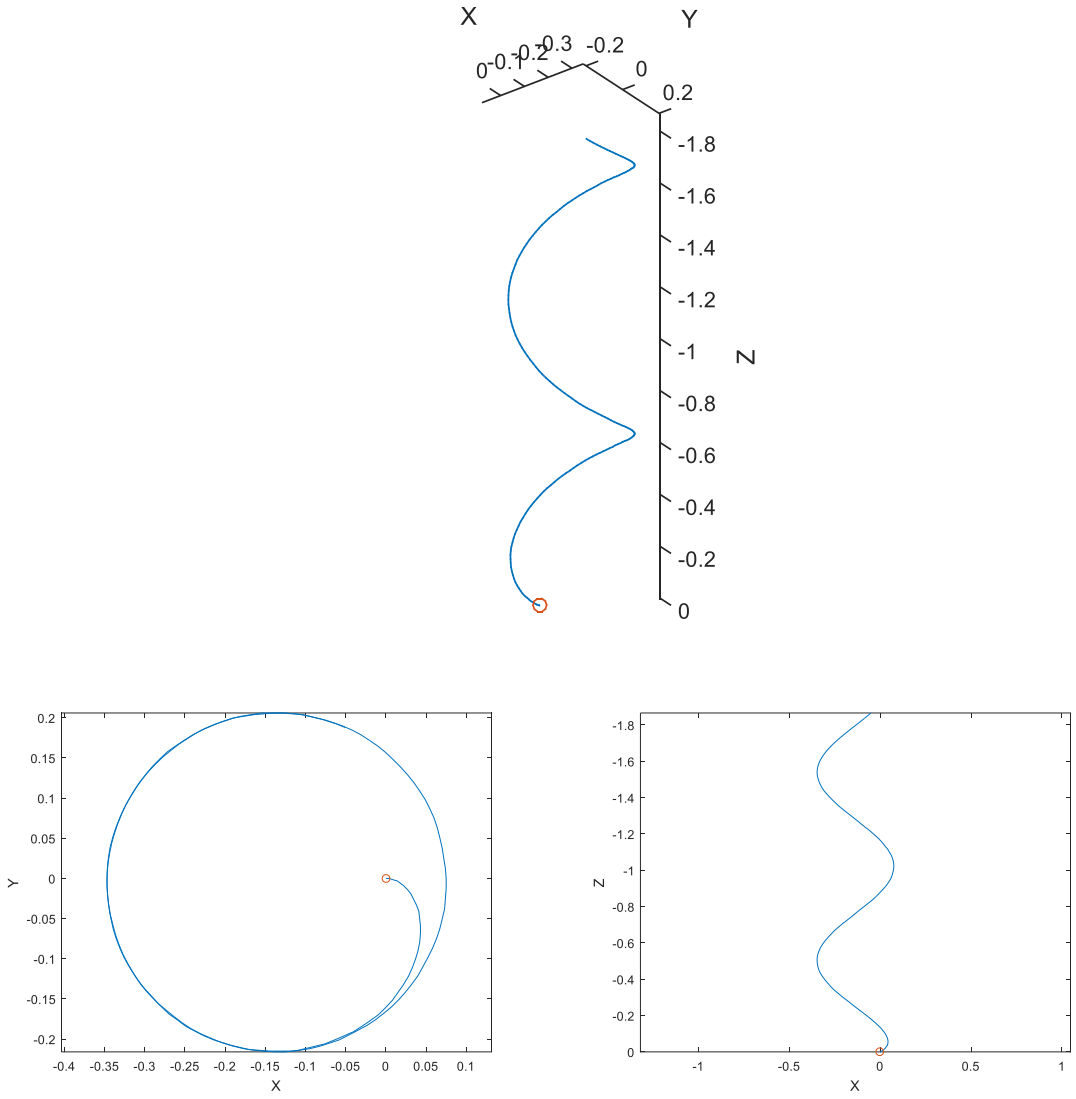


Ilustración 57 Simulación de movimiento de giro hacia estribor (Modelo 1)

	Giro estribor	Unidades
Distancia total	3.06	m
Distancia vertical	-1.866	m
Angulo rotado	690	grados
Velocidad total	0.153	m/s
Velocidad vertical	-0.0933	m/s
Velocidad angular	0.602	rad/s
Radio	0.19	m

Tabla 31 Resultados simulación de giro hacia estribor (Modelo 1)

Al recibir la orden de giro hacia estribor el vehículo comienza a girar en el sentido requerido, describiendo circunferencias de radio muy pequeño, lo cual es equivalente a girar sobre sí mismo. Simultáneamente el realiza un movimiento de ascenso.

7.2.1.4. Giro hacia babor

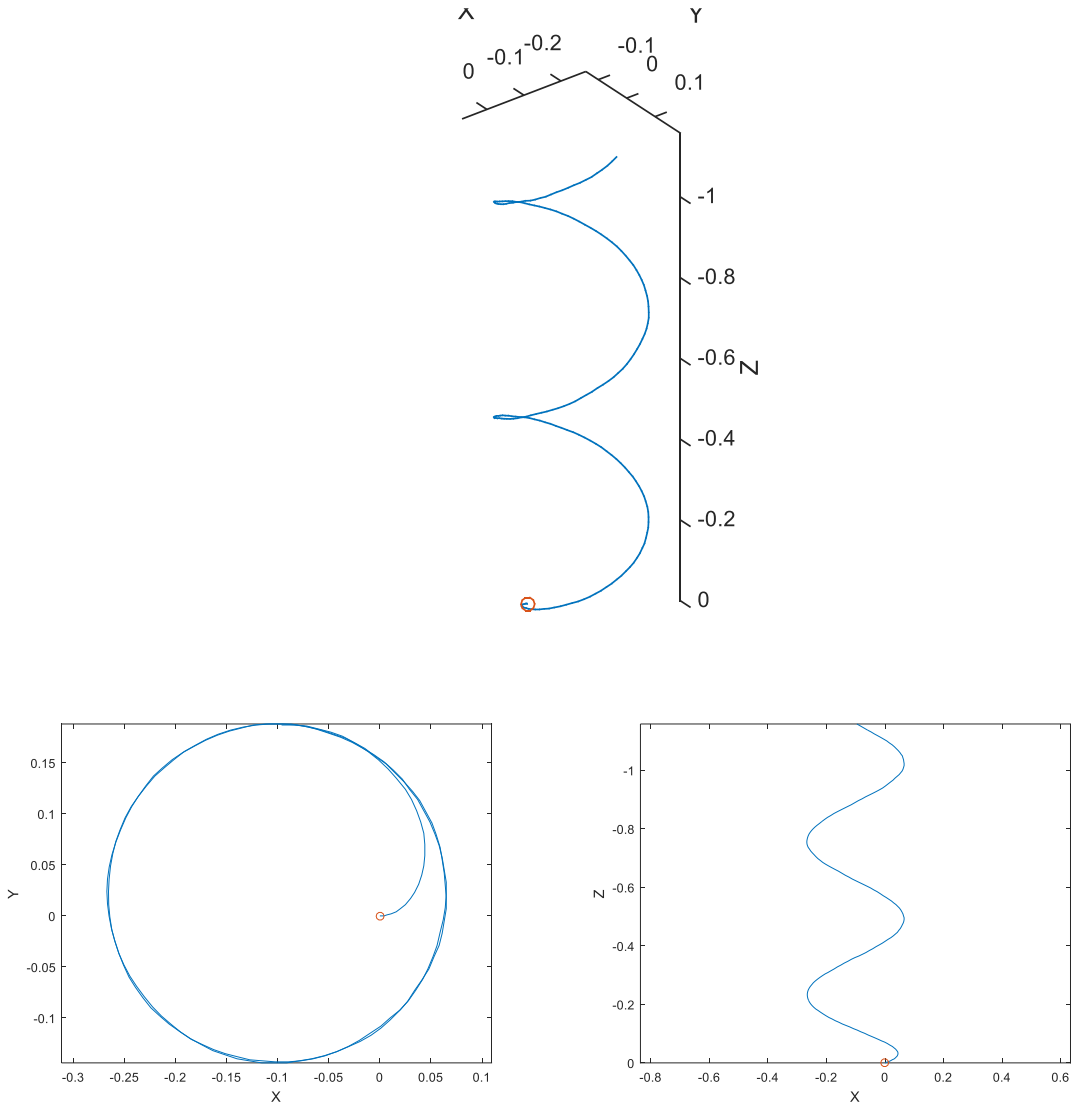


Ilustración 58 Simulación de movimiento de giro hacia babor (Modelo 1)

	Giro babor	Unidades
Distancia total	2.6462	m
Distancia vertical	-1.158	m
Angulo rotado	-847	grados
Velocidad total	0.13231	m/s
Velocidad vertical	-0.0579	m/s
Velocidad angular	-0.739	rad/s
Radio	0.15	m

Tabla 32 Resultados simulación de giro hacia babor (Modelo 1)

Al mandar la orden de giro hacia babor, análogamente a lo visto en el giro hacia estribor, el vehículo realiza la orden mandada girando sobre sí mismo y realizando un movimiento de ascenso.

7.2.1.5. Descenso

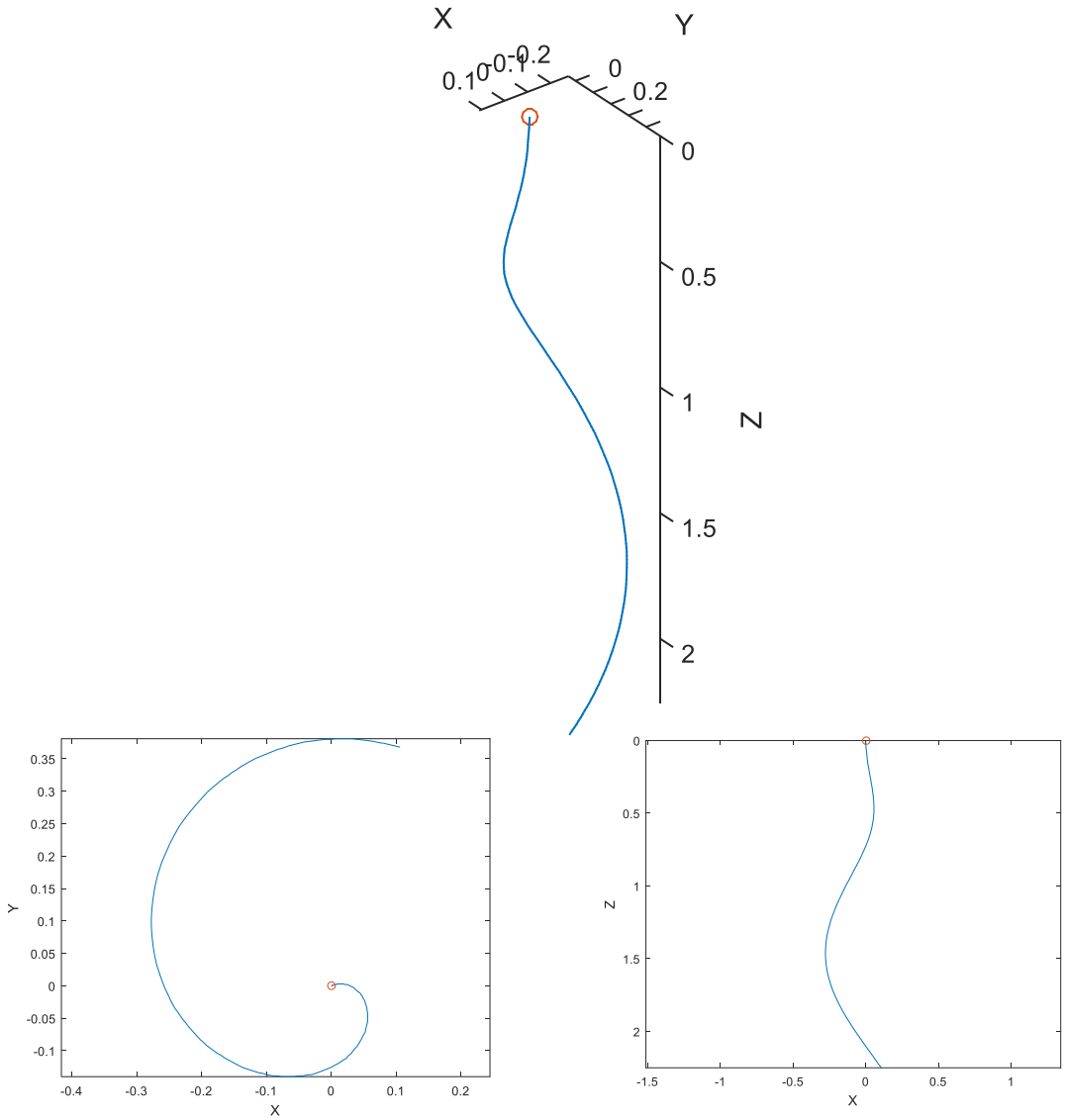


Ilustración 59 Simulación de movimiento de descenso

	Descenso	Unidades
Distancia total	2.49	m
Distancia vertical	2.315	m
Angulo rotado	605	grados
Velocidad total	0.1245	m/s
Velocidad vertical	0.11575	m/s
Velocidad angular	0.528	rad/s
Radio	0.3	m

Tabla 33 Resultados simulación de movimiento de descenso

El movimiento de descenso se realiza mediante la activación de la hélice superior del OpenROV, la cual hace que este se sumerja mientras realiza un giro hacia estribor. La circunferencia realizada por esta trayectoria es menos clara en este caso que en los anteriores, pero si dejamos a la simulación avanzar vemos que esta tiene un radio de aproximadamente 0.3m.

7.2.1.6. Ascenso

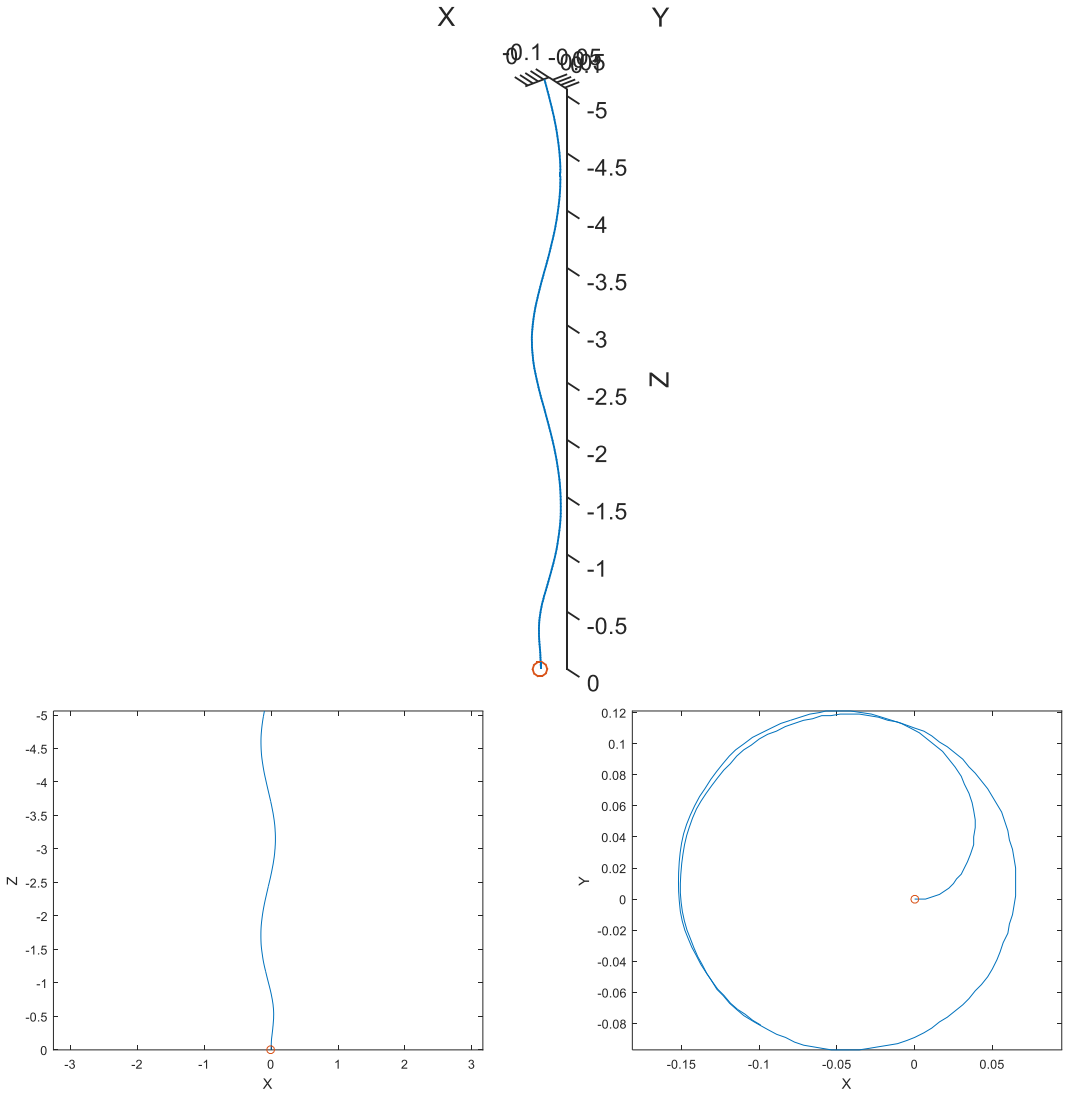


Ilustración 60 Simulación de movimiento de ascenso

	Ascenso	Unidades
Distancia total	5.2	m
Distancia vertical	-5	m
Angulo rotado	-620	grados
Velocidad total	0.26	m/s
Velocidad vertical	-0.25	m/s
Velocidad angular	-0.541	rad/s
Radio	0.1	m

Tabla 34 Resultados simulación de movimiento de ascenso

El movimiento de descenso se realiza mediante la activación de la hélice superior del OpenROV en sentido contrario que en el descenso, además se suma la propia flotación del vehículo por lo que la velocidad vertical es mayor que en el descenso. Mientras realiza este movimiento también gira hacia babor.

7.2.1.7. Resumen movimientos básicos

A continuación, podemos ver los resultados de los movimientos básicos resumidos en una tabla:

	Avance	Retroceso	Giro estribor	Giro babor	Descenso	Ascenso	Unidades
Distancia total	10.05	3	3.06	2.6462	2.56	5.2	m
Distancia vertical	0.05	-1.51	-1.866	-1.158	2.25	-5	m
Angulo rotado	-230	460	690	-847	350	-620	grados
Velocidad total	0.5025	0.15	0.153	0.13231	0.13	0.26	m/s
Velocidad vertical	0.0025	-0.0755	-0.0933	-0.0579	0.11	-0.25	m/s
Velocidad angular	-0.2	0.401	0.602	-0.739	0.3054	-0.541	rad/s
Radio	1.75	0.2	0.2	0.15	0.3	0.1	m

Tabla 35 Resumen resultados simulación (Modelo 1)

7.3. Experimentos en el simulador con el modelo 2.

Como ya hemos comentado en el punto 5.5.2 los dos propulsores traseros del vehículo no son iguales en la práctica. El motor derecho es capaz de proporcionar una mayor velocidad de giro que el izquierdo y por lo tanto genera una mayor fuerza de propulsión, por lo que el modelo anteriormente considerado no es fiel a la realidad.

Aunque no hemos realizado ensayos precisos para conocer los parámetros reales de cada propulsor, el rendimiento del propulsor izquierdo lo podemos estimar como un 70% del teórico.

A continuación, mostraremos los resultados obtenidos con el modelo anteriormente descrito.

7.3.1. Movimientos básicos

7.3.1.1. Avance

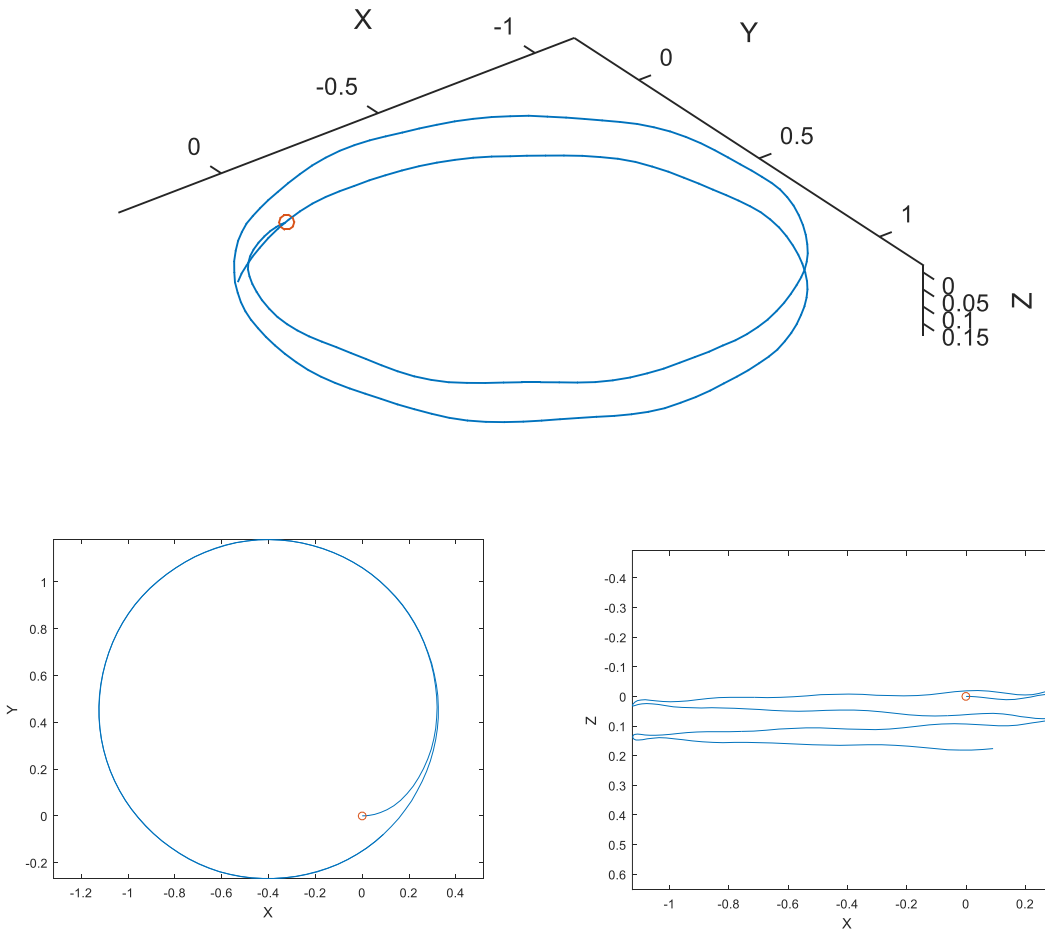


Ilustración 61 Simulación de movimiento de avance (Modelo 2)

	Avance	Unidades
Distancia total	9.23	m
Distancia Vertical	0.2	m
Angulo rotado	-737	grados
Velocidad total	0.4615	m/s
Velocidad vertical	0.01	m/s
Velocidad angular	-0.64	rad/s
Radio	0.7375	m

Tabla 36 Resultados simulación de movimiento de avance (Modelo 2)

En este modelo cuando mandamos la orden de avance el OpenROV realiza un giro más rápido hacia babor que en el modelo anterior. Esto hace que la circunferencia descrita sea de menor diámetro. Simultáneamente el vehículo se sumerge.

7.3.1.2. Retroceso

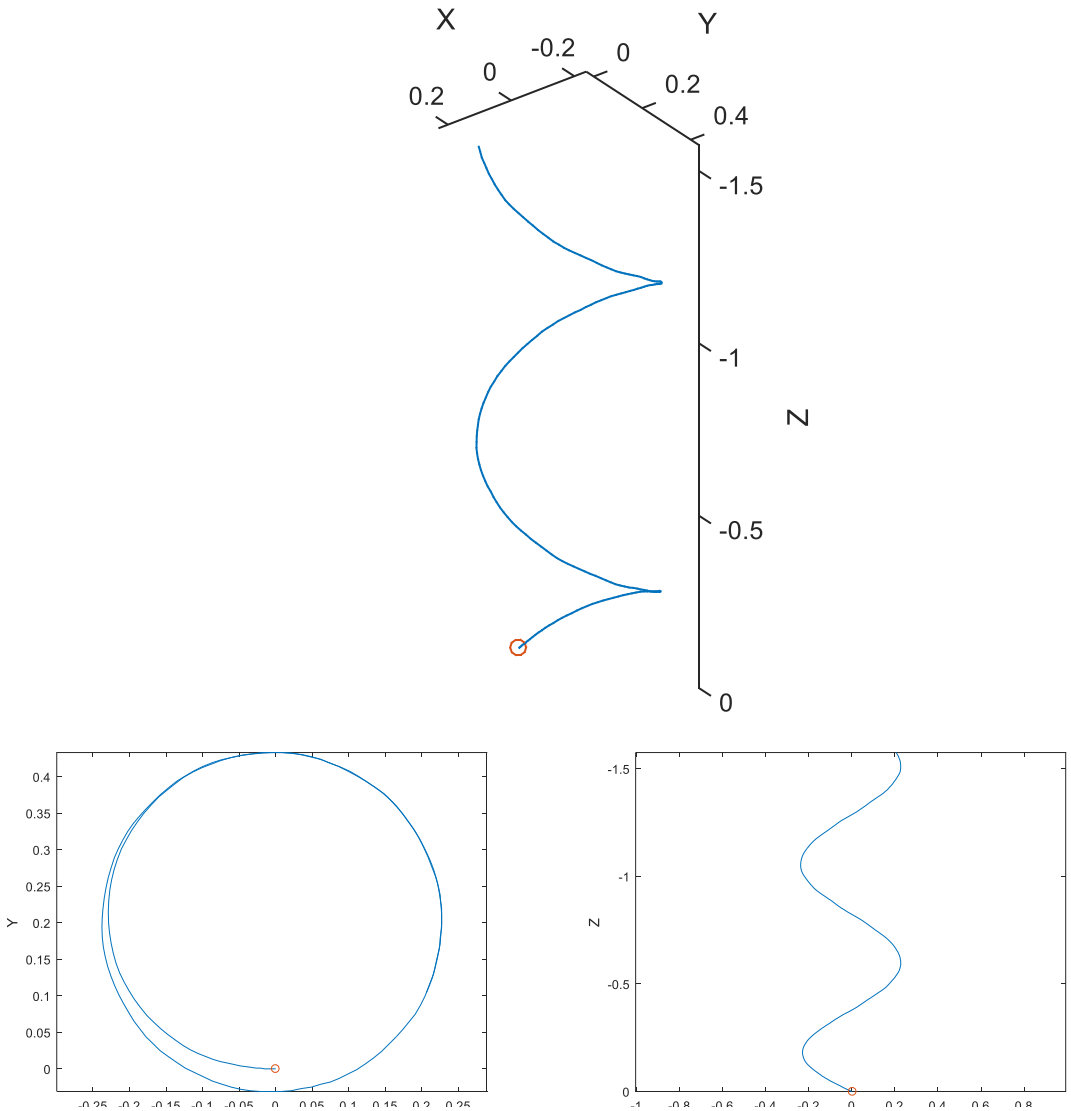


Ilustración 62 Simulación de movimiento de retroceso (Modelo 2)

	Retroceso	Unidades
Distancia total	3	m
Distancia Vertical	-1.575	m
Angulo rotado	620	grados
Velocidad total	0.15	m/s
Velocidad vertical	-0.07875	m/s
Velocidad angular	0.54	rad/s
Radio	0.23	m

Tabla 37 Resultados simulación de movimiento de retroceso (Modelo 2)

Cuando mandamos la orden de retroceso la simulación del vehículo realiza una trayectoria circular, moviéndose en la dirección trasera mientras asciende.

7.3.1.3. Giro hacia estribor

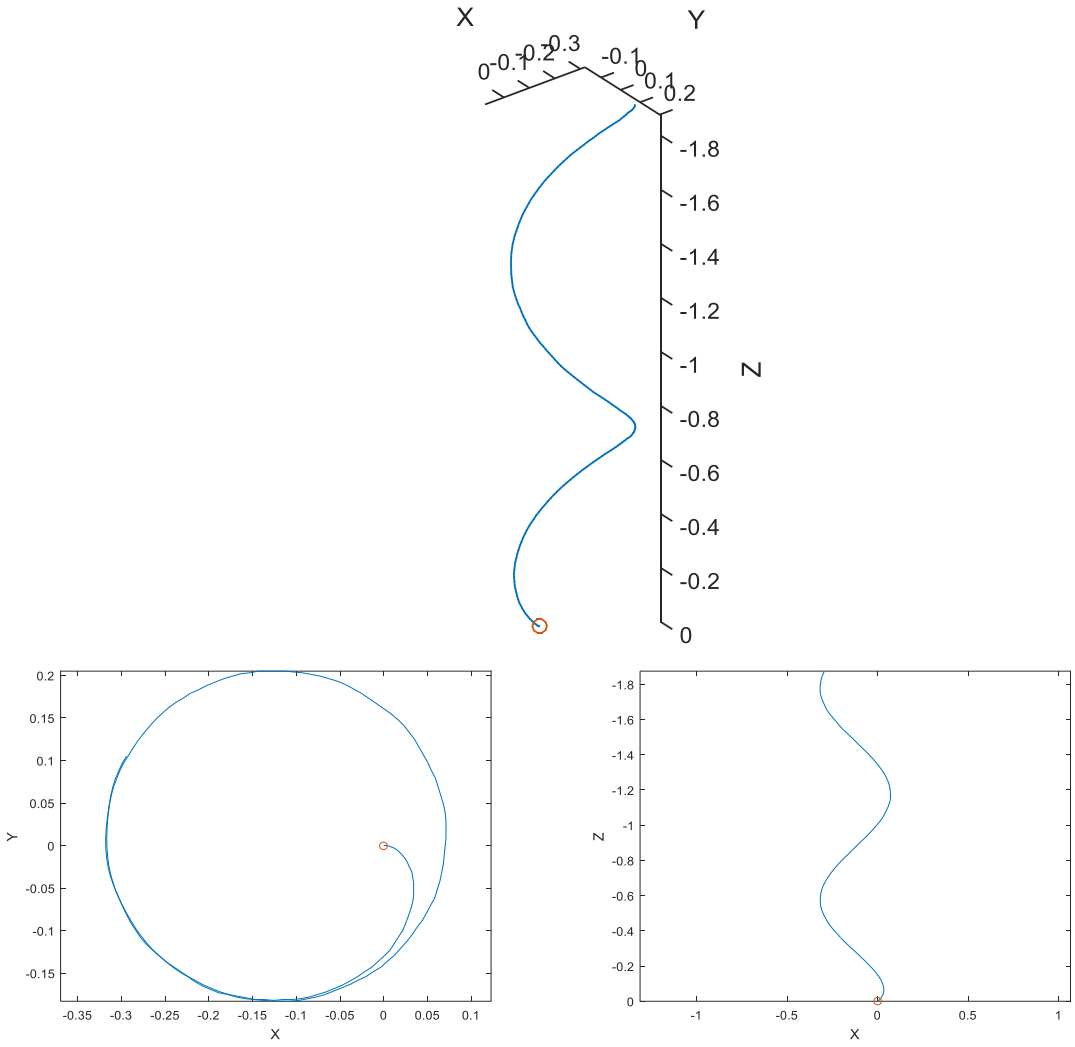


Ilustración 63 Simulación de movimiento de giro hacia estribor (Modelo 2)

	Giro estribor	Unidades
Distancia total	2.6833	m
Distancia Vertical	-1.875	m
Angulo rotado	602	grados
Velocidad total	0.134165	m/s
Velocidad vertical	-0.09375	m/s
Velocidad angular	0.525	rad/s
Radio	0.2	m

Tabla 38 Resultados simulación de giro a estribor (Modelo 2)

La orden de giro a estribor en este modelo hace que el vehículo gires sobre sí mismo, de manera más lenta que en el modelo anterior, lo cual es lógico ya que este movimiento viene generado principalmente por el propulsor izquierdo, el cual tiene una potencia menor en esta simulación. A la vez el vehículo asciende.

7.3.1.4. Giro hacia babor

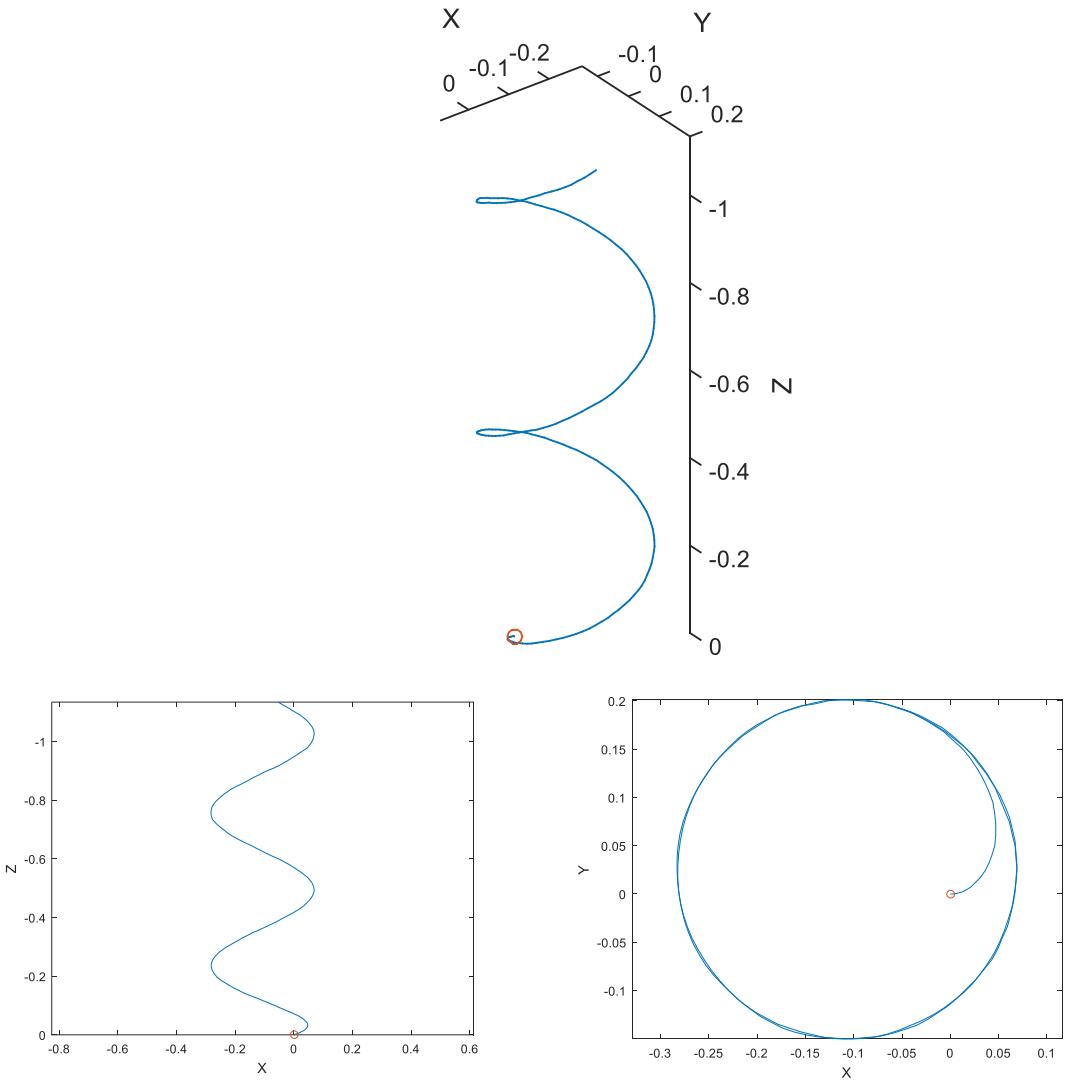


Ilustración 64 Simulación de movimiento de giro hacia babor (Modelo 2)

	Giro babor	Unidades
Distancia total	2.722	m
Distancia Vertical	-1.134	m
Angulo rotado	-833	grados
Velocidad total	0.1361	m/s
Velocidad vertical	-0.0567	m/s
Velocidad angular	-0.72	rad/s
Radio	0.23	m

Tabla 39 Resultados simulación de giro a babor (Modelo 2)

La orden de giro hacia babor la realiza de manera muy similar a la observada en el modelo anterior.

Los movimientos de descenso y ascenso serán iguales a los expuestos en los puntos 7.2.1.5 y 7.2.1.6 ya que en estos los motores traseros no intervienen.

7.3.1.5. Resumen movimientos básicos

A continuación, podemos ver los resultados de los movimientos básicos en el modelo en los cuales los dos propulsores traseros no son iguales resumidos en una tabla:

	Avance	Retroceso	Giro estribor	Giro babor	Unidades
Distancia total	9.23	3	2.6833	2.722	m
Distancia Vertical	0.2	-1.575	-1.875	-1.134	m
Angulo rotado	-737	620	602	-833	grados
Velocidad total	0.4615	0.15	0.134165	0.1361	m/s
Velocidad vertical	0.01	-0.07875	-0.09375	-0.0567	m/s
Velocidad angular	-0.64	0.54	0.525	-0.72	rad/s
Radio	0.7375	0.23	0.2	0.23	m

Tabla 40 Resumen resultados (Modelo 2)

7.3.2. Ensayos PID

Con la configuración expuesta anteriormente se han emulado los dos experimentos realizados en la realidad con los controladores PID activados. En ellos se ha intentado que todas las condiciones se parezcan lo máximo posible a las reales en la medida que nos permite el estado actual del simulador, estos cambios están descritos en el punto 4.2.8.

Para obtener una simulación más precisa se podría simular el funcionamiento real de los sensores (tiempos de retardo de la IMU, imprecisión de los sensores…)

7.3.2.1. Ensayo PID 1

Ensayo 1	Kp	Kd	Ki
PID guiñada	1	0	0
PID profundidad	Desactivado		

Tabla 41 Parámetros PID ensayo 1

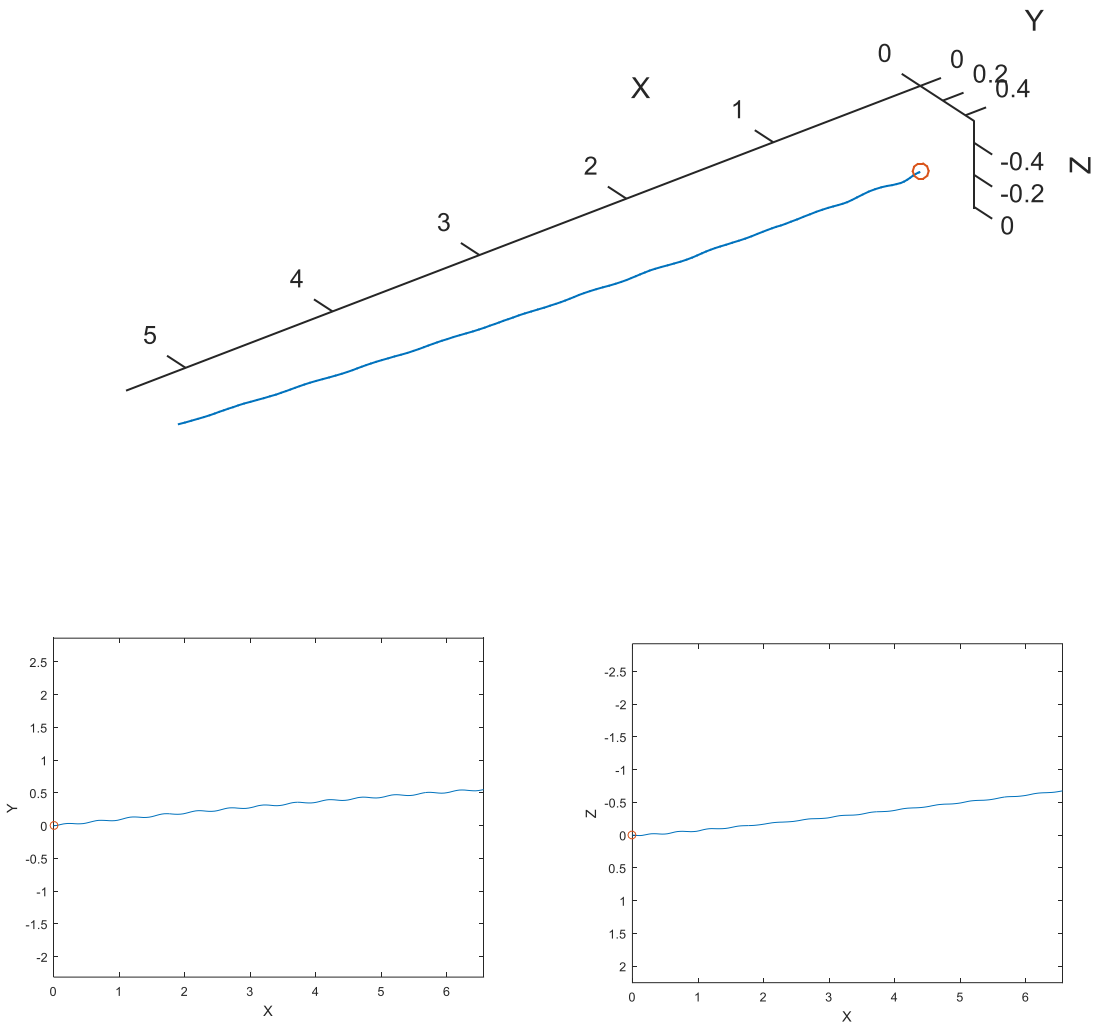


Ilustración 65 Simulación de ensayo PID 1

Ensayo PID 1		Unidades
Tiempo	14	s
Distancia recorrida	7	m
Profundidad	1	m
Velocidad horizontal	0.5	m/s
Velocidad vertical	0.07142857	m/s
Ángulo de desviación de trayectoria	6.5	grados

Tabla 42 Resumen resultados ensayo PID 1

En este ensayo donde la guiñada se intenta mantener en 0 mediante el uso de un PID, vemos como el vehículo traza una trayectoria recta, desviándose de su objetivo 6.5º. Simultáneamente asciende ligeramente.

7.3.2.2. Ensayo 2

Potencia de los motores: 100%

Ensayo 2	Kp	Kd	Ki
PID guiñada	1	0	0
PID profundidad	1	0	0

Tabla 43 Parámetros PID ensayo 2

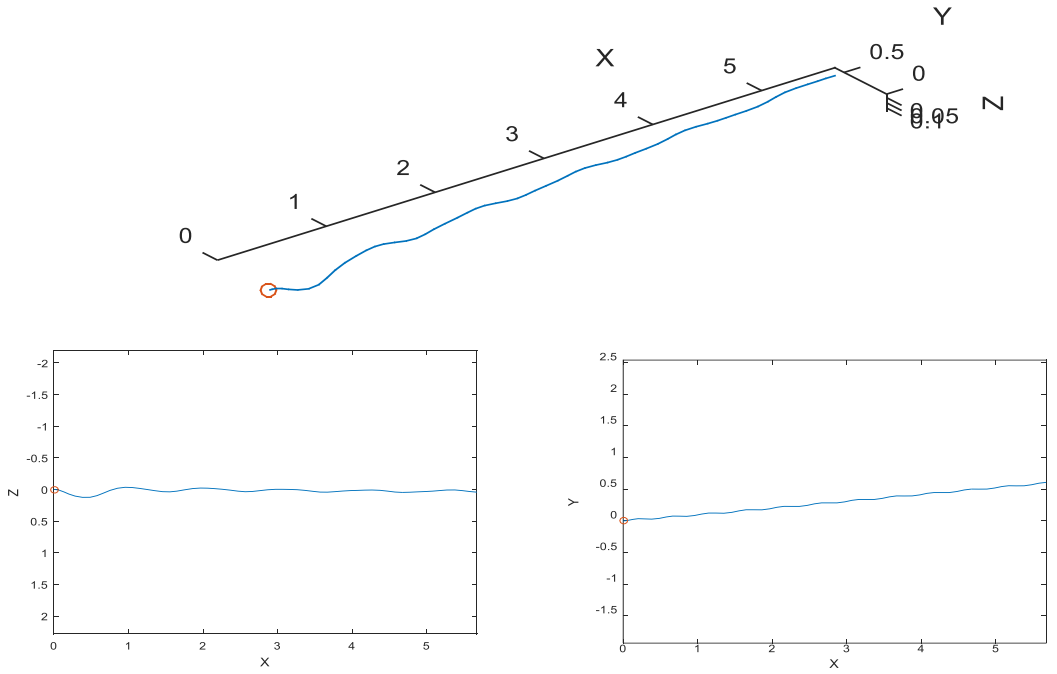


Ilustración 66 Simulación de ensayo PID 2

Ensayo PID 2		Unidades
Tiempo	6	s
Distancia recorrida	6.1	m
Profundidad	0	m
Velocidad horizontal	1.02	m/s
Velocidad vertical	0	m/s
Ángulo de desviación de trayectoria	4.76	grados

Tabla 44 Resumen resultados ensayo PID 2

En este ensayo los PIDs tratan de mantener tanto la guiñada como la profundidad inicial. Vemos que cuando aplicamos la orden de avance el vehículo se desplaza en línea recta con un error en la guiñada de 4.76 grados. En cuanto a la profundidad observamos que es capaz de mantenerla tras unos primeros segundos de oscilaciones.

Capítulo 8. Resultados

8.1. Movimientos básicos

En este capítulo compararemos los resultados obtenidos en las diferentes simulaciones y los observados en los ensayos reales.

Usaremos la siguiente escala de colores para visualizar los errores relativos entre los resultados reales y simulados:

272%	200%	150%	100%	50%	0%

8.1.1. Avance

AVANCE						
	Realidad	Modelo	Error	Modelo con motores diferentes	Error modelo con motores diferentes	Unidades
Distancia recorrida	9.42	10.05	6.69%	9.23	2.02%	m
Profundidad	0.75	0.05	93.33%	0.2	73.33%	m
Angulo recorrido	900	230	74.44%	737	18.11%	grados
Velocidad lineal	0.471	0.5025	6.69%	0.4615	2.02%	m/s
Velocidad vertical	0.0375	0.0025	93.33%	0.01	73.33%	m/s
Velocidad angular	0.78539816	0.200712864	74.44%	0.64315383	18.11%	rad/s
Radio de giro	0.6	1.75	191.67%	0.7375	22.92%	m

Tabla 45 Comparación entre modelos experimentos en avance

Vemos que la respuesta en la simulación con el modelo normal como en el ensayo ha sido cualitativamente similar, es decir, cuando se le manda la orden de avance, en los dos casos gira hacia la izquierda trazando una circunferencia mientras se sumerge. Las diferencias principales están en el radio de la circunferencia y en la velocidad a la que se sumerge, mientras que con la velocidad y la distancia lineal sí encontramos bastante similitud.

Si comparamos el modelo con motores diferentes y el ensayo real vemos que los resultados son también coherentes y notablemente más parecidos a los observados en la realidad. Vemos que realiza una circunferencia algo más grande que la observada, que su velocidad de giro es ligeramente más lenta que la real y que la velocidad a la que se sumerge también es menor, pero obtenemos mejores resultados que en el modelo considerando los dos motores traseros iguales.

8.1.2. Retroceso

RETROCESO						
	Realidad	Modelo	Error	Modelo con motores diferentes	Error modelo con motores diferentes	Unidades
Distancia recorrida	0.6	0.9	50.00%	0.9	50.00%	m
Profundidad	-0.24	-0.453	88.75%	-0.4725	96.88%	m
Angulo recorrido	0	138	No comparable	186	No comparable	grados
Velocidad lineal	0.1	0.15	50.00%	0.15	50.00%	m/s
Velocidad vertical	-0.04	-0.0755	88.75%	-0.07875	96.88%	m/s
Velocidad angular	0	0.401425728	No comparable	0.541052068	No comparable	rad/s

Tabla 46 Comparación entre modelos experimentos en retroceso

Podemos ver que en este caso la simulación y los resultados reales no son completamente coherentes, ya que en los ensayos reales se observa como el vehículo traza una trayectoria recta en el sentido de retroceso, mientras que en las simulaciones por el contrario traza una trayectoria circular en ambos modelos. Sin embargo, la velocidad total y como la vertical tienen errores relativamente grandes, pero tienen el mismo orden de magnitud.

Cabe destacar que en los ensayos reales el vehículo estaba situado en la superficie del agua, por lo tanto, no se pudo medir el desplazamiento vertical, ya que iría en sentido de ascenso. Por ello se ha estimado con la velocidad de flotación del OpenROV.

8.1.3. Giro estribor

GIRO ESTRIBOR						
	Realidad	Modelo	Error	Modelo con motores diferentes	Error modelo con motores diferentes	Unidades
Profundidad	-0.68	-1.5861	133.25%	-1.59375	134.38%	m
Angulo recorrido	650	586.5	9.77%	511.7	21.28%	grados
Velocidad vertical	-0.04	-0.0933	133.25%	-0.09375	134.38%	m/s
Velocidad angular	0.667	0.602	9.77%	0.5253	21.28%	rad/s
Radio de giro	≈0	0.17	Coherente	0.17	Coherente	m

Tabla 47 Comparación entre modelos experimentos en giro a estribor

Para el movimiento de giro hacia estribor podemos observar que los movimientos predichos por el simulador y los obtenidos en la realidad son coherentes. En ambos casos el vehículo rota sobre sí mismo en el sentido mandado, a la vez que realiza un movimiento de ascenso. El modelo con los dos motores iguales predice algo mejor la velocidad angular de la guiñada, pero no es una diferencia muy notable.

Consideramos que los resultados del radio de giro de la simulación y los ensayos son iguales ya que a la hora de realizar las medidas experimentales del radio de giro con exactitud no es posible con los medios que disponemos, por ello se aproxima a 0. Por lo tanto, si en la simulación este radio es próximo a 0 consideramos que es un resultado aceptable. Esta consideración la realizaremos en los sucesivos análisis de los resultados.

Al igual que en el caso del retroceso, en los ensayos reales el vehículo estaba situado en la superficie del agua, por lo tanto, no se pudo medir el desplazamiento vertical, ya que iría en sentido de ascenso. Por ello se ha estimado con la velocidad de flotación del OpenROV.

8.1.4. Giro babor

GIRO BABOR						
	Realidad	Modelo	Error	Modelo con motores diferentes	Error modelo con motores diferentes	Unidades
Profundidad	-0.44	-0.6369	44.75%	-0.6237	41.75%	m
Angulo recorrido	-800	-465.85	41.77%	-458.15	42.73%	grados
Velocidad vertical	-0.04	-0.0579	44.75%	-0.0567	41.75%	m/s
Velocidad angular	-1.269	-0.739	41.77%	-0.727	42.73%	rad/s
Radio de giro	≈0	0.0825	Coherente	0.1265	Coherente	m

Tabla 48 Comparación entre modelos experimentos en giro a babor

Al igual que en el movimiento de giro hacia estribor, en el movimiento de giro hacia babor podemos observar que los movimientos predichos por el simulador y los obtenidos en la realidad son coherentes. En ambos casos el vehículo rota sobre sí mismo (el radio de giro en este caso también es prácticamente despreciable) en el sentido de babor, a la vez que realiza un movimiento de ascenso. Ambos modelos tienen resultados casi iguales.

En este caso, al igual que en los anteriores, en los ensayos reales el vehículo estaba situado en la superficie del agua, por lo tanto, no se pudo medir el desplazamiento vertical, ya que iría en sentido de ascenso. Por ello se ha estimado con la velocidad de flotación del OpenROV.

8.1.5. Descenso

DESCENSO			
	Realidad	Modelo	Error
Profundidad	1.2	1.0125	15.63%
Angulo recorrido	-100	-157.5	57.50%
Velocidad vertical	0.133	0.1125	15.63%
Velocidad angular	-0.193	-0.305	57.50%
Radio de giro	≈0	0.09	Coherente

Tabla 49 Comparación entre modelos experimentos en descenso

En el movimiento de descenso se podemos ver que los movimientos observados en la realidad concuerdan con los simulados, el vehículo desciende a la vez que realiza una rotación sobre sí mismo en la dirección de babor. El simulador nos da también resultados numéricamente bastante parecidos a los observados en los experimentos, con una velocidad de descenso próxima a la experimentes, aunque el ángulo de giro nos dé un error algo mayor, pero aceptable.

8.1.6. Ascenso

ASCENSO			
	Realidad	Modelo	Error
Profundidad	-1.2	-2.25	87.50%
Angulo recorrido	75	279	272.00%
Velocidad vertical	-0.133	-0.25	87.50%
Velocidad angular	0.145	0.541	272.00%
Radio de giro	≈0	0.1485	Coherente

Tabla 50 Comparación entre modelos experimentos en ascenso

La simulación del movimiento de ascenso también nos arroja resultados coherentes con los reales, la trayectoria realizada es de ascenso mientras el vehículo gira hacia estribor. El

pequeño radio de giro nos confirma que el vehículo gira sobre sí mismo tanto en la simulación como en la realidad. Aunque numéricamente los resultados presentan un gran error relativo, son del mismo orden de magnitud, por lo que consideramos que los resultados son aceptables.

8.2. Comparación de resultados con los PIDs

8.2.1. Ensayo 1

Ensayo PID1				
	Realidad	Modelo	Error	Unidades
Distancia recorrida	7	6.4	8.57%	m
Profundidad	1	-0.7	170.00%	m
Velocidad horizontal	0.5	0.457142857	8.57%	m/s
Velocidad vertical	0.071428571	-0.05	170.00%	m/s
Ángulo de desviación de trayectoria	20	6.5	67.50%	grados

Tabla 51 Comparación entre modelos experimentos en el ensayo PID 1

En este ensayo podemos comprobar que el movimiento en el plano horizontal sí es coherente, es decir cuando intentamos mantener la guiñada y mandamos la orden de avance, el vehículo avanza en línea recta con una pequeña desviación hacia la izquierda, aunque en la simulación esta desviación es algo menor. Sin embargo, si encontramos un problema en el movimiento vertical del vehículo, en la simulación el vehículo asciende, mientras que en el ensayo real este desciende. Una causa posible es que esto se deba a que el control del vehículo no está siendo simulado de manera idéntica a la realidad ya que el tiempo de retardo de la IMU es notable en el ensayo real, mientras que en la simulación este efecto no se tiene en cuenta, por lo que las dinámicas pueden variar.

8.2.2. Ensayo 2

Ensayo PID2				
	Realidad	Modelo	Error	Unidades
Distancia recorrida	6	6.1	1.67%	m
Profundidad	0	0	Iguales	m
Velocidad horizontal	1	1.02	2.00%	m/s
Velocidad vertical	0	0	Iguales	m/s
Ángulo de desviación de trayectoria	5	4.76	4.80%	grados

Tabla 52 Comparación entre modelos experimentos en avance en el ensayo PID 2

En este ensayo los resultados son notablemente mejores. Pese a esto, los resultados de la tabla pueden resultar engañosos, ya que en el ensayo se obtenía que la profundidad deseada se alcanzaba casi al final de la trayectoria, mientras que, en la simulación, aunque también tarda un tiempo en alcanzarlo, este es menor.

8.3. Análisis de los resultados

A la vista de os resultados obtenidos podemos decir que el modelo considerado se asemeja de manera suficiente al vehículo a simular. En casi todos los casos, excepto en el movimiento de retroceso, el cual no es un movimiento demasiado relevante en el control del vehículo, y en la velocidad vertical en primer ensayo con PID, pero este probablemente venga causado porque en la simulación no se están teniendo en cuenta las dinámicas de los sensores reales, los cuales tienen tiempos de retardo notables y una imprecisión asociada, por lo que afectan a los resultados obtenidos. Otra posible causa para esta discrepancia es algún error en los coeficientes de resistencia hidrodinámica que fueron obtenidos en el proyecto de Ignacio de la Cotera [4]. Como se ha comentado anteriormente se han tenido que realizar algunas modificaciones a errores evidentes que tenían estos, por lo que puede que sigan existiendo otros no detectables a simple vista. Estos pueden ser debidos a que hubo una reasignación en los ejes de referencia a la hora de dar los resultados finales, por lo que puede haberse introducido alguna errata al realizar este cambio, como por ejemplo algún cambio de signo. Por falta de tiempo no se ha podido investigar en profundidad esta posibilidad.

Dejando de lado estas excepciones, todos los resultados son tanto cualitativamente como cuantitativamente aceptables, en especial en el modelo que tiene en cuenta las diferencias entre los motores traseros, lo cual es lógico, ya que es el más cercano a lo observado experimentalmente.

Capítulo 9. Conclusiones y desarrollos futuros

9.1. Conclusiones

Durante la realización de este TFG se han desarrollado una variedad de trabajos relacionados con campos muy distintos, desde el estudio del modelado y simulación de vehículos submarinos, hasta el control de estos mediante distintas técnicas, pasando por la programación en varios leguajes con arquitecturas y enfoques diferentes. La heterogeneidad de estas tareas ha ayudado a desarrollar mi capacidad de resolución de problemas con los medios disponibles, teniendo que recurrir a fuentes de información muy diversas como son libros, artículos científicos, foros, en especial de programación, compañeros de otras carreras etc.

Una vez expuesto el trabajo realizado, se puede valorar que las aportaciones más destacadas al campo de estudio son las siguientes:

- Una parte de las tareas realizadas permiten extender la capacidad de aplicación y la precisión del simulador. Se incluyen, entre otras, la posibilidad de realizar simulaciones en una variedad mucho mayor de tipos de vehículos, controladores, condiciones del entorno… Esto permite que el simulador pueda ser usado sin necesidad de conocer en profundidad el modelo matemático en el que se basa, ni el código mediante el cual está programado, por lo tanto, se puede convertir en una valiosa herramienta para avanzar en el desarrollo de vehículos o sistemas de control de estos.
- Se ha conseguido avanzar de forma significativa en la implementación de sistemas de control más precisos para UUVs. En este trabajo se han explorado diferentes métodos de control, estudiando sus características y particularidades, además de ser implementados en el simulador. Al poder programarlos en el simulador se permite que, para cada vehículo, se puedan probar los diferentes sistemas de control propuestos mediante simulaciones y así determinar fácilmente cual es más adecuado para cada vehículo concreto y cuáles son sus parámetros asociados óptimos. Esto permite implementar el controlador más adecuado sobre el vehículo real sin necesidad de realizar costosos ensayos experimentales.
- El software del vehículo OpenROV se ha modificado para implementar sobre él mejoras que lo dotan de mayor capacidad y sencillez en su gobierno. Además se ha dejado plasmado en esta memoria los pasos a seguir y el funcionamiento general del código

programado en la electrónica del vehículo. De esta forma se facilita en gran medida las posibles modificaciones que se quieran realizar en un futuro. Cabe recordar que el software de OpenROV es un sistema en codigo abierto que ha sido desarrollado por diferentes personas en todo el mundo, por lo que la estructuración de su código presenta la dificultad que conlleva este tipo de filosofía de programación: estructuras poco intuitivas, parches y excepciones continuos, código poco estable, multitud de funcionalidades sin terminar o abandonadas… Por ello, encontrar y entender las partes clave de la programación del OpenROV, es decir, las que nos permiten relacionar las entradas (datos proporcionados por los sensores, botones accionados por el piloto…) con las salidas (potencia suministrada a los motores) puede ser una tarea compleja y que requiere de conocimientos de programación avanzados. Sin embargo, con los avances conseguidos en este trabajo, no solo se ha conseguido mejorar notablemente el control y pilotaje del OpenROV, sino que se ha creado un precedente al dejar bien documentado en esta memoria los pasos seguidos para poder introducir modificaciones. Además, se ha especificado la forma de acceder a estas partes clave del código y de qué manera modificarlas. Esto permitirá a quien tenga que continuar el trabajo comenzado en este proyecto partir de una base que le facilitará en gran medida realizar avances.

- El control y pilotaje del OpenROV se ha simplificado mediante mejoras en su software. Estas modificaciones permiten manejar el vehículo de manera más sencilla, ya que mejora el sistema de control preexistente (controlador proporcional) por uno más preciso (PID), además de permitir un uso más flexible de esta característica, ya que se permite el cambio de consignas en tiempo de vuelo, lo que permite hacer movimientos y maniobras que antes no eran posibles, como girar de forma controlada o ascender o descender una distancia deseada, dejando así menos responsabilidad en la habilidad y experiencia del piloto.
- La comparación entre los resultados del simulador y los obtenidos experimentalmente permite valorar la exactitud tanto del modelo matemático como de los parámetros obtenidos para el vehículo OpenROV. Esto ha permitido evaluar de manera fundada los puntos fuertes y las debilidades del modelo considerado. Además, se han podido corregir errores en los coeficientes hidrodinámicos calculados para el OpenROV y obtener las características de sus propulsores. Se ha podido observar que detalles que pueden parecer poco determinantes, como el tiempo de retardo de medida de los sensores, parecen tener una influencia notable en la discrepancia entre la simulación y los resultados reales. En general se puede decir que los resultados confirman que el

modelo se ajusta razonablemente a lo observado experimentalmente, por lo nos permite ratificar que tanto el trabajo realizado anteriormente por otros compañeros, como el desarrollado en este proyecto ha sido productivo y tiene utilidad real.

9.2. Tareas realizadas

A modo de resumen podemos destacar como tareas más importantes realizadas en este trabajo las siguientes:

- Cambio del programa simulador para compilar con MinGW.
- Estructuración y separación del simulador en archivos independientes de forma que la arquitectura sea más modular y sencilla de ampliar y modificar.
- Implementación de sistema de introducción de parámetros de simulación y del vehículo desde Excel.
- Cambios en modelo para permitir la introducción de diferentes coeficientes hidrodinámicos para cada sentido de movimiento y de giro.
- Programación de sistema PID para un vehículo de 6 grados de libertad.
- Programación de una variación del PID más avanzada.
- Desarrollo de entorno grafico en Unity.
- Recreación del sistema de control de OpenROV en el simulador.
- Corrección de error en sentido de giro de los motores en OpenROV.
- Implementación de PID completo en OpenROV.
- Creación de un nuevo plugin en Cockpit donde añadir nuevas funcionalidades.
- Implementación de sistema para cambiar la referencia de los PIDs del OpenROV.
- Programación de sistema para cambio de parámetros de los PIDs.
- Permitir la calibración y puesta a cero de los sensores del OpenROV.
- Añadir a la información mostrada en el navegador Cockpit variables del PID.
- Ensayos de movimientos básicos del OpenROV real.
- Ensayos de con PID de profundidad y ángulo de guiñada en el OpenROV.
- Simulación del modelo de OpenROV imitando los experimentos reales.
- Comparación de los resultados obtenidos de la simulación con las medidas adquiridas en los ensayos con el vehículo real.

9.3. Desarrollos futuros

- Comparación resultados del PID básico con el PID avanzado.
- Implementación PID avanzado en OpenROV.

- Mejor ajuste de parámetros de los PIDs en el OpenROV.
- Realización de más ensayos, mejor planificados y obteniendo medidas con mayor precisión.
- Mejora del sistema de introducción de datos desde Excel (Mejorar mensajes de error, aumentar robustez).
- Obtener los parámetros del modelo de cada propulsor de manera más precisa.
- Experimentación con movimientos combinados (Avance-Descenso, Avance-Ascenso…).
- Implementación de un sistema de control de posición en el OpenROV, es decir, no solo controlar la guiñada y la profundidad gracias a los valores directos que nos dan los sensores, sino integrarlos para conseguir mantener la posición también en el plano horizontal.
- Realizar una funcionalidad en Excel que te permita ver más claramente los parámetros con los que se van a cargar a la simulación para no tener que navegar entre las distintas hojas.
- Modelizar las IMUS para modelizar los tiempos de retraso en envío de la señal, rango de error en la medida etc.
- Cambiar el objeto de los timones en el simulador para que se calculen en las fuerzas resultantes que ejerce cada timón, en lugar de que este cálculo se realice en el objeto UUV como en la actualidad.
- Relacionar la selección de entorno y vehículo en la interfaz gráfica Unity con los parámetros introducidos la simulación del modelo matemático.

Referencias bibliográficas

[1] I. De la cotera Lopez, «Construcción, modelizacion y simulacion de un pequeño vehiculo submarino comercial», *Construction, modeling and simulation of a small commercial submarine vehicle*, sep. 2018, Accedido: mar. 04, 2020. [En línea]. Disponible en: http://repositorio.upct.es/handle/10317/7597.

[2] A. Garrido Pellicer, «Estimación de los coeficientes hidrodinámicos de vehículos autónomos submarinos mediante CFD», may 2015, Accedido: mar. 04, 2020. [En línea]. Disponible en: http://repositorio.upct.es/handle/10317/4657.

[3] J. A. Ruiz Ruiz, «Cálculo de coeficientes hidrodinámicos de UUVs mediante CFDs», *Calculation of hydrodynamic coefficients for UUVs using CFDs*, nov. 2016, Accedido: mar. 04, 2020. [En línea]. Disponible en: http://repositorio.upct.es/handle/10317/6510.

[4] J. GARCÍA GARCÍA, «CÁLCULO DE LOS COEFICIENTES HIDRODINÁMICOS DE VARIOS VEHÍCULOS SUBMARINOS MEDIANTE ANSYS».

[5] J. J. García García, «Desarrollo de una herramienta informática para la simulación dinámica de vehículos submanrinos no tripulantes», nov. 2013, Accedido: mar. 04, 2020. [En línea]. Disponible en: http://repositorio.upct.es/handle/10317/3615.

[6] A. Resina Sánchez, «Diseño mecánico de detalle y fabricación de un ROV», *Detail mechanical design for manufacturing an ROV*, oct. 2018, Accedido: mar. 04, 2020. [En línea]. Disponible en: http://repositorio.upct.es/handle/10317/7585.

[7] «Ros», *Wikipedia, la enciclopedia libre*. nov. 30, 2019, Accedido: mar. 04, 2020. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Ros&oldid=121698051.

[8] «Cómo y porqué aprender robótica con ROS, Robot Operating System». https://descubrearduino.com/ros/ (accedido mar. 04, 2020).

[9] *uuvsimulator/uuv_simulator*. UUV Simulator - Unmanned Underwater Vehicle Simulator, 2020.

[10] T. I. Fossen, *Guidance and Control of Ocean Vehicles*. Wiley, 1994.

[11] D. R. Yoerger, J. G. Cooke, y J.-J. E. Slotine, «The influence of thruster dynamics on underwater vehicle behavior and their incorporation into control system design», *IEEE J. Ocean. Eng.*, vol. 15, n.º 3, pp. 167-178, jul. 1990, doi: 10.1109/48.107145.

[12] W. M. Bessa, M. S. Dutra, y E. Kreuzer, «THRUSTER DYNAMICS COMPENSATION FOR THE POSITIONING OF UNDERWATER ROBOTIC VEHICLES THROUGH A FUZZY SLIDING MODE BASED APPROACH», p. 8, 2005.

[13] «Gauss–Markov process», *Wikipedia*. nov. 10, 2019, Accedido: mar. 04, 2020. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Gauss%E2%80%93Markov_process&oldid=925496705.

[14] *uuvsimulator/eca_a9*. UUV Simulator - Unmanned Underwater Vehicle Simulator, 2019.

[15] *uuvsimulator/lauv_gazebo*. UUV Simulator - Unmanned Underwater Vehicle Simulator, 2019.

[16] «uwsim - ROS Wiki». http://wiki.ros.org/uwsim (accedido mar. 04, 2020).

[17] S. McMillan, D. E. Orin, y R. B. McGhee, «Efficient dynamic simulation of an underwater vehicle with a robotic manipulator», *IEEE Trans. Syst. Man Cybern.*, vol. 25, n.º 8, pp. 1194-1206, ago. 1995, doi: 10.1109/21.398681.

[18] Feijun Song, P. E. An, y A. Folleco, «Modeling and simulation of autonomous underwater vehicles: design and implementation», *IEEE J. Ocean. Eng.*, vol. 28, n.º 2, pp. 283-296, abr. 2003, doi: 10.1109/JOE.2003.811893.

[19] «¿Qué es Hardware-in-the-Loop? - National Instruments». https://www.ni.com/es-es/innovations/white-papers/17/what-is-hardware-in-the-loop-.html#section-2091996175 (accedido mar. 04, 2020).

[20] «VMAX - Forum Energy Technologies, Inc.» https://www.f-e-t.com/subsea/software-and-control-system-solutions/vmax/ (accedido mar. 04, 2020).

[21] «ROV App | unicomstudios.co.uk». http://unicomcommunications.co.uk/ROV/ (accedido mar. 04, 2020).

[22] «Simulation». http://unicomcommunications.co.uk/simulation.html (accedido mar. 04, 2020).

[23] «Sistema de control», *Wikipedia, la enciclopedia libre*. mar. 21, 2020, Accedido: mar. 24, 2020. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Sistema_de_control&oldid=124439055.

[24] «¿Qué es la visión artificial y para qué sirve?», *CONTAVAL*, feb. 18, 2016. https://www.contaval.es/que-es-la-vision-artificial-y-para-que-sirve/ (accedido mar. 04, 2020).

[25] R. Pérez-Alcocer, L. A. Torres-Méndez, E. Olguín-Díaz, y A. Maldonado-Ramírez, «Vision-Based Autonomous Underwater Vehicle Navigation in Poor Visibility Conditions Using a Model-Free Robust Control», *J. Sens.*, vol. 2016, pp. 1-16, ene. 2016, doi: 10.1155/2016/8594096.

[26] S. Matsumoto y Y. Ito, «Real-time vision-based tracking of submarine-cables for AUV/ROV», en *OCEANS '95 MTS/IEEE «Challenges of Our Changing Global Environment». Conference Proceedings*, oct. 1995, vol. 3, pp. 1997-2002 vol.3, doi: 10.1109/OCEANS.1995.528883.

[27] «Handbook of Marine Craft Hydrodynamics and Motion Control | Wiley», *Wiley.com*. https://www.wiley.com/en-es/Handbook+of+Marine+Craft+Hydrodynamics+and+Motion+Control-p-9781119991496 (accedido mar. 04, 2020).

[28] «Sliding mode control», *Wikipedia*. mar. 03, 2020, Accedido: mar. 04, 2020. [En línea]. Disponible en: https://en.wikipedia.org/w/index.php?title=Sliding_mode_control&oldid=943786875.

[29] M. Castilla Fernández, «Modelos no lineales y control En modo deslizamiento de Convertidores de estructura resonante».

[30] L. G. García-Valdovinos, T. Salgado-Jiménez, M. Bandala-Sánchez, L. Nava-Balanzar, R. Hernández-Alvarado, y J. A. Cruz-Ledesma, «Modelling, Design and Robust Control of a Remotely Operated Underwater Vehicle», *Int. J. Adv. Robot. Syst.*, vol. 11, n.º 1, p. 1, ene. 2014, doi: 10.5772/56810.

[31] T. Salgado-Jiménez, L. G. García-Valdovinos, y G. Delgado-Ramírez, «Control of ROVs using a Model-free 2nd-Order Sliding Mode Approach», *Sliding Mode Control*, abr. 2011, doi: 10.5772/15951.

[32] R. Hernández-Alvarado, L. G. García-Valdovinos, T. Salgado-Jiménez, A. Gómez-Espinosa, y F. Fonseca-Navarro, «Neural Network-Based Self-Tuning PID Control for

Underwater Vehicles», *Sensors*, vol. 16, n.º 9, p. 1429, sep. 2016, doi: 10.3390/s16091429.

[33] E. Fiorelli, N. E. Leonard, P. Bhatta, D. A. Paley, R. Bachmayer, y D. M. Fratantoni, «Multi-AUV Control and Adaptive Sampling in Monterey Bay», *IEEE J. Ocean. Eng.*, vol. 31, n.º 4, pp. 935-948, oct. 2006, doi: 10.1109/JOE.2006.880429.

[34] R. Cui, S. Sam Ge, B. Voon Ee How, y Y. Sang Choo, «Leader–follower formation control of underactuated autonomous underwater vehicles», *Ocean Eng.*, vol. 37, n.º 17, pp. 1491-1502, dic. 2010, doi: 10.1016/j.oceaneng.2010.07.006.

[35] A. De la Red, «Modelado, Simulación y Control de un Vehículo Submarino Manipulado de forma Remota (ROV)», UPCT, 2014.

[36] «Mecanica de Fluidos - MM Sanchez Nieto 2014 (2016!06!27 13-16-46 UTC) | Boquilla | Presión», *Scribd*. https://es.scribd.com/document/384406375/Mecanica-de-Fluidos-MM-Sanchez-Nieto-2014-2016-06-27-13-16-46-UTC (accedido mar. 04, 2020).

[37] J. Kim, «Thruster Modeling and Controller Design for Unmanned Underwater Vehicles (UUVs)», *Underw. Veh.*, ene. 2009, doi: 10.5772/6705.

[38] «Socket de Internet», *Wikipedia, la enciclopedia libre*. nov. 13, 2019, Accedido: mar. 04, 2020. [En línea]. Disponible en: https://es.wikipedia.org/w/index.php?title=Socket_de_Internet&oldid=121290406.

Anexo 1. Macro de Excel

Nótese que C:\RutaAlSimulador debe ser sustituido por la ruta al ejecutable del simulador en C++

```
Sub Intento3()
'
' Intento3 Macro
'
' Acceso directo: CTRL+n
'
Dim wbkExport As Workbook
Dim shtToExport As Worksheet

Sheets("P_FISICAS").Select

Set shtToExport = ThisWorkbook.Worksheets("P_FISICAS")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:=" C:\RutaAlSimulador\P_FISICAS", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Sheets("C_HIDRODINAMICOS2").Select

Set shtToExport = ThisWorkbook.Worksheets("C_HIDRODINAMICOS2")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:=" C:\RutaAlSimulador \C_HIDRODINAMICOS2", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Sheets("C_HIDRODINAMICOS3").Select
```

```
Set shtToExport = ThisWorkbook.Worksheets("C_HIDRODINAMICOS3")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:=" C:\RutaAlSimulador \C_HIDRODINAMICOS3", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Set shtToExport = ThisWorkbook.Worksheets("C_HIDRODINAMICOS")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:=" C:\RutaAlSimulador \C_HIDRODINAMICOS", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Set shtToExport = ThisWorkbook.Worksheets("PROPULSORES")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:="  C:\RutaAlSimulador\PROPULSORES", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Set shtToExport = ThisWorkbook.Worksheets("HELICES")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:="C:\RutaAlSimulador\HELICES", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Set shtToExport = ThisWorkbook.Worksheets("TIMONES")     'Sheet to export as CSV
```

```
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:="C:\RutaAlSimulador\TIMONES", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Set shtToExport = ThisWorkbook.Worksheets("CONTROLADORES_H")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:="C:\RutaAlSimulador\CONTROLADORES_H", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Set shtToExport = ThisWorkbook.Worksheets("CONTROLADORES_G")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:="C:\RutaAlSimulador\CONTROLADORES_G", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Set shtToExport = ThisWorkbook.Worksheets("COMPARADORES")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:="C:\RutaAlSimulador\COMPARADORES", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Set shtToExport = ThisWorkbook.Worksheets("IMU")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
```

```
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:="C:\RutaAlSimulador\IMU", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Set shtToExport = ThisWorkbook.Worksheets("ENTORNO")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:="C:\RutaAlSimulador\ENTORNO", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Set shtToExport = ThisWorkbook.Worksheets("DATOS_SIMULACION")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:="C:\RutaAlSimulador\DATOS_SIMULACION", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Set shtToExport = ThisWorkbook.Worksheets("CONDICIONES_INI")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:="C:\RutaAlSimulador\CONDICIONES_INI", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Set shtToExport = ThisWorkbook.Worksheets("TRAYECTORIAS")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
```

```
wbkExport.SaveAs Filename:="C:\RutaAlSimulador\TRAYECTORIAS", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Set shtToExport = ThisWorkbook.Worksheets("OBJETIVO")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:="C:\RutaAlSimulador\OBJETIVO", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Set shtToExport = ThisWorkbook.Worksheets("CONFIGURACION")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:="C:\RutaAlSimulador\CONFIGURACION", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

Set shtToExport = ThisWorkbook.Worksheets("CONTROLADORES_T")     'Sheet to export as CSV
Set wbkExport = Application.Workbooks.Add
shtToExport.Copy Before:=wbkExport.Worksheets(wbkExport.Worksheets.Count)
Application.DisplayAlerts = False                    'Possibly overwrite without asking
wbkExport.SaveAs Filename:="C:\RutaAlSimulador\CONTROLADORES_T", FileFormat:=xlCSV
Application.DisplayAlerts = True
wbkExport.Close SaveChanges:=False

End Sub
```

Anexo 2 parámetros de la simulación OpenROV

Tabla de configuración:

COMPONENTE:	OPEN_ROV_MODELO_2	OPEN_ROV_MODELO 1
NUMERO VEHICULOS	1	1
ENTORNO	OPEN_ROV	OPEN_ROV
SIMULACION	OPEN_ROV	OPEN_ROV
IDENTIFICADOR	1	1
VEHICULO	OPEN_ROV	OPEN_ROV
TIMONES	SIN_TIMONES	SIN_TIMONES
COMPARADOR	CMP1	CMP1
CONTROLADORES_T	P1	P1
CONTROLADORES_H	OPEN_ROV	OPEN_ROV
IMU	IMU2	IMU2
MOTOR	MOTOR1	MOTOR1
PROPULSORES	OPEN_ROV_MOTORES_DIFERENTES	OPEN_ROV
TRAYECTORIA	Petrolero	Petrolero
CONDICIONES_INICIALES	OPEN_ROV	OPEN_ROV
CONTROLADORES_G	PIDG1	PIDG1
OBJETIVO	Petrolero	Petrolero

Propiedades físicas:

I	0.0222
	0.0139
	0.01944
Ip	0
	0
	0
L	0.3
cdg	0
	0
	0.02
	0

cc	0
	0
m	2.523
v	0.00254
proa	0.15
popa	-0.15
PRPSauv	0.1500
	0
	0
Di	0.1910

Coeficientes hidrodinámicos lineales:

AVANCE	Xu	-0.1352
	Yu	-0.0065
	Zu	0.1528
	Ku	-0.0028
	Mu	-0.002
	Nu	0.0087
RETROCESO	Xu	0.0891
	Yu	0.0036
	Zu	-0.0973
	Ku	-0.0013
	Mu	0.005
	Nu	0.0025
BABOR	Xv	-0.1107
	Yv	0.2024
	Zv	0.3614
	Kv	-0.0092
	Mv	0.0189
	Nv	0.0043
ESTRIBOR	Xv	-0.1107
	Yv	0.2024
	Zv	0.3614
	Kv	-0.0092
	Mv	0.0189
	Nv	0.0043
DESCENSO	Xw	0.2737
	Yw	0.2832
	Zw	-5.8357
	Kw	-0.0154
	Mw	0.1071
	Nw	0.0602
ASCENSO	Xw	-0.4345
	Yw	0.1792
	Zw	0.5473
	Kw	-0.007

	Mw	-0.0311
	Nw	-0.0032
BALANCE POSITIVO	Xp	-0.0028
	Yp	-0.0008
	Zp	-0.0022
	Kp	0.0009
	Mp	0.0003
	Np	-0.0002
BALANCE NEGATIVO	Xp	-0.015
	Yp	-0.0161
	Zp	-0.0093
	Kp	-0.0069
	Mp	-0.0053
	Np	-0.0032
CABECEO POSITIVO	Xq	0.0143
	Yq	0.0358
	Zq	-0.0538
	Kq	0.0027
	Mq	0.0065
	Nq	0.0027
CABECEO NEGATIVO	Xq	0.0099
	Yq	-0.0189
	Zq	0.0311
	Kq	-0.0008854
	Mq	-0.0047
	Nq	0.0002384
GUIÑADA POSITIVA	Xr	0.0075
	Yr	0.0357
	Zr	0.0109
	Kr	-0.0002
	Mr	-0.0015
	Nr	0.0323
GUIÑADA NEGATIVA	Xr	0.0822
	Yr	0.0542
	Zr	0.065
	Kr	-0.0099
	Mr	-0.0044
	Nr	0.0081

Coeficientes hidrodinámicos no lineales:

		Valor usado	Valor original
AVANCE	Xuu	-7.95	
	Yuu	-0.0222	
	Zuu	2.8709	
	Kuu	0.0005	
	Muu	-0.01948	0.1948
	Nuu	-0.0086	

RETROCESO	Xuu	-11.1908	-11.1908
	Yuu	0.0777	
	Zuu	-2.0755	
	Kuu	-0.013	
	Muu	0.5118	
	Nuu	0.0543	
BABOR	Xvv	0.474	
	Yvv	-18.403	-18.403
	Zvv	-1.6941	
	Kvv	0.2569	
	Mvv	-0.0748	
	Nvv	-0.1893	
ESTRIBOR	Xvv	0.474	
	Yvv	-18.403	18.403
	Zvv	-1.6941	
	Kvv	0.2569	
	Mvv	-0.0748	
	Nvv	-0.1893	
DESCENSO	Xww	8.00065	
	Yww	0.0877	
	Zww	-17.5718	
	Kww	0.0309	
	Mww	-0.036	0.36
	Nww	-0.0598	
ASCENSO	Xww	2.2548	
	Yww	-0.483	
	Zww	-17.9462	-17.9462
	Kww	-0.0047	
	Mww	-0.05402	-0.5402
	Nww	0.0116	
BALANCE POSITIVO	Xpp	0.0369	
	Ypp	0.2011	
	Zpp	-0.0577	
	Kpp	-0.0297	0.0297
	Mpp	-0.0009	
	Npp	-0.0014	
BALANCE NEGATIVO	Xpp	0.0309	
	Ypp	-0.1614	
	Zpp	-0.0281	
	Kpp	-0.028	
	Mpp	0.0015	
	Npp	0.004	
CABECEO POSITIVO	Xqq	0.2417	
	Yqq	0.0262	

	Zqq	-0.2417	
	Kqq	0.0003	
	Mqq	-0.0346	0.0346
	Nqq	0.0002	
CABECEO NEGATIVO	Xqq	-0.1522	
	Yqq	0.0098	
	Zqq	0.2663	
	Kqq	0.0003863	
	Mqq	-0.0329	
	Nqq	-0.0001449	
GUIÑADA POSITIVA	Xrr	-0.0522	
	Yrr	-0.2277	
	Zrr	0.0095	
	Krr	0.0059	
	Mrr	-0.0009	
	Nrr	-0.0714	
GUIÑADA NEGATIVA	Xrr	-0.0589	
	Yrr	0.2398	
	Zrr	-0.012	
	Krr	-0.0025	
	Mrr	0.0001	
	Nrr	-0.0614	

Coeficientes cruzados (los que no aparecen son 0)

AVANCE_BABOR	Xuv	-9.8333
	Yuv	67.6564
	Zuv	4.4189
	Kuv	-0.8055
	Muv	0.3778
	Nuv	0.8026
AVANCE_ESTRIBOR	Xuv	-28.838
	Yuv	-68.8917
	Zuv	-0.4951
	Kuv	-0.7151
	Muv	-0.0935
	Nuv	0.825
AVANCE_DESCENSO	Xuw	-3.7418
	Yuw	0.1362
	Zuw	-15.4666
	Kuw	-0.0898
	Muw	0.695
	Nuw	0.0607
AVANCE_ASCENSO	Xuw	-3.7418

	Yuw	-0.1083
	Zuw	21.8596
	Kuw	-0.0663
	Muw	0.17211
	Nuw	0.1149
RETROCESO_BABOR	Xuv	7.1064
	Yuv	75.4193
	Zuv	-2.907
	Kuv	0.7458
	Muv	-0.2767
	Nuv	2.6712
RETROCESO_ESTRIBOR	Xuv	7.9381
	Yuv	-75.8644
	Zuv	-0.4817
	Kuv	0.8083
	Muv	0.1098
	Nuv	2.5879
RETROCESO_DESCENSO	Xuw	12.9319
	Yuw	0.576
	Zuw	-3.7601
	Kuw	0.0965
	Muw	1.2702
	Nuw	-0.0269
RETROCESO_ASCENSO	Xuw	-1.9417
	Yuw	-0.9163
	Zuw	-2.0775
	Kuw	-0.0492
	Muw	1.4674
	Nuw	-0.0295

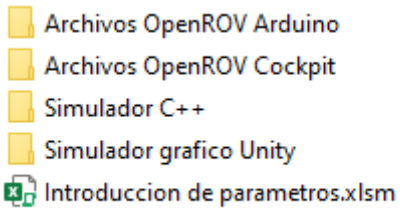
CARACTERISTICAS		VARIABLES	OPEN_ROV	OPEN_ROV_MOTORES_DIFERENTES
Dato general:		Numero	3	3
CARACTICAS HELICE		Tipo	OPEN_ROV_H_DER	OPEN_ROV_H_IZQ
		SG	1	1
POSICON DE LA HÉLICE	Posición de la hélice	rh	0	0
			0.1	0.1
			0	0
	Dirección de avance de la hélice	uh	1	1
			0	0
	Vector unitario		0	0
CARACTICAS HELICE		Tipo	OPEN_ROV_H_DER	OPEN_ROV_H_DER
		SG	-1	-1
POSICON DE LA HÉLICE	Posición de la hélice	rh	0	0
			-0.1	-0.1
			0	0
	Dirección de avance de la hélice	uh	1	1
			0	0
	Vector unitario		0	0
CARACTICAS HELICE		Tipo	OPEN_ROV_V	OPEN_ROV_V
		SG	1	1
POSICON DE LA HÉLICE	Posición de la hélice	rh	0	0
			0	0
			1	1
	Dirección de avance de la hélice	uh	0	0
			0	0
	Vector unitario		1	1

COEFICIENTES		VARIABLES	OPEN_ROV_H_IZQ	OPEN_ROV_H_DER	OPEN_ROV_V
PARAMETROS DE LA HÉLICE	Diametro de la hélice	Dh	0.057	0.057	0.058
	Masa de la hélice	mh	0.01	0.01	0.01
	Momento de Inercia	Jh	0.000001	0.000001	0.000001
	Número de RPM minimo	Nmin	-2000	-2000	-5700
	Número de RPM maximo	Nmax	5700	5700	5700
	Coeficientes de laa hélice	k	0	0	0
			0	0	0
			7.00E-04	1.10E-03	1.00E-03
	constantes positivas	a	1.00E-01	1.00E-01	1.00E-01
			1.00E-03	1.00E-03	1.00E-03
			1.00E-01	1.00E-01	1.00E-01
	constante experimental	Ka	-1.01E+00	-1.01E+00	-1.01E+00
	relacion PAR/FUERZA	nu	0.00E+00	0.00E+00	-7.50E-03

	g	9.81
	rho	1.00E+03
	Numero	0.00E+00
CORRIENTE	Identificador	1
	Vmax	0
	Rumbo	0
	Cabeceo	0
	AnchoM	0
	AnchoT	0
	X0	0
	Y0	0
	Z0	0

Anexo 3. Archivos adjuntos

Se adjuntarán los archivos más importantes con los que se ha trabajado en este proyecto para que el departamento pueda seguir avanzando en base al trabajo realizado. Se facilitará una carpeta con los siguientes documentos:



Archivos OpenROV Arduino: en esta carpeta se guarda toda la programación del Arduino modificada durante este trabajo. En el sistema de archivos de OpenROV se guarda en *OpenROV/opt/arduino/*

Archivos OpenROV Cockpit: en esta carpeta se guarda toda la programación del Arduino modificada durante este trabajo. En el sistema de archivos de OpenROV se guarda en *OpenROV/opt/cockpit/src*

Simulador C++: El simulador en C++ desarrollado. En esta carpeta es donde se deben exportar los archivos de configuración .csv que genera Excel. También en esta carpeta se genera el archivo de salida con las coordenadas de las simulaciones.

Simulador gráfico Unity: Aquí se guarda en simulador desarrollado en Unity. Para su funcionamiento tan solo debemos ejecutar “Submarino.exe”

Introducción de parámetros.xlsm: Tabla Excel con todos los parámetros y la macro para exportarlos al formato .csv. Es importante recordar que hay que modificar la macro para que exporte estos parámetros en la carpeta donde se guarde el simulador.

Anexo 4. Variables y métodos de los objetos del simulador

Para caracterizar cada una de las clases que usamos en el simulador plasmaremos el código de los archivos headers donde son definidas cada una de ellas.

Comunica

```
class Comunica
{
    private:
        struct sockaddr_in server, si_other;
        SOCKET s;
        int slen,recv_len;
        double *DatoSalida;
        double *DatoEntrada;
        int Ne;
        int Ns;
        int port;
        char const DIP[14];
        bool tipo;

   public:
        //void Crear(char const *,int, bool);           // Crear comunica
cion  (direccion IP del servidor, puerto, tipo:server=0/client=1)
        Comunica();
        void Crear(char const *,int, bool);             // Crear comunica
cion  (direccion IP del servidor, puerto, tipo:server=0/client=1)
        //void Comunicar(double *,double *,int, int);             // Envi
ar y recibir dato
        void Comunicar(double *,double *,int , int );           // Enviar
 y recibir dato
        char *INI(char const * );
        void Cerrar();
        void PasarParametrosAlHiloComunicacion();
};
```

Control Basic H

```
class CONTROL_BASIC_H
{
    private:
        int Nh;
        double *acciones;                 // Numero de helices a controla
r
    public:
```

```
        double * RecuperarAcciones();     // Valores de las acciones real
izadas sobre las helices
        void Crear(int);
  // Crear el controlador
};
```

Control manual

```
class Control_Manual
{
    private:
        struct sockaddr_in server, si_other;
        SOCKET s;
        int slen,recv_len;
        //char buf[BUFLEN];
        //char message[BUFLEN];
        double datoE[BUFLEN];
        double datoR[BUFLEN];
        int port;
        const char DIP[14]; //char const
        bool tipo;
   public:
        int Crear();   // Crear comunicacion  (direccion IP del servidor,
 puerto, tipo:server=0/client=1)
        void Comunicar(double *,double *);   // Enviar y recibir dato
        void Cerrar();
};
```

Corriente

```
class Corriente
{
    private:
        double Vmax;       // Velocidad maxima de la corriente
        double Rumbo;      // Rumbo de la corriente en rad
        double Cabeceo;    // angulo de cabeceo de la corriente en rad
        double AnchoM;     // Anchura de la corriente de velocidad maxima
        double AnchoT;     // Anchura total de la corriente
        double Punto0[3];  // Punto central de la corriente
        double VectorU[3]; // Vector director unitario de la corriente
    public:
        void Crear();
        // Crear la corriente
        void Cambiar(ChorrosDatos);  // Cambiar la corriente
        void VelocidadC(double [3],double [3]);
        // Devuelve la velocidad del punto que se le introduce
};
```

Datos

```
class Datos {
private:

    char nombre[50];
    int maximocol;
    string ** data;
    bool ArchivoAbierto;
    bool PalabraEncontrada=false;
    bool ColumnaVariablesEnc=false;
    bool TodoBien=false;
    int maximocol1;
public:
    Datos();
    bool Cargar_Nuevo(string,string,int*);
    void AsignarTama(string);
    bool cargar(string);
    void mostrar();
    string VerElemento(int, int);
};
```

Entorno

```
class Entorno
{
   private:
        double g;              // Aceleracion de la gravedad
        double rho;            // Densidad del agua
        int n_corrientes;      // NUmero de corrientes
        Corriente * CORRIENTE; // Corrientes del Entorno
   public:
        void Crear(EntornoDatos);                    // Crear comunicacion
  (direccion IP del servidor, puerto, tipo:server=0/client=1)
        void CambiaG(double);           // Cambia la gravedad del entorno
        void CambiaRHO(double);         // Cambia la densidad del entorno
        double RHO();       // Devuelve la densidad del agua
        double G();       // Devuelve el valor de la gravedad
        void Nueva_Corriente(EntornoDatos,int);
        void Quitar_Corriente(int);
        void Ventorno(double [3],double [3]);
};
```

Hélice

```
class Helice
{
    private:
        double n;
        double Dh;
        double Jh;
        double mh;
```

```
        double rh[3];
        double uh[3];
        double Nmin;
        double Nmax;
        double k[3];
        double a[3];
        double Ka;
        double nu;
        int SG;       // Sentido de giro de la helice para empuje positiv
o (1 o -1)
        double vDh[6];
   public:
        void Crear();
        void F(double*);      // Crear e inicializar una helice
        void Cargar(PropulsorDatos);   // Cargar e inicializar los parame
tros de una helice
        void Girar(double);     // Dar movimiento a la helice a una veloc
idad
        double RPM();   // Sacar la velocidad de la helice rpm
        double KT(double);      // Calcula la constante de la helice en f
uncion de J0
        void Fuerza(double,double [3],double*); // Calcula la fuerza de l
a helice
};
```

IMU

```
class IMU
{
   private:
        double xm_a[13];   // = [0;0;0;0;0;0;0;0;0;0;0;0;0;0];
        double MT_a[4][3]; // Matriz de giros medidos
        double MR_a[3][3]; // Matriz de rotacion
        double R_imu_a;    // = 0.001;     % Resolucion del acelerometro
        double R_imu_w;    // = 0.001;     % Resolucion del giroscopo
        double E_imu_a[3]; // = [0.1;0.1;0.1];  % Exactitud de la medida
de aceleracion
        double E_imu_w[3]; // = [0.1;0.1;0.1];  % Exactitud de la medida
de velocidad
        double P_imu_a[3]; // = [0.1;0.1;0.1];  % precision en la medida
de aceleraciones lineales
        double P_imu_w[3]; // = [0.1;0.1;0.1];  % precision en la medida
de velocidades angulares
        double Posicion_m[6]; // Posicion del vehiculo medida por la IMU
        double XYZ_m[3];      // XYZ del vehiculo medidos por la IMU
        double ANG_m[3];       // Angulos del vehiculo medidos por la IMU
   public:
        void Crear();
              // Crear IMU
```

```
        void Cambiar_Parametros(IMUDatos); // Cambiar parametros de error
        void Inicializar(double [13]);
               // Inicializar la IMU
        void Medir(double [13],double,double [13]);
               // Medida de la IMU
        void MR(double [3][3]);
               // Sacar el valor de MR segun datos de la IMU
        void MT(double [4][3]);
               // Sacar el valor de MT segun datos de la IMU
        void Posicion_Medida(double [6]);
        void Angulos_Medidos(double [3]);
        void XYZ_Medidos(double [3]);
};
```

LeerDeExcel

```
    class LeerDeExcel{
private:
    Datos * Configuracion=new Datos();
    Datos * Cargados = new Datos();
    VehiculoDatos * VehiculoD;
    EntornoDatos EntornoD;
    SimulacionDatos SimulacionD;
    int colConf;
    int colAux;
    string simulacion;
    bool encontrado;
    int numeroVehiculos;
    int NumeroTimones;
    int NumeroControladores=0;
    int NumeroControladores_T;
    int NumeroControladores_H;
    int NumeroPropulsores=0;
    int NumeroControladores_G;
    int ContadorControladores=-1;


public:
    LeerDeExcel(string);
    void CalcularNumeroControladores();
    void mostrarTodo();
    void DefinirMRB();
    void DefinirSG();
    void EncontrarNombre();
    void LeerTrayectoria(string,int,string,string);
    void LeerHelices();
    void LeerGeneral(string, int, string,string);
    VehiculoDatos RecuperarVehiculo(int);
```

```
    EntornoDatos RecuperarEntorno();
    SimulacionDatos RecuperarSimulacion();
    int ObtenerTamanoVector(Datos*, int);
    int RecuperarNumeroVehiculos();
    void ErrorDeAccesoMemoria(string);
};
```

PID_H

```
class PID_H : public PID{

private:

        double **FuerzaS1;    // Matriz de ponderacion de cada helice en
cada grado de libertad
        double *H;            // Vector de ordenes para las helices
        int N;                //Numero de helices a controlar
    public:

        void Crear(VehiculoDatos);
        void CalcularDistribucion(VehiculoDatos); //Calcula la matriz Fue
rzaS1
        void Controlar(double [6], double [13],double [3][3],double [4][3
],double*); //Funcion general de control
        void ResetearErrorAngulo();
        void Calcular_Propulsion(double [6]); //Reparte las acciones de c
ontrol a las helices

};
```

PID_Heredado

```
class PID_Heredado : public PID{

private:

        double **FuerzaS1;    // Matriz de ponderacion de cada helice en
cada grado de libertad
        double *H;            // Vector de ordenes para las helices
        int N;
    public:

        void Crear(VehiculoDatos);
        void CalcularDistribucion(VehiculoDatos);
        void Controlar(double [6], double [13],double [3][3],double [4][3
],double*);
        void ResetearErrorAngulo();
        void Calcular_Propulsion();
```

```
};
```

PID OpenROV

```
class PID_OpenRov : public PID{

private:

        double *H;            // Vector de ordenes para las helices
        int N;

    public:

        void Crear(VehiculoDatos);
        void CalcularDistribucion(VehiculoDatos); //Calcula la matriz Fue
rzaS1
        void Controlar(double [6], double [13],double [3][3],double [4][3
],double*); //Funcion general de control
        void ResetearErrorAngulo();
        void Calcular_Propulsion(double [6]); //Reparte las acciones de c
ontrol a las helices

};
```

PID_T

```
class PID_T : public PID{
private:
    int N;
    double *T;

public:
    void Crear(VehiculoDatos);
};
```

PID

```
class PID
{
protected:

        double acciones[6];    //Acciones a la salida del controlador
        double error_a[6];    // Error actual cometido en el seguimiento
de la ruta en (x,y,z,shi,ti,phi)
        double acciones_a[6]; // Valores anteriores de salida del control
ador
```

```
        double error_i[6];    // Integral del error
        double dT_control;    // Tiempo de control
        double Kp[6];        // Parametros proporcionales para posicion y
 angulo del PD (KpX,KpY,KpZ,KpSHI,KpTI,KpPHI)
        double Kd[6];        // Parametros derivativos para posicion y an
gulo del PD (KdX,KdY,KdZ,KdSHI,KdTI,KdPHI)
        double Ki[6];        // Parametros integrales para posicion y ang
ulo del PD (KdX,KdY,KdZ,KdSHI,KdTI,KdPHI)
        double D_f[6];       // Factor Derivativo
        double D[6];         //Valor derivativo de la salida
        double I[6];        //Valor integral de la salida
        double Td[6];
        bool primerScan=true;

public:

        void Calcular_Acciones(double [6], double *); // Calcular accione
s de movimiento a partir del error
        void Calcular_Acciones_Avanzado(double [6], double *);
        void CambiarParametros(double [6],double [6],double [6]);
                    // Cambiar los parametros del controlador PD
        void ObtenerParametros(double [6],double [6],double [6]);
                    // Obtener los parametros del controlador PD
        void Error_Medido(double [6], double [13] , double [6],double [3]
[3],double *);         // Calculo del error a partir de los datos de la I
MU
        void Fijar_Tiempo_Control(double);
        double Sacar_Tiempo_Control();
        void ResetearErrorAngulo();   //Pone el error integral del angulo
 a 0
};
```

Ruta

```
class Ruta
{
    private:
        double **Puntos_Control;       // Puntos de control de la ruta
        int n_puntos;
    public:
        void Crear(TrayectoriaDatos);
        void Cargar(TrayectoriaDatos);
             // Crear ruta
        void Mover(double[3],double [4]); // Mover ruta poniendo nueva po
sicion del punto final de la ruta:(X,Y,Z) y con la matriz de rotacion (MR
) de la ruta
        void SacaPunto(double [6]);                                    //
Devuelve la ruta
```

```
        //      void SacaRuta(double *[6]);
     // Devuelve la ruta
};
```

Simulación

```
class Simulacion
{
    private:
        char Tipo_Simulacion;
        LeerDeExcel Leer;
        int NumeroVehiculos;
        double dt_mostrar;
        double dt_control;
        bool tiempo_real=false;

    public:
        long int ni;
        Simulacion(string);
        double Td;              // Duracion maxima de la simulacion
        double T;           // Tiempo acumulado de ejecucion
        double dt;              // Incremento de tiempo
        UUV *uuvs;              // Vehiculos que intervienen en la simulac
ion
        Entorno entorno;        // Entorno de la simulacion
        Ruta ruta;              // Ruta a seguir por el UUV
        Comunica com_sim;       // Elemento de comunicacion para carga la
 simulacion
        void Crear();           // Crear simulacion
        void Cargar(char);          // Cargar elementos de la simulación
        void Simular(char);     // Realizar la simulación
        LeerDeExcel RecuperarLeer();
        void Salir();           // Salir de la simulación
        void CargarPosicionInicial(double*,VehiculoDatos);
        //void HiloComunicacion(void * );   //
};
```

Timón

```
class Timon
{
private:
    string nombre;
    double an_min=0;        //angulo minimo
    double an_max=0;        //angulo max
    double tr=0;            //tiempo de retraso (desde que recibe una pos
icion hasta que intenta empezar a ir
```

```
    double Wmax=0;          //Velocidad angular maxima
    double amax=0;          //Aceleracio
    double Cuud[6];         //
    double angulo;          //angulo actual
    double W;               //Velocidad actual
    double movi;            //0=Quieto 1= moviendose positivo   -
1=Moviendose negativo
    double t_esp;           //Tiempo que lleva esperado
    double pos_marcada;     //Posicion a la que esta intentado ir en cada
 momento (No tiene por que coincidir con la que le estas mandadno)
    double pos_siguiente;


public:
    void Crear();
    void Cargar(TimonesDatos);
    void Inicializar();
    void Orden_Mover_Timon(double,double); //Da la orden de mover el timo
n
    void Mover_Timon(double,double);       //Aqui se hace la simulacion c
inematica del timon
    double Cuud_(int);
    double Angulo();
    void cambiar_angulo_timon(double);


};
```

UUV

```
class UUV
{
    private:
        FisicaDatos PFV_UUV;              // Propiedades físicas del UUV
        CHidroDatos CHD_UUV;              // Coeficientes Hidrodinámicos
del UUV
        Helice *Hs;               // Hélices del UUV
        CONTROL_BASIC_H CBasic;   // Controlador básico de hélices
        VehiculoDatos  VehiculoD;
        Comunica *Coms_UUV;       // Comunicaciones del UUV
        Timon * TimonO;
        PID_OpenRov CH;
        PID_T CT;
        IMU *IMUS_UUV;            // IMUS del UUV
        //Timon T_UUV[];          // Timones del UUV
        //PID_H CH_UUV;              // Controladores de h�lices del UUV
        //Controlador_T CT_UUV[]; // Controladores de timones del UUV
        double n_Helices;         // Número de hélices del vehículo
        double Pot[6]={0,0,0,0,0,0};
```

```
        double n_Timones;          // Número de timones del vehículo
        double n_Imus;             // Número de IMUs del vehículo
        double n_comunicaciones;   // Número de comunicaciones del vehículo
        double n_Controlador_H;    // Número de controladores de hélice de
l vehículo
        double n_Controlador_T;    // Número de controladores de timon del
 vehículo
        double velocidadR[6];      // Velocidad relativa SR movil u,v,w,r,
p,q
        //double posicionR[3];
        //double orientacionR[3];
        double velocidadA[6];    // Velocidad Absoluta SR absoluto
        double posicionA[3];     // Posiciín absoluta del vehículo
        double orientacionA[3]; // Orientación absoluta del UUV
        double velocidadF[6];    // Velocidad Respecto del fluido
        //double posicionF[3];     // Posición del vehículo respecto al flu
ido
        //double orientacionF[3]; // Orientación del vehículo respecto al
fluido
        double x[13];            // Vector de Estado del vehículo u,v,w,p,q
,r,X,Y,Z,Shi,Ti,Fhi
        double xdot[13];         // Derivada del vector de Estado del vehíc
ulo u,v,w,p,q,r,X,Y,Z,Shi,Ti,Fhi
        double flotabilidad[6]; // Vector de flotabilidad
        double Minv[6][6]; // Inversa de la matriz inversa
        double CRB[6][6];  // Matriz de Coriolis del SR
        double CA[6][6];   // Matriz de Coriolis de la masa añadida
        double MT[4][3];   // Matriz de transformación de giros
        double MR[3][3];   // Matriz de rotación
        double M11[3][3], M12[3][3],M21[3][3],M22[3][3];   // Submatrices
 de la matriz de coriolis del SR
        double A11[3][3], A12[3][3],A21[3][3],A22[3][3];   // Submatrices
 de la matriz de coriolis de la masa añadida


    public:
        void Crear();    // Crear UUV
        void CargarPropiedadesFisicas(); // Cargar las propiedades físicas
 del UUV
        void CargarCoeficientesHidrodinamicos();  // Cargar los coeficien
tes hidrodinámicos del UUV
        void CargarHelices();      // Cargar las hélicas del UUV
        void CargarTimones();
        void CargarPosicionInicial();
        void CargarIMUs();         // Cargar las IMUs del UUV
        void CargarComunicacion(int);      // Cargar las comunicacciones
del UUV
        void CargarPDH();      // Cargar controlador del UUV
```

```
        void CargarConfiguracionCompleta(char const ,VehiculoDatos);
     // Cargar todos los elementos del UUV
        void Mover(Entorno &, double);              // Enviar y recibir da
to
        void Propulsar(int , double );              // nh = número de héli
ce ; nc = valor de velocidad de giro
        void CalcularCHidro();
        double U();
        double V();
        double W();
        double X();
        double Y();
        double Z();
        double SHI();
        double TI();
        double PHI();
        double shi();
        double ti();
        double phi();
        double RPM(int);
        void Posicion(double [6]);
        void Llevar_a_Posicion(double [6]);
        void ControlarUUV(char, double [6] ,double [7], double, double [6
],bool [2]);
        void Controlar6H(double [6], double [6]); // Controla las hélices
 del ROV de 6 gdl
        void ControlarHelicesP(double[6],double[6],double[6]);
        void Medir(double ,double [13]);            // Tomar medida de la
 IMUs
        //void Error_Medido(double [6],double ,double [6]);  // Calcula e
l error según la medida tomada por la IMU
        void Error_Real(double [6],double [6]);          // Calcula el er
ror real del UUV respecto al punto de la ruta
        int JOYSTIC();
        void Fijar_Tiempo_Control(double);
        void Calcular_Propulsion_OpenROV(double*, double, double, double)
;
};
```