Proyecto Fin de Grado

Ingeniería en Electrónica, Robótica y Mecatrónica

Modelado, simulación y control de un vehículo submarino ligero

Autor: Alejandro Casado Pérez

Tutor: Carlos Bordons Alba

Dpto. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2021

Proyecto Fin de grado

Ingeniería en Electrónica, Robótica y Mecatrónica

Modelado, simulación y control de un vehículo submarino ligero

Autor:

Alejandro Casado Pérez

Tutor:

Carlos Bordons Alba

Profesor Catedrático

Departamento de Ingeniería de Sistemas y Automática

Escuela Técnica Superior de Ingeniería

Universidad de Sevilla

Sevilla, 2021

Trabajo Final de Grado: Modelado, simulación y control de un vehículo submarino ligero

Autor: Alejandro Casado Pérez

Tutor: Carlos Bordons Alba

El tribunal nombrado para juzgar el proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente;

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2021

El Secretario del Tribunal

A mi familia por su apoyo y amor incondicional

A mis maestros por guiarme hasta aquí

A mis amigos por nunca dudar de mi

Agradecimientos

"La tecnología no es nada. Lo importante es que tengas fe en la gente, que sean básicamente buenas e inteligentes, y si les das herramientas, harán cosas maravillosas con ellas"

- Steve Jobs, cofundador y presidente ejecutivo de Apple Inc. -

Cuando di por finalizado este Trabajo Final de Grado, comprendí que este nunca hubiera sido posible sin la ayuda de las muchas personas a las que les guardo un afecto especial. A todas ellas quiero expresar mi reconocimiento.

En primer lugar, a Don Carlos Bordons Alba, profesor del Departamento de Ingeniería de Sistemas y Automática de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla, que me planteó aquel tema inicial que se ha convertido en este TFG. Él guio los primeros pasos en este trabajo; recomendándome que acotara cada vez más el mismo; orientando la utilización de la bibliografía consultada, del modelo y simulación, del control y la presentación de sus resultados; exigiéndome que siguiera un guion de contenidos, que fue redefinido en varias ocasiones, así como que fuera disciplinado en su desarrollo. Se que todo ello era necesario y me ha servido mucho para llegar a las correcciones de lo que he ido escribiendo hasta darle el formato que ahora se presenta. En la redacción final de este trabajo, el no dejaba de decirme que con este trabajo se inicia un camino con muchas mejoras para el futuro. A él le agradezco, además, su estímulo y disponibilidad hasta llegar a la finalización de este trabajo.

A mis compañeros y compañeras de grado, por la invitación a las reflexiones del tema elegido para este trabajo.

No puedo dejar de reconocer que ha sido una tarea larga, ardua y llena, por tanto, de momentos más o menos felices, momentos también de desaliento, en los que siempre estuvieron cerca de mí mi familia y mis amigos y amigas.

He dejado para el final la mención de mi familia (mis padres y hermano) y, no porque sean los menos importantes, sino por todo lo contrario.

Querría hacer también una mención a este Grado en Robótica, Electrónica y Mecatrónica y agradecer al profesorado con el que me he formado el que me haya transmitido los conocimientos necesarios para desarrollar las competencias en los ámbitos deseados y darme la oportunidad de sumergirme en un mundo en constante desarrollo que me permite saciar mis ansias de conocimiento, que me ilusiona para el futuro profesional.

Alejandro Casado Pérez

Sevilla, 2021

Resumen

El presente Trabajo Final de Grado realiza un estudio de las leyes que gobiernan el movimiento de vehículos submarinos indicando las magnitudes físicas necesarias para la descripción de un modelo, analizando los grados de actuación disponibles, así como los métodos de percepción, sensores y estrategias de control más utilizadas.

Se realizará un estudio del submarino ligero Sibiu Nano Plus de la empresa Nido Robotics, describiendo sus componentes y estimando sus características físicas.

Por último, mediante ROS, se diseñará un sistema de control basado en PID y se evaluará su desempeño utilizando el simulador UUVSim.

Abstract

This project analyses the general characteristics of marine vehicles, making a description of the laws that govern movement of marine vehicles looking into the physical magnitudes needed for a complete description of a model and analyzing the available actuation degrees as well as the most used sensors, perception methods and control systems.

It will make a study of the light ROV Sibiu Nano Plus from Nido Robotics, describing all its components and making an estimation of its physical characteristics.

Lastly, via ROS, there will be an evaluation of the performance of a designed control system using the UUV Simulator.

Índice

Índice de Tablas

Índice de Figuras

Índice de Ilustraciones

Índice de Abreviaturas y Símbolos

AUV	Autonomous Underwater Vehicle
CFD	Computational Fluid Dinamics
ECEF	Earth Centered Earth Fixed
ECI	Earth Centered Inertial
HOV	Human Occupied Vehicle
IMU	Inertial Measurement Unit
LUV	Light Autonomous Underwater Vehicle
MEMS	Sistema MicroElectroMecánico
NED	North East Down
OB	Origen de coordenadas del cuerpo
OE	Origen de coordenadas de la Tierra
PID	Proportional-Integral-Derivative
PLC	Power Line Communication
ROS	Robot Operating System
ROUV	Remotely Operated Underwater Vehicle
ROV	Remotely Operated Vehicle
SOG	Speed Over Ground
TFG	Trabajo Final de Grado
URDF	United Robotics Description Format
UUV	Unmanned Underwater Vehicles
$p_{eb}^{e} = \begin{bmatrix} x^{e} & y^{e} & z^{e} \end{bmatrix}^{T}$	Posición ECEF
$p_{nb}^{n} = \begin{bmatrix} x^{n} & y^{n} & z^{n} \end{bmatrix}^{T}$	Posición NED

$v_{nb}^{b} = \begin{bmatrix} u & v & w \end{bmatrix}^{T}$	Velocidad linear
$v_{nb}^{b} = \begin{bmatrix} u & v & w \end{bmatrix}^{T}$	Fuerza
$\Theta_{en} = \begin{bmatrix} l & \mu \end{bmatrix}^{T}$	Longitud y Latitud
$\Theta_{nb} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^{T}$	Orientación (ángulos de Euler)
$\omega_{nb}^{b} = \begin{bmatrix} p & q & r \end{bmatrix}^{T}$	Velocidad angular
$m_{b}^{b} = \begin{bmatrix} K & M & N \end{bmatrix}^{T}$	Momento
$\eta = \begin{bmatrix} p_{nb}^{n} & \Theta_{nb} \end{bmatrix}^{T}$	Vector generalizado de posición
$v = \begin{bmatrix} v_{nb}^{b} & \omega_{nb}^{b} \end{bmatrix}^{T}$	Vector generalizado de velocidad
$\tau = \begin{bmatrix} f_{b}^{b} & m_{b}^{b} \end{bmatrix}^{T}$	Vector generalizado de fuera
$R\left(\Theta_{nb}\right)$	Matriz de rotación NED to Body
$T\left(\Theta_{nb}\right)$	Matriz de translación NED to Body
W	Fuerza ejercida por la gravedad
B	Fuerza de empuje de flotación
M_{RB}	Matriz de masas del sólido rígido
C_{RB}	Matriz de Coriolis
$r_{b}^{b} = [x_{b} \quad y_{b} \quad z_{b}]^{T}$	Vector posición del centro de gravedad
$r_{b}^{b} = [x_{b} \quad y_{b} \quad z_{b}]^{T}$	Vector posición del
ρ	densidad del agua
Δ	Volumen de agua desplazado
m	masa
g	gravedad
M_{A}	Matriz de masa agregada
C_{A}	Matriz hidrodinámica de fuerzas centrípetas de Coriolis

D	Matriz de amortiguamiento hidrodinámico
$v_c = \begin{bmatrix} u_c & v_c & w_c \end{bmatrix}^T$	Velocidad de la corriente de agua
$v_r = \begin{bmatrix} u_r & v_r & w_r \end{bmatrix}^T$	Velocidad relativa de la corriente
$u_r = \begin{bmatrix} x_u & y_u & z_u \end{bmatrix}^T$	Posición Umbilical
ϕ	Diámetro
$C_{A,umbilical}$	Coeficiente de arrastre del umbilical
T	Fuerza de empuje del propulsor
n	Velocidad angular del propulsor
K_T	Coeficiente de propulsión
J	Coeficiente de avance del propulsor
$\omega_{umbiical}$	Peso del umbilical por unidad de longitud.
P	Presión
C_r	Constante de empuje del propulsor
σ	Duty cycle

Glosario

Posicionamiento dinámico	Control de posición caracterizado por mantener la posición horizontal del barco
Masa agregada	La masa hidrodinámica agregada, se puede definir como una masa virtual añadida al sistema debida al volumen de fluido movido por un cuerpo acelerando o decelerando al moverse por él. Además, el objeto y el fluido no pueden ocupar el mismo espacio físico simultáneamente
Boundary Element Method	Es un método numérico computacional utilizado para resolver ecuaciones en derivadas parciales formuladas como ecuaciones integrales
Filtro de Kalman	Algoritmo desarrollado por Rudolf E. Kalman que permite identificar estados no medibles del sistema
Script	Término utilizado para designar un programa relativamente simple. No requieren de compilación y suelen ejecutarse a través de un interprete

1 INTRODUCCIÓN

Este proyecto tiene como base el vehículo submarino ligero Sibiu Nano Plus, ROV de la empresa Nido Robotics.

ROV hace referencia a cualquier vehículo operado remotamente, independientemente del medio en el que lo haga.

Las siglas exactas para los vehículos submarinos serían ROUV (Remote Operated Underwater Vehicle) o UUV (Unmanned Underwater Vehicles), aunque por tradición, o por el hecho de que los vehículos que operan en otros medios reciben otras siglas (UAV para los operados remotamente en el aire y RCV para los que operan en medio terrestre), se habla de ROV para referirse a los ROUV/UUV (Rodríguez, 2017).

El proyecto Aquacollet del Departamento de Ingeniería de Sistemas y Automática hace uso de robots acuáticos en ríos, utilizando GNSS en combinación con otros sistemas para monitorizar y mapear diversas variables ambientales de interés en campos como la agricultura. Se incluye también el desarrollo del software necesario para controlar todo el proceso y que analice los datos obtenidos por los módulos.

En este contexto, una de las partes necesarias, y motivación de este trabajo, es la del modelado del robot Sibiu Nano Plus en el software de simulación submarina UUVSim, para posteriores investigaciones y simulaciones, debido a que el fabricante del ROV no provee de un simulador propio. Para dicho fin se ha de crear tanto el modelo 3D del vehículo, como las herramientas software que permitan su control y adquisición de datos.

Por ello, este TFG realiza un análisis del estado del arte de la robótica aplicada del vehículo submarino ligero Sibiu Nano Plus. Se analizan tanto los antecedentes de los vehículos submarinos operados en remoto, categorías distinguibles, el estado actual de los mismos y las consideraciones de diseño que poseen.

A continuación, se analizan las herramientas informáticas utilizadas, con una breve descripción de las características más destacables de cada una.

El grueso del trabajo se centra en el vehículo a integrar, el trabajo desarrollado para dicho modelado (archivos creados, y explicación de los mismos), reservando un capítulo para el análisis de los resultados obtenidos y para las futuras mejoras que se pueden realizar para ampliar y perfeccionar el proyecto.

Varios apéndices se sitúan al final del documento para ampliar la información sin entorpecer demasiado la lectura del documento principal, y 4 índices, uno de figuras, otro de tablas, otro de ilustraciones y un último de abreviaturas y símbolos; son empleados para facilitar la búsqueda de los mismos. Las últimas páginas se reservan para detallar la bibliografía empleada para la realización del trabajo.

1.1 Objetivo

El objetivo general de este Trabajo Final de Grado es el diseño del control del vehículo submarino ligero Sibiu Nano Plus en ROS, mediante el diseño de un modelo del submarino y validando su correcto funcionamiento mediante simulación.

Los objetivos específicos que se persiguen son:

- Respecto al modelo: El modelo del submarino debe mantenerse dentro de los estándares y órdenes de magnitud de esta clase de submarinos
- Respecto a la simulación: Debe realizarse en ROS y ser compatible con las características del

modelo.

- Respecto al control: Debe realizarse en ROS y el submarino debe ser capaz de contrarrestar el efecto de la flotación, corregir su orientación y reducir el efecto de las corrientes marinas manteniendo su posición y siendo capaz de llegar a puntos objetivo.
- Respecto a la planificación: El submarino debe ser capaz de elaborar una serie de puntos (waypoints) por los que desplazarse hasta llegar al punto objetivo.

1.2 Metodología

Para el desarrollo del trabajo se parte de los conocimientos básicos de control/navegación y planificación/guiado.

En un primer lugar, se realiza una búsqueda de información sobre el entorno que rodea a los submarinos, así como las estrategias de control más utilizadas.

En segundo lugar, se hace una selección del simulador.

En tercer lugar, se analiza la creación de un modelo del submarino Sibiu Nano Plus.

Este modelo se divide en 3 partes: Investigación sobre las características del submarino Sibiu Nano Plus, creación del modelo 3D y cálculo de las magnitudes físicas del modelo.

Las características del submarino se analizarán en la página web oficial de la empresa y documentos oficiales. En caso de falta de alguna información, se analizarán los datos de submarinos parecidos, realizando una estimación para el Sibiu Nano Plus.

El modelo 3D se realiza mediante la herramienta FreeCAD, manteniendo en la medida de lo posible la fidelidad al submarino real. Para mayor detalle en el modelo se utilizará la herramienta MeshLab reduciendo polígonos y añadiendo detalles.

El cálculo de las características físicas se obtendrá de la ficha técnica del submarino. Obteniendo el resto de datos de la herramienta MeshLab. Para coeficientes hidrodinámicos se planteará la posibilidad de utilizar CFD o tratar con datos de submarinos con características parecidas.

En cuarto lugar, se realiza el diseño del control mediante linealización con PID, haciendo las oportunas transformaciones de los ejes del submarino.

Por último, se comprobará que se cumple con todos los objetivos anteriormente descritos mediante el simulador elegido, realizando un análisis del comportamiento obtenido.

2 ESTADO DEL ARTE

2.1 Antecedentes

Alrededor del 71% de la superficie del planeta está ocupada por agua, y de esta, casi el 96.5% la ocupan los océanos (Acosta, 2020). Sin embargo, más del 80% del océano permanece inexplorado (Juan José Villena & Davi Moura, 2020).

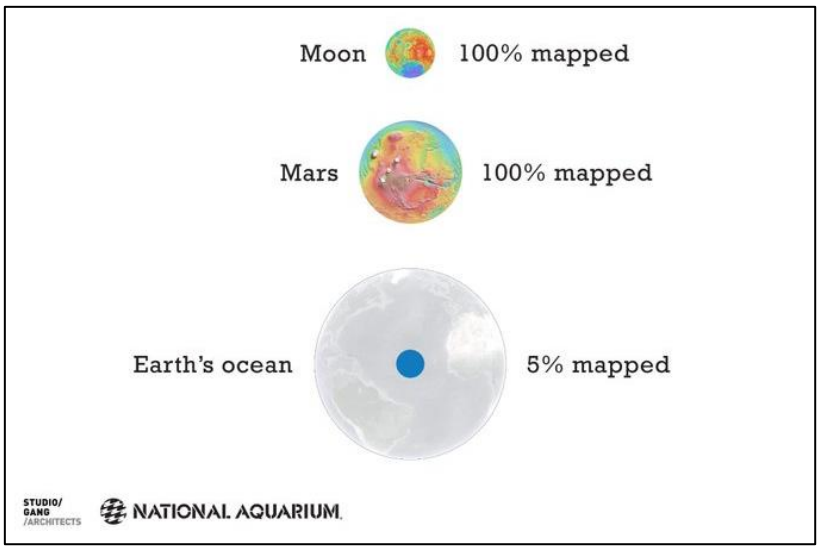
El oceanógrafo del Goddard Space Flight Center de la NASA explica: *"el océano, a grandes profundidades, se caracteriza por una visibilidad nula, temperaturas extremadamente frías y enormes cantidades de presión"*

La presión del aire que empuja hacia abajo nuestro cuerpo al nivel del mar es de aproximadamente 101 325 Pa, lo que se conoce como 1 atm. Sin embargo, conforme nos sumergimos, esta aumenta rápidamente.

"En una inmersión en el fondo de la Fosa de las Marianas, que tiene casi 11 kilómetros de profundidad, se está hablando de más de 1000 veces más presión que en la superficie"

Dicho esto, es de esperar que conozcamos mucho más del espacio exterior donde la presión desciende a 0, que de nuestro propio planeta.

Figura 1: mapa del océano frente a Marte y la Luna



Fuente: Juan José Villena & Davi Moura, 2020

La mayor parte de la investigación oceánica se realiza mediante imágenes satelitales, acelerando un trabajo que realizado mediante barcos tardaría años.

La investigación suboceánica, por otra parte, está realizada mediante flotadores y derivadores, dispositivos que dependen de las corrientes oceánicas para cargarse, y flotas de vehículos submarinos, controlados por humanos (HOV) o a distancia (UUV). Entre los que se incluyen los ROUV, UAV y URV (Juan José Villena & Davi Moura, 2020).

Los ROUV son una especie de drones subacuáticos que, debido a las dificultades del entorno, poseen un cable denominado "umbilical", a través del cual se realiza toda la comunicación entre el ROUV y

el piloto. Entre sus características se encuentran la capacidad de portar cargas, herramientas como pinzas, soldadores, etc, así como sumergirse hasta varios cientos de metros. En los últimos años su uso ha aumentado exponencialmente con el objetivo de reducir el riesgo laboral de submarinistas, reducir costes, así como facilitar tiempos de respuesta.

Entre otros se encuentran el Sibiu Pro de Nido Robotics utilizado por Endesa para labores de inspección y mantenimiento (*Robot Submarino - Endesa*, n.d.) y el Bluerov2 de Bluerobotics, que ha realizado misiones de exploración de hasta 500 metros de profundidad (Cross, 2020).

Ilustración 1: Sibiu Pro

Fuente: Nido Robotics (Nido Robotics, n.d.)

Ilustración 2: Bluerov2

Fuente: (BlueRobotics, n.d.)

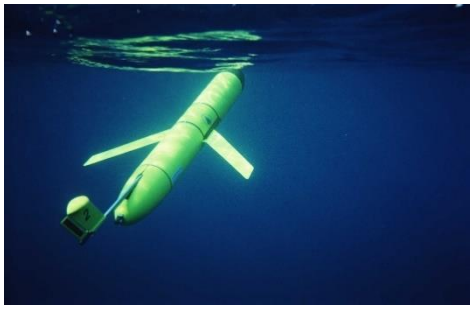
Los UAV, son un sistema único, poseyendo baterías y CPU, capaces de resolver los problemas de planificación y control sin intervención humana. Estos deben resolver un problema de 6 grados de libertad, corregir los errores debidos a la integración doble de los acelerómetros, y contrarrestar los efectos debidos a las dificultades del entorno (Jorge Luis Lemus Ramos, 2018). Su uso más común es en misiones de detección de desastres ecológicos y monitorización de tuberías, búsqueda de objetos o personas desaparecidas, arqueología marina o criaturas marítimas y su ecosistema (Lodovisi et al., 2018).

Entre otros se encuentran el Scarlet Knight, que cruzó el océano Atlántico (Schofield, 2009), en misiones de búsqueda de restos marinos (*Comienza La Operación de Búsqueda Con AUV Del Submarino ARA San Juan*, n.d.) y grupos como Dive & Discover, que ha realizado varias expediciones a distintos océanos con el objetivo de conocer las profundidades (*Dive & Discover*, 2021).

Ilustración 3: Scarlet Knight

Fuente: (Schofield, 2009)

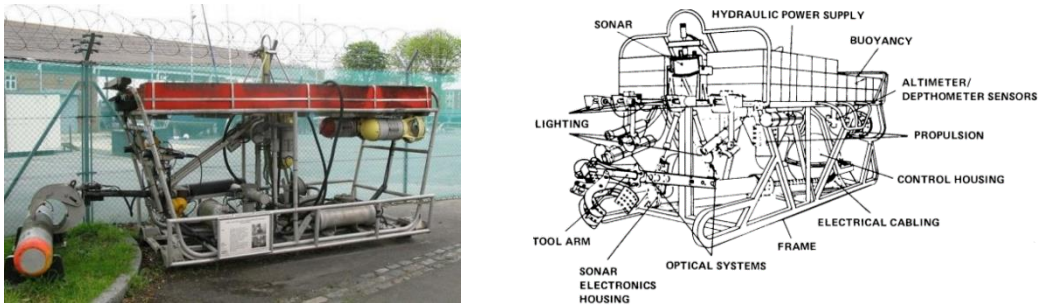
Ilustración 4: Dive&Discover



Fuente: (Dive & Discover - Expedition 17: Technology , n.d.)

Por último, los URV forman un grupo más amplio, formado por submarinos caracterizados por poseer algún tipo de manipulador. Entre ellos se encuentra el CUTLET robot pionero en este campo utilizado para recuperar torpedos, y la familia CURV, cuyo modelo CURV III fue capaz de sumergirse a más de 1800 metros de profundidad con 5.85 toneladas (Ahmad & Iqbal, 2014).

Ilustración 5: CUTTLET y CURV III



Fuente: (Ahmad & Iqbal, 2014)

2.1.1 Tipos de ROV

Los ROV presentan otra clasificación (Berg et al., 2012) dependiendo de su tamaño, potencia o trabajo:

- Micro: ROV muy pequeños de entre 2-10 kg principalmente usados para acceder a zonas donde humanos no pueden.
- Mini: Algo más grande que los ROV en torno a 15kg, utilizado en tareas de supervisión.
- General: Pueden realizar tareas de intervención, suelen tener menos de 5 Caballos (3750 W) de propulsión y sumergirse hasta 1000 m de profundidad.
- Ligeros: Poseen menos de 50 Caballos (37.5 kW) de propulsión, pueden sumergirse hasta 2000m y suelen llevar 1 o más manipuladores.
- Pesados: Menos de 220 Caballos (165 kW) de propulsión, poseen al menos 2 manipuladores y pueden sumergirse hasta 3500m.
- De zanja: Entre 200 y 500 Caballos (150-375 kW) de propulsión, equipados con un deslizador de tendidos de cable y capaces de operar a profundidades de hasta 6000 metros.

2.2 Entorno submarino

La Real Academia Española define submarino como *"Nave capaz de sumergirse y desplazarse bajo la superficie del agua"(RAE, n.d.)*.

Como cuerpo sumergido este está expuesto a dos presiones (Tristan Perez & Thor I. Fossen, 2011), cada una de ellas con una fuerza asociada.

- Fuerza de flotación debida a la presión hidrostática
- Fuerza hidrodinámica debida a la presión hidrodinámica (aproximadamente proporcional al cuadrado de la velocidad relativa respecto al agua).

Además, existen perturbaciones debidas a los efectos del oleaje, viento y corrientes oceánicas.

Las olas producen cambios de presión en el casco de la nave, induciendo fuerzas con componentes

oscilatorias que dependen linealmente de la altura de las olas y comparten su frecuencia. Además, estas fuerzas tienen una componente no lineal llamada deriva de las olas.

El viento y las corrientes oceánicas inducen fuerzas debidas a la variación de presiones, Estas tienen una componente caracterizada por un valor medio, y otra oscilatoria debida a los remolinos. En naves pequeñas, como es el caso de este proyecto, en el caso del viento solo se tendrá en cuenta la media debido a la alta frecuencia de las oscilatorias.

Las fuerzas debidas al oleaje, viento y corrientes oceánicas (Tristan Perez & Thor I. Fossen, 2011) se suelen dividir en:

- Baja frecuencia
- Alta frecuencia
- Debidas al oleaje.

Se va a considerar que las componentes de alta frecuencia son demasiado altas como para afectar al movimiento de la nave, pero son capaces de generar vibraciones en el casco (Faltinsen, 1993).

Las componentes de baja frecuencia formadas por la deriva de las olas, el viento y las corrientes causan que la nave se mueva, requiriendo de un control de posición para mantenerse ancladas.

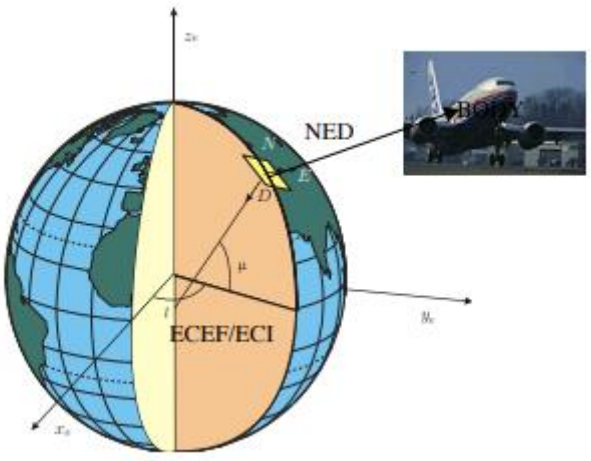
Las componentes oscilatorias pueden excitar modos de resonancia que producen movimientos horizontales, pero no suelen afectar al movimiento de la nave por lo que en ciertas ocasiones se decide despreciar su efecto ahorrando el consumo de energía y desgaste de los propulsores debidos a corregir el movimiento debido a cada ola.

2.3 Sistemas de referencia

Para describir el movimiento de un cuerpo marino se consideran 3 sistemas de referencia (Fossen, 2021):

- El sistema de referencia de la tierra (E) viene definido por ECEF/ECI con origen el centro de la tierra, eje z_e paralelo al eje de rotación de la tierra, eje x_e en el plano ecuatorial apuntando al meridiano 0 y el eje y_e completando el plano ecuatorial. Es utilizado para geolocalizar.
- Un sistema de referencia local (N) con valores fijados de latitud y longitud, y tangente a la superficie de la Tierra, cuyos ejes utilizan el sistema NED. Es utilizado para describir el movimiento del submarino.
- El cuerpo marino (B), tomando como eje x_b la dirección longitudinal hacia proa, y_b la transversal hacia estribor y z_b la normal. Es utilizado para describir la velocidad relativa del agua.

Ilustración 6: CUTTLET y CURV III

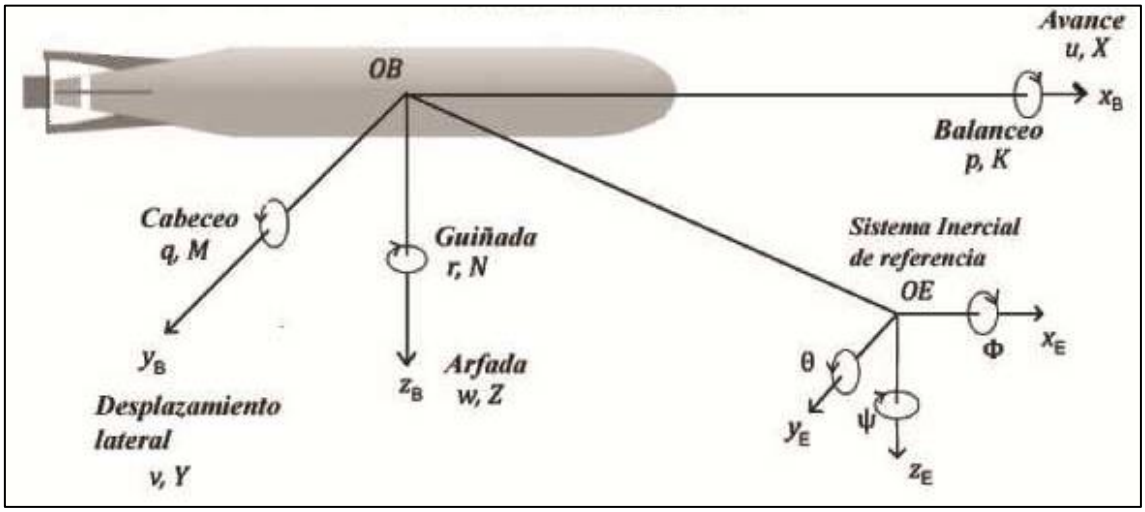


Fuente: (Fossen, 2021)

El cuerpo se considera un sólido rígido, y por tanto un sistema de referencia, su posición está dada por la posición relativa del origen respecto a la Tierra. Generalmente, ya que los submarinos se mueven a una velocidad relativamente baja, considerar la tierra como un sistema de referencia inercial es una buena aproximación, permitiendo adoptar un sistema de referencia local (NED), aplicable en un rango de 10x10km según la teoría de navegación terraplanista (Fossen, 2021).

Este sistema local de referencia tiene la forma:

Figura 2: Sistema de referencia local del submarino



Fuente: Imagen elaborada a partir de Yunier Valeriano (Yunier Valeriano-Medina et al., 2015)

Sistema sin ninguna restricción de movimiento y por tanto con 6 grados de libertad:

Tabla 1: Grados de libertad

Grado de libertad	Eje asociado	Movimiento	Fuerza/Momento	Velocidades	Posición
1	Eje X	Avance	X	u	z
2	Eje Y	Deriva	Y	v	y
3	Eje Z	Ascenso	Z	w	z
4	Rotación X	Balance/escora	K	p	φ
5	Rotación Y	Arfada	M	q	θ
6	Rotación Z	Viraje	N	r	ψ

Fuente: SNAME. 1950 (SNAME, 1950)

Una vez descritos los sistemas, podemos describir la posición, velocidades y fuerzas más relevantes.

$p_{eb}^{e} = \begin{bmatrix} x^e & y^e & z^e \end{bmatrix}^T$ Posición ECEF $\quad \Theta_{en} = \begin{bmatrix} l & \mu \end{bmatrix}^T$ Longitud y Latitud

$p_{nb}^{n} = \begin{bmatrix} x^n & y^n & z^n \end{bmatrix}^T$ Posición NED $\quad \Theta_{nb} = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$ Orientación (ángulos de Euler)

$v_{nb}^{b} = \begin{bmatrix} u & v & w \end{bmatrix}^T$ velocidad linear $\quad \omega_{nb}^{b} = \begin{bmatrix} p & q & r \end{bmatrix}^T$ velocidad angular

$f_b^b \begin{bmatrix} X & Y & Z \end{bmatrix}^T$ fuerza $\qquad$ $m_b^b = \begin{bmatrix} K & M & N \end{bmatrix}^T$ momento

Así como, los vectores generalizados de posición, velocidad y fuerzas:

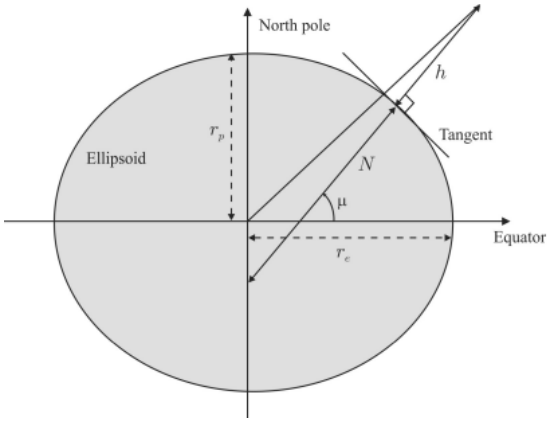
$$\eta = \begin{bmatrix} p_{nb}^n \\ \Theta_{nb} \end{bmatrix}, \quad \nu = \begin{bmatrix} v_{nb}^b \\ \omega_{nb}^b \end{bmatrix}, \quad \tau = \begin{bmatrix} f_b^b \\ m_b^b \end{bmatrix}$$

Fuente: Fossen (Fossen, 2021) nomenclatura (SNAME, 1950).

2.3.1 Transformación ECEF/ECI NED

La transformación entre las coordenadas terrestres y el sistema local, se puede realizar por geometría conociendo las características del elipsoide terrestre.

Figura 3: Composición elipsoide



Fuente: Fossen (Fossen, 2021)

$r_e = 6378137m$ Radio ecuatorial

$r_e = 6356752m$ Radio polar

$\omega_e = 7.292115 \cdot 10^{-5}$ rad/s Velocidad angular de la tierra

$e = \sqrt{1 - \left(\frac{r_p}{r_e}\right)^2} = 0.0818$ Excentricidad del elipsoide

$$N = \frac{r_e^2}{r_e^2 \cos^2 \mu + r_p^2 \sin^2 \mu} \qquad p = \sqrt{x^2 + y^2}$$

ECEF/ECI → NED

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} (N+h)\cos(\mu)\cos(l) \\ (N+h)\cos(\mu)\sin(l) \\ \left(\frac{r_p^2}{r_e^2} N + h\right)\sin(\mu) \end{bmatrix}$$

NED → ECEF/ECI

$$\begin{bmatrix} l \\ \mu \\ h \end{bmatrix} = \begin{bmatrix} a\tan\left(\frac{y}{x}\right) \\ a\tan\frac{z}{p}\left(1 - e^2 \frac{N}{N+h}\right) \\ \frac{p}{\cos(\mu)} - N \end{bmatrix}$$

Fuente: Fossen (Fossen, 2021)

2.3.2 Tranformación Cuerpo NED

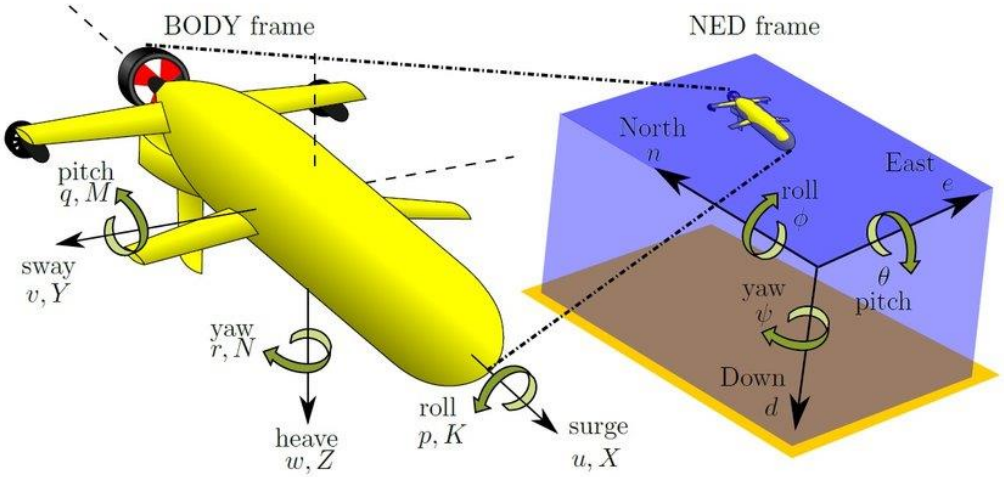
Las relaciones entre los sistemas de referencia vienen dadas por el teorema de Euler aplicado a la rotación. Obteniendo las matrices de rotación y translación:

$$R(\Theta_{nb}) = \begin{bmatrix} c\psi c\theta & -s\psi c\phi + c\psi s\theta s\phi & s\psi s\phi + c\psi c\phi s\theta \\ s\psi c\theta & c\psi c\phi + s\phi s\theta s\psi & -c\psi s\phi + s\theta s\psi c\phi \\ -s\theta & c\theta s\phi & c\theta s\phi \end{bmatrix}$$

$$T(\Theta_{nb}) = \begin{bmatrix} 1 & s\phi t\theta & c\phi t\theta \\ 0 & c\phi & -s\phi \\ 0 & s\phi / c\theta & c\phi / c\theta \end{bmatrix} \text{ Punto singular } \theta = \pm 90^{\circ}$$

Donde $c(\cdot) = \cos(\cdot)$ $s(\cdot) = \sin(\cdot)$ y $t(\cdot) = \tan(\cdot)$

Figura 4: Cuerpo y NED



Fuente: (Masmitja et al., 2018)

La transformación viene dada por la relación

$$\dot{\eta} = J_{\Theta}(\eta)v$$

También expresada como:

$$\begin{bmatrix} \dot{p}^{n}_{nb} \\ \dot{\Theta}^{n}_{nb} \end{bmatrix} = \begin{bmatrix} R(\Theta_{nb}) & O_{3x3} \\ O_{3x3} & T(\Theta_{nb}) \end{bmatrix} \begin{bmatrix} v^{n}_{nb} \\ \omega^{n}_{nb} \end{bmatrix}$$

Fuente: Fossen (Fossen, 2021)

2.3.3 Nave y corrientes oceánicas

Para una nave marina expuesta a corrientes oceánicas, resultan interesante introducir el concepto de velocidades relativas.

La velocidad de la corriente se define

$$v_c = \begin{bmatrix} u_c & v_c & w_c \end{bmatrix}^T$$

Por tanto, podemos definir las velocidades relativas como

$$v_r = \begin{bmatrix} u - u_c & v - v_c & w - w_c \end{bmatrix}^T = \begin{bmatrix} u_r & v_r & w_r \end{bmatrix}^T$$

2.4 Ecuaciones de movimiento

Las ecuaciones de movimiento han sido profundamente estudiadas por Fossen (Fossen, 2021), Estas pueden describirse en forma vectorial como:

$$M\dot{v} + C(v)v + D(v)v + g(\eta) + g_0 = \tau$$

Fuente: (Fossen, 2021)

Para una mejor compresión se va a realizar una descomposición en 4 partes: Dinámica, hidrostática, hidrodinámica y fuerzas externas

2.4.1 Dinámica del sólido rígido

$$M\dot{v} + C(v)v + D(v)v + g(\eta) + g_0 = \tau$$

$$M_{RB}\dot{v} + C_{RB}v = \tau_{RB}$$

Fuente: (Fossen, 2021)

Aplicando la segunda ley de Newton y los dos primeros axiomas de Euler se puede expresar el movimiento de translación y rotación del centro de gravedad expresado en el sistema de referencia de la nave como:

$$m\left[\dot{v}_{ng}^{b} + S\left(\omega_{nb}^{n}\right)v_{ng}^{b}\right] = f_g^b \qquad\qquad I_g^b\dot{\omega}_{nb}^{b} - S\left(I_g^b\omega_{nb}^{b}\right)\omega_{nb}^{b} = m_g^b$$

Donde I_g^b es el tensor de Inercia de la nave. y $S(\cdot)$ denota una matriz antisimétrica.

Posteriormente se realiza una transformación a los ejes locales, logrando la expresión final en forma matricial:

$$M_{RB}\dot{v} + C_{RB}v = \tau_{RB}$$

En la que M_{RB} es la matriz de masas del sólido rígido, C_{RB} es la matriz representante del efecto de Coriolis y fuerzas centrípetas debidas a la rotación del cuerpo alrededor del NED y τ_{RB} es el vector generalizado de fuerzas externas y momentos expresadas en el cuerpo.

$$M_{RB} = \begin{bmatrix} mI_{3x3} & -mS(r_g^b) \\ mS(r_g^b) & I_b^b \end{bmatrix} = \begin{bmatrix} m & 0 & 0 & 0 & mz_g & -my_g \\ 0 & m & 0 & -mz_g & 0 & mx_g \\ 0 & 0 & m & my_g & -mx_g & 0 \\ 0 & -mz_g & my_g & I_x & -I_{xy} & -I_{xz} \\ mz_g & 0 & -mx_g & -I_{yx} & I_{yy} & -I_{yz} \\ -my_g & mx_g & 0 & -I_{zx} & -I_{zy} & I_{zz} \end{bmatrix}$$

$$C_{RB} = \begin{bmatrix} mS(\omega_{nb}^b) & -mS(\omega_{nb}^b)S(r_{bg}^b) \\ mS(r_{bg}^b)S(\omega_{nb}^b) & -mS(r_{bg}^b)S(\omega_{nb}^b)S(r_{bg}^b) - S(I_g^b\omega_{nb}^b) \end{bmatrix}$$

Donde $r_g^b = [x_g \quad y_g \quad z_g]^T$ es el vector de posición del centro de gravedad.

Tanto M_{RB} como C_{RB} pueden expresarse de otras formas.

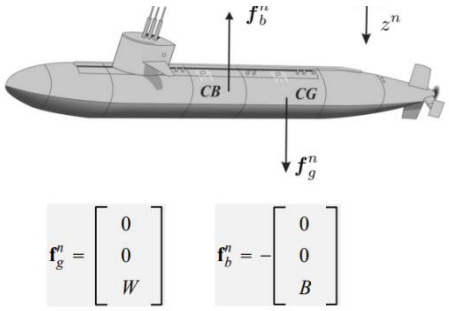
Fuente: (Fossen, 2021)

2.4.2 Hidrostática

$$M\dot{v} + C(v)v + D(v)v + g(\eta) + g_0 = \tau$$

De acuerdo a (SNAME, 1950) las fuerzas de flotación y debidas a la gravedad se expresan como:

Figura 5: Fuerzas hidrostáticas



Fuente: (Fossen, 2021)

$$W = mg \qquad B = \rho g \Delta$$

ρ densidad del agua

Δ volumen de fluido desplazado por el vehículo

m masa del vehículo

g aceleración de la gravedad

$$g(\eta) = \begin{bmatrix} (W-B)\sin(\theta) \\ -(W-B)\cos(\theta)\sin(\phi) \\ -(W-B)\cos(\theta)\cos(\phi) \\ -(y_g W - y_b B)\cos(\theta)\cos(\phi) + (z_g W - z_b B)\cos(\theta)\sin(\phi) \\ (z_g W - z_b B)\sin(\theta) + (x_g W - x_b B)\cos(\theta)\cos(\phi) \\ -(x_g W - x_b B)\cos(\theta)\sin(\phi) + (y_g W - y_b B)\sin(\theta) \end{bmatrix}$$

$r_b^b = [x_b \quad y_b \quad z_b]^T$ es el vector de posición del centro de flotación

Fuente:(Fossen, 2021)

g_0 es el efecto debido al lastre y su valor suele calcularse mediante análisis hidrostáticos (en régimen permanente).

En submarinos como el Sibiu Nano Plus, la flotabilidad suele ser positiva, para que en caso de avería el submarino suba a la superficie (Cárcel, 2020)

Además, la hidrostática añade un efecto de estabilidad. El empuje debido a la flotación y a la gravedad no se aplican en el mismo punto (Cárcel, 2020). Para que no se produzca un par, ambos centros deben encontrarse en la misma vertical. Cuando se produce una perturbación que produce un balanceo los centros dejan de ser coaxiales y se produce un par; dicho par anula el efecto de balanceo de la perturbación.

2.4.3 Hidrodinámica

$$M\dot{v} + C(v)v + D(v)v + g(\eta) + g_0 = \tau$$
$$M_A\dot{v}_r + C_A(v_r)v_r + D(v_r)v_r = \tau$$

Los efectos debidos a la hidrodinámica están descritos por Fossen y son un punto clave de diseño de

vehículos marítimos, y por tanto descritos por la arquitectura naval clásica (Fossen, 2021).

El modelo hidrodinámico clásico comprende la matriz de masa agregada M_A, la matriz hidrodinámica de fuerzas centrípetas de Coriolis C_A, y la matriz que representa el amortiguamiento hidrodinámico D.

$$M_A = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} = \begin{bmatrix} X_{\dot{u}} & X_{\dot{v}} & X_{\dot{w}} & X_{\dot{p}} & X_{\dot{q}} & X_{\dot{r}} \\ Y_{\dot{u}} & Y_{\dot{v}} & Y_{\dot{w}} & Y_{\dot{p}} & Y_{\dot{q}} & Y_{\dot{r}} \\ Z_{\dot{u}} & Z_{\dot{v}} & Z_{\dot{w}} & Z_{\dot{p}} & Z_{\dot{q}} & Z_{\dot{r}} \\ K_{\dot{u}} & K_{\dot{v}} & K_{\dot{w}} & K_{\dot{p}} & K_{\dot{q}} & K_{\dot{r}} \\ M_{\dot{u}} & M_{\dot{v}} & M_{\dot{w}} & M_{\dot{p}} & M_{\dot{q}} & M_{\dot{r}} \\ N_{\dot{u}} & N_{\dot{v}} & N_{\dot{w}} & N_{\dot{p}} & N_{\dot{q}} & N_{\dot{r}} \end{bmatrix}$$

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

$$v_1 = \begin{bmatrix} u_r & v_r & w_r \end{bmatrix}^T \qquad v_2 = \begin{bmatrix} p_r & q_r & r_r \end{bmatrix}^T$$

Fuente: (Fossen, 2021)

Su cálculo viene definido por las teorías de maniobrabilidad (Maneuvering Theory) y de estabilidad (Seakeeping Theory). Fossen realiza una descripción de ellas y los modelos matemáticos que las gobiernan (Fossen, 2011), estas se pueden resumir en:

- Teoría de estabilidad (seakeeping): estudia los cambios producidos respecto al estado de equilibrio de un vehículo sometido a oleaje desplazándose a velocidad constante. Además:
    - Introduce la fuerza disipativa conocida como memoria de fluido (Cummins, 1962).
    - Introduce las ecuaciones de Cummins en el dominio del tiempo y de la frecuencia (Cummins, 1962), a partir de las cuales llevan a cabo simulaciones por ordenador asumiendo teoría lineal y movimientos armónicos para calcular los coeficientes hidrodinámicos. A esto se conoce como CFD.
- Teoría de maniobrabilidad (Maneuvering): Asume que la nave no recibe excitación debida al oleaje (frecuencia cero), lo que permite definir las matrices de masa agregada y amortiguamiento mediante derivadas hidrodinámicas (parámetros constantes). Esta teoría es aplicable a profundidades de más de 20m, donde el efecto oscilatorio de las olas es despreciable. Además:
    - Hace uso de la segunda ley de newton describiendo todas las fuerzas externas que actúan sobre el submarino, definiendo así una matriz de 36 elementos de masa y otra de 36 de amortiguamiento, proporcionales a la velocidad y a la aceleración, que corresponden a los coeficientes hidrodinámicos lineales de masa agregada y de amortiguamiento respectivamente.
    - Para los términos no lineales se utiliza un desarrollo de Taylor, adoptando términos de orden impar en el amortiguamiento. La masa agregada se asume lineal.
    - Esta teoría no se puede aplicar para los ejes de balance y arfada, debido a que su frecuencia natural no se puede despreciar (no es cercana a 0).

El amortiguamiento hidrodinámico tiene distintas causas:

- Amortiguamiento potencial: Causado por la oscilación de un cuerpo a la frecuencia del oleaje. Si el cuerpo se encuentra fuera de la zona del oleaje (20m de profundidad), este término se puede despreciar.

- Fricción superficial: Aparece debido a la fricción entre la superficie de un cuerpo en movimiento y la capa de fluido a través de la que se mueve.
- Amortiguamiento de deriva de las olas: Desistencia añadida a superficies avanzando en olas.
- Amortiguamiento debido al desprendimiento de vórtice en bordes pronunciados.
- Amortiguamiento debido a fuerzas de ascenso.

2.4.3.1 Estimación de los coeficientes

El cálculo de los coeficientes hidrodinámicos ha sido tratado en artículos como el de Tang (Tang et al., 2009). En él se llega a conclusión que se refleja a continuación.

La matriz de masa agregada, y los coeficientes lineales de amortiguamiento suelen hallarse a partir de un prototipo, realizando pruebas en un túnel de viento o un tanque. Sin embargo, esto requiere un modelo físico preciso del vehículo, o el propio vehículo, así como un laboratorio o instalaciones de campo que permitan realizar estas pruebas. Las cuales no suelen estar disponibles, bien por coste, o porque el vehículo no está construido, como es el caso de este proyecto. Los modelos predictivos son una alternativa cuando el vehículo aún se está diseñando, o cuando el coste no permite un programa de pruebas a gran escala.

El modelo predictivo más simple es el puramente analítico. Sin embargo, este suele proveer resultados lejanos a la realidad para cuerpos romos complejos. Los métodos empírico y semi-empírico son más utilizados y se conoce que proveen resultados razonables en submarinos clásicos como los tipo torpedo.

Otro método para submarinos torpedo fue a través de dos observadores no lineares, uno deslizante y un filtro de Kalman, en el que un algoritmo de estimación del modelo fue usado, obteniendo los 15 coeficientes lineales de amortiguamiento considerados representativos por Sen (Sen, 2000).

Un acercamiento predictivo alternativo es usar CFD, el cual, gracias al avance de la tecnología computacional se considera uno de los acercamientos más realistas.

Entre los programas de CFD más reconocidos se encuentran los que utilizan los algoritmos basados en Boundary Element Methods. Entre ellos nos encontramos algunos como Capytaine (Ancellin & Dias, 2019) , Nemoh (Jamet, n.d.) y HAMS(Yingyi Liu, n.d.).

2.4.4 Fuerzas externas

Las fuerzas y momentos externos que actúan sobre un vehículo submarino se pueden dividir en debidas al viento, al oleaje, al umbilical, y al control y propulsión.

Como se ha indicado en el punto 2.1 del estado del arte, las fuerzas debidas al viento y el oleaje se modelan mediante una componente media y otra oscilatoria.

2.4.4.1 Debidas al Umbilical

Según Berg, (Berg et al., 2012) la fuerza de arrastre debida al umbilical se puede modelar mediante la fórmula de Morrison:

$$f(u_r) = \frac{1}{2}\rho\phi|\vec{v}_r|\vec{v}_r C_{A,umbilical}$$

Donde:

$u_r = \begin{bmatrix} x_u & y_u & z_u \end{bmatrix}^T$ posición del extremo conectado al ROV del umbilical

$\vec{v}_r$ velocidad relativa del agua

ρ densidad del agua

ϕ diámetro del umbilical

$C_{A,umbilical}$ coeficiente de arrastre del umbilical, generalmente Entre 0.2 y 1.2

Integrando esta ecuación en la longitud del umbilical, obtenemos la fuerza total que actúa sobre él. Al ignorar las fuerzas y corrientes verticales, se puede aproximar la longitud del umbilical a la altura h entre en operador y el ROV.

$$F_x^u = \int_0^h f(u_r) dz$$

Bajo la influencia de corrientes, y conociendo que el umbilical no está tenso, el umbilical formará una curva. Existe un punto llamado el punto de tangente vertical como se aprecia en la figura 6.

Si la corriente es uniforme, el punto vertical estará situado aproximadamente a la mitad de longitud, repartiendo la fuerza equitativamente entre el ROV y el operador.

Para la componente vertical, se considera el peso total del umbilical sumergido en agua. La longitud del umbilical se estima un 20% mayor que la altura (Berg et al., 2012). La fuerza vertical total se define, por tanto:

$$F_z^u = 1.2\omega_{umbilical} h$$

Donde $\omega_{umbiical}$ es el peso del umbilical por unidad de longitud.

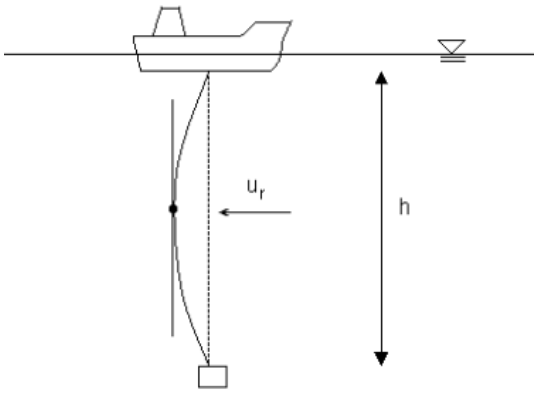
Finalmente, para calcular los momentos producidos de balance y arfada, el brazo entre el centro de gravedad y el punto de anclaje del umbilical debe hallarse. El momento de viraje se despreciará debido a el tamaño del brazo.

La fuerza queda definida:

$$\tau_{umbilical} = \begin{bmatrix} -\frac{1}{4}\rho d C_A \int_0^h u_r |u_r| dz \\ -\frac{1}{4}\rho d C_A \int_0^h v_r |v_r| dz \\ 1.2\omega_{cable} h \\ -r_z \cdot \frac{1}{4}\rho d C_A \int_0^h u_r |u_r| dz \\ -r_z \cdot \frac{1}{4}\rho d C_A \int_0^h v_r |v_r| dz \\ 0 \end{bmatrix}$$

Fuente: (Berg et al., 2012)

Figura 6: Disposición del umbilical



Fuente: (Berg et al., 2012)

2.4.4.2 Debidas al Control

Las fuerzas de control de un vehículo submarino pueden estar producidas por propulsores o por aletas.

Fossen (Fossen, 2021) describe un modelo simple del propulsor, mediante el cual la fuerza emitida por este es proporcional al cuadrado de la velocidad

$$T = K_T \rho n |n| \phi^4$$

Donde n es el número de revoluciones por minuto, ρ es la densidad del agua, ϕ es el diámetro del propulsor, y K_T es el coeficiente de propulsión (dependiente de la velocidad), calculable a partir de las curvas K_T-J descritas en el data sheet del propulsor.

Para describir la fuerza ejercida por los propulsores, se realiza una proyección de las fuerzas y momentos producidos por cada propulsor.

$$\tau_{control} = \sum_{i=1}^{r} \begin{bmatrix} F_{x_i} \\ F_{y_i} \\ F_{z_i} \\ F_{z_i} l_{y_i} - F_{y_i} l_{z_i} \\ F_{x_i} l_{z_i} - F_{z_i} l_{x_i} \\ F_{y_i} l_{x_i} - F_{x_i} l_{y_i} \end{bmatrix}$$

Fuente (Fossen, 2021)

John Carlton (Carlton, 2007) describe el funcionamiento del propulsor y los fenómenos que originan la fuerza producida por un propulsor. Fossen (Fossen, 2021) describe también el modelo eléctrico del propulsor. Además, describe un modelo para controlar la velocidad de los propulsores mediante un controlador P.

No se hace un análisis de la fuerza de control debida a las aletas, ya que no va a ser de aplicación en este TFG.

Tras analizar todos los efectos, a modo de resumen se puede desarrollar la ecuación final como:

$$\underbrace{M_{RB}\dot{v} + C^*_{RB}(v)v + M_A\dot{v} + C^*_A(v)v}_{\text{Fuerzas de inercia}} + \underbrace{(D_P + D_V)v_r + \mu_T}_{\text{Fuerzas de amortiguamiento}} + \underbrace{G(\eta) + g_0}_{\text{Fuerzas potenciales (restoring)}} = \underbrace{\tau_{viento} + \tau_{olas} + \tau_{control} + \tau_{umbilical}}_{\text{Fuerzas externas}}$$

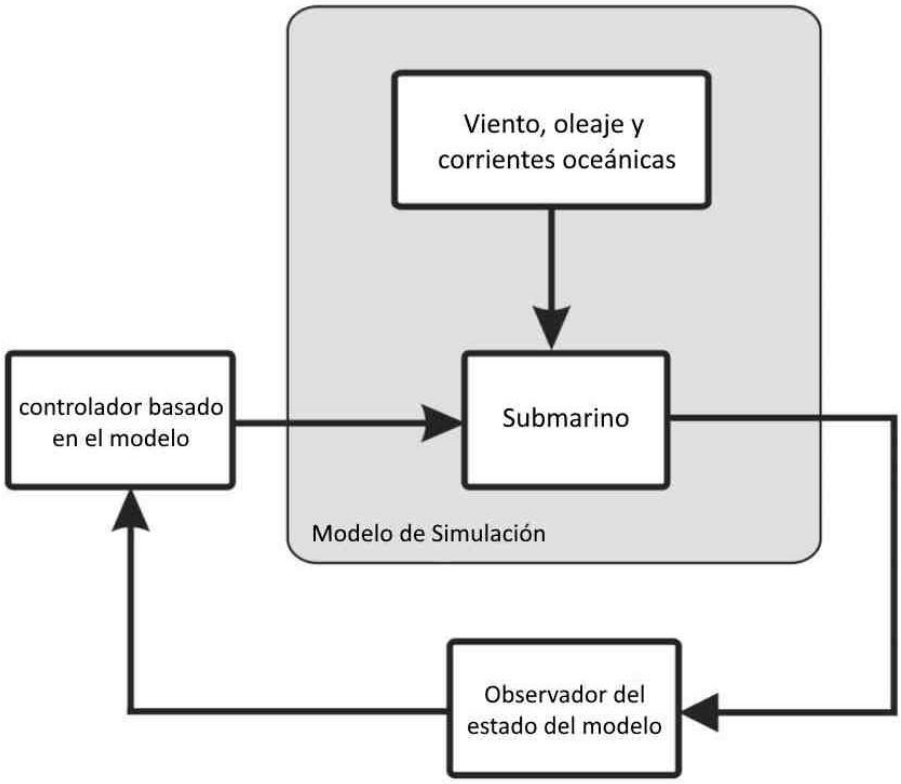
2.5 Modelos de predicción

Siguiendo a Fossen, se distinguen 3 tipos de modelos de predicción (Fossen, 2011):

- **Modelo de simulación**: Formado por la descripción más detallada del sistema, incluye dinámica, propulsión, medidas, fuerzas externas. Debe ser capaz de recrear las respuestas del sistema real y simular errores para simular eventos como accidentes y señales erróneas.
- **Modelo de diseño del control**: Es una versión simplificada del modelo de simulación, usado para computar una serie de ganancias constantes para controladores PID. El seguimiento de trayectorias requiere estados adicionales para la realimentación, además de filtrado, por lo que se suelen usar reglas control de un orden mayor.
- **Modelo de diseño del observador**: Su objetivo es capturar las dinámicas asociadas a los sensores y la navegación, así como las perturbaciones. Para naves marinas, suele incluir un

modelo de perturbaciones para estimar las fuerzas debidas al viento, oleaje y corrientes marinas; descritas como ruido de color.

Figura 7: Modelos de predicción del Submarino



Fuente: elaborado a partir de (Fossen, 2011)

2.6 Simulación

Las misiones subacuáticas con vehículos son costosas y consumen mucho tiempo, incluso en el mejor escenario, la comunicación acústica con submarinos es escasa. Dada la escasa capacidad de percepción, la evitación de obstáculos no es sencilla.

Resulta difícil repetir o reproducir ciertas misiones en la vida real. Debido a estas dificultades evaluar una misión por simulación antes de llevarla a cabo es crucial.

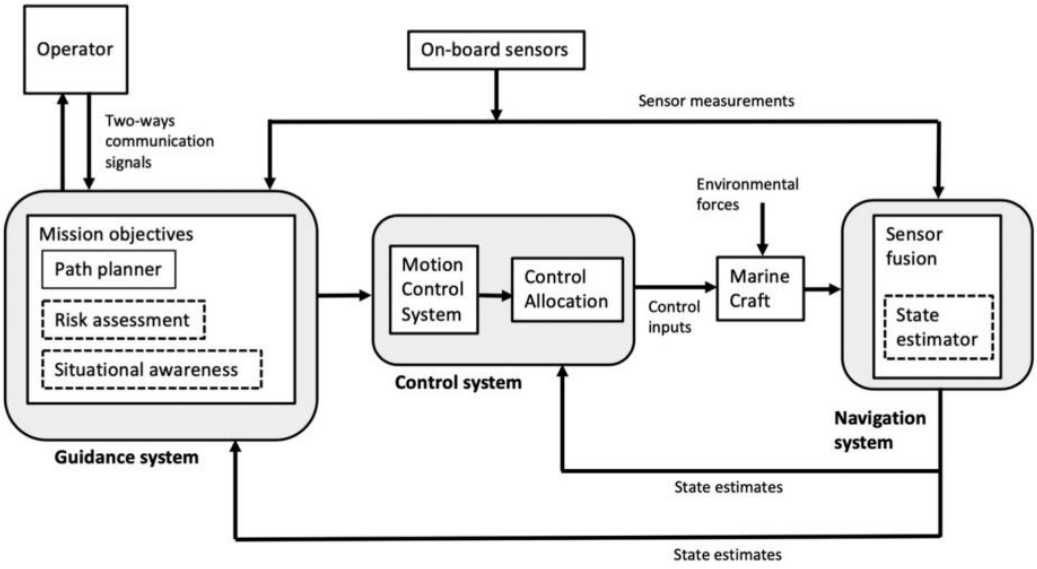
Entre los simuladores de submarinos presentes en ROS, existen una gran variedad que tienen uso genérico y que se podrían aplicar a submarinos, pero específicos para esta tarea se construye en 2011 UWSim. Más tarde en 2016 apareció UUVSim, incorporando una interfaz más amigable y simplificando el modelo del submarino (Rauschenbach, 2016), participando en el Primer RobotX Forum, siendo galardonado como un gran simulador submarino de código abierto.

Por último, en 2020 nació DaveSim, basado en UUVSim y ds_sim (un paquete de apoyo a la simulación) añadiendo nuevos modelos de sensores más modernos (presenta varias actualizaciones cada año) para simular la percepción y navegación así como manipuladores. Tiene como objetivo la simulación en torno a tuberías y plantas petrolíferas, luego facilita la creación de escenarios más complejos, gráficos más realistas, parametrización de corrientes oceánicas y realizar batimetrías del fondo oceánico, así como simular el comportamiento de interfaces eléctricas o de comunicación como protocolos serial o internet. Entre sus mayores logros está el de lograr llevar a cabo una labor de alta precisión debajo de agua en una planta petrolífera.

2.7 Control del submarino

El sistema completo de un ROV se puede describir con la siguiente figura:

Figura 8: Diagrama de un ROV



Fuente: (Fossen, 2021)

En él, distinguimos 3 subsistemas:

- Planificación: Dedicado a calcular el movimiento a realizar por el submarino.
- Control: Elabora la señal de actuación a introducir en los propulsores.
- Navegación: Calcula el estado actual del sistema a partir de sensores.

2.7.1 Clasificación del control

En algunos casos resulta interesante reducir el problema de 6 grados de libertad, bien porque las naves no pueden actuar sobre todos los grados de libertad o porque la tarea a realizar no nos requiere actuar sobre todos ellos. Los modelos de control más comunes incluyen(Fossen, 2011):

- 1 grado de libertad:
    - Vehículos que solo permiten **avance** y retroceso, autopilotos que corrigen la orientación en barcos (**viraje**), y sistemas de amortiguamiento del **balanceo**.
- 3 grados de libertad:
    - Modelos horizontales (**avance**, **deriva**, v**iraje**), para vehículos que requieran control de la trayectoria, seguimiento de caminos, o posicionamiento dinámico.
    - Modelos longitudinales (**avance**, **ascenso**, **arfada**) para controlar la velocidad, altura y cabeceo.
    - Modelos laterales (**deriva**, **balanceo** y **viraje**) para controlar el giro y la orientación.
- 4 grados de libertad:
    - Control de **avance**, **deriva**, **balanceo** y **viraje.** Es utilizado en situaciones de maniobrabilidad en la que es importante reducir el efecto del roll mediante aletas, propulsores o tanques de líquido estabilizador.

- 6 grados de libertad:
    - Actúa sobre todos los grados de libertad, Es usado para la simulación y predicción del movimiento de vehículos, así como sistemas de control avanzados para vehículos subacuáticos que actúan sobre todos los grados de libertad.

2.7.2 Estrategias de control

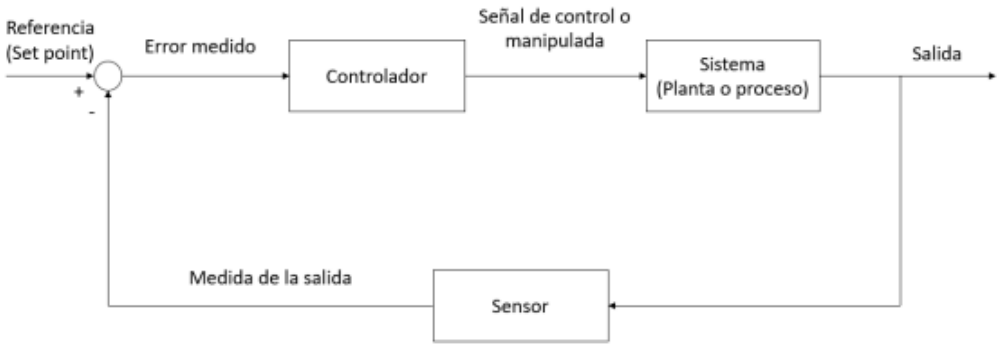
Dadas las perturbaciones citadas en el punto 2.2 del estado del arte se consideran para el control las estrategias de baja frecuencia, del oleaje, y la de ambos. Sus descripciones fundamentales son las siguientes:

- El control de baja frecuencia suele ser suficiente en la mayoría de casos en los que solo se busca la orientación y el control asistido de los propulsores para lograr alcanzar y mantener la posición deseada o realizar maniobras lentas para emerger a la superficie.
- El control del oleaje incluye el control del movimiento de balanceo y arfada; ejes que sufren las mayores aceleraciones y son causantes del efecto de mareo que sufren los pasajeros. Además, también es usado para soltar cargas, y reducir la carga media en los anclajes.
- Conforme las olas crecen en tamaño y se reducen en frecuencia, las fuerzas debidas a las olas aparecen en la frecuencia del control de movimiento del sistema, requiriendo tener en cuenta el oleaje además de la baja frecuencia. Este es el caso de vehículos posicionados cerca de orillas o tuberías, en los que el control en baja frecuencia se usa en bajo oleaje, y el control del oleaje y baja frecuencia cuando las olas aumentan en intensidad. Además, controlar la baja frecuencia y el oleaje es necesario para mantener el curso de la nave y reducir los efectos de balance en vehículos de superficie (Perez, 2005)

2.7.3 Tipos de control

El objetivo principal del control es que la salida del sistema adquiera un valor estable y constante. Dado el gran número de perturbaciones presentes en un submarino se adopta un modelo de control de lazo cerrado con el objetivo de conseguir un error nulo.

Figura 9: Control de lazo cerrado



Fuente: (Cárcel, 2020)

Entre los modelos más utilizados para realizar el control de un submarino se encuentra el uso de PID lineales, PID no lineales, reguladores lineales cuadráticos y control en modo de deslizamiento.

2.7.3.1 PID lineal

El PID lineal es la forma clásica de control. En él se elige un punto de funcionamiento y se realiza un modelo lineal de la planta en ese punto.

La salida de este tipo de controlador está formada por 3 parámetros. Uno proporcional, otro derivativo y un último parámetro integral. Sea $x(\tau)$ el error asociado al PID, la salida del PID sigue la forma:

$$salida = K_p x + K_d \dot{x} + K_i \int_0^t x(\tau) d\tau$$

Desde un punto de vista de la frecuencia, los valores de estos parámetros dictan el ancho de banda y la frecuencia natural del controlador.

Las formas de ajustar estos parámetros son variadas, entre otras nos encontramos: métodos heurísticos como el modelo de Ziegler Nichols, uso del lugar de las raíces, análisis de diagramas de Bode o algoritmos basados en redes neuronales.

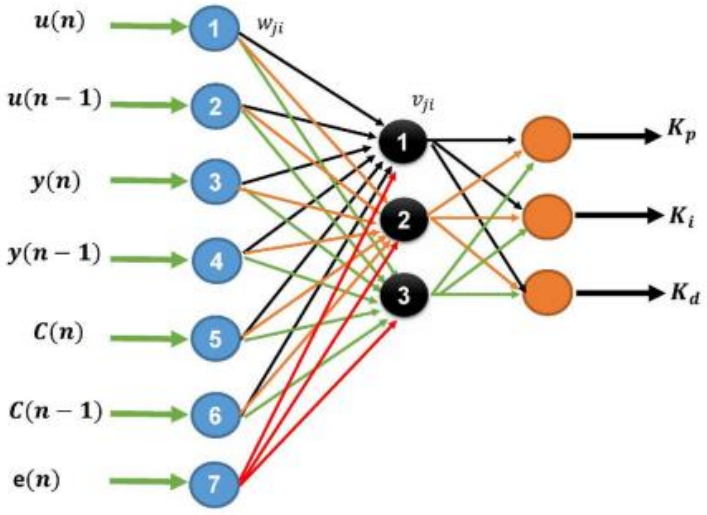
2.7.3.2 PID no lineal

Presenta la posibilidad de adaptarse a problemas no lineales realizando distintas aproximaciones lineales en distintos puntos de la planta. Se crea así, una matriz que almacena las distintas linealizaciones y los parámetros de cada una de ellas.

Pablo Aguirre Cárcel (Cárcel, 2020) describe en su TFG un modelo de PID con un ajuste de ganancias continuo, diseñado para sistemas sujetos a cambios paramétricos continuos o con perturbaciones externas, mejorando la respuesta del ROV ante corrientes oceánicas, cambios de herramienta o uso de lastres.

Este modelo se basa en un PID de autoajuste basado en redes neuronales que estiman automáticamente el valor de ganancias manteniendo la estabilidad del sistema.

Figura 10: Esquema gráfico de una red neuronal

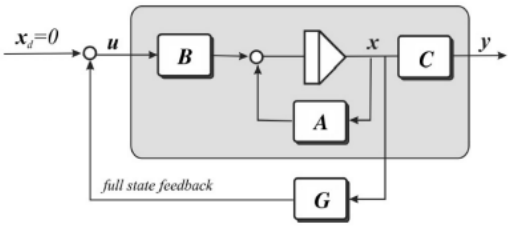


Fuente: (Cárcel, 2020)

2.7.3.3 Regulador lineal-cuadrático

Este tipo de control busca operar un sistema dinámico a un costo mínimo. Para ello define una función cuadrática de coste y una serie de ecuaciones lineales que definen la dinámica del sistema (Bru Cervantes Anaya, n.d.).

Figura 11: Esquema regulador lineal-cuadrático



Fuente (Fossen, 2021)

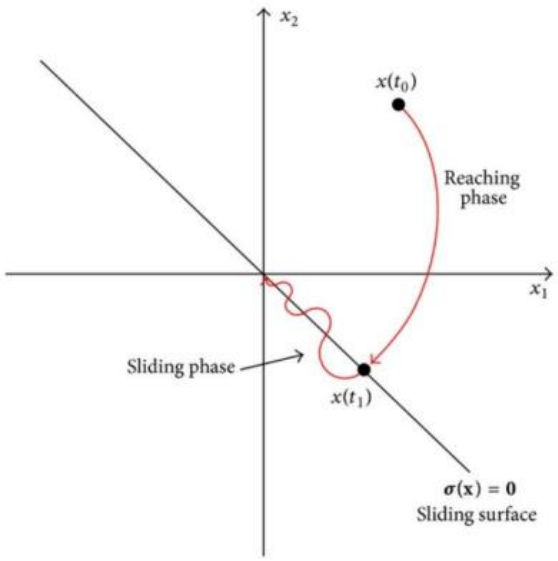
Existe una versión mejorada que añade efecto integrador, permitiendo aumentar el espacio de estados del modelo (Fossen, 2021).

2.7.3.4 Control en modo de deslizamiento

Es un método robusto de control que incorpora técnicas para manejar sistemas con comportamientos no conocidos o no modelados. Alterando la dinamica no lineal del sistema mediante la aplicación de una señal de control discontinua.

Presenta una estructura variable que permite cambiar entre estructuras continuas en el tiempo a otras basadas en la posición en el espacio de estados. El sistema así sufre un "deslizamiento" respecto al comportamiento descrito mediante controladores tradicionales.

Figura 12: Comportamiento control en modo de deslizamiento



Fuente: (Cárcel, 2020)

2.7.4 Planificación

El Sistema de guiado de un submarino puede tener varios objetivos (Fossen, 2021):

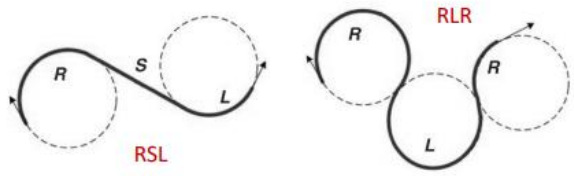
- Regulación de posición: basado en el posicionamiento dinámico, manteniendo la posición y orientación constantes.
- Seguimiento de objetivos: no hay información del camino ni trayectoria que seguir, se utiliza la posición y velocidad de un objetivo como referencia.
- Seguimiento de caminos; Se debe circular por un camino predefinido independientemente del tiempo utilizado.

Para poder realizar estos objetivos, se utilizan una serie de puntos objetivos intermedios denominados waypoints. La correcta elección de estos necesita tener en cuenta:

- La misión: punto de inicio y final.
- Datos ambientales: oleaje, corrientes, para evitar consumos de energía o zonas peligrosas.
- Factibilidad: debe ser posible dirigirse al siguiente punto sin exceder la velocidad máxima ni ratio de giro.
- Datos geográficos: zonas prohibidas o protegidas.
- Obstáculos: deben evitarse.

Suele utilizarse el camino de Dubins que dicta que el camino más corto (menos tiempo) entre dos posiciones (x,y,viraje) de un vehículo moviéndose a velocidad constante es un camino formado por líneas rectas y segmentos circulares.

Figura 13: Camino de Dubins



Fuente: (Fossen, 2021)

2.7.5 Navegación

Los vehículos subacuáticos usan sistemas de referencia hidroacústicos para determinar su posición, velocidad y trayectoria.

La solución de navegación se divide en 2 tipos (Fossen, 2021):

- Basada en modelo: utiliza un filtro de Kalman integrando las medidas de aceleración, velocidad y profundidad, mediante un modelo matemático.
- Basada en un sistema de navegación de inercia: utiliza los datos de aceleración y velocidades angulares como entradas para integrar las ecuaciones cinemáticas. Este tipo de navegación presenta problemas debidos a las imprecisiones del modelo.

Pablo Aguirre Cárcel (Cárcel, 2020) describe en su TFG el sistema de navegación de un AUV guiado mediante un sistema de visión artificial.

Este sistema hace uso de redes neuronales que analizan la imagen subdividiéndola diferentes regiones, extrayendo parámetros y comparando la relación entre regiones o filtrando la imagen utilizando umbrales dinámicos.

2.7.6 Distribución del control

Como se ha explicado en el punto 2.4.4.2 del estado del arte, la señal de control se modela como un vector de fuerzas.

Dado un vector fuerza de control, es necesario hacer una transformación cinemática inversa al empuje realizado por cada propulsor, eligiendo así la entrada de potencia correspondiente en cada motor. Esta función inversa suele ser un problema con infinitas soluciones.

Los objetivos de la distribución de control se basan en el uso de la mínima potencia necesaria reduciendo el consumo y aumentando el tiempo de vida de los propulsores, así como mantener el empuje asociado a cada propulsor dentro de los límites de saturación.

Fossen describe 2 métodos para resolver computacionalmente la optimización del sistema de ecuaciones con infinitas soluciones (Fossen, 2021);

- Optimización por mínimos cuadrados
- Aplicando restricciones y resolviendo una función lineal para reducir la fuerza máxima

Sin embargo, estos métodos suelen requerir del uso de iteraciones para encontrar una mejor solución.

Una de las ventajas de la distribución de control aparece cuando un motor falla, ya que esta permite reordenar la distribución de control manteniendo el vector fuerza de referencia.

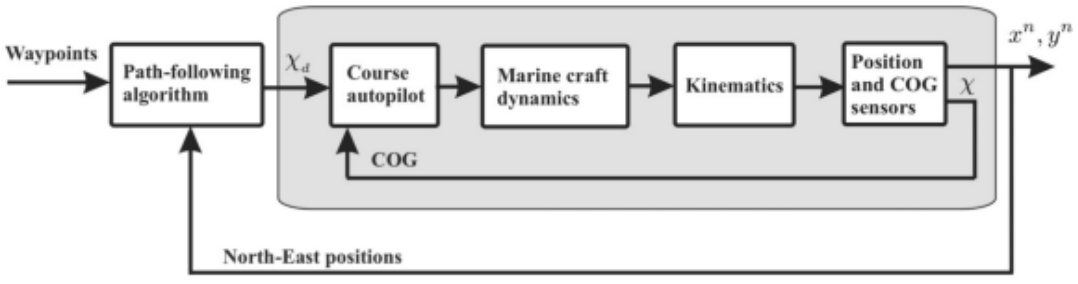
2.7.7 Autopilotos

Los vehículos marinos suelen poseer un autopiloto, este es un sistema de control que ejecuta la planificación, control y navegación de forma automática.

Fossen describe los autopilotos según el modelo de Nomoto (Fossen, 2021). Realizando una división de estos en:

- Autopilotos de trayectoria: siguen una lista de puntos. La realimentación incluye la posición, velocidad y posición del centro de gravedad.

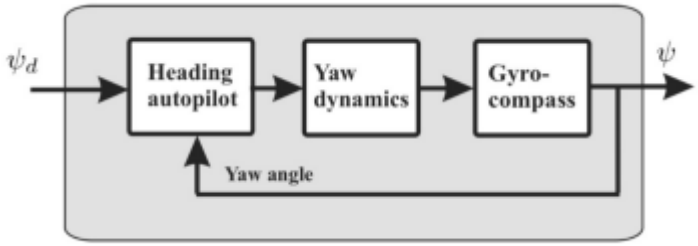
Figura 14: Autopiloto de trayectoria



Fuente: (Fossen, 2021)

- Autopiloto de orientación: la entrada del control es un momento de viraje generado por los propulsores. La realimentación se realiza mediante SOG y una brújula.

Figura 15: Autopiloto de orientación



Fuente: (Fossen, 2021)

2.8 Sensores

2.8.1 Cámara

Como se ha descrito anteriormente, el ambiente submarino esta caracterizado por tener poca visibilidad, lo que hace que la cámara en este ambiente pocas veces puede ser usada como sensor de posición. Takimoto realizó un experimento de posicionamiento mediante cámara en agua limpia (Takimoto et al., 2008), tras el que llegó a la conclusión de que para obtener una buena estimación es necesario mantener unas condiciones de poco oleaje e inmersiones cortas y no muy profundas. Sin embargo, es muy utilizada para recopilar imágenes, realizar seguimiento y en el manejo de ROVs.

2.8.2 Global Positioning System (GPS)

Es un sistema formado por una antena y un procesador que hace uso de una constelación de satélites como BeiDou, GLONASS o Galileo para localizarse en el plano de la Tierra. Se barre un rango de frecuencias buscando un satélite. Una vez encontrado, a través de una lista se buscan satélites cercanos en sus respectivas frecuencias y mediante trilateración de estos se estima la posición representada como [longitud, latitud, altura] además de una estimación de la precisión de estos datos.

Esta información es de gran utilidad en AUV ya que permite realizar un seguimiento del recorrido del submarino y hacer correcciones en misiones transoceánicas.

Las coordenadas GPS se han desarrollado en el punto 2.3.1 del estado del arte.

2.8.3 IMU

Una unidad de medición inercial es un sistema microelectromecánico que permite conocer la orientación de un cuerpo.

Existen diferentes tipos de IMU, generalmente están formados por un acelerómetro, un giróscopo y un magnetómetro, pero se dan casos en los que solo se utiliza uno o dos de ellos.

El cálculo de la orientación se realiza mediante algoritmos, siendo los más conocidos el algoritmo de

Integración por Mahony (Mahony et al., 2008), Madgwick (Madgwick, 2010) o Kalman (Shan et al., 2017).

2.8.4 Barómetro

Un barómetro es un sistema microelectromecánico que permite conocer la presión atmosférica.

Uno de los usos más comunes es la estimación de la profundidad. La presión se puede calcular como:

$$P = P_{atmosférica} + \rho g h$$

Sin embargo, este método presenta cierto error, debido a la vulnerabilidad ante cambios de densidad del agua marina y de temperatura, necesitando calibrado.

2.8.5 Temperatura

Existen una gran cantidad de sensores que nos proporcionan valores de temperatura, siendo los más comunes los termopares.

2.9 Comunicación

Las ondas de radio no funcionan igualen el medio submarino que en el medio aéreo. Si se quisiera usarlas se necesitarían longitudes de onda mayores, y por tanto antenas de mayor tamaño.

Una posible solución es recurrir a ondas acústicas como las utilizadas por la empresa DSPComm (Ellepola & Gustavino, 2020). Las ondas acústicas se propagan a mayor velocidad en el agua, haciendo de este sistema plausible. Sin embargo, presentan una gran cantidad de perturbaciones debido a otros sonidos como los propulsores del submarino, lo que limita su rango de funcionamiento.

Otros submarinos mantienen antenas de radio las cuales utilizan para transmitir información por satélite realizando ascensos periódicos a superficie.

En el caso de los ROV, este sucede mediante una serie de pares trenzados de cable o fibra óptica denominado umbilical a través de los que se realiza el tráfico de información. Estos mantienen una línea de tensión por la que circula una señal de Ethernet, siendo uno de los protocolos más usados MAVLink (*MAVLink*, n.d.),

2.10 Software

Las tareas de modelado, simulación y control se apoyan en una serie de herramientas software. En este TFG se ha optado por la utilización de herramientas de código abierto.

Así, para modelado se ha utilizado Freecad, Para el cálculo de las características del modelo MeshLab, y para el diseño del control y la simulación ROS, mediante el uso del simulador UUVSim basado en Gazebo, creando scripts en Python.

Freecad

Freecad es un software abierto multiplataforma de modelación paramétrica en 3D altamente usado en diseño de productos en el sector industrial, permitiendo modificar fácilmente diseños mediante la modificación de parámetros. Este permite crear partes a partir de bocetos basados en geometrías 2d isorestringidas y ajustar dimensiones para crear modelos 3D de gran calidad. Dispone de una gran cantidad de complementos gratuitos, así como una gran comunidad activa. Admite una gran cantidad de formatos de archivo (*FreeCAD: Your Own 3D Parametric Modeler*, n.d.).

Meshlab

Meshlab es un software abierto multiplataforma utilizado para procesar y editar modelos 3D basados en mallas de triángulos. Incluye una gran cantidad de herramientas para editar, limpiar, corregir, inspeccionar, renderizar, añadir texturas y convertir modelos 3D (*MeshLab*, n.d.).

Además, permite calcular las propiedades físicas de un modelo 3D.

ROS

ROS es un meta-sistema operativo de código libre para robots superpuesto a Unix. Este provee los servicios esperados en un sistema operativo: abstracción de hardware, control de dispositivos a bajo nivel, implementación de funcionalidades usuales, intercambio de mensajes entre servicios y gestión de paquetes. Así como herramientas para obtener, construir, escribir y ejecutar código a través de múltiples ordenadores.

ROS se organiza mediante una red entre iguales de procesos que están libremente enlazados a través de la infraestructura de comunicación de ROS. En esta estructura cada uno de los procesos en ejecución se denomina 'Nodo'. La comunicación entre programas se realiza mediante los denominados 'topics', además posee otro tipo de comunicación bidireccional de espera activa llamado 'servicio'. Destacar que ROS no es una infraestructura en tiempo real.

Entre sus características destaca su compatibilidad con otras infraestructuras software de robots, la multitud de librerías que incluye, la independencia de lenguaje, la facilidad de testeo y el escalado a procesos mayores (*Documentation - ROS Wiki*, n.d.).

Gazebo

Gazebo es un simulador 3D dinámico con la capacidad de realizar de forma precisa y eficiente la simulación de poblaciones de robots en entornos cerrados y abiertos. Al igual que los motores de juego, ofrece una simulación de físicas con un mayor nivel de fidelidad, una serie de señales e interfaces para usuarios y programadores (*Gazebo*, n.d.).

UUVSim

UUVSim (Rauschenbach, 2016) es un simulador de robots subacuáticos utilizado para probar algoritmos de manipulación subacuática, planificación, y el comportamiento de los vehículos ante perturbaciones y escenarios hostiles. (Manhães, 2021) Tiene a su disposición distintos ROVs, así como una herramienta de creación de nuevos modelos vía URDF.

Dispone de extensiones para simular el comportamiento de propulsores, aletas, objetos subacuáticos, así como concentración de partículas. Además, al estar basado en Gazebo contiene una gran cantidad de modelos de sensores y actuadores, así como la capacidad de crear nuevos modelos.

Por último, dispone de una serie de módulos para el control de trayectoria y de los propulsores.

Python

Python es un lenguaje de programación de alto nivel y código abierto que permite un simple pero efectivo acercamiento a programación orientada a objetos.

Su naturaleza interpretada lo hacen un lenguaje ideal para realizar scripts y desarrollar rápidamente aplicaciones en diversas áreas en la gran mayoría de plataformas. Manteniendo un código limpio y fácil de entender.

Cuenta con una gran variedad de librerías y una gran comunidad (*Welcome to Python.Org*, n.d.).

3 SIBIU NANO PLUS

El submarino a tratar es el Sibiu Nano Plus, diseñado y desarrollado por la empresa Nido Robotics, cuyas características se pueden observar en su página web www.nidorobotics.com y el manual del Sibiu Nano Plus:(ROBOTICS, n.d.-c).

Ilustración 7: Sibiu Nano Plus

Fuente: https://www.nidorobotics.com

Se trata de un producto nuevo en proceso de fabricación de las primeras unidades, a fecha actual, dicha información es incoherente con los datos visuales, y tras ponerse en contacto con la empresa, se deduce errónea y perteneciente a un modelo antiguo "Sibiu Nano".

Con el objetivo de poder realizar un modelo fiel, se va a optar por realizar una serie de estimaciones:

- El Sibiu Nano Plus está formado por los mismos materiales que el Sibiu Nano.
- El Sibiu Nano Plus posee los mismos sensores y actuadores que el Sibiu Nano.
- La electrónica que gobierna el Sibiu Nano Plus es la misma que el Sibiu Nano.

Para ello, se ha utilizado la información de la página web citada anteriormente (ROBOTICS, n.d.-c), el manual del Sibiu Nano Plus (ROBOTICS, n.d.-a) y la información descrita en el TFG de Enrique González Sancho (Enrique González Sancho, 2018) recopilando los datos siguientes.

3.1 Características básicas

A partir de la pagina web del fabricante y conversación con personal técnico del mismo se ha recogido la siguiente información del prototipo.

Tabla 2: Características básicas del Sibiu Nano Plus

Dimensiones	[42 30 24]cm
Posibilidad de incluir un lastre	800 g
Material	Resina de flotación R-3312
Conector de batería	XT90
Batería	4s 6750mAh (14.8V)
Diámetro umbilical	4mm
Densidad del umbilical	1-1,05 g/cm3
Profundidad Máxima	100 m
Propulsores	NM-150
ESC	20 A
Corriente máxima de trabajo	2 nudos
luces	4 x 1500 lumens
Velocidad máxima	2 nudos (3.7 km/h)

Fuente: Elaboración propia a partir de la empresa fabricante del submarino

3.1.1 Sensores

Entre los sensores disponibles se encuentran:

- IMU (Giroscopio, Acelerómetro y Magnetómetro)
- Sensor de presión
- GPS
- Cámara
- Barómetro
- Humedad
- Temperatura
- Voltaje y Corriente

3.1.2 Actuadores

Entre los actuadores disponibles se encuentran:

- 4 propulsores verticales
- 4 propulsores horizontales

- La posibilidad de colocar una garra

3.1.2.1 Propulsores

Los propulsores son el modelo NM-150 cuyas características se resumen en la siguiente doble tabla.

Tabla 3: Características NM-150

Dimensiones	Motor	ø 28 x 48.5mm
	Hélice	ø 68 mm
Propulsión Máxima	3kg	

Ilustración 8: Propulsor NM-150

Fuente: (Enrique González Sancho, 2018)

电压 Voltage(V)		No Load 空载		Rated Load 额定负载				Stall 堵转	
Operating Range	Rated	Speed (rpm)	Current (A)	Torque (mNm)	Speed (rpm)	Current (A)	Output P. (W)	Torque (mNm)	Current (A)
3S、4S	12	6300	0.34	170.3	3150	8	56.2	340.5	15.64
3S、4S	14.8	8120	0.38	208	4070	10.54	89	416.8	20.8
3S、4S	16	8951	0.41	226.7	4476	11	106.3	453.4	21.6
3S、4S	24	13000	0.62	329	6500	15.43	224	658.1	30.25

Fuente: (Enrique González Sancho, 2018)

Tomando en cuenta el modelo descrito en el apartado 2.4.4.3 del estado del arte se puede definir el empuje del propulsor proporcional al cuadrado de la velocidad. Pese a poseer un valor dependiente de la velocidad, en nuestro modelo se a suponer K_T constante.

Ya que la velocidad es proporcional a la potencia mecánica, y esta puede modelar su relación con la eléctrica como una eficiencia. se puede decir que la velocidad del propulsor es proporcional a la potencia eléctrica y, por tanto, existe una relación cuadrática entre la entrada (potencia eléctrica) y salida del propulsor (empuje),

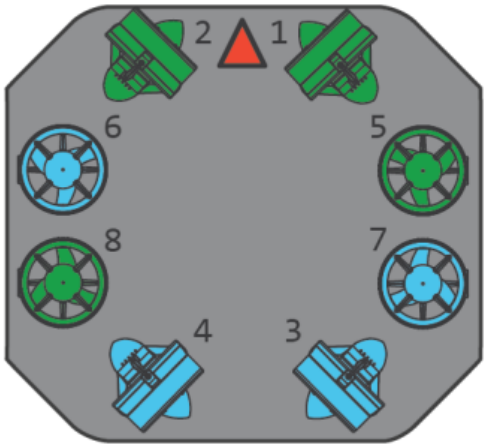
Observando la tabla 3, dado que la batería es de 14.8 V, para 156 W se observa una velocidad de 4070 rpm y se encuentra la fuerza máxima de propulsión de 3 kg. Es de esperar que para una potencia 0 la velocidad y fuerza ejercidas sean nulas.

A partir de estos datos, ignorando pérdidas, se puede realizar una linealización obteniendo la constante de empuje del propulsor como:

$$C_r = \frac{3kg \cdot 9.8 \frac{m}{s^2}}{(156W)^2} = 0.0012081\left[\frac{N}{W^2}\right]$$

La configuración estructural de los propulsores se puede observar en la figura 16, en la que se observa una vista superior del submarino donde los propulsores, en verde. indican sentido antihorario y los azules, sentido horario.

Figura 16: Configuración propulsores Sibiu Nano Plus

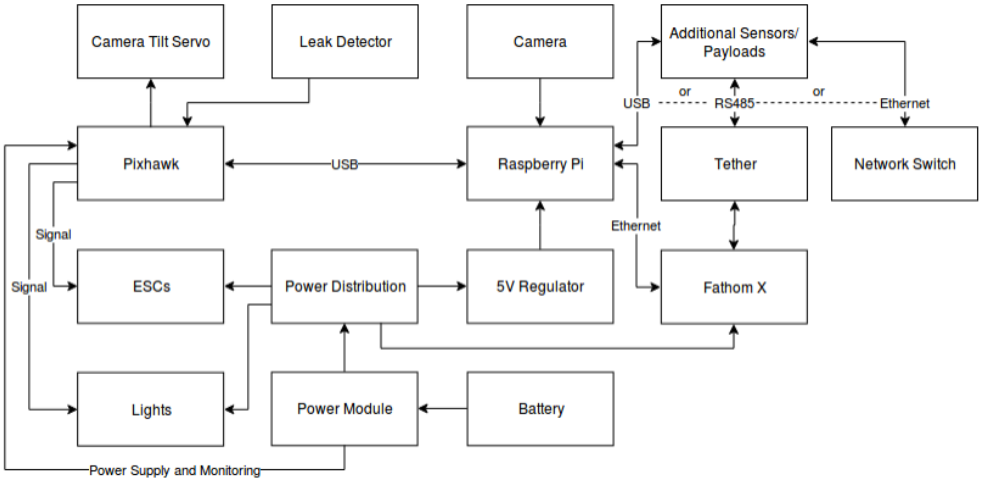


Fuente: (Robotics, n.d.)

3.1.3 Electrónica

La arquitectura de la electrónica del submarino puede resumirse en el siguiente diagrama.

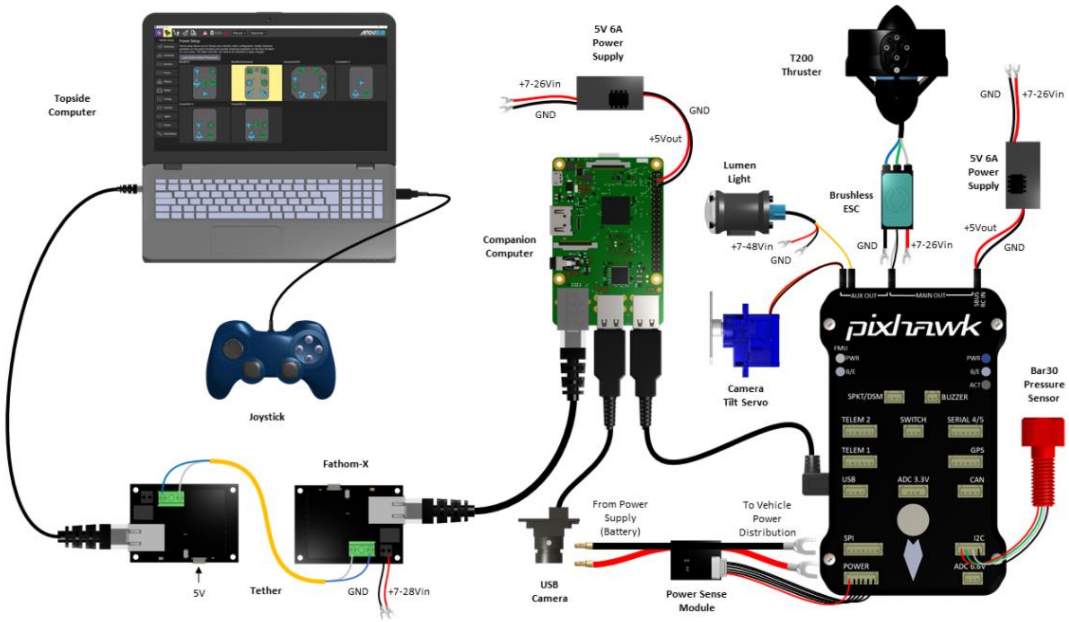
Figura 17: Arquitectura electrónica Sibiu Nano Plus



Fuente: (Enrique González Sancho, 2018)

Un ejemplo más visual de esta estructura se aprecia en la figura 18, donde se aprecia la conexión de todos los componentes electrónicos.

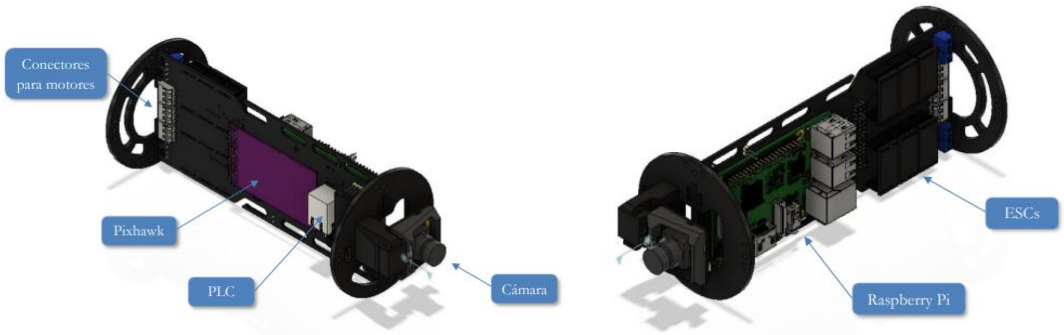
Figura 18: Conexionado Sibiu Nano Plus



Fuente: (Robotics, n.d.)

La disposición de estos componentes en el Sibiu Nano Plus se aprecia en la siguiente imagen,

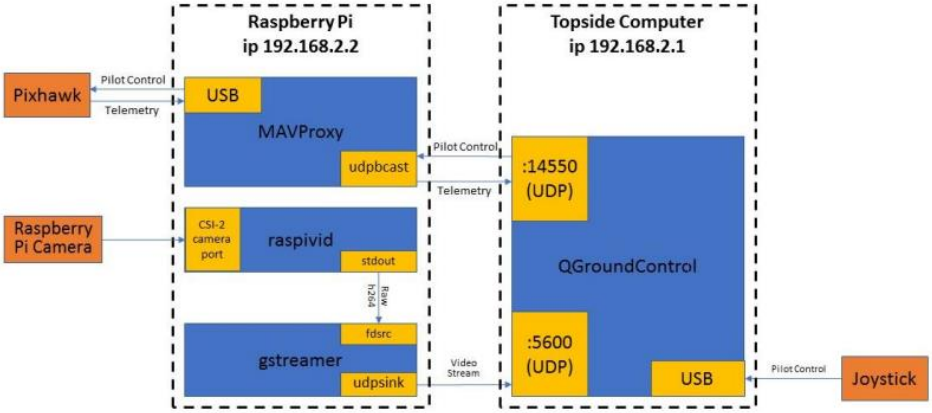
Figura 19: Electrónica Sibiu Nano Plus



Fuente: (Enrique González Sancho, 2018)

De la misma forma puede describirse el funcionamiento de los módulos Software.

Figura 20: Módulos Software Ardusub



Fuente: (Robotics, n.d.)

3.1.3.1 Autopiloto

El submarino Sibiu Nano Plus posee un autopiloto Pixhawk programado por el software de código abierto Ardusub de Ardupilot.

Ilustración 9: Autopiloto Pixhawk

Fuente: (Enrique González Sancho, 2018)

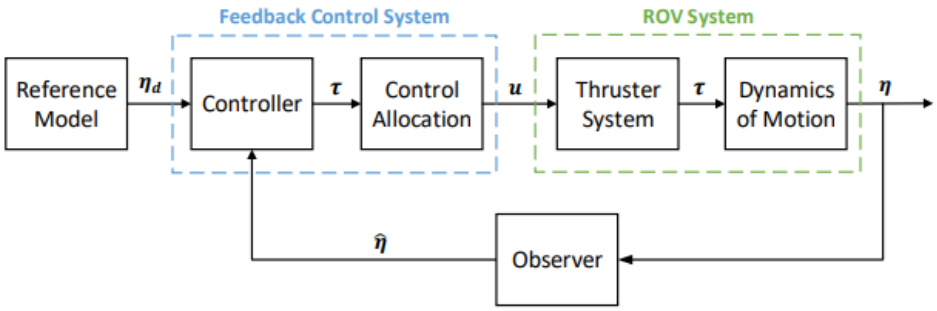
Este autopiloto es capaz de realizar múltiples funciones

- Utiliza un filtro de Kalman Extendido para estimar la posición del submarino.
- Regula las aceleraciones de Pitch y Roll, manteniendo el submarino estable.
- Resuelve el problema de cinemática inversa, permitiendo controlar el submarino remotamente mediante un Joystick como está descrito en la figura 22.

Descritas mediante el sistema de control de la figura 21.

Además, como se puede observar en la figura 23, Ardusub nos provee con una interfaz de usuario que nos permite observar las medidas tomadas por el submarino y el estado de este.

Figura 21: Esquema de Control Ardusub

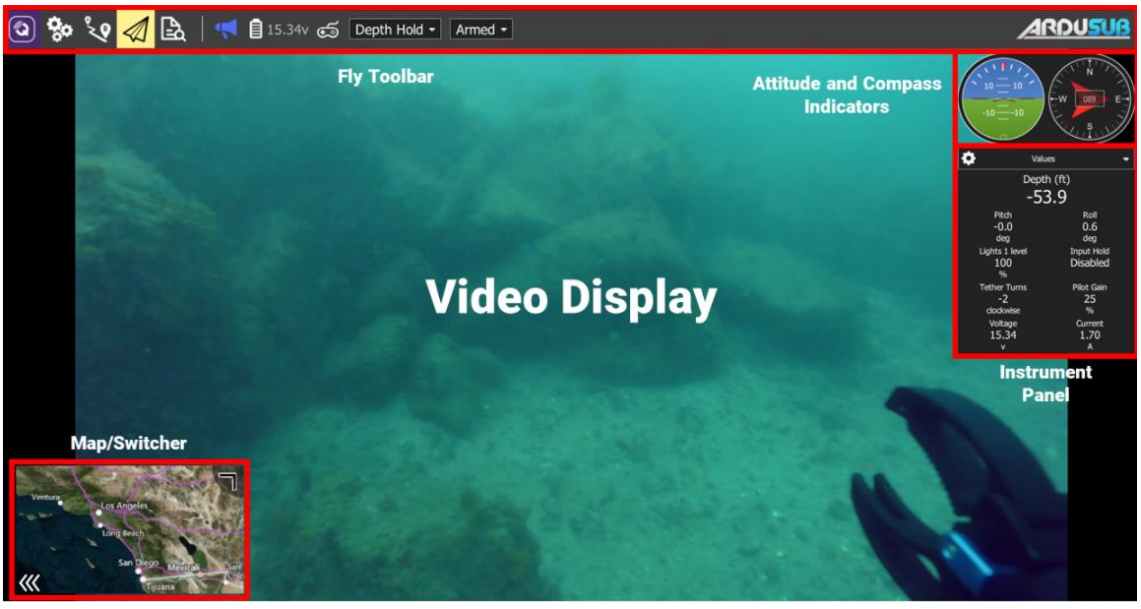


Fuente: (Wu, 2018)

Figura 22: Funcionamiento joystick Sibiu Nano Plus

Fuente: (ROBOTICS, n.d.-a)

Ilustración 10: Interfaz Ardusub



Fuente: (Robotics, n.d.)

3.1.3.2 Comunicación

La comunicación se realiza vía Ethernet mediante el protocolo MavLink a través de un único par trenzado. Es necesario convertir la señal utilizando un PLC. Esta tecnología permite extender el rango del cable umbilical hasta 2km (Enrique González Sancho, 2018).

3.1.3.3 Bateria

El submarino posee una batería de 6750 mAh de capacidad a 14.8 V, el consumo total del sistema se caracteriza principalmente por el consumo debido a los propulsores. Según datasheet los motores consumen de forma nominal 10.54 A. El consumo de la electrónica se va estimar como 3 A según Enrique González (Enrique González Sancho, 2018).

El consumo total será de 87.32 A.

Suponiendo un uso normal de los motores (10% potencia durante el 70% del tiempo) se obtiene un consumo medio aproximado de 9 Ah.

Obteniendo una duración aproximada de 45 minutos. Sin embargo, en la práctica, el consumo se estima menor y, por tanto, la duración de la batería mayor.

3.2 Modelo 3D

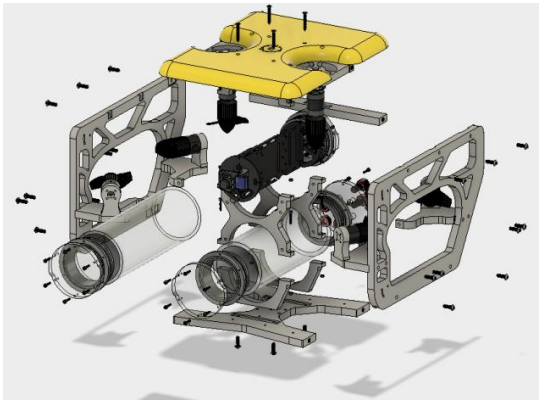
Conocida la apariencia del submarino en la ilustración 7 se procedió por crear un modelo 3D del submarino, haciendo uso de la herramienta Freecad antes mencionada.

El primer modelo del submarino se a partir del Sibiu Nano, la versión anterior del Sibiu Nano Plus; del cual existe una ficha técnica que describe completamente sus componentes en la página web oficial (ROBOTICS, n.d.-b) o el TFM realizado por Enrique González Sancho (Enrique González Sancho, 2018)

Ilustración 11: Sibiu Nano

Fuente: (Nido Robotics Sibiu Nano, n.d.)

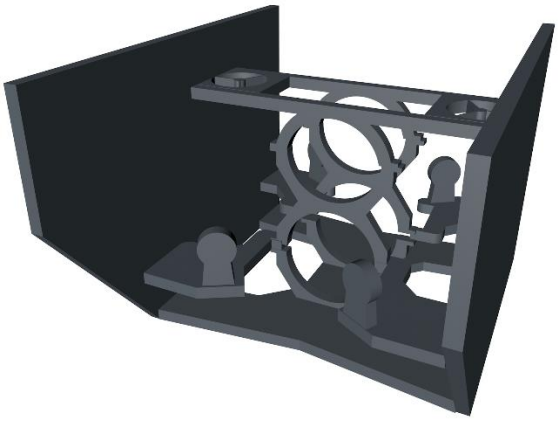
Ilustración 12: Despiece Sibiu Nano



Fuente: (BackerClub, n.d.)

Obteniendo un primer prototipo que se denominará ‘v0’

Ilustración 13: prototipo v0



Fuente: Elaboración propia

Tras analizar este primer acercamiento, se observa que no se cumplen las proporciones respecto a la imagen del Sibiu Nano Plus. Por lo que se decide realizar un nuevo modelo, tomando como medida estándar del borde 1cm, y analizando la imagen en más detalle, para mantener una proporción adecuada. Además se analizó la estructura de otros submarinos como el Sibiu Pro (ROBOTICS, n.d.-c) y el Rex ROV (*UUVSim RexROV*, n.d.), encontrando puntos en común como que los propulsores horizontales forman un ángulo de 45º con la longitudinal. Esta es una característica crítica para el buen funcionamiento del modelo y difícil de apreciar en las imágenes, que en caso de ser errónea debe actualizarse.

Ilustración 14: Sibiu Pro

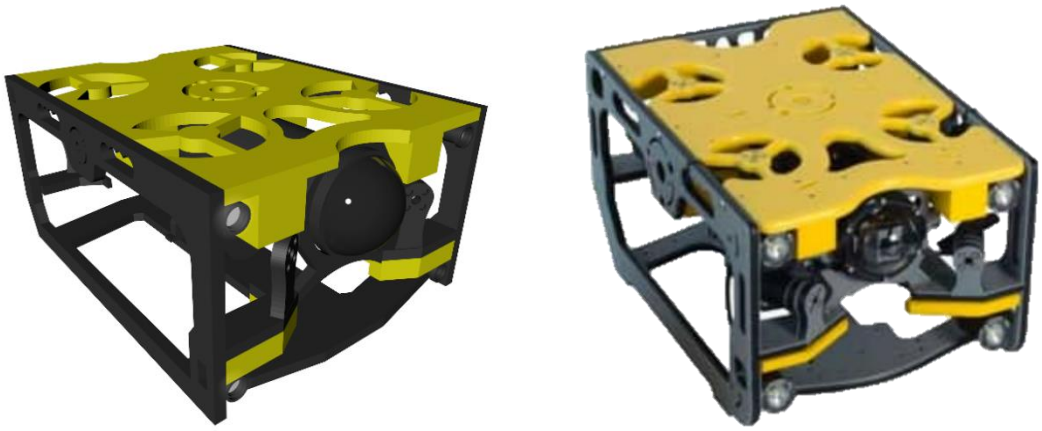
Fuente: (Nido Robotics, n.d.)

Ilustración 15: Rex ROV

Fuente: Elaboración propia a partir de (UUVSim RexROV, n.d.)

Tras este análisis se llegó al prototipo ‘v1’, visualizado en la ilustración 16.

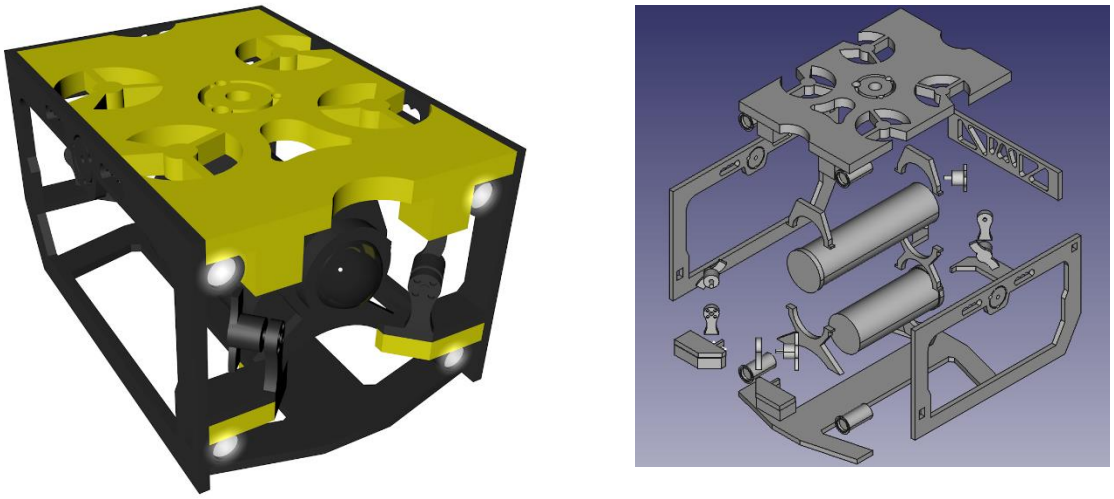
Ilustración 16: Prototipo v1 vs Sibiu Nano Plus



Fuente: Elaboración propia a partir de (ROBOTICS, n.d.-c)

Por último, se hizo más hincapié en las dimensiones y detalles presentes en el submarino, llegando al prototipo final.

Ilustración 17: Prototipo final y despiece



Fuente: elaboración propia

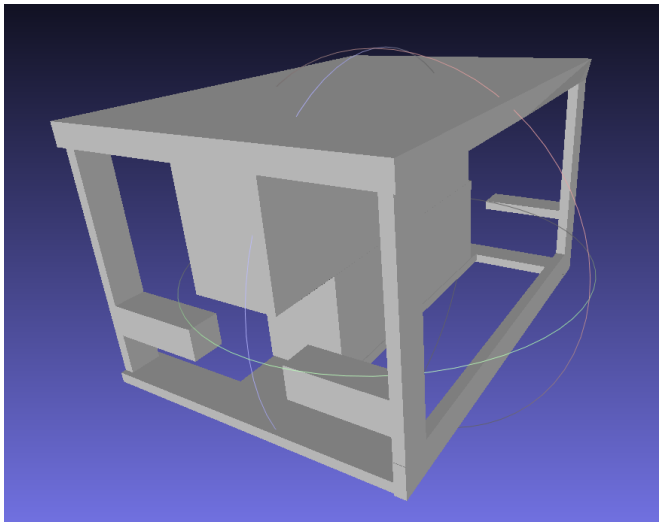
Una vez realizado el modelo 3D se ha trasladado a la herramienta Meshlab, en la que se han hecho los arreglos convenientes: color, bordes, así como la unión final de piezas.

3.3 Características físicas del modelo

Por último, además de la estética, se considera la necesidad de un modelo físico del robot a tratar, describiendo la matriz de inercias, de arrastre y coeficientes hidrodinámicos.

Dada la multitud de piezas, así como las geometrías complejas que posee el submarino, realizar el cálculo de todas estas magnitudes físicas supone un problema difícil de resolver Para solventar este problema, se va a optar por un modelo simplificado del submarino.

Ilustración 18: Modelo 3D simplificado



Fuente: Elaboración propia

Dado un objeto 3D cerrado, Meshlab es capaz de calcular la matriz de inercias, centro de gravedad, baricentro, volumen, área exterior y dimensiones. Los resultados para el submarino y el propulsor se pueden ver en los apéndices 6.1 y 6.2 respectivamente.

Debido a la simetría del submarino, los centros de gravedad y flotación se deben encontrar en el eje z.

Enrique González (Enrique González Sancho, 2018), en su TFG, utiliza un modelo sólido de densidad homogénea para calcular el centro de flotación. Dadas las características de Meshlab, no se permite realizar el cálculo del centro de gravedad utilizando densidades heterogéneas. Luego, se considerará el centro de flotación, la coordenada 14 cm (ubicación del centro de gravedad según meshlab).

La posición absoluta del centro de gravedad se estimará a partir de la posición relativa respecto al de flotación en submarinos parecidos como el BlueROV2 (Einarsson & Lipenitis, 2020) y Sibiu Nano (Enrique González Sancho, 2018), aproximadamente 3 cm por debajo del de flotación.

Se toma como origen de coordenadas del modelo 3D del submarino, el centro de gravedad.

Para el cálculo de la masa se ha tenido en cuenta el estudio realizado por Enrique González (Enrique González Sancho, 2018) en el que se realiza un análisis de la relación masa/dimensiones de los submarinos más comerciales, llegando a la conclusión de que dado un cubo con las dimensiones del submarino, la masa del submarino se puede estimar como la masa de ese cubo con una densidad de 270 kg/m^3.

El cálculo de las matrices de masa agregada y amortiguamiento lineal y cuadrático es complejo, y requiere un trabajo de CFD que no forma parte de los objetivos de este TFG. Dicho esto, como estimación, se van a tomar los valores de un submarino con perfiles parecidos. En específico el BlueROV2 cuyos valores se describen en el Trabajo de Fin de Master de Emil Már Einarsson y Andris Lipenitis (Einarsson & Lipenitis, 2020).

Obteniendo así las características físicas del modelo necesarias para la simulación resumidas en la siguiente tabla.

Tabla 4: Resumen Modelo Físico Sibiu Nano Plus

Dato	**Valor**
Centro de gravedad	$\begin{bmatrix}0 & 0 & 0\end{bmatrix} m$
Centro de flotación	$\begin{bmatrix}0 & 0 & 0.03\end{bmatrix} m$
Tensor de inercia del cuerpo	$\begin{bmatrix}3.1120128 & 0.0018816 & 0.0363264 \\ 0.0018816 & 1.6492608 & 0.0322944 \\ 0.0363264 & 0.0322944 & 2.7139968\end{bmatrix} [kg \cdot m^2]$
Volumen	8.705 litros
Masa	8’1648 kg
Propulsión máxima	30 N
Dimensiones	$\begin{bmatrix}42 & 30 & 24\end{bmatrix}$ cm [largo, ancho, alto]
Masa agregada	$\begin{bmatrix}25 & 0 & 0 & 0 & 0 & 0 \\ 0 & 25 & 0 & 0 & 0 & 0 \\ 0 & 0 & 17.87 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 17.87 & 0 \\ 0 & 0 & 0 & 0 & 0 & 6.7\end{bmatrix} \begin{bmatrix} kg \\ \frac{kgm^2}{rad}\end{bmatrix}$
Coeficientes lineales de Arrastre	$\begin{bmatrix}-25.15 & -7.364 & -17.955 & -1.888 & -0.761 & -3.744\end{bmatrix} \frac{Ns}{m}$
Coeficientes cuadráticos de arrastre	$\begin{bmatrix}-17.77 & -125.9 & -72.36 & -0.1246 & -0.1246 & -0.1857\end{bmatrix} \frac{Ns^2}{m^2}$
Tensor de inercia de los propulsores	$\begin{bmatrix}0.00007538 & 0 & 0 \\ 0 & 0.00008859 & 0 \\ 0 & 0 & 0.000047955\end{bmatrix} [kg \cdot m^2]$
Coeficiente propulsión	0. 0.0012081 N/W

Fuente: Elaboración a partir de datos de (Enrique González Sancho, 2018), (Einarsson & Lipenitis, 2020) y propios

4 SIMULACIÓN Y CONTROL

Dadas la descripción del modelo del Sibiu Nano Plus y las necesidades de control y simulación de este TFG se decide utilizar como simulador UUVSim.

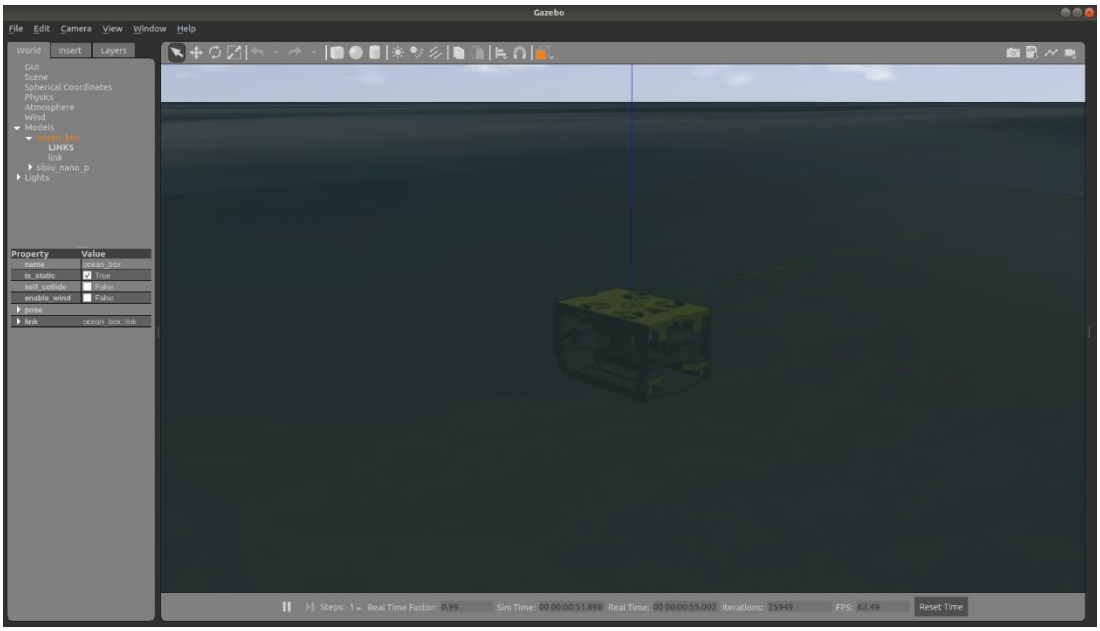
Mediate simulación se logra una representación del comportamiento del modelo del submarino, permitiendo conocer el efecto debido a perturbaciones y fuerzas externas en mayor detalle, así como realizar análisis del sistema de control.

4.1 Simulación

4.1.1 Entorno de simulación

Como se ha indicado anteriormente, UUVSim es un simulador basado en Gazebo, utilizado para la simulación de vehículos subacuáticos. Una vez iniciado el simulador aparece un mundo sumergido, en el que podemos hacer aparecer el modelo del submarino como se muestra en la ilustración 19.

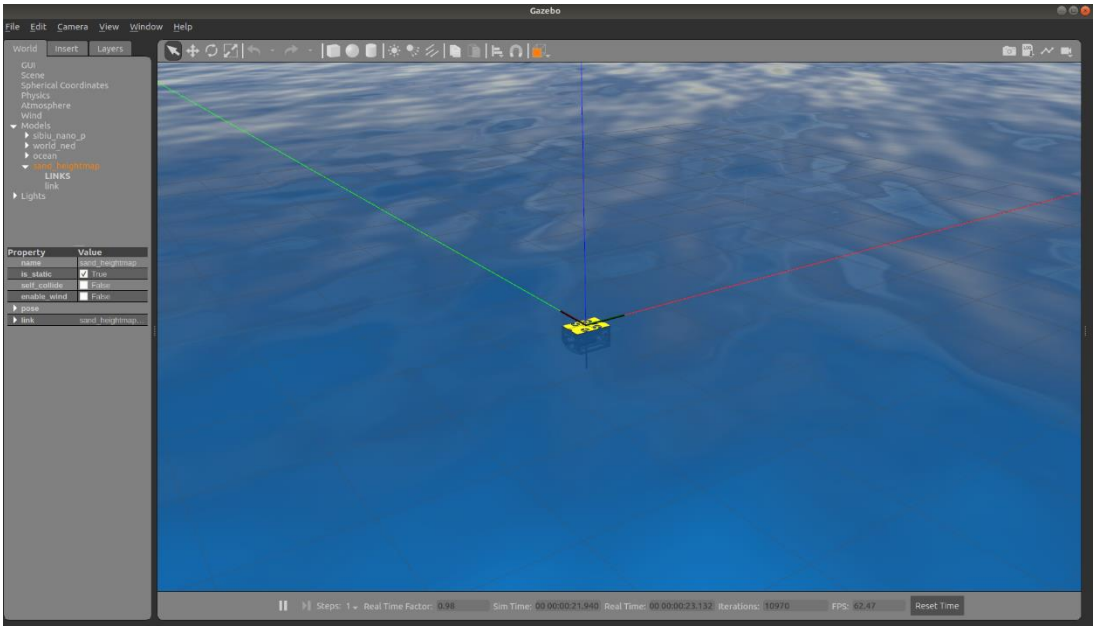
Ilustración 19: UUVSim mundo por defecto



Fuente: Elaboración propia

El simulador, nos ofrece la posibilidad de crear distintos mundos, algunos de ellos con efecto de oleaje como se observa en la ilustración 20. Sin embargo, este efecto no produce ninguna fuerza en el submarino, siendo simplemente visual.

Ilustración 20: UUVSim Mundo con olas



Fuente: Elaboración propia

4.1.2 Descripción del modelo

Para describir nuestro modelo se hará uso del formato URDF, muy utilizado en ROS para realizar descripciones físicas de robots. Mediante lenguaje XML se realiza una división por bloques del cuerpo, actuadores y sensores del submarino, y se crean las conexiones necesarias entre ellos. Esta descripción puede observarse en el apéndice 6.3.

Para comenzar, se crea el primer enlace del submarino, formado por el cuerpo, realizando una descripción completa de las características físicas del submarino como la masa, centro de gravedad, matriz de inercia y espacio de colisión. Además, se definirán una serie de parámetros a utilizar por el simulador en el cálculo de fuerzas, como son el centro de flotabilidad, volumen, dimensiones y modelo hidrodinámico. Adicionalmente se incluye el modelo 3D, creado anteriormente, para darle un aspecto visual.

Al mismo tiempo, se creará una macro de las características de los propulsores. En esta se describirán las características básicas de inercia y masa, y se añadirá el modelo visual. Además, se describirán parámetros a utilizar por el simulador como son el tipo de respuesta dinámica, saturación de la señal de entrada, eficiencia y relación entre la señal de entrada y el empuje realizado.

A continuación, se definirá la posición de los propulsores y de los sensores, así como el tipo de sensores a incluir en el modelo.

UUVSim y Gazebo incluyen la gran mayoría de sensores normalmente presentes en un submarino, pero permiten la posibilidad de describir nuevos sensores y utilizar paquetes de apoyo al simulador.

4.1.3 Interacciones

Al estar basado en ROS, el simulador proporciona una serie de topics y servicios que permiten modificar en tiempo real el comportamiento del entorno, el modelo del submarino e incluso las leyes físicas que rigen el movimiento.

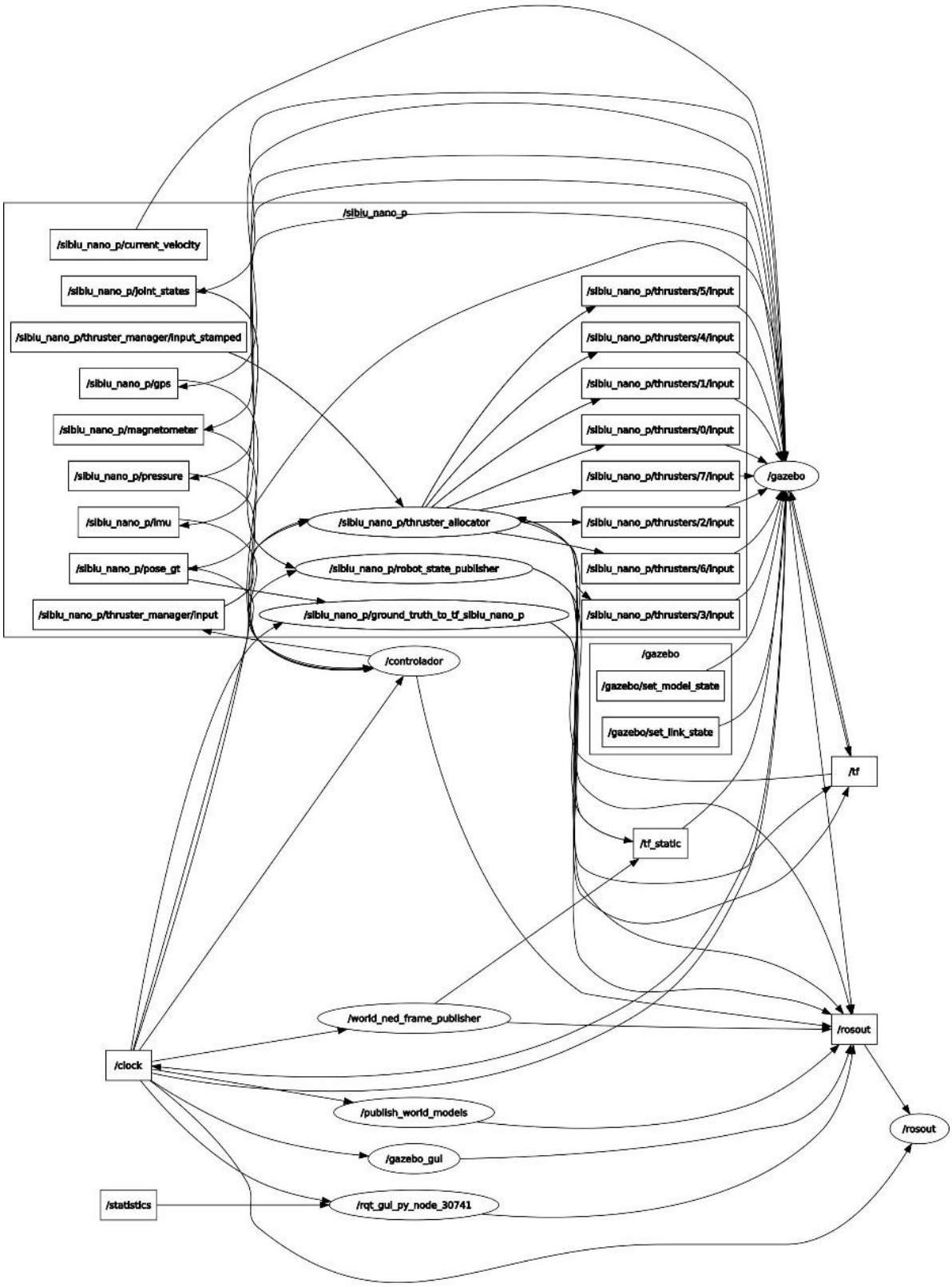
Entre otras interacciones resulta interesante destacar:

- Aplicar fuerzas describiendo, dirección, sentido, magnitud y duración.

- Pausar el tiempo.
- Realizar transformaciones de coordenadas.
- Consultar los valores de amortiguamiento y masa agregada afectando al submarino en un instante de tiempo.
- Modificar la eficiencia de los propulsores
- Desactivar propulsores (simulando bloqueos)
- Leer datos de sensores
- Aplicar señales a los actuadores.

El paquete de ROS 'rqt_graph' permite obtener un grafo de los procesos en ejecución, permitiendo observar en detalle los módulos que participan en la simulación y el control, así como la comunicación entre ellos. En la figura 23 aparece el grafo asociado a este modelo de simulación.

Figura 23: Grafo de simulación



Fuente: Elaboración propia

4.2 Control

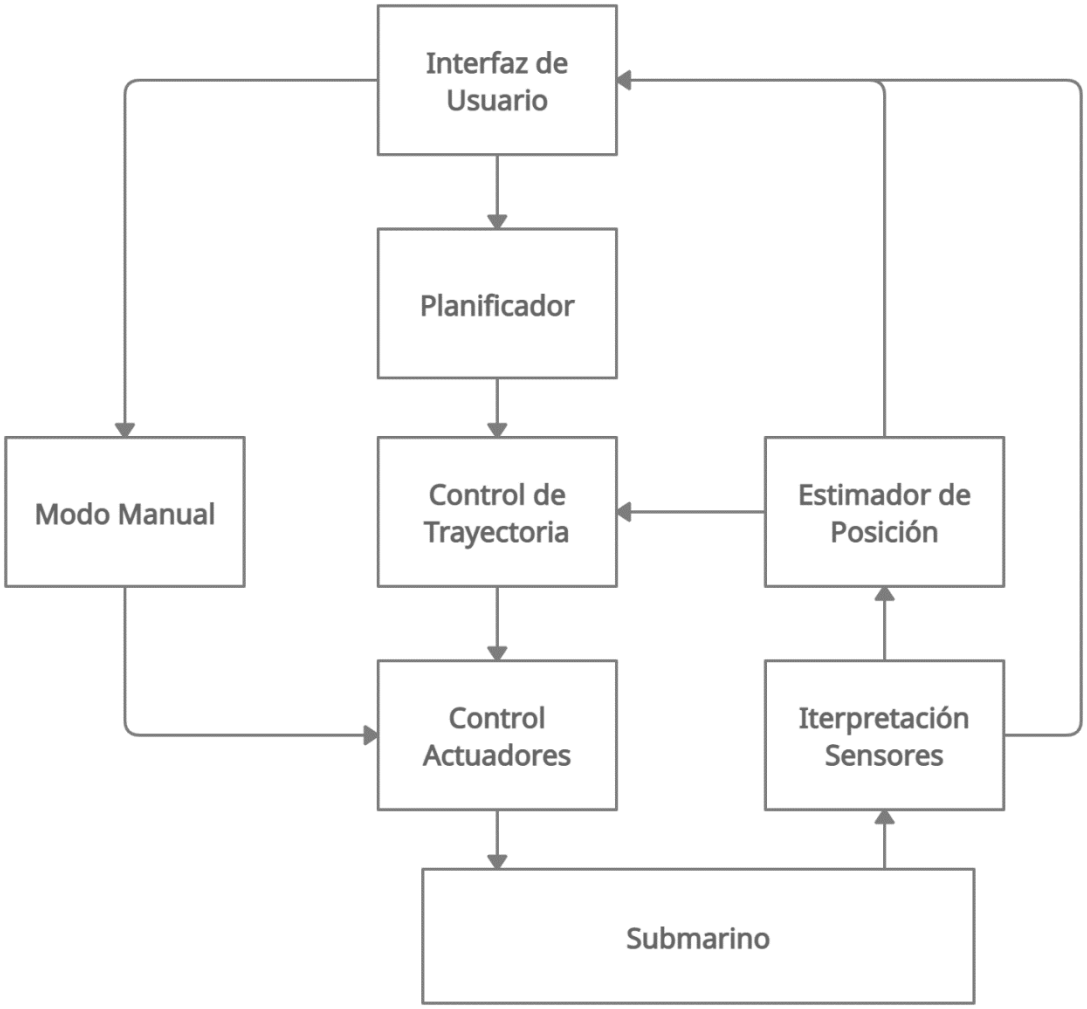
Una vez definidas todas las características del submarino, así como el entorno de simulación, se procede al diseño del control. Para cumplir objetivos, el submarino debe ser capaz de desplazarse desde un punto y orientación de origen a otro de destino.

El submarino debe ser capaz de localizar esa posición en el espacio respecto a su propio sistema de referencia, elaborar un camino hacia ella mediante una serie de puntos objetivo (waypoints) y desplazarse por ellos hasta la posición final.

4.2.1 Estructura del control

El modelo de control del submarino sigue la siguiente estructura:

Figura 24: Arquitectura de control



Fuente: Elaboración propia

Analizando cada uno de los bloques:

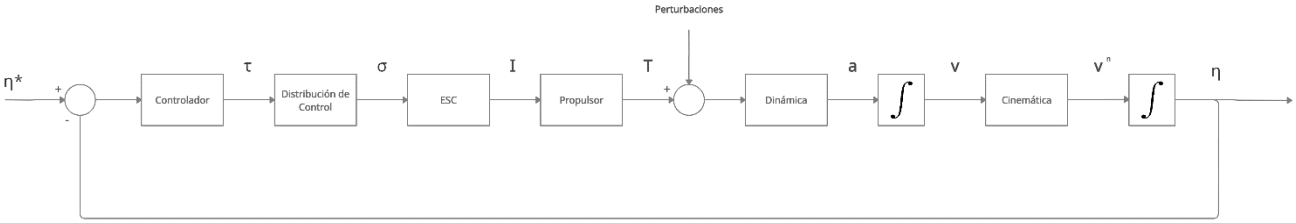
- Interfaz de Usuario: En este caso la interfaz de ROS. Mediante servicios se permite enviar las órdenes y consultar topics de los que poder extraer las lecturas de los sensores. Para una

mejora visual, se tiene el visualizador de la simulación (gazebo) y un programa que imprime las lecturas en gráficas para su posterior análisis.

- Planificador: Mediante una simplificación se opta por una estrategia simple, sin evitación de obstáculos, en la que se crea una lista de puntos interpolando entre la posición actual y la posición de destino.
- Control de trayectoria: Este es el controlador principal. Formado por 3 PID para la altura, orientación (viraje) y posición horizontal, y 2 PD para la regulación de los movimientos de arfada (pitch) y balance (roll).
- Control de actuadores: Realiza la distribución del control, recibiendo como entrada la fuerza a efectuar en cada uno de los ejes, y como salida, la potencia a entregar a cada propulsor.
- Submarino: En este caso, el modelo diseñado del Sibiu Nano Plus, cuyo comportamiento viene dado por el simulador.
- Interpretación de los sensores: Realiza el trabajo de los Drivers, obteniendo las lecturas necesarias de los sensores.
- Estimador de posición: A través de la lectura de los sensores, se realiza una estimación del estado actual del vehículo.
- Modo manual: Permite el control del submarino a través de la interfaz de usuario.

Desde un punto de vista de la realimentación del control, se puede definir el sistema, como aparece en la figura 25.

Figura 25: Esquema de control



Fuente: Elaboración Propia

4.2.2 Distribución de control

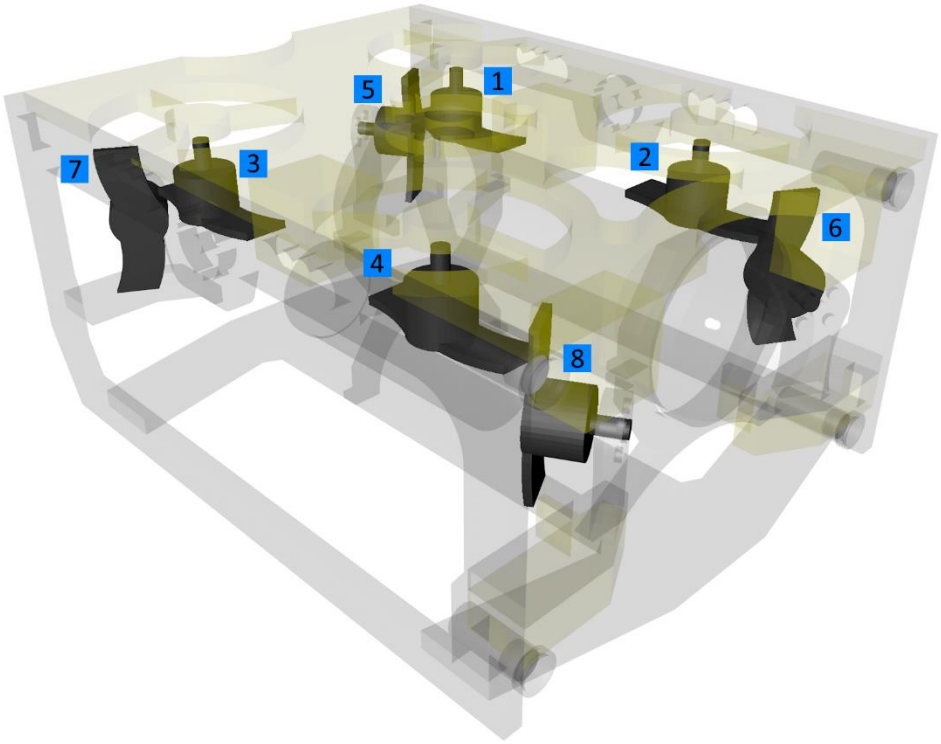
Como se ha descrito anteriormente, un submarino dispone de 6 grados de libertad (**x**,**y**,**z**, **φ**, **θ**, **ψ**) y el tipo de control a realizar será a partir de las fuerzas y momentos.

El simulador provee de un gestor de propulsores que realiza los algoritmos de optimización y distribución de control, aun así, resulta necesario especificar la matriz de pesos de los propulsores.

Dado el modelo, con la posición y orientación de los propulsores, resulta sencillo hacer una representación de la fuerzas y momentos realizadas por cada propulsor en cada eje del submarino.

Esta descripción puede observarse en el apéndice 6.7.

Ilustración 21: Localización de los propulsores



Fuente: Elaboración propia

Siendo los propulsores verticales 1-4, y los horizontales 5-8.

$$F_X^{sub} = \left(F^{P5} + F^{P7} - F^{P6} - F^{P8}\right) \cdot \cos(\alpha)$$

$$F_Y^{sub} = \left(F^{P5} + F^{P6} - F^{P7} - F^{P8}\right) \cdot \sin(\alpha)$$

$$F_Z^{sub} = F^{P1} + F^{P2} + F^{P3} + F^{P4}$$

$$M_X^{sub} = \left(F^{P3} + F^{P4} - F^{P1} - F^{P2}\right) \cdot d1 + \left(F^{P7} + F^{P8} - F^{P5} - F^{P6}\right) \cdot d2 \cdot \sin(\alpha)$$ [1]

$$M_Y^{sub} = \left(F^{P2} + F^{P4} - F^{P1} - F^{P3}\right) \cdot d3 + \left(F^{P6} + F^{P8} - F^{P5} - F^{P7}\right) \cdot d4 \cdot \cos(\alpha)$$

$$M_Z^{sub} = \left(F^{P5} + F^{P8} - F^{P6} - F^{P7}\right) \cdot d5$$

Fuente: elaboración propia

Esta distribución de control aplica un control PID a cada propulsor para mantener el empuje deseado a la salida.

4.2.3 Controladores

El control se ha realizado mediante controladores PID. Los movimientos del submarino se han dividido en regulación de los movimientos de balance y arfada, altura, orientación y posición horizontal; analizando cada uno de ellos por separado.

Para el cálculo de los parámetros de los PID se ha utilizado un método heurístico mediante pruebas de

[1] Se denomina α al ángulo formado entre los propulsores y la longitudinal, como indicado anteriormente 45º y características de diseño d1= 0.07m d2=0.201m d3=0.095m d4=0.07m d5=0.2012m

simulación. Se ha elegido valores de constantes proporcionales y derivativas relativamente altos para obtener una respuesta rápida ante nuevas posiciones y añadir un efecto de amortiguación al movimiento del vehículo. Además, se ha elegido una pequeña componente integral para eliminar el error en régimen permanente.

El programa de control se encuentra en el apéndice 6.4

4.2.3.1 Regulación balanceo y arfada

Pese a que uno de los factores clave en el diseño de submarinos es la estabilidad, en muchos casos se dan situaciones en las que las fuerzas externas o de propulsión (en caso de saturación) generan perturbaciones en el movimiento que pueden llegar a producir cambios importantes en los ejes de balanceo y arfada, afectando al resto de sistemas de control.

Con el objetivo de reducir las aceleraciones y mantener la estabilidad del submarino, suele introducirse un control para regular estos movimientos.

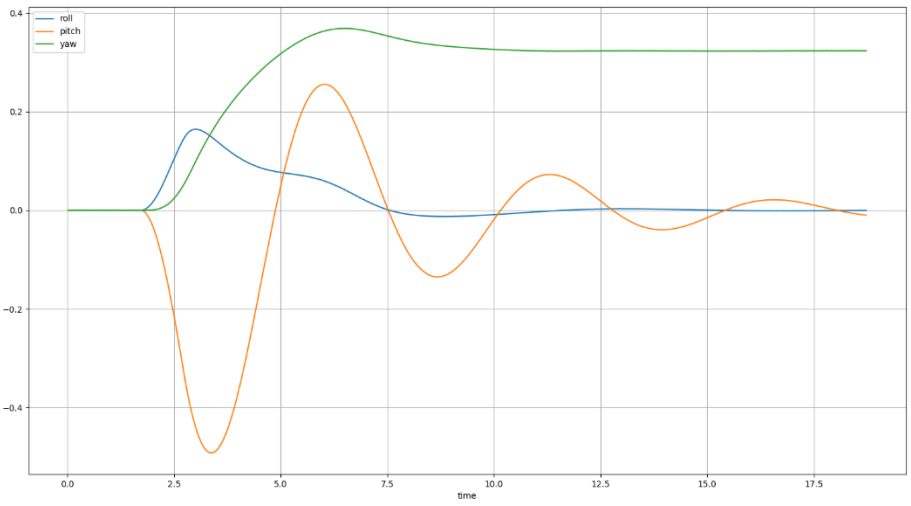
En el caso del Sibiu Nano Plus se ha decidido adoptar una estrategia PD, utilizando uno por cada eje con parámetros $\left[k_p = 5 \quad k_i = 0.0 \quad k_d = 150 \quad sat = 1500 \right]$. La realimentación se ha realizado a través de la IMU definida en el modelo.

Para comprobar su funcionamiento, se ha sometido al submarino a un momento de 2 Nm durante un segundo. Los resultados del comportamiento del submarino, con y sin aplicación de regulación, se pueden apreciar en las figuras 22 y 23.

Ya que el control se encuentra diseñado sobre los ejes del submarino, se requiere de este control para mantener la estabilidad del sistema. Para el diseño del resto de controladores se ha utilizado este como apoyo.

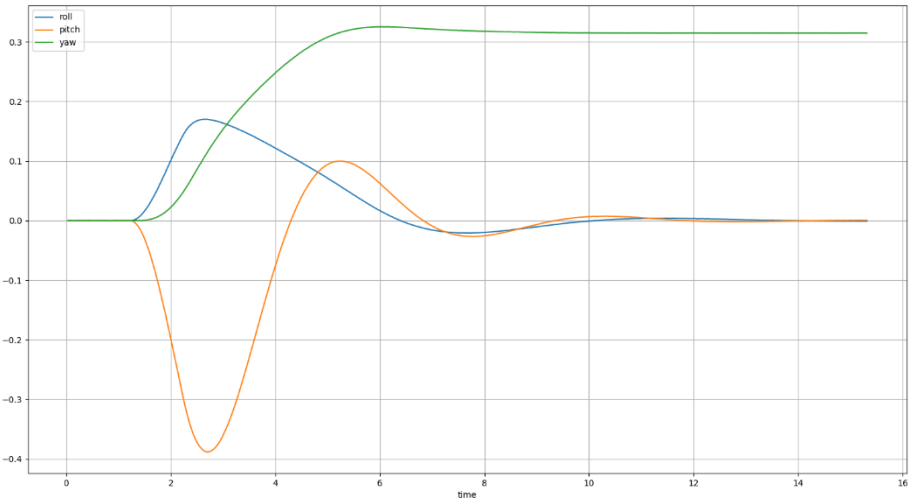
Por último, en la figura 26 se puede apreciar el consumo de cada propulsor al realizar la regulación descrita en la figura 27.

Figura 26: Efectos balance y arfada ante fuerzas sin regulación



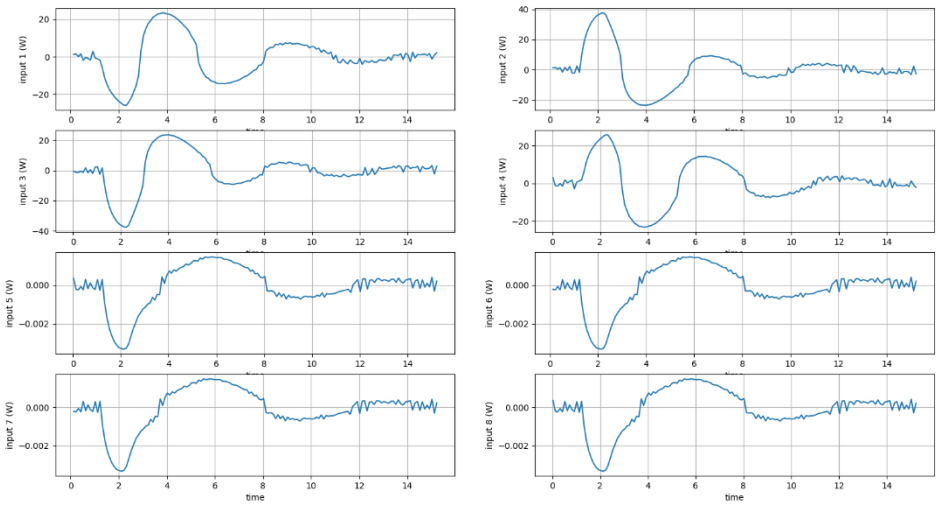
Fuente: Elaboración propia

Figura 27: Efectos balance y arfada ante fuerzas aplicando regulación



Fuente: Elaboración propia

Figura 28: Consumo debido a la regulación



Fuente: Elaboración propia

4.2.3.2 Flotación / profundidad

El control de flotación o profundidad regula el eje z del submarino. Como se ha indicado en el punto 2.4.2 del estado del arte, los ROV se caracterizan por poseer una fuerza de flotación mayor a la de la gravedad, permitiendo así su extracción en caso de avería o falta de batería. Para mantener la profundidad, es necesario contrarrestar dicha fuerza de flotación mediante control.

Este control se ha configurado mediante un PID con parámetros $\left[k_p = 620 \quad k_i = 7 \quad k_d = 320 \quad sat = 6500\right]$, La realimentación de la altura se calcula a partir del sensor de presión presente en el submarino.

El comportamiento de control se puede observar en las tablas 5-7 y las figuras 29-31

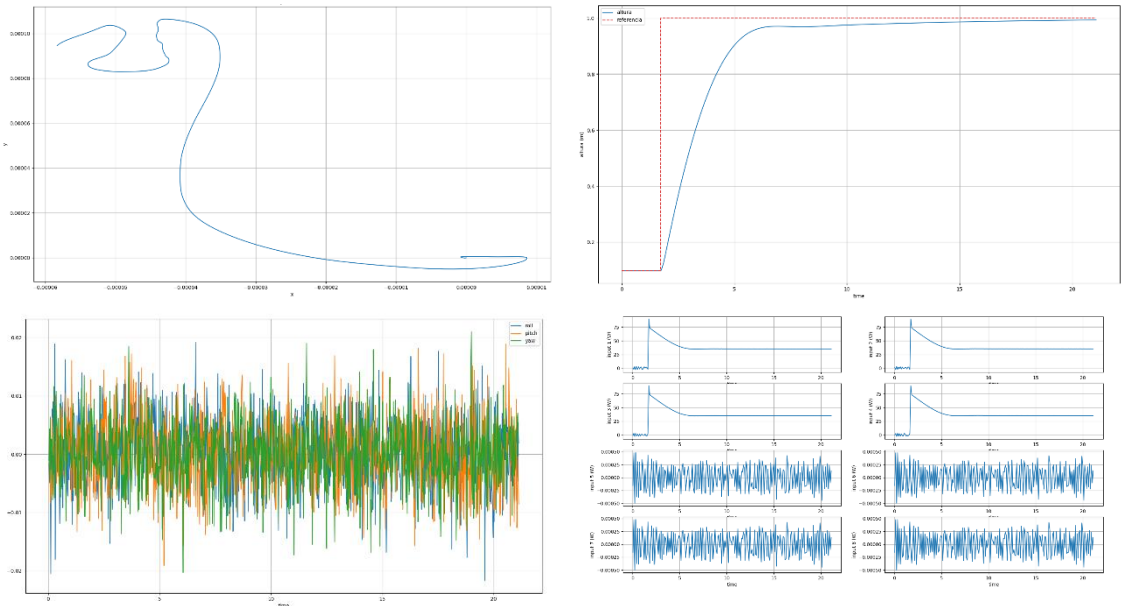
Tabla 5: Resultado 1 del comportamiento del control de profundidad

Punto inicial	0.098 m
Referencia	1 m
Sobreoscilación	0%
Tiempo de respuesta (68%)	1.87 s
Tiempo de subida (95%)	4.09 s
Variación máxima de balanceo	0.02º
Variación máxima de arfada	0.02º

Fuente: Elaboración propia

Figura 29: Resultado 1 del comportamiento del control de profundidad

(Las gráficas describen posición, profundidad, orientación y consumo).



Fuente: Elaboración propia

Tabla 6: Resultado 2 del comportamiento del control de profundidad

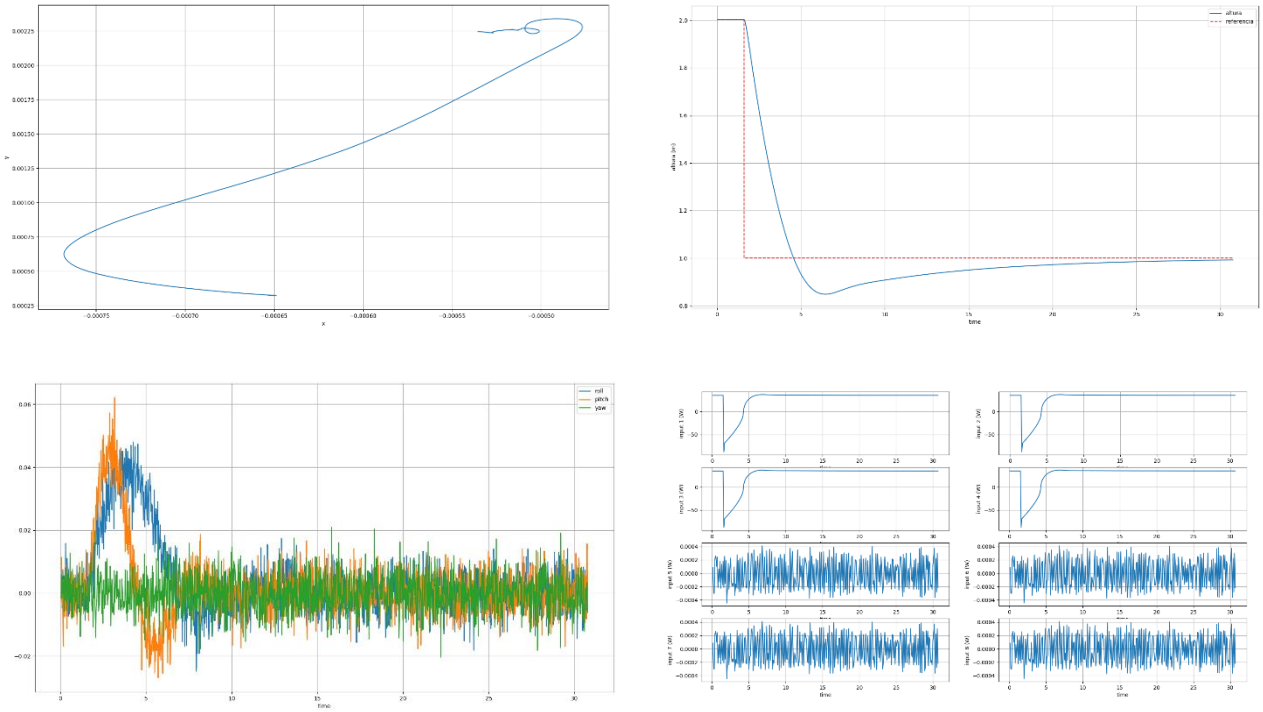
Punto inicial	2 m
Referencia	1 m
Sobreoscilación	15.2%
Tiempo de respuesta (68%)	1.87 s

Tiempo de subida (95%)	4.09 s
Tiempo de establecimiento (5%)	13.4 s
Variación máxima de balanceo	0.045º
Variación máxima de arfada	0.06º

Fuente: Elaboración propia

Figura 30: Resultado 2 del comportamiento del control de profundidad

(Las gráficas describen posición, profundidad, orientación y consumo).



Fuente: Elaboración propia

Tabla 7: Resultado 3 del comportamiento del control de profundidad

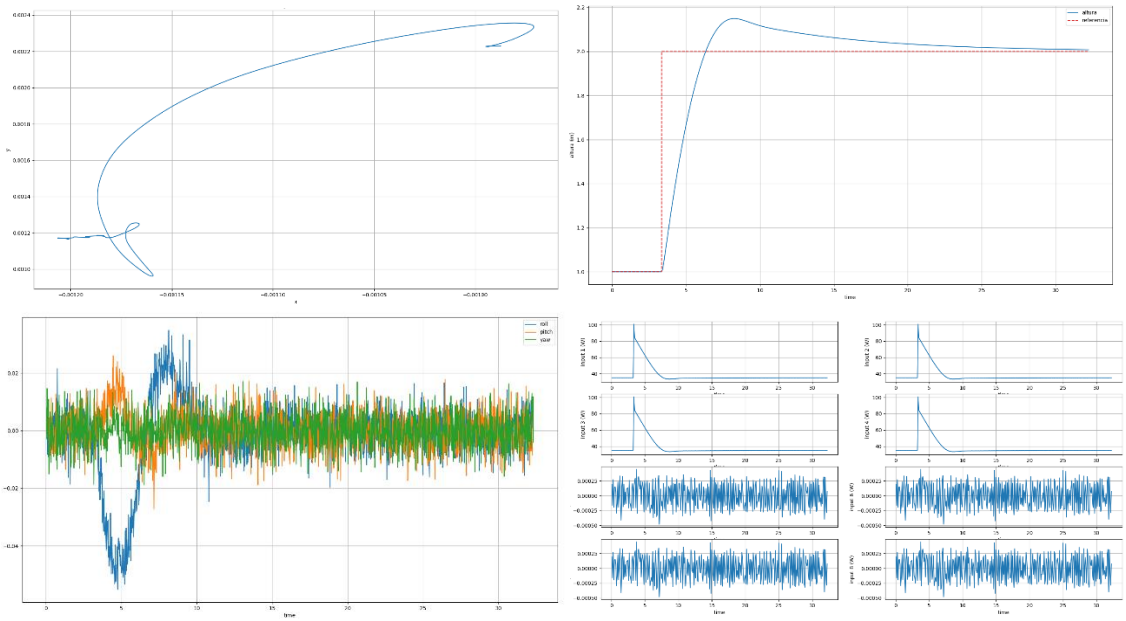
Punto inicial	1 m
Referencia	2 m
Sobreoscilación	14.7%
Tiempo de respuesta (68%)	1.7 s
Tiempo de subida (95%)	2.72 s
Tiempo de establecimiento	13.45 s
Variación máxima de balanceo	0.05º

Variación máxima de arfada	0.02º

Fuente: Elaboración propia

Figura 31: Resultado 3 del comportamiento del control de profundidad

(Las gráficas describen posición, profundidad, orientación y consumo).



Fuente: Elaboración propia

4.2.3.3 Orientación

Entre los objetivos de control se encuentra virar el submarino hacia una orientación deseada.

Para este control se utiliza un PID realimentado mediante la IMU y el magnetómetro, cuyos parámetros son $\left[k_p = 23 \quad k_i = 0.005 \quad k_d = 11 \quad sat = 1500\right]$. Su desempeño puede observarse en las tablas 8 y 9 y las figuras 32 y 33.

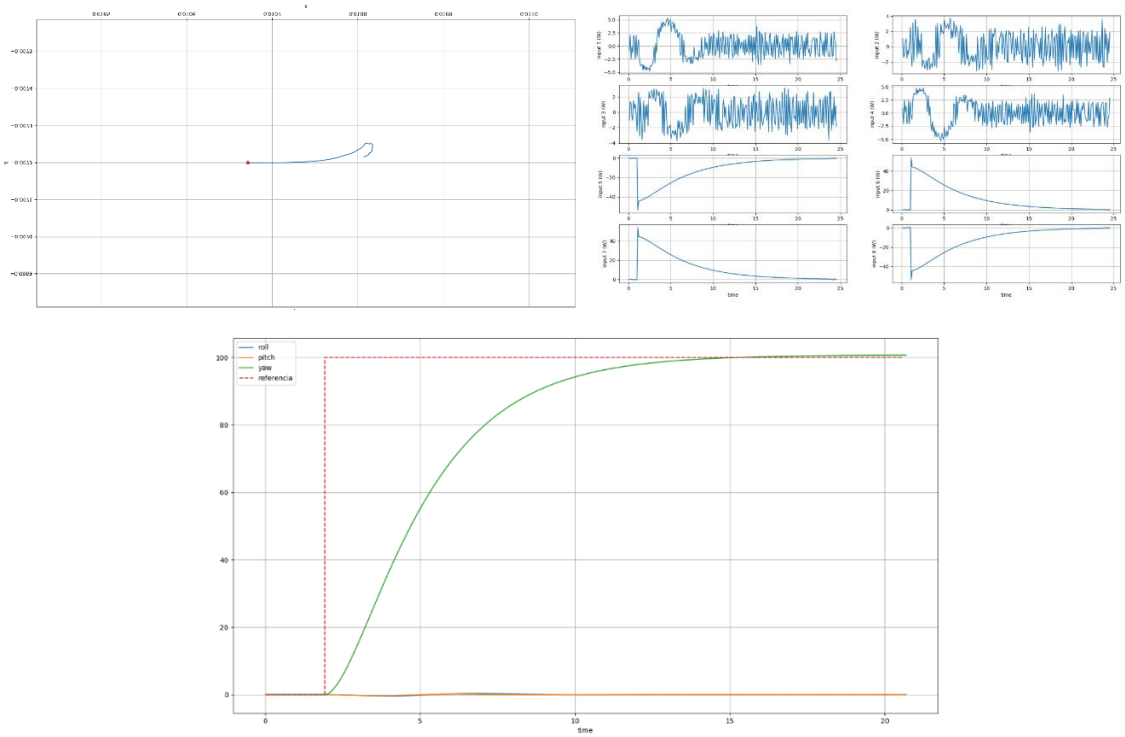
Tabla 8: Resultado 1 del comportamiento del control de orientación

Punto inicial	0º
Referencia	100º
Sobreoscilación	0.9%
Tiempo de respuesta (68%)	3.96 s
Tiempo de subida (95%)	10.02 s
Variación máxima de balanceo	0.36º
Variación máxima de arfada	0.18º

Fuente: Elaboración Propia

Figura 32: Resultado 1 del comportamiento del control de orientación

(Las gráficas describen posición, consumo y orientación).



Fuente: Elaboración propia

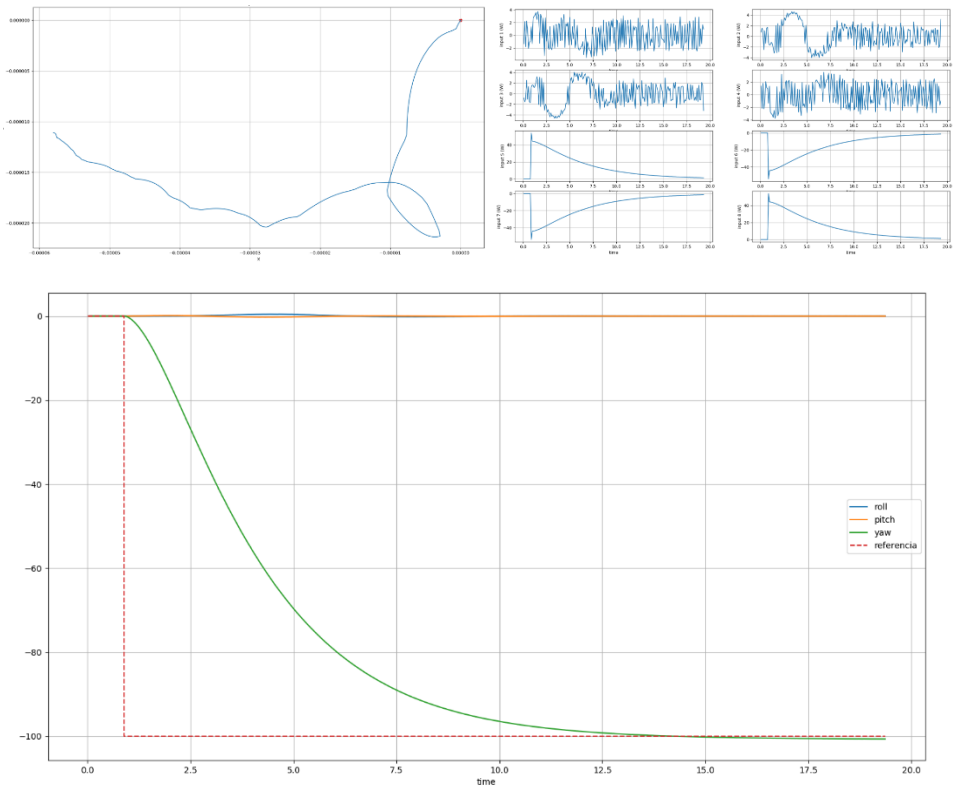
Tabla 9: Resultado 2 del comportamiento del control de orientación

Punto inicial	0º
Referencia	-100º
Sobreoscilación	0.7%
Tiempo de respuesta (68%)	3.98 s
Tiempo de subida (95%)	8.39 s
Variación máxima de balanceo	0.45º
Variación máxima de arfada	0.24º

Fuente: Elaboración Propia

Figura 33: Resultado 2 del comportamiento del control de orientación

(Las gráficas describen posición, consumo y orientación).



Fuente: Elaboración propia

4.2.3.4 Movimiento horizontal

Para terminar el objetivo de control es necesario alcanzar una posición destino; para ello se utiliza un control de movimientos en el plano horizontal.

Como se ha mencionado anteriormente, la posición del submarino se podría estimar mediante un filtro de Kalman extendido, teniendo en cuenta la doble integración de las aceleraciones y los datos recibidos por el resto de sensores. Sin embargo, para facilitar la fase de diseño del control, se ha utilizado el topic 'Odom' de Gazebo, que facilita la posición exacta del robot.

Para este control es necesario realizar una transformación de los ejes absolutos a los ejes del submarino utilizando los datos de orientación. Consiste en un PID con parámetros $\left[k_p = 180 \quad k_i = 0.01 \quad k_d = 20 \quad sat = 140\right]$, que ofrece la respuesta apreciable en las tablas 10-12 y las figuras 34-36.

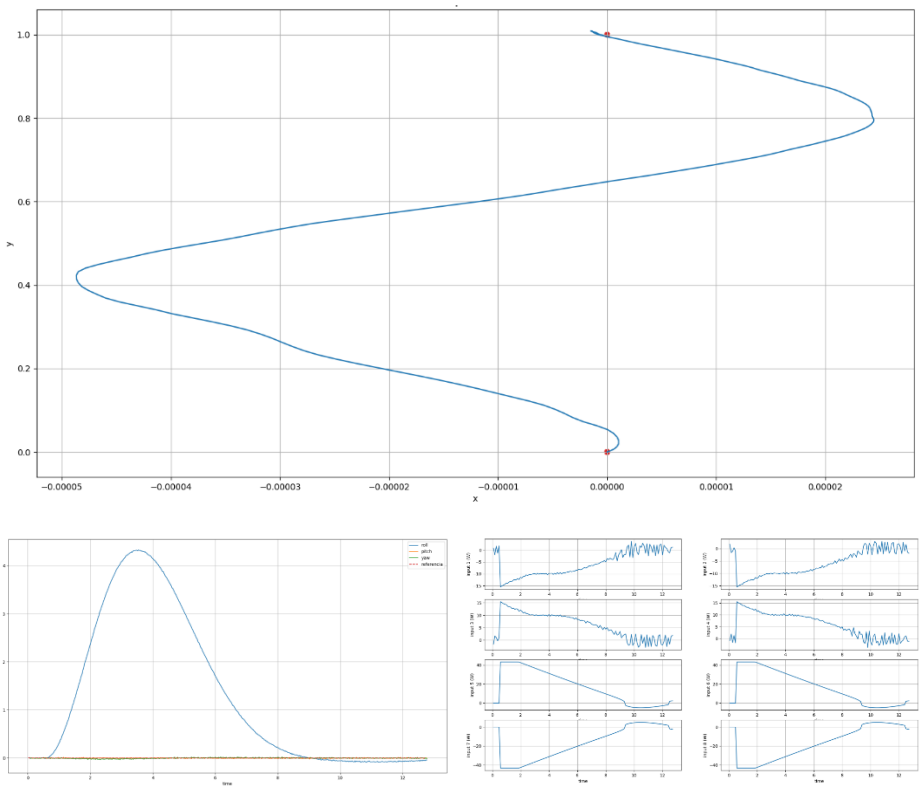
Tabla 10: Resultado 1 del comportamiento del control horizontal

Punto inicial	[0, 0]
Referencia	[0, 1]
Sobreoscilación	0.08 %
Tiempo de llegada	9.12 s
Variación máxima de balanceo	4.32º
Variación máxima de arfada	0.03º

Fuente: Elaboración Propia

Figura 34: Resultado 1 del comportamiento del control horizontal

(Las gráficas describen posición, orientación y consumo).



Fuente: Elaboración propia

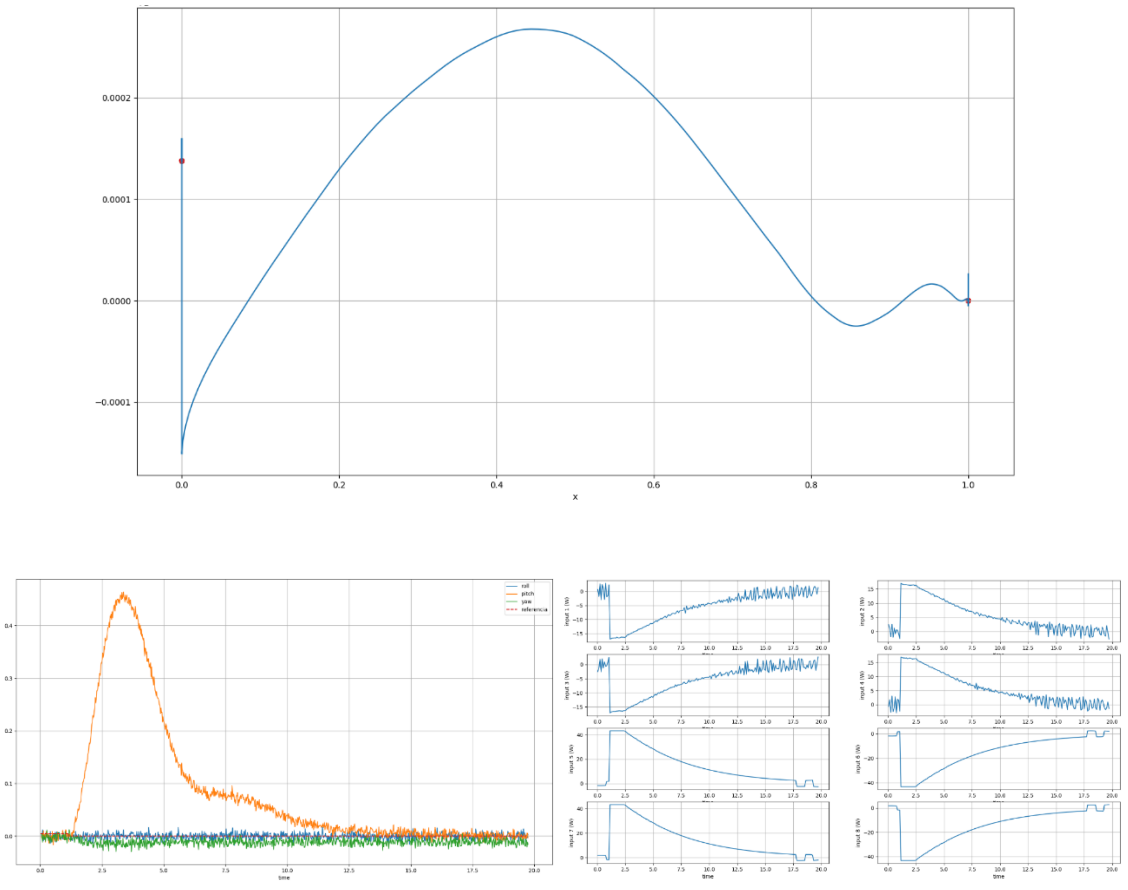
Tabla 11: Resultado 2 del comportamiento del control horizontal

Punto inicial	[0, 1]
Referencia	[1, 1]
Sobreoscilación	0.002 %
Tiempo de llegada	16.75s
Variación máxima de balanceo	0º
Variación máxima de arfada	0.46º

Fuente: Elaboración Propia

Figura 34: Resultado 2 del comportamiento del control horizontal

(Las gráficas describen posición, orientación y consumo).



Fuente: Elaboración propia

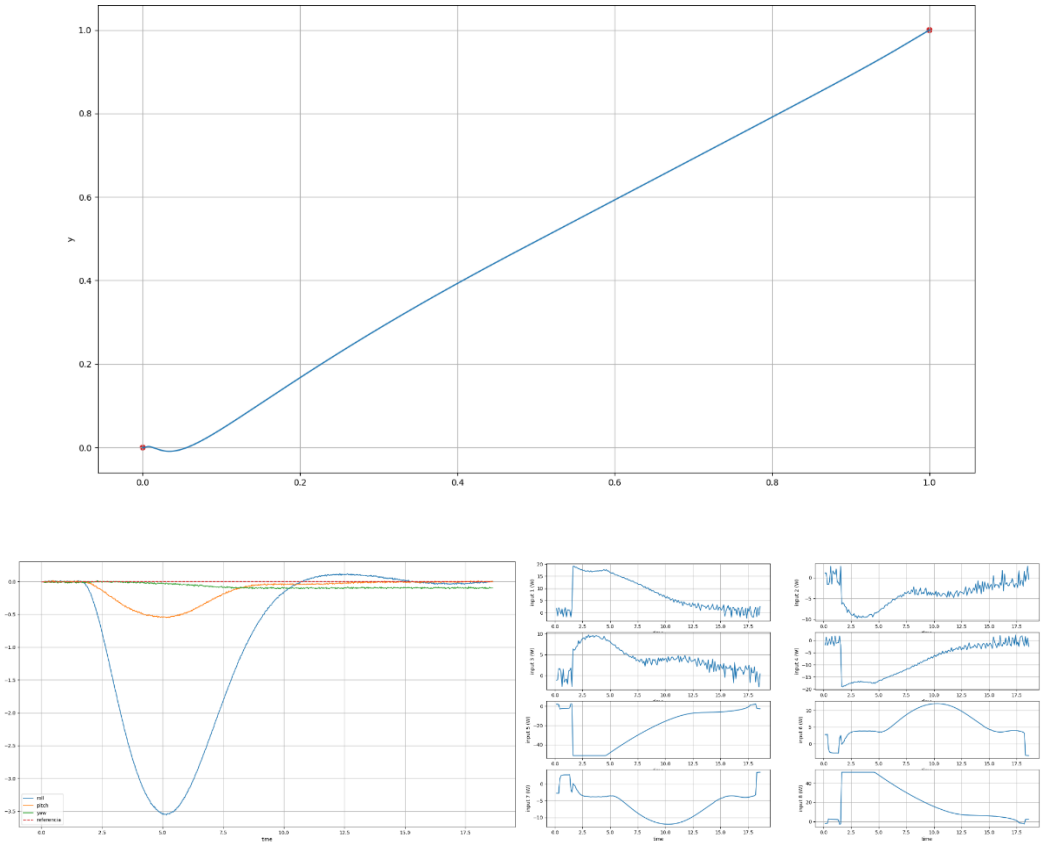
Tabla 12: Resultado 3 del comportamiento del control horizontal

Punto inicial	[1, 1]
Referencia	[0, 0]
Sobreoscilación	[0.05, 0.8]%
Tiempo de llegada	16.77s
Variación máxima de balanceo	3.55º
Variación máxima de arfada	0.54º

Fuente: Elaboración Propia

Figura 35: Resultado 3 del comportamiento del control horizontal

(Las gráficas describen posición, orientación y consumo).



Fuente: Elaboración propia

4.2.4 Planificación

La planificación se ha realizado mediante interpolación de puntos. Se establece como parámetro la distancia de interpolación, una vez establecida, se traza una línea recta entre el punto origen y el punto objetivo de cada uno de los ejes, y se toma una lista de puntos sobre esta recta separados entre ellos la distancia de interpolación. Los puntos objetivo se comunican al controlador mediante servicios de ROS.

El programa que describe el planificador se puede ver en el apéndice 6.5

4.3 Resultados

Una vez diseñado y analizado el control, se procede a realizar pruebas del sistema completo. Estas se reducen a las descritas en las tablas 13 y 14, y figuras 36-39.

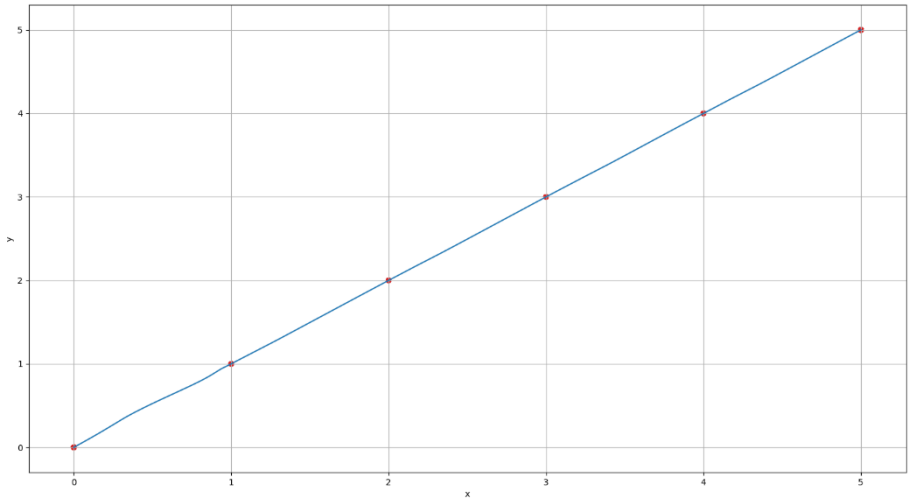
Tabla 13: Resultado 1 del sistema completo

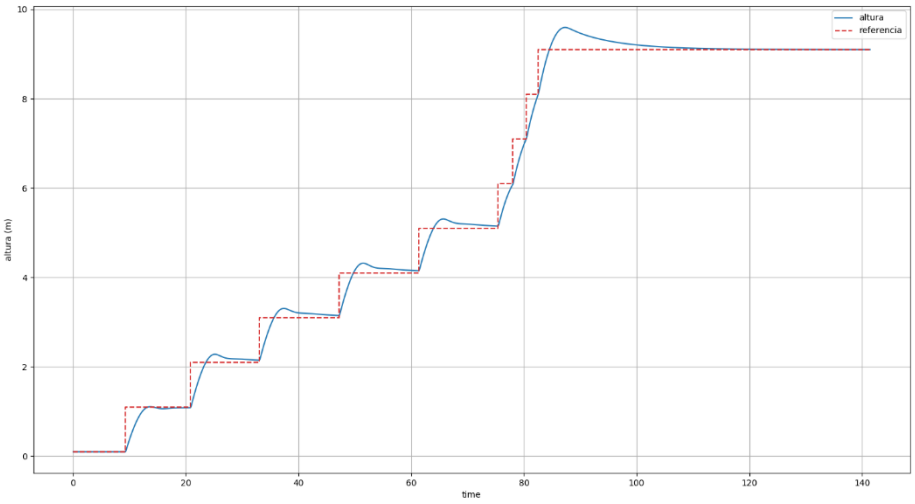
Punto inicial	[0, 0, 0, 0º]
Referencia	[5, 5, 10, 45º]
Tiempo de llegada	85.87 s
Variación máxima de balanceo	0.4º
Variación máxima de arfada	17.2º

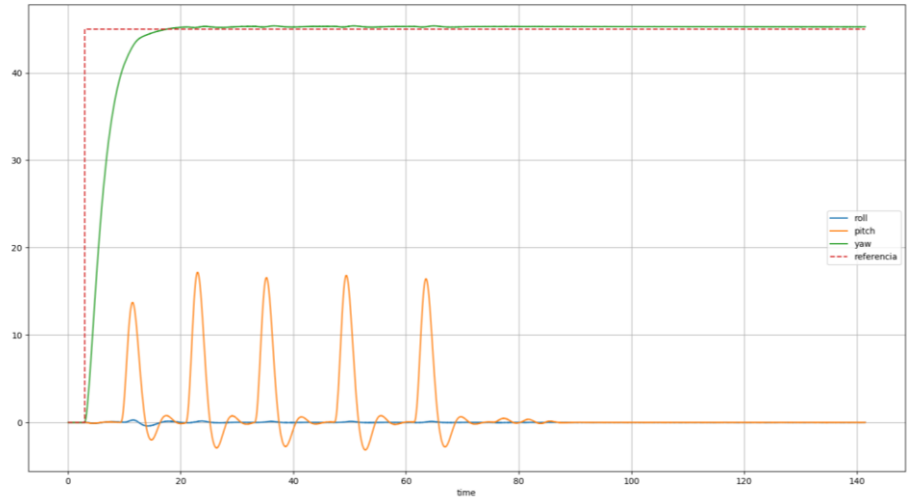
Fuente: Elaboración Propia

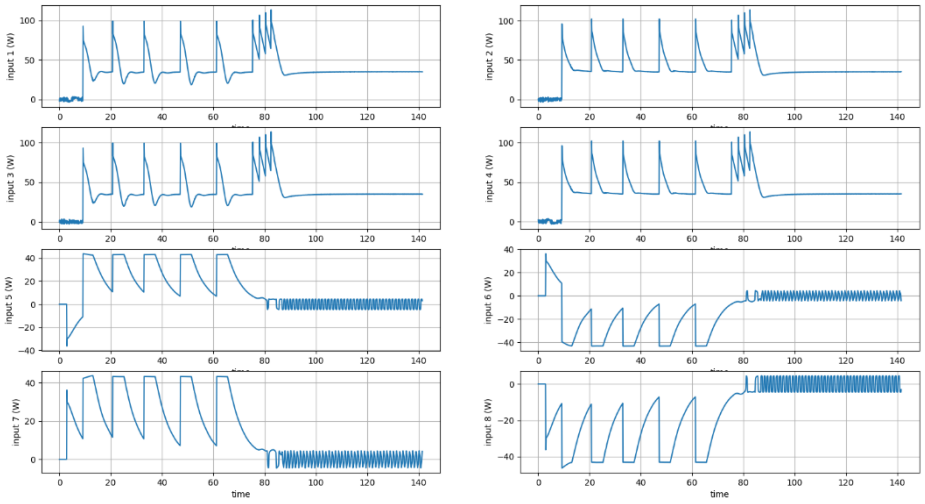
Figura 36: Resultado 1 del sistema completo

(Las gráficas describen posición, profundidad, orientación y consumo).



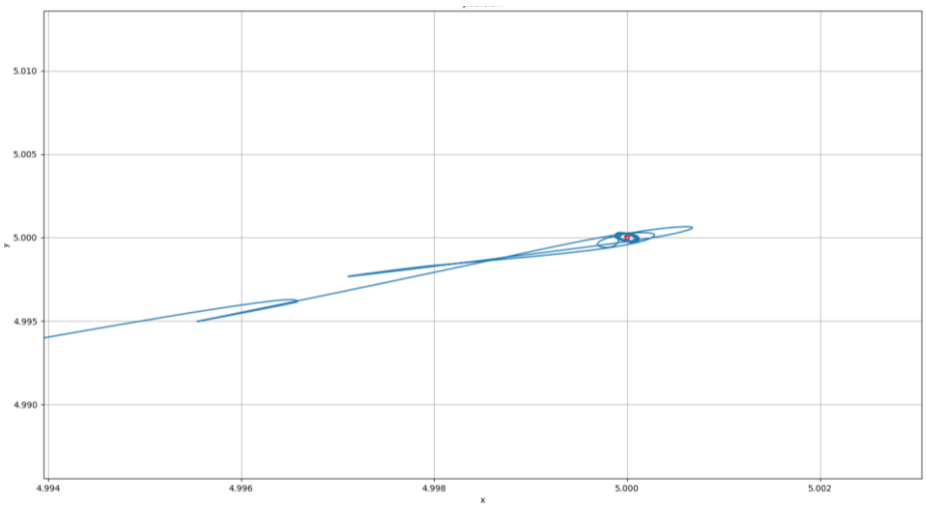






Fuente: Elaboración Propia

Figura 37: Detalle resultado 1 del sistema completo



Fuente: Elaboración Propia

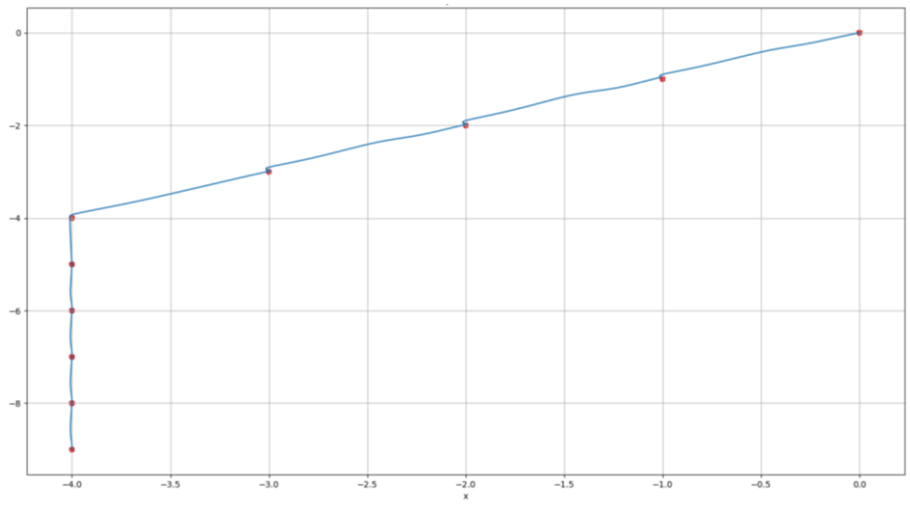
Tabla 14: Resultado 2 del sistema completo

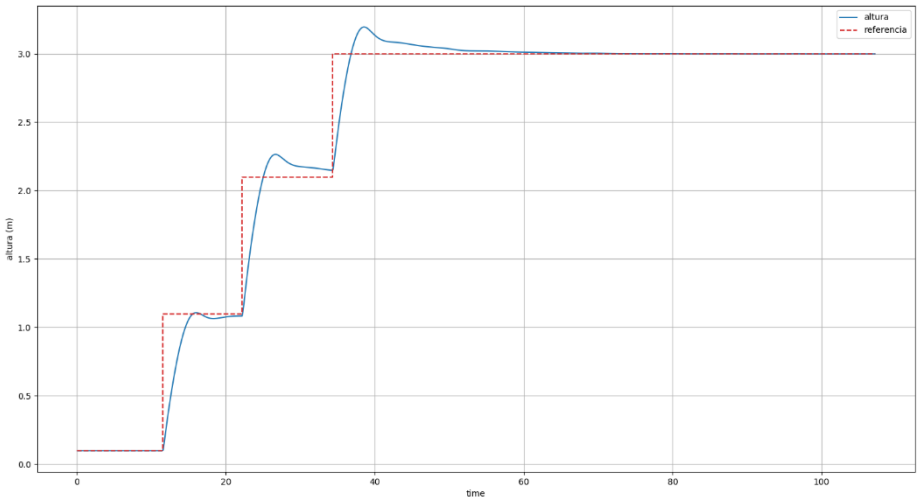
Punto inicial	[0, 0, 0, 0º]
Referencia	[-4, -10, 3, 90º]
Tiempo de llegada	95.2 s
Variación máxima de balanceo	5.2º
Variación máxima de arfada	12.9º

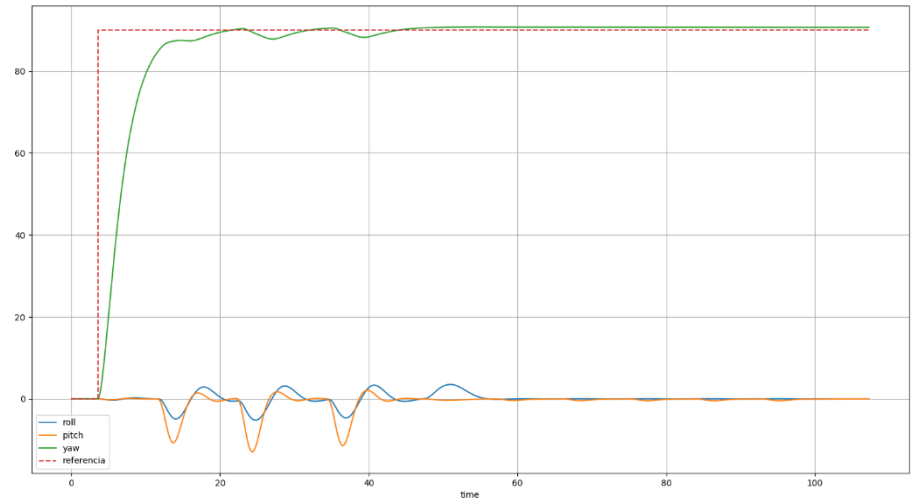
Fuente: Elaboración Propia

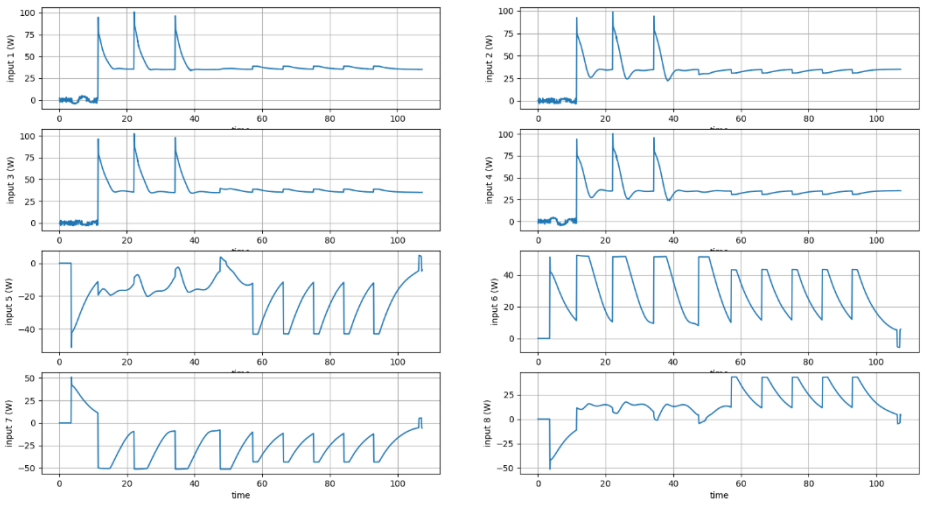
Figura 38: Resultado 2 del sistema completo

(Las gráficas describen posición, profundidad, orientación y consumo).



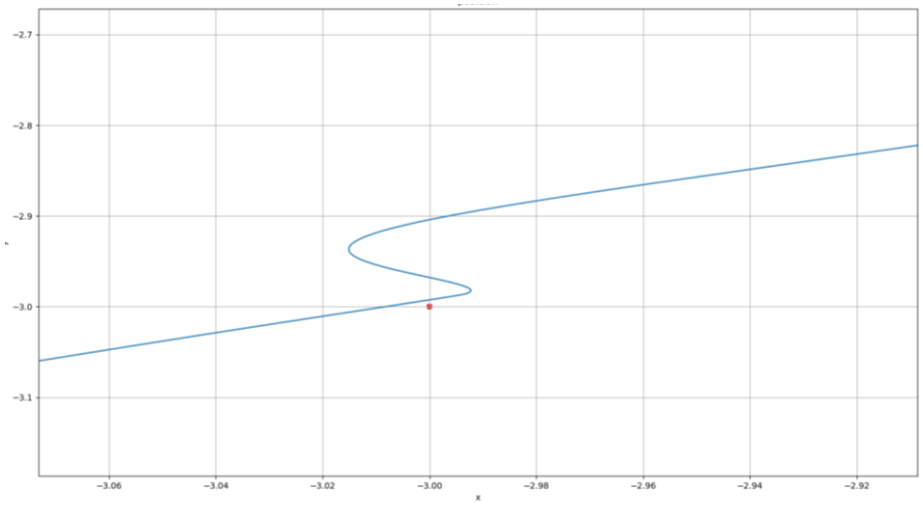






Fuente: Elaboración Propia

Figura 39: Detalle resultado 2 del sistema completo



Fuente: Elaboración Propia

5 CONCLUSIONES

El Sibiu Nano Plus como vehículo submarino ligero presenta unas características que permiten su uso como ROV en una gran multitud de aplicaciones submarinas.

El TFG desarrollado ha analizado la viabilidad del control de este vehículo submarino ligero mediante una simulación a partir de un modelo.

El modelo desarrollado proporciona los suficientes detalles para describir su comportamiento ante cualquier tipo de excitación, haciendo uso del entorno de simulación de UUVSim, permitiendo planificar misiones y estimar un comportamiento del submarino describiendo los resultados de cada uno de los 6 grados de libertad.

Tras analizar los resultados el submarino es capaz de desplazarse a cualquier punto del espacio y mantener su posición, cumpliendo así con los objetivos propuestos.

A partir de los resultados de control, se deduce una diferencia de los tiempos de respuesta de cada uno. Por ejemplo, el control de profundidad es el más rápido, el control horizontal presenta diferencias ante los movimientos en los ejes 'x' e 'y' del submarino, debidas principalmente a diferencia entre los perfiles y, por tanto, la resistencia al avance en cada uno de ellos.

Debido a esto, al observar los resultados de un movimiento completo se aprecia que un modelo de planificación, a través de interpolación de puntos, provoca que los controles más rápidos se encuentren en estado de espera de los más lentos. Esto hace tiempos de misión mayores y movimientos discontinuos del submarino.

Una de las motivaciones de este TFG es una visión preliminar del comportamiento y del diseño de control del Sibiu Nano Plus para un posterior diseño en el submarino real, con el objetivo de realizar misiones de investigación como las descritas en el proyecto Aquacollet del departamento de Ingeniería de Sistemas y Automática de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla.

Con este objetivo en mente, este TFG presenta una gran cantidad de mejoras a futuro, como pueden ser:

- Respecto al modelo, al tener la posibilidad de tratar con el submarino físico se podrían tomar medidas reales, aumentando así la fidelidad del modelo. Además, se podría aplicar CFD a estas para calcular los coeficientes hidrodinámicos, así como definir más parámetros pertenecientes al comportamiento de los propulsores y sensores.
- Respecto a la simulación, en las pruebas realizadas no se ha explotado todo el potencial presente en el simulador. Una mejora sería utilizar los servicios que permiten reducir la eficiencia o desactivar propulsores o aplicar fuerzas simulando corrientes oceánicas y remolinos; con el objetivo de obtener datos ante situaciones de fallo.
- Respecto al control, los parámetros que constituyen los PID se han configurado mediante métodos heurísticos. A lo largo del TFG, se ha indicado como un submarino no constituye un sistema lineal, luego una mejor solución podría consistir en la aplicación de un sistema de control adaptativo, permitiendo así una mejor respuesta.
- El módulo de planificación utilizado es bastante simple, se podría plantear utilizar un algoritmo más elaborado que mantenga un movimiento continuo, integre evitación de obstáculos y haga uso de maniobras que minimicen la resistencia del agua.
- Otro de los puntos mencionados en el estado del arte, es la posibilidad de montar un manipulador en el submarino, cuyo estudio de control no se ha tenido en cuenta.

Con las aportaciones de este TFG se permite facilitar la tarea de control del vehículo submarino ligero Sibiu Nano Plus, permitiendo a un operario llegar a un punto objetivo con mayor precisión y velocidad

que utilizando un control manual, siendo así útil para el desarrollo empresarial, económico y de necesidades sociales.

Como dron submarino tiene muchas posibilidades de futuro en los sectores de exploración, búsqueda y rescate, submarinismo profesional, acuacultura, mantenimiento de instalaciones submarinas e investigación subacuática.

6 APÉNDICES

6.1 Datos MeshLab del Submarino

```
 Mesh Bounding Box Size 30.043041 42.000000 24.240000
 Mesh Bounding Box Diag 57.045265
 Mesh Bounding Box min -15.043042 -21.000000 -0.240000
 Mesh Bounding Box max 15.000000 21.000000 24.000000
 Mesh Surface Area is 7139.682617
 Mesh Total Len of 444 Edges is 5152.043457 Avg Len 11.603702
 Mesh Total Len of 444 Edges is 5152.043457 Avg Len 11.603702
(including faux edges))
 Thin shell (faces) barycenter: 0.443526 -1.045152 13.364415
 Vertices barycenter 1.271806 -0.499287 9.355787
 Mesh Volume is 8705.036133
 Center of Mass is 0.150924 -0.935026 14.328722
 Inertia Tensor is :
 | 1620841.000000 987.221741 18921.701172 |
 | 987.221741 858994.500000 16829.474609 |
 | 18921.701172 16829.474609 1413541.750000 |
 Principal axes are :
 | -0.000543 0.999541 -0.030288 |
 | -0.090476 0.030114 0.995443 |
 |  0.995898 0.003280 0.090418 |
 axis momenta are :
 | 858483.937500 1412331.125000 1622562.125000 |
 Applied filter Compute Geometric Measures in 121 msec
```

Analizando el mesh, las medidas de centímetros se han interpretado como m, Así el volumen se debe escalar 1/1000, las dimensiones 1/100, y el tensor de inercia 1/100000, además el tensor de inercia se ha multiplicado por la densidad de la resina que forma la estructura del submarino (192kg/m^3).

6.2 Datos MeshLab del Propulsor

```
 Mesh Bounding Box Size 4.100000 2.800000 6.800000
 Mesh Bounding Box Diag 8.419620
 Mesh Bounding Box min 0.000000 -1.400000 -3.400000
 Mesh Bounding Box max 4.100000 1.400000 3.400000
 Mesh Surface Area is 56.895092
 Mesh Total Len of 633 Edges is 538.787109 Avg Len 0.851165
 Mesh Total Len of 633 Edges is 538.787109 Avg Len 0.851165
(including faux edges))
 Thin shell (faces) barycenter: 2.968203 -0.000066 0.000000
 Vertices barycenter 2.614554 0.000000 0.000000
 Mesh Volume is 20.502277
 Center of Mass is 2.910889 -0.000253 0.000000
 Inertia Tensor is :
```

```
| 39.260490 -0.002124 0.000000 |
| -0.002124 46.140850 3.244261 |
| 0.000000 3.244261 24.976788 |
Principal axes are :
| -0.000021 -0.148194 0.988958 |
| -1.000000 -0.000279 -0.000063 |
| -0.000285 0.988958 0.148194 |
axis momenta are :
| 24.490625 39.260487 46.626987 |
Applied filter Compute Geometric Measures in 115 msec
```

Analizando el mesh, las medidas de centímetros se han interpretado como m, Así el volumen se debe escalar 1/1000, las dimensiones 1/100, y el tensor de inercia 1/100000, además el tensor de inercia se ha multiplicado por la densidad de la resina que forma la estructura del submarino (192kg/m^3).

6.3 URDF

Macro propulsores

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">

  <!--                                                                    -->
  <!-- MACRO FOR THRUSTER UNITS  -->
  <!--                                                                    -->

  <!-- Provide the propeller mesh in a separate file with the rotation axis
  over propeller's frame X-axis in DAE (Collada) or STL format.
  -->
  <xacro:property name="prop_mesh_file" value="file://$(find sibiu_nano_p_description
)/meshes/propeller.dae"/>

  <!--
    Thruster macro with integration of joint and link. The thrusters should
    be initialized in the actuators.xacro file.
  -->
  <xacro:macro name="thruster_macro" params="robot_namespace thruster_id *origin">

    <!--
    Dummy link as place holder for the thruster frame,
    since thrusters can often be inside the collision geometry
    of the vehicle and may cause internal collisions if set otherwise
    -->
    <link name="${robot_namespace}/thruster_${thruster_id}">

      <visual>
```

```
      <geometry>
        <mesh filename="${prop_mesh_file}" scale="1 1 1" />
      </geometry>
    </visual>

    <inertial>
      <mass value="0.001" />
      <origin xyz="0 0 0" rpy="0 0 0"/>
      <inertia ixx="0.00007538" ixy="0.0" ixz="0.0"
               iyy="0.00008859" iyz="0.0"
               izz="0.000047955" />
    </inertial>
  </link>

  <!-- Joint between thruster link and vehicle base link -->
  <joint name="${robot_namespace}/thruster_${thruster_id}_joint" type="continuous">
    <xacro:insert_block name="origin" />
    <axis xyz="1 0 0" />
    <parent link="${robot_namespace}/base_link" />
    <child link="${robot_namespace}/thruster_${thruster_id}" />
  </joint>

  <gazebo>
    <!-- Thruster ROS plugin -->
    <plugin name="${robot_namespace}_${thruster_id}_thruster_model" filename="libuu
v_thruster_ros_plugin.so">
      <!-- Name of the thruster link -->
      <linkName>${robot_namespace}/thruster_${thruster_id}</linkName>

      <!-- Name of the joint between thruster and vehicle base link -->
      <jointName>${robot_namespace}/thruster_${thruster_id}_joint</jointName>

      <!-- Make the thruster aware of its id -->
      <thrusterID>${thruster_id}</thrusterID>

      <!-- Gain of the input command signal -->
      <gain>1</gain>

      <!-- Maximum allowed input value for the input signal for thruster unit -->
      <clampMax>155</clampMax>

      <!-- Minimum allowed value for the input signal for thruster unit -->
      <clampMin>-155</clampMin>

      <!-- Minimum and maximum thrust force output allowed -->
      <thrustMin>-30</thrustMin>
      <thrustMax>30</thrustMax>

      <!--
```

```
        Value from 0 to 1 to set the efficiency of the output thrust force
        Default value is 1.0
        -->
        <thrust_efficiency>1</thrust_efficiency>

        <!--
        Value from 0 to 1 to set the efficiency of the propeller as a factor
        to be multiplied to the current value of the state variable at each
        iteration.
        Default value is 1.0
        -->
        <propeller_efficiency>1</propeller_efficiency>

        <!--
        Choose one of the propeller dynamics models below for your implementation
        This will describe the dynamic model for the state variable of your thruster
unit,
        which can be, e.g., the angular velocity of the propeller.
        -->

        <!-- Simple zero-order model -->
        <dynamics>
          <type>ZeroOrder</type>
        </dynamics>

        <!-- Basic curve
        Input: x
        Output: thrust
        Function: thrust = rotorConstant * x * abs(x)
        -->
        <conversion>
          <type>Basic</type>
          <rotorConstant>0.0012081</rotorConstant> <!-
- As the implementation is about input², we have divided the root by the max input in
 order to maintain proportionality -->
        </conversion>

      </plugin>
    </gazebo>

    <gazebo reference="${robot_namespace}/thruster_${thruster_id}">
      <selfCollide>false</selfCollide>
    </gazebo>
  </xacro:macro>
</robot>
```

Descripción del submarino

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
  <!-- Loading some constants -->
  <xacro:include filename="$(find uuv_descriptions)/urdf/common.urdf.xacro"/>
  <!-- Loading file with sensor macros -->
  <xacro:include filename="$(find uuv_sensor_ros_plugins)/urdf/sensor_snippets.xacro"
/>
  <!-- Loading the UUV simulator ROS plugin macros -->
  <xacro:include filename="$(find uuv_gazebo_ros_plugins)/urdf/snippets.xacro"/>
  <!-- Loading vehicle's specific macros -->
  <xacro:include filename="$(find sibiu_nano_p_description)/urdf/snippets.xacro"/>

  <!--
    Vehicle's parameters (remember to enter the model parameters below)
  -->

  <xacro:property name="mass" value="8.1648"/>
  <!-- Center of gravity -->
  <xacro:property name="cog" value="0 0 0"/>
  <!-- Fluid density -->
  <xacro:property name="rho" value="1028"/>

    <!-- Center of buoyancy -->
    <xacro:property name="sibiu_nano_p_cob" value="0 0 0.03"/>

    <!-- Vehicle's actual volume (Gazebo cannot compute the volume out of the mesh) -
->
    <xacro:property name="sibiu_nano_p_volume" value="0.008705"/>

    <!-- Describing the dimensions of the vehicle's bounding box -->
    <xacro:property name="sibiu_nano_p_length" value="0.42"/>
    <xacro:property name="sibiu_nano_p_width"  value="0.3"/>
    <xacro:property name="sibiu_nano_p_height" value="0.24"/>


  <!--
    Visual mesh file for the vehicle, usually in DAE (Collada) format. Be sure to sto
re the
    mesh with the origin of the mesh on the same position of the center of mass, othe
rwise
    the mesh pose will have to be corrected below in the <visual> block.
    Open the meshes for the RexROV vehicle in Blender to see an example on the mesh p
lacement.
  -->
```

```
  <xacro:property name="visual_mesh_file" value="file://$(find sibiu_nano_p_descripti
on)/meshes/vehicle.dae"/>

  <!-- Collision geometry mesh, usually in STL format (it is recommended to keep
  this geometry as simple as possible to improve the performance the physics engine
  regarding the computation of collision forces) -->
  <xacro:property name="collision_mesh_file" value="file://$(find sibiu_nano_p_descri
ption)/meshes/vehicle.stl"/>

  <!-- Vehicle macro -->
  <xacro:macro name="sibiu_nano_p_base" params="namespace *gazebo">

      <!-- Rigid body description of the base link -->
      <link name="${namespace}/base_link">
        <!--
          Be careful to setup the coefficients for the inertial tensor,
          otherwise your model will become unstable on Gazebo
        -->
        <inertial>
          <mass value="${mass}" />
          <origin xyz="${cog}" rpy="0 0 0"/>
          <inertia ixx="3.1120128" ixy="0.0018816" ixz="0.0363264" iyy="1.6492608" iy
z="0.0322944" izz="2.7139968" />
        </inertial>

        <visual>
          <origin xyz="0 0 0" rpy="0 0 0"/>
          <geometry>
            <mesh filename="${visual_mesh_file}" scale="1 1 1" />
          </geometry>
        </visual>

        <!-- This flag will make the link neutrally buoyant -->
        <neutrally_buoyant>0</neutrally_buoyant>

        <!-- Link's volume -->
        <volume>${sibiu_nano_p_volume}</volume>

        <!-- Link's bounding box, it is used to recalculate the immersed
        volume when close to the surface.
        This is a workaround the invalid bounding box given by Gazebo-->
        <box>
          <width>${sibiu_nano_p_width}</width>
          <length>${sibiu_nano_p_length}</length>
          <height>${sibiu_nano_p_height}</height>
        </box>

        <!-- Center of buoyancy -->
        <center_of_buoyancy>${sibiu_nano_p_cob}</center_of_buoyancy>
```

```
        <!--
        hydrodynamic model
        -->

        <!-- 1) Fossen's equation of motion -->
        <hydrodynamic_model>
          <type>fossen</type>
          <added_mass>
           25   0     0      0     0     0
           0     25  0      0     0     0
           0     0     17.87  0     0     0
           0     0     0      2.1   0     0
           0     0     0      0     17.87  0
           0     0     0      0     0     6.7
          </added_mass>
          <!--
            The linear damping coefficients can be provided as a diagonal (6 elements
)
            or a full matrix (36 coefficients), like the added-
mass coefficients above
          -->
          <linear_damping>
          -25.15 -7.364 -17.955 -1.888 -0.761 -3.744
          </linear_damping>

          <!--
            The quadratic damping coefficients can be provided as a diagonal (6 eleme
nts)
            or a full matrix (36 coefficients), like the added-
mass coefficients above
          -->
          <quadratic_damping>
          -17.77 -125.9 -72.36 -0.1246 -0.1246 -0.1857
          </quadratic_damping>

        </hydrodynamic_model>




      </link>

    <gazebo reference="${namespace}/base_link">
      <selfCollide>false</selfCollide>
    </gazebo>

    <!-- Set up hydrodynamic plugin given as input parameter -->
    <xacro:insert_block name="gazebo"/>
```

```
    <!-- Include the thruster modules-->
    <xacro:include filename="$(find sibiu_nano_p_description)/urdf/actuators.xacro"/>


    <!-- Include the sensor modules -->
    <xacro:include filename="$(find sibiu_nano_p_description)/urdf/sensors.xacro"/>

  </xacro:macro>

</robot>
```

Posición sensores y actuadores

```
<robot xmlns:xacro="http://www.ros.org/wiki/xacro">

  <!-- Mount a GPS. -->
  <xacro:default_gps namespace="${namespace}" parent_link="${namespace}/base_link" />

  <!-- Mount a Pose 3D sensor. -->
  <xacro:default_pose_3d namespace="${namespace}" parent_link="${namespace}/base_link
" />

  <!-- Pressure  -->
  <xacro:default_pressure namespace="${namespace}" parent_link="${namespace}/base_lin
k">
    <origin xyz="0 0 0" rpy="0 0 0"/>
  </xacro:default_pressure>

  <!-- IMU  -->
  <xacro:default_imu namespace="${namespace}" parent_link="${namespace}/base_link">
    <origin xyz="0 0 0" rpy="0 0 0"/>
  </xacro:default_imu>

  <!-- Mount a camera  -->
  <xacro:default_camera namespace="${namespace}" parent_link="${namespace}/base_link"
 suffix="">
    <origin xyz="0 0 0" rpy="0 0 0"/>
  </xacro:default_camera>

<!-- Magnetometer  -->
  <xacro:default_magnetometer namespace="${namespace}" parent_link="${namespace}/base
_link"/>
```

```
    <!-- Adding the thruster units with the macro created in snippets.xacro -->
    <!--
    Important:
        - The thruster IDs must be given as integers and must be unique to each thrus
ter unit
        - The thruster pose in the <origin> block is relative to the body's center of
 mass. Be
          aware that Gazebo does not use the SNAME convention per default.
    -->


    <!-- verticales -->
    <xacro:thruster_macro robot_namespace="${namespace}" thruster_id="0">
        <origin xyz="-0.084 0.095 0.088" rpy="0 ${0.5*pi} 0" />
    </xacro:thruster_macro>

    <xacro:thruster_macro robot_namespace="${namespace}" thruster_id="1">
        <origin xyz="0.084 0.095 0.088" rpy="0 ${0.5*pi} 0" />
    </xacro:thruster_macro>

    <xacro:thruster_macro robot_namespace="${namespace}" thruster_id="2">
        <origin xyz="-0.084 -0.095 0.088" rpy="0 ${0.5*pi} 0" />
    </xacro:thruster_macro>

    <xacro:thruster_macro robot_namespace="${namespace}" thruster_id="3">
        <origin xyz="0.084 -0.095 0.088" rpy="0 ${0.5*pi} 0" />
    </xacro:thruster_macro>

    <!-- horizontales -->

    <xacro:thruster_macro robot_namespace="${namespace}" thruster_id="4">
        <origin xyz="-0.201 0.07 -0.015" rpy="0 0 ${1.0/4.0*pi}" />
    </xacro:thruster_macro>

    <xacro:thruster_macro robot_namespace="${namespace}" thruster_id="5">
        <origin xyz="0.201 0.07 -0.015" rpy="0 0 -${5.0/4.0*pi}" />
    </xacro:thruster_macro>

    <xacro:thruster_macro robot_namespace="${namespace}" thruster_id="6">
        <origin xyz="-0.201 -0.07 -0.015" rpy="0 0 -${1.0/4.0*pi}" />
    </xacro:thruster_macro>

    <xacro:thruster_macro robot_namespace="${namespace}" thruster_id="7">
        <origin xyz="0.201 -0.07 -0.015" rpy="0 0 ${5.0/4.0*pi}" />
    </xacro:thruster_macro>
```

```
</robot>
```

6.4 Controlador

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist, Point, Wrench
from sensor_msgs.msg import FluidPressure, MagneticField, Imu, NavSatFix
from nav_msgs.msg import Odometry
from math import pow, atan2, sqrt, atan, cos, sin, tan, pi
import tf.transformations
from visualization_msgs.msg import Marker
from uuv_gazebo_ros_plugins_msgs.msg import FloatStamped
import sys
from sibiu_nano_p_control.srv import *



#global variables

#parameters
ns=rospy.get_param('/controlador/namespace', '/sibiu_nano_p') #namespace
debug=int(rospy.get_param('/controlador/debug_level', 2)) #nivel de debug
tolerance=float(rospy.get_param('/controlador/tolerance', 0.1)) #tolerancia enviar re
spuesta servicios
tolerance_angular=float(rospy.get_param('/controlador/tolerance_angular', 0.1)) #tole
rancia angular

#input parameter



#we define a simple PID
class pid:
    def __init__(self, kp, ki, kd, sat):
        self.kp=kp
        self.ki=ki
        self.kd=kd
        self.dererr=0.0
        self.interr=0.0
        self.sat=sat

    def derivative(self, error):
        a=self.dererr
```

```
        self.dererr=error
        return self.kd*(error-a)

    def integral(self, error):
        if self.sat==0 or (abs(self.proportional(error)+self.interr * self.ki)<self.s
at):
            self.interr+=error
        return self.interr * self.ki

    def proportional(self, error):
        return error*self.kp

    def PD(self, error):
        return self.proportional(error)+self.derivative(error)

    def PI(self, error):
        return self.proportional(error)+self.integral(error)

    def PID(self, error):
        output=self.proportional(error)+self.integral(error)+self.derivative(error)
        if abs(output)<self.sat:
            return output
        if output>self.sat:
            return self.sat
        return -self.sat

#this function translates the debug level from number to value
def debug_level(number):
    if number==0:
        return rospy.NONE
    if number==1:
        return rospy.FATAL
    if number==2:
        return rospy.ERROR
    if number==3:
        return rospy.INFO
    if number==4:
        return rospy.WARN
    if number==5:
        return rospy.DEBUG

class controller:
    def __init__(self):
        rospy.init_node('control_publisher', anonymous=True, log_level=debug_level(de
bug)) # we init the node
        #we subscribe to the sensors
        self.pressure_sub=rospy.Subscriber(ns+'/pressure', FluidPressure, self.update
_pressure)
```

```
        self.compass_sub=rospy.Subscriber(ns+'/magnetometer', MagneticField, self.upd
ate_magentometer)
        self.IMU_sub=rospy.Subscriber(ns+'/imu', Imu, self.update_imu)
        self.GPS_sub=rospy.Subscriber(ns+'/gps', NavSatFix, self.update_GPS)
        self.pose_sub=rospy.Subscriber(ns+'/pose_gt', Odometry, self.update_pose)

        #we prepare the publisher for the control allocation
        self.thruster_manager_pub=rospy.Publisher(ns+'/thruster_manager/input', Wrenc
h, queue_size=10)

        #we prepare the publisher to plot the reference
        self.reference_pub=rospy.Publisher('controller/reference', Twist, queue_size=
10)

        #we create the services for the planner
        self.altitude_service = rospy.Service('/controller/'+'_reference_altitude', a
ltitude, self.height_state)
        self.position_service = rospy.Service('/controller/reference_position', posit
ion, self.position_state)
        self.horizontal_plane_service = rospy.Service('/controller/'+'_reference_hori
zontal', horizontal_plane, self.horizontal_plane_state)
        self.heading_service = rospy.Service('/controller/'+'_reference_heading', hea
ding, self.heading_state)


        #we define one PID for each DOF
        self.x_axis=pid(180, 0.01, 20, 140)
        self.y_axis=pid(180, 0.01, 20, 140)
        self.z_axis=pid(620, 7, 320, 6500)
        self.roll_axis=pid(5, 0, 150, 1500)
        self.pitch_axis=pid(5, 0, 150, 1500)
        self.yaw_axis=pid(23, 0.005 , 11, 1500)

        #State variables to deactivate/activate control when needed
        self.control_buoyancy=False
        self.control_horizontal=False
        self.control_heading=False

        #variables we will make use of
        self.rate = rospy.Rate(10) #10Hz
        self.signal=Wrench() # output signal
        self.rate.sleep() #we wait until the sensors give the first value
        self.reference=Twist()
        self.last_reference=Twist()
        self.control()


    def update_pose(self, data):
        self.pose=data.pose.pose.position
```

```
        orientation = [data.pose.pose.orientation.x, data.pose.pose.orientation.y, da
ta.pose.pose.orientation.z, data.pose.pose.orientation.w]
        self.rot=tf.transformations.euler_from_quaternion(orientation) #roll pitch ya
w



    def update_pressure(self, data):
        self.pressure=data
        self.deepness=(self.pressure.fluid_pressure-
101.325)*10/103.25 #transformation between pressure and deepness

    def update_magentometer(self, data):
        self.magnetic_field=data.magnetic_field

    def update_imu(self, data):
        self.imu_angular=data.angular_velocity
        self.imu_linear=data.linear_acceleration
        orientation = [data.orientation.x, data.orientation.y, data.orientation.z, da
ta.orientation.w]
        (self.roll,self.pitch,self.yaw) = tf.transformations.euler_from_quaternion(or
ientation)


    def update_GPS(self, data):
        self.gps=[data.latitude, data.longitude, data.altitude]


    def control(self):
        while True:

            #roll control
            self.signal.torque.x=self.roll_axis.PID(self.roll)

            #pitch control
            self.signal.torque.y=self.pitch_axis.PID(-
self.pitch) #negative due to axis of submarine vs axis IMU

            #horizontal control
            if self.control_horizontal==True:
                angulo_ataque=atan2(self.reference.linear.y-
self.pose.y,self.reference.linear.x-self.pose.x)-self.yaw
                distance=sqrt(pow(self.reference.linear.x-
self.pose.x,2)+pow(self.reference.linear.y-self.pose.y,2))
                self.signal.force.x=self.x_axis.PID(distance)*sin(angulo_ataque)
                self.signal.force.y=self.y_axis.PID(distance)*cos(angulo_ataque)


            #heigth control
```

```
            if self.control_buoyancy==True:
                self.signal.force.z=self.z_axis.PID(self.reference.linear.z-
self.deepness)


            #heading control
            if self.control_heading==True:
                if abs(self.reference.angular.z-
self.yaw) > pi: #we choose the shortest path
                    if self.reference.angular.z>0:
                        self.reference.angular.z-=2*pi
                    else:
                        self.reference.angular.z+=2*pi
                self.signal.torque.z=self.yaw_axis.PID(self.reference.angular.z-
self.yaw)

            self.thruster_manager_pub.publish(self.signal) #we publish the control ou
tput to the control allocator
            self.rate.sleep()
            if self.last_reference.linear.x != self.reference.linear.x or self.last_r
eference.linear.y != self.reference.linear.y or self.last_reference.linear.z != self.
reference.linear.z or self.last_reference.angular.z != self.reference.angular.z:
                self.reference_pub.publish(self.reference) #we send the reference to
plot
                self.last_reference.linear.x=self.reference.linear.x
                self.last_reference.linear.y=self.reference.linear.y
                self.last_reference.linear.z=self.reference.linear.z
                self.last_reference.angular.z=self.reference.angular.z




    def height_state(self, data):
        self.reference.linear.z=data.z #update reference value
        self.control_buoyancy=True#activate control
        while abs(self.reference.linear.z-
self.deepness)>tolerance: #stablish control exit
            self.rate.sleep()
        return True

    def position_state(self, data):
        self.reference.linear.x=data.x#update reference value
        self.reference.linear.y=data.y
        self.reference.linear.z=data.z
        self.reference.angular.z=data.yaw
        self.control_horizontal=True #activate control
        self.control_buoyancy=True
        self.control_heading=True
```

```
        while (sqrt(pow(self.reference.linear.x-
self.pose.x,2)+pow(self.reference.linear.y-
self.pose.y,2))>tolerance) or (abs(self.reference.linear.z-
self.deepness)>tolerance) or (abs(self.reference.angular.z-
self.yaw)>tolerance_angular):#stablish control exit
            self.rate.sleep()
        return True
    def horizontal_plane_state(self, data):
        self.reference.linear.x=data.x#update reference value
        self.reference.linear.y=data.y
        self.control_horizontal=True#activate control
        while sqrt(pow(self.reference.linear.x-
self.pose.x,2)+pow(self.reference.linear.y-
self.pose.y,2))>tolerance:#stablish control exit
            self.rate.sleep()
        return True
    def heading_state(self, data):
        self.reference.angular.z=data.yaw#update reference value
        self.control_heading=True#activate control
        while abs(self.reference.angular.z-
self.yaw)>tolerance_angular:#stablish control exit
            self.rate.sleep()
        return True




if __name__ == '__main__':
    try:
        x = controller()
        rospy.spin()
    except rospy.ROSInterruptException:
        print('publisher::Exception')
    print('Leaving controller')
```

6.5 Planificador

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist, Point, Wrench
from sensor_msgs.msg import FluidPressure ,MagneticField, Imu, NavSatFix
from nav_msgs.msg import Odometry
from math import pow, atan2, sqrt, atan, cos, sin, tan, pi
import tf.transformations
from visualization_msgs.msg import Marker
```

```
from uuv_gazebo_ros_plugins_msgs.msg import FloatStamped
import sys
from sibiu_nano_p_control.srv import *



#global variables

#parameters
ns=rospy.get_param('/planner/namespace', '/sibiu_nano_p')
debug=int(rospy.get_param('/planner/debug_level', 2))
interpolation_distance=float(rospy.get_param('/planner/interpolation_distance', 1.0))


#input parameter
reference=Twist()

#transformation coeficients
degree_to_rad=pi/180.0


#this function translates the debug level from number to value
def debug_level(number):
    if number==0:
        return rospy.NONE
    if number==1:
        return rospy.FATAL
    if number==2:
        return rospy.ERROR
    if number==3:
        return rospy.INFO
    if number==4:
        return rospy.WARN
    if number==5:
        return rospy.DEBUG

def sentido(dist): #this function returns 1 or -1 depending on the given value
    if dist>0:
        return 1
    return -1

def dist(p1,p2):
    return sqrt(pow(p1-p2,2))

class planner:
    def __init__(self):
        rospy.init_node('planner_publisher', anonymous=True, log_level=debug_level(de
bug)) #init the node
```

```
        #subscribe to sensors
        self.pressure_sub=rospy.Subscriber(ns+'/pressure', FluidPressure, self.update
_pressure)
        self.compass_sub=rospy.Subscriber(ns+'/magnetometer', MagneticField, self.upd
ate_magentometer)
        self.IMU_sub=rospy.Subscriber(ns+'/imu', Imu, self.update_imu)
        self.GPS_sub=rospy.Subscriber(ns+'/gps', NavSatFix, self.update_GPS)
        self.pose_sub=rospy.Subscriber(ns+'/pose_gt', Odometry, self.update_pose)

        #variables we will use
        self.rate = rospy.Rate(10) #10 Hz
        self.rate.sleep() #we wait until we have a first value for the sensors
        self.plan()

    def update_pose(self, data):
        self.pose=data.pose.pose.position
        orientation = [data.pose.pose.orientation.x, data.pose.pose.orientation.y, da
ta.pose.pose.orientation.z, data.pose.pose.orientation.w]
        self.rot=tf.transformations.euler_from_quaternion(orientation) #[roll pitch y
aw]


    def update_pressure(self, data):
        self.pressure=data
        self.deepness=(self.pressure.fluid_pressure-
101.325)*10/103.25 #transformation between pressure and deepness

    def update_magentometer(self, data):
        self.magnetic_field=data.magnetic_field

    def update_imu(self, data):
        self.imu_angular=data.angular_velocity
        self.imu_linear=data.linear_acceleration
        orientation = [data.orientation.x, data.orientation.y, data.orientation.z, da
ta.orientation.w]
        (self.roll,self.pitch,self.yaw) = tf.transformations.euler_from_quaternion(or
ientation)


    def update_GPS(self, data):
        self.gps=[data.latitude, data.longitude, data.altitude]

    def plan(self):
        #we interpolate heigth
        heigth_list=[self.deepness] #we set as first point the current heigth

        while abs(heigth_list[len(heigth_list)-1]-
reference.linear.z)>interpolation_distance: #until we reach the goal point
```

```
            heigth_list.append(heigth_list[len(heigth_list)-
1]+interpolation_distance*sentido(reference.linear.z-heigth_list[len(heigth_list)-
1])) #we add the next heigth point within the interpolation distance
        heigth_list.append(reference.linear.z) #we add the goal point

        #we interpolate horizontal position
        list_x=[self.pose.x] #we set as first point the current position
        while dist(list_x[len(list_x)-
1],reference.linear.x)>interpolation_distance: #until we reach the goal point
            list_x.append(list_x[len(list_x)-
1]+interpolation_distance*sentido(reference.linear.x-list_x[len(list_x)-
1])) #we add the next heigth point within the interpolation distance
        list_x.append(reference.linear.x) #we add the goal point

        list_y=[self.pose.y] #we set as first point the current position
        while dist(list_y[len(list_y)-
1],reference.linear.y)>interpolation_distance: #until we reach the goal point
            list_y.append(list_y[len(list_y)-
1]+interpolation_distance*sentido(reference.linear.y-list_y[len(list_y)-
1])) #we add the next heigth point within the interpolation distance
        list_y.append(reference.linear.y) # we add the goal point


        #we make both lists have the same size
        while len(list_x)>len(list_y):
            list_y.append(list_y[len(list_y)-1])

        while len(list_y)>len(list_x):
            list_x.append(list_x[len(list_x)-1])

        horizontal_list=[]
        for i in range(len(list_x)):
            horizontal_list.append([list_x[i],list_y[i]])

        #we calculate how many times we call the service
        number=max([len(heigth_list),len(horizontal_list)])

        #both list must have the same size

        while len(heigth_list)<number:
            heigth_list.append(heigth_list[len(heigth_list)-1])

        while len(horizontal_list)<number:
            horizontal_list.append(horizontal_list[len(horizontal_list)-1])

        #we call the controller service
        rospy.wait_for_service('/controller/reference_position')
        for i in range(number-1):
            try:
```

```
                position_controller = rospy.ServiceProxy('/controller/reference_posit
ion', position)
                resp1 = position_controller(horizontal_list[i][0], horizontal_list[i]
[1], heigth_list[i],reference.angular.z)
            except rospy.ServiceException as e:
                print("Service call failed: %s"%e)
        sys.exit(1)



if __name__ == '__main__':
    if len(sys.argv) == 5:        #asign the reference value
        reference.linear.x = float(sys.argv[1])
        reference.linear.y = float(sys.argv[2])
        reference.linear.z = float(sys.argv[3])
        reference.angular.z = float(sys.argv[4])*degree_to_rad
    else:
        print("incorrect call \nYou must use \"rosrun sibiu_nano_p_control planner x
y z yaw\"")
        sys.exit(1)
    try:
        x = planner() #call the planner
        rospy.spin() #stay
    except rospy.ROSInterruptException:
        print('publisher::Exception')
    print('Leaving controller')
```

6.6 Graficas

```
#!/usr/bin/env python
import rospy
import matplotlib.pyplot as plt
from geometry_msgs.msg import Twist, Point, Wrench
from sensor_msgs.msg import FluidPressure ,MagneticField, Imu, NavSatFix
from nav_msgs.msg import Odometry
import tf.transformations
from uuv_gazebo_ros_plugins_msgs.msg import FloatStamped
from math import pi

ns='/sibiu_nano_p' #namespace

rad_to_degree=180.0/pi
degree_to_rad=pi/180


class graficas:
    def __init__(self):
```

```
        rospy.init_node('grabadora', anonymous=True)#we init the node
        self.rate = rospy.Rate(10)
        while rospy.get_time()<1:#we wait until time starts
            self.rate.sleep()
        self.time=rospy.get_time() #save time to check variation

        #we subscribe to the sensors
        self.pressure_sub=rospy.Subscriber(ns+'/pressure', FluidPressure, self.update
_pressure)

        self.IMU_sub=rospy.Subscriber(ns+'/imu', Imu, self.update_imu)
        self.pose_sub=rospy.Subscriber(ns+'/pose_gt', Odometry, self.update_pose)
        self.reference_sub=rospy.Subscriber('controller/reference', Twist, self.updat
e_reference)

        #we subscribe to the sensors we init the lists
        self.reference_t=[self.tiempo()]

        self.lista_p=[[], [], [], [], [], [], [], []]
        self.lista_p_t=[[], [], [], [], [], [], [], []]
        self.lista_a=[]
        self.lista_a_t=[]
        self.lista_angles=[[],[],[]]
        self.lista_angles_t=[]
        self.lista_h=[[],[]]
        self.lista_h_t=[]

        for ident in range(8):#we subscribe to the inputs
            self.thruster=rospy.Subscriber(ns+'/thrusters/'+str(ident)+'/input', Floa
tStamped, self.read_input, ident)

        #aux value to initialize reference
        aux=Twist()
        self.rate.sleep()
        self.rate.sleep()
        aux.linear.x=self.lista_h[0][0]
        aux.linear.y=self.lista_h[1][0]
        aux.linear.z=self.lista_a[0]*degree_to_rad
        aux.angular.z=self.lista_angles[2][0]
        self.reference=[aux]

    def tiempo(self):# this function returns time since program started runing
        return rospy.get_time()-self.time


    def read_input(self, data, ident):
        self.lista_p[ident].append(data.data)
        self.lista_p_t[ident].append(self.tiempo())
```

```
    def update_pose(self, data):
        self.lista_h[0].append(data.pose.pose.position.x)
        self.lista_h[1].append(data.pose.pose.position.y)
        self.lista_h_t.append(self.tiempo())


    def update_pressure(self, data):
        self.lista_a.append((data.fluid_pressure-
101.325)*10/103.25) #deepness formula
        self.lista_a_t.append(self.tiempo())

    def update_imu(self, data):
        orientation = [data.orientation.x, data.orientation.y, data.orientation.z, da
ta.orientation.w]
        (self.roll,self.pitch,self.yaw) = tf.transformations.euler_from_quaternion(or
ientation)
        self.lista_angles[0].append(self.roll*rad_to_degree)
        self.lista_angles[1].append(self.pitch*rad_to_degree)
        self.lista_angles[2].append(self.yaw*rad_to_degree)
        self.lista_angles_t.append(self.tiempo())

    def update_reference(self, data):
        self.reference.append(self.reference[len(self.reference)-
1])#as we want a vertical line for reference changes, we store the value just before
and after the change
        self.reference.append(data)
        self.reference_t.append(self.tiempo()-0.001)
        self.reference_t.append(self.tiempo())

    def imprime(self):
    ##########################################
    # plot the data
    ##########################################
        self.reference_t.append(self.lista_a_t[len(self.lista_a_t)-1])


    #thrusters input

        fig=plt.figure(1)
        ax = fig.add_subplot(4, 2, 1)
        ax.plot(self.lista_p_t[0], self.lista_p[0],label='0')
        ax.grid(True)
        ax.set(xlabel='time', ylabel='input 1 (W)')
        ax = fig.add_subplot(4, 2, 2)
        ax.plot(self.lista_p_t[1], self.lista_p[1],label='1')
        ax.grid(True)
        ax.set(xlabel='time', ylabel='input 2 (W)')
        ax = fig.add_subplot(4, 2, 3)
```

```
        ax.plot(self.lista_p_t[2], self.lista_p[2],label='2')
        ax.grid(True)
        ax.set(xlabel='time', ylabel='input 3 (W)')
        ax = fig.add_subplot(4, 2, 4)
        ax.plot(self.lista_p_t[3], self.lista_p[3],label='3')
        ax.grid(True)
        ax.set(xlabel='time', ylabel='input 4 (W)')
        ax = fig.add_subplot(4, 2, 5)
        ax.plot(self.lista_p_t[4], self.lista_p[4],label='4')
        ax.grid(True)
        ax.set(xlabel='time', ylabel='input 5 (W)')
        ax = fig.add_subplot(4, 2, 6)
        ax.plot(self.lista_p_t[5], self.lista_p[5],label='5')
        ax.grid(True)
        ax.set(xlabel='time', ylabel='input 6 (W)')
        ax = fig.add_subplot(4, 2, 7)
        ax.plot(self.lista_p_t[6], self.lista_p[6],label='6')
        ax.grid(True)
        ax.set(xlabel='time', ylabel='input 7 (W)')
        ax = fig.add_subplot(4, 2, 8)
        ax.plot(self.lista_p_t[7], self.lista_p[7],label='7')
        ax.set(xlabel='time', ylabel='input 8 (W)')
        ax.grid(True)

        ########################################
        #deepness

        fig2=plt.figure(2)
        ax2= fig2.add_subplot(1, 1, 1)
        ref_list=[]
        for i in self.reference:
            ref_list.append(i.linear.z)
        ref_list.append(ref_list[len(ref_list)-1])

        ax2.plot(self.lista_a_t, self.lista_a, color='tab:blue', label='altura')
        ax2.plot(self.reference_t, ref_list, linestyle='dashed', color='tab:red', lab
el='referencia')
        ax2.set(xlabel='time', ylabel='altura (m)')
        ax2.grid(True)
        ax2.legend()

        ########################################
        #yaw

        ref_list=[]
        for i in self.reference:
            ref_list.append(i.angular.z*rad_to_degree)
        ref_list.append(ref_list[len(ref_list)-1])
```

```
        fig3=plt.figure(3)
        ax3 =fig3.add_subplot(1, 1, 1)
        ax3.plot(self.lista_angles_t, self.lista_angles[0], color='tab:blue', label='
roll')
        ax3.plot(self.lista_angles_t, self.lista_angles[1], color='tab:orange',label=
'pitch')
        ax3.plot(self.lista_angles_t, self.lista_angles[2], color='tab:green',label='
yaw')
        ax3.plot(self.reference_t, ref_list, linestyle='dashed', color='tab:red', lab
el='referencia')
        ax3.set(xlabel='time')
        ax3.grid(True)
        ax3.legend()



        ########################################
        #horizontal position

        ref_list=[[],[]]
        for i in self.reference:
            ref_list[0].append(i.linear.x)
            ref_list[1].append(i.linear.y)
        fig4=plt.figure(4)
        ax4 = fig4.add_subplot(1, 1, 1)
        ax4.plot(self.lista_h[0], self.lista_h[1], color='tab:blue', label='position'
)
        ax4.scatter(ref_list[0], ref_list[1], linestyle='dashed', color='tab:red', la
bel='referencia')
        ax4.set(xlabel='x', ylabel='y', title='position')
        ax4.grid(True)



        # set the limits
        #ax.set_xlim([0, 1])
        #ax.set_ylim([0, 1])
        #plt.plot(range(len(lista)),lista)
        plt.show()


if __name__ == '__main__':
    try:
        x = graficas()
        rospy.spin()
    except rospy.ROSInterruptException:
        print('publisher::Exception')
    print('Leaving controller')
    x.imprime()
```

6.7 Matriz de distribución de control

```
tam:
- [0.0, 0.0, 0.0, 0.0,
  0.7071068613706476, 0.7071068613706476, -0.7071065406341925, -0.7071065406341925]
- [0.0, 0.0, 0.0, 0.0,
  0.7071067010024383, -0.7071067010024383, 0.7071070217388206, -0.7071070217388206]
- [1.0, 1.0, 1.0, 1.0,
  0.0, 0.0, 0.0, 0.0]
- [0.084, 0.084, -0.084, -0.084,
  0.010606602, 0.010606602, -0.010606602, -0.010606602]
- [-0.07, 0.07, -0.07, 0.07,
  -0.010606602, 0.010606602, -0.010606602, 0.010606602]
- [0.0, 0.0, 0.0, 0.0,
  -0.1916259377, 0.1916259377, 0.1916259377, -0.1916259377]
```

7 BIBLIOGRAFÍA

Acosta, S. A. (2020, September 22). *10 curiosidades sobre los océanos*. https://www.nationalgeographic.com.es/ciencia/10-curiosidades-sobre-oceanos_15577

Ahmad, T., & Iqbal, J. (2014). Underwater robotic vehicles: Latest development trends and potential challenges. *Science International*, *26*, 1111–1117. https://www.researchgate.net/publication/280642801_Underwater_robotic_vehicles_Latest_development_trends_and_potential_challenges

Ancellin, M., & Dias, F. (2019). Capytaine: a Python-based linear potential flow solver. *Journal of Open Source Software*, *4*(36), 1341. https://doi.org/10.21105/JOSS.01341

BackerClub. (n.d.). *Sibiu Nano: Affordable & Portable Underwater Robot on BackerClub*. Retrieved September 11, 2021, from https://backerclub.co/project.php?id=12476

Berg, V., Supervisor, M. T., Asgeir, :, & Sørensen, J. (2012). *Development and Commissioning of a DP system for ROV SF 30k*.

BlueRobotics. (n.d.). Retrieved September 7, 2021, from https://bluerobotics.com/store/rov/bluerov2/

Bru Cervantes Anaya. (n.d.). *Regulador Lineal Cuadrático Optimo*. Retrieved September 10, 2021, from https://es.scribd.com/document/400658573/Regulador-Lineal-Cuadratico-Optimo

Cárcel, P. A. (2020). *Modelado, simulación y control de pequeños vehículos submarinos no tripulados*.

Carlton, J. S. (2007). Marine Propellers and Propulsion. *Marine Propellers and Propulsion*. https://doi.org/10.1016/B978-0-7506-8150-6.X5000-1

Comienza la operación de búsqueda con AUV del submarino ARA San Juan. (n.d.). ElSnorkel. Retrieved August 26, 2021, from https://www.elsnorkel.com/2018/09/comienza-la-operacion-de-busqueda-con.html

Cross, C. (2020). *Mission to the Bottom of Lake Tahoe*. https://discourse.ros.org/t/ros2-embarks-on-a-mission-to-the-bottom-of-lake-tahoe/17139

Cummins, W. E. (1962). The Impulse Response Function and Ship Motion. *Report 1661, Department of the Navy, David W. Taylor Model Basin, Hydromechanics Laboratory, Research and Development Report, October 1962*. https://repository.tudelft.nl/islandora/object/uuid%3A222ffea6-85c2-4e34-ad69-bddfaa750081

Dive & Discover. (2021). https://divediscover.whoi.edu/

Dive & Discover - Expedition 17: Technology . (n.d.). 2021. Retrieved September 9, 2021, from https://divediscover.whoi.edu/expedition17/technology/

Documentation - ROS Wiki. (n.d.). Retrieved August 25, 2021, from http://wiki.ros.org/

Einarsson, E. M., & Lipenitis, A. (2020). *Model Predictive Control for the BlueROV2 Theory and Implementation*. https://www.et.aau.dk/

Ellepola, M., & Gustavino, I. (2020). *Underwater Acoustic Modem : Aquacomm Gen2 Reliable Communication*. https://www.dspcommgen2.com/

Enrique González Sancho. (2018). Diseño, modelización y simulación mecánica de un robot submarino controlado remotamente. In *Universidad Politécnica de Cartagena*. https://repositorio.upct.es/xmlui/bitstream/handle/10317/7592/tfm-gon-dis.pdf?sequence=1&isAllowed=y

Faltinsen, O. . (1993). Sea Loads on Ships and Offshore Structures. Cambridge University Press. *Cambridge University Press*, 340. https://books.google.com/books?hl=zh-CN&lr=&id=qZq4Rs2DZXoC&oi=fnd&pg=PP9&dq=Faltinsen+O.+Sea+loads+on+ships+and+offshore+structures.+Cambridge+University+Press%3B+1993.&ots=68ptYooUoq&sig=8nXbglfkXm3KIXi1E7uUkOqlmMc

Fossen, T. I. (2011). *HANDBOOK OF MARINE CRAFT HYDRODYNAMICS AND MOTION CONTROL Vademecum de Navium Motu Contra Aquas et de Motu Gubernando*. www.wiley.com.

Fossen, T. I. (2021). *Handbook of Marine Craft Hydrodynamics and Motion Control* (2nd ed.). Wiley. https://www.fossen.biz/wiley/

FreeCAD: Your own 3D parametric modeler. (n.d.). Retrieved August 25, 2021, from https://www.freecadweb.org/

Gazebo. (n.d.). Retrieved August 25, 2021, from http://gazebosim.org/

Jamet, S. (n.d.). *NEMOH-Presentation - LHEEA*. Retrieved September 11, 2021, from https://lheea.ec-nantes.fr/valorisation/logiciels-et-brevets/nemoh-presentation

Jorge Luis Lemus Ramos. (2018). *Sistema de navegación inercial para un AUV en presencia de corrientes marinas*. http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1815-59282018000200004

Juan José Villena, & Davi Moura. (2020, January 24). *¿Por qué gran parte de los océanos permanecen sin explorar?* https://www.tiempo.com/noticias/ciencia/por-que-oceanos-siguen-sin-explorar-ciencia.html

Lodovisi, C., Loreti, P., Bracciale, L., & Betti, S. (2018). Performance analysis of hybrid optical-acoustic AUV swarms for marine monitoring. *Future Internet*, *10*(7). https://doi.org/10.3390/FI10070065

Madgwick, S. O. H. (2010). *An efficient orientation filter for inertial and inertial/magnetic sensor arrays*.

Mahony, R., Hamel, T., Pflimlin, J.-M., Pflimlin Nonlinear, J.-M., & Mahony, R. (2008). Complementary Filters on the Special Orthogonal Group. *IEEE Transactions on Automatic Control, Institute of Electrical and Electronics Engineers*, *53*(5), 1203–1217. https://doi.org/10.1109/TAC.2008.923738ï

Manhães, M. (2021). *Unmanned Underwater Vehicle Simulator: Enabling the Simulation of Multi-Robot Underwater Missions with Gazebo*. https://doi.org/10.36288/roscon2018-900849

Masmitja, I., Gonzalez, J., Galarza, C., Gomariz, S., Aguzzi, J., & del Rio, J. (2018). New vectorial propulsion system and trajectory control designs for improved AUV mission autonomy. *Sensors (Switzerland)*, *18*(4). https://doi.org/10.3390/S18041241

MAVLink. (n.d.). Retrieved September 11, 2021, from https://mavlink.io/en/

MeshLab. (n.d.). Retrieved August 25, 2021, from https://www.meshlab.net/#description

Nido Robotics. (n.d.). Retrieved September 7, 2021, from https://www.nidorobotics.com/

Nido Robotics Sibiu Nano. (n.d.). Retrieved September 11, 2021, from https://geo-matching.com/rovs-remotely-operated-underwater-vehicles/sibiu-nano

Perez, T. (2005). *Ship motion control : course keeping and roll stabilisation using rudder and fins*. 300.

RAE. (n.d.). *submarino, submarina | Definición | Diccionario de la lengua española | RAE - ASALE*. Retrieved August 30, 2021, from https://dle.rae.es/submarino?m=form

Rauschenbach, M. M. M. M. and S. A. S. and M. V. and L. R. D. and T. (2016). *{UUV} Simulator: A Gazebo-based package for underwater intervention and multi-robot simulation*. IEEE. https://doi.org/10.1109/oceans.2016.7761080

Robot submarino - Endesa. (n.d.). Retrieved August 23, 2021, from https://www.endesa.com/es/prensa/sala-de-prensa/noticias/transicion-energetica/digitalizacion/endesa-lanza-un-robot-submarino-para-la-inspeccion-y-mantenimiento-de-sus-plantas

Robotics, B. (n.d.). *Ardusub*. Retrieved September 10, 2021, from https://www.ardusub.com/

ROBOTICS, N. (n.d.-a). *Manual de Operacion Sibiu Nano+*. https://drive.google.com/file/d/1D_-phPOzPbAuYERQWfV1VZXqXOH_guti/view?usp=sharing

ROBOTICS, N. (n.d.-b). *MANUAL ENSAMBLAJE SIBIU NANO PLUS*. https://www.manualslib.com/manual/1834056/Nido-Robotics-Sibiu-Nano.html?page=2#manual

ROBOTICS, N. (n.d.-c). *Sibiu Nano Plus*. Retrieved June 30, 2021, from https://es.nidorobotics.com/sibiu-nano-plus

Rodríguez, I. S. (2017). *Modelado de robot submarino mediante el simulador UWSim*.

Schofield, O. (2009). *The Scarlet Knight's Trans-Atlantic Challenge*. https://rucool.marine.rutgers.edu/atlantic/

Sen, D. (2000). A Study on Sensitivity of Maneuverability Performance on the Hydrodynamic Coefficients for Submerged Bodies. *Journal of Ship Research*, *44*(03), 186–196. https://doi.org/10.5957/JSR.2000.44.3.186

Shan, S., Hou, Z., & Wu, J. (2017). Linear Kalman Filter for Attitude Estimation from Angular Rate and a Single Vector Measurement. *Journal of Sensors*, *2017*. https://doi.org/10.1155/2017/9560108

SNAME. (1950). NOMENCLATURE FOR TREATING THE MOTION OF A SUBMERGED BODY THROUGH A FLUID. *T&R Bulletin*, *1–5*.

Takimoto, R. Y., Hirayama, T., & Kawaoka Takase, F. (2008). *UUV DEPTH MEASUREMENT USING CAMERA IMAGES*.

Tang, S., Ura, T., Nakatani, T., Thornton, B., & Jiang, T. (2009). Estimation of the hydrodynamic coefficients of the complex-shaped autonomous underwater vehicle TUNA-SAND. *Journal of Marine Science and Technology*, *14*(3), 373–386. https://doi.org/10.1007/S00773-009-0055-4

Tristan Perez, & Thor I. Fossen. (2011). *Motion Control of Marine Craft*. https://www.fossen.biz/home/papers/PerezFossen CRC Handbbook 2011.pdf

UUVSim RexROV. (n.d.). Retrieved September 11, 2021, from https://github.com/uuvsimulator/uuv_simulator/tree/master/uuv_descriptions/meshes

Welcome to Python.org. (n.d.). Retrieved August 25, 2021, from https://www.python.org/

Wu, C.-J. (2018). *6-DoF Modelling and Control of a Remotely Operated Vehicle*.

Yingyi Liu. (n.d.). *HAMS: An open-source computer program for the analysis of wave diffraction and radiation of three-dimensional floating or submerged structures*. Retrieved September 11, 2021, from https://github.com/YingyiLiu/HAMS

Yunier Valeriano-Medina, Garcia-Garcia, D., Jorge A. Portal-Linares, & Luis Hernández. (2015). Sistema de navegación basado en modelo dinámico no lineal de Vehículo Autónomo Sumergible. *RIELAC*, *36*, 83–97. https://www.researchgate.net/publication/284550684_Sistema_de_navegacion_basado_en_modelo_dinamico_no_lineal_de_Vehiculo_Autonomo_Sumergible