Trabajo Fin de Máster

Ingeniería en Electrónica, Robótica y Automática

Diseño del sistema de control para navegación autónoma 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, 2022

Proyecto Fin de Máster

Ingeniería en Electrónica, Robótica y Automática

Diseño del sistema de control para navegación autónoma 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, 2022

Trabajo Final de Máster: Diseño del sistema de control para navegación autónoma 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, 2022

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

“Toda tecnología tiene su esplendor y su lado oscuro
Forma parte de nuestro trabajo superar dichos problemas”

- Dicho Popular -

Tras dar por finalizado este Trabajo Final de Máster no puedo evitar acordarme todas aquellas personas que han hecho que este trabajo haya sido posible. A todas ellas les guardo un cariño especial y quiero expresarles mi más sincero agradecimiento.

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 por darme la oportunidad de realizar este TFM. Él guio mis primeros pasos indicándome los campos en los que avanzar y todo lo necesario para hacer de este un buen Trabajo Final de Máster. Orientándome en el transcurso de este, así como indicando las correcciones necesarias hasta darle la forma que ahora se presenta. A él le agradezco, además, su estimulo y disponibilidad hasta llegar a la finalización de este trabajo, proveyéndome de todo el material necesario para ello.

A mis compañeros y compañeras de máster, por sus consejos y reflexiones del tema elegido para este trabajo: Jaime, David, Mikel, Nicolás, Lucía, Borja, Mario, Isabel y todos aquellos a quienes no he nombrado. Sinceramente gracias.

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

Por último, quiero mencionar a mi familia (mis padres y mi hermano). No porque sean los menos importantes si no por todo lo contrario.

Quería hacer también una mención a este Máster en Robótica, Electrónica y Automática 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, 2022

Resumen

El presente Trabajo Final de Master realiza un estudio de distintas implementaciones de control en vehículos submarinos que operan con el firmware de Ardupilot, indicando las características y capacidades presentes en dicho software.

Con este objetivo se ha diseñado un sistema apoyado por USBL que permite utilizar el ROV Sibiu Nano + de la empresa Nido Robotics como AUV utilizando un sistema supervisable de puntos de interés indicados por un usuario en tierra.

Por último, se evalúan los desempeños de los distintos sistemas de control mediante la realización de pruebas en un ambiente controlado.

Abstract

This project analyses the implementation of different control systems in lightweight underwater vehicles operating with the firmware Ardupilot, making a complete description of their charasteristics and different capabilities.

A software aided by USBL has been designed to use the ROV Sibiu Nano + made by Nido Robotics as an AUV using a supervised system of waypoints indicated by an user in the Base Station.

Lastly the behaviour of each designed control system will be evaluated via real tests in a controlled environment.

Índice

Índice de Tablas

Índice de Figuras

Índice de Ilustraciones

Índice de Abreviaturas y Símbolos

AUV	Autonomous Underwater Vehicle
CFD	Computational Fluid Dinamics
HOV	Human Occupied Vehicle
IMU	Inertial Measurement Unit
LAUV	Light Autonomous Underwater Vehicle
NED	North East Down
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
TFM	Trabajo Final de Master
UUV	Unmanned Underwater Vehicles
GCS	Ground Control Station
INS	Inertial Navigation System
LLC	Low Level Controller
LBL	Long BaseLine
SBL	Short BaseLine
USBL	Ultra-Short BaseLine
GUI	Graphical User Interface
HAL	Hardware Abstraction Layer

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
Thread	Secuencia de tareas ejecutada por un sistema Operativo, permite realizar distintas tareas a la vez simplificando el diseño de una aplicación
Posix	Norma de la IEE que define la interfaz estándar de un sistema operativo y el entorno
Worst Case Timing	Tiempo máximo que una tarea tarda en ejecutarse en una plataforma hardware específica
Event Handler	Rutina que se ejecuta de forma asíncrona en reacción a un evento.

1 INTRODUCCIÓN

Este proyecto describe el desarrollo del sistema que gobierna el funcionamiento del vehículo submarino ligero Sibiu Nano +, ROV de la empresa Nido Robotics.

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

Este tipo de vehículos submarinos recibe también los nombres de ROUV (Remote Operated Underwater Vehicle) o UUV (Unmanned Underwater Vehicles) evolucionando hasta AUV (Autonomous Underwater Vehicle) cuando no requieren ningún tipo de interacción humana.

El proyecto Indalo en el que participa el Departamento de Ingeniería de Sistemas y Automática hace uso de robots acuáticos en combinación con otros sistemas para monitorizar y mapear diversas variables ambientales de interés en masas de agua como pantanos o embalses. Se incluye también un análisis de los sensores y parámetros ambientales más importantes usados en el sector.

En este contexto, una de las partes necesarias y motivación de este trabajo, es el estudio y diseño del sistema de control del UUV Sibiu Nano +. Siguiendo así con la línea de investigación de trabajos anteriores. [1]

Por ello este TFM realiza un análisis del estado del arte del control de vehículos submarinos haciendo detalle en el sistema presente en el Sibiu Nano +. Se analizan las estrategias más utilizadas en el sector, así como estrategias en desarrollo y futuras.

A continuación, se analiza el sistema presente en el Sibiu Nano +, así como sus características únicas, haciendo hincapié en el tiempo de respuesta, interacción con el usuario y calidad del control.

El grueso del trabajo se centra en el desarrollo de varias estrategias de control a partir de los sistemas que componen el submarino, 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 o esta información. 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 Máster es el diseño del control del vehículo submarino ligero Sibiu Nano +, estudiando diversas estrategias y validando su correcto funcionamiento mediante pruebas en ambientes controlados.

Los objetivos específicos que se persiguen son;

- Respecto al submarino: Debe comprenderse el funcionamiento de este y mantenerse en la línea de trabajo presente.
- Respecto al control: El submarino debe ser capaz de desplazarse por sí solo hasta unas coordenadas elegidas, sin interacción humana más allá de la indicación de estas
- Respecto a la comunicación: El usuario debe tener en todo momento conocimiento del estado del submarino.

1.2 Metodología

Para el desarrollo del trabajo se parte de los conocimientos respecto al Sibiu Nano + y el entorno submarino desarrollados en trabajos anteriores que aparecen en la bibliografía.

En primer lugar, se realiza una toma de contacto con el Submarino. Utilizándolo como ROV mediante operación manual escribiendo un manual de buenas prácticas para la realización de pruebas.

En segundo lugar, se realiza un estudio del funcionamiento del sistema presente en el submarino con el objetivo de almacenar las lecturas del estado del submarino en todo momento.

En tercer lugar, se estudia en profundidad el sistema presente en el submarino, elaborando una lista de estrategias de control aplicables.

Así este proyecto queda dividido en tres partes: Análisis de los componentes del sistema y buenas prácticas, características del sistema presente en el Sibiu Nano + y diseño del sistema de control.

El sistema presente en Sibiu Nano + es de código abierto y presenta guías de programación. De la misma forma todo el código desarrollado en este TFM estará disponible en la web para cualquier tipo de consulta.

Los lenguajes de programación usados son C++ y Python con el objetivo de hacer un sistema de respuesta rápida en el ciclo de control y mantener un rápido desarrollo en el resto de procesos.

Por último, se comprobará que el control realizado cumple con todos los objetivos previamente descritos.

2 ESTADO DEL ARTE

2.1 Impacto social, económico y ecológico del agua y los UUV

El agua, además de ser el líquido más abundante del planeta, representa el recurso natural más importante y base de toda forma de vida. Es un bien común fundamental para la naturaleza y, en particular, para el ser humano. Además de cumplir necesidades básicas como beber, regar o para uso industrial, otorga un gran número de beneficios como la conservación de la biodiversidad, la pesca, el ocio y el turismo.

Cumple con un gran número de funciones clave, siendo un recurso irremplazable [2].

- El agua interviene en la mayor parte de los procesos metabólicos de los seres vivos, es un recurso indispensable para la vida y resulta imprescindible para el desarrollo de la agricultura y la ganadería. El agua constituye más del 80% del cuerpo de la mayoría de los organismos y el 70% del ser humano[3].
- El agua es un recurso industrial importantísimo usado como materia prima, como disolvente y como refrigerante.
- Se denomina energía hidráulica a la que se obtiene del aprovechamiento de las energías cinética y potencial de la corriente de los ríos, saltos de agua y mareas.
- En la naturaleza, el agua sufre una serie de cambios en los que adquiere y cede energía en distintas etapas de un recorrido circular denominado ciclo hidrológico.
- Tiene un sentido estratégico de aprovechamiento para la agricultura, ganadería, la pesca y acuicultura, incluyendo la industria de transformación de los productos o la ligada al turismo y ocio.
- Es un recurso limitado. Aunque el agua cubre el 71% de la superficie del planeta, el agua dulce solo forma el 2.5%, y de estas solo el 0.007% se encuentra disponible para uso humano. [4].

El agua, es un recurso escaso en muchas partes del planeta, siendo este problema acentuado por los episodios de sequía que provoca la emergencia climática actual. Además, En mundo en desarrollo las necesidades de agua están aumentando, agotando las reservas disponibles de agua a un ritmo alarmante y poniendo en peligro un bien fundamental para el futuro de la humanidad.

La falta de agua obliga a buscar nuevas fuentes, requiriendo de tiempo y dinero. El consumo de agua en Canarias está directamente relacionado con el consumo de energía, ya que gran parte de ella se produce en plantas desalinizadoras. Por tanto, ahorrar agua es ahorrar energía. Realizar una buena gestión de este recurso puede ayudar a reducir una gran cantidad de costes operativos. [5]

Este problema de la sequía se está acentuando mucho en los últimos años, dejando imágenes impactantes como las que aparecen en las ilustraciones 1 y 2.

Ilustración 1: Cerro Prieto 2015-2022

Fuente: elpais.com [6]

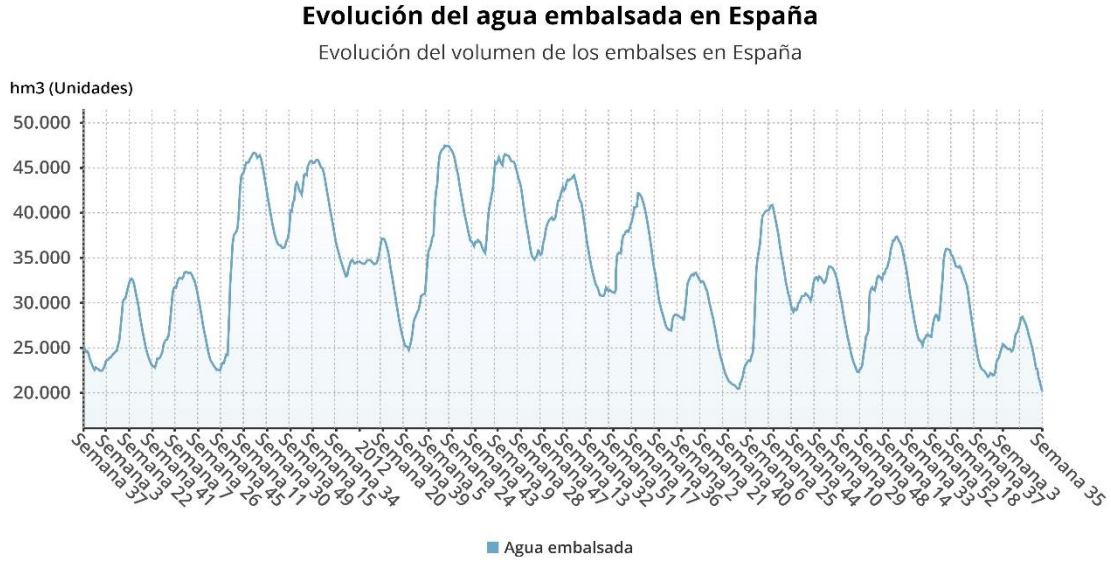
Ilustración 2: Doñana 2021-2022

Fuente: eldiario.es [7]

El ciclo del agua es parte consustancial del medio ambiente de la tierra y es tenido en cuenta en el ecosistema terrestre en la gestión hídrica. Según la UNESCO vertemos al medio ambiente el 80% de las aguas residuales sin tratamiento, aumentando la contaminación. [8]

Si no se pone remedio a estos problemas el agua potable apta para el consumo puede agotarse. Por ello es importante protegerla, cuidarla, mantenerla y gestionarla más allá de los intereses económicos. Todos los actores públicos y privados, incluidas las empresas y la sociedad civil organizada, han de preocuparse por cumplir los objetivos de desarrollo sostenible relacionados en manifiestos y normativa de aplicación, evitando toda influencia nociva sobre los ríos lagos y océanos; haciendo un uso racional de las reservas existentes y llevando a cabo un consumo responsable.

Figura 1: Evolución del agua embalsada en España



Fuente: epdata.es [9]

2.1.1 El valor del Agua para el Desarrollo Sostenible

2.1.1.1 La agenda 2030 de Naciones Unidas

En esta declaración universal se reafirman los compromisos sobre el derecho humano al agua potable y al saneamiento, en una situación del mundo actual donde la escasez del agua es un hecho acrecentado por el episodio de emergencia climática conocido y la sequía asociada.

El desarrollo social y económico racional depende y ha de ser consciente de la gestión sostenible de los recursos naturales de nuestro planeta. Por ello, desde un punto de vista multiactor se ha de preservar y utilizar sosteniblemente los océanos y los mares, los recursos de agua dulce y los bosques, las montañas y las zonas áridas, y a proteger la diversidad biológica, los ecosistemas y la flora y fauna silvestres.

En este sentido, la consideración del recurso agua está presente en los ODS[10]:

2. Hambre cero. Considera el agua como recurso esencial en el ecosistema para asegurar el alimento a la población del mundo.

3. Garantizar una vida sana y promover el bienestar para todos. Con la necesidad de aumento del acceso al agua limpia y el saneamiento, evitando la transmisión de enfermedades debidas al agua.

6. Garantizar la disponibilidad de agua y su gestión sostenible y el saneamiento para todos. Con todas sus metas asegurando el agua como recurso, al tratar de:

6.1 De aquí a 2030, lograr el acceso universal y equitativo al agua potable a un precio asequible para todos

6.2 De aquí a 2030, lograr el acceso a servicios de saneamiento e higiene adecuados y equitativos para todos y poner fin a la defecación al aire libre, prestando especial atención a las necesidades de las mujeres y las niñas y las personas en situaciones de vulnerabilidad

6.3 De aquí a 2030, mejorar la calidad del agua reduciendo la contaminación, eliminando el vertimiento y minimizando la emisión de productos químicos y materiales peligrosos, reduciendo a la mitad el porcentaje de aguas residuales sin tratar y aumentando considerablemente el reciclado y la reutilización sin riesgos a nivel mundial

6.4 De aquí a 2030, aumentar considerablemente el uso eficiente de los recursos hídricos en todos los sectores y asegurar la sostenibilidad de la extracción y el abastecimiento de agua dulce para hacer frente a la escasez de agua y reducir considerablemente el número de personas que sufren falta de agua

6.5 De aquí a 2030, implementar la gestión integrada de los recursos hídricos a todos los niveles, incluso mediante la cooperación transfronteriza, según proceda

6.6 De aquí a 2020, proteger y restablecer los ecosistemas relacionados con el agua, incluidos los bosques, las montañas, los humedales, los ríos, los acuíferos y los lagos

6.a De aquí a 2030, ampliar la cooperación internacional y el apoyo prestado a los países en desarrollo para la creación de capacidad en actividades y programas relativos al agua y el saneamiento, como los de captación de agua, desalinización, uso eficiente de los recursos hídricos, tratamiento de aguas residuales, reciclado y tecnologías de reutilización

6.b Apoyar y fortalecer la participación de las comunidades locales en la mejora de la gestión del agua y el saneamiento

11. Lograr que las ciudades y los asentamientos humanos sean inclusivos, seguros, resilientes y sostenibles. A tratar de asegurar de reducir significativamente el número de muertes causadas por los desastres, incluidos los relacionados con el agua,

12. Garantizar modalidades de consumo y producción sostenibles. Al asegurar que las prácticas de producción y consumo no aporte externalidades negativas al agua.

14. Conservar y utilizar en forma sostenible los océanos, los mares y los recursos marinos para el desarrollo sostenible. Con interés, entre otras, de la meta 12.2 relativa a gestionar y proteger sosteniblemente los ecosistemas marinos y costeros para evitar efectos adversos importantes, incluso fortaleciendo su resiliencia, y adoptar medidas para restaurarlos a fin de restablecer la salud y la productividad de los océanos.

15. Gestionar y proteger sosteniblemente los ecosistemas marinos y costeros para evitar efectos adversos importantes, incluso fortaleciendo su resiliencia, y adoptar medidas para restaurarlos a fin de restablecer la salud y la productividad de los océanos. Donde el agua sigue siendo un recurso esencial.

16.1. Reducir significativamente toda forma de violencia y las correspondientes tasas de mortalidad en todo el mundo.

17.6 y 17.17. Posibilitar la cooperación regional e internacional en materia de ciencia, tecnología e innovación, así como fortalecer y promover la constitución de alianzas eficaces en al esfera público-privada, aprovechando la experiencia y colaboración de las mismas.

Los múltiples actores que se relacionan con el agua encuentran un denominador común en su uso sostenible, utilizando este lenguaje universal de identificación a través de la Agenda 2030 y sus ODS. La Agenda 2030 "Transformar nuestro mundo para el Desarrollo Sostenible", aprobada por las Naciones Unidas el 25 de septiembre de 2015, es la hoja de ruta global para hacer frente a los grandes retos de la humanidad. Una Agenda que insta a proteger el medio ambiente, hacer frente al cambio climático, desarrollar economías dinámicas y inclusivas y garantizar el trabajo digno para todos. Para alcanzar los ODS, la ONU alienta a desarrollar respuestas nacionales y subnacionales, con la participación multiactor, multitema y multinivel en ello, en torno a las metas de sus 17 ODS.

Este TFM tiene contenidos que aportan componentes de gran significado a los ODS, sobre todo los relacionados con el ODS n.º 6, 9, 14, 16 y 17; concretada en apoyo a la consecución de las siguientes metas:

Meta 6.6. Al aportar una tecnología utilizable para proteger y restablecer los ecosistemas relacionados con el agua, incluidos los bosques, las montañas, los humedales, los ríos, los acuíferos y los lagos

Meta 9.5. Porque a través del programa departamental de la ETSI se contribuye a aumentar la investigación científica y mejorar la capacidad tecnológica de estos UUV, fomentando la innovación y aumentando considerablemente, el número de personas que trabajan en investigación y desarrollo por millón de habitantes y sus gastos asociados.

Meta 14.2. Porque con los UUV podrán contribuir a gestionar y proteger sosteniblemente los ecosistemas marinos y costeros para evitar efectos adversos importantes, incluso fortaleciendo su resiliencia, y adoptar medidas para restaurarlos a fin de restablecer la salud y la productividad de los océanos.

Meta 16.1. Al utilizarse los UUV como tecnología para reducir significativamente toda forma de violencia y las correspondientes tasas de mortalidad en todo el mundo.

Meta16.7 Al garantizar la adopción en todos los niveles de decisiones inclusivas, participativas y representativas que respondan a las necesidades donde puede intervenir estos UUV.

Metas 17.6. Al posibilitar esta investigación la cooperación regional e internacional en materia de ciencia, tecnología e innovación, así como su acceso para su uso en distintos ámbitos para el Desarrollo Sostenible.

Meta 17.17. Al fortalecer y promover la constitución de alianzas eficaces en al esfera público-privada con la empresa fabricante del Sibiu Nano +, aprovechando la experiencia y colaboración de las mismas.

2.1.1.1 Referencias a normativa vigente

Existen una gran cantidad de normas que se aplican en este contexto a varios niveles, desde normas relativas al agua hasta de aplicación de la tecnología que se pueden encontrar en el Marco de Agua de la Unión Europea [11]:

- El derecho en tratados internacionales es amplio, en relación con el acceso a agua potable, con obligaciones a los Estados a que garanticen a todas las personas el acceso a una cantidad suficiente de agua potable para el uso personal y doméstico, que comprende el consumo, el saneamiento, el lavado de ropa, la preparación de alimentos y la higiene personal y doméstica.
- Igualmente es amplia la normativa específica del agua a nivel de la Unión Europea, el Estado español, las comunidades autónomas y de los gobiernos locales.

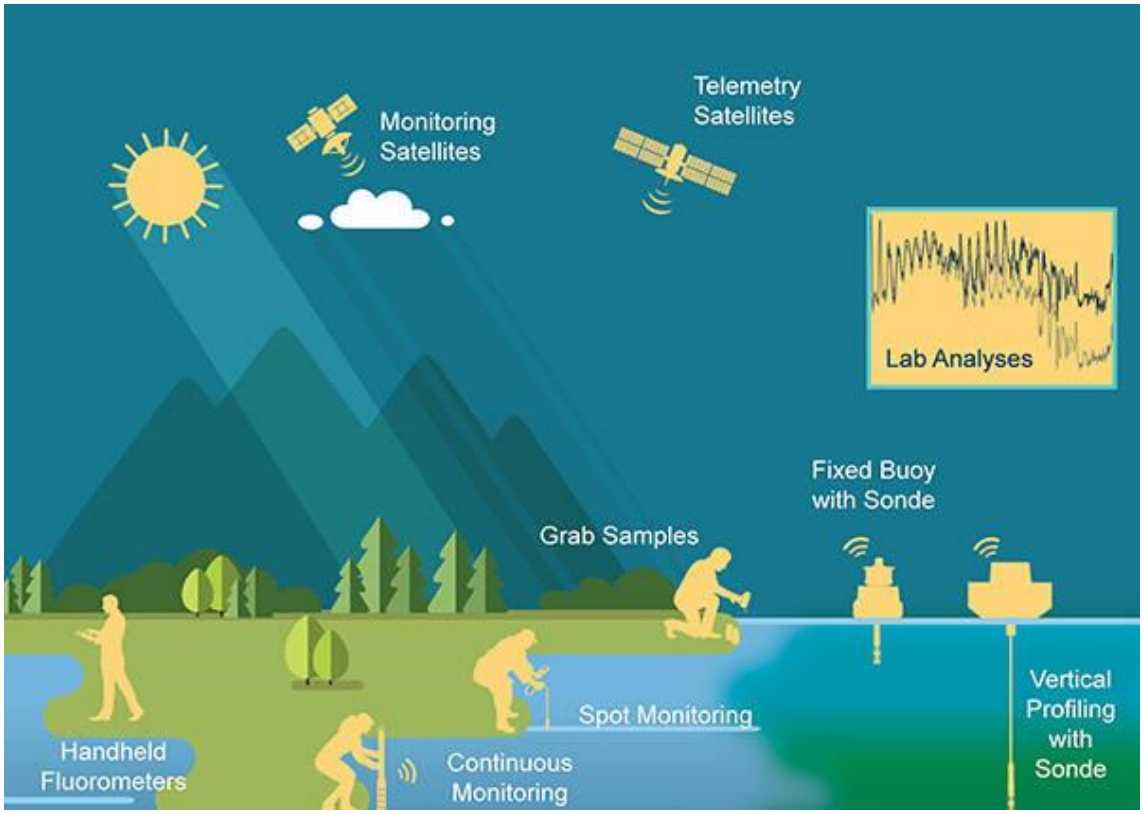
Respecto al empleo de la normativa de UUV, no existe una normativa clara, la más cercana es la Convención de las Naciones Unidas sobre el Derecho del Mar (UNCLOS)[12]. Esta normativa divide el mar en zonas dependiendo de su distancia a la tierra en ‘mar territorial’ (12 millas náuticas), ’zona contigua’ (24), ‘zona económica exclusiva’ (200) y ‘plataforma continental‘.

Esta normativa permite en sus artículos 19 y 20, el paso de naves extranjeras si no son perjudiciales para la paz, orden o seguridad del territorio. Dejando vía abierta a los UUV para realizar prácticamente cualquier operación.

2.1.2 El impacto de los UUV en el Agua

La gestión de recursos hídricos busca conocer el estado del agua con el objetivo de prevenir problemas como la salinización, contaminación y sequía de embalses, lagunas o lagos. Para ello se realizan análisis de las características del agua, aplicando soluciones químicas mientras se mantiene la salubridad del agua necesaria para su uso humano.

Figura 2: Toma de muestras acuáticas medioambientales



Fuente: ysi water blog [13]

Todos los muestreos tradicionales recogen las muestras en un único punto. Sólo la boya perfiladora puede alcanzar distintas coordenadas transmitiendo los datos a través de internet (satélites o torres de comunicación). Los satélites de monitorización pueden cubrir grandes masas de agua, pero tienen una resolución que no permite tratar con pequeñas masas de agua (embalses, playas)

Por lo que, si el objetivo es obtener datos de alta resolución en una superficie como un lago, tomar muestras en zonas puntuales o utilizar satélites no son soluciones válidas.

Una solución ante esto sería desplegar una gran cantidad de sensores, limitada únicamente por el coste de estos y en el caso de ser baratos, en la calidad de las muestras.

La más económica se basa en el uso de flotas de vehículos de superficie (USV/ASV) y submarinos (UUV/AUV). Con ella, además de la monitorización de cuestiones y recursos medioambientales, este tipo de vehículos permite realizar tareas muy variadas como inspección e intervención; estudios oceanográficos como batimetrías, cartografía oceánica, muestreo de océanos o inspección o estudio del suelo marino; Proteger habitas y gestionarlas creando mapas de la fauna y flora marina; y tareas científicas y militares como inteligencia, reconocimiento, vigilancia y adquisición de objetivos, neutralización de minas, seguridad e inspección.

2.2 Antecedentes

Los vehículos submarinos presentan los mismos problemas que el resto de robots de servicio (heterogeneidad en el hardware, incertidumbre de los sistemas de medida, complejidad del software, etc) sumados a los problemas de ingeniería debidos al entorno no estructurado y hostil del fondo marino en el que operan (visibilidad nula, incertidumbre en cuanto a posición y velocidad, restricciones energéticas). Sin embargo, este tipo de vehículos resultan de gran interés ya que presentan innumerables ventajas respecto a las tecnologías tradicionales, reduciendo el riesgo laboral de submarinistas, costes y facilitando tiempos de respuesta, superando retos científicos y otros problemas asociados. [11]

En trabajos anteriores "*Modelado, simulación y control de un vehículo submarino ligero*" [1] se han tratado aspectos técnicos como la hidrostática e hidrodinámica, el sistema de control, y características propias del medio submarino como el movimiento en un espacio tridimensional con perturbaciones dinámicas importantes.

Desde el primer vehículo submarino no tripulado, "POODLE" en 1954 [14], el número de vehículos submarinos ha sufrido un crecimiento exponencial. Esto ha dado lugar a una gran cantidad de nuevos diseños y aplicaciones, evolucionando en distintas ramas de la robótica y ganando nuevas habilidades como la capacidad de portar cargas, herramientas como pinzas o soldadores y sumergirse hasta varios cientos de metros entre otras.

Esto dificulta realizar una división única de este tipo de vehículos. Una primera división se puede realizar teniendo en cuenta su autonomía, diferenciándolos entre "ROUV" y "AUV" como se indica en el punto 3.2.1.

Los ROUV son una especie de drones subacuáticos que dependen de un piloto para desplazarse por el agua. 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.

Los AUV por otro lado son vehículos completamente autónomos que no requieren ningún tipo de interacción con un operador. Son un sistema único, con baterías y CPU, capaces de resolver los problemas de planificación y control sin intervención humana y capaces de obtener datos y enviarlos remotamente a un servidor. 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 [15]. 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 estudio de criaturas marítimas y su ecosistema [16].

En los últimos años con el objetivo de desarrollar y promover este tipo de tecnologías, ha aparecido un gran número competiciones de estos vehículos acuáticos que proponen retos científicos y soluciones ante ellos. Entre las más famosas encontramos algunas como la Mate ROV, Seaperch, Robosub, TEKNOFEST, Unmanned Underwater Systems Competition, ROBOTX y ROSCON.

2.2.1 Tipos de UUV

Ilustración 3: Tipos de UUV

Fuente: Rui Lima [17]

Existe una gran cantidad de vehículos submarinos, pudiendo realizar diferentes clasificaciones. La clasificación más usada es en cuanto a su tamaño, potencia o trabajo, donde aparecen:

- ROV
    - Mini/Micro: ROV muy pequeños de entre 2-15 kg principalmente usados para acceder a zonas donde humanos no pueden y realizar tareas de observación.

Ilustración 4: VideoRay // Observer

Fuente: MDM Marine [18]

Fuente: Aditech [19]

- Inspección/Trabajo: Pueden realizar tareas de intervención, En muchos casos poseen actuadores y sumergirse hasta 400 metros.

Ilustración 5: Sibiu Pro // Bluerov2

Fuente: Nido Robotics [20]

Fuente: BlueRobotics [21]

- Pesados: Pueden poseer hasta 500 caballos de propulsión, varios manipuladores y operar a profundidades de hasta 6000 metros.

Ilustración 6: Cuttlet // Oceaneering

Fuente: T. Ahmad y J. Iqbal [22]

Fuente: Oceaneering [23]

- ASV
    - Ligeros: Son móviles y se pueden usar para una gran multitud de aplicaciones como monitorización de recursos o militares

Ilustración 7: Oceanserver Iver 2 // MEPUS ARV 150

Fuente: Utm.csic.es [24]

Fuente: Nauticexpo [25]

    - Gran escala: Pueden llevar una gran cantidad de equipamiento y suelen utilizarse para misiones en el fondo oceánico operando a profundidades superiores a los 5000m

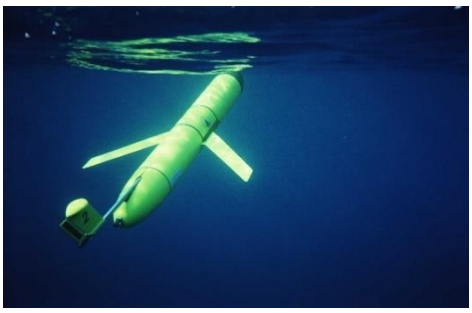
Ilustración 8: Torpedo // Urashima

Fuente: nauticexpo [26]

Fuente: nauticexpo [27]

    - Gliders: Poseen un método de propulsión eficiente permitiéndoles realizar misiones transoceánicas de gran distancia, realizando emersiones periódicas para enviar datos por satélite.

Ilustración 9: WHOI // Scarlet Knight



Fuente: Fuente: Dive & Discover [28]

O. Schofield [29]

2.3 Software

Para un desarrollador es de interés que el sistema a desarrollar se mantenga simple, de tal forma que se puedan crear algoritmos accediendo al estado del arte sin necesidad de enfrentarse al problema de la integración ni ser un experto en software

El software robótico siempre tiene que enfrentarse a la complejidad inherente de las actividades concurrentes incluyendo sincronización, el despliegue de componentes software en ordenadores entrelazados, una gran cantidad de plataformas, sistemas operativos y lenguajes de programación. Todo esto garantizando la seguridad de los hilos sin ignorar los requerimientos de ancho de banda y sin definir una arquitectura robótica concreta.

Así aparecen los framework. Softwares que proveen un cierto nivel de transparencia mientras resuelven problemas de middleware y sincronización. Así proveen una gran cantidad de drivers y componentes para un amplio grupo de sensores y plataformas.

El patrón en la robótica es el uso de componentes software. El sistema se encuentra dividido en módulos con un razonable nivel de granularidad y con conexión dinámica, permitiendo realizar cambios en las conexiones durante la ejecución de tal forma que se pueden configurar los flujos de control y datos. No existe ningún mecanismo, salvo la disciplina del programador, que prevenga que cualquier objeto que forma parte de un módulo envíe mensajes a otros módulos.

Los módulos deben evitar utilizar una gran cantidad de enlaces y soportar enlaces pobres pero con una rigurosa y estandarizada interfaz. Además, deben usar procesos asíncronos siempre que sea posible, comunicados de la forma más simple posible y su estructura interna debe poder seguir distintos diseños, poseyendo pocas limitaciones estructurales. [11] [30]

2.3.1 Técnicas de desarrollo

El desarrollo e integración del software es un elemento crucial que establece una separación entre la elaboración de prototipos y su fabricación en serie. Desde el punto de vista de un constructor de aplicación, el sistema debe estar compuesto de componentes estándar y reutilizables y un framework capaz de garantizar de forma clara la estructura y consistencia de las interfaces para un fácil ensamblado de los componentes verificados. Esta práctica es crucial no solo para prevenir la reescritura de código, si no para garantizar su calidad, estabilidad y robustez [31].

Una posible clasificación de técnicas de diseño de software puede ser:

- Librerías de algoritmos: Caracterizadas por la invocación de algoritmos proporcionados por la librería. El usuario debe proporcionar el software para el resto del sistema, como drivers, comunicaciones, gestión de la concurrencia, etc. Entre ellas se encuentran algunas como OpenSLAM e Interpora.
- Toolkits y middlewares robóticos: Amplían el soporte proporcionado por la categoría anterior con middleware de comunicación para conseguir distribución y modularidad. Las aplicaciones se construyen como un conjunto de binarios heterogéneos. Entre ellos se encuentran algunos como ROS y Player/Stage.
- Frameworks basados en componentes: Proporcionan el mismo nivel de soporte que los middleware robóticos, pero intentan superar las limitaciones de estos mediante el empleo de componentes software: artefactos que especifican qué servicios proporcionan y requieren a través de sus puertos, que son los únicos canales de comunicación permitidos. Algunos ejemplos son OROCOS, ORCA2, ROBOCOMP, Geno o SmartSoft.

2.3.2 Modelo arquitectónico de control

Muchas arquitecturas de control proponen converger a una estructura similar que busca el uso de paquetes software reusables y modularizados como módulos de tareas que se enlazan para ser predecibles y reactivos [31].

La estrategia de control se suele dividir en 3 tipos, dependiendo de la filosofía.

- Deliberativa: basada en planificación y en un modelo de mundo. Permite razonar y hacer predicciones del entorno. Los datos de los sensores se aplican a un modelo de mundo (bottom-up), el cual se utiliza para planear nuevas acciones para los actuadores (top-down). Este estrategia presenta la desventaja de que en un entorno altamente dinámico existe un delay de tiempo debido al procesamiento.
- Reactiva: las misiones se describen como una serie de fases con una serie de comportamientos designados. Los comportamientos continuamente reaccionan a la situación interpretada por el sistema de percepción, El comportamiento global del sistema surge de la combinación de estos subcomportamientos.

  Dado que cada comportamiento sigue su propia meta pueden existir interferencias entre ellos, haciendo al robot impredecible. Para solventar este problema los comportamientos suelen funcionar en paralelo teniendo unos prioridad frente a otros.
- Híbrida: Normalmente están formados por 3 capas, la deliberativa basada en planificación, la capa de control de la ejecución y una capa funcional reactiva.

  Así, las misiones se construyen secuenciando procesos en tiempo real y definiendo funciones que procesan eventos.

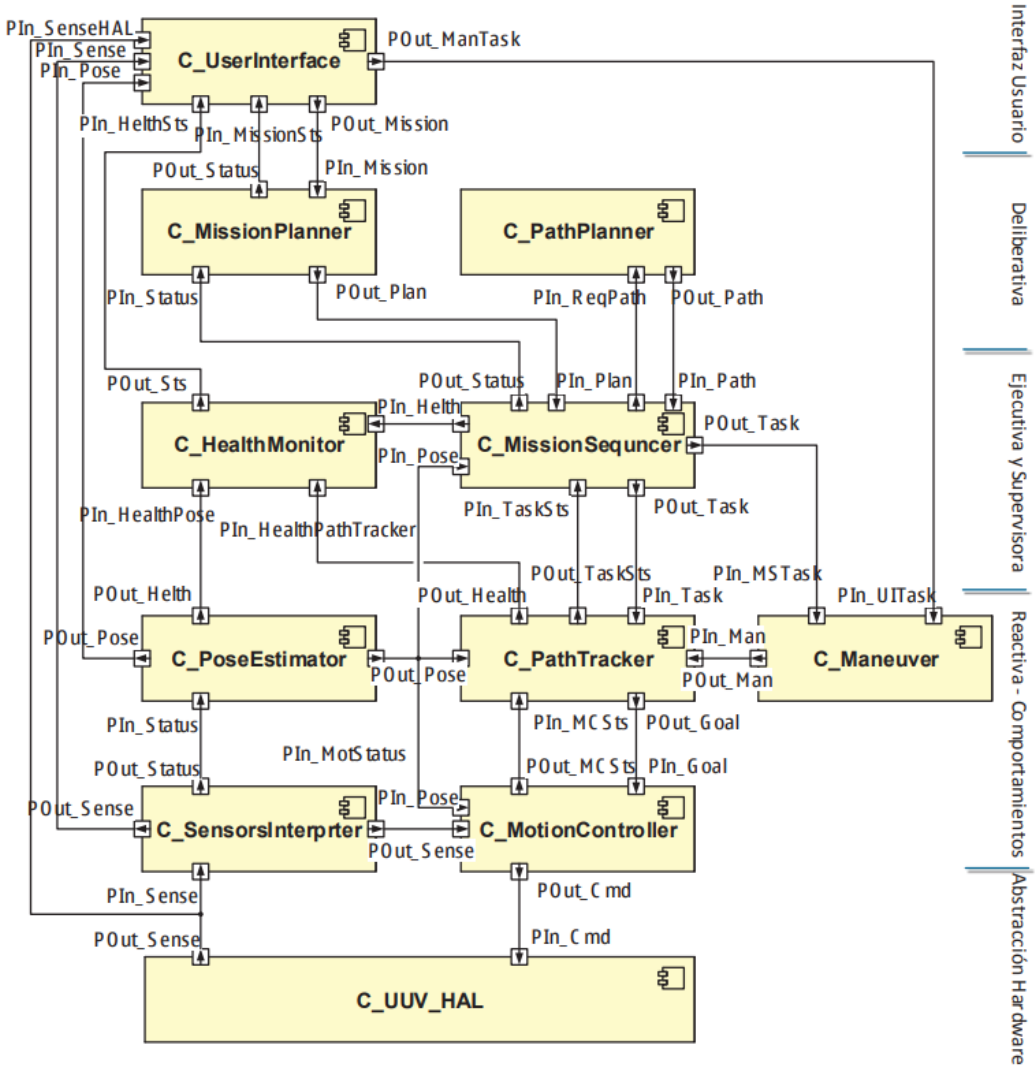
2.3.3 Modelo arquitectónico de la aplicación

Una vez definidos todos los módulos que componen el sistema es necesario establecer las relaciones entre estos. Un acercamiento típico divide el sistema en capas, cada una con una función especial y una frecuencia de ejecución distinta haciendo un uso eficiente de los recursos disponibles. [31]

- Capa de ejecución: Contiene los grupos de sensores y actuadores, es la interfaz entre el hardware y el software del vehículo (Hardware Abstraction Layer).
- Capa de control: Contiene los controladores de bajo nivel del vehículo. Está a cargo del control no lineal del vehículo y actuadores externos. En sistemas complejos puede contener un controlador de aprendizaje adaptativo.
- Capa de planificación: Está a cargo del control de alto nivel del vehículo durante la misión, es responsable de la planificación de la misión, ejecución y supervisión. La misión se descompone por el planificador supervisor en una serie de submetas. Cada submeta tiene un supervisor de la tarea responsable de actuar sobre los módulos de tal forma que estas se cumplan. Estos módulos deben estar diseñados para cumplir una tarea bien definida y proporcionar una solución sólida. Deben leer los datos de los sensores, de otros módulos y usar la función de procesamiento de tareas para computar las salidas para el resto de módulos o el supervisor.

  Las submetas pueden ser muy variadas, entre ellas se incluyen: atracar/acoplamiento, vigilancia, búsqueda, posicionamiento dinámico, navegación, repulsión, construcción de mapas locales, seguimiento de objetivos, análisis y almacenamiento de datos, maniobrabilidad, captura de imagen, etc.
- Capa de decisión: Indica la misión a realizar o el objetivo a cumplir por el vehículo. En muchos casos está formada por un operario especializado (HITL) y una interfaz de usuario. Con los avances de la tecnología están apareciendo alternativas con algoritmos definidos que buscan cumplir un objetivo específico o redes neuronales.

Figura 3: Ejemplo Arquitectura de un Vehículo acuático



Fuente: AEGIR (Francisco Ortiz et al) [11]

2.4 El medio acuático

2.4.1 Características del medio submarino

El agua es un medio 800 veces más denso que el aire, conductor de la electricidad y un gran número de propiedades únicas. Esto provoca que un gran número de los componentes hardware y tecnologías aplicables a vehículos de superficie o aéreos no sean aplicables para vehículos submarinos.

2.4.1.1 Ondas electromagnéticas

La propagación de una señal en el agua es diferente ya sea agua dulce o salada, superficial o subterránea. Sin embargo, en todas ellas se caracteriza por una gran atenuación de la señal., Tomando por ejemplo el experimento llevado a cabo por María de la Vega [32] en agua dulce superficial. Se observa:

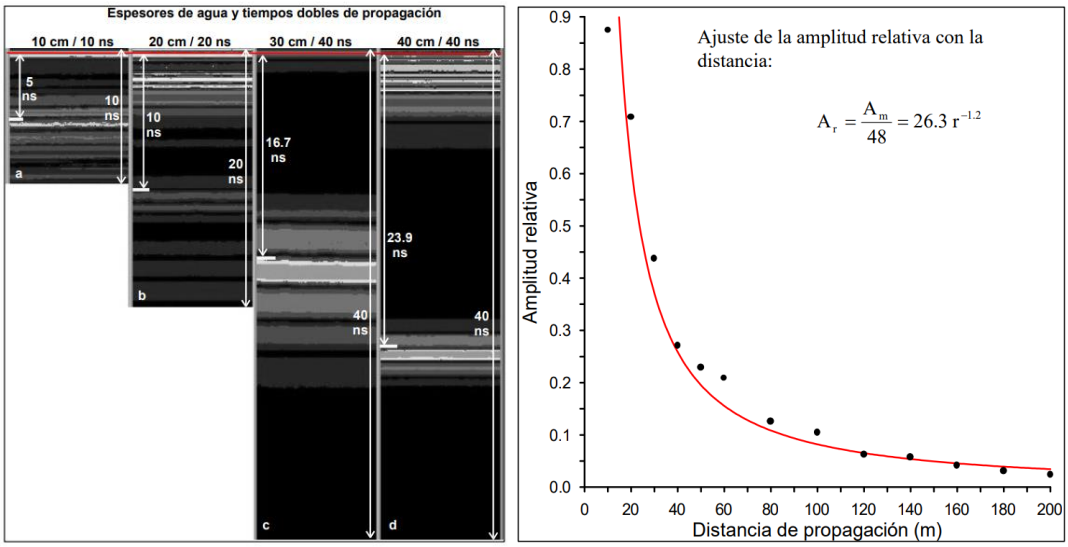
En el experimento la muestra de agua presenta una constante dieléctrica de 79, un factor de atenuación de 8.5 cm^{-1}, una conductividad de 400mS/m. Con estos datos se obtiene que la velocidad de una onda

de 1GHz es de 3.3293 cm/ns, y un factor de atenuación de 8.48 m^{-1}.

Estos números dan como resultado que la onda se verá atenuada en un 63% a una distancia entre 5cm y 13cm.

Para una misma antena, el aire tiene un factor de atenuación de 1.2. Obteniendo la misma atenuación para una distancia de 26.3m.

Figura 4: Propagación señal electromagnética en el Agua/Aire



Fuente: María de la Vega [32]

En la teoría de comunicaciones tradicional, la calidad de la señal se suele medir mediante la SNR para señales analógicas, y la tasa de error de bits para las señales digitales.

2.4.1.2 Ondas acústicas

A la vista de estos datos es de esperar que en el Agua la SNR de una señal sea muy pequeña (no se diferencia del ruido). Sin embargo, para señales acústicas de baja frecuencia, en concreto en torno a 10 kHz, la atenuación presente no es tan grande. Unavlab [33], realizó una serie de experimentos y dispositivos a base de altavoces e hidrófonos para evaluar este comportamiento. La SNR de esta señal en el agua se puede calcular mediante la siguiente ecuación

$$10^{\left(-k\cdot\frac{\log(r)}{10\cdot\log(10)}-0.0001\alpha_e\cdot r\right)} = P_N \cdot SNR/P_{TX}$$

Fuente: Unavlab [33]

Siendo 'r' la distancia, 'k' el coeficiente de la forma de onda (1 para cilíndrica, 2 para esférica) 'α_e' el coeficiente de absorción del medio (típicamente 3.48 dm/km), P_{TX} la potencia (presión) de la señal y P_N la potencia (presión) del ruido.

Con estos datos, una señal teóricamente podría alcanzar más de 10km de distancia pese a tener una baja tasa de transferencia de datos (70-300 bps).

En la vida real, esta distancia se ve decrementada en gran medida, debido a ciertos factores. Por ejemplo, la señal puede rebotar en una pared o encontrarse un obstáculo físico.

2.4.1.3 Velocidad del sonido

La velocidad del sonido en el aire es de 331 m/s. En el agua la velocidad del sonido depende de 3 parámetros: la temperatura, la presión hidrostática y la salinidad. Y en la mayoría de masas de agua,

estos parámetros no tienen valores uniformes.

En el agua la temperatura puede variar entre -2 y 40º, 0-42 PSU para salinidad y 1 a 1100 bares para la presión.

Estos valores pueden hacer variar la velocidad del sonido desde 1400m/s (en agua fría, limpia y a bajas presiones) a 1700m/s (en agua templada, salada y a altas presiones) [33].

2.4.1.4 Cálculo de la Profundidad

Una aproximación de la profundidad del vehículo se puede obtener mediante la ecuación

$$P = \rho g h$$

Donde 'P' es la presión, 'ρ' la densidad del agua, 'g' la gravedad y 'h' la profundidad.

Esta ecuación se basa en la hipótesis de que el agua es un medio homogéneo y una aceleración de la gravedad homogénea, así como que el agua es un medio no compresible, donde el error no suele ser superior al 0.7% si la medida de la presión superficial se tiene en cuenta. Sin embargo, existen métodos más exactos.

En la vida real, el agua es compresible, usualmente con la profundidad, aumentando su densidad y la temperatura y salinidad no son homogéneas.

Para realizar un cálculo más exacto del efecto de la profundidad, se divide la masa de agua en columnas de agua, cada capa más baja debe ser más densa y de menor altura debido a la compresión. Para cada capa se utiliza un perfil de salinidad y temperatura que depende de la profundidad. Estos perfiles están definidos y se pueden encontrar en páginas web públicas como el Centro Nacional de Datos Oceanográficos. [33]

Tabla 1: Efecto en el cálculo de la profundidad

Parameter	Notation	Value	Units	Description
Water density	ρ	1033.755	kg/m^3	For the point with given ***P*** and ***t***
Gravity acceleration	g	9.8089	m/s^2	For the specified φ
Depth	h_{swg}	986.112	m	For ρ=1023.6[4] kg/m^3, g=9.80665[5] m/s^2
Depth	h_{sw}	985.885	m	For ρ=1023.6[4] kg/m^3, $g=g(\varphi)$
Depth	h_{fwg}	1011.386	m	For ρ=998.02[6] kg/m^3, g=9.80665[5] m/s^2
Depth	h_{fw}	1011.154	m	For ρ=998.02[6] kg/m^3, $g=g(\varphi)$
Depth	$h_{\rho P_0}$	980.384	m	$\rho=\rho(t,P_0,s)$, $g=g(\varphi)$
Depth	$h_{\rho P}$	976.199	m	$\rho=\rho(t,P,s)$, $g=g(\varphi)$
Depth	$h_{\rho P_m}$	978.277	m	$\rho=\rho_m=\rho(t,(P+P_0)/2,s)$, $g=g(\varphi)$

Fuente: Unavlab [33]

2.4.1.5 Interferencias

Una de las características de las ondas acústicas es que son capaces de rebotar sin sufrir una atenuación notable.

Ya sea un objeto, una pared o incluso la superficie o fondo oceánicos, pueden aparecer otros efectos que dificulten el uso de señales acústicas o que provoquen que la señal no se transmita dentro del agua. Uno de estos efectos es el debido a cambios bruscos de las características del agua.

Estas diferencias de las características del agua son conocidas por los submarinistas y reciben el nombre de termoclinas, haloclinas y picnoclinas según sean debidas a la temperatura, salinidad o densidad del agua respectivamente. Estas diferencias pueden ser muy grandes, en concreto en desembocaduras de ríos.

Además, el agua puede presentar burbujas y partículas en suspensión. Estas partículas además de bloquear la luz y el campo de visión atenúan las comunicaciones acústicas[34].

2.4.2 Monitorización

La monitorización de las propiedades de los recursos acuáticos es de gran interés para la ciencia. Como se ha indicado anteriormente, el agua juega un papel muy importante en el medio ambiente, agricultura y consumo humano. Luego es de gran interés garantizar la calidad de esta. Al igual que las estaciones meteorológicas se utilizan para medir la calidad del aire, existen estaciones que monitorizan parámetros de calidad del agua, y estos sensores se pueden utilizar a bordo de un submarino.

Empresas como EMASESA[35] monitorizan para regular que los parámetros del agua se encuentren dentro de valores saludables y minimizar el impacto de los vertidos. Entre los parámetros de monitorización más importantes encontramos:

- Temperatura: La temperatura afecta directamente a la medida del resto de parámetros del agua.
- Conductividad: Es una medida de la concentración de iones en solución. Se puede relacionar con la cantidad de sólidos presentes en disolución. Valores normales se encuentran alrededor de los 2500 µS/cm
- Oxígeno disuelto: Es fundamental para la vida de la fauna y flora marina y otros organismos. Aguas estancadas tienen un nivel bajo de oxígeno disuelto, y este suele ser menor cuanto más lejos nos encontramos de la superficie. Un nivel inferior a 3ppm es dañino para la vida.
- PH: Es la medida de la concentración de iones de hidrógeno H^+. Determina la solubilidad y biodisponibilidad, siendo una medida directa de la toxicidad del agua. Valores saludables para consumo humano entre 6.5-9.5.
- Irradiancia: Indica el valor de la intensidad energética (proveniente del sol) que se recibe en el punto de muestra. Es un valor útil ya que esta energía puede producir reacciones químicas entre compuestos.
- Turbidez: Es una propiedad óptica de la transparencia del agua. Indica la capacidad de una suspensión acuosa de dispersar la luz y hacer que no se transmita a través. Según la OMS debe ser menor a 5NTU
- Cloro: Es altamente usado para desinfección. Su concentración debe ser menor a 1 mg/l.
- Nitritos: Es un nivel de la contaminación de las aguas naturales por compuestos nitrogenados. Debe ser menor a 0.5 mg/l.

2.4.3 Comunicación

Dadas las características del medio acuático descritas, establecer una comunicación no resulta sencillo. En AUVs este problema no es tan drástico, al ser completamente autónomos, pero en ROVs es necesario un "Human in the Loop". Los métodos de comunicación más utilizados son:

- Uso de un "Umbilical": Un cable a través del cual sucede toda la comunicación Piloto-Vehículo. En la mayoría de los casos este cable es flexible y se usa solo para comunicación. También existen aplicaciones en las que el cable es rígido y proporciona energía al vehículo además de transmisión de datos.

Esta alternativa limita el rango de operación del vehículo. Además, el umbilical puede quedarse atascado en objetos y es responsable de una gran fuerza de arrastre que altera la hidrodinámica del vehículo.

- Comunicación inalámbrica: Esta sucede principalmente mediante el empleo de routers que emplean las ondas acústicas.

  Son dispositivos caros, que funcionan normalmente en la frecuencia de 10kHz y que proveen de tasas de transferencia de 70bps hasta 314bps a distancias superiores a 3000m.[36]

2.5 Navegación

2.5.1 Maniobrabilidad

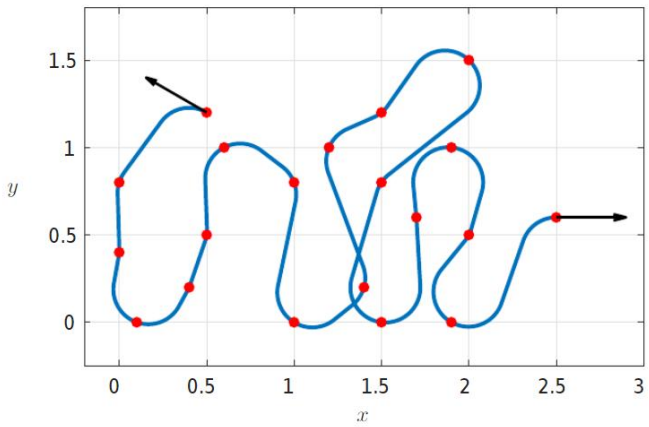
En la teoría náutica clásica el desempeño de un vehículo náutico se mide por su capacidad de realizar maniobras, aceleraciones y velocidades angulares y lineales. Existen una serie de movimientos básicos a los que se somete la embarcación y cuyos resultados otorgan un certificado de capacidades. Estos aspectos están descritos en archivos como el "Guide for Vessel Maneuverability"[37] o libros de construcción naval, y su estudio es uno de los motivos por el que los vehículos acuáticos tienen la forma que tienen.

Al igual que en la aviación, existen maniobras acuáticas que resultan de interés para los submarinos. Por citar algunas, las trayectorias más utilizadas son:

- Trayectorias en línea recta.
- Trayectorias curvas.
- Trayectorias helicoidales, para realizar maniobras de inmersión o emersión o cambio de altura.

Cuando se busca crear una trayectoria esta se suele generar mediante interpolaciones, lineales o cúbicas. Cuando se dan varios puntos a seguir en la trayectoria se suele utilizar el interpolador de Dubins. Según el cual, conocida una serie de puntos objetivo y dada una curvatura, busca la curva plana más corta que pasa por 2 puntos respecto a sus tangentes. Es un algoritmo óptimo que realiza una combinación de trayectorias lineales y curvas entre 2 puntos. Logrando así la trayectoria más corta posible.

Figura 5: Interpolador de Dubins



Fuente:C. Yalçin Kaya

2.5.2 Localización

2.5.2.1 Geolocalización

Como se ha indicado anteriormente, las ondas electromagnéticas sólo pueden atravesar unos centímetros de agua antes de atenuarse en gran medida. Esto provoca que el vehículo solo disponga de señal GPS cuando se encuentra en superficie.

El posicionamiento subacuático se realiza mediante señales acústicas. Teniendo en cuenta las características de la propagación del sonido en el agua, se puede calcular la posición desde un sistema de referencia local. Si este sistema se encuentra en la superficie, se puede usar junto a una señal GPS para obtener la posición del vehículo. En el agua se conocen 3 tipos distintos de posicionamiento:

- LBL: Utiliza el mismo principio de posicionamiento que las balizas, donde 3 o más dispositivos de posición conocida y fija se utilizan para triangular la posición. Se pueden lograr precisiones inferiores a 1 metro.
- SBL: Utiliza el mismo posicionamiento, pero no requiere que las balizas se encuentren en una posición fija, normalmente se encuentran ancladas a un barco. Se logran precisiones inferiores a un metro
- USBL: En lugar de balizas posee 4 receptores situados en una antena, conectados a un ordenador que se encarga de realizar los cálculos de posición. En este caso la posición se conoce en tierra y no en el dispositivo. Requiere que el vehículo u objeto a localizar posea un transpondedor.

2.5.2.2 Navegación por estima

Se denomina navegación por estima al proceso de calcular la posición actual del vehículo aplicando estimaciones de la velocidad, tiempo y dirección desde la última posición conocida.

Este proceso era utilizado en la antigüedad por los marineros para llegar desde un puerto a otro. En la actualidad, poseemos sensores más potentes, pero un error de 0.5º (normal en un magnetómetro) puede traducirse en un error de 8 metros tras 1km.

El uso de sistemas de navegación inercial (INS) para la creación de sistemas de referencia de actitud y rumbo (AHRS), junto a filtros de Kalman ha logrado reducir en gran medida este error. Sin embargo aparece otro problema para la navegación submarina, la estimación de la velocidad puede tener un error muy grande, para tiempos mayores a 20 segundos los errores dejan de ser aceptables.

La solución adoptada es el uso de DVL (perfilador acústico de corrientes por efecto Doppler). Este dispositivo es capaz de estimar la velocidad relativa al fondo del mar mediante el uso de ondas acústicas.

Fusionando todos estos elementos se han conseguido errores de posición menores al 0.05% en misiones de larga distancia..[38]

2.6 Visiones futuras en el campo

El campo de la robótica acuática está en constante desarrollo, superando barreras tecnológicas y proponiéndose nuevos retos. Existen una gran cantidad de grupos de investigación desarrollando nuevas tecnologías con el objetivo de aumentar el número de aplicaciones y abaratar costes. La gran mayoría de avances tecnológicos se centran en el ámbito militar. Entre otros se encuentran:

- El empleo de cámaras para realizar mapas 3d de visión aumentada y combinación con visión por ordenador para localizar vehículos, objetos o seres vivos[39].
- Uso de flotas de vehículos utilizando algoritmos tolerantes a fallo para realizar tareas colectivas[40] y algoritmos de optimización de tareas[41]

- Uso de sonar o batimetrías para localización de vehículos o mapeado de zonas acuáticas[42]
- Vehículos capaces de maniobrar por aire, agua y superficie.
- Vehículos biomiméticos, capaces de camuflarse con la fauna marina.
- Vehículos AUV con posibilidad de teleoperarse (combinación de ROV y AUV)

3 DESCRIPCIÓN DEL SISTEMA

El vehículo utilizado en este TFM es el Sibiu Nano + de Nido Robotics[20].

Ilustración 10: Sibiu Nano +

Fuente: Nido Robotics [20]

Este es un vehículo submarino ligero controlado de forma remota por un operario desde tierra mediante el uso de un mando o joystick, necesitando de un ordenador para poder operarlo.

A partir de la página web del fabricante y mediciones en el submarino real se ha recogido la siguiente información.

Tabla 2: Tabla características Sibiu Nano +

Dimensiones	[35 25.8 23.7]cm
Masa	5.15 kg
Lastre	0.8 kg
Material	Resina de flotación R-3312
Duración Batería	2 Horas
Diámetro umbilical	4mm
Densidad del umbilical	1-1,05 g/cm3
Profundidad Máxima	100 m
Corriente máxima de trabajo	2 nudos
Velocidad máxima	2 nudos (3.7 km/h)
Cámara	Streaming 1080px

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

3.1 Hardware

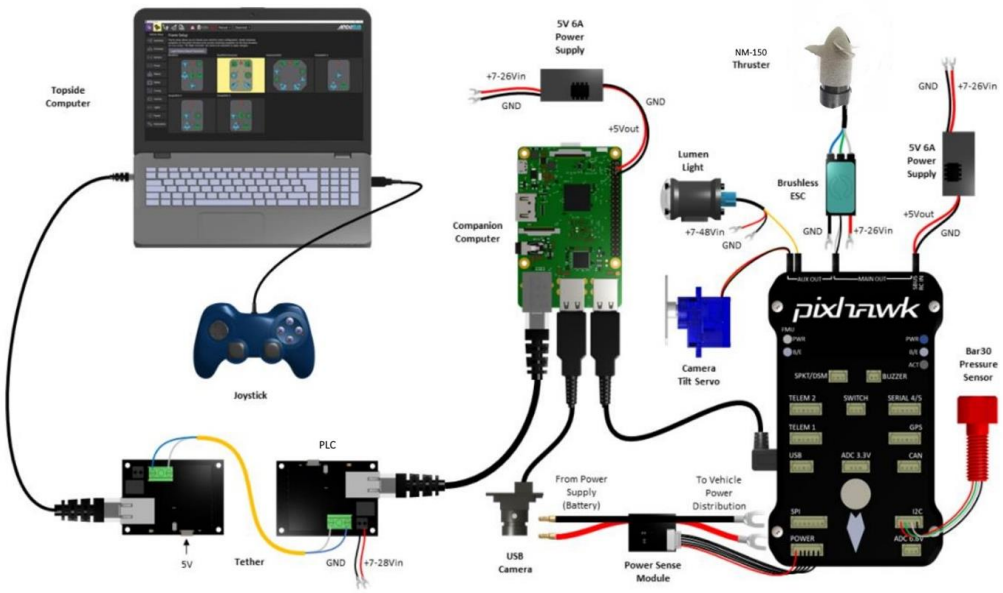
El Sibiu Nano + está compuesto por 8 motores (4 en posición vertical y 4 en posición horizontal) dispuestos en configuración vectorial. Como sensores tiene una cámara, un sensor de presión, un sensor de corriente y voltaje, humedad y temperatura. En las siguientes tablas y figuras se puede ver su disposición.

Tabla 3: Tabla Hardware Sibiu Nano +

Batería	4s 6750mAh (14.8V)
Unidad de Medición Inercial (IMU)	MPU6050
Motores	NM-150
Propulsores	NM-150
ESC	20 A
Luces	4 x 1500 lumens
Velocidad máxima	2 nudos (3.7 km/h)
Autopiloto	Pixhawk4
Companion Computer	Raspberry Pi 3B+
PLC	WISPLC
Cámara	Low-Light HD USB Camera Sony IMX322/323

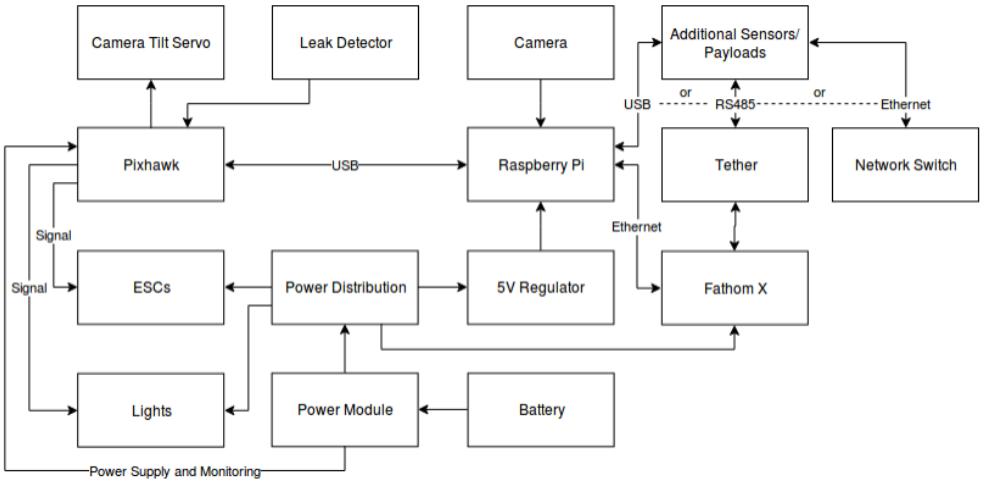
Fuente: Elaboración propia

Figura 6: Esquema Hardware Sibiu Nano +



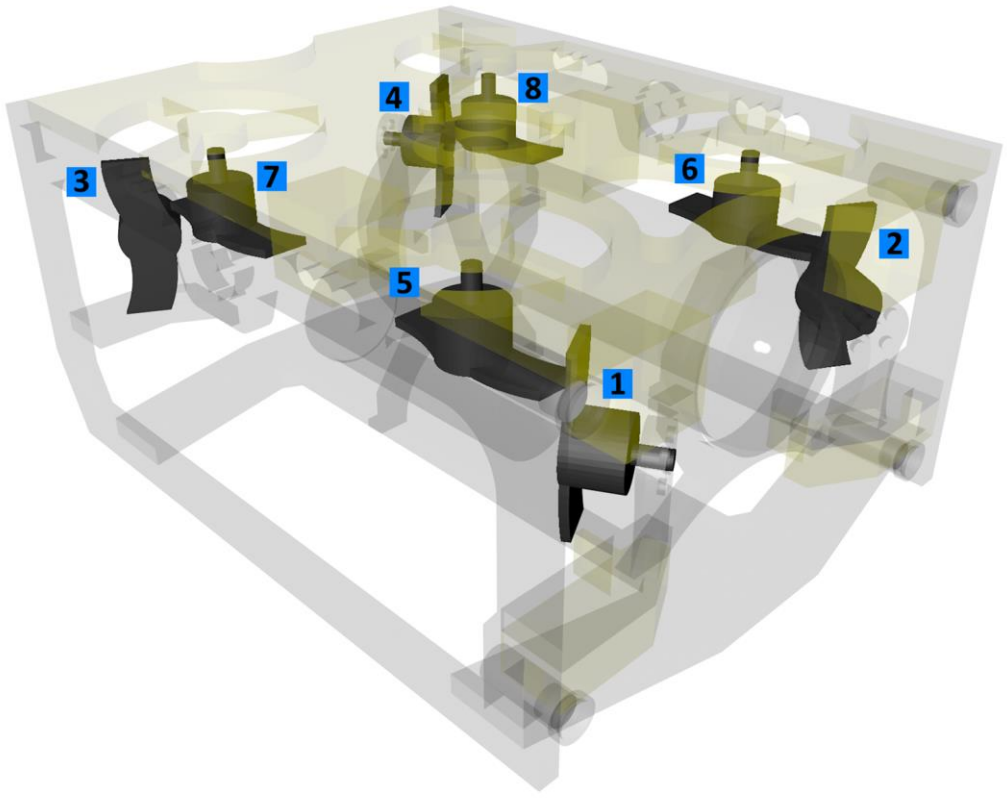
Fuente: Ardusub

Figura 7: Esquema Conexiones Sibiu Nano +



Fuente: Enrique González Sancho [43]

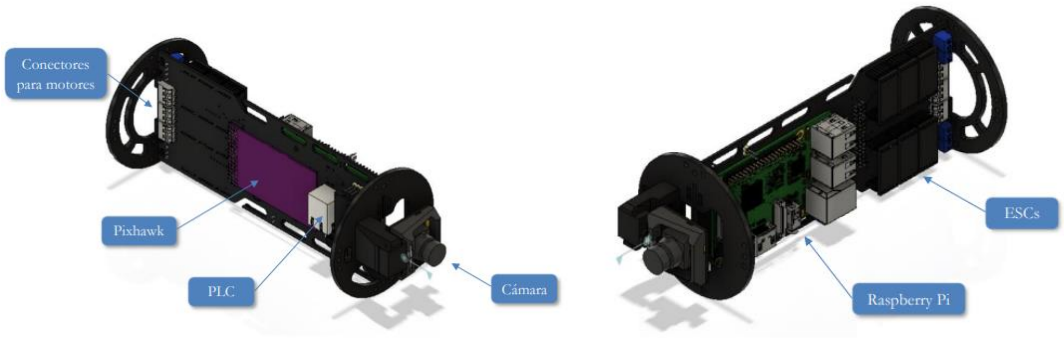
Figura 8: Disposición motores Sibiu Nano +



Fuente: Elaboración propia

La disposición de los motores permite que el submarino rote y se deplace en todas las direcciones. Debido a que los motores son controlados por un ESC, la señal de entrada a los motores es un PWM con valores entre 1100 y 1900, con posición neutra en torno a 1500 y una zona muerta asimétrica de entorno a 15 PWM (la zona muerta es menor fuera del agua).

Figura 9: Distribución tubo estanco Sibiu Nano +



Fuente: Enrique González Sancho [43]

El sistema está alimentado por una batería lipo extraíble de 4 celdas, con voltaje nominal 14.8V 6750mAh de capacidad y una transmisión de 70C y es capaz de transmitir datos a una estación base de superficie a través de un cable de 200 metros utilizando conexión Ethernet. Para esto utiliza 2 WISPLC líneas de poder.

Entre los sensores disponibles se encuentran:

- IMU (Giroscopio, Acelerómetro y Magnetómetro) MPU 6050
- Sensor de presión
- Cámara 1080p 110º
- Barómetro
- Humedad
- Temperatura
- Voltaje y Corriente
- Sonar lineal (Opcional)
- Sonar Lineal (Opcional)
- Sonar 360º (Opcional)
- Garra (Opcional)
- DVL (Opcional)
- USBL (Opcional)

3.1.1 USBL

El USBL+GPS presente en el submarino pertenece a la empresa Water Linked, en concreto se trata del modelo "Underwater GPS G2".[44]

Este modelo está formado por un ordenador de tierra (Topside) conectado a una antena (Receptor) con 4 receptores y un transceptor (Localizador) situado en el submarino.

Ilustración 11: Waterlinked Topside / Antenna / Locator

Fuente: Waterlinked [44]

Para un funcionamiento adecuado se recomienda que tanto la antena como el objeto a localizar se encuentren sumergidos a más de 1 metro de profundidad para reducir los rebotes con la superficie. Además, es recomendable que exista una visual de la antena con el transceptor.

Otras indicaciones para una correcta utilización del sistema es la elección de un canal para la transmisión de datos (cambiar de canal si se detectan interferencias) e indicar al ordenador de superficie (Topside) la posición relativa entre el Topside Computer y la antena. Además, el sistema debe estar alineado con la antena.

Para realizar estas configuraciones se debe conectar al sistema vía internet (web o cable) y acceder a él en la página "http://192.168.7.1/". El dispositivo también posee una API para utilizar los datos de la antena.

En el anexo 7.3.2 Aparecen imágenes de la interfaz web y funcionamiento del sistema.

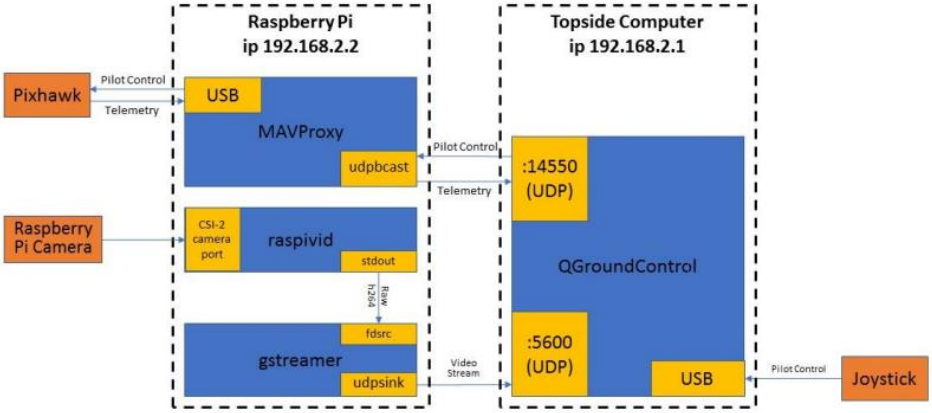
3.2 Software

El submarino Sibiu Nano + utiliza el framework de Ardupilot.

Se puede dividir la estructura software completa del submarino en tres subsistemas:

- Estación Base: Formada por un ordenador y pantalla. Permite al usuario leer los datos de los sensores y ver en tiempo real la cámara del submarino. También tiene conectado un mando para su control manual.
- Raspberry Pi: Actúa como un router, conectada via UART a 115200 baudios con la Pixhawk y abriendo una serie de puertos TCP/UDP para comunicación con la Estación Base.
- Pixhawk: Ejecuta el control a bajo nivel generando las señales PWM para los ESC y realiza la lectura de los sensores. En ella se ejecuta el código abierto en C++ diseñado por Ardupilot adaptado para vehículos submarinos.

Figura 10: Módulos Software Ardusub



Fuente:Nido Robotics [20]

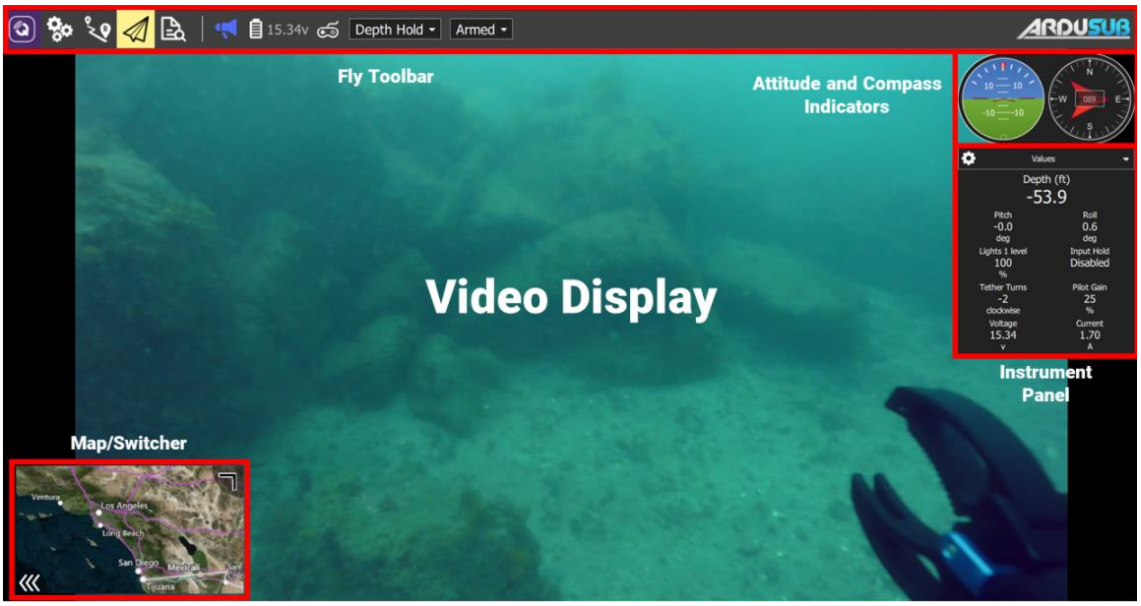
3.2.1 Estación Base

La Estación Base promueve el uso de la herramienta QGroundControl. Esta herramienta utiliza el puerto 14550 del ordenador de tierra para lectura de datos y el puerto 14570 de la Raspberry Pi para lectura de órdenes.

Esta herramienta provee de una interfaz simplificada y completa para un usuario normal del submarino. Permitiendo tener una lectura rápida y amigable de la imagen captada mediante la cámara del submarino, así como de la IMU y otros sensores.

También dispone de un menú de configuración para cambiar parámetros básicos del submarino, así como enviar mensajes personalizados.

Ilustración 12: QGroundControl



Fuente: Ardusub [45]

Esta herramienta también ofrece las opciones de configuración y actúa como driver, codificando las entradas de un controlador conectado al ordenador a órdenes para el submarino.

Figura 13: Diagrama de controles del Mando

Fuente: Ardusub [46]

Ardupilot por su parte promueve el uso de la herramienta Mission Planner

Ilustración 12: Mission Planner

Fuente: Ardupilot [47]

La herramienta Mission Planner no ofrece la traducción controlador-submarino, no permitiendo controlar remotamente el submarino. Sin embargo, ofrece una lectura completa de todos los datos y parámetros del submarino permitiendo ajustar parámetros de control y de comportamiento en tiempo real así como visualizar datos guardados en el log interno del submarino. Además te permite ver todo el tráfico de mensajes y 'heartbeats', ofreciendo así una mayor cantidad de información al desarrollador de lo que sucede internamente.

Pulsando la combinación de teclas Ctrl+F te permite entrar a un modo desarrollador permitiendo simular sensores y mensajes para test.

Ilustración 13: Opciones Avanzadas Mission Planner

Fuente: Elaboración propia

3.2.2 Raspberry Pi

La Raspberry Pi corre una imagen provista por Ardusub [48]. Esta contiene una imagen de Raspbian Jessie con Ardupilot instalado. Raspbian Jessie llegó al fin de su ciclo de vida en septiembre de 2017 y no recibe ningún tipo de soporte desde entonces, haciendo la instalación y actualización de algunos programas difícil de lograr.

Sus funciones principales son de inicialización de parámetros de la Pixhawk y el papel de router entre la Estación Base y la Pixhawk, además se encarga de controlar el servo de la cámara y retransmitir la imagen. También hace Host de una página web de configuración en 192.168.2.2.

Esta interfaz está formada por 4 paneles que ofrecen una configuración alternativa a ssh para la configuración y análisis del sistema.

Se puede observar una representación de estos en el anexo 7.3.1, están divididos principalmente en:

- Gestión de actualizaciones y conexión a red.
- Administración de ficheros
- Gestión de redirecciones
- Ajustes de cámara
- Terminal web

En el caso de acceder mediante terminal se requiere de un usuario y contraseña, para la imagen proporcionada:

- Usuario: “pi”
- Contraseña: “companion”

Ardusub corre como un servicio en la Raspberry Pi. En Linux, los servicios son scripts de código que se ejecutan automáticamente en segundo plano cuando se dan una serie de condiciones especificadas en la definición del servicio.

Esto es útil ya que permite el reinicio de la conexión en caso de bloqueo o pérdida de señal mediante el uso en una terminal Linux del comando:

“Sudo systemctl restart ardusub",

fácilmente ejecutable desde un script de Python mediante la librería ‘subprocess’ que se puede encontrar en el Anexo 7.4.1.

3.2.3 Pixhawk

La Pixhawk contiene un ARM Cortex M4 de 32 bits a 168MHz [49]y está a cargo de la lectura de la mayoría de los sensores y generar las señales PWM para los ESC. En ella se ejecuta el código abierto en C++ diseñado por Ardupilot adaptado para vehículos submarinos[46]

La comunicación con la Pixhawk utiliza un protocolo llamado Mavlink explicado con más detalle en el apartado 3.4.

Ilustración 14: Pixhawk

Fuente: Enrique González Sancho[43]

El código de Ardusub se actualiza cada año y hay una gran comunidad detrás de él, con una gran cantidad de guías[50] para nuevos desarrolladores al tratarse de un código de más de 700000 líneas en el que interactúan paralelamente una gran cantidad de variables y funciones.

3.3 Ardusub

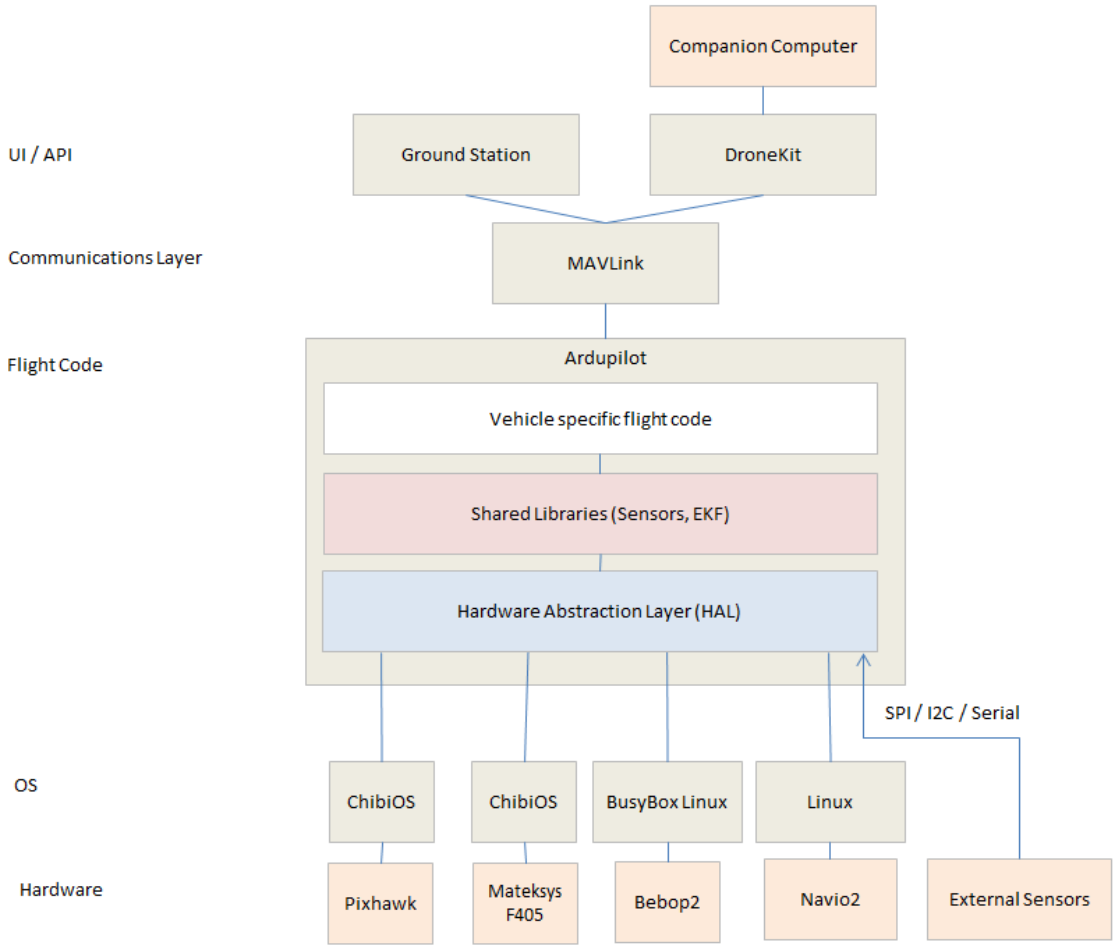
Como se ha indicado en el estado del arte, un buen software robótico debe permitir el uso de componentes hardware heterogéneos que se comuniquen entre ellos asíncronamente y facilitar el desarrollo de algoritmos sin tener que ser un experto.

El comportamiento de Ardupilot es complejo y está formado por más de 700.000 líneas de código. Para todo aquel que quiera iniciarse dispone en su página web de una guía para iniciarse en el comportamiento general [50].

Ardupilot es un sistema estructurado en bloques, ejecutados a distintas frecuencias y diferentes threads. La placa Pixhawk permite el uso de hilos Posix, con prioridades en tiempo real

En la siguiente imagen se puede observar la estructura y papel que juega Ardupilot en un sistema completo.

Figura 14: Estructura Completa Ardupilot



Fuente: Ardupilot [50]

Ardusub es un módulo presente en ardupilot. Este hace uso de todas las librerías y componentes presentes en Ardupilot adaptados a la estructura presente en un submarino.

Estos componentes podemos estructurarlos en las siguientes categorías:

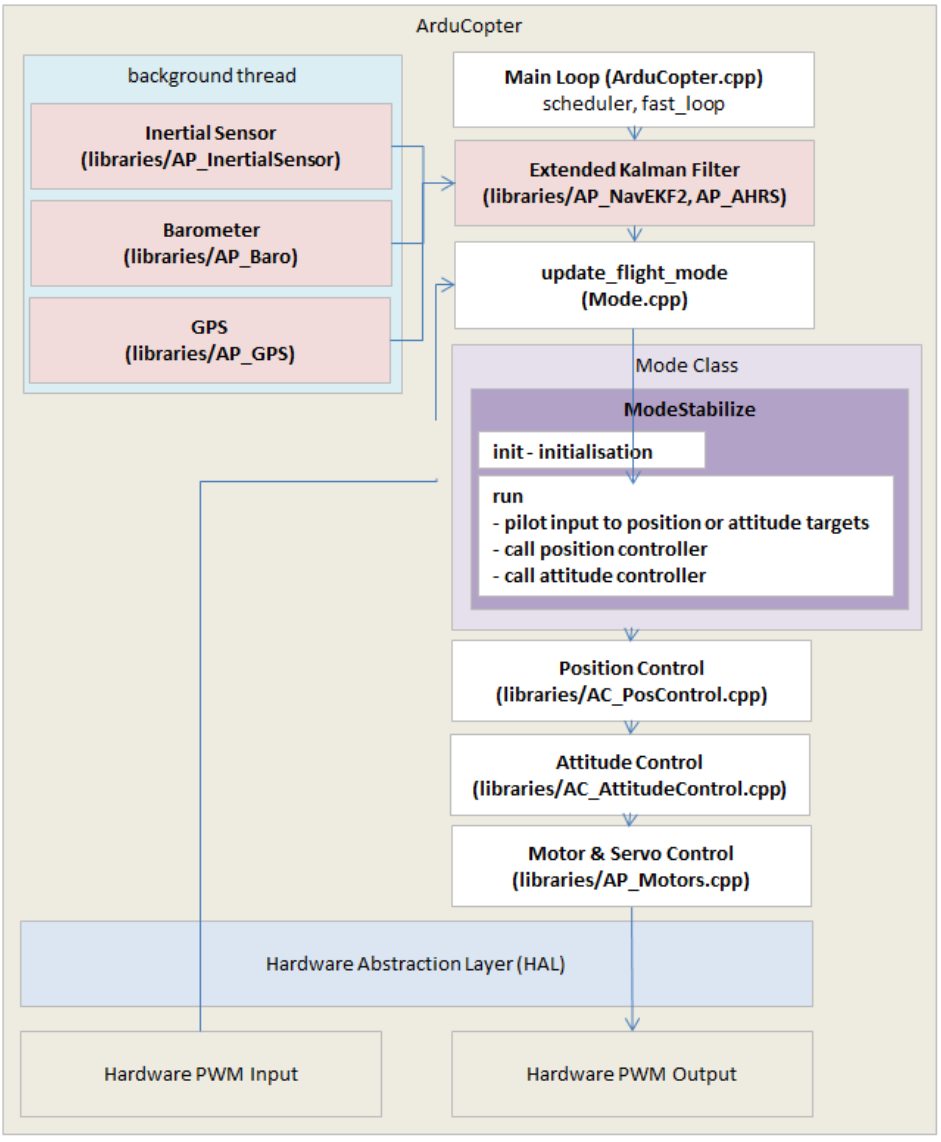
- Librerías de sensores y actuadores.
- Hardware Abstraction Layer.
- Tipo de vehículo: Incluye los modos específicos de operación de cada vehículo.

- Mode Class
- Failsafe
- Controladores
- Log
- Communication: GCS and mavlink
- Inertia
- Joystick
- Motors
- EKF

3.3.1 Estructura

En la siguiente imagen podemos ver un ejemplo de una construcción típica de Ardupilot, en este caso para vehículos aéreos tipo dron.

Figura 15: Estructura de Arducopter



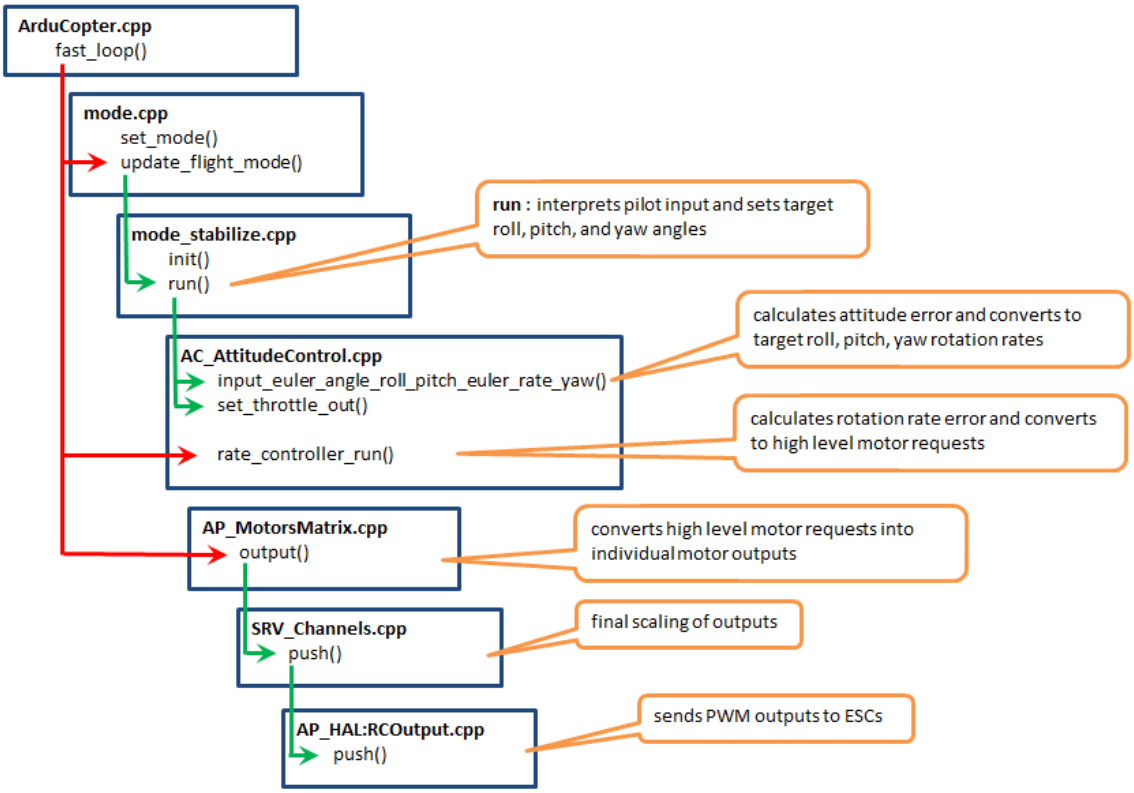
Fuente: Ardupilot [50]

Estos módulos se ejecutan en distintos threads. De tal forma que los procesos se llaman mediante event handlers con un periodo dado y una prioridad determinadas. Para un correcto funcionamiento todos los procesos tienen un Worst Case Timing indicado.

Así Ardusub dispone de varios Bucles de código (denominados loops) ejecutados de forma periódica.

Un ejemplo de aplicación para el modo de operación Stabilize se encuentra en la siguiente figura.

Figura 16: Diagrama de bloques modo Stabilize



Fuente: Ardupilot [50]

Así establecemos las siguientes distinciones

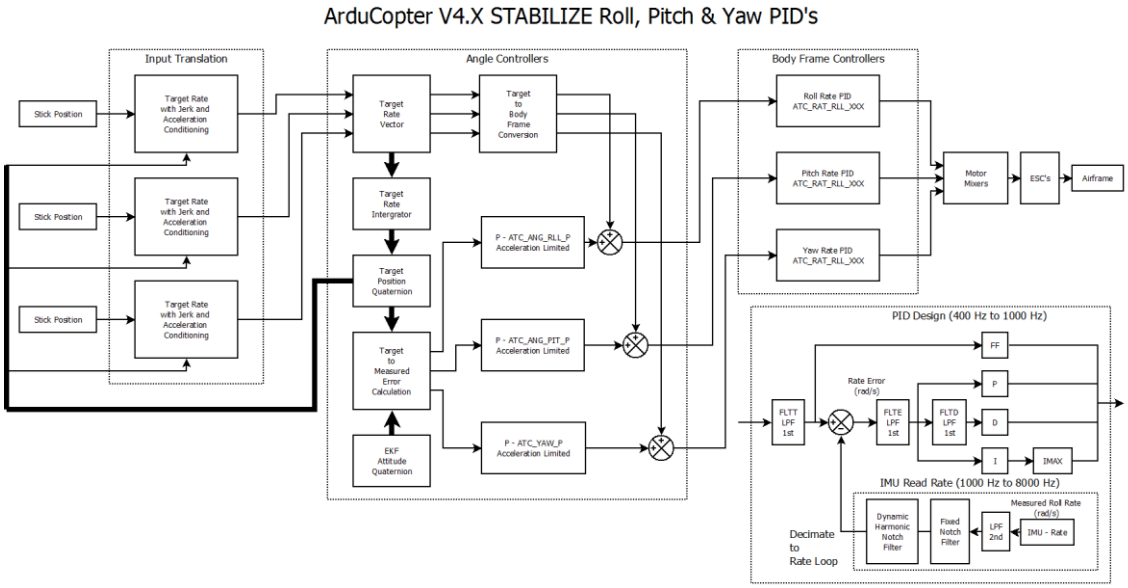
- **400Hz**: Código principal, realiza los cálculos del control de bajo nivel y la lectura de la IMU, EKF, Inercia y la función principal del modo en el que opera el submarino.
- **50Hz**: Lee RC y comprueba las flags de failsafe.
- **10Hz**: Lecturas de Batería, Brújula y Log de datos
- **3Hz**: Comprobaciones más lentas (temperatura, leak) y logs.
- **1Hz**: Flag mensaje de configuración.

3.3.2 Control

En el trabajo anterior "Modelado, simulación y control de un vehículo submarino ligero"[1] se trataron los aspectos a tener en cuenta a la hora de realizar un control. Ardusub opta por la misma estrategia. El control está basado en PID aunque con diagrama con mayor funcionalidad.

Por ejemplo, para el modo de estabilización (el piloto puede mover el ROV, este mantiene el roll y el pitch en valores normales), se puede ver como se calcula un vector deseado a partir del estado actual del dron y las órdenes del piloto, se traduce a aceleraciones para cada grado de libertad.

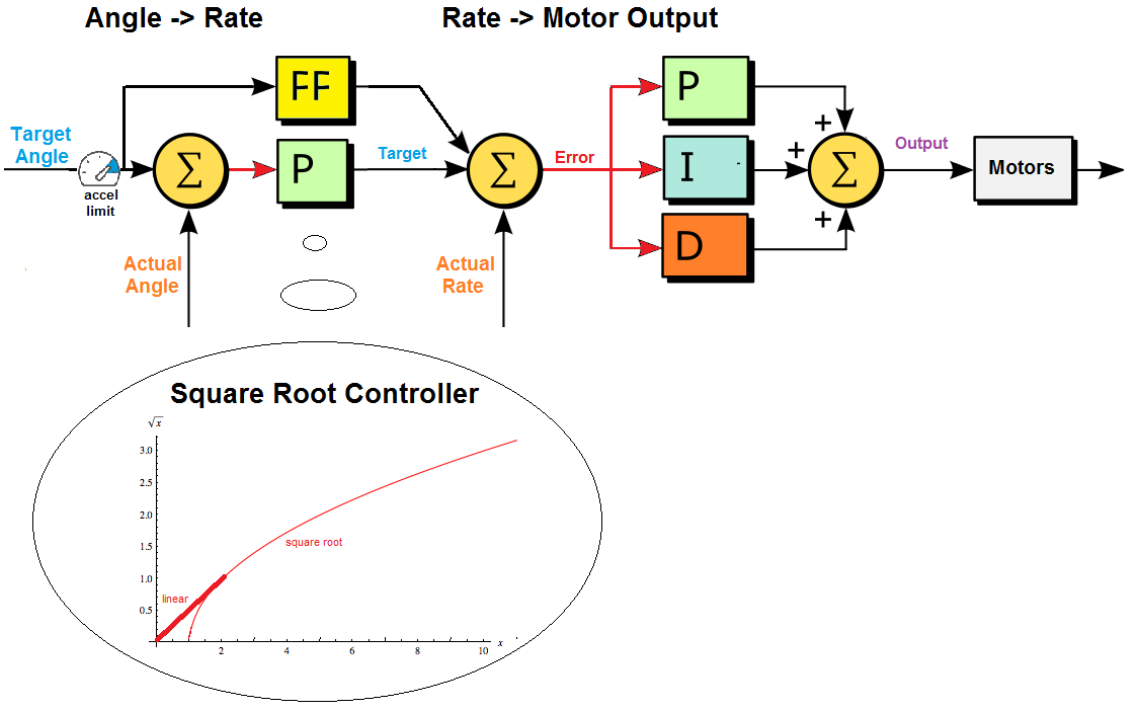
Figura 17: Diagrama de control modo Stabilize



Fuente: Ardupilot [50]

Desde el punto de vista de la actitud estas aceleraciones se saturan según las capacidades del ROV. Un PID convierte este ratio de rotación en un comando de motor de alto nivel (escalado [-1,1]), que se traduce en salidas individuales para cada motor de acuerdo a la arquitectura del ROV.

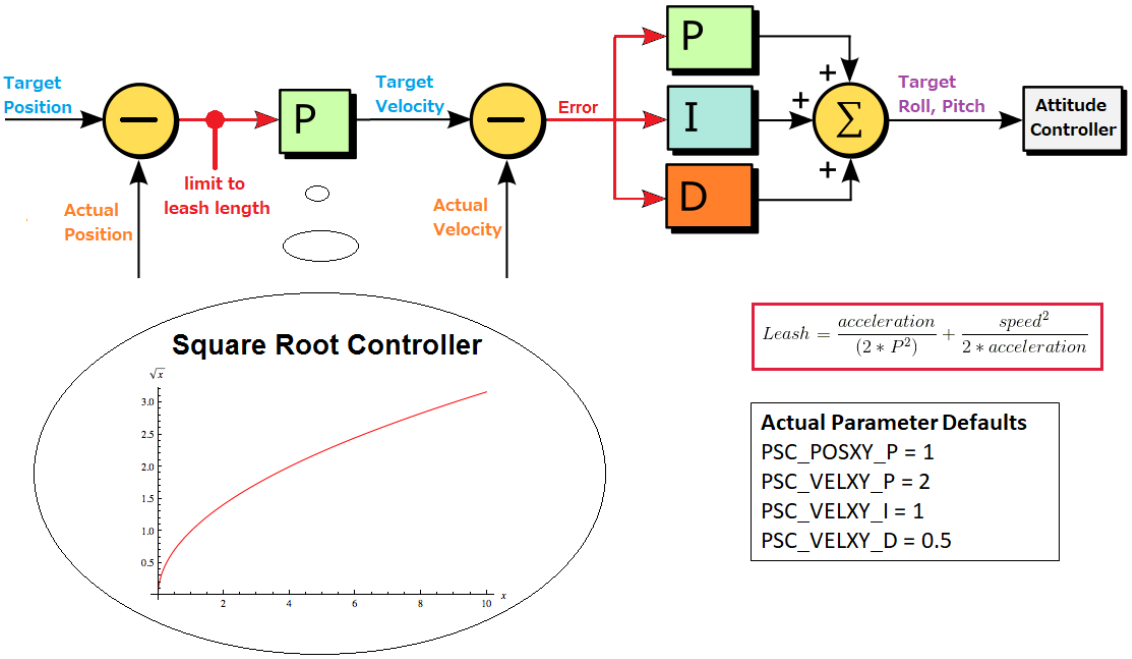
Figura 18: Funciones de transferencia Control de Actitud



Fuente: Ardupilot [50]

Desde el punto de vista de la posición, hay una división entre el eje Z, y los ejes XY para los que aparece un interpolador (combina comportamientos de aceleración, lineal, curvo y frenada). Se crea así un vector velocidad deseado para el ROV.

Figura 19: Funciones de transferencia Control de Posición



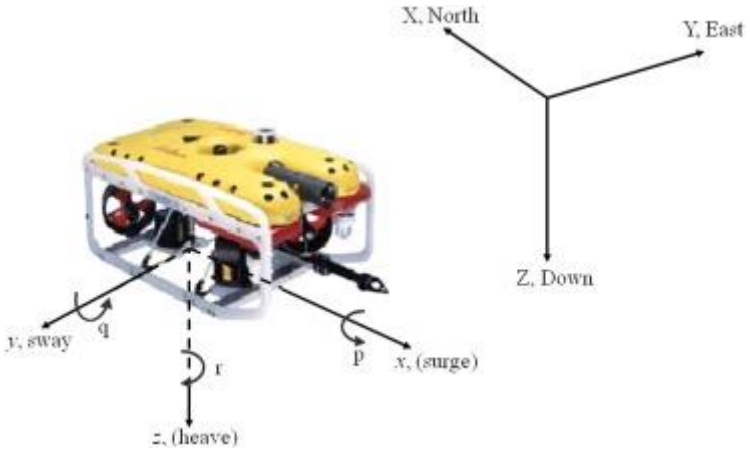
Fuente: Ardupilot [50]

Se puede observar que para un valor del grado de libertad deseado, un controlador P, convierte la posición en velocidad antes de pasar por el PID que la convierte en aceleración. Este controlador P busca limitar el valor máximo de la entrada además de calcular el valor objetivo, viene acompañado de una etapa de FeedForward para aumentar el tiempo de respuesta. El PID busca crear la reacción en los motores que sigue este valor objetivo.

Uno de los principales papeles del controlador P es mitigar los efectos de acoplo entre el movimiento en el eje Z y en los ejes XY

En el siguiente diagrama se puede observar el efecto de la señal enviada a los motores. Esta es un valor normalizado [-1,1] indicando el ratio de giro en el eje deseado.

Figura 20: Ejes de Movimiento ROV



Fuente: Serdar Soylu et al [51]

3.3.3 Modos de operación

El código de Ardusub actualmente ha desarrollado 3 modos de operación para el vehículo.

- Manual: Permite introducir una señal RC para controlar el submarino en cada uno de sus 6 grados de libertad. Este modo llama al componente que realiza el mapeado de los motores de acorde a las señales RC.
- Deep Hold: Añade una señal vertical al control de profundidad de tal forma que el nivel de profundidad del submarino se mantenga. Se actúa cuando la derivada de la velocidad vertical es distinta de 0, tratando de mantener la altura actual. No funciona cerca de la superficie.
- Stabilize: Incluye un controlador para regular la dirección del robot. Se centra en el Pitch y el Roll para mantenerlos en valores seguros.

Ardupilot nos permite programar modos personalizados para realizar otros tipos de control. Vía MAVLink podemos cambiar de modo libremente en cualquier momento, sin embargo, para hacerlo en una interfaz fácil para el usuario, la interfaz provista por Ardupilot tiene que reprogramarse.

3.3.4 EKF

Conocer la posición del vehículo en cada uno de los grados de libertad no es sencillo y en una gran cantidad de casos se comenten errores no aceptables. Existe una gran cantidad de formas de mitigar estos errores como podría ser el uso de algoritmos DCM utilizados en frameworks como INAV. El código de Ardupilot hace uso de un Filtro de Kalman Extendido

Mediante la fusión de sensores, el objetivo es eliminar medidas de sensores que tengan una gran cantidad de error (siendo así menos susceptible a errores que afectan a un solo sensor). Además de ser capaz de estimar los errores de offset en la brújula, así como estimar el campo magnético de la Tierra. También permite utilizar otros sensores para medidas como sensores de distancia ópticos o láser.

El Filtro de Kalman Extendido desarrollado por Ardupilot hace uso de 24 estados[52].

- Orientación (cuaterniones).
- Velocidad (Norte, Este, Abajo).
- Posición.
- Giroscopio bias offset (X, Y, Z).
- Factores de escala del Giroscopio (X,Y,Z).
- Bias de la aceleración del eje Z.
- Campo magnético de la Tierra (Norte, Este, Abajo).
- Campo magnético del cuerpo (X,Y,Z).
- Velocidad del viento o corrientes (Norte, Este).

La ejecución del mismo sigue los siguientes pasos:

1. Se integran los ratios angulares de la IMU para obtener la posición angular.
2. Las aceleraciones de la IMU se convierten usando la posición angular a Norte Este Abajo y se corrige la gravedad.
3. Las aceleraciones se integran para calcular la velocidad.
4. La velocidad se integra para calcular la posición.
5. Se calcula la matriz de covarianza de estados, estima los errores de los estados.

Los estados 1 a 5 se repiten cada vez que se recibe un nuevo dato de la IMU.

6. Si llega un dato GPS, el filtro calcula la diferencia entre la posición y la estimación (Innovacion).
7. La innovación, la matriz de covarianza de estados y el error de GPS se combinan para calcular una corrección a cada uno de los filtros de los estados (Corrección de estados).

   Esta es la parte inteligente del Filtro de Kalman, utiliza la correlación entre los diferentes errores y estados para corregir estados distintos al que se mide.

   Por ejemplo GPS puede usarse para corregir errores de posición, velocidad, ángulos y giro.
8. Una vez se obtiene una nueva medida, el grado de no certeza de los estados se reduce, actualizando la matriz de estados de covarianza y volviendo al estado 1.

Se pueden ejecutar distintas instancias del EKF realizando estos cálculos utilizando distintas combinaciones de sensores y analizando las matrices de estados de covarianza de cada una de ellas, eligiendo aquella cuya matriz de estados de covarianza sea menor.

Aparecen así distintas líneas de ejecución. Esta estrategia permite cambiar entre distintas IMUs haciendo más fácil la recuperación de situaciones de datos y depende menos de la calidad de los datos otorgando una salida suave que no requiere una gran capacidad de cálculo.

Como datos extra del EKF utilizado, este tiene en cuenta el timestamp de los datos que recibe y en lugar de estimar la orientación de los cuaterniones estima un error del vector de rotación y aplica la corrección al cuaternion de las ecuaciones inerciales de navegación.

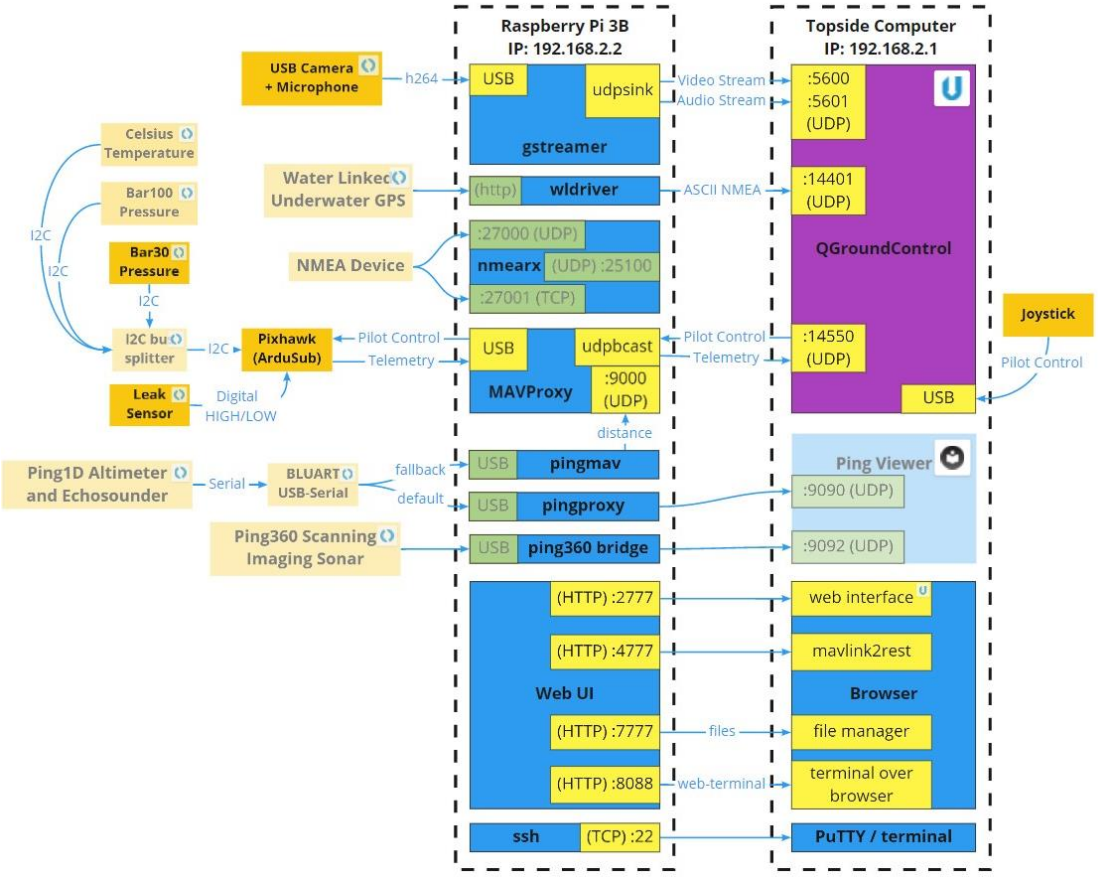
3.3.5 Otros datos de interés

El código de Ardupilot está muy optimizado y cuenta con una gran cantidad de funcionalidades secundarias. Algunas de ellas pueden ser:

- Ardupilot tiene en cuenta el tiempo restante de ejecución (worst case timing), el estado del procesador y badblocks antes de ejecutar un hilo de código para asegurar la concurrencia.
- Es capaz de codificar tonos armónicos en los motores para enviar avisos sonoros al piloto. Esto sucede en situaciones como cuando se inicia el vehículo, hay un error o llega a un objetivo.
- Dispone de flags de seguridad denominados failsafes capaces de alterar el comportamiento del vehículo.
    - Se puede utilizar failsafes de GCS para entrar en modos de seguridad o vuelta a tierra.
- El submarino guarda en memoria el número de giros que ha realizado durante la misión con el objetivo de evitar la torsión del cable.

3.4 Comunicación

Figura 21: Comunicaciones Ardusub Extendido



Fuente: Ardupilot [50]

A través del cable ethernet que conecta la Raspberry pi con la Estación Base suceden un gran número de distintas comunicaciones: Streaming de video y audio UDP (a través de Gstreamer), interfaz web (a través de http), terminales (a través de ssh) e intercambio de mensajes de lectura de datos e instrucciones (a través de MAVLink). Este último apartado es el que tiene más importancia desde el punto de vista del sistema de control utilizado para la navegación del vehículo submarino.

3.4.1 MAVLink

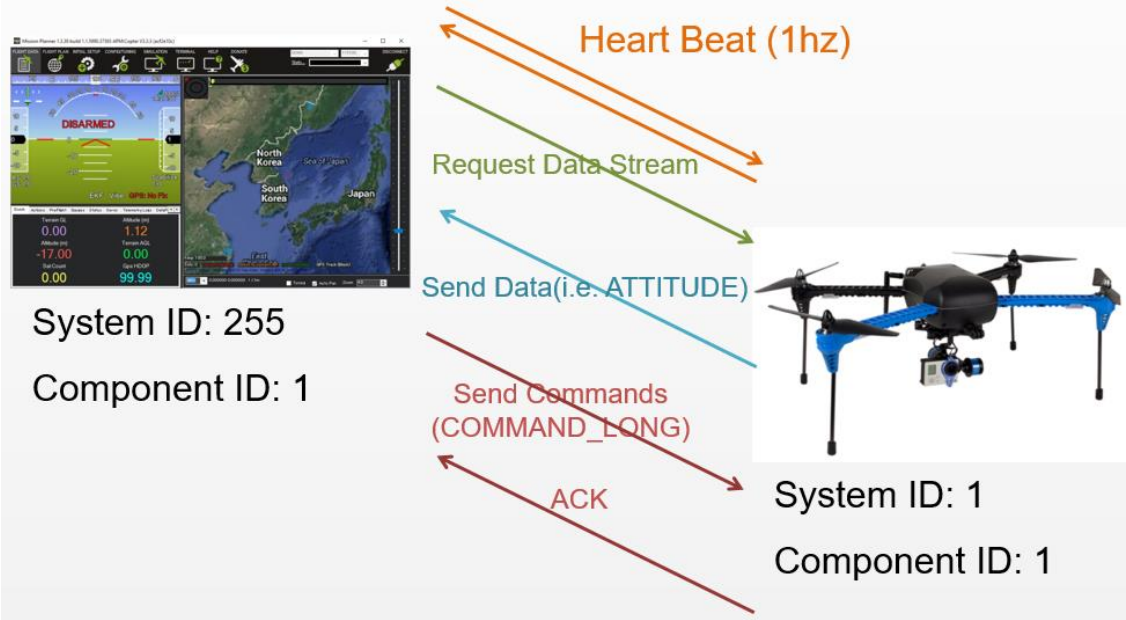
MAVLink es un protocolo de comunicación, Ardupilot en concreto corre la versión 2 de MAVLink, retrocompatible. En el vehículo, la comunicación sucede en dos fases:

- Una fase UART a 112500 baudios entre la Raspberry Pi y la Pixhawk
- Una fase UDP/TCP entre la Raspberry Pi y la Estación de Tierra.

Cada mensaje se debe publicar periódicamente para ser reconocido por el recibidor. Se dice que cada mensaje tiene un "latido" mayor a 1Hz de tal forma que es capaz de detectar si el dron pierde conexión o si un módulo se rompe o falla si este no envía un mensaje pasados 4-5 segundos (parámetro configurable).

Al inicio de la comunicación, cada sistema emite un latido, indicando que está activo. Para recibir información esta debe solicitarse al sistema mediante un mensaje, indicando qué datos se quieren recibir así como la frecuencia a la que quiere que este se envíe. MAVLink no tiene control de flujo, luego debes guardar en memoria lo que esperas recibir.

Figura 22: Inicio de conexión MAVLink

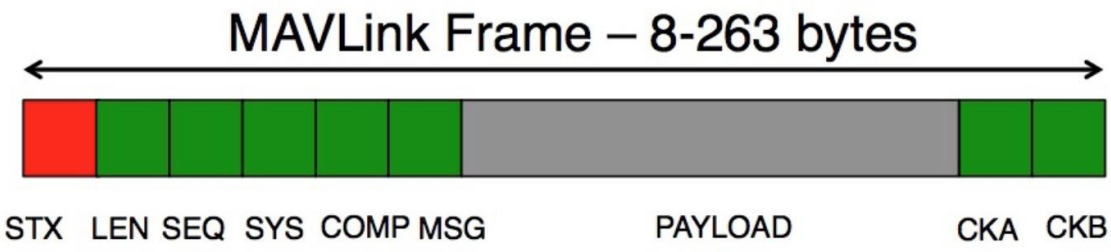


Fuente: Ardupilot [53]

3.4.1.1 Trama

Cada mensaje tiene una trama de 8 bytes en su versión 1 y 14bytes en su versión 2

Figura 23: Trama MAVLink



Fuente: Pedro Albuquerque [54]

- **STX** indica inicio de nuevo paquete.
- **LEN** indica longitud del paquete.
- **SEQ** indica la secuencia, para detectar pérdida de paquetes.
- **SYS** indica la ID del sistema que envía el mensaje (por ejemplo GCS o Submarino).
- **COMP** indica la ID del componente que envía el mensaje (por ejemplo IMU o Barómetro).
- **MSG** indica el tipo de payload.
- Después del **PAYLOAD** existen 2 bytes de checksum, **CKA** y **CKB**.

En el payload se codifica el sistema y componentes de destino, utilizando el valor 0 como valor de broadcast para todos los componentes.

El componente que recibe la orden decide si aceptarla o rechazarla dependiendo del SEQ y SYS de origen. Si una instrucción se envía con SEQ y SYS 0 el componente destino la acepta.

3.4.1.2 Mensajes

Todos los parámetros presentes en Ardupilot se pueden configurar vía mensajes MAVLink. Los mensajes más utilizados se pueden encontrar en la página web oficial. El protocolo permite crear mensajes personalizados. [55]

Los mensajes de más interés para el sistema a desarrollar en este Trabajo de Fin de Máster (TFM) son los siguientes.

- ATTITUDE
- POWER_STATUS
- VFR_HUD
- SERVO_OUTPUT_RAW
- RC_CHANNELS_RAW
- SCALED_PRESSURE2
- SYS_STATUS
- NAV_CONTROLLER_OUTPUT
- GLOBAL_POSITION_INT
- EKF_STATUS_REPORT
- BATTERY_STATUS

4 SISTEMA DESARROLLADO

El sistema de control de un submarino y las características de su entorno han sido tratadas en trabajos anteriores [1]. Como se ha visto en este TFM, el Sibiu Nano + está formado por un conjunto heterogéneo de sistemas. Dando lugar a 3 dispositivos en los que se puede llevar a cabo el control. Cada uno de ellos con una tasa de refresco y nivel de abstracción diferentes.

En este apartado se van a desarrollar distintas estrategias en cada uno de ellos.

Analizando las características principales de cada uno:

- Pixhawk: Código abierto de difícil acceso y desarrollo pero gran tasa de refresco permitiendo trabajar en tiempo real. La memoria y capacidad de procesamiento disponible es limitada.
- Raspberry: Fácil acceso y desarrollo. Utiliza una versión EOL de Raspbian que dificulta la actualización e instalación de aplicaciones.
- Estación Base: Fácil acceso y desarrollo, mejor potencia computacional. Se encuentra situado fuera del ROV.

Teniendo en cuenta que las características del entorno submarino hacen que el movimiento del submarino sea lento, se puede despreciar el retraso presente en la transmisión de datos entre el submarino y la Estación de Tierra. Existe una limitación de comunicación que, a efectos prácticos, limita la tasa de refresco de los mensajes externos a la Pixhawk a 10Hz.

Así se realiza una división en 2 posibles estrategias de diseño del control:

- Una a bajo nivel, desarrollando el control en la Pixhawk modificando el código de Ardusub, permitiéndonos realizar un algoritmo de control lo suficientemente rápido.
- Una externa, que utiliza el análisis de datos e indicaciones del usuario para enviar consignas y comprobar el correcto funcionamiento, utilizando mensajes MAVLink.

La solución adoptada combina ambas estrategias, realizando una etapa de HAL, drivers de sensores y actuadores, controladores de bajo nivel y funciones de comportamiento a bajo nivel en la pixhawk y una etapa de navegación, planificación, interfaz de usuario y análisis de datos a nivel superior en la Estación de Tierra.

Por seguridad, la etapa de alto nivel se ha clonado en la Raspberry presente en el interior del submarino para, en caso de desconexión del submarino con la Estación de Tierra, este entre en un estado de emergencia, deteniendo su movimiento y si pasado un tiempo no se recupera la conexión, tratar de volver al punto inicial de forma autónoma.

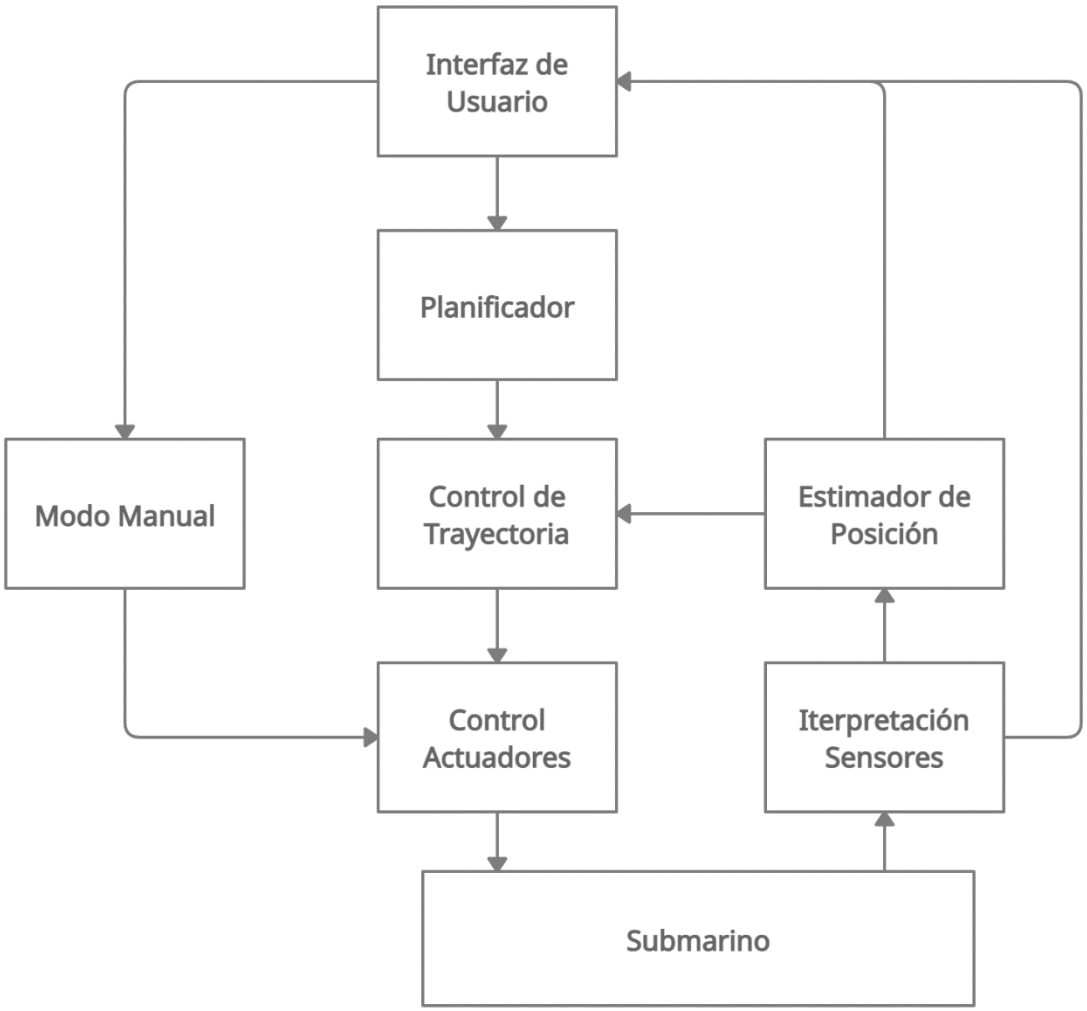
4.1.1 Requisitos

Como se ha visto al principio de este TFM, el objetivo del control es ser capaz de regular las variaciones de pitch y roll manteniendo el dron estable y alcanzar una determinada profundidad y posición X e Y indicada por un usuario.

4.1.2 Sistema previo

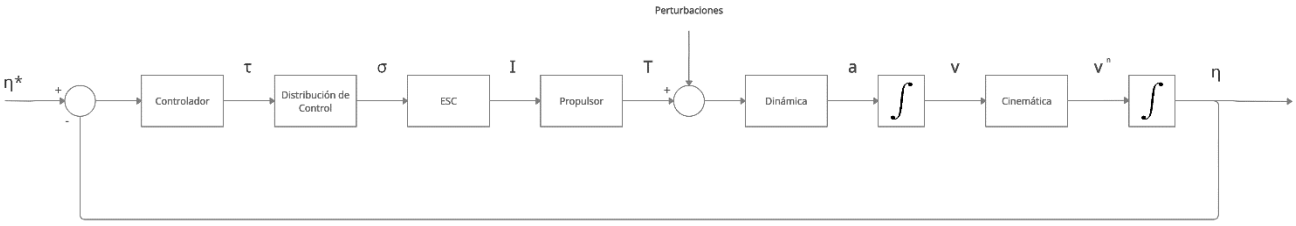
En el Trabajo de Fin de Grado *Modelado, Simulación y Control de un Vehículo Submarino Ligero*[1] se realizó un estudio de las características y comportamiento del medio submarino y se creó un modelo del Sibiu Nano + evaluado en simulación. En él se propone el siguiente sistema.

Figura 24: Diagrama de Aplicación en el trabajo previo



Fuente: Alejandro Casado Pérez [1]

Figura 25: Funciones de transferencia del control en el trabajo previo



Fuente: Alejandro Casado Pérez [1]

d

4.1.3 Problemas del Sistema Real

Tras analizar el sistema real, se han encontrado las siguientes diferencias respecto a lo planteado anteriormente.

- El modelo presenta diferencias respecto al submarino real, los valores de PID tienen que ajustarse de nuevo.

- El simulador utilizado, pese a ser potente, ha sufrido cambios desde la realización del trabajo anterior debido a problemas de físicas no realistas. No se puede esperar que el submarino se comporte de la misma forma
- En simulación la entrada del submarino utilizaba la potencia consumida por cada motor, en el submarino real se trabaja con señales PWM. Es necesario realizar una traducción.
- El código por defecto del submarino no contempla la acción individual sobre cada motor, ya que el código de bajo nivel realiza la descomposición de fuerzas y ajustes de cada motor automáticamente. No permite actuar sobre cada motor de forma individual.
- La posición real del submarino no es perfecta, durante la ejecución existirán errores de posición. En algunas situaciones como ausencia de satélites, pérdida de conexión o interferencias estos errores pueden no asegurar la convergencia de datos.
- Al realizar lecturas de mensajes desde el exterior Pymavlink las gestiona internamente como una cola con una espera (delay), provocando una lectura de datos asimétrica y en muchos casos con un retraso que aumenta indefinidamente con el tiempo.

4.2 Implementación

En este TFM se valoran diferentes formas de implementar el control, cada una con sus ventajas e inconvenientes dependiendo de la aplicación final que se le dé al submarino. Esta división dicta donde sucede el control, dando así las siguientes posibilidades:

- Dos estrategias de bajo nivel: La Pixhawk se encarga del control de los motores, o cede el control a la estación de tierra
- Dos estrategias de alto nivel: Via MAVROS y via Pymavlink.

Pudiendo realizar cualquier tipo de combinación entre ellas.

Antes de entrar en detalle, indicar que la implementación de este control en el submarino requiere realizar algunas consideraciones previas para que todo siga funcionando.

4.2.1 Consideraciones previas

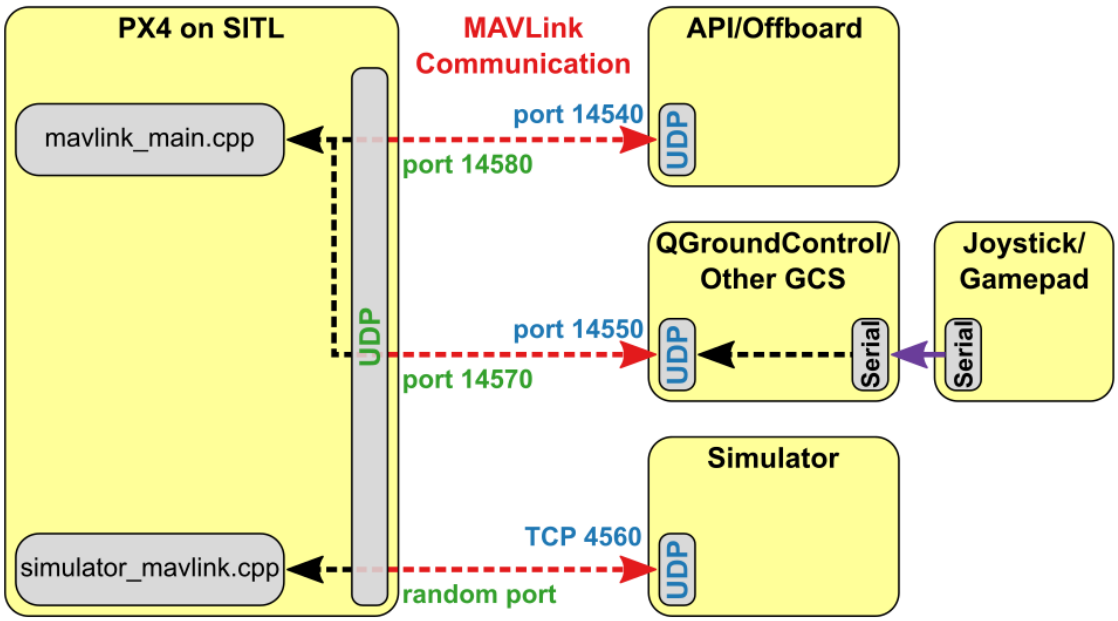
4.2.1.1 Configuración de puertos

Ardupilot reserva un puerto para cada conexión. Esto significa que necesita un puerto para cada programa que lea telemetría, cambie el estado del submarino o quiera introducir una acción de control en los motores, como se puede apreciar en la figura 26.

El puerto utilizado para la interfaz de usuario de QGroundControl es el 14550. Modificando el archivo mavproxy.param ubicado en la carpeta raíz de la Raspberry Pi tal y como se indica en el Anexo 7.2. Añadiendo un nuevo puerto capaz de acceder desde fuera especificando protocolo UDP o TCP en broadcast (0.0.0.0).

Para experimentos futuros, los puertos 8001-8003 serán usados ya que en caso de cierre inesperado del programa el puerto activo quedará bloqueado hasta que se reinicie el sistema (bien desconectando la alimentación o ejecutando mediante terminal o script “sudo systemctl restart ardusub”).

Figura 26: Puertos de comunicación MAVLink



Fuente: Ardupilot [50]

4.2.1.2 Interferencias de órdenes

Cualquier dispositivo conectado en la red puede conectarse a la Pixhawk siempre que el puerto de acceso no esté ocupado. Esto puede provocar problemas si dos dispositivos tratan de enviar órdenes a un mismo vehículo. Este puede ser el caso de las órdenes enviadas por QGroundControl a través del mando o la interfaz contra las órdenes enviadas por el sistema de control desarrollado. La herramienta utilizada para solventar este problema es inherente al protocolo MAVLink.

La trama MAVLink posee un campo obligatorio SYS, campo que actúa como identificador del sistema que envía el mensaje. Cuando se recibe una orden el sistema la compara con parámetro propio llamado MYSYS_IDGCS. Si el valor coincide con el campo SYS la orden se ejecuta, en caso contrario la orden se rechaza.

Por defecto este parámetro tiene valor 255, mismo valor que QGroundControl tiene por defecto. Este valor se puede cambiar a partir de un mensaje de configuración, permitiendo cambiar el control entre QGroundControl y el programa de navegación.

4.2.1.3 GPS

En el punto 3.1.1 se realizó un estudio del funcionamiento del sistema GPS. Debido a que los cálculos de posición se realizan en la base ubicada en tierra (Topside) el submarino no tiene ninguna forma de saber su posición si no tiene conexión a esta. Luego, si se pierde la conexión con la Estación Base el vehículo pierde toda referencia de posición.

Además, para que el submarino sea capaz de recibir información GPS e incluirla en el EKF presente en Ardusub es necesario realizar dos configuraciones:

- En Ardusub es necesario modificar dos parámetros: **SERIAL3_PROTOCOL** debe utilizarse como puerto de entrada de comunicación GPS a través de MAVLink2 mediante el valor "**2**" y para que el valor GPS se introduzca en el EKF es necesario modificar el tipo de GPS a MAVLink mediante el parámetro **GPS_TYPE** "**14**".
- El valor de posición del submarino se obtiene de una página web. Es necesario recoger los valores de GPS y enviarlos al submarino a través de protocolo MAVLink. Para ello se ha

creado un script que utiliza la API de Water Linked para descargar los datos y enviar el mensaje MAVLink en un hilo a parte para mantener la concurrencia.

4.2.2 Implementación de alto nivel

4.2.2.1 ROS

ROS es un framework robótico ampliamente utilizado en todo tipo de vehículos y flotas. Tiene un gran número de ventajas las cuales lo hacen una buena estrategia de diseño como se trató en el trabajo anterior [1]. Siguiendo este diseño, Bluerobotics diseñó un paquete de ROS llamado MavROS que lograba romper la barrera de MAVLink realizando una traducción bidireccional completa de los mensajes mediante el uso de topics y servicios. Permitiendo recibir telemetría y enviar mensajes. Permitiendo por tanto diseñar un control para el submarino. Sin embargo, en 2016 la idea se desechó ya que el código de Ardusub no estaba finalizado y un control directo no era posible. [56]

El paquete MavROS sigue recibiendo actualizaciones incorporando nuevos mensajes. Sin embargo, aún no están disponibles todos los mensajes de MAVLink v1.0 y v2.0.

Por otro lado, instalar ROS en la Raspberry no es tarea fácil debido al EOL de Jessie. Anne [57] intentó instalar ROS Kame en la Raspberry Pi desde binarios. Sin embargo, esta opción ya no es posible ya que los datos se han eliminado de los servidores. Existen imágenes alternativas como BlueOS o la imagen de Emlid para Navio2, pero el cambio de imagen requiere acceder físicamente a la Raspberry Pi por lo que ROS se ha utilizado únicamente en el ordenador de tierra.

Indicar que MavROS presenta problemas de compatibilidad con ROS2, permitiendo lectura de mensajes y estado del dron pero no envío de instrucciones.

4.2.2.1.1 Sistema

Se han diseñado dos nodos de ROS en Python que se pueden observar en el Anexo 7.4.2:

- Logger: Guarda los datos del comportamiento del submarino en un archivo para posteriormente realizar una representación gráfica de datos.
- Controller: Utiliza mensajes de Pymavlink para enviar las órdenes al submarino, posée un control PID para cada grado de libertad tal y como se hizo en el trabajo anterior.

4.2.2.2 Pymavlink

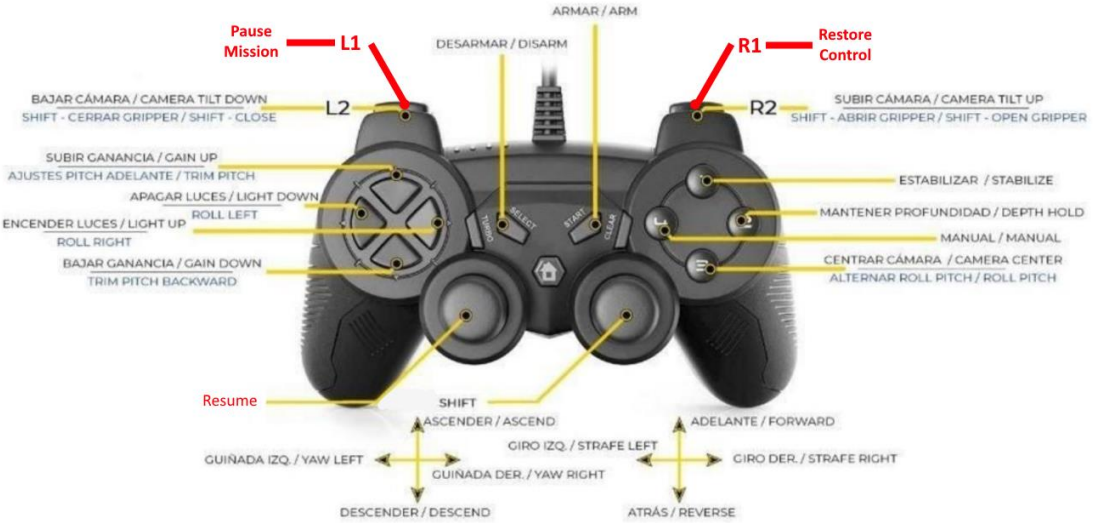
Pymavlink es una librería de Python que facilita al desarrollador la conexión, lectura y envío de mensajes a través de MAVLink. Existen otras librerías de un nivel más alto como Dronekit que hacen uso de Pymavlink para hacer invisible todo el protocolo, facilitando así el desarrollo de código y el tiempo necesario para ello.

El código desarrollado comprende los siguientes bloques

- Script de log de datos: Incluye timestamp de todos los datos y estados por los que pasa tanto el dron como el programa
- Script de comunicaciones: Gestiona todas las comunicaciones del vehículo leyendo el estado de este mediante callbacks de mensajes y codifica mensajes para que el vehículo se dirija a una profundidad, oriente hacia una posición, dirija hacia un punto o cambie de modo de operación.
- Script de supervisión: Se encarga de comprobar que todos los módulos funcionan correctamente. Si un módulo falla o el operario detecta un error permite poner el submarino en un modo seguro o detenerlo completamente.

Se utiliza el mando como herramienta externa para el control del submarino. Se le ha otorgado funciones adicionales como detener el programa (hardreset), pausar el programa, y reanudarlo

Figura 27: Controles con funciones secundarias



Fuente: Elaboración propia a partir de Ardusub [46]

Las referencias del submarino se envían a través de los mensajes MAVLink:

- SET_POSITION_TARGET_GLOBAL_INT: Permite fijar referencias de posición, velocidad y/o aceleración (3 ejes de posición).
- SET_ATTITUDE_TARGET_MESSAGE: Permite fijar las referencias de orientación del vehículo (3 ejes de Rotación).

El código asociado se puede ver en el anexo 7.4.3

4.2.3 Implementación de bajo nivel

Para poder actuar sobre cada motor de forma individual es necesario poder enviar a cada uno individualmente una señal PWM. Existen una serie de estudios de cómo controlar los motores automáticamente [58]. Esta función se puede realizar mediante mensajes MAVLink de tres formas diferentes:

- **RC_OVERRIDE** y **MANUAL_CONTROL** permiten escribir directamente en las señales PWM enviadas por el RC. Sin embargo, este método presenta el problema de no poder acceder directamente a los motores, si no al mapa provisto por ardupilot, donde se puede indicar el movimiento indicando la dirección (frontal, lateral, profundidad, roll, pitch y yaw) y no señales enviadas a cada motor.
- **MAV_CMD_DO_SET_SERVO** y **SERVO_OUTPUT_RAW** permiten enviar señales PWM a cada canal servomotor. Sin embargo, internamente los motores son un tipo especial de servomotor con una máscara de seguridad que impide modificar su salida usando estos mensajes.
- **MAV_CMD_DO_MOTOR_TEST** permite enviar una señal PWM a los motores. Sin embargo, el mensaje permite controlar un único motor a la vez, impidiendo establecer una señal en los 8 motores a la vez.

En todos los casos es necesario utilizar un script de Python para la comunicación Tierra-submarino, no pudiendo ser implementado en QGround Control (requiere desarrollo en C# de un programa, el código es abierto, pero requiere demasiado tiempo de desarrollo para las ventajas que obtenemos).

4.2.3.1 Ardusub Mode

La primera opción requiere diseñar un nuevo modo de control en Ardupilot, cambiando el mapa de los motores de forma que el RC pase sobre el primer mapa logrando enviar una señal directa a cada motor.

El modo implementado se puede observar en el Anexo 7.4.4.

4.2.3.1.1 Reproducción del sistema

Para reproducir los cambios realizados en el submarino se puede reutilizar la versión de Ardupilot instalada en la Raspberry. Para ello es necesario clonar el respositorio

"https://github.com/AloePacci/ardupilot"

Compila el código mediante *'./waf sub'*

Cargarlo en la Pixhawk mediante *'./waf –target bin/ardusub --upload'*

En caso de fallo de configuración utilizar *'./waf configure –board Pixhawk4'*

4.2.3.2 Ardusub bypass

La segunda opción requiere reasignar las salidas PWM de motores a servos de forma que los podamos controlar de forma individual al coste de perder el control manual y que los motores funcionen en modo desarmado.

Esto se puede realizar a través de misión planner en la ventana de configuración del dron.

5 PRUEBAS Y RESULTADOS

EL desempeño del sistema se ha evaluado en entornos controlados lénticos (piscina de agua) donde no existen corrientes ni olas, con el objetivo de que el submarino se encuentre en un estado estable y las pruebas partan de las mismas condiciones iniciales.

Los comportamientos a analizar son los debidos a los movimientos longitudinal, transversal, vertical, cabeceo, alabeo y guiñada con el objetivo de controlar la orientación y posición del submarino. En el Anexo 7.1, se pueden encontrar enlaces a vídeos de las pruebas realizadas.

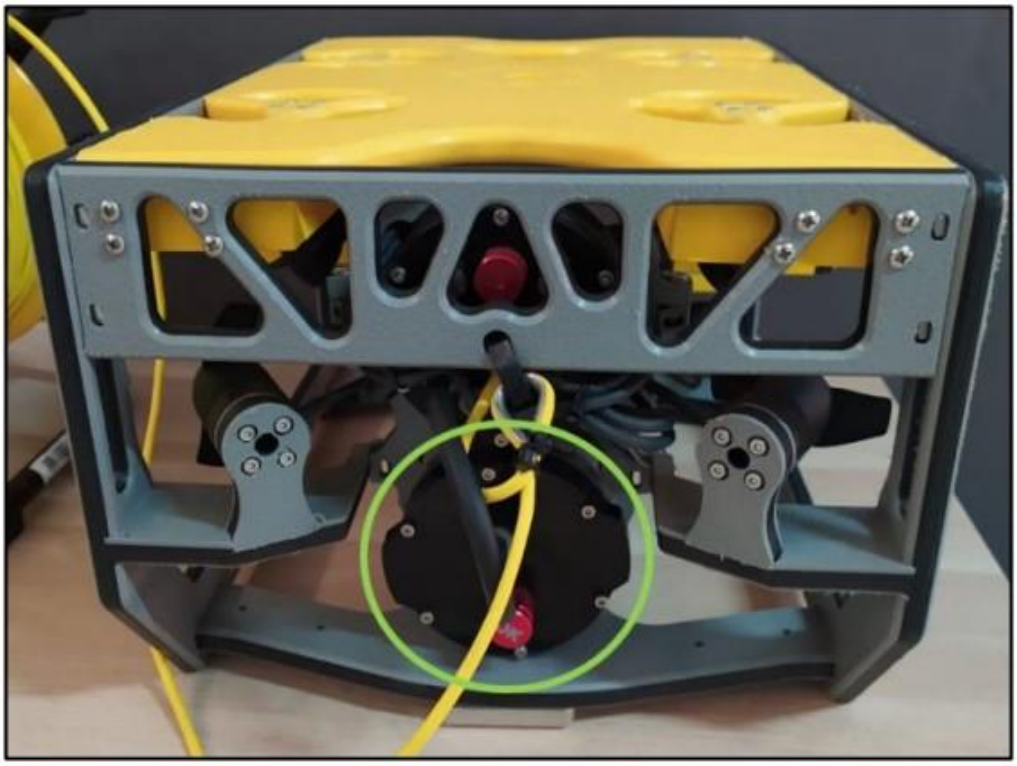
Para recoger los datos se utilizarán dos métodos dependiendo del experimento. Un nodo que utiliza MAVROS para la traducción de mensajes o un objeto de Python que recoge las lecturas de los mensajes MAVLink. Posteriormente se ha utilizado otro script para la representación de los datos.

5.1 Buenas Prácticas

Antes de realizar el experimento es conveniente realizar una serie de buenas prácticas con el objetivo de evitar errores, comprobar que todo funciona correctamente y que es seguro realizar experimentos previniendo romper la electrónica del submarino y otros componentes, que son las siguientes

- Hacer un autotest de los motores, comprobando que todos funcionan.
- Comprobar el nivel de batería antes de la inmersión.
- Activar las notificaciones de batería, manteniendo el nivel siempre por encima de los 13V.
- Desatornillar el tornillo de presión, cerrar la tapa de la cámara hidrófuga y volver a atornillar el tornillo. Este paso es bastante importante para mantener a la batería y la electrónica alejada del agua y la humedad.

Ilustración 15: Detalle Sibiu Nano Plus



Fuente: Nido Robotics [45]

- Situar el Topside de Water Linked paralelo a la antena y lo más próximo a esta, dejando siempre el cielo descubierto para lograr una buena señal GPS.
- Encender el localizador 5 minutos antes de realizar la prueba, para un correcto funcionamiento de estos. Ña luz verde debe estar fija y para ello deben captar señal GPS.

5.2 Pruebas Modo RC

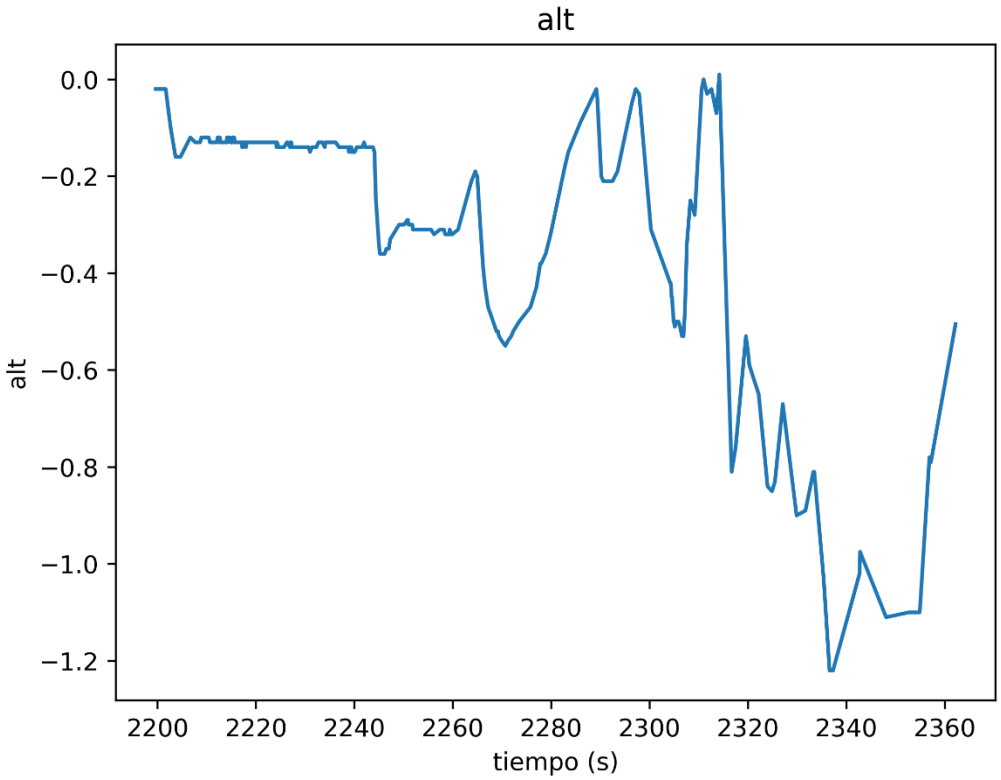
En este experimento se prestó especial atención a los movimientos del robot realizando movimientos aleatorios utilizando el mando y los tres modos de soporte a este disponibles en Ardupilot (manual, stabilize y deephold). Testeando la estabilidad, velocidad, agilidad etc.

En estos experimentos no se utilizó el sistema de posicionamiento GPS con el objetivo de estudiar el comportamiento de los sistemas de posicionamiento sin referencias de posición.

5.2.1 Gráficas

5.2.1.1 Profundidad

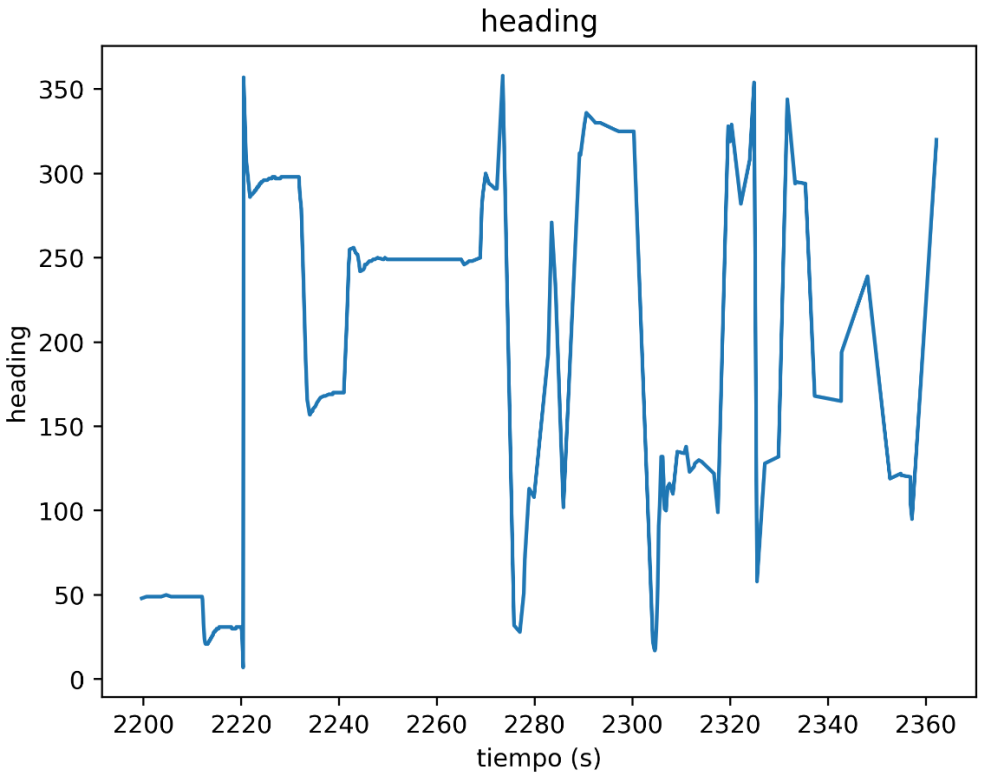
Figura 28: Profundidad modo RC



Fuente: Elaboración propia

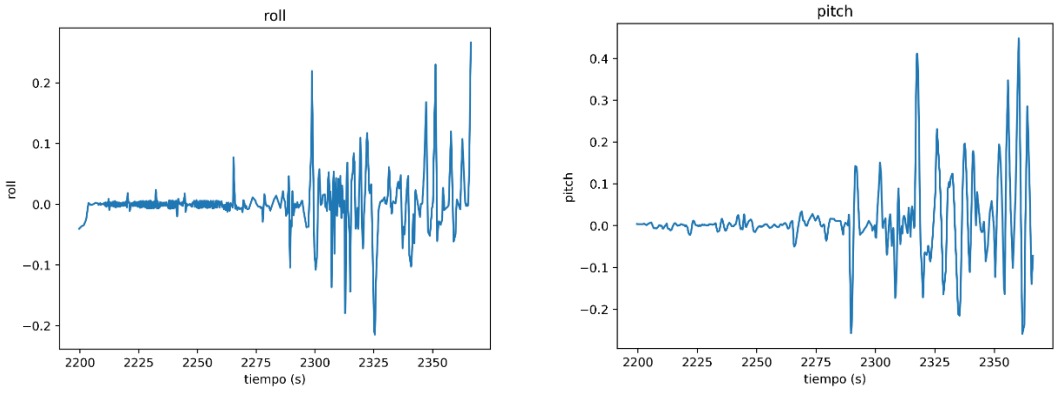
5.2.1.2 Orientación

Figura 29: Heading modo RC



Fuente: Elaboración propia

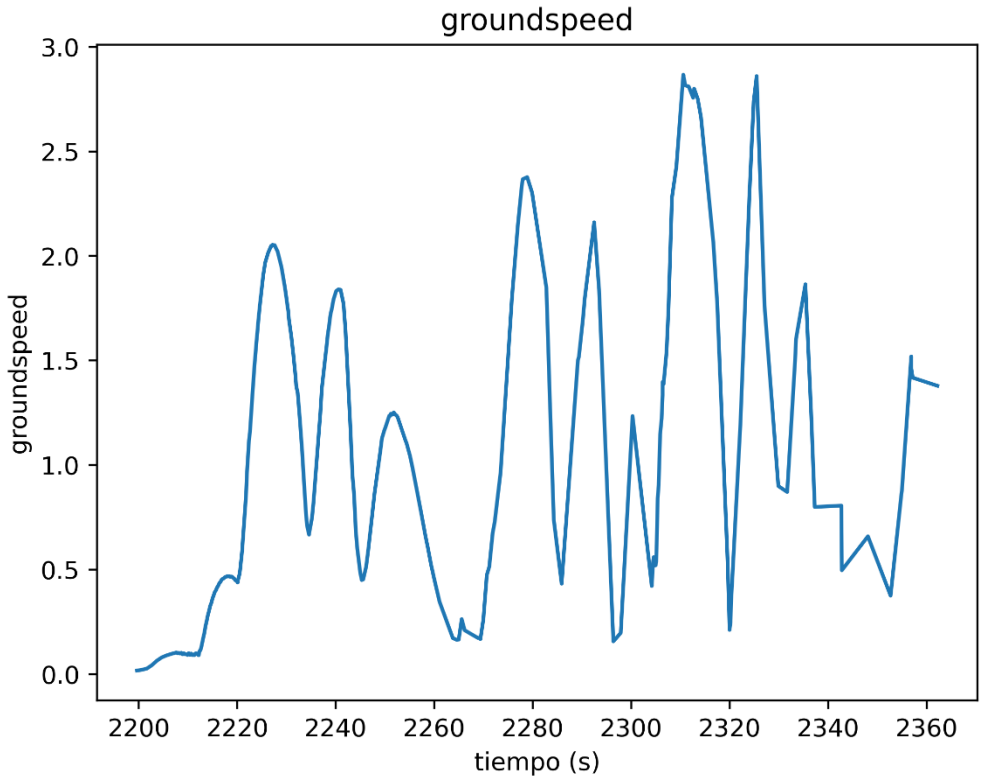
Figura 30: Pitch y Roll modo RC



Fuente: Elaboración propia

5.2.1.3 Velocidad

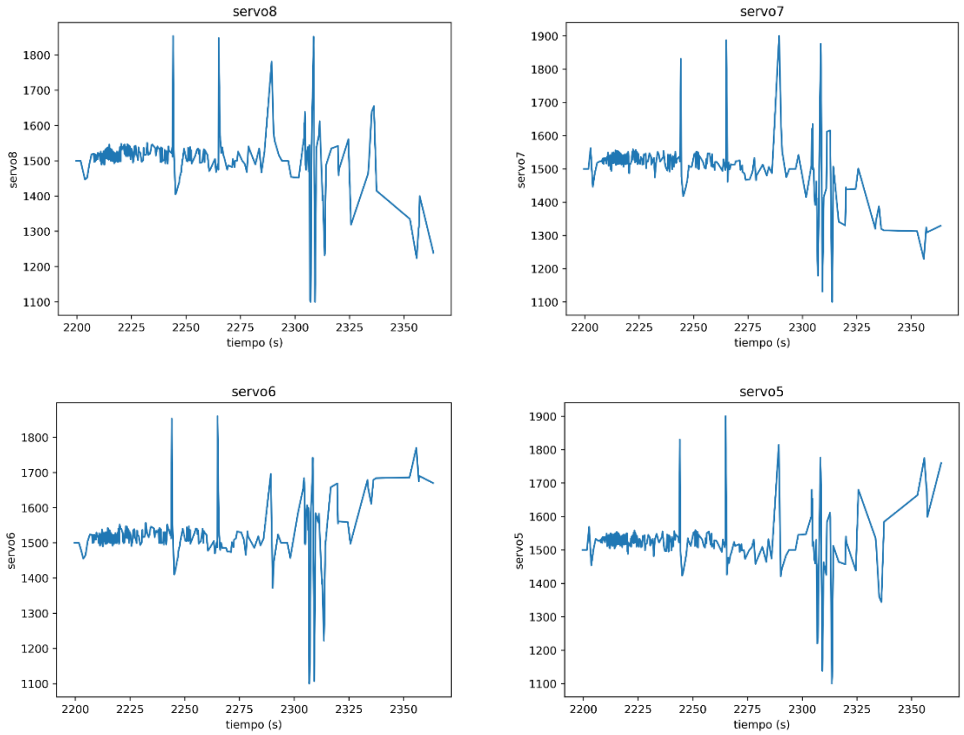
Figura 31: Velocidad modo RC

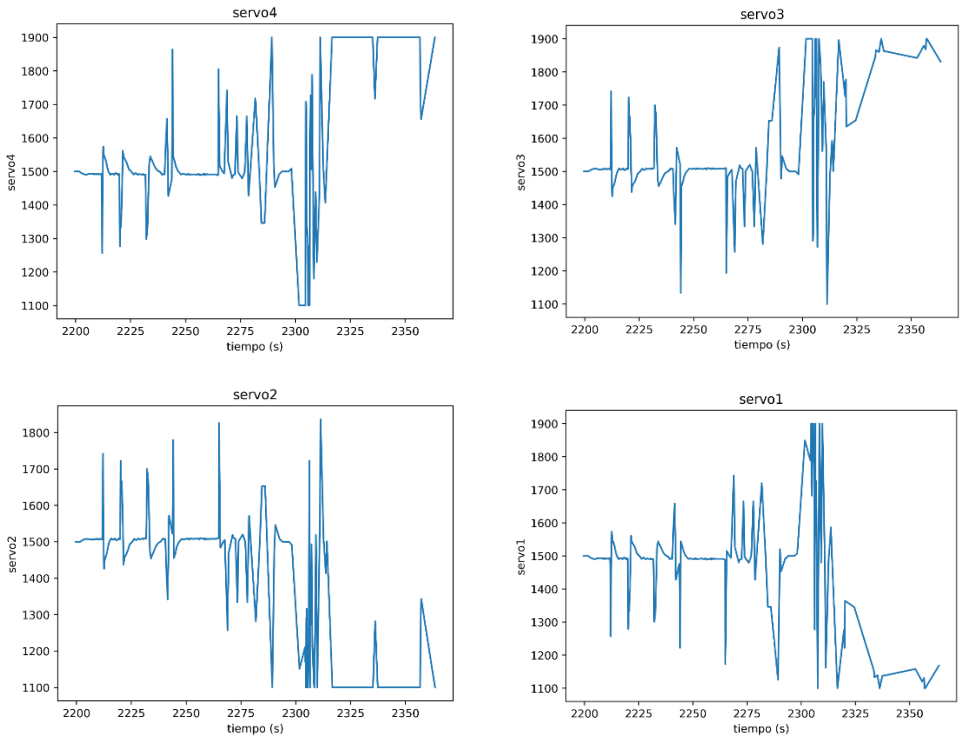


Fuente: Elaboración propia

5.2.1.4 Motores

Figura 32: Señales entrada ESC modo RC

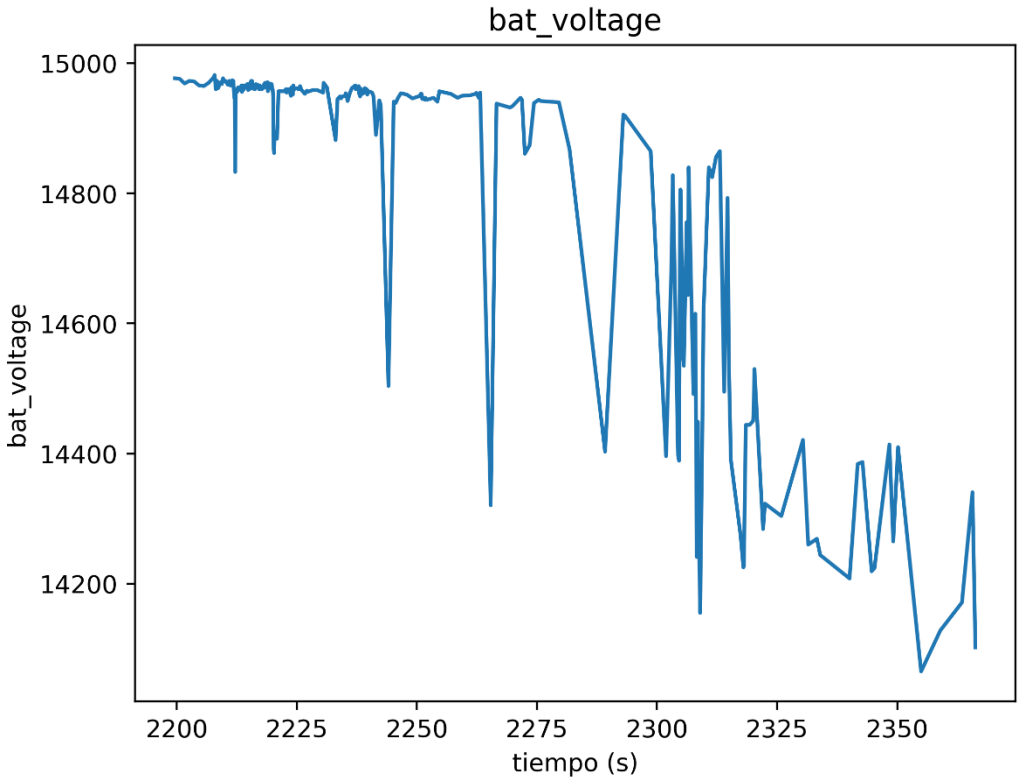




Fuente: Elaboración propia

5.2.1.5 Batería

Figura 33: Evolución del voltaje de la batería



Fuente: Elaboración propia

5.2.2 Resultados

Los resultados de estabilidad fueron satisfactorios en los que el roll y el pitch no se separaron del plano horizontal en gran medida. El mayor efecto se detectó con el arrastre del cable. Si se realizan movimientos más agresivos el robot empieza a desbalancearse realizando incluso giros de 360º. Además, aunque el comportamiento es bueno, los modos que el submarino posee por defecto no logran que mantenga el nivel de profundidad.

Analizando los datos de batería y nivel de voltaje, como se dijo en la sección de buenas prácticas, es importante comprobar que el estado de carga es suficiente antes de la inmersión. Sin embargo, como se aprecia en las figuras, las lecturas de voltaje no son fiables durante el experimento.

En lo que respecta a la duración de la batería, esta depende de diferentes factores. Limitar la salida de los motores, saturando antes de los valores límite 1100-1900 se ahorrará suficiente batería. Por otro lado, el uso de lastre ayuda a que el submarino tenga flotabilidad neutra, ahorrando la potencia utilizada en que los propulsores mantengan la profundidad del submarino.

Los resultados determinan que los datos de odometría son buenos, confirmando que se puede utilizar un buen control de profundidad y orientación. Por otro lado, al carecer de señal GPS se necesitan sensores diferentes o sistemas para realizar el control de posición en el plano ya que a partir de pocos segundos estos muestran errores no aceptables.

5.3 Pruebas Control externo

Este lazo de control se realiza en el ordenador de tierra. El usuario indica una referencia de posición en el plano XY, de profundidad en el eje Z y de orientación en el eje de Yaw (Guiñada).

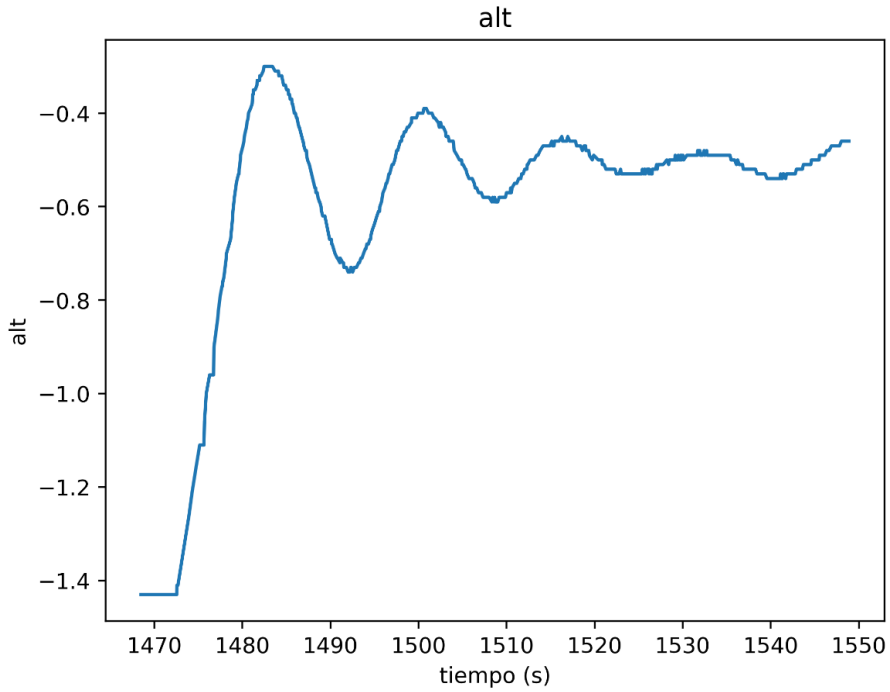
En tierra se reciben los datos de los sensores, se filtran y transforman y se introducen en una serie de PIDs para obtener la salida correspondiente para cada uno de los motores.

El mando se puede utilizar como herramienta externa para parar la prueba y recuperar el control manual del vehículo.

5.3.1 Gráficas

5.3.1.1 Profundidad

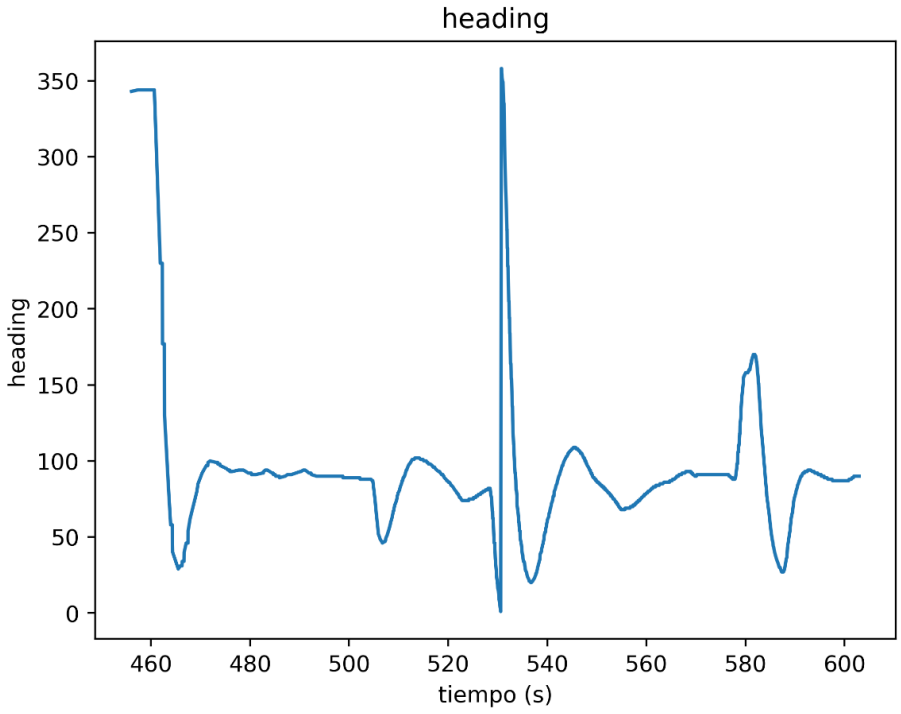
Figura 34: Control de profundidad externo



Fuente: Elaboración propia

5.3.1.2 Orientación

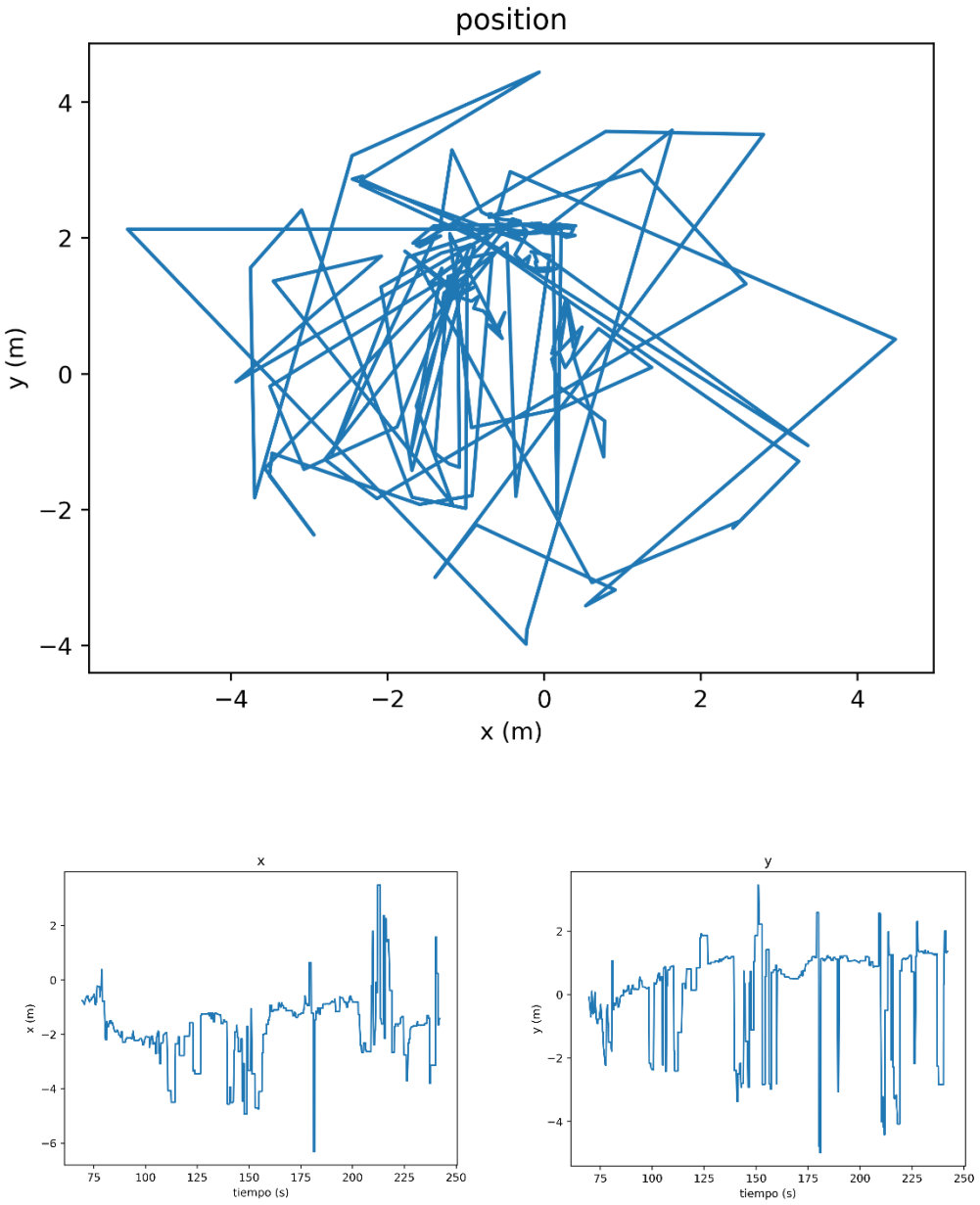
Figura 35: Control de orientación externo



Fuente: Elaboración propia

5.3.1.3 Posición

Figura 36: Control de posición externo



Fuente: Elaboración propia

5.3.2 Resultados

El primer test realizado con el control mostró que el ruido debido a las componentes derivativas de los controladores se anula debido a que los motores reales poseen una zona muerta.

Las señales originales provistas por la entrada del simulador (en vatios) se han remapeado en señales PWM para cada propulsor, con un valor entre 1100-1900.

Respecto al control. Cuando la comunicación es bidireccional aparecen una serie de retrasos que provocan una actualización asimétrica y de frecuencia variable de los sensores (de 0.5Hz a 10Hz) y en algunos casos se producen colas en la comunicación de datos de los sensores provocando un retraso que aumenta junto con la duración de la prueba dificultando la tarea de control. Los puntos más destacables de cada control individual son:

- Control de profundidad: Presenta una ligera sobreoscilación y una pequeña oscilación que no dura más de dos oscilaciones. El localizador GPS situado en el submarino altera la estabilidad y flotabilidad de este, aumentando en cierta medida el tiempo necesario para alcanzar el régimen permanente.
- Control de orientación: Presenta un comportamiento aceptable con una ligera sobreoscilación debida a la insuficiente tasa de refresco de los controladores y un buen tiempo de establecimiento.
- Control de posición: No presenta sobreoscilación ni oscilación, manteniéndose en la posición deseada con un margen de error inferior al metro. Su desempeño podría mejorarse utilizando un mejor filtrado de datos ya que estos muestran una gran cantidad de ruido.

5.4 Pruebas Control Interno

Este lazo de control se realiza modificando el código de Ardusub y utilizando funciones internas para el control. Las referencias de posición se indican desde tierra codificadas en mensajes MAVLink.

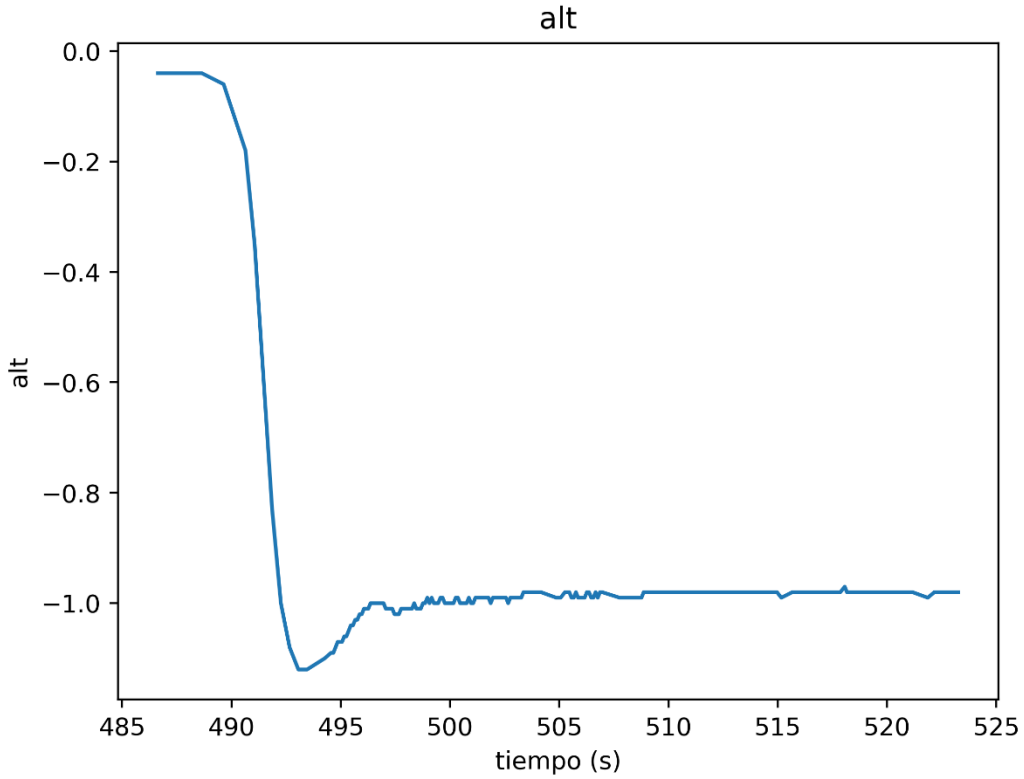
Ardusub utiliza el sistema de localización global (GPS) para la localización del submarino realimentando los datos GPS desde el ordenador de tierra.

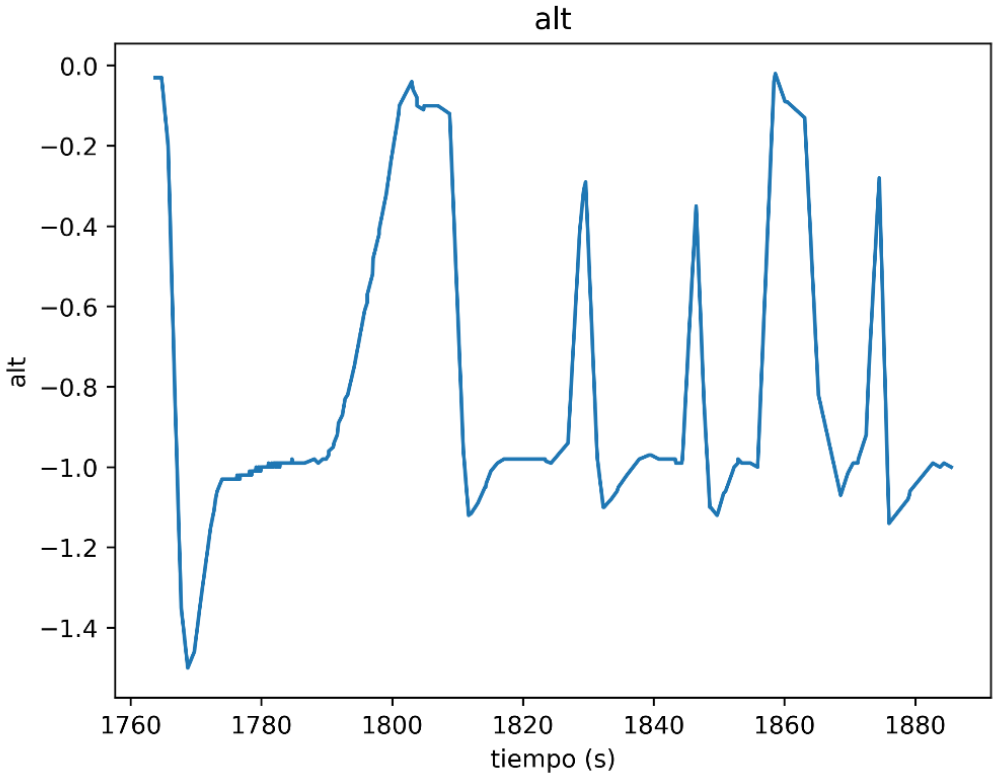
El mando se puede utilizar como herramienta externa para detener el programa de control y recuperar el control manual.

5.4.1 Gráficas

5.4.1.1 Profundidad

Figura 37: Control de profundidad interno

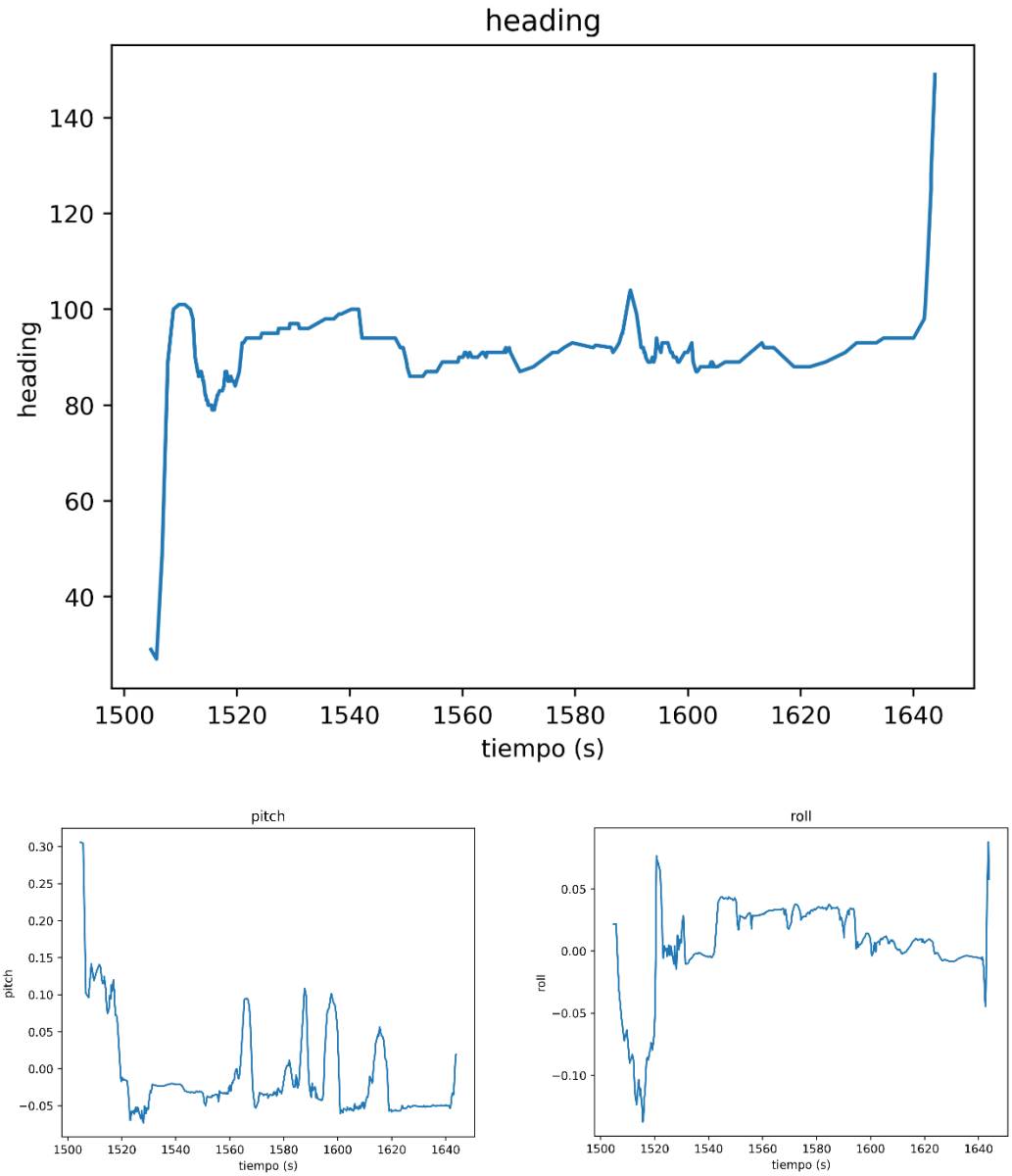




Fuente: Elaboración propia

5.4.1.2 Orientación

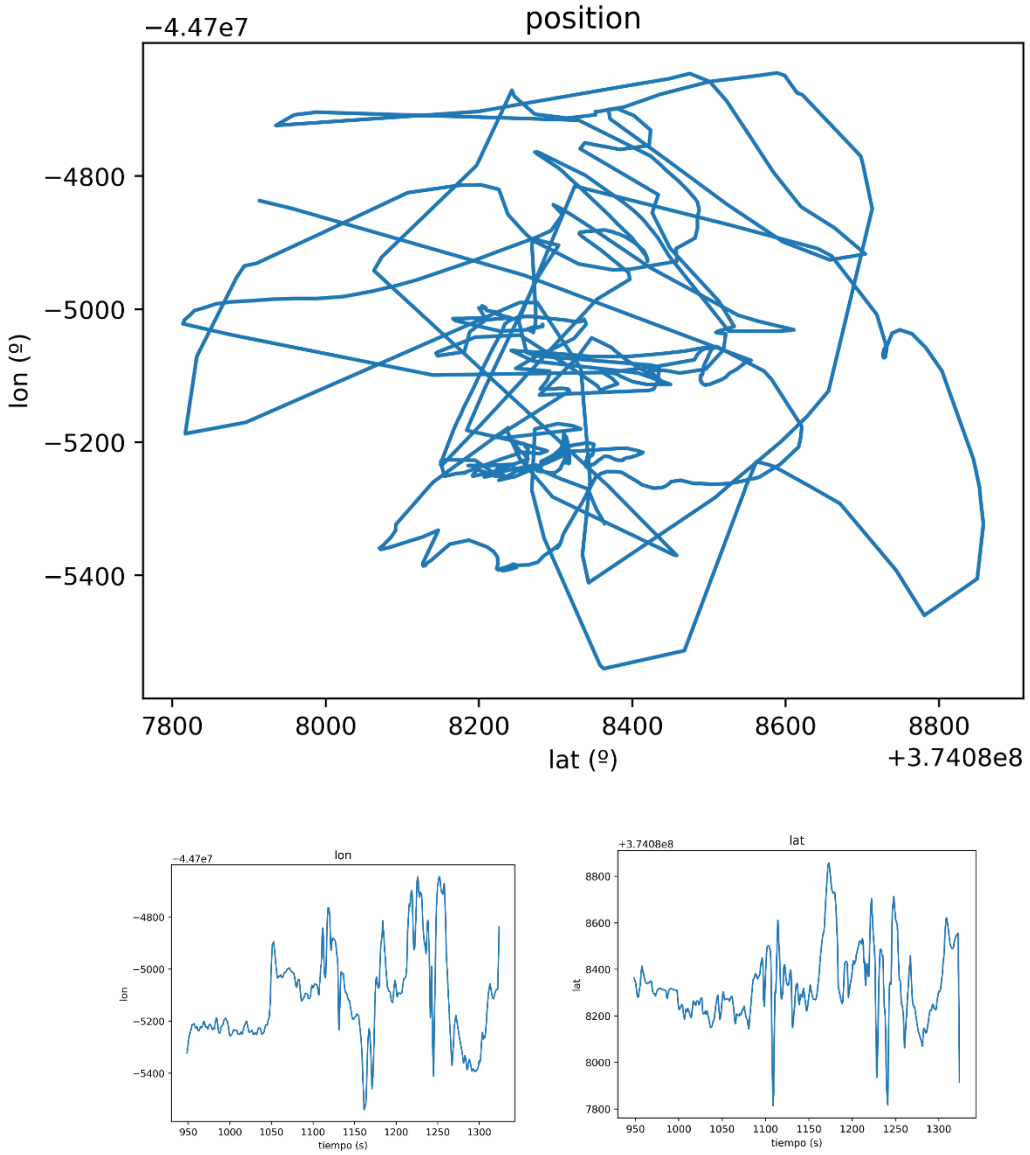
Figura 38: Control de orientación interno



Fuente: Elaboración propia

5.4.1.3 Posición

Figura 39: Control de posición interno



Fuente: Elaboración propia

5.4.2 Resultados

Las pruebas de control interno presentan comportamientos razonables y no suponen ningún gasto computacional de control para el ordenador de tierra. Los puntos más destacables de cada control son:

- Control de profundidad: Presenta una presentando una ligera sobreoscilación al cambiar la referencia. Al someter al submarino a perturbaciones externas (fuerzas en el eje vertical) la referencia se mantiene con relativa facilidad oponiendo bastante resistencia.
- Control de orientación: Presenta un buen comportamiento alcanzando la referencia casi instantáneamente. Resulta bastante difícil alterar la referencia mediante perturbaciones.

Control de posición: Como se puede observar en la ilustración 16, la posición GPS puede presentar errores. Las pruebas se han realizado en un entorno pequeño (piscina), por lo que

las tareas de geoposicionamiento presentan errores mayores al espacio de trabajo provocando que el submarino se chocase con las paredes en varias ocasiones, pero presentando comportamientos razonables.

Ilustración 16: Posición GPS

Fuente: Elaboración propia

6 CONCLUSIONES

El Sibiu Nano + como vehículo submarino ligero gobernado por Ardusub presenta una serie de características que permiten su uso como ROV o UAV en una multitud de aplicaciones submarinas.

El Trabajo de Fin de Máster desarrollado ha analizado la viabilidad del uso de este vehículo como AUV utilizando distintas estrategias de control comentadas en el punto 4.

Analizando los resultados, el submarino es capaz de desplazarse por sí solo hasta la ubicación indicada por un usuario en tierra manteniendo la posición con un radio de error inferior al metro siempre que las características ambientales lo permitan. Cumpliendo así con el objetivo principal de este Trabajo de Fin de Máster.

Comparando los resultados de control de los siguientes sistemas se han llegado a las siguientes conclusiones:

- El Sibiu Nano + no provee de las herramientas necesarias para utilizarse como AUV, necesitando del desarrollo de un programa externo para ello.
- Realizar el bucle de control en la Estación de Tierra presenta problemas de retrasos y baja tasa de actualización de datos deteriorando en gran medida la calidad de este. Por tanto, resulta de mayor interés realizar todo desarrollo y realimentación del bucle de control en la Pixhawk.
- La interfaz de usuario por defecto resulta poco intuitiva y dificulta en gran medida el uso del Sibiu Nano + como AUV. Además, no está preparada para trabajar con más de un submarino.

Comparando estos resultados con la línea de trabajo realizada anteriormente en el Trabajo de Fin de Grado *Modelado, Simulación y Control de un Vehículo Submarino Ligero* [1], se ha comprobado que la estrategia utilizada es válida logrando cumplir con los objetivos propuestos al aplicar los conceptos de simulación en el submarino real. Respecto a este TFG se han logrado las siguientes mejoras:

- Se ha realizado un análisis del funcionamiento del submarino identificando las diferencias entre el sistema simulado y el presente en el submarino real y desarrollando estrategias que permiten resolver problemas no detectados anteriormente, así como una guía de buenas prácticas.
- Se ha adaptado el programa de control propuesto a los requerimientos de entrada del submarino, proponiendo cuatro estrategias distintas.
- Los controladores se han ajustado, observando una mejora del comportamiento general del sistema al reducir el tiempo de establecimiento del control de estabilidad respecto el resto de los ejes. Este efecto es más notable debido a un centro de flotabilidad más alejado del centro de gravedad del calculado anteriormente.

La motivación principal de este Trabajo de Fin de Máster busca utilizar el sistema de control desarrollado como herramienta para la monitorización de recursos acuáticos para aplicación en el proyecto INDALO en el que participa el departamento de Ingeniería de Sistemas y Automática de la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla. En este sentido, este Trabajo de Fin de Máster propone las siguientes mejoras:

- El sistema de posicionamiento está formado por dos sistemas cuyos errores se suman. El sistema de posicionamiento global (GPS) y el sistema de posicionamiento Local (USBL). Realizar una descomposición de ambas coordenadas de posición realizando un filtrado de estas por separado permite reducir el error total.
- Desarrollar un modo de control autónomo que utilice coordenadas locales en lugar de coordenadas globales. Logrando un error de posición menor. Estas coordenadas locales se

pueden traducir a globales fácilmente en un submódulo paralelo. Además, en un entorno más amplio las reflexiones acústicas de la antena serán menores obteniendo una mejor medida de la posición del submarino.

- Trasladar el software presente a dos nuevos sistemas:
    - BlueOS en la Raspberry Pi: Logrando un sistema más moderno que permite utilizar el sistema de control en la Raspberry Pi sin necesidad de utilizar un ordenador de Tierra.
    - PX4 en la Pixhawk: Siendo un sistema más actualizado y con mayor número de funciones compatible con ROS2, resolviendo problemas de tasa de transmisión de datos y facilitando el uso de flotas de vehículos.
- Desarrollo de algoritmos de navegación autónoma que decidan la posición objetivo del submarino optimizando costes. El objetivo puede ser realizar un mapa de calor de medidas ambientales, patrullar una zona, etc.
- Uso de algoritmos de visión artificial para detección de objetos de interés y realizar una estimación de la posición del submarino con el objetivo de mantener la posición si falla el sistema USBL.
- Desarrollo de un sistema de control predictivo, permitiendo realizar un mejor control del submarino, así como reducir errores.
- Diseñar una cámara estanca para albergar sensores acuáticos o actuadores submarinos. La Raspberry Pi dispone de dos puertos USB y varios GPIO libres disponiendo de vías de comunicación suficientes para instalar sensores de calidad de agua, sónares o herramientas.

Con las aportaciones de este Trabajo de Fin de Máster se permite utilizar cualquier submarino gobernado por el sistema Ardusub como AUV, permitiendo a un operario seleccionar una lista de puntos de interés, logrando así el uso de cualquier ROV sin la necesidad de un piloto formado para realizar la tarea.

En este sentido las posibilidades de actuación del vehículo Sibiu Nano + se ven ampliamente aumentadas, logrando realizar autónomamente tareas que anteriormente requerían de un piloto, de gran interés en los sectores de la inspección submarina, estudios oceanográficos, monitorización de recursos, vigilancia y seguridad submarina.

7 ANEXO

7.1 Videos Pruebas

Control de profundidad interno	https://youtube.com/shorts/_cddA2RbDAM
Control de orientación interno	https://youtu.be/Loan73NnJNo
Control de posición interno	https://youtube.com/shorts/AmKx1RiAjFs
Control de profundidad externo	https://youtube.com/shorts/nmntNI7S1Nk
Control de orientación externo	https://youtube.com/shorts/D2wXgIJ_0Js
Control de posición externo	https://youtu.be/DKO6QKbUS9s

7.2 Mavlink.config

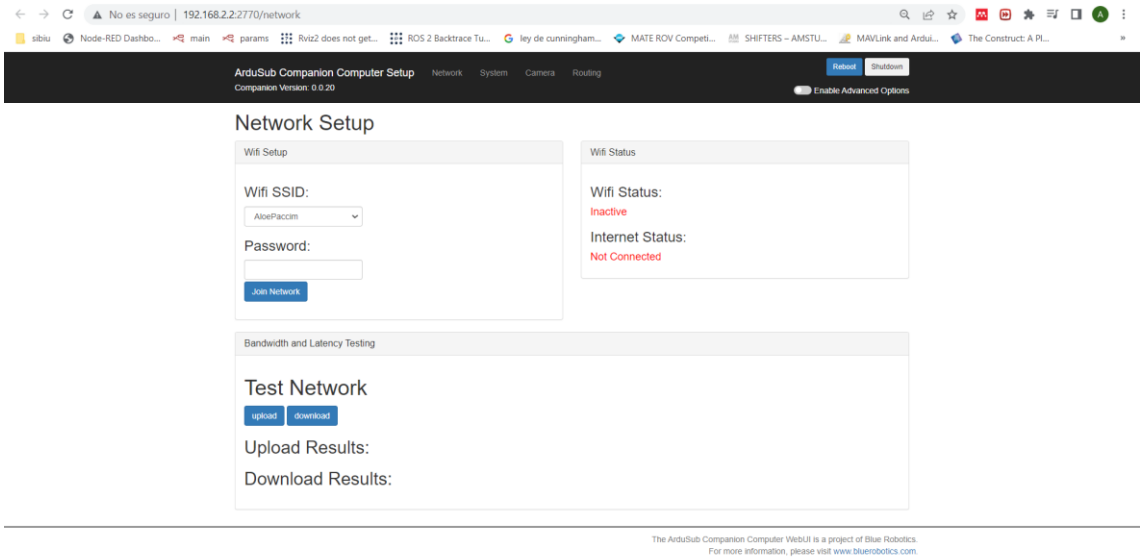
```
--master=/dev/autopilot,115200
--load-module='GPSInput,DepthOutput'
--source-system=200
--cmd="set heartbeat 0"
--out udpin:localhost:9000
--out udpout:localhost:9002
--out udpin:0.0.0.0:14660
--out udpbcast:192.168.2.255:14550
--out udpbcast:192.168.2.255:8001
--out udp:192.168.2.1:8002
--out udp:192.168.2.1:8003
--mav20
--aircraft telemetry
--streamrate 10
--out udpout:
```

7.3 Interfaz web

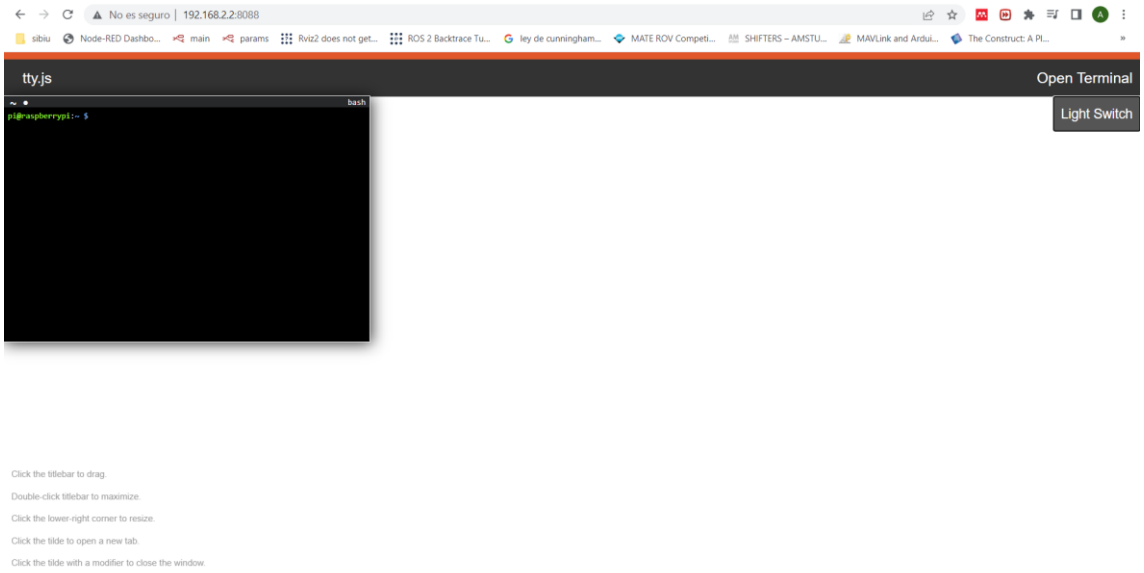
7.3.1 Interfaz Raspberry Pi

Todas estas imágenes son de elaboración propia

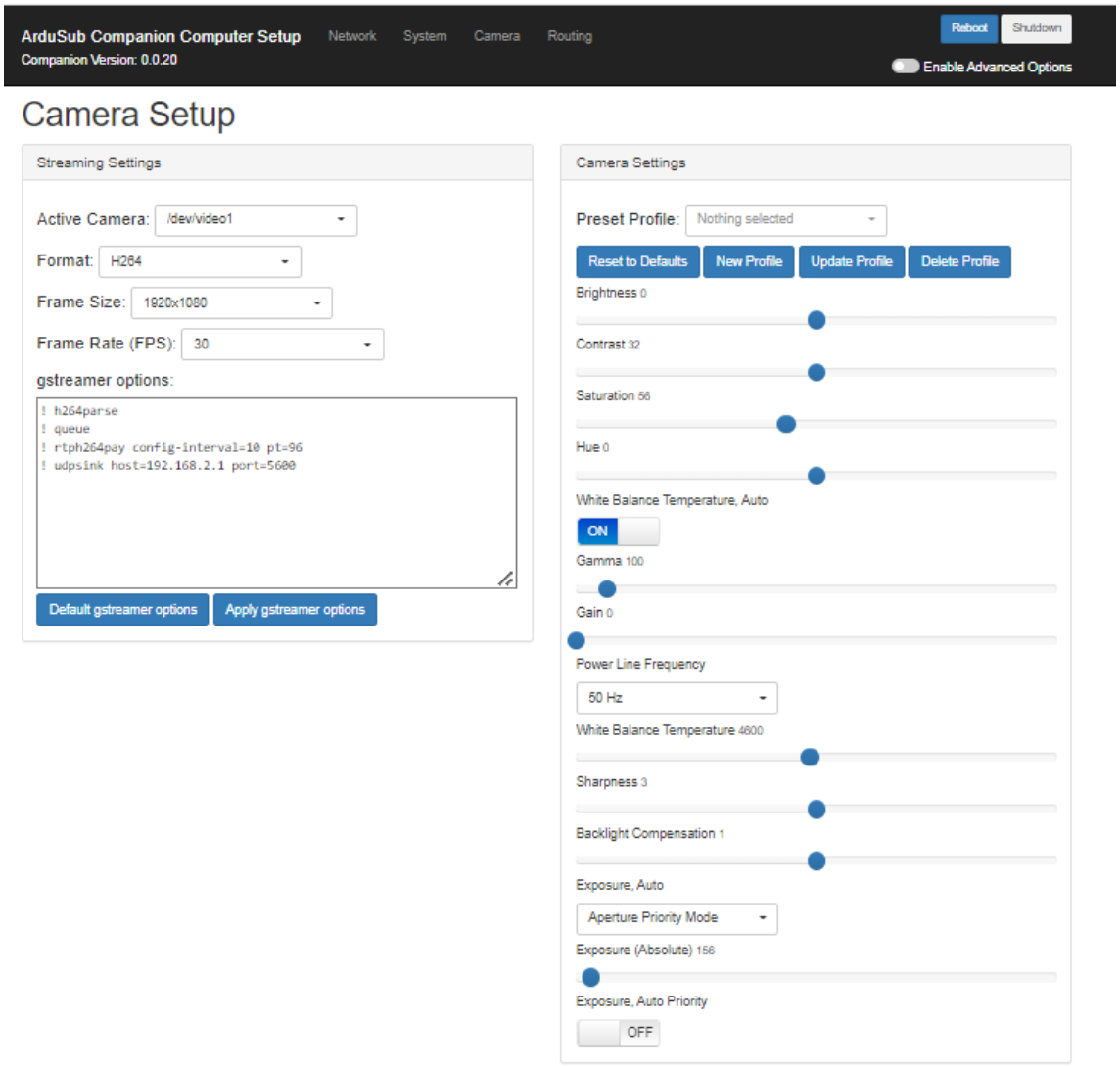
Panel de conexión de red :2770/network



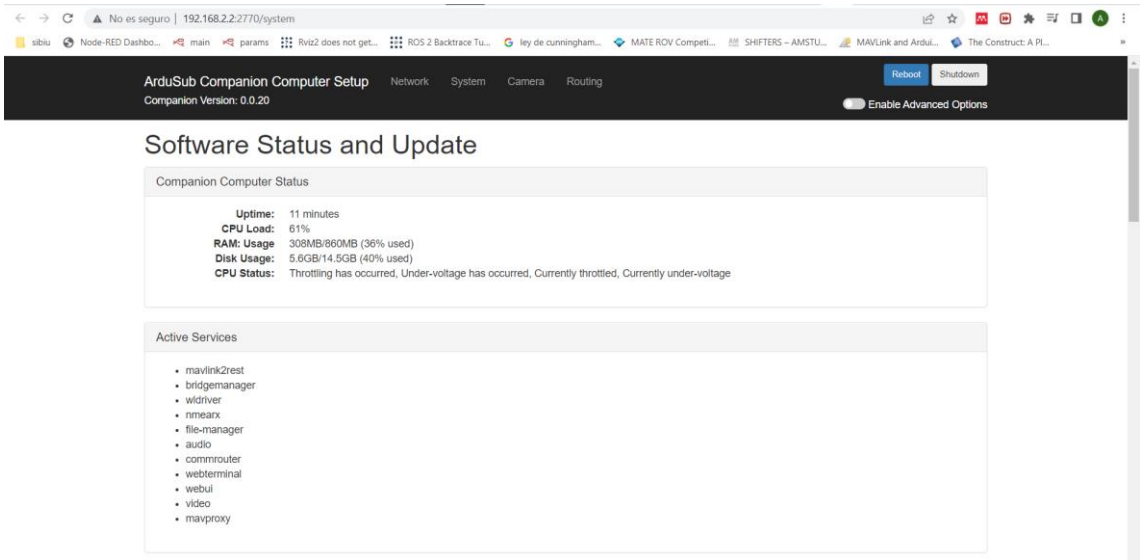
Panel de terminal :8008

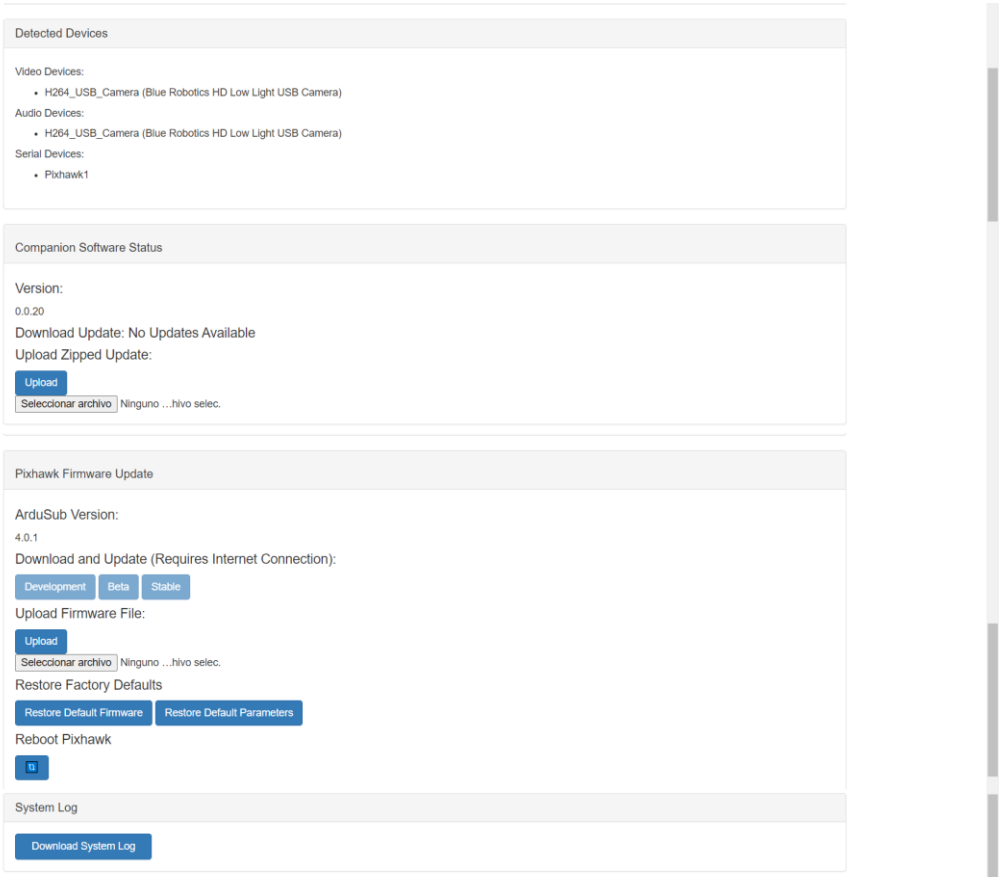


Panel configuración de Cámara :2770/camera

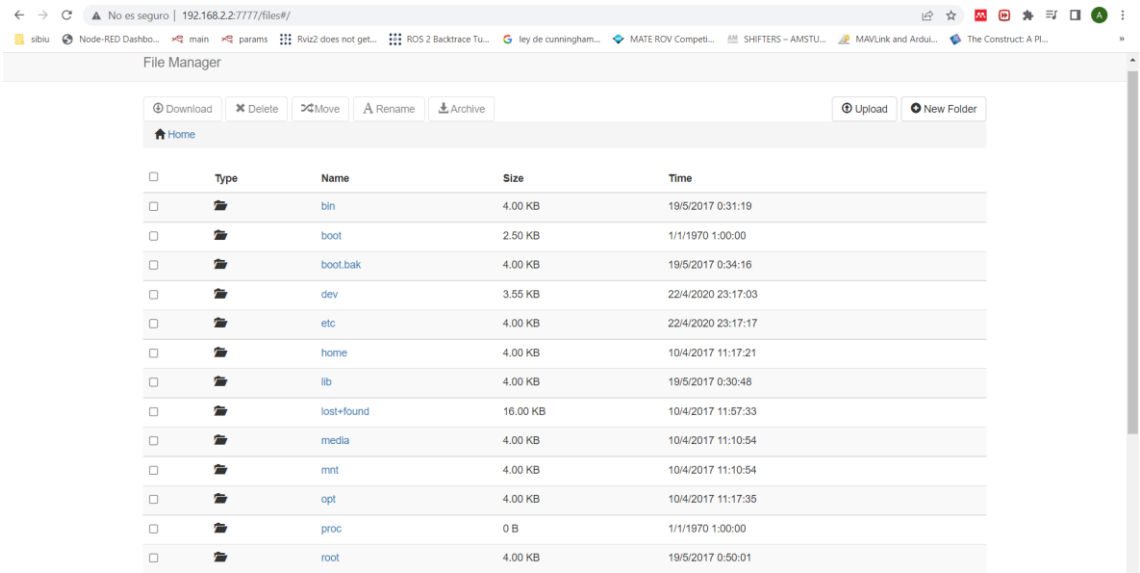


panel de ajuste del sistema :2770/system



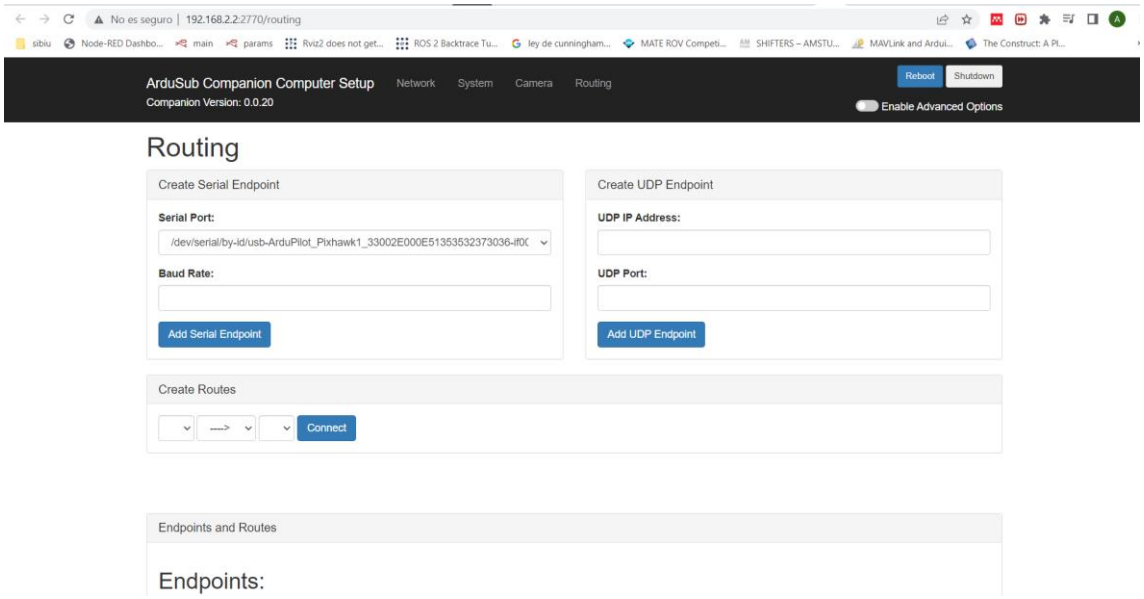


Panel de interambio de archivos :7777

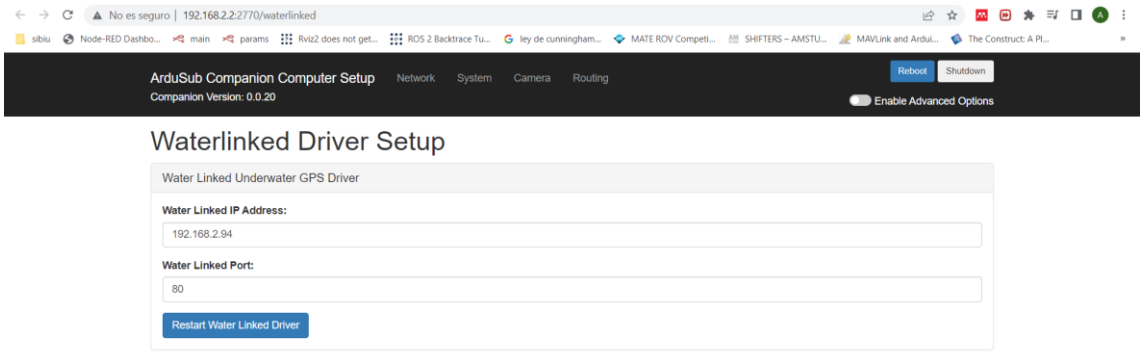


panel de mensajes mavlink :4777

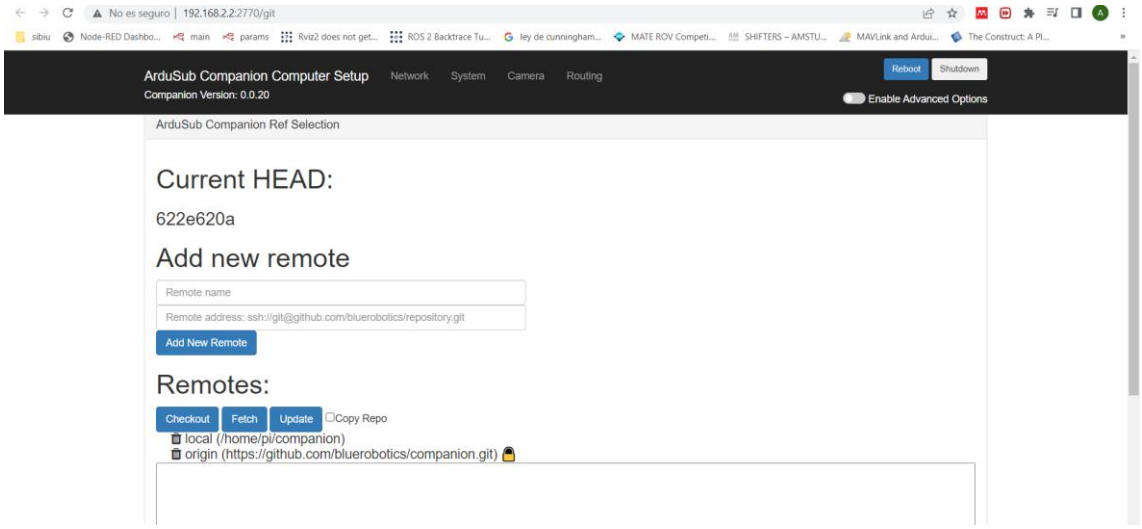
Panel de rutado :2770/routing



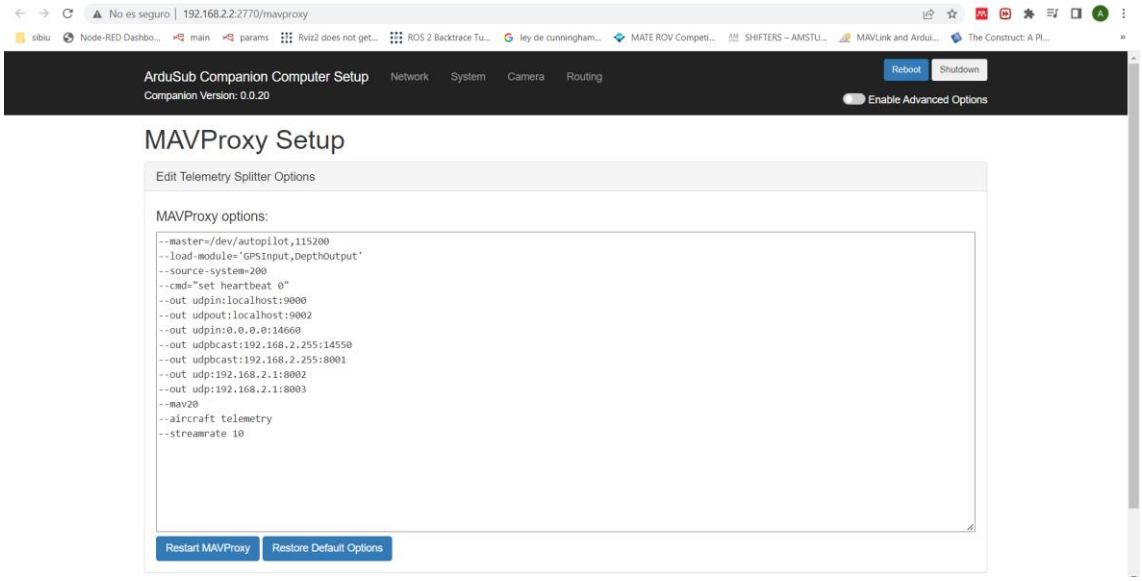
Waterlink setup :2770/waterlinked



Git remotes :2770/git

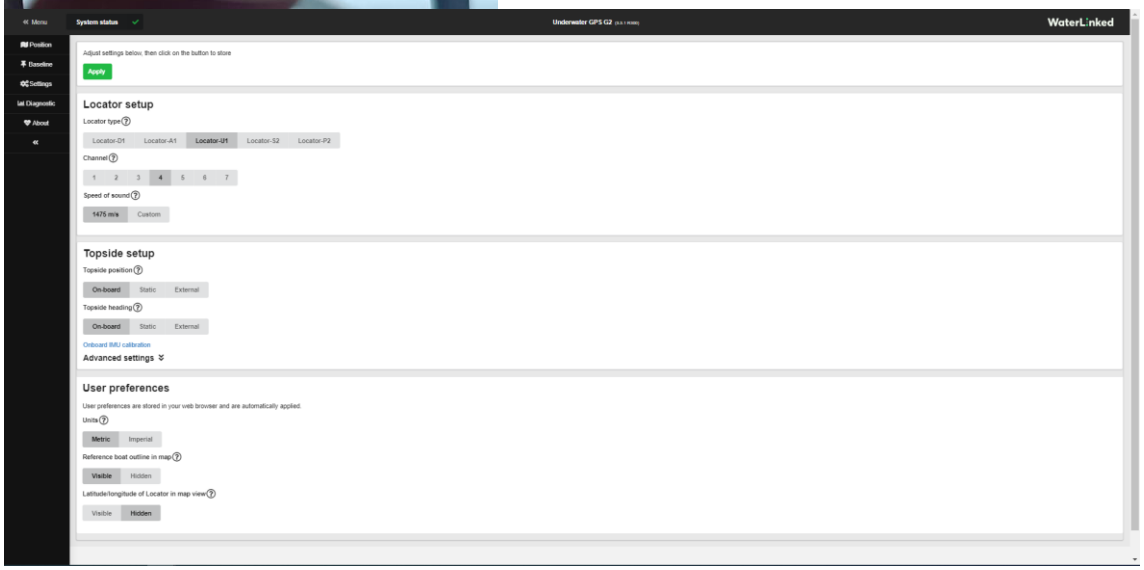


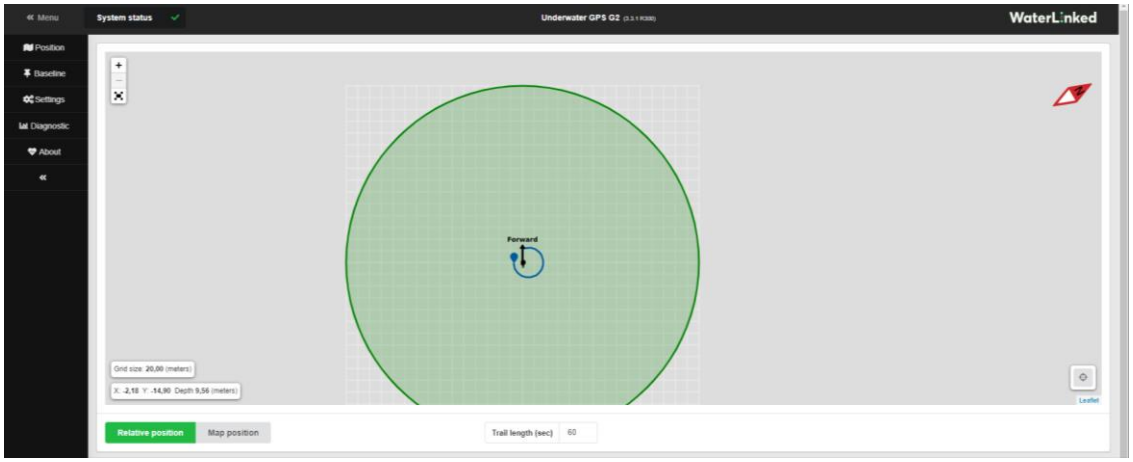
Mavproxy settings :2770/mavproxy

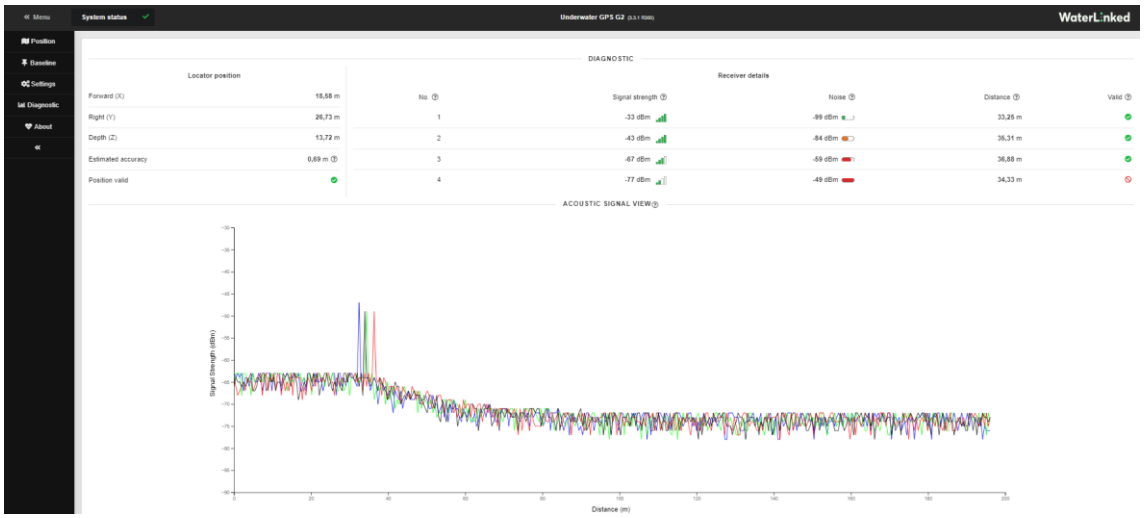


7.3.2 Interfaz Underwater GPS

Todas estas imágenes son de elaboración propia






Menu
System status
Underwater GPS G2
WaterLinked
Position
Baseline
Settings
Diagnostic
About
DIAGNOSTIC
Locator position
Forward (X) 19,58 m
Right (Y) 26,73 m
Depth (Z) 13,72 m
Estimated accuracy 0,69 m
Position valid
Receiver details
No.
Signal strength
Noise
Distance
Valid
1 -33 dBm -99 dBm 33,26 m
2 -43 dBm -84 dBm 35,31 m
3 -67 dBm -59 dBm 36,88 m
4 -77 dBm -49 dBm 34,33 m
ACOUSTIC SIGNAL VIEW
Signal Strength (dBm)
Distance (m)

7.4 Código

Todos estos códigos se pueden encontrar en los github:

https://github.com/AloePacci/ardupilot

https://github.com/AloePacci/Sibiu_Nano_p

7.4.1 Restart

```
#this returns 0 if restart is successfull
def restart():
    command = ["sudo", "systemctl", "restart", "ardusub"]

    return subprocess.run(args=command, stdout=subprocess.DEVNULL,
                          stderr=subprocess.DEVNULL).returncode == 0
```

7.4.2 ROS

7.4.2.1 Translator

```
class translator:
    def __init__(self):
        #nos suscribimos a los sensores
        rospy.init_node('traductor', anonymous=True)

        for ident in range(8):

self.thruster=rospy.Subscriber(ns+'/thrusters/'+str(ident)+'/input',
FloatStamped, self.translate_input, ident)
        self.publisher=rospy.Publisher('/mavros/rc/override',
OverrideRCIn, queue_size=10)
        self.override=OverrideRCIn()
        self.override.channels=[1500, 1500, 1500, 1500, 1500, 1500,
1500, 1500, 1100, 1400, 0, 0, 0, 0, 0, 0] #default values
        print(self.override.channels)



    def translate_input(self, data, ident):
        self.channel[indent]=1500.0+data.data
        self.publisher.publish(self.override)
```

7.4.2.2 Logger

```
class logger:
    def __init__(self):
        #nos suscribimos a los sensores

        rospy.init_node('grabadora', anonymous=True)
        self.altitude_service = rospy.Service('/save_data',
CommandBool, self.save_data)

        self.lista_p_diff=[]
        self.lista_s_diff=[]
```

```
        self.lista_roll=[]
        self.lista_pitch=[]
        self.lista_yaw=[]
        self.lista_armed=[]
        self.lista_battery=[]
        self.lista_p_static=[]
        self.lista_h=[]
        self.lista_state=[]
        self.lista_channels=[]


self.spressure_sub=rospy.Subscriber('/mavros/imu/static_pressure',
FluidPressure, self.update_pressure_static)

self.dpressure_sub=rospy.Subscriber('/mavros/imu/diff_pressure',
FluidPressure, self.update_pressure_diff)
        self.IMU_sub=rospy.Subscriber('/mavros/imu/data', Imu,
self.update_imu)
        self.state_sub=rospy.Subscriber('/mavros/state', State,
self.update_state)
        self.rc_sub=rospy.Subscriber('/mavros/rc/out', RCOut,
self.update_signal)
        self.battery_sub=rospy.Subscriber('/mavros/battery',
BatteryState, self.update_battery)

    def update_state(self, data):
        self.lista_state.append(data)

    def update_pose(self, data):
        self.lista_h[0].append(data.pose.pose.position.x)

    def update_signal(self, data):
        self.lista_channels.append(data)

    def update_battery(self, data):
        self.lista_battery.append(data)

    def update_pressure_static(self, data):
        #self.lista_a.append((data.fluid_pressure-101.325)*10/103.25)
#profundidad respecto al nivel del mar en m
        self.lista_p_static.append((data.fluid_pressure-
101.325)*10/103.25) #profundidad respecto al nivel del mar en m

    def update_pressure_diff(self, data):
        #self.lista_a.append((data.fluid_pressure-101.325)*10/103.25)
#profundidad respecto al nivel del mar en m
        self.lista_p_diff.append((data.fluid_pressure-
101.325)*10/103.25) #profundidad respecto al nivel del mar en m
    def update_imu(self, data):
        orientation = [data.orientation.x, data.orientation.y,
data.orientation.z, data.orientation.w]
        (self.roll,self.pitch,self.yaw) =
tf.transformations.euler_from_quaternion(orientation)
        self.lista_roll.append(self.roll)
        self.lista_pitch.append(self.pitch)
        self.lista_yaw.append(self.yaw)
```

```
def save_data(self, data):
    file_p_diff = open('Save_data_pdif.txt', 'w')
    file_sdiff = open('Save_data_sdiff.txt', 'w')
    file_roll = open('Save_data_roll.txt', 'w')
    file_pitch = open('Save_data_pitch.txt', 'w')
    file_yaw = open('Save_data_yaw.txt', 'w')
    file_arm = open('Save_data_arm.txt', 'w')
    file_batter = open('Save_data_batter.txt', 'w')
    file_p_static = open('Save_data_p_static.txt', 'w')
    file_h = open('Save_data_h.txt', 'w')
    file_state = open('Save_data_state.txt', 'w')
    file_channels = open('Save_data_channels.txt', 'w')

    file_p_diff.write(str(self.lista_p_diff))
    file_sdiff.write(str(self.lista_s_diff))
    file_roll.write(str(self.lista_roll))
    file_pitch.write(str(self.lista_pitch))
    file_yaw.write(str(self.lista_yaw))
    file_arm.write(str(self.lista_armed))
    file_batter.write(str(self.lista_battery))
    file_p_static.write(str(self.lista_p_static))
    file_h.write(str(self.lista_h))
    file_state.write(str(self.lista_state))
    file_channels.write(str(self.lista_channels))
    return True
```

7.4.3 Pymavlink

7.4.3.1 Logger

```
class CustomError(Exception):
    pass

class Logger:
    def __init__(self, vehiculo=None):
        self.__once=True #variable to instanciate the file
        self.__stop=False #variable to stop logging to file
        aux=datetime.today()
        self.filename=aux.strftime("data %m.%d.%Y..%H.%M")
        #instanciate columns so the order is always the same in the
file.

self.data=pd.DataFrame(columns=["pressure","differential_pressure","pr
essure_temperature","roll","pitch","yaw","rollspeed","pitchspeed","yaw
speed","armed","ekf","state","heading","mode","Vcc","nav_roll","nav_pi
tch","nav_bearing","wp_dist","alt_error","airspeed","groundspeed","thr
ottle","alt","climb","servo1","servo2","servo3","servo4","servo5","ser
vo6","servo7","servo8","rc1","rc2","rc3","rc4","rc5","rc6","rc7","rc8"
,"xacc","yacc","zacc","xgyro","ygyro","zgyro","xmag","ymag","zmag","la
t","lon","altgps"])



        if vehiculo is None:
            self.vehicle=None
            #print("no vehicle found")
        else:
            self.append_vehicle(vehiculo)


    def append_vehicle(self, vehicle, outside=False):
        self.log("vehicle found, starting logging")
        self.vehicle=vehicle.vehicle
        self.gps_handler=vehicle.gps

        if outside:#listener for messages

self.external_save_thread=threading.Thread(target=self.external_save)
            self.external_save_thread.start()


        else:
            self.vehicle.add_message_listener('SCALED_PRESSURE2',
self.pressure_read)
            self.vehicle.add_message_listener('ATTITUDE',
self.attitude_read)
            self.vehicle.add_message_listener('SYS_STATUS',
self.sys_read)
            self.vehicle.add_message_listener('POWER_STATUS',
self.power_read)
            self.vehicle.add_message_listener('NAV_CONTROLLER_OUTPUT',
self.nav_read)
```

```
            self.vehicle.add_message_listener('VFR_HUD',
self.vfr_read)
            self.vehicle.add_message_listener('SERVO_OUTPUT_RAW',
self.servo_read)
            #self.vehicle.add_message_listener('RC_CHANNELS_RAW',
self.rc_read)
            #self.vehicle.add_message_listener('SCALED_IMU2',
self.imu_read)
            self.vehicle.add_message_listener('GLOBAL_POSITION_INT',
self.pos_read)
            #self.vehicle.add_message_listener('EKF_STATUS_REPORT',
self.ekf_read)
            self.vehicle.add_message_listener('BATTERY_STATUS',
self.battery_read)
            #download vehicle params
            self.cmds = self.vehicle.commands
            self.cmds.download()
            self.cmds.wait_ready()
            self.home = self.vehicle.home_location

        #create thread to save data into file
        self.save_thread=threading.Thread(target=self.save_data)
        self.save_thread.start()

    def external_save(self):
        try:
            x=self.gps_handler.x
            y=self.gps_handler.y
        except:
            x=0
            y=0
        last=[self.vehicle.attitude.roll, self.vehicle.attitude.roll,
self.vehicle.heading, self.vehicle.location.global_frame.alt, x]
        while not self.__stop:
            if last!=[self.vehicle.attitude.roll,
self.vehicle.attitude.roll, self.vehicle.heading,
self.vehicle.location.global_frame.alt,x]:
                last=[self.vehicle.attitude.roll,
self.vehicle.attitude.roll, self.vehicle.heading,
self.vehicle.location.global_frame.alt,x]
                a=datetime.now()
                timestamp=a.second+a.minute*60+a.microsecond/1000000
                self.data=pd.concat([self.data,
pd.DataFrame([self.vehicle.attitude.roll, self.vehicle.attitude.roll,
self.vehicle.heading, self.vehicle.location.global_frame.alt, x,
y],columns=[timestamp]
,index=["roll","pitch","heading","alt","lat","lon"]).T], sort=True)
                time.sleep(0.01)


    def save_data(self):
        print("starting log")
        while not self.__stop:
            if len(self.data)>200: #save data once we have few data to
save
                if self.__once:#first data?
```

```
self.data.to_csv("./data/"+self.filename+".csv",mode="w") #create file
at start
                    self.__once=False
                else:

self.data.to_csv("./data/"+self.filename+".csv",mode="a",
header=False) #update file for following data
                #empty dataframe

self.data=pd.DataFrame(index=[self.data.index.max()],columns=["pressur
e","differential_pressure","pressure_temperature","roll","pitch","yaw"
,"rollspeed","pitchspeed","yawspeed","armed","ekf","state","heading","
mode","Vcc","nav_roll","nav_pitch","nav_bearing","wp_dist","alt_error"
,"airspeed","groundspeed","throttle","alt","climb","servo1","servo2","
servo3","servo4","servo5","servo6","servo7","servo8","rc1","rc2","rc3"
,"rc4","rc5","rc6","rc7","rc8","xacc","yacc","zacc","xgyro","ygyro","z
gyro","xmag","ymag","zmag","lat","lon","altgps"])
            time.sleep(1)
        self.data.to_csv("./data/"+self.filename+".csv",mode="a",
header=False) #update file last data
        self.print("log save thread closed")


    def pressure_read(self,vehicle, name, msg):
        self.data=pd.concat([self.data, pd.DataFrame([msg.press_abs,
msg.press_diff, msg.temperature],columns=[msg.time_boot_ms]
,index=["pressure","differential_pressure","pressure_temperature"]).T]
, sort=True)
    def attitude_read(self,vehicle, name, msg):
        self.data=pd.concat([self.data, pd.DataFrame([msg.roll,
msg.pitch, msg.yaw, msg.rollspeed, msg.pitchspeed,
msg.yawspeed],columns=[msg.time_boot_ms]
,index=["roll","pitch","yaw","rollspeed","pitchspeed","yawspeed"]).T],
sort=True)
    def sys_read(self,vehicle, name, msg):
        self.data=pd.concat([self.data,
pd.DataFrame([int(self.vehicle.armed), int(self.vehicle.ekf_ok),
self.vehicle.system_status.state,
self.vehicle.mode.name],columns=[self.data.index.max()]
,index=["armed","ekf","state","mode"]).T], sort=True)
        try:
            if self.last_arm!=self.vehicle.armed:
                self.log("vehicle was armed" if self.vehicle.armed
else "vehicle was disarmed")
            if self.last_mode!=self.vehicle.mode.name:
                self.log(f"mode changed to {self.vehicle.mode.name}")
            self.last_arm=self.vehicle.armed
            self.last_mode=self.vehicle.mode.name
        except:
            self.last_arm=self.vehicle.armed
            self.last_mode=self.vehicle.mode.name
        #self.data=pd.concat([self.data,
pd.DataFrame([],columns=[self.data.index[-1]]  ,index=[]).T],
sort=True)
    def power_read(self,vehicle, name, msg):
```

```
        self.data=pd.concat([self.data,
pd.DataFrame([msg.Vcc],columns=[self.data.index.max()]
,index=["Vcc"]).T], sort=True)
    def nav_read(self,vehicle, name, msg):
        self.data=pd.concat([self.data, pd.DataFrame([msg.nav_roll,
msg.nav_pitch, msg.nav_bearing, msg.wp_dist,
msg.alt_error],columns=[self.data.index.max()]
,index=["nav_roll","nav_pitch","nav_bearing","wp_dist","alt_error"]).T
], sort=True)
    def vfr_read(self,vehicle, name, msg):
        self.data=pd.concat([self.data, pd.DataFrame([msg.airspeed,
msg.groundspeed, msg.heading, msg.throttle, msg.alt,
msg.climb],columns=[self.data.index.max()]
,index=["airspeed","groundspeed","heading","throttle","alt","climb"]).
T], sort=True)
    def servo_read(self,vehicle, name, msg):
        self.data=pd.concat([self.data, pd.DataFrame([msg.servo1_raw,
msg.servo2_raw, msg.servo3_raw, msg.servo4_raw, msg.servo5_raw,
msg.servo6_raw, msg.servo7_raw,
msg.servo8_raw],columns=[msg.time_usec//1000]
,index=["servo1","servo2","servo3","servo4","servo5","servo6","servo7"
,"servo8"]).T], sort=True)
    def rc_read(self,vehicle, name, msg):
        self.data=pd.concat([self.data, pd.DataFrame([msg.chan1_raw ,
msg.chan2_raw , msg.chan3_raw ,msg.chan4_raw , msg.chan5_raw ,
msg.chan6_raw ,msg.chan7_raw ,
msg.chan8_raw],columns=[msg.time_boot_ms]
,index=["rc1","rc2","rc3","rc4","rc5","rc6","rc7","rc8"]).T],
sort=True)
    def imu_read(self,vehicle, name, msg):
        self.data=pd.concat([self.data, pd.DataFrame([msg.xacc,
msg.yacc, msg.zacc, msg.xgyro, msg.ygyro, msg.zgyro, msg.xmag,
msg.ymag, msg.zmag],columns=[msg.time_boot_ms]
,index=["xacc","yacc","zacc","xgyro","ygyro","zgyro","xmag","ymag","zm
ag"]).T], sort=True)
    def pos_read(self,vehicle, name, msg):
        self.data=pd.concat([self.data, pd.DataFrame([msg.lat,
msg.lon, msg.alt],columns=[msg.time_boot_ms]
,index=["lat","lon","altgps"]).T], sort=True)
    def ekf_read(self,vehicle, name, msg):
        pass
        #self.data=pd.concat([self.data, pd.DataFrame([msg.press_abs,
msg.press_diff, msg.temperature],columns=[msg.time_boot_ms]
,index=["pressure","differential_pressure","pressure_temperature"]).T]
, sort=True)
    def battery_read(self,vehicle, name, msg):
        self.data=pd.concat([self.data, pd.DataFrame([msg.temperature,
msg.voltages[0], msg.current_consumed,
msg.battery_remaining],columns=[self.data.index[-1]]
,index=["bat_temp","bat_voltage","current","battery %"]).T],
sort=True)


    def stop_logging(self):
        self.__stop=True
```

```
    def print(self, message):
        print(message)
        stack = inspect.stack()
        if len(stack)>2:
            log=f"{self.data.index.max()}:"
            for i in stack[1:len(stack)]:
                aux=i[1].split('\\')[-1]
                log+=f"{aux} {i[2]}, {i[3]} > "
            log=log[0:-2]
            log+=F": {message}"
        else:
            log=f"{self.data.index.max()}:"
            info=stack[1]
            file, line, func = info[1:4]
            aux=file.split('\\')[-1]
            log+=f"{aux} {line}, {func} : {message}"
        with open("./log/"+self.filename+".txt","a") as f:
            f.write(f"{log}\n")


    def log(self, message):
        stack = inspect.stack()
        #here = stack[1]
        if len(stack)>2:
            log=f"{self.data.index.max()}:"
            for i in stack[1:len(stack)]:
                aux=i[1].split('\\')[-1]
                log+=f"{aux} {i[2]}, {i[3]} > "
            log=log[0:-2]
            log+=F": {message}"
        else:
            log=f"{self.data.index.max()}:"
            info=stack[1]
            file, line, func = info[1:4]
            aux=file.split('\\')[-1]
            log+=f"{aux} {line}, {func} : {message}"
        with open("./log/"+self.filename+".txt","a") as f:
            f.write(f"{log}\n")

    def wait_till_init(self):
        while len(self.data)<1:
            time.sleep(0.1)


    def error(self, message):
        self.stop_logging()
        error = traceback.format_exc()
        with open("./log/"+self.filename+"error.txt","a") as f:
            f.write(f"{message}\n")
        raise CustomError(f"{message}")
```

7.4.3.2 Main

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

```
        self.running=True
        self.log=Logger() #instanciate logger
        self.submarino=Sibiu(log=self.log) #instanciate submarine

self.rc_interrupt=threading.Thread(target=self.check_joy_interrupt)
#instanciate interruptions
        self.__close=False

    def start(self):
        self.rc_interrupt.start() #start threads
        self.submarino.start()

    def check_joy_interrupt(self):
        pygame.init()
        self.j = pygame.joystick.Joystick(0)
        self.j.init()
        while not self.__close:
            events = pygame.event.get()
            for event in events:
                if event.type == pygame.JOYAXISMOTION:
                    if self.j.get_axis(5)>0:
                        self.log.print("asked to close program")
                        self.close()
                    elif self.j.get_axis(4)>0:
                        print("izquierdo")
                        self.submarino.pause()
                if event.type == pygame.JOYBUTTONUP:
                    if self.j.get_button(8):
                        self.submarino.resume()
        self.log.log("joy read stopped")
        self.j.quit()


    def close(self):
        self.__close=True
        self.log.stop_logging()
        self.submarino.close()
        self.submarino.handler.safe_close()
        self.running=False
        self.log.print("close finished")
```

7.4.3.3 Program

```
class Sibiu(threading.Thread):
    def __init__(self, log=None):
        threading.Thread.__init__(self) #init thread
        if log is None:
            self.log=Logger() #instanciate logger and start logging
system statusç
        else:
            self.log=log #restore log

        self.handler=Message_sender(logger=self.log) #instatiate
communication handler
        time.sleep(1)
        self.log.append_vehicle(self.handler, outside=True) #start
logging vehicle data
```

```
        self.log.wait_till_init() #wait for things to init
        self.log.log("Connection Success")

    def pause(self):
        pass

    def resume(self):
        pass

    def test(self):
        self.log.print(f"performing test")
        self.handler.setmode("STABILIZE")
        self.handler.set_sys_id(1)
        self.handler.arm()
        #self.vehicle.play_tune("AAAA")
        for i in range (1000000):
            #self.handler.goto_deep(0)
            #self.handler.orientate(90)
            self.handler.external_goto_iteration(0,2,0.5,90)
            time.sleep(0.05)
        self.handler.disarm()
        self.handler.setmode("MANUAL")
        time.sleep(2)
        self.log.print("test finished")


    def run(self):
        self.test()
        pass

    def get_id(self):
        # returns id of the respective thread
        if hasattr(self, '_thread_id'):
            return self._thread_id
        for id, thread in threading._active.items():
            if thread is self:
                return id


    def raise_exception(self):
        thread_id = self.get_id()
        res = ctypes.pythonapi.PyThreadState_SetAsyncExc(thread_id,
              ctypes.py_object(SystemExit))
        if res > 1:
            ctypes.pythonapi.PyThreadState_SetAsyncExc(thread_id, 0)
            self.log.error('Exception raise failure')
        else:
            self.log.print("sibiu closed")
            self.handler.override([0,0,0,0,0,0])
            self.handler.set_sys_id(255)



    def close(self):
        self.raise_exception()
```

7.4.3.4 vehicle

```
#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.sat):
            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(e
rror)
        if abs(output)<self.sat:
            return int(output)
        if output>self.sat:
            return self.sat
        return -self.sat


#this class downloads gps data from Waterlinked
class GPS_handler():
    def __init__(self, url="http://192.168.7.1", logger=None,
antenna_position=None, depth = None):
        self.url=url
        if logger is None:
            self.log=Logger()
        else:
            self.log=logger
        self.__close__=False

        #create thread to read GPS
        self.gps_thread=threading.Thread(target=self.read_gps_thread)
        self.gps_thread.start()
```

```
    def get_data(self, url):
        try:
            r = requests.get(url)
        except requests.exceptions.RequestException as exc:
            self.log.log("Exception occured {}".format(exc))
            return None

        if r.status_code != requests.codes.ok:
            self.log.log("Got error {}: {}".format(r.status_code,
r.text))
            return None

        return r.json()


    def get_antenna_position(self):
        return self.get_data(f"{self.url}/api/v1/config/antenna")

    def get_acoustic_position(self):
        return
self.get_data(f"{self.url}/api/v1/position/acoustic/filtered")

    def get_global_position(self, acoustic_depth = None):
        return self.get_data(f"{self.url}/api/v1/position/global")

    def close(self):
        self.__close__=True

    def read_gps_thread(self):
        while not self.__close__:
            acoustic_position = self.get_acoustic_position()
            global_position = self.get_global_position()
            try:
                    #if acoustic_position["position_valid"]==True:
                    self.depth = acoustic_position["z"]
                    self.x=acoustic_position['x']
                    self.y=acoustic_position['y']
                    self.lat=global_position['lat']
                    self.lon=global_position['lon']
                    self.hdop=global_position['hdop']
                    #else:
                    #   print(f"invalid_pos {[acoustic_position['x'],
acoustic_position['y']]}")
            except:
                pass
            time.sleep(0.05) #limit refresh rate
        self.log.log("gps log closed")

class Message_sender(threading.Thread):
    """this function manages the connection to the submarine
    @params:
        -ip: ip of the submarine
        -port: port to connect to
        -timeout: timeout for connection success
        -source system: ID for the program in MAVLink
        -source component: Component ID for the program in MAVLink
```

```
    @returns:
        it creates the shared variable self.vehicle
    @notes:
        if connection fails, it will log and notice reason of fail,
killing the program
    """

    def __init__(self, logger=None, ip="192.168.2.1", port="8002",
timeout=6.0, source_system=255, external=False):
        if logger is None:
            self.log=Logger()
        else:
            self.log=logger
        try:
            self.vehicle = connect(ip+":"+port,wait_ready=False,
baud=115200, timeout=timeout, source_system=source_system)
            self.vehicle_order =
connect(ip+":"+"8003",wait_ready=False, baud=115200, timeout=timeout,
source_system=source_system)

        except ConnectionRefusedError:
            self.log.error("Connection to submarine was refused")
        except OSError:
            self.log.error("Sibiu was not found in the same network")
        except TimeoutError:
            self.log.error("Submarine connection timeout, port is
busy")
        except:
            self.log.error(f"Connectiom could not be made, unknown
error:\n")

        self.log.print(f"Connection SUCCESS")
        self.gps=GPS_handler(logger=self.log)
        self.sensor=Sensor()
        self.__stop__=False
        #create thread to send GPS data

        self.gps_thread=threading.Thread(target=self.send_GPS_Mavlink)
        self.gps_thread.start()
        self.override([0,0,0,0,0,0])
        if external:

self.control_thread=threading.Thread(target=self.send_override_Mavlink
)
            self.control_thread.start()

        #instanciate PID
        self.x_axis=pid(100, 0.01, 5, 500)
        self.y_axis=pid(100, 0.01, 5, 500)
        self.z_axis=pid(150, 1, 60, 500)
        self.roll_axis=pid(100, 0, 400, 500)
        self.pitch_axis=pid(100, 0, 400, 500)
        self.yaw_axis=pid(0.8, 0.02 , 0.1, 200)

        self.last=[]
```

```
    def goto_deep(self, deepness):
        deep=-int(abs(deepness))
        #self.log.log(f"asked to go to deep {-deep}")
        msg=mavlink2.MAVLink_set_position_target_global_int_message(
            0, 0, 0,  # time (not used), target system, target
component
            0, #MAV_FRAME_GLOBAL
            0b110111111000,  #
yaw_rate,yaw,unused,Az,Ay,Ax,Vz,Vy,Vx,Z,Y,X
            0,  # lat e7
            0,  # lon e7
            int(deep),
            0,0,0, #vx,vy,vz
            0,0,0, #ax,ay,az
            0,  # yaw
            0)  # yaw rate
        self.vehicle_order.send_mavlink(msg)

    def orientate(self, desired_yaw):
        #self.log.log("asked to rotate {desired_yaw}")
        msg = mavlink2.MAVLink_set_attitude_target_message(
            0, 0, 0,  # time (not used), target system, target
component
            0b00000111,  # attitude, throttle,
uns,uns,uns,yawrate,pitchrate,rollrate
            QuaternionBase([math.radians(angle) for angle in
(0,0,desired_yaw)]), #quaternion
            0,0,0,0)  # roll rate, pitch rate, yaw rate, thrust
        # send command to vehicle
        self.vehicle_order.send_mavlink(msg)

    def conditional_yaw(self, heading, relative=False):
        if relative:
            is_relative = 1  # yaw relative to direction of travel
        else:
            is_relative = 0  # yaw is an absolute angle
        # create the CONDITION_YAW command using command_long_encode()
        msg = self.vehicle_order.message_factory.command_long_encode(
            0, 0,  # target system, target component
            pymavlink.mavutil.mavlink.MAV_CMD_CONDITION_YAW,  #
command
            0,  # confirmation
            heading,  # param 1, yaw in degrees
            0,  # param 2, yaw speed deg/s
            1,  # param 3, direction -1 ccw, 1 cw
            is_relative,  # param 4, relative offset 1, absolute angle
0
            0, 0, 0)  # param 5 ~ 7 not used
        # send command to vehicle
        self.vehicle_order.send_mavlink(msg)

    def override(self, values):
        self.values=values

    def send_override_Mavlink(self):
        while not self.__stop__:
```

```
            self.vehicle_order.channels.overrides =
{'5':1500+self.values[0], '6':1500+self.values[1],
'3':1500+self.values[2], '2':1500+self.values[3],
'1':1500+self.values[4], '4':1500+self.values[5]}
            time.sleep(0.1)


    def play_tune(self, tune):
        self.log.log("asked to play tune {tune}")
        msg=mavlink2.MAVLink_play_tune_message(
        0, 0,  #target system, target component
        bytes('', 'utf-8'), #format
        bytes('', 'utf-8') #tune
        )
        self.vehicle_order.send_mavlink(msg)

    def arm(self):
        self.log.log("vehicle arm")
        self.vehicle_order.arm()

    def disarm(self):
        self.log.log("vehicle disarm")
        self.vehicle_order.disarm()

    def setmode(self, mode):
        self.log.log(f"mode changed to {mode}")
        self.vehicle_order.mode = VehicleMode(mode)

    def get_vehicle(self):
        return self.vehicle

    def safe_close(self):
        self.vehicle_order.mode = VehicleMode("MANUAL")
        self.vehicle_order.arm()
        self.close()

    def emergency_close(self):
        self.vehicle_order.disarm()
        self.vehicle_order.mode = VehicleMode("MANUAL")
        self.close()

    def close(self):
        self.gps.close()
        self.log.log("comm closed")
        self.vehicle.close()
        self.vehicle_order.close()
        self.__stop__=True

    def send_GPS_Mavlink(self):
        while not self.__stop__: #while we are not told to stop
            try:
                if self.gps.lon!=None:#if we dont have measure, skip
                    msg=mavlink2.MAVLink_gps_input_message(
                    0, 0,  #time_usec, gps_id
                    0b11111100, #flag ingnore[vertical accuracy, hori
acc, sped acc, vel vert, vel horiz, vdop, hdop, depth]
                    0,0, #time_week_ms, time_week,
```

```
                    5, #0-1: no fix, 2: 2D fix, 3: 3D fix. 4: 3D with
DGPS. 5: 3D with RTK
                    int(self.gps.lat*1e7),
                    int(self.gps.lon*1e7),
                    self.gps.depth,
                    self.gps.hdop,
                    65535, #uin16 max for Vdop
                    0,0,0, #vn ve vd
                    0,0,0, #speed, hori and vert accuracy
                    9, #satellites_visible, min value 4. actual value
3 from antenna
                    0, #yaw not available
                    )
                    self.vehicle_order.send_mavlink(msg)
            except:
                #print("gps problem")
                pass
            time.sleep(0.1) #5hz is enough, we give more cause yolo

    def calculate_distance(self, lat, lon):
        """
        This method is an approximation, and will not be accurate over
large distances and close to the
        earth's poles. It comes from the ArduPilot test code:

https://github.com/diydrones/ardupilot/blob/master/Tools/autotest/comm
on.py
        """

        # Convert to radians #
        lat1 =
np.radians(self.vehicle.location.global_relative_frame.lat)
        lat2 = np.radians(lat)
        lon1 =
np.radians(self.vehicle.location.global_relative_frame.lon)
        lon2 = np.radians(lon)

        # Obtains the latitude/longitude differences #
        d_lat = lat2 - lat1
        d_lon = lon2 - lon1

        # Returns True if the waypoint is within 1.5 meters the ASV
position
        a = np.sin(0.5 * d_lat) ** 2 + np.sin(0.5 * d_lon) ** 2 *
np.cos(lat1) * np.cos(lat2)
        c = 2.0 * np.arctan2(np.sqrt(a), np.sqrt(1.0 - a))
        return 6378100.0 * c

    def goto(self, lat, lon, depth):
        ekf_failed=False #memory for EKF
        ekf_counter=0
        travel_counter=0
        #if we are too far away cancel
        if self.calculate_distance(lat,lon)>100:
            self.log.print("we are too far away from the point")
            return
```

```
        self.log.print(f"asked to go to {[lat, lon]}, distance
{self.calculate_distance(lat,lon)}")
        #put vehicle in guided mode and ask to go to point
        self.vehicle_order.mode = VehicleMode("GUIDED")
        time.sleep(1) #wait for mode to change
        self.vehicle_order.simple_goto(LocationGlobal(lat, lon, -
abs(depth)))
        while self.calculate_distance(lat,lon)>1.5 and (depth-
self.vehicle.location.global_relative_frame.alt)>1.5: #while we have
not reached the point
            #check system is able to go to point
            if not self.vehicle.ekf_ok:
                self.log.print("ekf failed")
                self.vehicle_order.mode = VehicleMode("ALT_HOLD")
#maintain position
                ekf_failed=True
                if ekf_counter%30 == 0: #log each 3 secs
                    self.log.log(f"System Status: \nmode
{self.vehicle.mode.name}, GPS_status: {self.vehicle.gps_0}, System
status: {self.vehicle.system_status.state}, System able to arm
{self.vehicle.is_armable} ")
                ekf_counter+=1

            elif ekf_failed: #if system failed, recover
                ekf_failed=False
                self.log.print("EKF failsafe cleared, resuming
mission")
                self.log.log(f"System Status: mode
{self.vehicle.mode.name}, GPS_status: {self.vehicle.gps_0}, System
status: {self.vehicle.system_status.state}, System able to arm
{self.vehicle.is_armable} ")
                ekf_failed=False #reset flag
                ekf_counter=0
                self.vehicle_order.arm() #arm to avoid inconsistent
state
                self.vehicle_order.mode = VehicleMode("GUIDED")
#restore mode
                self.vehicle_order.simple_goto(LocationGlobal(lat,
lon, -abs(depth))) #restore point
            if travelling_counter%30 == 0: #each 3 seconds
                self.log.print(f"distance
[{self.calculate_distance(lat,lon)},{(depth-
self.vehicle.location.global_relative_frame.alt)}]")

            travelling_counter+=1
            time.sleep(0.1)
        # after reaching samplepoint
        self.vehicle_order.mode = VehicleMode("ALT_HOLD")
        self.sensor.take_sample()

    def external_goto_iteration(self, x,y,z,heading):
        #check if we have gps for XY controller
        try:
            errorx=x-self.gps.x
            errory=y-self.gps.y
        except:
            errorx=0
```

```
            errory=0

        errorz=z-self.vehicle.location.global_frame.alt
        #only update if we have new values
        if
self.last==[errorx,errory,errorz,self.vehicle.attitude.roll,self.vehic
le.attitude.pitch,self.vehicle.heading]:
            return
        else:

self.last=[errorx,errory,errorz,self.vehicle.attitude.roll,self.vehicl
e.attitude.pitch,self.vehicle.heading]

        #for heading
        #force simetry
        vehicleheading=self.vehicle.heading
        if vehicleheading>180:
            vehicleheading=360-vehicleheading
        if abs(heading-vehicleheading) > 180: #we choose the shortest
path
                    if heading>0:
                        heading-=360
                    else:
                        heading+=360

        rolsignal=self.roll_axis.PID(-self.vehicle.attitude.roll)
        pitchsignal=self.pitch_axis.PID(self.vehicle.attitude.pitch)
        yawsignal=self.yaw_axis.PID(heading-vehicleheading)

        angulo_ataque=self.vehicle.heading
        distance=sqrt(pow(errorx,2)+pow(errory,2))
        xvalue=int(self.x_axis.PID(distance)*math.sin(angulo_ataque))
        yvalue=int(self.y_axis.PID(distance)*math.cos(angulo_ataque))
        zvalue=self.z_axis.PID(errorz)

        self.override([xvalue, yvalue ,zvalue, rolsignal, pitchsignal,
yawsignal])

    def set_sys_id(self, id):
        self.vehicle_order.parameters['SYSID_MYGCS']=id
```

7.4.3.5 plotter

```
data=pd.read_csv("./data/data 11.25.2022..16.35.csv",index_col=0)
data["mode"]=data["mode"].replace('MANUAL',1)
data["mode"]=data["mode"].replace('ALT_HOLD',2)
data["mode"]=data["mode"].replace('GUIDED',3)
data["state"]=data["state"].replace('CRITICAL',1)
data=data.groupby(data.index)
data=data.mean()

fig=plt.figure()
ax = fig.add_subplot(1, 1, 1)
datosx=data[~data["lat"].isnull()]["lat"]
datosy=data[~data["lon"].isnull()]["lon"]
```

```
ax.plot(datosx.values, datosy.values, color='tab:blue')
plt.xlabel("lat (°)")
plt.ylabel("lon (°)")
plt.title("position")
plt.savefig("./images/"+"position"+'.png', dpi=400)
for i in data:
    fig=plt.figure()
    ax = fig.add_subplot(1, 1, 1)
    print(i)
    if i=="mode":
        datos=data[~data[i].isnull()][i]
        ax.plot(datos.index/1000, datos.values, color='tab:blue')
        ax.yaxis.set_ticks([1,2,3])
        ax.set_yticklabels(["MANUAL", "DEEP_HOLD", "GUIDED"])
    elif i=="state":
        datos=data[~data[i].isnull()][i]
        ax.plot(datos.index/1000, datos.values, color='tab:blue')
        ax.yaxis.set_ticks([1,2])
        ax.set_yticklabels(["CRITICAL", "Undefined"])
    else:
        datos=data[~data[i].isnull()][i]
        ax.plot(datos.index, datos.values, color='tab:blue')
    plt.plot()
    plt.xlabel('tiempo (s)')
    if i=="lat":
        plt.ylabel("x (m)")
        plt.title("x")
    elif i=="lon":
        plt.ylabel("y (m)")
        plt.title("y")
    else:
        plt.ylabel(i)
        plt.title(i)
    plt.savefig("./images/"+i+'.png', dpi=400)
```

7.4.4 Ardupilot

7.4.4.1 No_GPS

```
#include "Sub.h"

/*
 * this code wants to execute a control without gps
 */

// init control no_gps controller
bool Sub::no_gps_init()
{
    //safe mechanism, have at least some sensor data, we need a
barometer
    if(!control_check_barometer()) {
        return false;
    }

    // initialise waypoint controller,
    // this will init vertical maximun speeds and accels
    wp_nav.wp_and_spline_init();


    // initiate an empty destination objective
    Vector3f stopping_point;
    wp_nav.get_wp_stopping_point(stopping_point);

    // no need to check return status because terrain data is not used
    wp_nav.set_wp_destination(stopping_point, false);

    // initialise yaw
    set_auto_yaw_mode(get_default_auto_yaw_mode(false));

    return true;
}

// no_gps_run - runs the controllers
// should be called at 100hz or more
void Sub::no_gps_run()
{
    // initialize vertical speeds and acceleration
    pos_control.set_max_speed_accel_z(-get_pilot_speed_dn(),
g.pilot_speed_up, g.pilot_accel_z);

    // if not armed set throttle to zero and exit immediately
    if (!motors.armed()) {

motors.set_desired_spool_state(AP_Motors::DesiredSpoolState::GROUND_ID
LE);
        // Sub vehicles do not stabilize roll/pitch/yaw when not auto-
armed (i.e. on the ground, pilot has never raised throttle)
        attitude_control.set_throttle_out(0,true,g.throttle_filt);
        attitude_control.relax_attitude_controllers();
        wp_nav.wp_and_spline_init();
        //pos_control.relax_z_controller(motors.get_throttle_hover());
        return;
```

```
    }

    // set motors to full range

motors.set_desired_spool_state(AP_Motors::DesiredSpoolState::THROTTLE_
UNLIMITED);


    // Send pilot input to forward/lateral outputs
    motors.set_lateral(channel_lateral->norm_input());
    motors.set_forward(channel_forward->norm_input());

    // WP_Nav has set the vertical position control targets
    // run the vertical position controller and set output throttle
    pos_control.update_z_controller();

    attitude_control.input_euler_angle_roll_pitch_yaw(channel_roll-
>get_control_in(), channel_pitch->get_control_in(),
get_auto_heading(), true);
}
```

7.4.4.2 Bypass

```
#include "Sub.h"

// raw_init - initialise manual controller
bool Sub::raw_init()
{
    // set target altitude to zero for reporting
    pos_control.set_pos_target_z_cm(0);

    // attitude hold inputs become thrust inputs in manual mode
    // set to neutral to prevent chaotic behavior (esp. roll/pitch)
    set_neutral_controls();

    return true;
}

// raw_run - runs the manual (bypass) controller
// should be called at 100hz or more
void Sub::raw_run()
{
    // if not armed set throttle to zero and exit immediately
    if (!motors.armed()) {

motors.set_desired_spool_state(AP_Motors::DesiredSpoolState::GROUND_ID
LE);
        attitude_control.set_throttle_out(0,true,g.throttle_filt);
        attitude_control.relax_attitude_controllers();
        return;
    }


motors.set_desired_spool_state(AP_Motors::DesiredSpoolState::THROTTLE_
UNLIMITED);
```

```
    for(int motor_number=0;motor_number<8;motor_number++){
        motors.rc_write(motor_number,
RC_Channels::rc_channel(motor_number)->get_radio_in());
    }
}
```

8 BIBLIOGRAFÍA

[1] A. Casado Pérez Tutor and C. Bordons Alba, "Modelado, simulación y control de un vehículo submarino ligero".

[2] "Ciencias para el mundo contemporáneo." http://www3.gobiernodecanarias.org/aciisi/cienciasmc/web/u7/intro_u7.html#indice (accessed Nov. 24, 2022).

[3] "Porcentaje de agua en seres vivos - Aqua azul." https://sites.google.com/site/todosobreagua/porcentaje-de-agua-en-seres-vivos (accessed Nov. 17, 2022).

[4] "Distribución de Agua en el Planeta | Jumapam." http://jumapam.gob.mx/cultura-del-agua/distribucion-de-agua-en-el-planeta/ (accessed Nov. 17, 2022).

[5] Página, "El agua en Canarias".

[6] "Cerro Prieto: La NASA retrata desde el espacio la sequía extrema en México | EL PAÍS México." https://elpais.com/mexico/2022-07-23/la-nasa-retrata-desde-el-espacio-la-sequia-extrema-en-mexico.html (accessed Sep. 02, 2022).

[7] "Del verde al rojo: las imágenes por satélite certifican de manera drástica la alarmante sequía que ya sufre Doñana." https://www.eldiario.es/andalucia/sostenibilidad/verde-rojo-imagenes-satelite-certifican-manera-drastica-alarmante-sequia-sufre-donana_1_8816741.html (accessed Sep. 06, 2022).

[8] 2017- (Azoulay, A.) authorCorporate:UNESCO. Director-General, "Mensaje de la Sra. Audrey Azoulay, Directora General de la UNESCO, con motivo del Día Mundial del Agua, 22 de Marzo de 2021," 2021, Accessed: Sep. 02, 2022. [Online]. Available: https://unesdoc.unesco.org/ark:/48223/pf0000375948_spa

[9] "El estado de los embalses en España." https://www.epdata.es/datos/embalses-boletin-hidrografico/54?accion=1 (accessed Sep. 02, 2022).

[10] "Ministerio de Derechos Sociales y Agenda 2030 - Agenda 2030." https://www.mdsocialesa2030.gob.es/agenda2030/index.htm (accessed Nov. 24, 2022).

[11] "Vista de Diseño del software de control de un UUV para monitorización oceanográfica usando un modelo de componentes y framework con despliegue flexible." https://polipapers.upv.es/index.php/RIAI/article/view/9366/9362 (accessed Sep. 02, 2022).

[12] "Autonomous underwater vehicles and overview of the laws governing it - iPleaders." https://blog.ipleaders.in/autonomous-underwater-vehicles-overview-laws-governing/ (accessed Nov. 20, 2022).

[13] "Water Quality Monitoring with Vehicles for Better Coverage." https://www.ysi.com/ysi-blog/water-blogged-blog/2020/03/water-quality-monitoring-with-vehicles-for-better-coverage (accessed Sep. 05, 2022).

[14] "Los drones submarinos de Dimitri Rebikoff: orígenes del mundo ROV - SubaQuatica Magazine." https://www.subaquaticamagazine.es/los-drones-submarinos-de-dimitri-rebikoff-origenes-del-mundo-rov/ (accessed Sep. 06, 2022).

[15] Jorge Luis Lemus Ramos, "Sistema de navegación inercial para un AUV en presencia de corrientes marinas," 2018. http://scielo.sld.cu/scielo.php?script=sci_arttext&pid=S1815-59282018000200004 (accessed Aug. 24, 2021).

[16] C. Lodovisi, P. Loreti, L. Bracciale, and S. Betti, "Performance analysis of hybrid optical-acoustic AUV swarms for marine monitoring," *Future Internet*, vol. 10, no. 7, Jul. 2018, doi:

10.3390/FI10070065.

[17] "Monitoring water quality and ecology with underwater drones," 2017.

[18] "About us - M.D.B Marine." http://www.mdbmarine.com/about-us/ (accessed Sep. 08, 2022).

[19] "Mini-ROV Observer comprar buceo - Aditech".

[20] "Nido Robotics." https://www.nidorobotics.com/ (accessed Sep. 07, 2021).

[21] "BlueRobotics." https://bluerobotics.com/store/rov/bluerov2/ (accessed Sep. 07, 2021).

[22] T. Ahmad and J. Iqbal, "Underwater robotic vehicles: Latest development trends and potential challenges," *Science International*, vol. 26, pp. 1111–1117, 2014, [Online]. Available: https://www.researchgate.net/publication/280642801_Underwater_robotic_vehicles_Latest_development_trends_and_potential_challenges

[23] "Connecting What's Needed with What's Next ™," 2017.

[24] "Descripción de los AUV." http://www.utm.csic.es/es/servicios/auv/vehiculos/allpages (accessed Sep. 08, 2022).

[25] "AUV - MEPUS-ARV-150 - ZhiZheng Ocean Technology Company." https://www.nauticexpo.es/prod/zhizheng-ocean-technology-company/product-200288-581526.html (accessed Sep. 08, 2022).

[26] "AUV - TORPEDO - Subsea Tech." https://www.nauticexpo.es/prod/subsea-tech/product-30441-561783.html (accessed Sep. 08, 2022).

[27] "AUV - URASHIMA - MITSUBISHI HEAVY INDUSTRIES - Ship & Ocean." https://www.nauticexpo.es/prod/mitsubishi-heavy-industries-ship-ocean/product-32135-375346.html (accessed Sep. 08, 2022).

[28] "Dive & Discover - Expedition 17: Technology ," *2021*. https://divediscover.whoi.edu/expedition17/technology/ (accessed Sep. 09, 2021).

[29] O. Schofield, "The Scarlet Knight's Trans-Atlantic Challenge," 2009. https://rucool.marine.rutgers.edu/atlantic/ (accessed Aug. 24, 2021).

[30] C. Schlegel, "Communication Patterns as Key towards Component-Based Robotics:," *https://doi.org/10.5772/5759*, vol. 3, no. 1, pp. 049–054, Mar. 2006, doi: 10.5772/5759.

[31] P. Ridao, J. Yuh, J. Batlle, and K. Sugihara, "On AUV control architecture," *IEEE International Conference on Intelligent Robots and Systems*, vol. 2, pp. 855–860, 2000, doi: 10.1109/IROS.2000.893126.

[32] M. de la Vega. Pérez Gracia, "Radar de subsuelo.Evaluación para aplicaciones en arqueología y en patrimonio histórico-artístico," *TDX (Tesis Doctorals en Xarxa)*, Oct. 2001, Accessed: Nov. 07, 2022. [Online]. Available: http://www.tdx.cat/handle/10803/6216

[33] R. E. Francois and G. R. Garrison, "Sound absorption based on ocean measurements. Part II: Boric acid contribution and equation for total absorption," *Journal of the Acoustical Society of America*, vol. 72, no. 6, pp. 1879–1890, 1982, doi: 10.1121/1.388673.

[34] S. D. Richards and T. G. Leighton, "Sonar performance in turbid and bubbly environments," *J Acoust Soc Am*, vol. 108, no. 5, p. 2562, Nov. 2000, doi: 10.1121/1.4743513.

[35] "Home - Emasesa." https://www.emasesa.com/ (accessed Nov. 20, 2022).

[36] "Comunicaciones Submarinas - Gradiant." https://www.gradiant.org/noticia/comunicaciones-submarinas-2/ (accessed Nov. 18, 2022).

[37] J. Palfy, "Guide for Vessel Maneuverability 2017," 2002.

[38] "Nortek | A Complete Guide to Underwater Navigation." https://www.nortekgroup.com/knowledge-center/wiki/new-to-subsea-navigation (accessed

Nov. 18, 2022).

[39] L. Zhang, C. Li, and H. Sun, "Object detection/tracking toward underwater photographs by remotely operated vehicles (ROVs)," *Future Generation Computer Systems*, vol. 126, pp. 163–168, Jan. 2022, doi: 10.1016/J.FUTURE.2021.07.011.

[40] A. Freddi, · S Longhi, and · A Monteriù, "A coordination architecture for UUV fleets," *Intel Serv Robotics*, vol. 5, pp. 133–146, 2012, doi: 10.1007/s11370-012-0108-0.

[41] F. Maurelli, P. Patrón, J. Cartwright, J. Sawas, Y. Petillot, and D. Lane, "Integrated MCM missions using heterogeneous fleets of AUVs," *Program Book - OCEANS 2012 MTS/IEEE Yeosu: The Living Ocean and Coast - Diversity of Resources and Sustainable Activities*, 2012, doi: 10.1109/OCEANS-YEOSU.2012.6263537.

[42] J. Padial, S. Dektor, and S. M. Rock, "Correlation of imaging sonar acoustic shadows and bathymetry for ROV terrain-relative localization," *OCEANS 2014 - TAIPEI*, Nov. 2014, doi: 10.1109/OCEANS-TAIPEI.2014.6964351.

[43] Enrique González Sancho, "Diseño, modelización y simulación mecánica de un robot submarino controlado remotamente," Cartagena, 2018. Accessed: Sep. 01, 2021. [Online]. Available: https://repositorio.upct.es/xmlui/bitstream/handle/10317/7592/tfm-gon-dis.pdf?sequence=1&isAllowed=y

[44] "Underwater GPS G2 | Positioning System | Water Linked | Water Linked AS." https://www.waterlinked.com/underwater-gps/gps-g2 (accessed Nov. 21, 2022).

[45] N. ROBOTICS, "Manual de Operacion Sibiu Nano+." [Online]. Available: https://drive.google.com/file/d/1D_-phPOzPbAuYERQWfV1VZXqXOH_guti/view?usp=sharing

[46] B. Robotics, "Ardusub." https://www.ardusub.com/ (accessed Sep. 10, 2021).

[47] "Understanding Mission Planner for Configuring Drone for Flight - Building drones around Beagle Bone Blue." https://beaglebluevoyager.com/understanding-mission-planner-for-configuring-drone-for-flight/ (accessed Sep. 19, 2022).

[48] "Companion Web Interface · GitBook." https://www.ardusub.com/reference/companion-web-ui.html (accessed Mar. 06, 2022).

[49] "Pixhawk Overview — Copter documentation." https://ardupilot.org/copter/docs/common-pixhawk-overview.html (accessed Mar. 06, 2022).

[50] "Learning the ArduPilot Codebase — Dev documentation." https://ardupilot.org/dev/docs/learning-the-ardupilot-codebase.html (accessed Mar. 06, 2022).

[51] S. Soylu, A. A. Proctor, R. P. Podhorodeski, C. Bradley, and B. J. Buckham, "Precise trajectory control for an inspection class ROV," *Ocean Engineering*, vol. 111, pp. 508–523, Jan. 2016, doi: 10.1016/J.OCEANENG.2015.08.061.

[52] "InertialNav/GenerateNavFilterEquations.m at master · priseborough/InertialNav." https://github.com/priseborough/InertialNav/blob/master/derivations/RotationVectorAttitudeParameterisation/GenerateNavFilterEquations.m (accessed Sep. 20, 2022).

[53] "Welcome to the ArduPilot Development Site — Dev documentation." https://ardupilot.org/dev/index.html (accessed Nov. 21, 2022).

[54] "MAVLink Step by Step - Blog - ArduPilot Discourse." https://discuss.ardupilot.org/t/mavlink-step-by-step/9629 (accessed Sep. 21, 2022).

[55] "Messages (common) · MAVLink Developer Guide." https://mavlink.io/en/messages/common.html (accessed Sep. 28, 2022).

[56] "Robots/BlueROV - ROS Wiki." http://wiki.ros.org/Robots/BlueROV (accessed Sep. 28, 2022).

[57] "Installing ROS on BlueROV companion - Blue Robotics Software / Companion - Blue Robotics Community Forums." https://discuss.bluerobotics.com/t/installing-ros-on-bluerov-companion/10946 (accessed Mar. 06, 2022).

[58] "How to control thrusters independently - Blue Robotics Vehicles / BlueROV2 - Blue Robotics Community Forums." https://discuss.bluerobotics.com/t/how-to-control-thrusters-independently/9870 (accessed Mar. 06, 2022).