ESCUELA SUPERIOR DE
INGENIERÍA Y TECNOLOGÍA

# Trabajo de Fin de Grado

## Diseño de Dron Submarino: Diseño e implementación del sistema de comunicaciones y diseño del sistema de control

**Titulación:** Grado en Ingeniería Electrónica Industrial y Automática

**Estudiante:** Nicolás Adrián Rodríguez Linares

**Tutor:** Leopoldo Acosta Sánchez
**Tutor:** Fernando Luis Rosa González

12 de junio de 2019

Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

**IMPRESO DE AUTORIZACIÓN DEL TRABAJO DE FIN DE GRADO POR EL TUTOR**

**Curso 2018/2019**

---

Los Drs. D. **Leopoldo Acosta Sánchez** y D. **Fernando Luis Rosa González** como tutores del estudiante D. **Nicolás Adrián Rodríguez Linares** en el Trabajo de Fin de Grado titulado

**Diseño de dron submarino: Diseño e implementación del sistema de comunicaciones y diseño del sistema de control**,

dan su autorización, acreditada por la firma electrónica de este documento, para la presentación y defensa de dicho proyecto, a la vez que confirman que el estudiante ha cumplido con los objetivos generales y particulares que lleva consigo la realización del mismo. Quieren hacer mención especial del aprovechamiento del estudiante que ha sobresalido de la media en la realización de este trabajo.

La Laguna, a 12 de junio de 2019

# Agradecimientos

Quisiera agradecer especialmente a mis tutores, Leopoldo y Fernando por todo el conocimiento, apoyo y orientación que me han brindado a lo largo de este proyecto. También a Rafael Arnay, Jonay Toledo y Oswaldo González por toda la ayuda que nos han ofrecido tanto a mí como a mis compañeros David e Iván.

Gracias a todos por sumaros a este proyecto tan ambicioso que empezó a tomar forma hace apenas un año, en lo que parecía una locura inalcanzable, pero que ha acabado siendo una experiencia increíble, en la que hemos aprendido una infinidad de cosas.

Gracias también a mis padres y allegados por aguantarme durante todo este largo año, lleno de momentos estresantes, muchas noches largas, pocas comidas familiares y poco tiempo en casa. Gracias también Raquel por el apoyo y los ánimos que me has dado, sin los cuáles difícilmente podría haber aguantado la carga que ha supuesto este proyecto.

Finalmente, gracias José por todas esas tardes que nos has ayudado para poder probar el submarino y hacer que llegara a buen puerto. Y por supuesto, gracias a David e Iván por todos estos años juntos, y los que nos quedan, haciendo locuras como las de este proyecto.

Gracias a todos.

ESCUELA SUPERIOR DE
INGENIERÍA Y TECNOLOGÍA

# Trabajo de Fin de Grado

## Diseño de Dron Submarino: Diseño e implementación del sistema de comunicaciones y diseño del sistema de control

## TOMO I

# Memoria

**Titulación:** Grado en Ingeniería Electrónica Industrial y Automática

**Estudiante:** Nicolás Adrián Rodríguez Linares

**Tutor:** Leopoldo Acosta Sánchez
**Tutor:** Fernando Luis Rosa González

12 de junio de 2019

# Índice general

# Índice de figuras

# Índice de tablas

# Resumen/Abstract

## Resumen

Esta memoria va a explicar el desarrollo de los sistemas de comunicaciones y de control de un ROUV-Vehículo Submarino Operado Remotamente- diseñado en el contexto de un proyecto denominado 'U-Water Explorer Dron'. Se realizará una introducción al estado del arte y a las herramientas hardware y software empleadas a lo largo del proyecto. Además, se contrastarán las diferentes opciones disponibles para el sistema de comunicaciones y el de control. Este trabajo justificará las decisiones tomadas durante el proceso de diseño, así como se realizará un análisis detallado del comportamiento de cada sistema.

## Abstract

This memory will explain the development of the communication and control system of a ROUV- Remotely Operated Underwater Vehicle- designed in the context of a project called 'U-Water Explorer Dron'. An introduction to the state of art and to the hardware and software tools used thoughout the project will take place. Furthermore, different options for the communication system configuration and control strategies will be discused. This work will justify the decisions made in the design process, as well as a detailed explanation of the perfomance of each system.

# Capítulo 1

# Introducción

## 1.1. Proyecto

El presente TFG se enmarca dentro de un proyecto marco denominado *'U-Water Explorer Dron'* y del que se ve una foto en funcionamiento en la figura 1.1, cuyo objetivo es la creación de un dron submarino funcional capaz de operar en aguas tranquilas de puertos, con una limitación de profundidad de 20 metros.

El proyecto se desarrolla en 3 TFG:

- *Diseño de dron submarino: diseño estructural, control de movimiento y sistema de alimentación.* **David Henry González**, estudiante de Ingeniería Electrónica Industrial y Automática en la Universidad de La Laguna.
- *Diseño de dron Submarino: desarrollo de sistema de sensores y software para gestionar los sensores y controlar un dron submarino basado en ROS.* **Iván Jesús Torres Rodríguez**, estudiante de Ingeniería Electrónica Industrial y Automática en la Universidad de La Laguna.
- *Diseño de dron submarino: diseño e implementación del sistema de comunicaciones y diseño de sistema de control.* **Nicolás Adrián Rodríguez Linares**, estudiante de Ingeniería Electrónica Industrial y Automática en la Universidad de La Laguna.

El propósito del submarino es el de ofrecer una alternativa de mercado asequible a las empresas del sector de mantenimiento marítimo, buscando facilitar las labores de inspección visual de cascos de barcos y conductos, generalmente realizadas por equipos de buzos. Es por ello que las especificaciones del dron se han ajustado al sector industrial objetivo, siendo 16 metros el calado máximo de los mayores puertos del mundo se ha fijado la especificación de profundidad máxima a 20 metros.

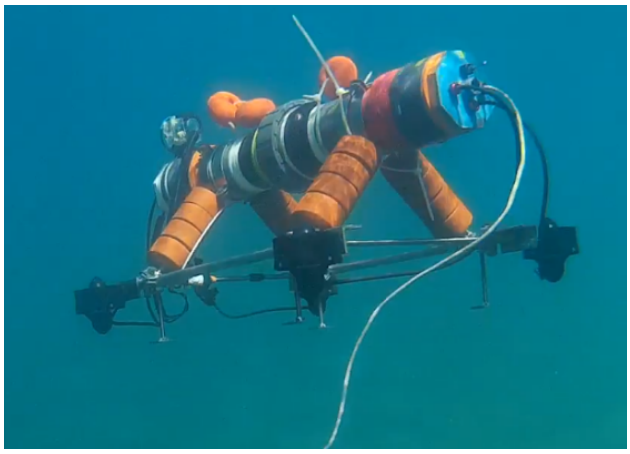


Figura 1.1: Dron submarino diseñado en el proyecto durante una inmersión

Por otro lado, se potencia la reducción de costes, rendimiento en tiempo real y calidad del sistema de visión, y se innova respecto a las configuraciones de motores más empleadas en micro ROV's, empleando 4 propulsores sobre el mismo plano, imitando al esquema de control de los quadrotors.

Además, el submarino porta sensores básicos que le permiten conocer el estado del sistema y del medio que le rodea, pudiendo realizar grabaciones de audio y vídeo, además de registros de diversos sensores, incluyendo la temperatura del agua.

Con la finalidad de apoyar la labor del operador, se incluye un sistema de control que facilite el manejo del dron, así como una interfaz de control y un sistema de visión y reconocimiento de objetos que maximice las posibilidades del submarino.

Por último, el dron se ha concebido para ser empleado en entornos donde cabe la posibilidad que no haya acceso a generadores o a la red eléctrica, por lo que la alimentación se realiza empleando unas baterías. Ello lleva aparejado un análisis exhaustivo del consumo de los distintos elementos para su optimización.

## 1.2. Objetivos del TFG

En el presente TFG se va a desarrollar el sistema de comunicaciones que se emplea en el dron submarino y el sistema de control a alto nivel del dron, lo que engloba el diseño e implementación de un controlador adecuado para el dron y el análisis de la movilidad del dron.

En detalle, los siguientes puntos son objetivos de diseño del presente trabajo:

1. Diseño de un sistema de comunicaciones que garantice una respuesta fluida del dron a las órdenes del operador

2. Sistema de comunicaciones capaz de transmitir imagen con un retraso aceptable

3. Diseño de una estrategia de control adecuada

4. Análisis de la estabilidad del dron

5. Análisis de la configuración de motores y discusión de su funcionalidad y viabilidad

6. Diseño del sistema de operación manual

Además de los contenidos específicos, también resultan del ámbito del trabajo los siguientes puntos transversales del proyecto marco:

– Sincronización del avance de las partes del proyecto

– Optimización de los costes del prototipo para hacerlo más accesible

– Montaje del dron y realización de diversas pruebas que verifiquen que los objetivos de diseño se cumplen

## 1.3. Análisis del estado del arte

Los ROV's -*Remoted Operated Vehicles*- son un tipo de vehículo submarino no tripulado y no autónomo que se opera de forma remota desde superficie. Este tipo de vehículos se suelen emplear en labores de inspección y de mantenimiento, tal y como se recoge en el artículo 'Inspection-Class Remotely Vehicles' de Romano Capocci et al [1]. A continuación, se puede ver una imagen de un ROV creado por la empresa BlueRobotics 1.2, con una configuración de 8 motores pensada para emplear el dron en trabajos ligeros, además de en labores de inspección.

Figura 1.2: Dron submarino desarrollado por la empresa BlueRobotics

Dentro de la categoría de ROV's de inspección se puede diferenciar dos subcategorías en función del tamaño y alcance de las prestaciones, en lo que a profundidad máxima y capacidad de carga se refiere. La categoría de micro ROV's engloba a aquellos vehículos que pueden ser cargados y manipulados por una persona sin la necesidad de emplear ningún tipo de cabestrante.

Dicho tipo de drones suelen pesar entre 3 y 20 kg en superficie, y a pesar de disponer de una buena relación entre peso y propulsión, apenas son capaces de elevar cargas. Es por este motivo que se suelen emplear en labores de inspección visual, por lo que se limita enormemente la capacidad de trabajo que tienen. Esto además va aparejado a que por las limitaciones de peso, no se pueden emplear paredes gruesas que aumenten la profundidad máxima en la que pueden operar, siendo por tanto una clase de ROV's con muchas limitaciones en lo que a aplicaciones se refiere, pero con la ventaja de que los costes de producción disminuyen notablemente, mejorando las posibilidades de inserción en un mercado más generalista, ampliando así el espectro de potenciales clientes.

En lo que a comunicaciones se refiere, todos los ROV's emplean un cordón umbilical debido a las limitaciones que tienen las comunicaciones inalámbricas dentro del medio. Por poner un ejemplo, a una distancia de 200 metros la máxima velocidad alcanzable para comunicaciones electromagnéticas es de 100 bps, según indica Palmeiro et al. en su publicación 'Underwater radio frequency communications' [2].

Habitualmente, los cordones umbilicales emplean conductores de cobre para reducir los costes. En la tabla 6 del artículo de Capocci et al. [1], se pueden consultar las características de los protocolos de capas físicas más empleados. Aquí se resumen las características más relevantes:

| Características | RS-232 | RS-485 | Gb Ethernet |
|---|---|---|---|
| Máxima longitud | 15 m | 1200 m | 100 m |
| Máxima velocidad de transmisión | 20 kbps | 10 Mbps (15 m) | 1 Gbs |

Tabla 1.1: Cuadro comparativo entre los protocolos empleados a nivel comercial en los cordones umbilicales.

Como se puede apreciar, el protocolo que se emplee en la capa física afecta directamente a la longitud del cordón umbilical, esencial en ciertas aplicaciones, y a la máxima velocidad de transmisión, también muy relevante en ROV's que necesiten visión en tiempo real.

Por último, los esquemas de control de los ROV's dependen directamente del número de propulsores y la disposición de los mismos. Si bien existen configuraciones de dos motores que emplean sistemas para modificar el lastre, generalmente empleando movimiento de fluidos en el interior del ROV, lo más habitual es que se parta de 4 motores para garantizar la maniobrabilidad necesaria dentro del medio. La configuración más habitual es la empleada por BlueRobotics en la configuración estándar de su BlueRov2 1.3, donde emplean una configuración de 6 propulsores dispuestos de forma vectorial 1.4.

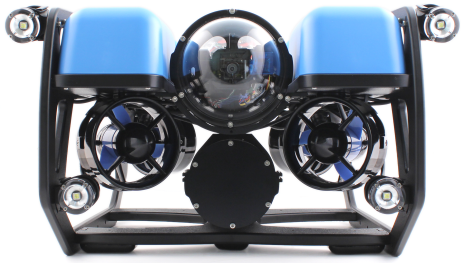


Figura 1.3: Configuración estándar de dron submarino de la empresa BlueRobotics.

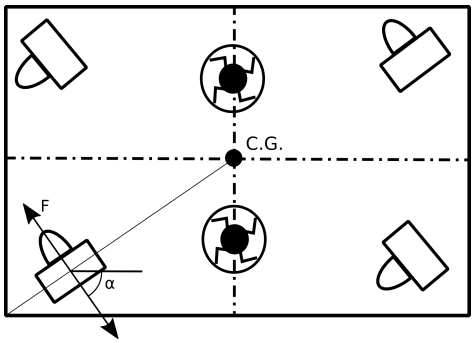



Figura 1.4: Disposición de los propulsores en una configuración vectorial

En la imagen se puede observar cómo 4 de los motores se disponen de tal forma que para avanzar y retroceder se equilibren las componentes horizontales de sus fuerzas de propulsión. Además, se usan 2 propulsores centrados y perpendiculares para poder emerger y sumergirse con mayor facilidad.

La configuración de 6 motores garantiza 6 grados de libertad en unas condiciones de flotabilidad neutra, mientras que el empleo de 4 motores en flotabilidad neutra limita el número de grados de libertad, no pudiendo desligar, por ejemplo, el avance del ascenso o descenso del dron.

## 1.4. Metodología

El proceso empleado para la evaluación y evolución de los sistemas que van a ser descritos a continuación se ha realizado mediante pruebas empíricas en las que se ha analizado el desempeño de cada sistema.

Dichas pruebas han consistido en repetidas inmersiones con una duración comprendida entre 5 y 30 minutos en la playa de Las Teresitas, en Santa Cruz de Tenerife. Las condiciones de las inmersiones se pueden describir como aceptables, habiendo corrientes ligeras, oleaje suave y viento moderado. En lo que se refiere a las condiciones de visibilidad, se han llevado a cabo de día facilitando así la labor del operador de tierra.

Todos los datos tomados poseen una referencia de tiempo única provista por el maestro de ROS empleado, tal y como se desarrollará en el Capítulo 2 de la memoria, lo cual permite relacionar con facilidad el valor de los ángulos del submarino, profundidad y valor de los actuadores en cualquier momento registrado.

Por razones de seguridad, durante la realización de las primeras pruebas el submarino se encontraba asegurado con un cabo a tierra, indicando además su posición con una boya en la superficie, lo cual influía ligeramente en el comportamiento del dron.

Por último, durante todas las pruebas se han empleado 2 personas sumergidas para grabar y auxiliar al submarino en situaciones complejas, así como un operario encargado de la administración del cordón umbilical y otro operario dedicado al control del propio submarino desde tierra.

## 1.5. Estructura del documento

El presente documento se estructura en 3 tomos: memoria, pliego de condiciones y anexos.

En el primer capítulo del tomo I se ha introducido al proyecto marco del que emana este TFG y se ha hecho un repaso global del estado actual de las tecnologías empleadas en ROV's. En el capítulo 2 se describen los materiales hardware y las herramientas software empleadas durante la realización del subproyecto. En el capítulo 3 se detalla el sistema de comunicaciones desarrollado para el dron mientras que en el capítulo 4 se describe el sistema de control desarrollado. Finalmente, en el capítulo 5 se concluye la memoria con un resumen de los objetivos cumplidos.

En el tomo II, el capítulo 6 se describen los objetivos que debe cumplir el TFG y en el capítulo 7 se detalla el coste económico del desarrollo del mismo. Por último, en el tomo III se adjuntan los códigos realizados para la construcción de los diferentes sistemas descritos en la memoria.

# Capítulo 2

# Material

En este capítulo se describen los elementos físicos que han sido empleados para la consecución de los objetivos específicos previamente mencionados 1.2, así como las herramientas software usadas. Asimismo, se explicará la configuración básica de los sistemas operativos empleados y las dificultades que han aparecido durante el proceso de configuración. También se hace una breve explicación de ROS -*Robot Operating System*-, sus funcionalidades básicas y las ventajas que ofrece en el ámbito de la robótica móvil.

## 2.1. Hardware

### 2.1.1. Raspberry Pi

Las Raspberry Pi [3] son unos ordenadores en minuatura, que emplean tecnología SoC (System on Chip). Su reducido tamaño y alta potencia ha hecho que se extiendan para aplicaciones de bajo coste que requieran capacidad de procesamiento superiores a las de los microcontroladores.

Asimismo, resulta destacable la versatilidad que ofrece para cualquier proyecto en el ámbito de la robótica, pues existen diversos sistemas operativos diseñados para su uso en Raspbery, así como una amplia comunidad.

En la actualidad, los modelos más avanzados que existen son la Rapsberry Pi 3B y 3B+. Ambos modelos han sido usados en el dron, aunque por especificaciones de la tarjeta de red, se escogió finalmente la Raspbery Pi 3B+.

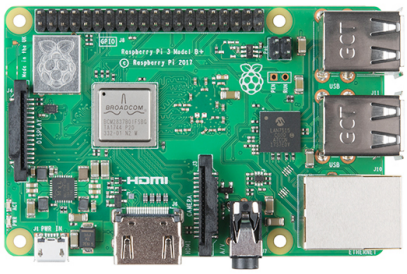


Figura 2.1: Raspberry 3B +

### 2.1.2. Arduino

Al igual que ocurre con las Raspberry Pi, los Arduino [4] son unos micro-controladores mundialmente extendidos para aplicaciones de bajas prestaciones debido a la facilidad de uso y de programación 2.2.

Figura 2.2: Arduino Mega Original

En el dron se han usado 2 Arduinos Mega, uno para administrar las lecturas de todos los sensores de a bordo y el segundo para controlar los motores. Este último es una versión adaptada por la comunidad, con el mismo procesador como base, para su uso en drones aéreos que se denomina *ArduPilot* 2.3 e integra en un menor espacio las mismas entradas y salidas que un Arduino Mega, además de poseer integrada una IMU.


Figura 2.3: ArduPilot

### 2.1.3. Controlador de PS3

El dispositivo empleado para que el usuario pueda manejar adecuadamente el dron submarino es un mando de videoconsolas. Este tipo de soportes tienen grandes ventajas a nivel funcional, ya que disponen de 2 joysticks, con 2 grados de libertad cada uno, y un total de 16 botones y 2 potenciómetros.

En concreto se ha empleado en el TFG un mando de la *PlayStation 3* 2.4 debido a que suele ser frecuente su uso en proyectos desarrollados en ROS, dada la compatibilidad existente entre el nodo *Joy*, incluido como paquete estándar para leer controladores de videoconsolas, y el mando de la PS3.


Figura 2.4: Mando de la PlayStation 3

Sin embargo, cualquier tipo de mando de videoconsolas puede ser empleado, haciendo uso de la conexión por cable USB al ordenador e instalando los drivers pertinentes.

## 2.2. Software

### 2.2.1. Ubuntu 16.04

Ubuntu [5] es un sistema operativo abierto y gratuito basado en Linux que está patrocinado por Canonical Ltd. Cada 6 meses publica una versión nueva del software soportada por la comunidad durante 9 meses, mientras que cada 2 años lanza una versión LTS, *Long Term Support*, que se mantiene durante 5 años.

En el momento en que el subproyecto empezó, existían dos versiones LTS con varios años de soporte, las versiones 16.04 (Xerial Xerus) y 18.04 (Bionic Beaver), que salió durante el año 2018. Debido a que ROS se vincula con cada versión LTS de Ubuntu y que los paquetes disponibles en la comunidad de ROS no tienen por qué ser compatibles entre versiones consecutivas, se eligió la versión 16.04 por disponer de mayor documentación en la comunidad de Ubuntu y de ROS.

### 2.2.2. Ubuntu Mate 16.04

Ubuntu Mate [6] es una versión ligera de Ubuntu pensada para sistemas de bajas prestaciones, entre los que se incluye la Raspberry Pi. Debido a que ROS ha sido diseñado para Ubuntu, la mayor parte de los desarrollos se suelen realizar en una versión ligera de Ubuntu, como lo es Ubuntu Mate. Esto suele reducir el número de problemas entre el sitema operativo y los repositorios de ROS.

Sin embargo, la versión de Ubuntu Mate 16.04 no es compatible con la Raspberry Pi 3B + ya que ésta incluye un gran número de cambios en el hardware que desfasa la versión 16.04. Por este motivo, y ante la imposibilidad de que en un mismo sistema de ROS existan dos versiones de Ubuntu sin que aparezcan graves problemas de estabilidad en las comunicaciones, se optó, tras realizar varias pruebas empleando Raspbian Jessie y Ubuntu Mate 18.04, por mantener Ubuntu Mate 16.04 y subsanar los problemas de compatibilidad existentes.

Para ello se siguieron diversos foros oficiales de la comunidad de Raspbery [7], Ubuntu Mate y Raspbian. El proceso seguido consistió en sustituir algunos archivos de configuración de la secuencia de boot por los correspondientes de un sistema Raspbian Jessie, que sí es compatible con el Hardware de la Raspbery 3B+. Esta solución no hace que el sistema funcione perfectamente, ya que aparecen problemas de conectividad WiFi, pero garantiza un sistema estable en el que todas las funciones al margen del WiFi se comportan adecuadamente.

### 2.2.3. ROS

ROS [8], *Robot Operating System*, es un framework destinado al desarrollo de software para robots. Ofrece herramientas, librerías y estándares que simplifican

el desarrollo de aplicaciones complejas en el ámbito de la robótica.

La filosofía de ROS se basa en ofrecer un marco común, libre y colaborativo en el que se pueden emplear desarrollos liberados por otros particulares, empresas e instituciones para facilitar el desarrollo de robots de diferente índole. Esta forma de colaboración ayuda a que los desarrollos de nuevos robots puedan partir de la tecnología existente, ya que bajo la plataforma ROS se puede conseguir la integración de varias tecnologías desarrolladas de forma independiente, como puede ser paquetes de mapeo, navegación y reconocimiento de objetos. De esta forma, resulta más sencillo que un grupo de investigación o desarrollo se centre en un área teniendo la posibilidad de disponer ya de tecnología funcional que les permita crear un sistema complejo.

Dentro de las herramientas que ofrece ROS, está la gestión automática de las comunicaciones entre dispositivos diferentes dentro de la misma red, con una relación maestro-esclavo. De esta forma, el maestro se encargaría de anunciar a todos los nodos de la red los tópicos activos. Asimismo, también administra las suscripciones a los diferentes tópicos, notificando y haciendo llegar a cada nodo los mensajes disponibles en cada tópico de interés.

La estructura de las comunicaciones en ROS se basa en el empleo de nodos y tópicos:

- Nodo: se puede asimilar a un programa o proceso dentro del sistema. Cada nodo debe definir los tópicos a los que se quiere subscribir y los tópicos en los que va a publicar mensajes.

- Tópico: los tópicos son buses donde se publica un determinado tipo de mensaje, bien estándar o bien definido por el usuario. El maestro gestiona las publicaciones y suscripciones a cada tópico, siendo los nodos inconscientes de la procedencia de los mensajes.

Con la finalidad de estandarizar las comunicaciones, en ROS se definen mensajes estándar. Estos mensajes pueden ser simples *Strings* o mensajes de mayor complejidad como de tipo *Joy* que transmiten de forma unificada y estandarizada los valores de los joysticks, botones y potenciómetros de cualquier tipo de mando de videoconsola. Además, los mensajes se pueden guardar en bases de datos conocidas como *rosbags*, para poder analizar y recrear el comportamiento exacto del sistema. Esto se puede hacer debido a que se dispone de un *time stamp* en cada mensaje, y el tiempo de ejecución es único para todos los nodos de la red. También se dispone de la posibilidad de definir nuevos tipos de mensajes según las necesidades de cada desarrollo.

Los lenguajes básicos de ROS son C++ y Python. Debido a que todos los nodos se ejecutan de manera aislada con un protocolo de comunicaciones común,

lo habitual es que se empleen diferentes tipos de lenguajes dentro de un mismo sistema.

Las versiones de ROS se desarrollan para cada versión estable de Ubuntu. A pesar de que es compatible con versiones Debian e incluso de Windows, el mejor desempeño de ROS solamente es alcanzable con Ubuntu, entendiendo por mejor desempeño el uso de ROS sin tener que afrontar numerosos errores a la hora de desarrollar y de emplear las herramientas predefinidas del sistema.

En el dron submarino se ha empleado la versión ROS Kinetic Kame [9], asociada a Ubuntu 16.04. La elección de esta versión se justifica en el numeroso número de paquetes disponibles para ella. Los paquetes de ROS son funcionalidades cerradas que comparten los diferentes usuarios, como si de una librería se tratase. Dichos paquetes no tienen por qué ser compatibles con otras versiones de ROS. A la hora de salir una nueva deben ser testeados y actualizados para garantizar esa compatibilidad. A consecuencia de esto, en el momento en que se empezó el desarrollo no existían numerosos paquetes muy potentes de reconocimiento de imagen y administración de la cámara en la versión ROS Melodic Morenia [10], asociada al Ubuntu 18.04, que sí estaban disponibles para ROS Kinetic.

Por otro lado, existe una gran comunidad y foros destinados a ROS Kinetic, lo cual simplificaba la solución de errores durante el desarrollo del software del dron submarino.

### 2.2.4. Arduino IDE

Arduino es una empresa que se dedica al diseño de software y hardware bajo licencias abiertas. Para simplificar la programación de sus placas de desarrollo, diseñaron un entorno gráfico de desarrollo denominado, que ofrece herramientas de compilación, correción de sintaxis, carga de programas y monitorización de las señales del arduino empleando un cable serial. Si bien el entorno gráfico simplifica la labor de programación, cuando no se dispone de una pantalla resulta imposible usarlo. Sin embargo, es posible emplear el Makefile de Arduino para compilar y cargar un código empleando la consola de comandos. Este procedimiento ha sido utilizado ampliamente en el dron submarino para cargar los programas empleando una terminal de la Raspberry, utilizando SSH desde el ordenador de superficie.

Una de las ventajas de emplear placas de Arduino es la gran comunidad que existe, estando extendidas las placas a nivel mundial. Asimismo, ROS dispone de una librería *ROSLIB* que posibilita emplear ROS dentro de una placa Arduino convencional, así como otro paquete *rosserialpython* que permite comunicarse por conexión serial con las placas. Se ha de tener en cuenta que ROS sobrecarga la memoria del microcontrolador debido a la complejidad de la librería que se emplea. Por este motivo, resulta de especial interés la optimización de las comunicaciones y las operaciones que se hacen dentro del microcontrolador, pues un

programa complejo puede ocasionar inestabilidad en las comunicaciones con la placa, debido a pérdidas de sincronización y retrasos frecuentes.

### 2.2.5. LaTeX

LaTeX es un lenguaje de composición de textos con una elevada calidad tipográfica que se basa en macros TeX, desarrolladas por Leslie Lamport. Este lenguaje se utiliza especialmente en el ámbito académico ya que cumple con el nivel tipográfico exigido por las revistas de mayor prestigio. Por otro lado, resulta muy común su uso cuando en el texto se deben incluir operaciones matemáticas, debido a la facilidad de introducirlas.

A diferencia de editores como Microsoft Word o LibreOffice Writer, en LaTeX no se ve el resultado final del texto mientras se escribe, sino que se incluye el texto plano y se añaden órdenes para alterar el formato, incluir figuras etc. Ello simplifica la labor de redacción del texto y el procesado gráfico del mismo, pues se automatiza el formateo en base a las órdenes indicadas.

El presente documento ha sido realizado completamente en LaTeX.

### 2.2.6. Git

Git [11] es un software de control de versiones diseñado por Linus Torvalds con la finalidad de manejar el desarrollo de código fuente en equipos de trabajo. Actualmente, es usado en proyectos de software muy complejos como el del Kernel de Linux, y también por compañías prestigiosas como Google, Facebook y Microsoft.

El control de versiones se utiliza para llevar un control temporal de los cambios realizados en un código fuente, así como identificar los cambios y tener la posibilidad de volver a versiones anteriores. También es habitual que se creen ramas dentro de los desarrollos para probar nuevas estrategias y luego unificarlas con la rama principal o descartar esos cambios. Gracias a este tipo de software varios ingenieros pueden trabajar de forma simultanea en un proyecto sin que ello acabe en problemas de unificación y pérdida de versiones estables.

En el proyecto se ha utilizado Git empleando el servicio online de GitHub Inc. [12], una empresa propiedad de Microsoft que ofrece una plataforma de trabajo colaborativo basado en Git para proyectos de desarrollo de software.

### 2.2.7. Geany

Geany [13] es un editor de texto diseñado para ofrecer un entorno integrado de desarrollo, también conocido como IDE, ligero y rápido que ofrece la posibilidad de desarrollar en múltiples lenguajes de programación.

### 2.2.8. Lenguajes de programación

A lo largo del TFG se han empleado diversos lenguajes de programación, pero de entre ellos se destaca el uso de C++ y Python:

- C++: es un lenguaje de programación de alto nivel multiparadigma, que incluye programación orientada a objetos y programación estructurada.
- Python: es un lenguaje de programación interpretado que intenta facilitar la lectura del código. Al igual que C++ es multiparadigma, ya que incluye programación orientada a objetos, imperativa y funcional. En aplicaciones complejas, el uso de python reduce la complejidad del código gracias a algunas de sus cualidades, como el tipado dinámico. Además, es multiplataforma y no debe ser compilado en cada dispositivo como es el caso de C++.

# Capítulo 3

# Comunicaciones

En este capítulo se van a explicar los detalles del desarrollo, implementación y verificación de las comunicaciones del dron submarino.

## 3.1. Elementos del sistema

El dron submarino consta de 4 elementos bien diferenciados que deben comunicarse:

- Estación de control: ordenador en superficie que posee la interfaz de control y el procesamiento de imágenes.
- Ordenado de abordo: actúa de ordenador dentro del dron, debe procesar la información de la IMU conectada a su interior, mandar la imagen de la cámara USB y el audio de su micrófono.
- *arduSensor*: placa Arduino Mega que se encarga de leer los sensores y controla una luz para mejorar la visibilidad en circunstancias de baja luminosidad.
- *arduMove*: placa ArduPilot que se encarga del control de los propulsores del dron.

De estos elementos se desprenden las siguientes necesidades:

1. Transmisión de datos entre la placa *arduSensor* y la estación de control: la lectura de los sensores deberá ser enviada a la superficie para que ésta sea incluida en la interfaz de control.
2. Transmisión de datos entre la estación de control y la placa *arduSensor*: envío de la señal de activación/desactivación de la luz del dron.

3. Transmisión de datos entre la estación de control y la placa *arduMove*: las instrucciones de control deben transmitirse a la placa *arduMove*, que se encargará de traducirlas para generar señales PWM que muevan los propulsores.

4. Transmisión de datos entre el ordenador de abordo y la estación de control: se deberá enviar la imagen, el sonido y los valores de la orientación del dron.

Por tanto, el sistema que se diseñe deberá realizar de forma exitosa todas las transmisiones anteriormente descritas empleando la topología y el transmisor que mejor se adapte a las necesidades de diseño, mejor integración tenga con el hardware a comunicar y mejor se ajuste al presupuesto disponible.

## 3.2. Topología

Como se expuso anteriormente, es necesario que la estación de control reciba la información de todos los elementos del sistema, y que las órdenes provistas por el controlador a instancias del operador se lleven correctamente a las placas *arduMove* y *arduSensor*. Si se tiene en cuenta que físicamente ambas placas y el ordenador de abordo se encuentran localizados en el interior del submarino y que la estación de control se encuentra en la superficie, queda patente que deberá existir un medio transmisor entre el dron submarino y la superficie.

Como se explicó en la introducción del documento, la comunicaciones inalámbricas debajo del agua reducen drásticamente su rendimiento, imposibilitanto su uso. Por este motivo, el medio transmisor ha de ser un cable o cordón umbilical que tenga suficiente capacidad para transmitir datos a una distancia mínima de 30 metros y máxima de 75 metros. Dichas distancias son consistentes con la profundidad máxima del dron, fijada en 20 metros, y con la distancia de operación, que por ser operaciones en puertos serán reducidas.

El cordón umbilical escogido es un cable Ethernet de categoría 5e de tipo UTP y con recubrimiento de PVC. Las especificaciones de este cable soportan hasta Gigabit Ethernet, aunque el estándar para ese nivel de velocidad es la categoría 6. Sin embargo, supera los 100 Mbps del estándar anterior, situando a las comunicaciones en un rango de velocidades alto que permite al sistema rendir de forma aceptable en cualquier circunstancia. Además, el coste del cable de categoría 5e es significativamente menor al de categoría 6, pudiéndose comprar 30 metros de un cable tipo Cat 5e UTP por la mitad de precio que el de longitud equivalente de categoría 6. Pese a ser un cable de mayores prestaciones, la diferencia de precio es excesiva teniendo en cuenta que la categoría 5e ofrece suficiente velocidad para enviar la información necesaria.

Como consecuencia de disponer de un único cordón umbilical, las comunicaciones entre el submarino y la superficie deberán centralizarse, por lo que los microcontroladores se conectarán al ordenador de abordo y éste a su vez establecerá una red con la estación de control. EL medio físico que se empleará para comunicar los microcontroladores y el ordenador de abordo será un cable USB sobre el que se establecerá una comunicación serial.

De esta manera, la red quedaría establecida con la topología y flujos de información establecidos en la figura 3.1. En dicha figura se ha destacado el ordenador de abordo debido a que en ROS se le escogió como maestro de la comunicaciones, explicándose más adelante la razón de dicha decisión.

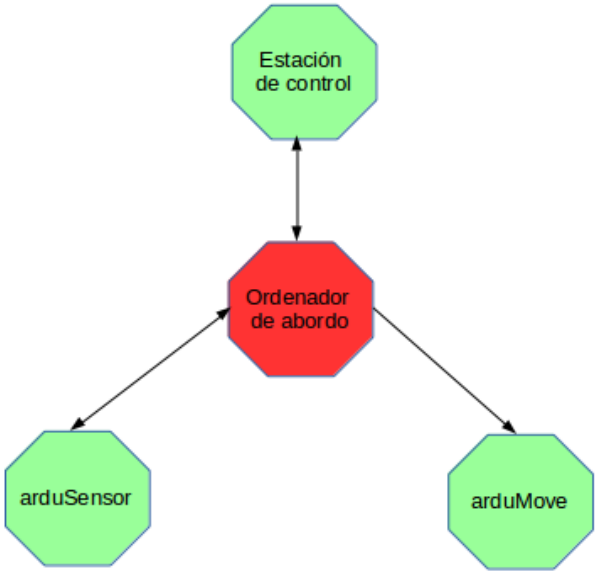



Figura 3.1: Topología de la red del dron submarino

## 3.3. ROS

### 3.3.1. Red

Las comunicaciones del dron giran entorno a una eficaz conexión entre la estación de control y el ordenador de abordo. Con la finalidad de que el enlace sea estable se ha configurado una red privada con entre estos 2 elementos. Esta red se caracteriza por centralizar el acceso a Internet a través de la estación de control en superficie 3.2, pasando todo el tráfico a través de la tarjeta de red de este dispositivo. El ordenador de abordo tiene asignado una IP fija dentro de la red privada -10.42.0.238- en la que la estación de control -10.42.0.1- actúa como servidor

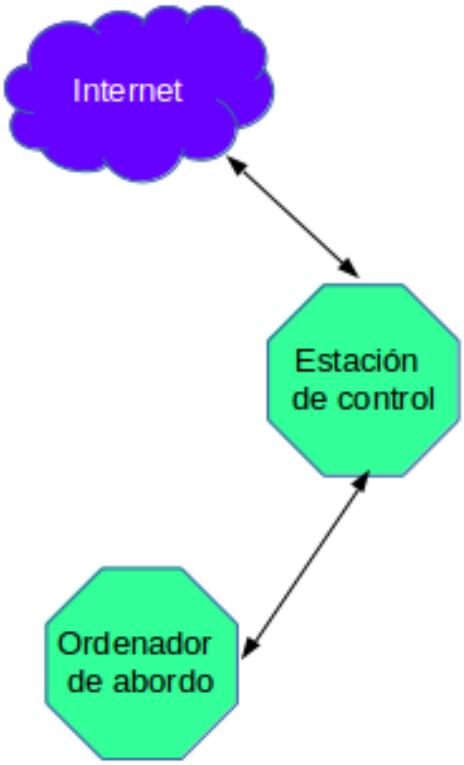



Figura 3.2: Esquema de la red privada establecida

DHCP. Sin embargo, esta arquitectura se mantiene incluso cuando no existe acceso a Internet, ya que la red se configura de la misma manera. En la figura 3.3 se observa la configuración que se ha realizado en la estación de control para establecer la red privada. En este proceso se ha aprovechado la capacidad de *compartir la conexión con otros equipos* que ofrece Ubuntu de forma nativa. Con este tipo de configuración, simplemente es necesario seleccionar como tipo de red cableada la asignada al submarino cuando se conecta un cable ethernet a la estación de control 3.4 para que se establezca un enlace directo y exitoso entre el ordenador de abordo y la estación de control. En el lado de el ordenador de abordo no es necesario hacer ninguna configuración de red en especial, simplemente se debe establecer una IP fija dentro de la red privada para que se pueda acceder de forma remota vía SSH y para que dentro de ROS se puedan indicar las IP con una única configuración.

El único requisito previo para poder comunicar dos dispositivos empleando ROS es que todos los dispositivos estén situados dentro de la misma red. El enlace entre la estación y el ordenador de abordo en el interior del submarino se establece siguiendo los pasos anteriores, mientras que el correspondiente a los dos microcontroladores y el propio ordenador de abordo se realiza desde que se

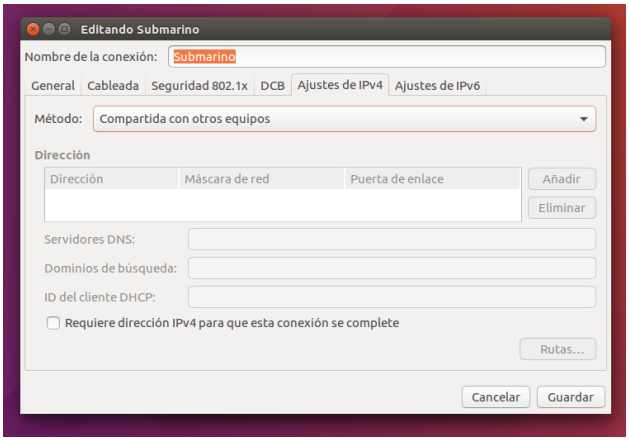



Figura 3.3: Configuración de la conexión cableada en la estación de control para establecer la red del submarino


Figura 3.4: Notificación de que se ha establecido exitosamente la conexión física previamente configurada

conecta el cable USB de cada microcontrolador.

Las comunicaciones en ROS se gestionan por un dispositivo maestro, denominado como *ROS Master*, que debe ser único. En el dispositivo maestro se va a lanzar el comando *roscore*, que inicializa todos los procesos internos de ROS. Todos los dispositivos, exceptuando aquellos que se conecten vía serial como los microcontroladores, deben de conocer la dirección IP del maestro, el puerto de acceso y también la dirección propia. Para ello deberán definirse las variables de entorno tanto en la estación de control como en el ordenador de abordo.

Las variables de entorno que se deben fijar son *ROS_MASTER_URI* y *ROS_HOSTNAME*, correspondiéndose la primera variable a la IP y puerto de acceso del dispositivo que va a actuar como maestro y la segunda variable con la IP del propio dispositivo. De esta manera deberán configurarse de la siguiente manera según el dispositivo:

- Ordenador de abordo: ROS_MASTER_URI = http://10.42.0.238:11311 y ROS_HOSTNAME = 10.42.0.238
- Estación de control: ROS_MASTER_URI = http://10.42.0.238:11311 y ROS_HOSTNAME = 10.42.0.1

En la figura 3.5 se puede ver el resultado de la configuración realizada en la estación de control. El caso del ordenador de abordo será similar con la salvedad de la dirección IP del dispositivo, tal y como se acaba de indicar.


Figura 3.5: Resultado de la configuración de las variables de entorno de ROS

Se puede observar que el ordenador de abordo fue escogido como maestro de ROS dada la configuración establecida. Habitualmente, en robótica se suele fijar el maestro de ROS dentro del ordenador que se encuentra conectado a los microcontroladores que actúan y sensan el sistema. De esta forma, en caso de que hubiese alguna falla en el enlace entre la estación de control y el ordenador de abordo, el sistema ROS seguiría siendo operativo y seguiría funcionando correctamente, incluso podría tomar acciones de emergencia para entrar en un modo seguro hasta que la estación de control recupere el contacto.

Si la estación de control fuera el maestro, el ordenador de abordo tendría una carga menor de procesos, pero el sistema en su conjunto sería más inestable, por lo que el ahorro de CPU no justificaría dicha decisión. Cabe destacar que en caso

de caída de la red, gracias a que el maestro de ROS sigue estando operativo, la estación de control podría retomar el contacto cuando se solucione el problema de conectividad sin que ello afecte a la operatividad del sistema y sin que deban reiniciarse los nodos en funcionamiento.

Por otro lado, de esta manera se habilita a que en futuras mejoras se integren funciones para administrar los errores en caso de pérdida de conexión, como podría ser dejar el submarino en equilibrio o ascender hacia la superficie para que sea recuperado con mayor facilidad. Se ha de tener en cuenta que estas opciones no serían posibles si se situara el maestro fuera del dron.

El manejo de errores en lo que respecta a poner el submarino en modo seguro o que ascienda a superficie no se ha realizado por excederse de los objetivos establecido en el trabajo, aunque se ha identificado como punto de mejora en caso de que el proyecto siguiera desarrollándose.

### 3.3.2. Esquema comunicaciones internas

Como se explicó en el capítulo 2, ROS las comunicaciones en ROS se basan en el uso de nodos, cuyo concepto más próximo es el de procesos; y tópicos, que son buses en los que se envían mensajes de un único tipo. En la figura 3.6 se puede observar un diagrama con los nodos operativos cuando todo el sistema del dron submarino está funcionando, exceptuando los asociados al reconocimiento de imágenes que se encontraban deshabilitados en el momento de obtener la captura. Este conjunto de nodos constituyen el esquema básico de operación, es decir, los procesos que como mínimo deben estar operativos para que el submarino pueda ser operado sin ningún problema.

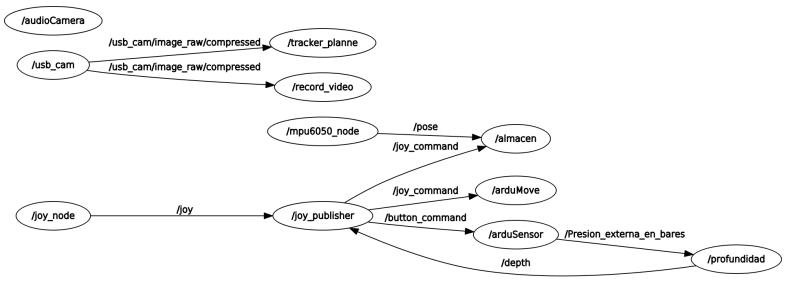



Figura 3.6: Diagrama de nodos básicos activos en el dron submarino

Los nodos asociados al reconocimiento de imágenes pueden ser vistos en la figura 3.7. La figuras se han obtenido con una herramienta de ROS denominada *rqt_graph* que crea diagrama de bloques con los nodos y tópicos activos durante una ejecución de ROS.

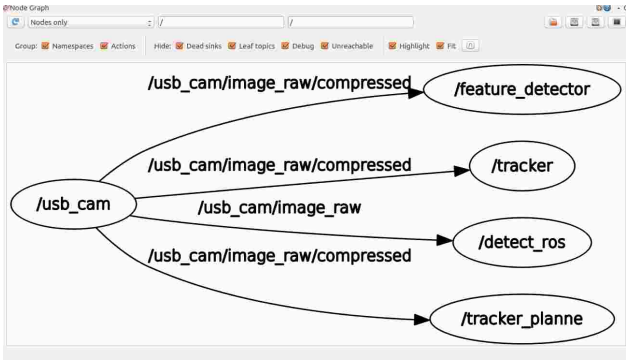



Figura 3.7: Diagrama de nodos de reconocimiento de imágenes

En la figura 3.8, se pueden ver todos los tópicos y nodos activos. El diagrama es interesante para explicar el envío de información que se debe realizar en el dron. Se puede observar que existen 11 nodos activos, identificados por elipses en el gráfico, y un total de 7 tópicos, identificados por rectángulos en la imagen.

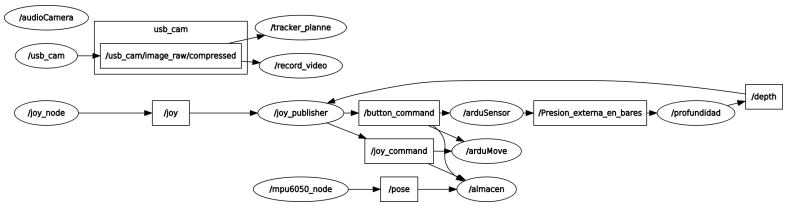



Figura 3.8: Diagrama de nodos básicos activos en el dron submarino

En líneas generales, el funcionamiento del dron es el siguiente:

– La placa arduSensor, leer los sensores y publica la información para que el operador la pueda ver en la interfaz de control. De esa información, el valor del sensor de presión externa se procesa de forma independiente para obtener la profundidad en metros a la que se sitúa el dron. La profundidad es de interés para el proceso de control, por lo que el nodo de control, *joy_publisher*, se subscribe al tópico de profundidad, *depth*.

– La cámara y el sonido también se encontrarán enviando información que será captada por la interfaz de usuario y que servirá para los nodos de reconocimiento de imagen y grabación de vídeo. Esta parte del funcionamiento del dron pertenece a otro de los TFG que componen el proyecto, desarrollado por Iván J. Torres Rodríguez 1.1, donde se explica en detalle el funcionamiento del sistema de visión. En lo que objeta a este TFG, solo importa la cantidad de información que se debe transmitir.

- Por un lado, el ordenador de abordo se encontrará supervisando constantemente la inclinación del submarino empleando al nodo *mpu6050_node*, que será publicada en el tópico *pose* en forma de cuaterniones. Este sensor y el software empleado también pertenece al TFG mencionado anteriormente.

- Por otro lado, el operador enviará órdenes empleando en mando de videoconsola disponible. Ese mando se lee con un nodo estándar de ROS, denominado *joy_node*, que publicará con un mensaje de tipo *Joy* el valor de todos los botones, joysticks y potenciómetros del mando. A ese tópico se subscribe otro nodo que será el encargado de calcular los valores que deben tomar los propulsores para cumplir con las órdenes del operador. Existen dos formas de interpretar las órdenes del operador, se puede estar en un modo manual en el que las órdenes del operador son interpretadas como velocidades angulares en cada uno de los 3 ejes del submarino y una cuarta componente de potencia; o se puede interpretar como incremento de ángulos respecto de la posición actual, para lo que se emplea un controlador que será descrito posteriormente. La salida de este nodo será por un lado un vector de 4 componentes de tipo *float32* que será publicado como comandos para los motores y por otro lado los botones del mando que resultan de interés ordenados en un vector de tipo *int32*. Esa información se publica en los tópicos *joy_command* y *button_command* respectivamente.

- Las órdenes del mando serán leídas por el nodo establecido en el arduPilot, denominado *arduMove*, que actuará sobre los motores. Si bien las comunicaciones con el el microcontrolador son parte de este TFG, la traducción de los valores de comando a señales de motor y el software de movimiento integrado dentro del arduPilot pertenecen a otro de los TFG que componen el proyecto, cuyo autor es David Henry González 1.1.

- Por último, existe un nodo denominado *almacen* que se encarga de almacenar los datos de la IMU, la profundidad en metros y los comandos que se envían desde la estación de control. La función de este nodo es la de guardar el comportamiento del sistema para poder modelar el dron y mejorar el controlador.

Si bien en la figura 3.8 aparecen todos los tópicos y nodos básicos para que el dron funcione, en el momento de obtener la captura el dron se encontraba en modo manual, por lo que no existe conexión entre el tópico de la IMU y el nodo que calcula los comandos. Asimismo, solo aparece el tópico asociado al sensor de profundidad, ya que la interfaz de control no estaba encendida. Cuando la interfaz se enciende, aparecen como activos todos los tópicos que se publican por el nodo establecido en el microprocesador *arduSensor*.

Se puede observar que las comunicaciones del dron son muy dinámicas y que existe una fuerte interacción entre los distintos dispositivos que componen la red del submarino. En especial, resulta de interés remarcar que la información transmitida desde los microcontroladores y hacia los microcontroladores se ha minimizado y optimizado en gran medida para mejorar el rendimiento.

En una etapa inicial, el nodo arduMove incluía un controlador interno y hacía un lectura de una IMU interna al hardware del micrcontrolador para aumentar la velocidad de respuesta del dron. Sin embargo, la carga que supone la librería de ROS para el microprocesador AVR ATMega2560 que se encuentra dentro del hardware del arduPilot es excesiva. Por este motivo, la implementación de muchas operaciones y lecturas de sensores dentro del nodo arduMove resultó ser más ineficaz, inestable y lenta que realizar las operaciones de forma externa en la estación de control y enviar un mensaje de tipo *float32Multiarray* con una frecuencia suficientemente elevada para que con la dinámica del sistema se pudiera implementar de forma exitosa el controlador, siendo dicha frecuencia de 5 Hz.

### 3.3.3. Limitaciones de ROS en placas Arduino

Acorde a la documentación oficial de ROS [14], existen una serie de limitaciones importantes en la librería de ROS [15] para Arduino que se han tenido en cuenta a la hora de diseñar la comunicación entre las placas *arduSensor*, Arduino Mega y el ordenador de abordo.

El punto más relevante de la documentación técnica de ROS es el que indica la limitación de *publishers* y *subscribers*. El procesador incluido en *arduSensor* y *arduMove* es el AVR ATMega2560, tal y como se indicó en el apartado 3.3.2. Además, la flash disponible es de 256KB, junto con una SRAM de 8KB y una EEPROM de 4 KB.

Los objetos *publishers* pertenecen a la clase *Publisher* que es un tipo de clase capaz de enviar mensajes del tipo declarado a un tópico. Por su parte, los objetos *subscribers* pertenecen a la clase recíproca que se encarga de leer los mensajes del tipo declarado en un tópico indicado en su declaración. En lo que se refiere a las comunicaciones, son dos de los elementos más importantes dentro de ROS ya que son los que implementan los procesos de envío y recepción de mensajes. Con las características de memoria de las placas instaladas, *arduMove* y *arduSensor*, el número máximo de objetos de tipo *Publisher* es de 25, al igual que del tipo *Subscriber*. Esta limitación resulta de especial importancia en la placa *arduSensor*, ya que se han implementado 9 objetos de tipo *Publisher* y 1 objeto de tipo *Subscriber*. Pese a que se encuentra por encima del máximo indicado en las especificaciones técnicas, durante la implementación su implementación se observa que la placa se encuentra en el límite de sus capacidad, resultando difícil aumentar el número de objetos *Publishers* y *Subscribers* sin que ello se traduzca en un em-

peoramiento del desempeño del programa. Esto se debe a la carga que suponen los propios mensajes que se van a enviar, pertenecientes en su mayoría al tipo interno de ROS *sensor_msgs* que incluyen marcado temporal.

Los máximos establecidos en la documentación técnica no especifican el tipo de mensaje estándar que usaron para establecer el límite. En ROS existen mensajes muy complejos, como lo es el tipo Pose, que contiene en su interior coordenadas cartesianas e inclinación en cuaterniones, además de las marcas temporales. Es por este motivo que el número de objetos para enviar y recibir datos debe ser administrado con cautela, así como el tipo de mensajes que se vayan a emplear. A pesar de que para este subproyecto no ha sido necesario, existen protocolos de comunicación como Mavlink que optimizan el envío de datos, reduciendo la memoria necesaria, y librerías de ROS como *rosabridge* [16] que modifican la estructura de mensajes demasiado pesados para las placas Arduino, como es el de odometría, *odom*.

Por otro lado, existe otra limitación relevante para la placa *arduMove*. Debido a las limitaciones del procesador, las placas con procesadores ATMega2560 no pueden trabajar con mensajes de 64 bits, como Float64. Esta condición se tuvo en cuenta a la hora de enviar los comandos de los motores, usando en su lugar mensajes de tipo Float32.

Por último, debido a las limitaciones de memoria de la placa, los *Arrays* no pueden ser tratados como el tipo *Vector* de C++. Por este motivo los mensajes que incluyan un *array* de números serán tratados como se hace habitualmente en Arduino. Para compensar que no existen formas eficaces de averiguar el tamaño del *array*, se añade una variable extra en la definición de los mensajes que emplean *arrays* que contendrá el tamaño del *array*. Esta situación afecta tanto a la placa *arduMove* como a la placa *arduSensor*. La placa *arduMove* recibe un mensaje *Float32MultiArray* y otro del tipo *Int32MultiArray*, ambos conteniendos sendos *arrays*. Por su parte, la placa *arduSensor* solo recibe un mensaje de tipo *Int32MultiArray*.

## 3.4. Implementación de las comunicaciones

En esta sección se van a desarrollar los nodos de mayor interés de entre los desarrollados en el presente TFG. Se ha de tener en cuenta que todo el sistema de visión se engloba dentro del TFG realizado por Iván J. Torres Rodríguez 1.1, y que en lo referente a este TFG solo se ha tenido en cuenta el sistema de visión para dimensionar el ancho de banda necesaria.

### 3.4.1. joy_publisher

El nodo *joy_publisher* se encarga de leer los mensajes enviados por el nodo *joy*, un nodo estándar de ROS que procesa la información de un mando de videoconsola y crea un mensaje con toda la información obtenida. Con la información obtenida, calcula los comandos que debe enviar a los motores para cumplir con las instrucciones del operador.

Desde el punto de vista de las comunicaciones, se puede observar en el fragmento de código posterior que se emplea un objeto perteneciente a la clase *NodeHandle*. Este objeto se encarga de administrar todos los eventos dentro del sistema en ROS. Dichos eventos tienen en cuenta desde interrupciones forzadas por el usuario hasta la propia detección de los nuevos mensajes disponibles.

Por otro lado, los objetos de tipo *NodeHandle* tiene un método llamado *advertise* que se emplea para indicar al maestro de ROS que se va a empezar a publicar un nuevo tópico, indicando también el tipo de mensajes que se van a publicar. En este nodo, se van a publicar dos nuevos tópicos:

- joy_command: tópico en el cual se van a publicar los comandos de los motores. El mensaje publicado es un *array* de 4 elementos de tipo Float32.

- button_command: tópico en el que se publican los botones de interés del mando de video consola. El mensaje publicado también es un *array*, pero de 8 elementos de tipo int32.

```
int main(int argc, char **argv)
{
  ros::init(argc, argv, "joy_command");


  ros::NodeHandle n;

  joy_command = n.advertise<std_msgs::Float32MultiArray>("
    joy_command", 1000);
  button_command = n.advertise<std_msgs::Int32MultiArray>("
    button_command", 1000);

  if(firstExec == 0){
    begin = ros::Time::now();
    firstExec = 1;
  }
  ros::Subscriber joy_sub = n.subscribe<sensor_msgs::Joy>("joy"
    ,10,joyCallback); // escuchamos nodo Joy de la libreria
  ros::spin();

}
```

Además de las declaraciones básicas que se han hecho en el fragmento de código anterior, existen unos *callbacks*, que son unas funciones que se activan cuando existe un mensaje nuevo de un tópico al que se ha suscrito previamente el nodo. En el caso de nodo joy_publisher, el *callback* empleado se llama *joyCallback*.

### 3.4.2. store

En el caso del nodo *store*, su funcionalidad es la de almacenar los datos de la inclinación, profundidad y comando de los propulsores de forma constante desde que se inicia la ejecución del sistema del dron submarino. Existe otro nodo, llamado *almacen* que realiza el mismo proceso, pero cuya ejecución puede lanzarse y pararse en cualquier momento, lo cual permite hacer un análisis del comportamiento del submarino sin tener que parar el sistema completo.

El esquema de funcionamiento del nodo *store* se puede ver en la figura 3.9. Si lo comparamos con el indicado anteriormente en la figura 3.8 se puede observar que se enlaza a los mismos tópicos que el nodo *almacen*: *joy_command*, *depth* y *pose*.

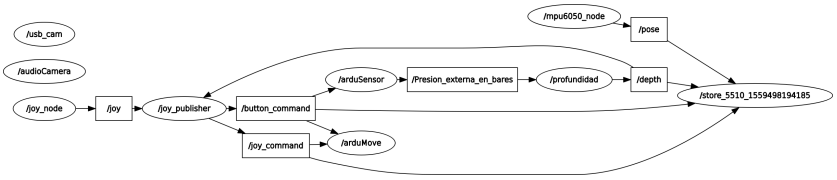



Figura 3.9: Esquema de nodos y tópicos activos cuando se usa el nodo store

A diferencia del nodo *joy_publisher*, el nodo *store* ha sido programado empleando Python. Los elementos desde el punto de vista de las comunicaciones son los mismos, se inicializará el nodo y se realizarán las subscripciones pertinentes. Cuando llegue un mensaje se activará el *callback* correspodiente y se procederá a guardar los datos. Como se puede ver en el código situado debajo, la ventaja de usar Python radica en el tipado dinámico. Los *callbacks* no necesitan que se les indique el tipo de mensaje que deben leer, sino que se emplea únicamente un nombre de parámetro formal, en este caso *data*.

```
1 def buttonsCallback(data):
2   #Detectamos la activacion y desactivacion de los motores para
      registrar correctamente
3   #si los valores que se envian se estan ejecutando o no
4   if (data.data[0] == 1) and (data.data[4] == 1):
5     motoresActivos = 1
6   if (data.data[1] == 1) and (data.data[4] == 1):
7     motoresActivos = 0
8
```

```
def ejecucion():

  rospy.init_node('store', anonymous=True)

  rospy.Subscriber("joy_command", Float32MultiArray,
    motorsCallback)
  rospy.Subscriber("button_command", Int32MultiArray,
    buttonsCallback)
  rospy.Subscriber("depth", Float32, profundidadCallback)
  rospy.Subscriber("pose", PoseStamped, imuCallBack)
  rospy.spin()
  #Version recortada. La completa se puede consultar en el anexo
     correspondiente
if __name__ == '__main__':
  home = expanduser("~")
  pathFolder = home + "/Documentos/data/"
  ejecucion()
```

### 3.4.3. arduMove

La implementación de las comunicaciones en una placa basada en Arduino se realiza de forma similar a la realizada en los apartados 3.4.1 y 3.4.2. En el código inferior, se puede observar que la estructura empleada es idéntica a C++. Es necesario tener un objeto de tipo *NodeHandle* para administrar todos los eventos y unos objetos de tipo *Subscriber* al que se le debe indicar durante su construcción el tipo de mensaje y el tópico al que se debe suscribir.

```
ros::NodeHandle nh;

ros::Subscriber<std_msgs::Int32MultiArray> sub_buttons("
    button_command",&botones);
ros::Subscriber<std_msgs::Float32MultiArray> sub_motors("
    joy_command",&motores);

void setup(){

  nh.initNode();
  nh.subscribe(sub_buttons);
  nh.subscribe(sub_motors);

}

void loop(){

  nh.spinOnce();

}
```

## 3.5. Rendimiento

En lo que se refiere a las comunicaciones, y siguiendo la línea de lo que se ha desarrollado en el capítulo actual, existe un enlace físico por ethernet que condiciona la velocidad y distancia a la que se puede situar el dron respecto del puesto en superficie. Sin embargo, el sistema debe de ser capaz de comunicarse con la suficiente fluidez como para que la imagen de cámara sea de utilidad para el operador y que las órdenes que se envíen no se encuentren desfasadas.

### 3.5.1. Análisis de la respuesta

La velocidad máxima que admite la conexión física viene determinada por las tarjetas de red de la estación de control y del ordenador de abordo, en este caso una Raspberry Pi 3B+. Con la finalidad de obtener el máximo rendimiento posible, la estación de control que se ha empleado posee una tarjeta de red que cumple el estándar Gigabit Ethernet. Asimismo, una de las razones de peso para escoger como ordenador de abordo el modelo 3B+ de la Raspberry Pi fue que por primera vez instalaba una tarjeta de red que cumplía el estándar Gigabit Ethernet, por lo que la velocidad máxima teórica entre ambos dispositivos es de 1Gbs.

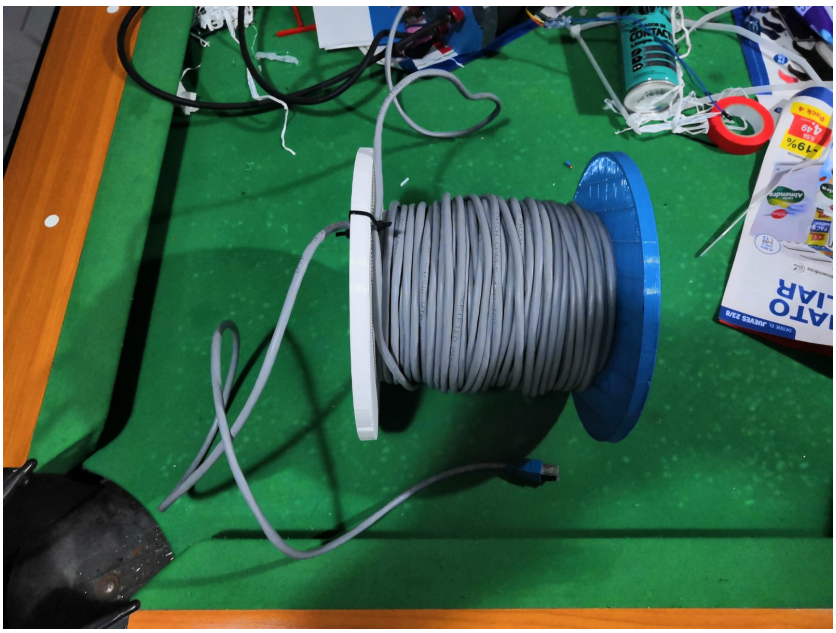


Figura 3.10: Cordón umbilical del submarino abobinado

La distancia fijada para el cordón umbilical es de 75 metros 3.10, siguiendo los objetivos de diseño especificados en el apartado 3.2. La velocidad máxima a la

que se pueden conectar los dispositivos a esa distancia es de 10Mbps Full Dúplex, con el autonegociado activo 3.11. En la figura 3.12 se puede ver que al realizar un test de las comunicaciones, utilizando el comando *ping*, no se pierde ningún paquete en el envío.

```
nicolas@nicolas-Aspire-ES1-524:~$ sudo ethtool -s enp1s0
[sudo] password for nicolas:
nicolas@nicolas-Aspire-ES1-524:~$ sudo ethtool enp1s0
Settings for enp1s0:
        Supported ports: [ TP MII ]
        Supported link modes:   10baseT/Half 10baseT/Full
                                100baseT/Half 100baseT/Full
                                1000baseT/Half 1000baseT/Full
        Supported pause frame use: No
        Supports auto-negotiation: Yes
        Advertised link modes:  10baseT/Half 10baseT/Full
        Advertised pause frame use: Symmetric Receive-only
        Advertised auto-negotiation: Yes
        Link partner advertised link modes:  10baseT/Half 10baseT/Full
                                             100baseT/Half 100baseT/Full
        Link partner advertised pause frame use: Symmetric
        Link partner advertised auto-negotiation: Yes
        Speed: 10Mb/s
        Duplex: Full
        Port: MII
        PHYAD: 0
        Transceiver: internal
        Auto-negotiation: on
        Supports Wake-on: pumbg
        Wake-on: g
        Current message level: 0x00000033 (51)
                               drv probe ifdown ifup
        Link detected: yes
nicolas@nicolas-Aspire-ES1-524:~$
```

Figura 3.11: Conexión establecida con autonegociado y un cable ethernet de 75 metros.

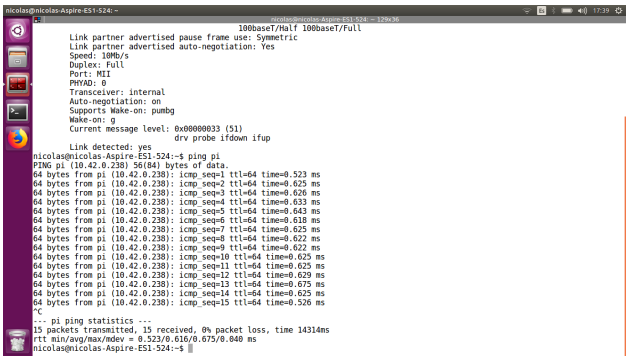



Figura 3.12: Verificación del enlace con 75 metros de cable y una velocidad de 10Mbps Full Dúplex

Sin embargo, la elevada longitud provoca que el enlace sea irregular y que se pierda la conexión entre los dispositivos en situaciones de bajo rendimiento, incluso imposibilitando el uso de SSH. Hay que tener en cuenta que cuando se pierde un paquete, el sistema intenta volver a enviarlo hasta que se reciba correctamente por la otra parte, pero si el porcentaje es demasiado elevado como lo es en este

caso, la conexión se demora excesivamente y resulta prácticamente imposible establecer cualquier tipo de conexión. A pesar de que no es un fenómeno habitual, se ha detectado realizando pruebas en puerto que en ciertas ocasiones el enlace no se llega a establecer debido a la pérdida excesiva de paquetes.

Por ese motivo, se recurrió a cambiar la configuración de la red a 10Mbps Half Dúplex 3.13 deshabilitando el autonegociado, con el fin de reducir la pérdida de paquetes incluso cuando el sistema se encuentra en la peor franja de rendimiento. Con esta nueva configuración se volvió a realizar la verificación que se había hecho previamente con una conexión Full Dúplex, cuyos resultados pueden verse en la figura 3.14. En este caso, la pérdida de paquetes vuelve a ser 0 %. Respecto a la configuración anterior se mejora la estabilidad, siendo una configuración menos susceptible a variaciones en el rendimiento de los dispositivos, en especial del ordenador de abordo.

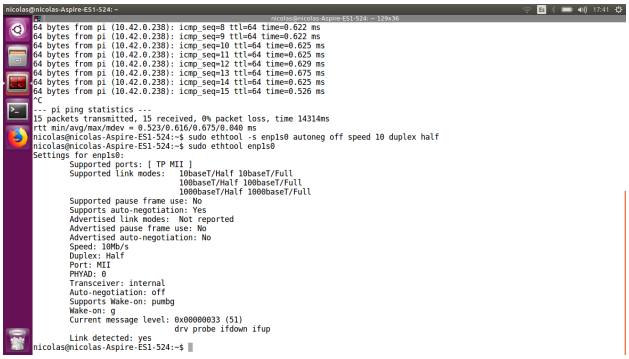



Figura 3.13: Conexión establecida sin autonegociado y un cable ethernet de 75 metros.

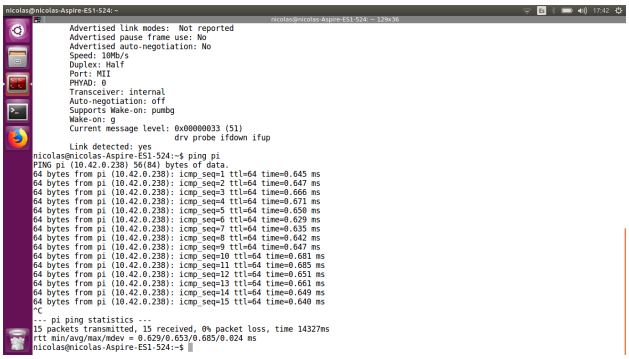



Figura 3.14: Verificación del enlace con 75 metros de cable y una velocidad de 10Mbps Half Dúplex

Sobre este sistema se probó todo el sistema funcionando para detectar, especialmente, retrasos mayores en la transmisión de los datos de la cámara. Las pruebas se llevaron a cabo inicialmente con el dron en tierra, pero tras comprobar que el enlace era fiable se realizaron pruebas dentro de un puerto de aguas tranquilas. Dichas pruebas se realizaron en las fechas siguientes y tuvieron una duración comprendida entre 1 hora y 1 hora y media en la que el dron se encontró conectado y funcionando de forma continua:

- 7-05-2019
- 8-05-2019
- 11-05-2019
- 12-05-2019
- 15-05-2019
- 27-05-2019

Durante el transcurso de las pruebas, el enlace del submarino se mantuvo siempre operativo y la estación de control recibía correctamente la imagen de la cámara con un retraso típico situado entre 1 y 3 segundos. Asimismo, el operador pudo controlador el submarino sin que se pudiera visualizar retraso entre el envío de una orden y su actuación, por lo que los retardos del flujo de datos de control es despreciable. Dentro de las órdenes de control se encuentra una orden de parada inmediata de los propulsores y su correspondiente activación. Estas acciones se producen de forma inmediata, por lo que la velocidad de respuesta en caso de emergencia se puede calificar como excelente.

De forma paralela, se realizaron pruebas en tierra para comprobar la velocidad máxima alcanzable en el enlace. La primera prueba consistió en sustituir el cable Ethernet por otro de la misma categoría de 20 metros de longitud. Para esa longitud se activó el autonegociado en la red y se obtuvo una conexión de 1Gps Full Dúplex 3.15 para la cual se realizó una verificación empleando el comando *ping* en la cual no se perdió ningún paquete 3.16. Con esta velocidad la imagen de cámara no tiene ningún retraso aparente, lo cual resulta especialmente beneficioso para luego aplicar algoritmos de reconocimiento de objetos en tiempo real. Esto es así ya que por capacidad de procesamiento, el reconocimiento de objetos introduce un retraso de varios segundos en el sistema de visión. Si existiese un retraso de por sí en la transmisión, la utilidad de que el reconocimiento fuese en tiempo de ejecución sería nula, pues sería tan elevado el retraso, del orden de 10 segundos, que el operador no podría tomar una acción de control adecuada.

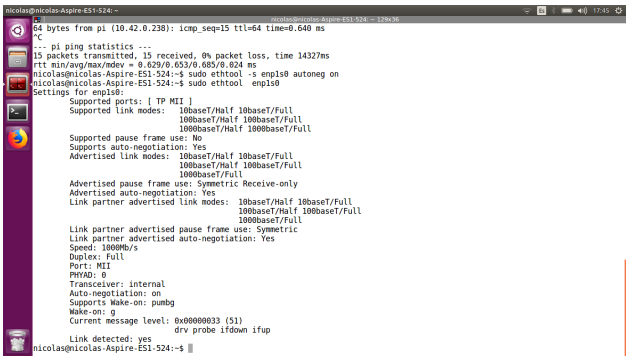



Figura 3.15: Conexión establecida con autonegociado y un cable ethernet de 20 metros.

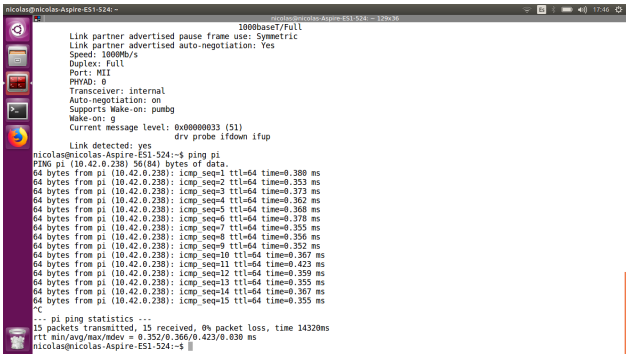



Figura 3.16: Verificación del enlace con 20 metros de cable y una velocidad de 1Gbps Full Dúplex

Por este motivo, para aplicaciones en las que se vayan a realizar operaciones de reconocimiento de imagen es necesario reducir la longitud del cable al menos hasta 55 metros, donde la velocidad de conexión es de 100Mbps Full Dúplex 3.17, siendo ésta suficiente para que el retraso del reconocimiento de imágenes sea aceptable. En dicha configuración también se llevó a cabo una verificación del enlace en el que el porcentaje de paquetes perdidos fue nulo 3.18.

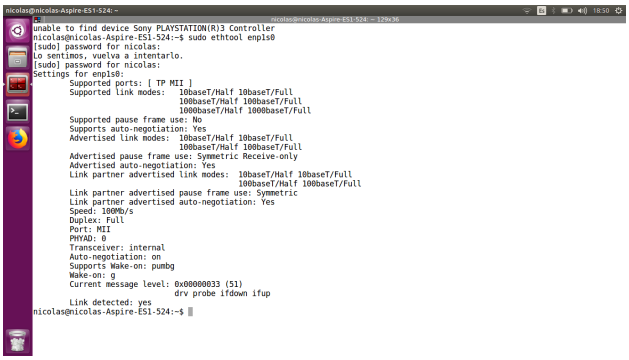



Figura 3.17: Conexión establecida con autonegociado y un cable ethernet de 55 metros.

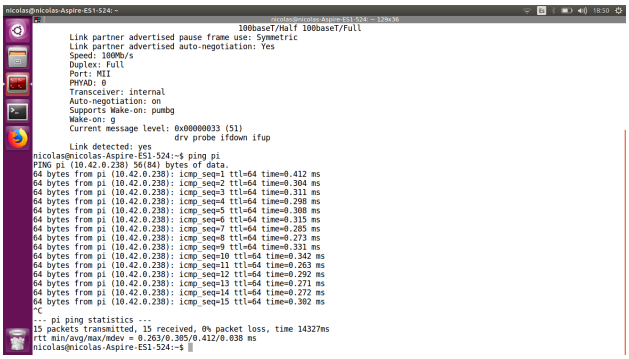



Figura 3.18: Verificación del enlace con 55 metros de cable y una velocidad de 100Mbps Full Dúplex

Para distancia mayores a 55 metros, la conexión se establece en 10Mbps Full Dúplex. A modo de demostración se realizó una prueba con un cable de distancia intermedia, de 65 metros. En dicha prueba 3.19, se consiguieron unos resultados similares a los obtenidos para un cable de 75 metros. Analizando los tiempos de envío de paquetes, empleando el comando *ping* 3.20, se puede observar que el recorte en 10 metros de cable supuso una reducción del tiempo de envío medio en 0.025 ms.

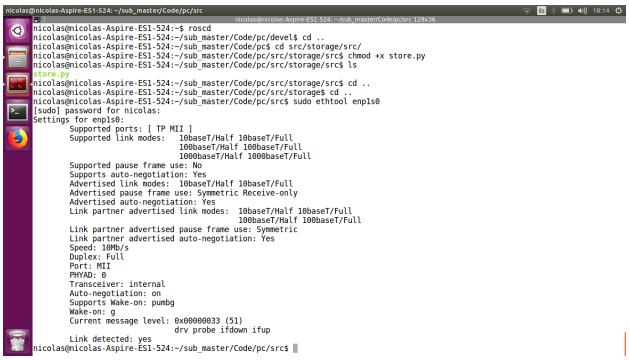



Figura 3.19: Conexión establecida con autonegociado y un cable ethernet de 55 metros.

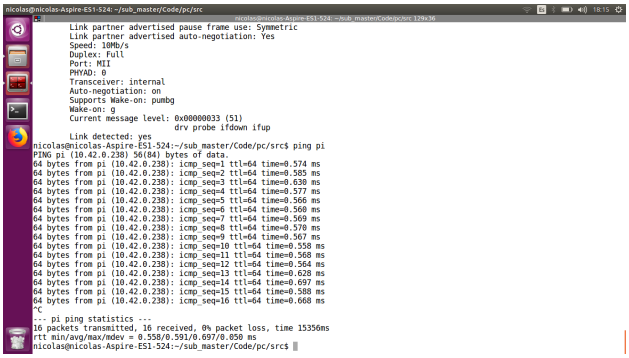



Figura 3.20: Verificación del enlace con 55 metros de cable y una velocidad de 100Mbps Full Dúplex

# Capítulo 4

# Sistema de control

En este capítulo se desarrollará un análisis detallado del comportamiento del dron submarino dentro del medio. Asimismo se realizará un contraste de diferentes estrategias de control, incluyendo los resultados de su implementación.

## 4.1. Comportamiento teórico del dron

### 4.1.1. Ángulos de Tait-Bryan

En el ámbito de la robótica móvil, se conoce como pose de un robot al conjunto de la posición y orientación del robot bajo análisis. Mientras que la posición viene definida por el desplazamiento del robot respecto del origen de coordenadas, empleando tres coordenadas (x,y,z); la rotación viene definida por los ángulos de Tait-Bryan (cabeceo, alabeo, guiñada) 4.1.

En lo referente a la operación del dron, la orientación del submarino resulta clave para poder emplearlo con libertad absoluta. Es por este motivo que toda la estrategia de control se basará en que el operador indique con el mando de videoconsola la velocidad angular sobre cada uno de los ejes del submarino, si estamos en el modo manual; o el ángulo absoluto que se quiere alcanzar si el controlador se encuentra activo.

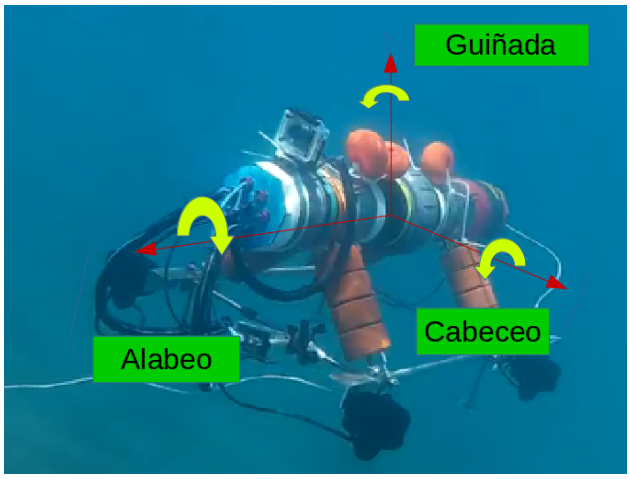



Figura 4.1: Ángulos de Tait Bryan situados sobre el dron submarino

El nombre de los ángulos de Tait-Bryan varía en función del ámbito en el que se emplea. El trío cabeceo, alabeo y guiñada -en inglés *pitch*, *roll* y *yaw*- es ampliamente utilizado en el ámbito de la navegación aérea, incluyendo aviones y drones aéreos. En cambio, en el sector náutico se les conoce también como inclinación, escora y deriva respectivamente.

A pesar de tratarse de un dron submarino, en el TFG se hablará exclusivamente empleando la terminología de aeroplanos, preferentemente los términos en inglés, ya que el esquema de control se basa en el de un *quadrotor*.

### 4.1.2. Comportamiento de un quadrotor

Los *quadrotors* son un tipo de drones aéreos pertenecientes a la categoría de helicópteros que poseen 4 motores. Una configuración típica de *quadrotor* puede ser vista en la figura 4.2.

Figura 4.2: Ejemplo de un quadrotor

Los *quadrotors* se basan en el uso de 4 motores orientados perpendicularmente al suelo para ascender y descender. Dichos motores poseen unas hélices que generan un fuerza de avance perpendicular, conocida como fuerza de *Thrust* o de propulsión, que se relaciona de forma cuadrática con la velocidad de rotación de la hélice, y cuyo sentido dependerá del sentido en que gire la hélice. Además de la fuerza de *Thrust*, cada propulsor tiene asociado un momento de giro que depende igualmente del sentido de giro de cada hélice 4.3.

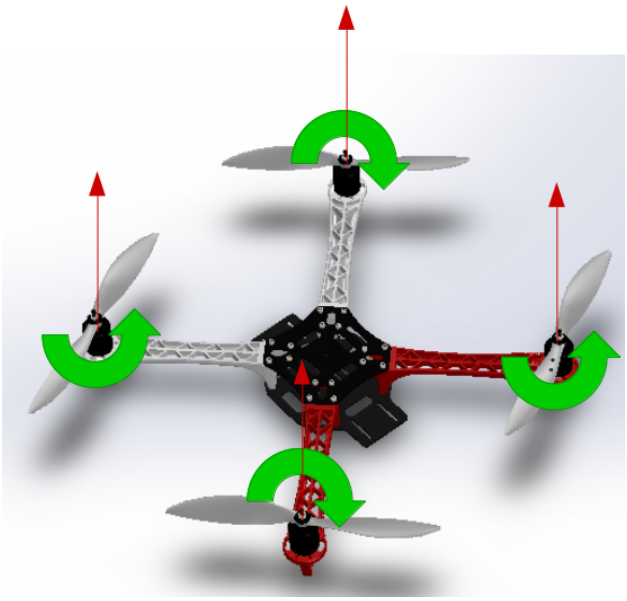


Figura 4.3: Fuerzas de Thrust y momentos de giro en un quadrotos común

En la figura 4.3 se puede observar que existen dos tipos de hélices, ya que para generar una fuerza neta de ascensión un par de hélices gira en sentido antihorario y el otro par en sentido horario. La razón detrás del empleo de dos tipos de hélices radica en el equilibrio de momentos en el dron. Las hélices CW- *clockwise*- generan una fuerza de *thrust* ascendente cuando giran en sentido horario, mientras que las hélices CCW- *counterclockwise*- hacen lo propio cuando giran en sentido antihorario.

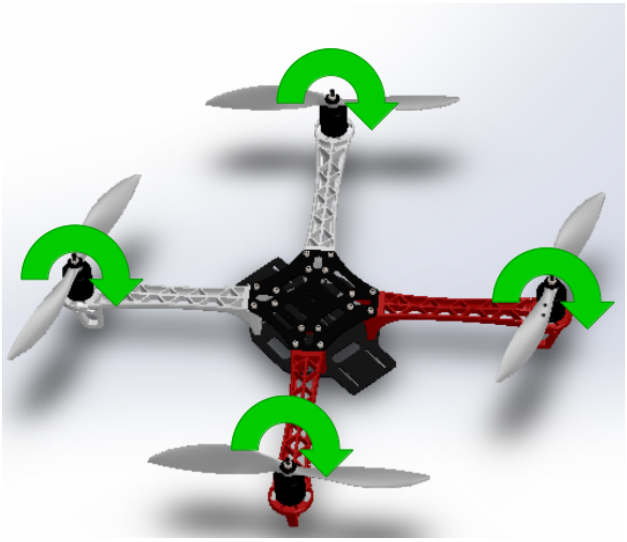


Figura 4.4: Quadrotor con los momentos descompensados

Si todas la hélices girasen en el mismo sentido, se crearía un momento neto que haría rotar al dron 4.4. Dicha rotación afectaría al dron en cualquier momento en el que las fuerzas apuntasen en el mismo sentido, y se anularía cuando las fuerzas se anulan entre ellas, apuntando en sentidos contrarios.

Los movimientos de un quadrotor se basan en la combinación de los momentos y fuerzas de las 4 hélices. Así, por ejemplo, una rotación positiva en *pitch* se realiza aumentando la fuerza de *thrust* de la hélice trasera manteniendo la fuerza de *thrust* de la hélice delantera 4.5. Las hélices giran de dos formas distintas, tal y como se ha explicado anteriormente, para evitar crear un efecto de rotación indeseado.

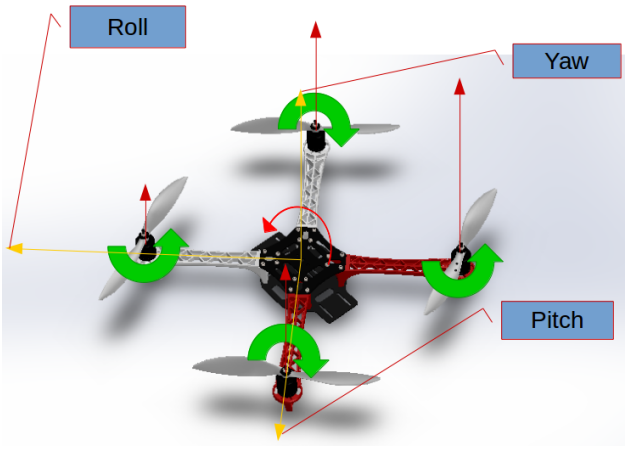



Figura 4.5: Pitch en un quadrotor

Las rotaciones en el eje *Roll* y *Yaw* se realizan de forma similar al ejemplo anterior. Asimismo, los movimientos de ascenso y descenso se realizan siguiendo un esquema con todos los momentos equilibrados y las fuerzas siempre en sentido ascendente, tal y como se muestra en la figura 4.3. La diferencia entre ascender y descender radica en la magnitud de la fuerza *thrust* resultante. Si dicha magnitud es mayor al peso del dron entonces el dron ascenderá, mientras que cuando la fuerza es menor la gravedad se impondrá y el dron descenderá.

Finalmente, el movimiento de avance se realiza aplicando un equilibrio de fuerzas. En la figura 4.6 se puede observar cómo es el esquema de fuerzas de un *quadrotor* cuando se inclina un determinado ángulo. Cuando la fuerza *thrust* resultante posee un ángulo de inclinación respecto del suelo, la fuerza se descompondrá en dos fuerza: una fuerza vertical y otro horizontal. La clave de que los *quadrotors* puedan avanzar está en el adecuado equilibrio entre la fuerza *thrust* vertical y el peso del dispositivo. Si el sumatorio de fuerzas en el eje vertical es nulo, entonces la fuerza *thrust* provocará una fuerza neta horizontal que provocará un desplazamiento horizontal puro del dron.

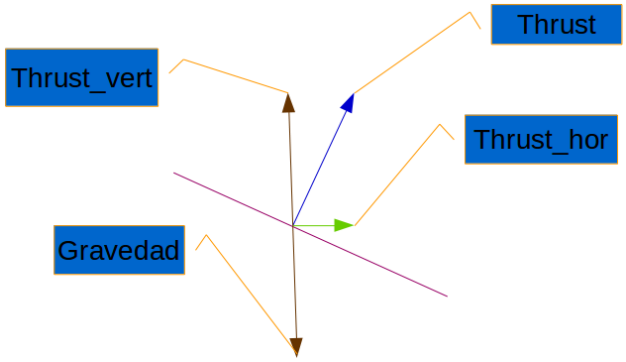



Figura 4.6: Esquema de fuerzas de un avance puro en un quadrotor

### 4.1.3. Diferencia entre medios: aire y agua

El dron submarino innova respecto a las configuraciones de ROV's habituales, en tanto en cuanto la disposición de los motores persigue simular el comportamiento de un *quadrotor* dentro de agua salada. Sin embargo las diferencias entre aire y agua como fluidos son notables y bien conocidas.

Dentro del medio acuoso aparece una nueva fuerza que deberá ser tenida en cuenta en las diferentes estrategias de control: el empuje. Esta fuerza es una fuerza de ascenso, opuesta a la gravedad, que viene definida en el Principio de Arquímedes y depende de la densidad y volumen de fluido desalojado.

En robótica submarina, se suele perseguir que el empuje y el peso del ROV estén equilibrados, buscando la flotabilidad neutra. Este fenómeno implica que el

dron se quede de forma estable a una determinada profundidad, siempre y cuando la densidad a esa profundidad sea la que equilibre el empuje con el peso. Las ventajas de esta configuración radican en el ahorro energético y la estabilidad natural desde el punto de vista del mantenimiento de la profundidad. Sin embargo, para que un ROV pueda tener todos los grados de libertad en estas circunstancias, necesita como mínimo 6 propulsores, tal y como se desarrollará en el apartado 4.2.1.

Por otro lado, existe otra técnica en la que se busca un desequilibrio entre empuje y peso, provocando que de forma natural el submarino se hunda. Normalmente, este caso se suele implementar en dispositivos sin capacidad para moverse y que deben poder sumergirse a varias cotas. Un ejemplo, sería cuando se lanza un hidrófono atado a un cabo. El hidrófono ve descompensada de forma intencionada su peso en relación al empuje para que se hunda hasta la profundidad deseada, que se encuentra limitada por el cabo. En robótica móvil, esta técnica no se suele emplear debido a que los motores deberían moverse de forma continua, con el consumo energético que ello lleva asociado. A pesar de esto, la estrategia de control que se ha implementado en el dron submarino necesita de una gravedad reducida, es decir, un desequilibrio entre empuje y peso a favor del peso. Las razones de esta decisión se detallan en el apartado 4.2.2, pero gracias a ella es posible emplear un ROV con 4 motores con todos los grados de libertad disponibles.

## 4.2. Estrategia de control

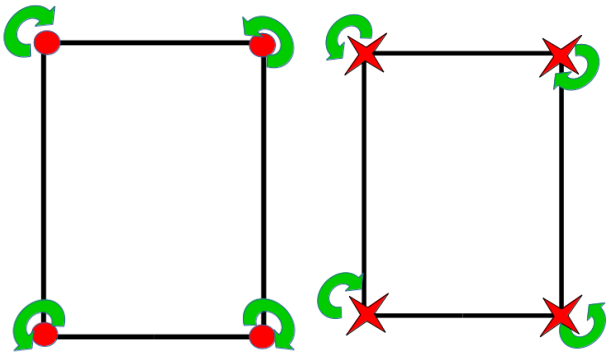
Las estrategias de control definen la aproximación a los diferentes tipos de movimientos que el ROV debe de ser capaz de realizar para que goce de suficiente s grados de libertad. Los diferentes tipos de movimientos que debe ser capaz de realizar son: ascenso, descenso, pitch, roll, yaw y avance horizontal.

### 4.2.1. Estrategia de control con flotabilidad neutra

Tal y como se ha explicado en el apartado 4.1.3,la flotabilidad neutra consiste en equilibrar la fuerza de empuje y el peso del ROV de forma que se quede de forma estable para una condiciones de densidad adecuadas. Se debe tener en cuenta que la densidad del agua de mar aumenta con la profundidad, aunque este efecto no es notable en un rango de profundidades reducido como el que se maneja en este proyecto.

La ventaja principal de emplear un ROV con flotabilidad neutra es el ahorro de energía. Debido a que el dron sería un cuerpo con sus fuerzas en equilibrio, solo sería necesario invertir energía para realizar maniobras, incluyendo dentro de estas maniobras el descenso del dron.

En estas condiciones, el ROV necesita emplear los propulsores para ascender y descender. Para hacer un movimiento de ascenso y descenso puro, los momentos de los 4 propulsores deben estar equilibrados, para lo cual se han empleado dos hélices CW y otras dos hélices CCW. Además, las 4 fuerzas *thrust* deben apuntar en el mismo sentido, para poder lograr un movimiento vertical. En la figura 4.7 se puede comprobar en el diagrama a) que todos los propulsores tienen sus fuerzas apuntando en sentido ascendente, mientras que los momentos de las diagonales se equilibran. En la misma línea en el diagrama 4.7 b) se puede observar la situación recíproca del caso del descenso. Si se comparan ambas imágenes, se puede observar que al invertir el sentido de las fuerzas todos los propulsores ven alterado su sentido de giro, pero se sigue manteniendo el equilibrio entre los momentos. Para realizar un giro en el eje de pitch existen dos alternativas. La primera em-

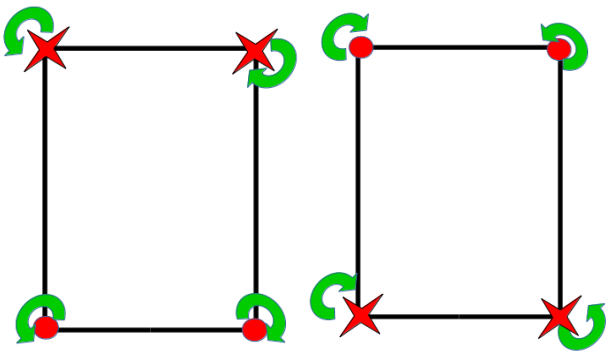


(a) Ascenso (b) Descenso

Figura 4.7: Movimientos verticales puros

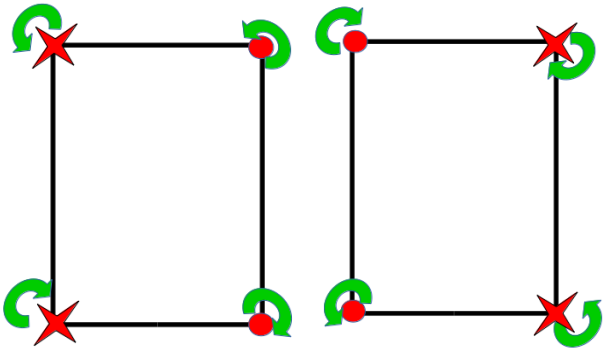
plea los 4 propulsores simultáneamente, haciendo que dos propulsores paralelos al eje de pitch ejerzan su fuerza *thrust* en un sentido y los otros dos en el sentido opuesto, consiguiendo un giro muy rápido. La otra opción emplea solamente dos motores paralelos al eje de pitch, en sentido ascendente o descendente en función del sentido que se desee.

En la figura 4.8 se puede ver el modelo de pitch frontal y trasero de la primera opción, que emplea los 4 propulsores disponibles. Debido a que la neutralidad del ROV ya supone un gran ahorro de energía, en los movimientos de rotación se ha priorizado la maniobrabilidad sobre el ahorro energético con la finalidad de que el operador tenga una respuesta más fluida. Por su parte, la rotación sobre el eje de roll también se implementa usando dos propulsores paralelos a dicho eje en un sentido y los otros dos en el contrario. En esta rotación también se puede aplicar un ahorro de energía, pero siguiendo el criterio establecido para el movimiento de pitch se han empleado los 4 propulsores. El último giro, sobre el eje



(a) Front Pitch (b) Back Pitch

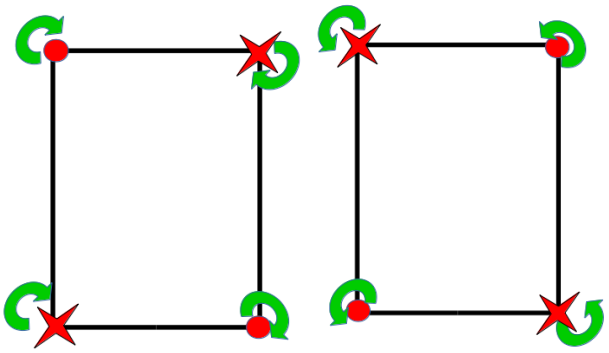
Figura 4.8: Rotaciones sobre el eje de Pitch



(a) Left Roll (b) Right Roll

Figura 4.9: Rotaciones sobre el eje de Roll

yaw, se caracteriza porque es necesario equilibrar las fuerzas de *thrust* mientras se desequilibra los momentos de giro del ROV. Por este motivo, las dos diagonales tendrán sentidos de fuerza distintas y momentos idénticos. En la figura 4.10,se puede observar el esquema de fuerzas y momentos del giro horario y antihorario en Yaw. Por último, para completar todos los grados de libertad faltaría por mo-



(a) CW Yaw (b) CCW Yaw

Figura 4.10: Rotaciones sobre el eje de Yaw

delar el movimiento de avance horizontal. Sin embargo, al estar el ROV en unas condiciones de flotabilidad neutra no existe ninguna gravedad que pueda equilibrar la componente vertical de la fuerza *thrust* como ocurría en la figura 4.6. Esta nueva situación se detalla en la figura 4.11, donde se puede ver claramente que no existe ninguna fuerza vertical que pueda equilibrar a la fuerza de *thrust*.

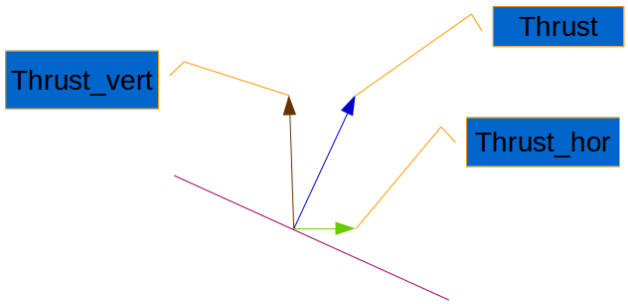



Figura 4.11: Esquema de fuerzas de un avance puro en un ROV con flotabilidad neutra y 4 propulsores

Por tanto, para lograr que el dron avance horizontalmente sólo sería posible avanzar con el dron girado 90º sobre el eje de pitch desde su posición de reposo. Esta posición es muy inestable, ya que el dron ha sido diseñado de tal forma que el metacentro se encuentre por encima del centro de gravedad, lo que crea una

tendencia natural a volver a un ángulo de $0^o$ en el eje de pitch. En la figura 4.12 se puede observar que con esta configuración sí sería posible que el dron avance horizontalmente.

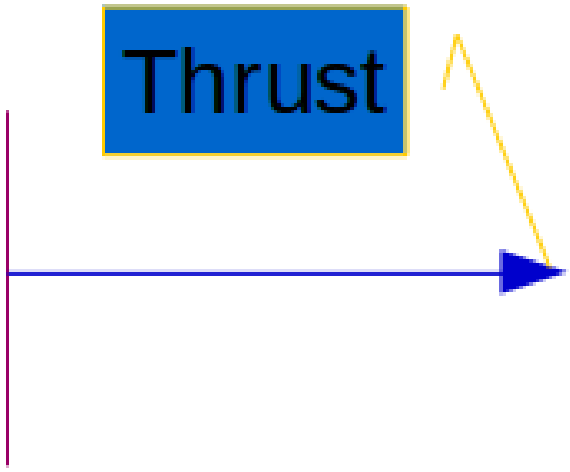



Figura 4.12: Esquema de fuerzas cuando el dron se encuentra girado $90^o$ sobre el eje de Pitch

La necesidad de tener que situar el dron en un ángulo completamente vertical se traduce en problemas de operación y de control. En esa posición, la cámara del submarino no tendría visión de la dirección de avance, por lo que el operador perdería toda referencia, exponiendo al dron a choques fortuitos. Además, la posición de $90^o$ se encuentra en una zona muy inestable por lo que la precisión con la que el operador maneje el ROV incluirá notablemente en la calidad del movimiento.

En conclusión, no resulta una estrategia factible pues el movimiento de avance resultaría tan complejo que su implementación a nivel práctico resulta casi imposible. Esto es coherente con que en el mercado no existan ROV's de 4 propulsores dispuestos como se ha establecido en este proyecto. Sin embargo, existen otras estrategias alternativas que pueden solucionar el problema con el movimiento horizontal puro.

### 4.2.2. Estrategia de control con gravedad reducida

Con la finalidad de hacer frente a los problemas de la estrategia basada en flotabilidad neutra, se diseñó otra estrategia que intenta simular dentro del agua unas condiciones más parecidas a las que enfrenta un *quadrotor* en el aire. Principalmente, se partió de la base de que el peso del submarino superaba al empuje del mismo, lo que creaba una fuerza neta que tiende a hundir el submarino. Esta

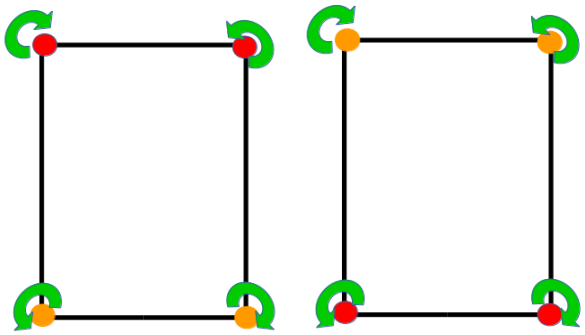
fuerza puede ser comparada con la gravedad en el aire, al menos en lo referente a ser una fuerza vertical que siempre intenta hundir el dron.

La ventaja de emplear una gravedad reducida está en que las condiciones básicas de un *quadrotor* se ven reflejadas para el ROV dentro del agua, lo que habilita un movimiento de avance puro además de el resto de movimientos necesarios para que el operador pueda manejar adecuadamente al dron.

En cambio, la principal desventaja es que los propulsores deben de emplearse constantemente, pues el submarino deberá mantener la profundidad además de realizar las maniobras que le ordene el operador. El único movimiento para el que no tendría que invertir energía es el de descenso, sin que ello suponga un ahorro excesivo en el consumo. Además, el consumo aumentará cuanto mayor sea la diferencia entre peso y empuje. Por este motivo, la diferencia se escogió de forma que la capacidad de la batería ofrezca entre una hora y dos horas de servicio, a pesar del consumo que implica mantener la profundidad del submarino.

El movimiento de ascenso y descenso se implementa de la misma forma que la indicada en la figura 4.7. Las diferencias empiezan con los movimientos de rotaciones.

Para que el dron pueda hacer un giro en torno al eje de pitch, es necesario tener en cuenta que existe una fuerza que tratará de hundir al dron, por lo que los 4 propulsores deberán equilibrar dicha fuerza además de crear una diferencia que les permita girar. Esto se logra replicando el esquema de ascenso del ROV, realizando los mismos momentos y aumentando la fuerza *thrust* de una pareja de propulsores que se encuentre paralela al eje pitch. Este esquema de fuerzas puede verse en la figura 4.13, siendo las fuerzas indicadas en amarillo mayores que las indicadas en rojo. En el caso de una rotación sobre el eje de roll, se emplea un estrategia

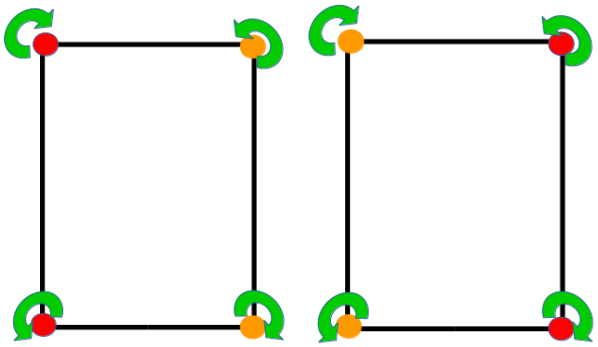


(a) Front Pitch (b) Back pitch

Figura 4.13: Rotaciones sobre el eje de Pitch

similar a la rotación sobre el eje de pitch. Las 4 fuerzas de *thrust* apuntarán en

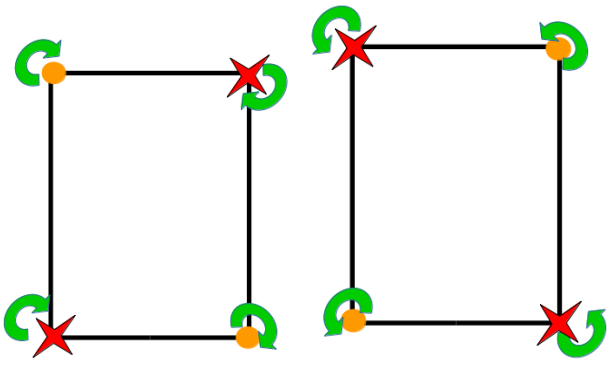
sentido ascendente, pero las fuerzas correspondientes a una pareja de propulsores paralela al eje de roll aumentarán para provocar un giro además de sustentar al submarino 4.14. En lo que respecta al giro sobre el eje de yaw, se deberá emplear



(a) Left Roll (b) Right Roll

Figura 4.14: Rotaciones sobre el eje de Roll

una aproximación al problema distinta a las otras dos rotaciones. El movimiento sobre el eje de yaw se caracteriza por tener todas las fuerzas equilibradas mientras que los momentos de los propulsores se suman, generando un momento de giro neto. Esta definición debe mantenerse, por lo que necesariamente los propulsores de las diagonales deberán tener fuerzas *thrust* de sentido contrario. Para lograr equilibrar la gravedad reducida, se le dará mayor potencia a la diagonal cuya fuerza *thrust* tenga sentido ascendente. Este comportamiento se ha representado en la figura 4.15 indicando en color amarillo las fuerzas *thrust* de mayor magnitud para cada caso. Finalmente, para lograr un movimiento de avance horizontal puro nos



(a) CW Yaw (b) CCW Yaw

Figura 4.15: Rotaciones sobre el eje de Yaw

encontraríamos en la misma situación que la de la figura 4.6. En la figura 4.16 se puede ver una imagen del ROV avanzando debajo del agua con el esquema de fuerzas correspondiente.

La fuerza de avance del dron dependerá del ángulo de inclinación $\alpha$ sobre el eje de pitch. La dependencia de las fuerzas verticales y horizontales se expresa en la ecuación 4.1 y 4.2 respectivamente.

$$Thrust_V = Thrust * \cos(\alpha) \tag{4.1}$$

$$Thrust_H = Thrust * \sin(\alpha) \tag{4.2}$$

A raíz de las ecuaciones 4.1 y 4.2 se desprende que cuando aumenta el ángulo $\alpha$ la componente vertical de la fuerza *thrust* se reduce en favor de la componente horizontal. Esto significa que cuanto mayor sea la inclinación, el dron se moverá a mayor velocidad horizontal, pero también será necesario aumentar el módulo de la fuerza *thrust* para que la componente vertical $Thrust_V$ se pueda igualar con el peso.

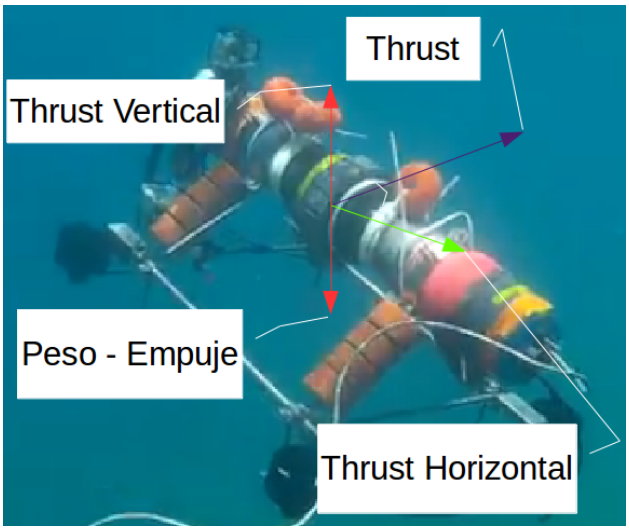



Figura 4.16: Esquema de fuerzas cuando el dron avanza con una inclinación $\alpha$

### 4.2.3. Eliminación del Roll

Las diferentes estrategias de control que se han explicado hasta ahora incluyen la rotación sobre el eje roll como una de las condiciones para considerar que el sistema posee suficientes grados de libertad como para que el operador pueda manejar correctamente el ROV.

Habitualmente, tanto los *quadrotors* como los ROV de gama media y baja deshabilitan el movimiento de roll en el software de control. Esta restricción se

debe a que para un operador *amateur* el control manual de un dron resulta muy complejo, necesitándose entrenamiento y experiencia para lograr manejar un ROV de forma segura y adecuada.

La inclusión del movimiento de roll lleva aparejada la habilitación de los 4 ejes de los dos *joysticks* empleados 4.17. Los ROV's son sistema MIMO con interacciones muy fuertes, por lo que las alteraciones del comportamiento al mezclar movimientos de forma accidental son muy notables. Ello se traduce en que un operador sin experiencia apenas consigue desplazar el dron sin realizar de forma frecuente movimientos erráticos. Por este motivo, es habitual que en drones donde no sea imprescindible contar con la rotación respecto de roll se elimine la parte de control asociada a este ángulo.

En el subproyecto, se hicieron pruebas iniciales en las que se comprobó de primera mano la complejidad y precisión requerida para operar un ROV con todos sus grados de libertad, por lo que se modificó el software de control para eliminar el uso de la rotación en roll con la finalidad de que el dron sea accesible para cualquier usuario.

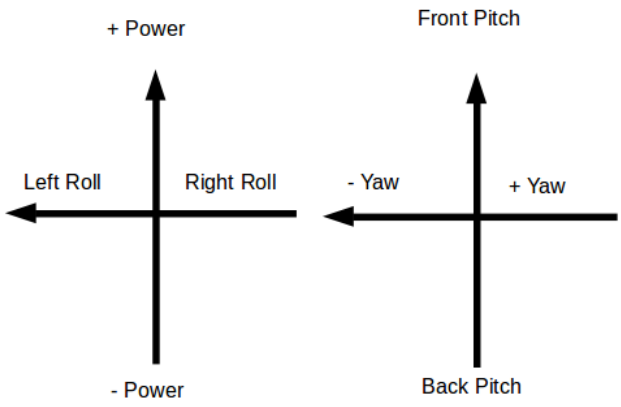



Figura 4.17: Joysticks con los 4 ejes identificados

## 4.3. Movimiento manual

### 4.3.1. Sistema de control manual

Con la finalidad de implementar la estrategia de control basada en la existencia de una gravedad reducida 4.2.2 se ha diseñado un nodo de ROS que traduzca las órdenes de control a comandos que los propulsores puedan interpretar. El nodo en cuestión se llama *joy_publisher* e implementa la lectura del mando de videoconsola que emplea el operador a través de un nodo intermediario de ROS, denominado

*joy*. Tras leer las órdenes del mando, transforma los valores de los joysticks para obtener un vector de 4 componentes que contiene las señales de los propulsores. Además de procesar la señal de los joysticks, procesa los botones de los mandos que tienen algún tipo de funcionalidad para enviar la menor cantidad de datos posibles a la placa *arduMove*. El ciclo que se ha descrito puede ser consultado en la figura *joysticks_nodes*.


Figura 4.18: Esquema de nodos y tópicos involucrados en el sistema de control manual

La clave de la implementación de este nodo se encuentra en la forma en la que se interpretan las señales de control de operador y la forma en que éstas son transformadas. Empezaremos revisando la forma en la que se van a interpretar las órdenes del operador.

En la figura 4.17 se puede ver la distribución que se hace de los distintos movimientos que puede realizar el submarino en el mando de PS3 empleado. Sin embargo, tal y como se desarrolló en el apartado 4.2.3 la posibilidad de girar alrededor del eje de roll ha sido eliminada mediante software para facilitarle al operador el control del ROV. En consecuencia, los ejes de los dos joysticks del mando quedarían como en la figura 4.19.

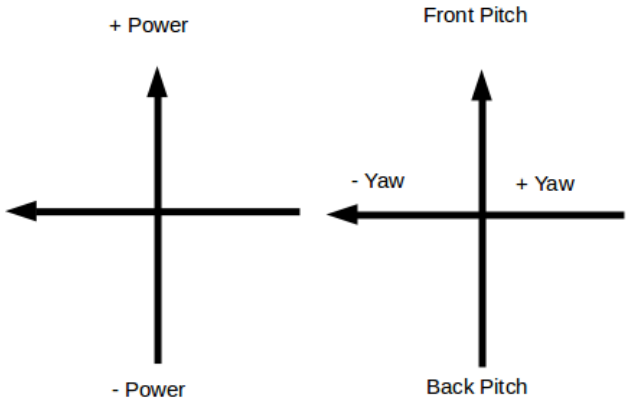



Figura 4.19: Identificación de los ejes de los 2 joysticks del mando de PS3

Los ejes *Power*, *Pitch* y *Yaw* se deben interpretar como aumento de la potencia de cada movimiento asignado, lo que se traduce en un aumento de la fuerza de *thrust* y de la velocidad angular o lineal del ROV según proceda. El eje *Power* se emplea para realizar el movimiento de ascenso, descenso y mantenimiento de la

cota de profundidad. Por esta razón, el valor de potencia que se obtiene de dicho eje se interpreta como una aportación a los 4 propulsores por igual, provocando un cambio simétrica de la potencia en cada uno de ellos.

Por su lado, el eje *Pitch* se emplea para provocar una asimetría entre las parejas de propulsores paralelas al eje de pitch, tal y como se ha explicado anteriormente en la figura 4.13. Con el fin de contrarrestar la gravedad reducida, los 4 propulsores siempre deben generar una fuerza *thrust* de ascenso.

Por otro lado, el eje de *Yaw* genera un desequilibrio de los momentos angulares provocando que todos los motores giren en el mismo sentido mientras existe un desfase de fuerzas *thrust* como el descrito en la figura 4.15. Cuanto mayor sea el valor del eje *Yaw* del mando, mayor será la velocidad de giro de los propulsores, pero la velocidad de giro no se ve alterada en una magnitud comparable a la del giro de *Pitch*. Esto se debe a que en las hélices la fuerza de *thrust* es del orden de 10 veces superior a la fuerza perpendicular.

Por último, el movimiento de avance se consigue mediante la combinación del eje de *Power* y de *Pitch* del mando. Recordemos que para avanzar se debe de generar una situación equivalente a la figura 4.16, en la que se equilibre la fuerza de *thrust* vertical y la gravedad reducida. En la ecuación 4.3 se puede observar la dependencia del ángulo de inclinación $\alpha$ con las fuerzas involucradas en el ROV. Cuando se mezclan ambos ejes, *Power* y *Pitch*, la fuerza de *thrust* aumenta, lo que implica que para obtener un movimiento de avance puro se deba de incrementar el ángulo de inclinación del dron. Como consecuencia del aumento del ángulo de inclinación, y siguiendo la expresión 4.2, la fuerza de avance horizontal aumenta significativamente lo que se traduce en un aumento significativo de la velocidad de avance.

$$\alpha = \arccos\left(\frac{Gravedad}{Thrust}\right) \tag{4.3}$$

De cara a transformar las órdenes del operador, desglosadas en 3 ejes, en un vector de 4 componentes que contenga los comandos de los motores, se ha seguido el planteamiento realizado en la tesis titulada *Estrategias de control para sistemas multivariables no lineales: aplicación a un helicóptero de 4 rotores* [17] del Dr. Jonay Toledo.

La propuesta realizada por el Dr. Jonay Toledo se basa en el empleo de una matriz de transformación que vincule de forma proporcional las órdenes de cada eje del mando de la PS3 con los propulsores del dron. Por ejemplo, en la ecuación 4.4 se cita un ejemplo en la que se tienen en cuenta los 3 ángulos de rotación y un cuarto eje correspondiente a lo que en este trabajo se denomina eje de *Power*.

$$\begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 1 & 1 \\ -1 & 0 & 1 & 1 \\ 0 & -1 & -1 & 1 \\ 0 & 1 & -1 & 1 \end{bmatrix} \begin{bmatrix} Power \\ Pitch \\ Roll \\ Yaw \end{bmatrix} \tag{4.4}$$

Si expresamos la ecuación matricial 4.4 en 4 ecuaciones, una para cada motor, nos quedaría las expresiones 4.5, 4.6, 4.7 y 4.8.

$$M_1 = Power + Roll + Yaw \tag{4.5}$$

$$M_2 = -Power + Roll + Yaw \tag{4.6}$$

$$M_3 = -Pitch - Roll + Yaw \tag{4.7}$$

$$M_4 = Pitch - Roll + Yaw \tag{4.8}$$

Este procedimiento de desacople es perfectamente aplicable al ROV, teniendo en cuenta las peculiaridades de prescindir del giro sobre el eje de roll y de que la disposición de los motores es la que aparece en la figura 4.20.


Figura 4.20: Disposición de los motores en el ROV

Teniendo en cuenta la casuística de nuestro dron, la relación entre motores y ejes del mando de PS3 quedaría como en la ecuación matricial 4.9. Se debe tener en cuenta que para equilibrar los momentos de giro de los diferentes motores la diagonal formada por $M_1$ y $M_4$ tiene montadas una hélices CW -*clockwise*- y la

diagonal opuesta formada por $M_2$ y $M_3$ tiene hélices CCW-*counterclockwise*-.

$$\begin{bmatrix} M_1 \\ M_2 \\ M_3 \\ M_4 \end{bmatrix} = \begin{bmatrix} 1 & -1 & 1 \\ -1 & 1 & 1 \\ -1 & -1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} Power \\ Pitch \\ Yaw \end{bmatrix} \tag{4.9}$$

A partir de ellas se desprende que los valores de los motores quedarían como sigue en la ecuaciones 4.10, 4.11, 4.12 y 4.13.

$$M_1 = Power - Pitch + Yaw \tag{4.10}$$

$$M_2 = -Power + Pitch + Yaw \tag{4.11}$$

$$M_3 = -Power - Pitch + Yaw \tag{4.12}$$

$$M_4 = Power + Pitch + Yaw \tag{4.13}$$

Sobre estas ecuaciones es imprescindible estudiar el rango en el que pueden oscilar los comandos de los motores, ya que en la placa *arduMove* se debe de implementar una conversión entre el rango de los comandos y el rango de PWM de los propulsores. Los ejes del mando de PS3 tiene un rango definido en [-1,1], por lo que si evaluamos las expresiones anteriores se podrá comprobar que el rango máximo para todos los motores se sitúa en [-3,3]. Para simplificar las operaciones que se deben realizar a posteriori, en la propia matriz de desacoplo se introduce un factor $k = \frac{1}{3}$ para normalizar los comando en un rango [-1,1].

Sin embargo, las expresiones anteriores se corresponden con la estrategia de control basada en flotabilidad neutra 4.2.1. Las ecuaciones correspondientes al caso de gravedad ficticia son más complejas pues aparecen expresiones condicionales para diferenciar los sentidos de giro. En las ecuaciones 4.14, 4.15, 4.16 y 4.17 se pueden ver las expresiones de cada motor para el caso de gravedad ficticia. El principio implementado es idéntico al ya explicado anteriormente.

$$\begin{aligned} M_1 = {} & Power + (0{,}6 * Pitch) * (Pitch \geq 0) + (abs(Pitch)) * (Pitch < 0) \\ & + (Yaw) * (Yaw \geq 0) + (0{,}85 * Yaw) * (Yaw < 0) \end{aligned} \tag{4.14}$$

$$\begin{aligned} M_2 = {} & -Power - (0{,}6 * Pitch) * (Pitch \geq 0) - (abs(Pitch)) * (Pitch < 0) \\ & + (0{,}85 * Yaw) * (Yaw \geq 0) + (Yaw) * (Yaw \geq 0) \end{aligned} \tag{4.15}$$

$$\begin{aligned} M_3 = {} & -Power - (Pitch) * (Pitch \geq 0) - (0{,}6 * abs(Pitch)) * (Pitch < 0) \\ & + (0{,}85 * Yaw) * (Yaw \geq 0) + (Yaw) * (Yaw \geq 0) \end{aligned} \tag{4.16}$$

$$\begin{aligned} M_4 = {} & Power + (Pitch) * (Pitch \geq 0) + (0{,}6 * abs(Pitch)) * (Pitch < 0) \\ & + (Yaw) * (Yaw \geq 0) + (0{,}85 * Yaw) * (Yaw < 0) \end{aligned} \tag{4.17}$$

Como se puede observar, los movimientos de rotación llevan asociados unos coeficiente de reducción sobre la orden del operador para provocar el giro, en el caso del giro sobre pitch; o para generar sustentación, es decir una fuerza de ascenso que compense la gravedad reducida mientras se gira, como es en el caso del giro sobre yaw.

En el giro de pitch es necesario provocar un desequilibrio entre las fuerzas de *thrust* de los motores $M_1$ y $M_2$ y la pareja $M_3$ y $M_4$ para que se incline el dron. Si se quiere realizar un movimiento de cabeceo frontal, la pareja $M_{34}$ deberá tener una fuerza de ascenso superior a la pareja $M_{12}$, por esta razón se le aplica a esta última pareja un coeficiente reductor sobre la instrucción de control. En el caso de cabeceo inverso, la relación de fuerzas debe ser la opuesta, es decir, la pareja $M_{12}$ debe tener una fuerza de ascenso superior a la pareja $M_{34}$. Sin embargo, a pesar de que la señal del operador se encuentre en un rango negativo, el movimiento debe mantenerse en sentido ascendente, para lo cual es necesario aplicar el valor absoluto a la señal de control de pitch.

Por su parte, el giro sobre yaw necesita que todos los propulsores giren en el mismo sentido para provocar un giro alrededor del eje yaw. Debido a que las hélices se encuentran invertidas en la diagonal $M_{23}$, cuando todos los propulsores giran en el mismo sentido las fuerzas *thrust* de cada propulsor se contrarresta, dejando como única fuerza neta a la gravedad reducida. Por este motivo, con independencia del sentido de giro se introduce un factor de reducción a la diagonal de motores cuya fuerza vaya en favor de la gravedad, que se corresponde con la diagonal $M_{23}$ en sentido horario y con la diagonal $M_{14}$ en sentido antihorario. En este caso, cuando la orden del operador se invierte sí se mantiene el sentido para que las hélices alteren su sentido de giro, al contrario de lo que sucedía con el giro respecto de pitch.

### 4.3.2. Implementación

La implementación de la matriz de desacoplo se ha desarrollado mediante la aplicación de las ecuaciones 4.14, 4.15, 4.16 y 4.17. Cuando el nodo *joy* publica un mensaje, el maestro notifica dicho mensaje al nodo *joy_publisher*. A partir del mensaje publicado, el nodo obtiene las componentes de los 4 ejes de los joysticks y procede a calcular el valor de los comandos de los motores empleando las ecuaciones referenciadas anteriormente. Este proceso es realizado en una función denominada *motores* que es llamada desde el *callback* correspondiente. Las variables *joysticks[1]*, *joysticks[2]* y *joysticks[3]* se corresponden con las órdenes del operador en los ejes *Power*, *Yaw* y *Pitch* respectivamente.

```
1 std::vector<float> motores(std::vector<float> joysticks){
2   std::vector<float> motors;
3   float m1, m2, m3, m4 = 0;
```

```
4   if((joysticks[2] < sensibilidadYaw) && (joysticks[2] > (-
      sensibilidadYaw))){
5     // cuando no hay movimiento de Yaw
6     if(joysticks[3] > 0){
7       // con m2 y m3 invertidos: Power - Pitch
8       // Con Pitch positivo
9       m1 = ((joysticks[1]/2) + (joysticks[3]/2)*(desfasePitch))
      * (-1);
10      m2 = ((-joysticks[1]/2) - (joysticks[3]/2)*(desfasePitch))
       * (-1);
11      m3 = ((-joysticks[1]/2) - (joysticks[3]/2)) * (-1);
12      m4 = ((joysticks[1]/2) + (joysticks[3]/2)) * (-1);
13    }else{
14      // con m2 y m3 invertidos: Power - Pitch
15      // Con Pitch negativo, se multiplica la componente Pitch
      por -1 para compensar el signo
16      m1 = ((joysticks[1]/2) - (joysticks[3]/2)) * (-1);
17      m2 = ((-joysticks[1]/2) + (joysticks[3]/2)) * (-1);
18      m3 = ((-joysticks[1]/2) + (joysticks[3]/2)*(desfasePitch))
       * (-1);
19      m4 = ((joysticks[1]/2) - (joysticks[3]/2)*(desfasePitch))
      * (-1);
20    }
21  }else{
22    // en el caso de que exista Yaw
23    // con m2 y m3 invertidos: joysticks[1] = Ganancia/Power,
      joysticks[2] = Yaw
24    if(joysticks[2] > 0){
25      m2 = (joysticks[2] / 2) * (1 + joysticks[1]);
26      m3 = m2*0.85;
27      m1 = reduccionPot* m2;
28      m4 = m1;
29    }else{
30      m1 = (joysticks[2] / 2) * (1 + joysticks[1]);
31      m4 = m1;
32      m2 = reduccionPot* m1;
33      m3 = m2*0.85;
34    }
35  }
36  motors.push_back(m1);
37  motors.push_back(m2);
38  motors.push_back(m3);
39  motors.push_back(m4);
40  return motors;
41 }
```

Listing 4.1: Función del nodo joy_publisher que transforma las órdenes del operador en comandos para los propulsores

En el código 4.1, se han añadido varias modificaciones respecto de las ecuaciones teóricas establecidas en el punto 4.3.1 para mejorar el comportamiento en una situación real. Dichas modificaciones se realizaron de forma empírica tras detectar asimetrías entre los propulsores empleados, así como para facilitar la labor de control al operador del dron.

Un de estas modificaciones consiste en la introducción de un rango de sensibilidad en el eje yaw que se sitúa en [-0.05,0.05]. Dentro de ese rango se entenderá que el operador no tiene la intención de realizar un giro sobre yaw sino que es un desvío involuntario respecto de un giro de pitch o una señal de parada. Esto intenta compensar que los joysticks no poseen carriles que limiten el movimiento de los ejes, lo cual complica la realización de un movimiento en un sólo eje.

Por otro lado, las ecuaciones de movimiento han sido modificadas para atender al estado de los propulsores. Los 4 han sido instalados con la fuerza *thrust* invertida, es decir, cuando giran en su sentido natural ejercen una fuerza que intentar hundir al dron submarino. Si se tiene en cuenta esta circunstancia, las ecuaciones 4.14, 4.15, 4.16 y 4.17 se modifican hasta obtener las ecuaciones 4.18, 4.19, 4.20 y 4.21. Aparte de tener en cuenta la inversión de los propulsores, se optó por eliminar la posibilidad de realizar un giro combinado en pitch y yaw. Tras realizar las primeras pruebas con el dron se encontró que en muchas ocasiones las interacciones entre los elementos de la ecuación de cada motor provocaban que el operador perdiera fácilmente el control del dron. Por este motivo, se decidió modificar las ecuaciones para independizar los giros y que el control manual disminuyera su dificultad, con la finalidad de que la formación necesaria para operar el ROV se minimizara.

$$
\begin{aligned}
M_1 = {} & [[(\frac{Power + Pitch * 0{,}6}{2}) * (-1)] * (Pitch > 0) + \\
& + [(\frac{Power - Pitch}{2}) * (-1)] * (Pitch \leq 0)] * (-0{,}05 < Yaw < 0{,}05) + \\
& + [(\frac{Yaw + Yaw * Power}{2}) * Red] * (Yaw > 0{,}05) + \\
& + [(\frac{Yaw + Yaw * Power}{2})] * (Yaw < -0{,}05)
\end{aligned}
\tag{4.18}
$$

$$M_2 = [[(\frac{-Power - Pitch * 0{,}6}{2}) * (-1)] * (Pitch > 0) + \\ + [(\frac{-Power + Pitch}{2}) * (-1)] * (Pitch \leq 0)] * (-0{,}05 < Yaw < 0{,}05) + \\ + [(\frac{Yaw + Yaw * Power}{2})] * (Yaw > 0{,}05) + \\ + [(\frac{Yaw + Yaw * Power}{2}) * Red] * (Yaw < -0{,}05) \tag{4.19}$$

$$M_3 = [[(\frac{-Power - Pitch}{2}) * (-1)] * (Pitch > 0) + \\ + [(\frac{-Power + Pitch * 0{,}6}{2}) * (-1)] * (Pitch \leq 0)] * (-0{,}05 < Yaw < 0{,}05) + \\ + [(\frac{Yaw + Yaw * Power}{2}) * 0{,}85] * (Yaw > 0{,}05) + \\ + [(\frac{Yaw + Yaw * Power}{2}) * Red * 0{,}85] * (Yaw < -0{,}05) \tag{4.20}$$

$$M_4 = [[(\frac{Power + Pitch}{2}) * (-1)] * (Pitch > 0) + \\ + [(\frac{Power - Pitch * 0{,}6}{2}) * (-1)] * (Pitch \leq 0)] * (-0{,}05 < Yaw < 0{,}05) + \\ + [(\frac{Yaw + Yaw * Power}{2}) * Red] * (Yaw > 0{,}05) + \\ + [(\frac{Yaw + Yaw * Power}{2})] * (Yaw < -0{,}05) \tag{4.21}$$

En todas las ecuaciones se puede observar cómo se han introducido los factores de reducción comentados en el apartado 4.3.1, con un valor de 0.6 como desfase entre fuerzas en el giro de pitch y como un factor de 0.65 en el caso de yaw, que en las ecuaciones anteriores se encuentra parametrizado por el parámetro *Red*.

Asimismo, se ha introducido en el movimiento de Yaw un nuevo parámetro en el motor $M_3$ con un valor de 0.85. Este factor pretende corregir el comportamiento de dicho propulsor, que posee mayor fuerza *thrust* que el propulsor $M_2$ situado en su misma diagonal. Este defecto se detectó de forma experimental, en una de las pruebas realizadas en la playa de Las Teresitas, en Santa Cruz de Tenerife. Su ajuste, por tanto, también se corresponde con un ajuste empírico mediante prueba y error. La descompensación del propulsor se percibe con mayor facilidad en el movimiento de yaw debido a que el dron debería mantenerse en el mismo plano, es por este motivo que la corrección de potencia sólo se ha realizado en este giro. El comportamiento en el giro de pitch es aceptable, siendo innecesario añadir un parámetro de corrección extra.

Debido a que en las ecuaciones 4.18, 4.19, 4.20 y 4.21 se ha alterado el número de componentes que contribuyen a los comandos de los motores, reduciéndolo a solo 2 aportaciones como máximo. El rango sin normalizar se ve también alterado respecto de la ecuaciones 4.14, 4.15, 4.16 y 4.17, reduciéndose los límites hasta [-2,2]. Por este motivo, se ha introducido un factor de normalización $k = \frac{1}{2}$ que ya ha sido incorporado en las nuevas expresiones.

Además de la función *motores*, se ha implementado en el *callback* un sistema de control de la frecuencia de envío de los comandos en los motores. Tal y como se comentó en el capítulo 3, la librería de ROS para Arduino sobrecarga en exceso la memoria de los microcontroladores, por lo que se tuvo que establecer una frecuencia de envío que equilibrase una buena respuesta a las órdenes del operador con un buen comportamiento del microcontrolador. Esta frecuencia se estableció en 5 Hz por no ser necesaria una mayor velocidad de respuesta dada la dinámica del ROV. Sin embargo, se han realizado pruebas en las que el rendimiento era aceptable hasta una frecuencia de 20Hz.

Por encima de 20Hz, las comunicaciones con la placa *arduMove* se volvían inestables, produciéndose errores de sincronización de forma ocasional. Además, en el caso extremo donde no se establecía una frecuencia en el envío de los comandos al dron, sino que se enviaban de forma inmediata en cuanto se recibiera una orden distinta por parte del operador, la placa *arduMove* detenía su ejecución a los 14 minutos de ejecución continua. Esto se debía a la saturación del buffer de la placa y fue un fenómeno que se pudo comprobar con exactitud en 3 pruebas realizadas, donde se enviaban datos de forma continua mientras se cronometraba el tiempo de ejecución.

Debido a que se debe garantizar como mínimo que la placa responde a la señal de parada, que se corresponde con la combinación de L1 + X del mando de PS3, se estableció una frecuencia para la cual las comunicaciones fueran muy estables. El código completo puede ser consultado en los anexos del final de la memoria 8.1.

### 4.3.3. Resultados

El sistema de control manual se ha verificado mediante la inmersión del dron submarino en 7 pruebas realizadas a los largo del mes de mayo en la playa de Las Teresitas. Dichas pruebas, con una duración comprendida entre 1 hora y 1 hora y media, han consistido en la inmersión del prototipo en una condiciones marinas controladas, en un ambiente con corrientes bajas y visibilidad adecuada. Además, se han empleado 2 buzos de apoyo para supervisar y grabar al submarino dentro del agua.

Los ajustes empíricos que se han realizado, como el desfase introducido para provocar el giro de Pitch y el desequilibrio de fuerza que mantenga sobre el mismo

plano al giro sobre Yaw, han sido realizado en base a la experiencia obtenida de dichas pruebas. Otras decisiones de calado dentro del subproyecto, como la eliminación del control manual sobre el giro de roll y forzar la independencia de los giros de pitch y yaw también se han realizado como consecuencia de los resultados obtenidos en las pruebas realizadas.

En las pruebas realizadas, se empezó por verificar de forma controlada los diferentes tipos de giros. En la figura 4.21 se puede observar al dron en una prueba realizada el 15 de mayo mientras se encontraba atado a un cabo para recuperarlo en caso de fallo del sistema.

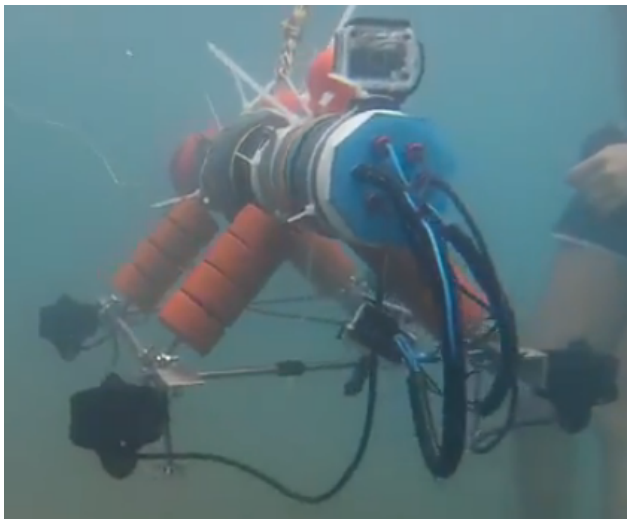


Figura 4.21: Dron atado con un cabo mientras se realizaba una prueba en una zona de profundidad inferior a 1.8 metros

La estación de control se encontraba elevada sobre el nivel del mar en unas escaleras de acceso desde un espigón en la playa 4.22 para que el operador pudiera visualizar correctamente al dron, mientras que en el borde del muelle se situaba otra persona para administrar el cabo y el cordón umbilical 4.23.

Figura 4.22: Vista desde la estación de control del dron mientras se preparaba para la inmersión

Figura 4.23: Técnico administrando el cable mientras un buzo supervisa al submarino

Los resultados obtenidos de la implementación se pueden calificar como satisfactorios. En lo referente al movimiento de ascenso y descenso, el dron se encuentra muy equilibrado con independencia de la potencia de los propulsores. En la figura 4.24 se muestra al ROV mientras asciende desde el fondo marino, situado a una profundidad aproximada de 4 metros.

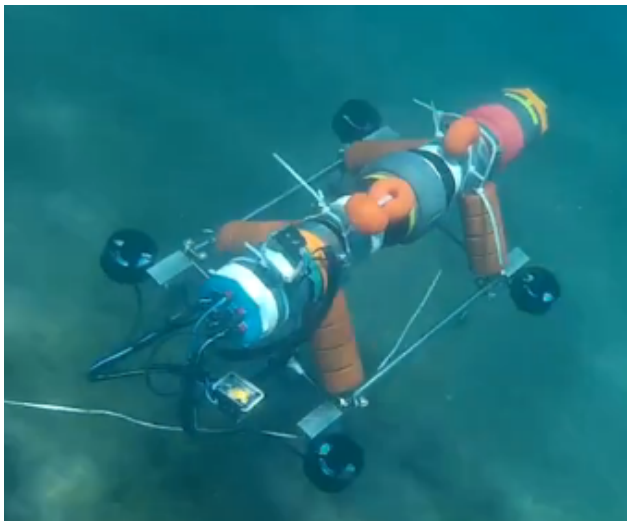


Figura 4.24: Dron ascendiendo desde el fondo marino, situado a 4 metros de profundidad

Por su lado, los giros de pitch 4.25 y yaw 4.26 también se realizan correctamente. El giro de pitch es un movimiento muy rápido, al igual que pasa con el movimiento de ascenso y descenso, debido a que las fuerzas de *thrust* de los propulsores equilibran fácilmente a la gravedad reducida, equivalente a 2.3 kgf según mediciones experimentales en agua salada. Cada propulsor es capaz de generar una fuerza de 5 kgf aproximadamente, según las especificaciones técnicas del fabricante. En el entorno donde se han llevado a cabo las pruebas, la velocidad resulta excesiva, teniendo el operador que tener mucho cuidado a la hora de operar el dron. Sin embargo, esto ha otorgado al submarino de la capacidad de operar bajo circunstancias marinas moderadas, pudiendo operar bajo corrientes moderadas.

En cuanto al giro respecto de yaw, se verificó que es posible realizarlo sin que afecte a la cota de profundidad, es decir, manteniendo al dron en el mismo plano. Asimismo, también se corroboró que el momento resultante del dron no se veía influido notablemente al aumentar la fuerza de *thrust*, lo cual concuerda con el modelo teórico que se ha venido exponiendo en el presente capítulo. Sin embargo, el dron es capaz de generar suficiente momento de giro como para realizar un giro en contra de la marea, lo cual resulta suficiente para operaciones en aguas tranquilas de puerto, aunque no para operaciones en mar abierto.

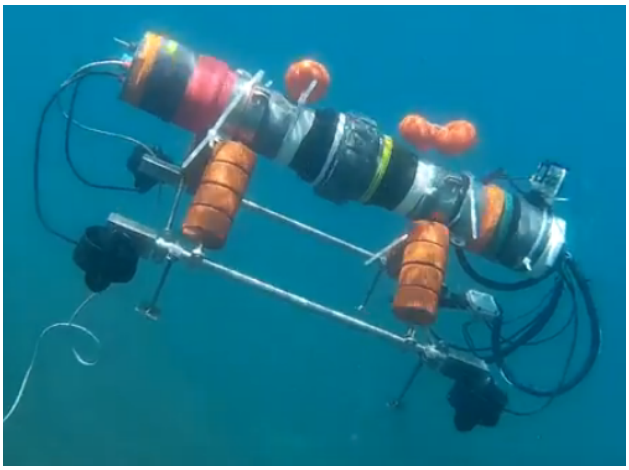


Figura 4.25: Dron realizando un giro sobre el eje de Pitch

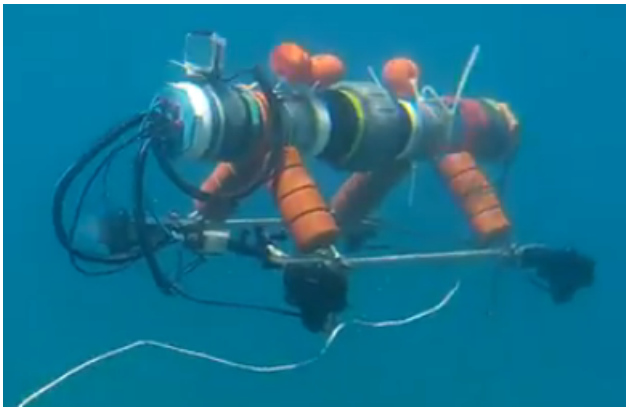


Figura 4.26: Dron realizando un giro sobre el eje de Yaw

Además, también se ha comprobado que el dron es capaz de avanzar debajo del agua 4.27. El movimiento de avance requiere un ángulo de pitch preciso, que depende de la fuerza de *thrust* y de la gravedad reducida según se indica en la ecuación 4.3. Para lograr alcanzar el ángulo adecuado, el operador debe realizar movimientos lentos, que requieren de cierta experiencia manejando el dron. Un controlador que fijase el ángulo de pitch ayudaría en gran medida a realizar esta operación.

Figura 4.27: Dron avanzando debajo del agua

Por último, también se ha comprobado que el dron es capaz de moverse en la superficie del agua 4.28 mientras tiene los propulsores sumergidos. Este movimiento es lento en comparación al resto debido a que con el dron parcialmente sumergido la gravedad reducida aumenta significativamente.

Figura 4.28: Dron parcialmente sumergido avanzando

# Capítulo 5

# Resultados/Results

## 5.1. Resultados

A lo largo del documento se han desarrollado las propuestas de diseño de los sistemas de comunicaciones y control, haciendo un análisis detallado del comportamiento de cada sistema y de la consecución de los objetivos establecidos para el subproyecto. A continuación se realizada una enumeración resumiendo los logros del TFG.

1. Selección, experimentación y verificación de un cordón umbilical capaz de ofrecer un ancho de banda suficiente para transmitir imagen en vivo desde el dron a una distancia comprendida en el rango establecido en los objetivos del proyecto.

2. Análisis del ancho de banda en función de la longitud del cordón umbilical y de las limitaciones y retrasos asociados.

3. Diseño, configuración e implementación de una red que comunique todos los elementos del submarino de forma eficaz, estable y continua.

4. Configuración de ROS en todos los dispositivos de la red para su correcta integración.

5. Análisis detallado de las diferentes estrategias de control disponibles para la configuración del ROV.

6. Diseño, implementación y verificación de una estrategia de control manual que permita a un operador controlar el ROV de forma remota

7. Diseño de un programa de recogida de datos relacionados con el comportamiento dinámico del dron.

8. Integración efectiva de todos los sistemas diseñados para que sean compatibles con el resto de sistemas desarrollados para el proyecto 'U-Water Explorer Dron' por parte de David Henry 1.1 e Iván Torres 1.1.

## 5.2. Results

This memory has developed the different design proposals of the communication and control systems by doing a detailed analysis of the behaviour of every system and the consecution of the objectives set for this project. Below is done an enumeration of the achievements of the project.

1. Selection, experimentation and verification of a tether capable of offering enough bandwidth for streaming video from the drone at a distance within the range stablished at the objectives of the project.

2. Analysis of the dependence between the length of the tether and the network's bandwidth as well as the related retards and the limitations.

3. Design, configuration and implementation of a network that communicates all the devices of the underwater drone in a effective, stable and permanent way.

4. ROS configuration on all devices of the network for its correct integration.

5. Detailed analysis of the different control strategies available for the ROV.

6. Design, implementation and verification of a manual control strategy that enables an operator to control the drone remotely.

7. Design of a code for data acquisition to register the dynamic behaviour of the drone.

8. Integration of all the systems developed at this memory to gurantee the compatibility with the rest of the systems developed for the project 'U-Water Explorer Dron' by David Henry 1.1 and Iván Torres 1.1.

# Bibliografía


[1] Romano Capocci et al. «Inspection-Class Remotely Operated Vehicles». En: *Journal of Marine Science and Engineering* (2017). DOI: 10.3390/jmse5010013. URL: https://www.mdpi.com/2077-1312/5/1/13.

[2] Palmeiro et al. «Underwater radio frequency communications». En: *IEEE* (2011). DOI: 10.1109/Oceans-Spain.2011.6003580. URL: https://ieeexplore.ieee.org/document/6003580.

[3] Raspberry Pi Foundation. *Raspberry Pi Official Website*. English. [Online]. Visitado: 2019-05-10. URL: https://www.raspberrypi.org/about/.

[4] Arduino. *Arduino official website*. English. [Online]. Visitado: 2019-05-10. URL: https://www.arduino.cc/.

[5] Ubuntu by Canonical. *Ubuntu Official Website*. English. [Online]. Visitado: 2019-05-10. URL: https://www.ubuntu.com/.

[6] Ubuntu Mate Team. *Ubuntu Mate Official Website*. English. [Online]. Visitado: 2019-05-10. URL: https://ubuntu-mate.org/.

[7] Raspberry Pi Community. *RaWebsite Pi Forum*. English. [Online]. Visitado: 2019-05-10. 2019. URL: https://www.raspberrypi.org/forums/viewtopic.php?f=56&t=208538.

[8] Standford University y Willow Garage. *Robot Operating System Official Website*. English. [Online]. Visitado: 2019-05-10. URL: https://www.ros.org/.

[9] Standford University y Willow Garage. *ROS Kinetic Kame Official Wiki*. [Online]. Visitado: 2019-05-10. URL: http://wiki.ros.org/kinetic.

[10] Standford University y Willow Garage. *ROS Melodic Morenia Official Wiki*. [Online]. Visitado: 2019-05-10. URL: http://wiki.ros.org/melodic/.

[11] Linus Torvalds. *Git Official Website*. English. [Online]. Visitado: 2019-05-10. URL: https://git-scm.com/.

[12] GitHub Inc. English. [Online]. Visitado: 2019-05-10. URL: https://github.com/.

[13] Colomban Wendling et al. *Geany Official Website*. English. [Online]. Visitado: 2019-05-10. URL: https://www.geany.org/Main/HomePage.

[14] ROS Official Wiki. English. [Online]. Visitado: 2019-05-10. URL: http://wiki.ros.org/rosserial/Overview/Limitations.

[15] ROS Official Wiki. English. [Online]. Visitado: 2019-05-10. URL: http://wiki.ros.org/rosserial_arduino/Tutorials.

[16] Echostech GitHub Repository. English. [Online]. Visitado: 2019-05-10. URL: https://github.com/ecostech/rosabridge.

[17] Jonay Tomás Toledo Carrillo. «ESTRATEGIAS DE CONTROL PARA SISTEMAS MULTIVARIABLES NO LINEALES APLICACIÓN A UN HELICÓPTERO DE 4 ROTORES». Tesis doct. Escuela Técnica Superior de Ingeniería Informática de la Universidad de La Laguna, 2008.

ESCUELA SUPERIOR DE
INGENIERÍA Y TECNOLOGÍA

# Trabajo de Fin de Grado

## Diseño de Dron Submarino: Diseño e implementación del sistema de comunicaciones y diseño del sistema de control

## TOMO II

# Pliego de condiciones y presupuesto

**Titulación:** Grado en Ingeniería Electrónica Industrial y Automática

**Estudiante:** Nicolás Adrián Rodríguez Linares

**Tutor:** Leopoldo Acosta Sánchez
**Tutor:** Fernando Luis Rosa González

12 de junio de 2019

# Capítulo 6

# Pliego de Condiciones

El presente TFG desarrolla los sistemas de comunicaciones y de control de un ROV submarino diseñado en el marco de un proyecto particular titulado *'U-Water Explorer Dron'* *1.1*. Debido a que en sí mismo el subproyecto debe encajar con los otros sistemas del ROV diseñados, se deben cumplir una serie de condiciones de ligadura que garantice la operatividad del conjunto. Dichas condiciones de ligadura se detallan en la tabla 6.1.

Tabla 6.1: Tabla de condiciones que se deben satisfacer en el proyecto

| **Ámbito** | **Condición** |
|---|---|
| Sistema de comunicaciones | Selección de un cordón umbilical con una longitud comprendida entre 20 y 75 metros |
| Sistema de comunicaciones | Velocidad de transmisión suficiente para visión en directo |
| Sistema de comunicaciones | Diseño y configuración de la red interna del ROV en base a los elementos básicos del sistema |
| Sistema de comunicaciones | Análisis de los tipos de mensajes de ROS que se deben implementar en base a las capacidades de los dispositivos de la red |
| Ingeniería de Sistemas | Selección y configuración básica del ordenador de abordo del ROV |
| Ingeniería de Sistemas | Empleo de placas basadas en un procesador ATMega2560 como controladores del sistema sensorial y motriz |

*(Continúa en la página siguiente)*

*(Viene de la página anterior)*

Tabla 6.1: Tabla de condiciones que se deben satisfacer en el subproyecto

| **Ámbito** | **Condición** |
|---|---|
| Ingeniería de Sistemas | Uso de ROS como *framework* del proyecto |
| Ingeniería de Sistemas | Selección de la versión de ROS y el sistema operativo adecuado para los ordenadores del sistema |
| Sistema de control | Empleo de una configuración de 4 propulsores dispuestos en el mismo plano |
| Sistema de control | Selección de un dispositivo de control para el operador que sea compatible con el sistema |
| Sistema de control | Diseño y análisis de las posibles estrategias de control en base a la disposición de los motores |
| Sistema de control | Implementación de una estrategia de control que de al submarino la capacidad de ascender, descender, avanzar y girar en torno a los 3 ángulos de Tait-Bryan 4.1 |
| Sistema de control | Diseño de unos comandos de operación del sistema coherentes con otros drones del mercado. |
| Sistema de control | Diseño e implementación de una parada de emergencia independiente del control de motores |

Las condiciones anteriores fueron establecidas de forma coordinada entre los 3 integrantes del proyecto *U-Water Explorer Dron* y autores de los 3 TFG indicados en la introducción del documento 1.1 y limitan el marco de trabajo del presente TFG.

# Capítulo 7

# Presupuesto

En este capítulo se van a desglosar los costes materiales y de mano de obra pertenecientes al presente TFG. Todos los costes que se van a detallar a continuación incluyen impuestos directos e indirectos, además de la seguridad social en el caso que proceda.

En la tabla 7.1 se pueden consultar los elementos que forman parte del submarino y que se han tenido que comprar para integrarlos dentro del sistema o como herramientas necesarias para su montaje.

Tabla 7.1: Tabla de costes de los materiales del subproyecto

| **Material** | **Proveedor** | **P.V.P(€)** | **Unidad** | **Coste(€)** |
|---|---|---|---|---|
| ArduPilot APM 2.5 | ArduPilot | 21.18 | 1 | 21.18 |
| Conector RJ45 | Leroy Merlin | 0.42 | 16 | 6.72 |
| Raspberry Pi Model 3B+ | Alcampo | 41.9 | 1 | 41.9 |
| Crimpadora RJ45 | Leroy Merlin | 13.25 | 1 | 13.25 |
| Arduino Mega Original | Nalbert | 51.12 | 1 | 51.12 |
| Controller PS3 | GAME España | 9.85 | 1 | 9.95 |
| Bobina 100m cable Ethernet CAT5-E UTP | APPInformatica | 24.1 | 1 | 24.1 |
| Coste total(€) | | | | 168.22 |

Por otro lado, en la tabla 7.2 se puede consultar el desglose de las horas empleadas en el diseño de los sistemas objetos de este trabajo, así como el coste de la mano de obra con impuestos y seguridad social incluida.

Tabla 7.2: Tabla de la mano de obra del subproyecto

| **Concepto** | **Horas** | **Coste(€/h)** | **Importe(€)** |
|---|---|---|---|
| Diseño de los sistemas | 100 | 25 | 2500 |
| Programación | 250 | 20 | 5000 |
| Verificación de los sistemas | 100 | 18 | 1800 |
| Realización de pruebas | 70 | 20 | 1400 |
| Redacción del TFG | 30 | 18 | 540 |
| Coste total(€) | | | 11240 |

Finalmente, se puede consultar en la tabla 7.3 el coste acumulado de las tablas 7.1 y 7.2.

Tabla 7.3: Tabla de coste total del subproyecto

| **Concepto** | **Coste(€)** |
|---|---|
| Coste materiales | 168.22 |
| Coste mano de obra | 11240 |
| Coste total(€) | 11408.22 |

ESCUELA SUPERIOR DE
INGENIERÍA Y TECNOLOGÍA

# Trabajo de Fin de Grado

## Diseño de Dron Submarino: Diseño e implementación del sistema de comunicaciones y diseño del sistema de control

## TOMO III

## Anexos

**Titulación:** Grado en Ingeniería Electrónica Industrial y Automática

**Estudiante:** Nicolás Adrián Rodríguez Linares

**Tutor:** Leopoldo Acosta Sánchez
**Tutor:** Fernando Luis Rosa González

12 de junio de 2019

# Capítulo 8

# Anexos: Códigos realizados

## 8.1. Nodo joy_publisher

```
#include <ros/ros.h>
#include <sensor_msgs/Joy.h>
#include <std_msgs/Float32MultiArray.h>
#include <std_msgs/Int32MultiArray.h>
#include <std_msgs/Float32.h>
#include <vector>

// Declaracion de variables globales
int firstExec = 0;
ros::Time begin;
ros::Time actual;
const double freq = 5; // En Hz
const float errorMaximo = 3; // maximo error de profundidad admisible
bool controlProfundidad = 0;
float profundidadActual, profundidadPrevia, profundidadConsigna = 0;

const float sensibilidadYaw = 0.05;
const float reduccionPot = 0.65; // constante que controla el desequilibrio de ascenso en Yaw
const float desfasePitch = 0.6; // constante que controla el desequilibrio que provoca el Pitch

const float kp = 1.3;
const float kd = 1.1;
// Declaramos objetos publishers globales

ros::Publisher joy_command; // publisher seleccion de comandos de los joysticks
```

```
26 ros::Publisher button_command; // publisher seleccion de
       comandos de botones
27
28 // Prototipo de funciones
29
30 std::vector<float> motores(std::vector<float> joysticks);
31 float depthController(float depth);
32
33 // Callbacks
34
35 void joyCallback(const sensor_msgs::Joy::ConstPtr& joys){
36
37   std_msgs::Float32MultiArray command_motors;
38   std_msgs::Int32MultiArray sel_buttons;
39
40   std::vector<float> joysticks;
41
42   command_motors.data.clear();
43   sel_buttons.data.clear();
44
45   joysticks.push_back(-(joys->axes[0])); // eje roll
46   joysticks.push_back(joys->axes[1]); // eje power
47   joysticks.push_back(-(joys->axes[3])); // eje yaw
48   joysticks.push_back(joys->axes[4]); // eje pitch
49
50   joysticks = motores(joysticks);
51
52   command_motors.data.push_back(joysticks[0]); // M1
53   command_motors.data.push_back(joysticks[1]); // M2
54   command_motors.data.push_back(joysticks[2]); // M3
55   command_motors.data.push_back(joysticks[3]); // M4
56
57   for(int i = 0; i < 8; i++){
58
59     sel_buttons.data.push_back(joys->buttons[i]);
60     /* X,O,Triangulo,Cuadrado
61      * L1,R1,L2,R2*/
62
63   }
64
65   actual = ros::Time::now();
66   if((actual - begin).toSec() >= (1/freq)){
67
68     // En caso de que este establecido el control de profundidad
69
70     /*if(controlProfundidad = 1){
71
72       ROS_INFO("Acceso al control de profundidad");
73       command_motors.data.clear();
```

```
74
75        joysticks[0] = (-(joys->axes[0])); // eje yaw
76        joysticks[1] = depthController(profundidadActual); //
      sustituimos eje power
77        joysticks[2] = (-(joys->axes[3])); // eje roll
78        joysticks[3] = (joys->axes[4]); // eje pitch
79
80        joysticks = motores(joysticks);
81
82        command_motors.data.push_back(joysticks[0]); // M1
83        command_motors.data.push_back(joysticks[1]); // M2
84        command_motors.data.push_back(joysticks[2]); // M3
85        command_motors.data.push_back(joysticks[3]); // M4
86
87      }*/
88
89      // Comprobacion de la activacion del control de profundidad
90
91      /*if(sel_buttons.data[4] == 1 && sel_buttons.data[5] == 1){
      // L1 + R1
92        controlProfundidad = 1;
93        ROS_INFO("Activacion del control de profundidad");
94        profundidadConsigna = profundidadActual; //  se establece
      la consigna de profunidad
95        profundidadPrevia = profundidadActual; // actualizamos
      informacion para el controlador derivativo
96      }
97      if(sel_buttons.data[4] == 1 && sel_buttons.data[7] == 1){ //
      L1 + R2
98        controlProfundidad = 0;
99        ROS_INFO("Desactivacion del control de profundidad");
100     }*/
101     joy_command.publish(command_motors); // publicamos comandos
      de los joysticks
102     button_command.publish(sel_buttons); // publicamos comandos
      botones
103     begin = ros::Time::now();
104   }
105 }
106
107 void profundidadCallback(const std_msgs::Float32::ConstPtr&
      depth){
108   profundidadActual = (depth->data);
109 }
110 int main(int argc, char **argv)
111 {
112   ros::init(argc, argv, "joy_command");
113
114
```

```
115   ros::NodeHandle n;
116
117   joy_command = n.advertise<std_msgs::Float32MultiArray>("
        joy_command", 1000);
118   button_command = n.advertise<std_msgs::Int32MultiArray>("
        button_command", 1000);
119
120   if(firstExec == 0){
121     begin = ros::Time::now();
122     firstExec = 1;
123   }
124   ros::Subscriber joy_sub = n.subscribe<sensor_msgs::Joy>("joy"
        ,10,joyCallback); // escuchamos nodo Joy de la libreria
125   ros::Subscriber profundidad_sub = n.subscribe<std_msgs::
        Float32>("depth",10,profundidadCallback);
126   ros::spin();
127
128 }
129
130
131 // Declaracion de funciones
132
133 std::vector<float> motores(std::vector<float> joysticks){
134
135   std::vector<float> motors;
136   float m1, m2, m3, m4 = 0;
137
138   if((joysticks[2] < sensibilidadYaw) && (joysticks[2] > (-
        sensibilidadYaw))){
139     // cuando no hay movimiento de Yaw
140     if(joysticks[3] > 0){
141       // con m2 y m3 invertidos: Power - Pitch
142       // Con Pitch positivo
143       m1 = ((joysticks[1]/2) + (joysticks[3]/2)*(desfasePitch))
        * (-1);
144       m2 = ((-joysticks[1]/2) - (joysticks[3]/2)*(desfasePitch))
         * (-1);
145       m3 = ((-joysticks[1]/2) - (joysticks[3]/2)) * (-1);
146       m4 = ((joysticks[1]/2) + (joysticks[3]/2)) * (-1);
147     }else{
148       // con m2 y m3 invertidos: Power - Pitch
149       // Con Pitch negativo, se multiplica la componente Pitch
        por -1 para compensar el signo
150       m1 = ((joysticks[1]/2) - (joysticks[3]/2)) * (-1);
151       m2 = ((-joysticks[1]/2) + (joysticks[3]/2)) * (-1);
152       m3 = ((-joysticks[1]/2) + (joysticks[3]/2)*(desfasePitch))
         * (-1);
153       m4 = ((joysticks[1]/2) - (joysticks[3]/2)*(desfasePitch))
        * (-1);
```

```
154     }
155   }else{
156     // en el caso de que exista Yaw
157     // con m2 y m3 invertidos: joysticks[1] = Ganancia/Power,
       joysticks[2] = Yaw
158     if(joysticks[2] > 0){
159       m2 = (joysticks[2] / 2) * (1 + joysticks[1]);
160       m3 = m2*0.85;
161       m1 = reduccionPot* m2;
162       m4 = m1;
163     }else{
164       m1 = (joysticks[2] / 2) * (1 + joysticks[1]);
165       m4 = m1;
166       m2 = reduccionPot* m1;
167       m3 = m2*0.85;
168     }
169   }
170
171   motors.push_back(m1);
172   motors.push_back(m2);
173   motors.push_back(m3);
174   motors.push_back(m4);
175
176   return motors;
177
178 }
179
180 float depthController(float depth){
181
182   float actuador = 0;
183   float error = (profundidadActual - profundidadConsigna)/
       errorMaximo;
184   if((error > 0.5) && (error <= 1.5)){
185     ROS_WARN("El controlador no esta manteniendo la profundidad
       adecuadamente. Desviacion de %f m", (error*errorMaximo));
186   }else if(error > 1.5){
187     ROS_ERROR("Error excesivo: recupere el control manual de
       profundidad. Desviacion de %f m", (error*errorMaximo));
188   }
189   actuador = kp * error + kd * ((profundidadPrevia -
       profundidadActual) * freq);
190   if(actuador > 1){
191     actuador = 1;
192     ROS_DEBUG("Saturacion en el controlador de altura");
193   }else if(actuador < -1){
194     actuador = -1;
195     ROS_DEBUG("Saturacion en el controlador de altura");
196   }
```

```
197   profundidadPrevia = profundidadActual; // Actualizacion para
        el siguiente ciclo
198
199   return actuador;
200 }
```

Listing 8.1: Nodo joy_publisher encargado de transformar las señales de control del operador en un vector de comandos para los propulsores

## 8.2. launchfile nodos joy y joy_publisher

```
1 <launch>
2   <node name="joy_publisher" pkg="joy_command" type="
      joy_publisher">
3   </node>
4   <node name="joy_node" pkg="joy" type="joy_node">
5   </node>
6 </launch>
```

Listing 8.2: Archivo de lanzamiento de los nodos implicados en la lectura del mando de PS3

## 8.3. Nodo storage

```
1 #!/usr/bin/python
2 #coding: utf-8
3 import pandas as pd
4 import rospy
5 from geometry_msgs.msg import PoseStamped
6 from std_msgs.msg import Float32MultiArray
7 from std_msgs.msg import Int32MultiArray
8 from std_msgs.msg import Float32
9 import os
10 from os.path import expanduser
11 import datetime
12 import sys
13 # Declaracion de las listas que van a ser empleadas para
      almacenar los datos
14 # que se recogen del submarino
15 profundidad = []
16 timestamp_profundidad = []
17 x = []
18 y = []
19 z = []
20 w = []
21 timestamp_angulos = []
22 m1 = []
23 m2 = []
```

```
m3 = []
m4 = []
timestamp_motors = []
#La variable motoresActivos indica si las ordenes de los motores se ejecutan o no
motoresActivos = 0

def profundidadCallback(data):
  #Guardamos el valor de la profundidad en la lista correspondiente
  profundidad.append(data.data)
  timestamp_profundidad.append(rospy.get_time())

def imuCallBack(data):
  #Guardamos los valores de los cuaterniones en su lista correspondiente
  x.append(data.pose.orientation.x)
  y.append(data.pose.orientation.y)
  z.append(data.pose.orientation.z)
  w.append(data.pose.orientation.w)
  timestamp_angulos.append(rospy.get_time())

def motorsCallback(data):
  #Guardamos los valores de los motores en su lista correspondiente
  if motoresActivos == 1:
    m1.append(data.data[0])
    m2.append(data.data[1])
    m3.append(data.data[2])
    m4.append(data.data[3])
    timestamp_motors.append(rospy.get_time())
  else:
    m1.append(0)
    m2.append(0)
    m3.append(0)
    m4.append(0)
    timestamp_motors.append(rospy.get_time())

def buttonsCallback(data):
  #Detectamos la activacion y desactivacion de los motores para registrar correctamente
  #si los valores que se envian se estan ejecutando o no
  if (data.data[0] == 1) and (data.data[4] == 1):
    motoresActivos = 1
  if (data.data[1] == 1) and (data.data[4] == 1):
    motoresActivos = 0

def ejecucion():

```

```
68    rospy.init_node('store', anonymous=True)
69
70    rospy.Subscriber("joy_command", Float32MultiArray,
        motorsCallback)
71    rospy.Subscriber("button_command", Int32MultiArray,
        buttonsCallback)
72    rospy.Subscriber("depth", Float32, profundidadCallback)
73    rospy.Subscriber("pose", PoseStamped, imuCallBack)
74    rospy.spin()
75
76    angles_tupla = list(zip(timestamp_angulos,x,y,z,w))
77    depth_tupla = list(zip(timestamp_profundidad,profundidad))
78    motors_tupla = list(zip(timestamp_motors,m1,m2,m3,m4))
79
80    angles_df = pd.DataFrame(angles_tupla, columns=['Time','X','Y'
        ,'Z','W'])
81    depth_df = pd.DataFrame(depth_tupla, columns=['Time','
        Profundidad'])
82    motors_df = pd.DataFrame(motors_tupla, columns=['Time','m1','
        m2','m3','m4'])
83
84    date = datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S")
85
86    angles_csv = angles_df.to_csv (pathFolder + 'angles_' + date +
        '.csv', index = None, header=True)
87    depth_csv = depth_df.to_csv (pathFolder + 'depth_' + date + '.
        csv', index = None, header=True)
88    motors_csv = motors_df.to_csv (pathFolder + 'motors_' + date +
        '.csv', index = None, header=True)
89  if __name__ == '__main__':
90    home = expanduser("~")
91    pathFolder = home + "/Documentos/data/"
92    ejecucion()
```

Listing 8.3: Nodo de almacenamiento de datos en ejecución

## 8.4. Nodo almacen_datos

```
1  #include <ros/ros.h>
2  #include <std_msgs/Float32MultiArray.h>
3  #include <std_msgs/Int32MultiArray.h>
4  #include <std_msgs/Float32.h>
5  #include <sensor_msgs/Imu.h>
6  #include <geometry_msgs/PoseStamped.h>
7
8  #include <vector>
9  #include <string>
10 #include <iostream>
11 #include <fstream>
```

```
12 #include <sstream>
13 #include <time.h>
14 #include <math.h>
15 #include <stdlib.h>
16
17 // Declaracion de variables globales
18
19 bool motoresEnable = 0;
20
21 // Nombre ficheros a escribir
22
23 std::string homedir = std::getenv("HOME");
24 std::string nombreAlmacenMotores = homedir + "/Documentos/data/
    motors_";
25 std::string nombreAlmacenProfundidad = homedir + "/Documentos/
    data/depth_";
26 std::string nombreAlmacenImu = homedir + "/Documentos/data/
    angles_";
27
28 // Funciones
29
30 void creacionFicheros(){
31
32   time_t t = time(0);   // get time now
33   struct tm * now = localtime( & t );
34
35   char buffer [80];
36   strftime (buffer,80,"%Y-%m-%d.",now);
37
38   nombreAlmacenMotores = nombreAlmacenMotores + buffer + ".csv";
39   nombreAlmacenProfundidad = nombreAlmacenProfundidad + buffer +
       ".csv";
40   nombreAlmacenImu = nombreAlmacenImu + buffer + ".csv";
41
42
43 }
44 // Callbacks
45
46 void joyCallback(const std_msgs::Float32MultiArray::ConstPtr&
    command){
47
48   double tiempoActual = (ros::Time::now()).toSec();
49   std::string linea = "";
50
51   std::string m1 = std::to_string(command->data[0]);
52   std::string m2 = std::to_string(command->data[1]);
53   std::string m3 = std::to_string(command->data[2]);
54   std::string m4 = std::to_string(command->data[3]);
55
```

```
56    if (motoresEnable == 1){
57
58      linea = std::to_string(tiempoActual) + ',' + m1 + ',' + m2 +
        ',' + m3 + ',' + m4;
59
60    }else{
61      linea = std::to_string(tiempoActual) + ',' + '0' + ',' + '0'
        + ',' + '0' + ',' + '0';
62    }
63
64    std::ofstream almacenMotores(nombreAlmacenMotores, std::
      ofstream::out | std::ofstream::app);
65    almacenMotores << linea << std::endl;
66    almacenMotores.close();
67  }
68
69  void profundidadCallback(const std_msgs::Float32::ConstPtr&
      depth){
70
71    double tiempoActual = (ros::Time::now()).toSec();
72    std::string profundidadActual = std::to_string(depth->data);
73    std::string linea = std::to_string(tiempoActual) + ',' +
      profundidadActual;
74
75    std::ofstream almacenProfundidad(nombreAlmacenProfundidad, std
      ::ofstream::out | std::ofstream::app);
76    almacenProfundidad << linea << std::endl;
77    almacenProfundidad.close();
78
79  }
80
81  void imuCallback(const geometry_msgs::PoseStamped::ConstPtr&
      angles){
82
83    double tiempoActual = (ros::Time::now()).toSec();
84
85    float x = (angles->pose.orientation.x);
86    float y = (angles->pose.orientation.y);
87    float z = (angles->pose.orientation.z);
88    float w = (angles->pose.orientation.w);
89
90    std::string linea = std::to_string(tiempoActual) + ',' + std::
      to_string(x) + ',' +
91        std::to_string(y) + ',' + std::to_string(z) + ',' + std::
      to_string(w);
92
93    std::ofstream almacenImu(nombreAlmacenImu, std::ofstream::out
      | std::ofstream::app);
94    almacenImu << linea << std::endl;
```

```
95    almacenImu.close();
96
97  }
98  void buttonCallback(const std_msgs::Int32MultiArray::ConstPtr&
      button){
99
100   if((button->data[4] == 1) && (button->data[0] == 1)){
101     motoresEnable = 1;
102   }
103
104   if((button->data[4] == 1) && (button->data[1] == 1)){
105     motoresEnable = 0;
106   }
107
108 }
109 int main(int argc, char **argv)
110 {
111
112   creacionFicheros();
113
114   ros::init(argc, argv, "almacen");
115
116
117   ros::NodeHandle n;
118
119   ros::Subscriber joy_sub = n.subscribe<std_msgs::
        Float32MultiArray>("joy_command",10,joyCallback); //
        escuchamos consignas motor
120   ros::Subscriber button_sub = n.subscribe<std_msgs::
        Int32MultiArray>("button_command",10,buttonCallback); //
        escuchamos botones
121   ros::Subscriber profundidad_sub = n.subscribe<std_msgs::
        Float32>("abcd",10,profundidadCallback);
122   ros::Subscriber imu_sub = n.subscribe<geometry_msgs::
        PoseStamped>("pose",10,imuCallback);
123   ros::spin();
124
125 }
```

Listing 8.4: Nodo de almacenamiento de datos en ejecución capaz de lanzarse en medio de la ejecución

## 8.5. launchfile conjunto nodos joy, joy_publisher y storage

```
1 <?xml version="1.0"?>
2 <launch>
3   <include file="$(find joy_command)/launch/pc.launch"/>
```

```
  <node pkg='storage' name='storage' type='store.py'/>
</launch>
```

Listing 8.5: Archivo de lanzamiento de los nodos implicados en el movimiento del dron

## 8.6. launchfile remoto de los nodos de la Raspberry Pi

```
<?xml version="1.0"?>
<launch>
  <machine name="pi" address="pi" env-loader="/home/pi/
    sub_master/Code/raspberry/devel/setup.bash" user="pi" />
    <node machine ="pi" pkg="rosserial_python" type="serial_node
    .py" name="arduSensor" output="screen">
      <param name="port" value="/dev/ttyUSB0" />
      <param name="baud" value="57600"/>
    </node>
    <node machine="pi" pkg="rosserial_python" type="serial_node.
    py" name="arduMove" output="screen">
      <param name="port" value="/dev/ttyACM0" />
      <param name="baud" value="57600"/>
    </node>
    <node machine="pi" pkg="audio_pub" type="talker" name="
    audioCamera" output="screen"/>
    <include file="$(find usb_cam)/launch/usb_cam-test.launch"/>
    <include file="$(find ros_mpu6050_node)/launch/mpu6050.
    launch"/>
</launch>
```

Listing 8.6: Archivo de lanzamiento remoto de los nodos de la Raspberry Pi

## 8.7. launchfile de los nodos de la Raspberry Pi

```
<?xml version="1.0"?>
<launch>
  <node pkg="rosserial_python" type="serial_node.py" name="
    arduSensor" output="screen">
    <param name="port" value="/dev/ttyACM1" />
    <param name="baud" value="57600"/>
  </node>
  <node pkg="rosserial_python" type="serial_node.py" name="
    arduMove" output="screen">
    <param name="port" value="/dev/ttyACM0" />
    <param name="baud" value="57600"/>
  </node>

```

```
12   <node pkg="audio_pub" type="talker.py" name="audioCamera"
       output="screen" />
13   <node pkg="conversion_profundidad" type="profundidad_publisher
       " name="profundidad" output="screen"/>
14   <include file="$(find usb_cam)/launch/usb_cam-test.launch"/>
15   <include file="$(find ros_mpu6050_node)/launch/mpu6050.launch"
       />
16 </launch>
```

Listing 8.7: Archivo de lanzamiento de los nodos de la Raspberry Pi

## 8.8. Nodo conversión unidades de la profundidad

```
1  #include <ros/ros.h>
2  #include <sensor_msgs/FluidPressure.h>
3  #include <std_msgs/Float32.h>
4  #include <vector>
5
6  // Declaracion de variables globales
7  int firstExec = 0;
8  ros::Time begin;
9  ros::Time actual;
10 const double freq = 25; // En Hz
11 // Declaramos objetos publishers globales
12
13 ros::Publisher profundidad; // publisher de la profundidad en
       metros
14
15 // Callbacks
16
17 void profundidadCallback(const sensor_msgs::FluidPressure::
       ConstPtr& depth){
18
19   std_msgs::Float32 depth_msgs;
20   depth_msgs.data = depth->fluid_pressure;
21   if((depth_msgs.data > 22) && (depth_msgs.data < 26)){
22
23     ROS_ERROR("Profundidad excesiva, emerja.");
24
25   }else if(depth_msgs.data > 26){
26     ROS_FATAL("Profundidad excesiva, posibles danos. Recupere el
         ROV");
27   }
28
29   actual = ros::Time::now();
30   if((actual - begin).toSec() >= (1/freq)){
31     profundidad.publish(depth_msgs); // publicamos la
       profundidad en metros
32     begin = ros::Time::now();
```

```
33   }
34 }
35
36 int main(int argc, char **argv)
37 {
38   ros::init(argc, argv, "depth_conversion");
39
40
41   ros::NodeHandle n;
42
43   profundidad = n.advertise<std_msgs::Float32>("depth", 1000);
44
45   if(firstExec == 0){
46     begin = ros::Time::now();
47     firstExec = 1;
48   }
49   ros::Subscriber depth_sub = n.subscribe<sensor_msgs::
       FluidPressure>("Presion_externa_en_bares",10,
       profundidadCallback); // escuchamos nodo Joy de la libreria
50
51   ros::spin();
52
53 }
```

Listing 8.8: Nodo de conversión de la presión externa a profundidad medida en metros

## 8.9. Makefile arduMove

```
1 ARDUINO_DIR = /usr/share/arduino
2 ARDUINO_PORT = /dev/ttyACM0
3 BOARD_TAG = mega2560
4 USER_LIB_PATH = /home/pi/sub_master/Code/arduino/libraries
5 ARDUINO_LIBS =
6
7 include /usr/share/arduino/Arduino.mk
```

Listing 8.9: Archivo de compilación en consola del código de control de los propulsores de arduMove

## 8.10. Makefile arduSensor

```
1 ARDUINO_DIR = /usr/share/arduino
2 ARDUINO_PORT = /dev/ttyACM1
3 BOARD_TAG = mega2560
4 USER_LIB_PATH = /home/pi/sub_master/Code/arduino/libraries
5 ARDUINO_LIBS =
6
```

```
7 include /usr/share/arduino/Arduino.mk
```

Listing 8.10: Archivo de compilación en consola del código de control de los sensores de arduSensor