TRABAJO FIN DE MASTER EN INGENIERÍA INDUSTRIAL

DISEÑO E IMPLEMENTACIÓN DE ALGORITMOS DE EVITACIÓN DE OBSTÁCULOS PARA UN VEHÍCULO AUTÓNOMO SUBMARINO

AUTORA: PATRICIA ARGUIMBAU GUARINOS

TUTOR: F. XAVIER BLASCO FERRAGUD

COTUTORES: MIQUEL MASSOT CAMPOS
ANTONI BURGUERA BURGUERA

Curso Académico: 2017-18

Agradecimientos

Agradecer hoy y siempre a mi familia, en especial a mis padres y mi hermano por haberme brindado siempre su apoyo, por confiar en mis posibilidades y entenderme. Sobretodo por sus buenos consejos y su constante motivación para poder finalizar este trabajo y enseñarme a dar siempre lo mejor de mí cueste lo que cueste, pero más que nada por su amor y confianza en todo momento.

Quiero agradecer a mi tutor, y sobretodo en especial a mis cotutores de la Universidad de les Illes Balears ya que sin ellos no hubiese sido posible la realización de este Trabajo Final de Máster. Gracias a ellos que me han orientado, apoyado y corregido en esta labor.

Finalmente quiero agradecer a mis amigos, sin dejar a nadie en el olvido, por estar ahí siempre que os he necesitado y brindarme vuestra ayuda incondicional. Muy especialmente gracias a la persona que me ha hecho tener una permanente sonrisa durante todos los días de la recta final de este trabajo.

Gracias a todos.

Resumen

Los comportamientos y estrategias de evitación de obstáculos son una pieza esencial dentro de la arquitectura de control de cualquier robot móvil. Su función es evitar que el robot colisione con elementos de su entorno, a partir de los datos proporcionados por uno o varios sensores.

En robótica terrestre o aérea, actualmente los sensores de referencia para evitación de obstáculos son ópticos como cámaras o láseres. En cambio, en el entorno submarino, los sensores más usados son acústicos como los sonares, dada su mayor distancia operativa a pesar de su menor resolución espacial y temporal.

En este Trabajo de Fin de Máster se plantea la simulación de un sónar de imagen de barrido mecánico para una posterior implementación de un algoritmo de detección de obstáculos junto un algoritmo de evitación para un vehículo submarino autónomo (AUV). Concretamente, se usará el vehículo AUV Turbot Sparus II junto un sonar Miniking de Tritech. La arquitectura del robot está implementada en ROS (Robot Operating System), por lo que todo el desarrollo se realizará en este entorno. Es importante remarcar que sólo se considerará la evitación de obstáculos en 2D, es decir, el vehículo sólo maniobrará en un plano horizontal, paralelo a la superficie del agua.

Las pruebas realizadas permitirán analizar el comportamiento de la plataforma en entornos simulados de complejidad variable y controlada. A partir de los resultados obtenidos, se podrá reutilizar todo lo aprendido en detección y evitación de obstáculos en plataformas reales para así tener un mayor conocimiento del comportamiento en un entorno real como el océano.

Palabras Clave: AUV, detección, evitación, navegación autónoma.

Resum

Els comportaments i estratègies d'evitació d'obstacles són una peça essencial dins de l'arquitectura de control de qualsevol robot mòbil. La seva funció és evitar que el robot col·lisioni amb elements del seu entorn, a partir de les dades proporcionades per un o diversos sensors.

En robòtica terrestre o aèria, actualment els sensors de referència per evitació d'obstacles són òptics com les càmeres o els làsers. En canvi, en l'entorn submarí, els sensors més utilitzats són acústics com els sonars, donada la seva major distància operativa malgrat la seva menor resolució espacial i temporal.

En aquest Treball de Fi de Màster es planteja la simulació d'un sonar d'imatge d'escombrat mecànic per a una posterior implementació d'un algoritme de detecció d'obstacles juntament amb un algoritme d'evitació per a un vehicle submarí autònom (AUV). Concretament, es farà servir el vehicle AUV Turbot Sparus II amb un sonar Miniking de Tritech. L'arquitectura del robot està implementada en ROS (Robot Operating System), de manera que tot el desenvolupament es realitzarà en aquest entorn. És important remarcar que només es considerarà l'evitació d'obstacles en 2D, és a dir, el vehicle només maniobrarà en un pla horitzontal, paral·lel a la superfície de l'aigua.

Les proves realitzades permetran analitzar el comportament de la plataforma en entorns simulats de complexitat variable i controlada. A partir dels resultats obtinguts, es podrà reutilitzar tot l'après en detecció i evitació d'obstacles en plataformes reals, per així tenir un major coneixement del comportament en un entorn real com l'oceà.

Paraules clau: AUV, detecció, evitació, navegació autònoma.

Abstract

Obstacle avoidance behaviors and strategies are an essential part of the control architecture of any mobile robot. Their function is to prevent the robot from colliding with elements of its environment, from the data provided by one or several sensors.

In terrestrial or aerial robotics, currently the reference sensors for obstacle avoidance are optical like cameras or lasers. On the other hand, in the underwater environment, the most used sensors are acoustic like sonars, given their greater operational distance despite their lower spatial and temporal resolution.

In this Master's Thesis, we propose the simulation of a mechanical imaging sonar for a later implementation of an obstacle detection algorithm together with an avoidance algorithm for an autonomous underwater vehicle (AUV). Specifically, the AUV Turbot Sparus II vehicle will be used together with a Tritech Miniking sonar. The architecture of the robot is implemented in ROS (Robot Operating System), so all the development will be done in this environment. It is important to note that only obstacle avoidance in 2D will be considered, that is, the vehicle will only maneuver in a horizontal plane, parallel to the surface of the water.

The tests carried out will allow to analyze the behavior of the platform in simulated environments of variable and controlled complexity. From the results obtained, it can be reused everything learned in detection and obstacle avoidance in real platforms in order to have a better knowledge of the behavior in a real environment such as the ocean.

Keywords: AUV, detection, avoidance, autonomous navigation.

Índice general

Resumen V

Resum VII

Abstract IX

Índice general XI

I Memoria 1

1 Introducción 3

1.1 Exploración oceánica: límites, costes y peligros . . . 3
1.2 Tipos de Robots Submarinos . . . 5
1.3 Aplicaciones de los UUVs . . . 8
1.4 Objetivo . . . 8
1.5 Motivación . . . 8
1.6 Estructura del documento . . . 9

2 Fundamentos teóricos 11

2.1 Definición de sónar . . . 11
2.2 Tipos de sónares . . . 13
2.3 Detección y evitación de obstáculos . . . 15

3 Herramientas Software Utilizadas 25

3.1 Entornos de desarrollo . . . 25
3.2 Entorno de simulación: UWSIM y RVIZ . . . 26

4 Implementación 31

4.1 Sparus II AUV . . . 32
4.2 Simulación de un MSIS usando ROS, UWSIM y COLA2 . . . 32
4.3 Detección de obstáculos. . . . 38
4.4 Evitación de obstáculos usando *move_base* . . . 39

5 Resultados 41

5.1 Prueba 1. Escenario con piedras . . . 41
5.2 Prueba 2. Escenario con columnas. . . . 44

6 Conclusión 47

6.1 Conclusiones y líneas de trabajo futuro . . . 47

II Presupuesto 49

III Anexos 63

A Código de mb_sensor.py 65

B Código de twist2bvr.py 71

C Código de visor_obstaculos.py 73

D Código de sparus2.urdf 77

Bibliografía 79

Parte I

Memoria

Capítulo 1

Introducción

El objetivo de este primer capítulo es realizar una pequeña introducción a la robótica submarina. Se describen los tipos de vehículos submarinos que se encuentran en la industria así como sus aplicaciones. También se expone la motivación del trabajo y el objetivo del mismo. Para terminar, se comenta la estructura del presente documento.

1.1 Exploración oceánica: límites, costes y peligros

Hace cientos de millones de años surgió la vida en el océano. El medio marino es el ecosistema más importante de la Tierra, ocupando un área inmensa, con más de dos terceras partes de la superficie del planeta. Es decir, aproximadamente más de un 70 % de la superficie total está ocupada por océanos (ver figura 1.1).

Figura 1.1: Vista general de toda la superficie de la tierra
Fuente: Free Printable Map

Muchos científicos como por ejemplo Kim Juniper, investigador en el departamento de Biología de la Universidad de Victoria (Canadá) consideran que se sabe más de la superficie de Marte que de los fondos marinos y organismos que viven en ellos [7].

La exploración oceanográfica ha aparecido recientemente como una de las áreas tecnológicas de mayor expansión. El conocimiento científico del océano está creciendo rápidamente debido a la evolución de nuevas tecnologías. La primordial razón de este auge reside en la gran cantidad de yacimientos geológicos detectados por los satélites, además de las numerosas reservas de petróleo y gas que se encuentran en los fondos marinos, a la espera de ser descubiertas por el hombre para su futura explotación económica.

Los vehículos submarinos son dispositivos capaces de navegar sobre o bajo el nivel de la superficie del agua. Estos robots tienen multitud aplicaciones ya que acceden a lugares de difícil acceso para el ser humano causando un progreso en su utilización a pesar de su alto coste económico.

Hasta hace relativamente pocos años, el principal objetivo del diseño de los vehículos submarinos convencionales se restringía básicamente a trabajos relacionados con el tendido de cables destinados fundamentalmente a servicios de telecomunicaciones o trabajos vinculados con la extracción de petróleo y gas en las plataformas oceánicas.

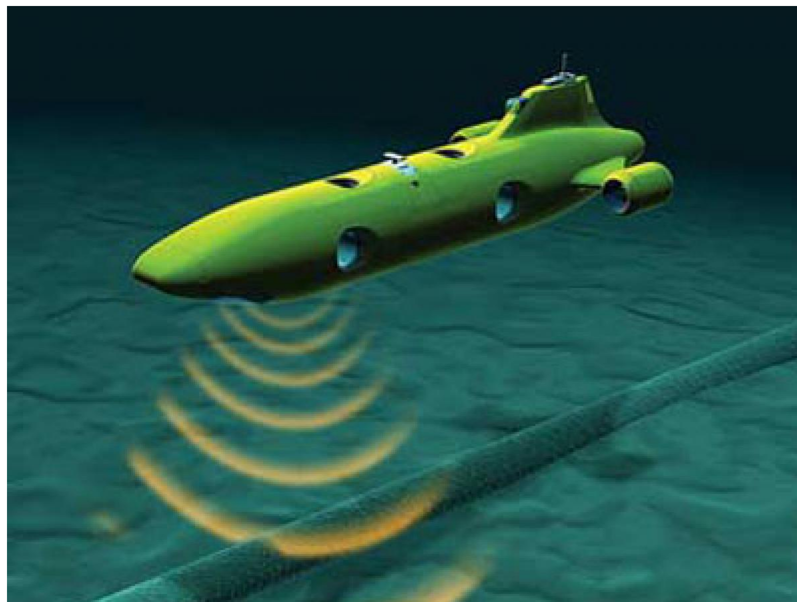

Figura 1.2: Imagen de un vehículo submarino autónomo en una aplicación de seguimiento de cables submarinos. Fuente: Mech Mecca

Sin embargo, hoy en día no sólo se requieren los vehículos submarinos convencionales para estas aplicaciones. Sino que en los últimos tiempos se han producido grandes avances en la robótica que, entre muchas otras cosas, han impulsado la evolución de los vehículos submarinos no tripulados, también conocidos como vehículos submarinos autónomos (figura 1.2).

Este tipo de vehículos está empezando a poblar parte del océano (ver figura 1.3), los cuales ya se disponen para diversos fines tales como pueden ser el reconocimiento e investigación del fondo marino, aplicaciones militares centradas en la localización y neutralización de minas así como la recuperación de armamento, misiones de recuperación y rescate de vehículos hundidos, tareas de inspección y reparación de buques, entre otras muchas posibilidades.

En todos estos entornos existe el riesgo de una colisión del robot con el entorno, ya sea con estructuras humanas o con orografía marina. Dado el frágil o inexistente canal de comunicaciones

Figura 1.3: Diferentes tipos de vehículos submarinos autónomos.
Fuente: Picture Lights

entre un operador y su AUV cuando éste se encuentra realizando una misión bajo el agua, junto con el alto coste de la operación y del equipo, hacen que la detección y evitación de obstáculos sea una característica deseada en estas plataformas.

1.2 Tipos de Robots Submarinos

El término robot fue acuñado por el escritor checoslovaco Karel Kapel en su libro "*Rossum's Universal Robots*", procede de la palabra *Robota*, que significa esclavo o trabajador. Este concepto fue popularizado por el escritor Isaac Asimov en su novela *"Yo, Robot"* en 1950. Desde sus inicios hasta nuestros días, la robótica ha ido explorando diferentes campos, dando lugar a varias líneas de investigación. Este TFM se engloba dentro del marco de una de ellas: la robótica submarina. Es una rama de la robótica dedicada al estudio y diseño de máquinas para su utilización en medios acuáticos.

En las últimas décadas ha habido un gran interés por investigar las profundidades del fondo marino. Muchas de estas investigaciones se centran en el propósito de la explotación económica marítima unida también a la preservación del medio marino.

Debido al aumento de este interés, para la ciencia de la robótica submarina es una necesidad el desarrollo de vehículos submarinos operados remotamente o robots marinos que realicen tareas autónomas. La investigación en la robótica submarina no es una idea nueva, sino que se ha ido desarrollando dependiendo de las necesidades del ser humano y gracias a la motivación de los investigadores de descubrir nuevos entornos en las inmensidades de los océanos.

Una necesidad es la detección y evitación de obstáculos en el fondo marino ya que es un problema común para todos los tipos de URV (*Underwater Robot Vehicle*) o UUV (*Unmaned Underwater Vehicle*. La exploración cerca del fondo o en zonas costeras presenta este peligro, dado que las plataformas son caras y los entornos desconocidos se han perdido y/o hundido robots por culpa de golpes o de cuevas submarinas.

Debido a las grandes diferencias entre robots, estos se pueden dividir en el grado de autonomía, el tipo de misión a realizar o los sistemas de propulsión que posea el robot.

Pero principalmente se dividen dependiendo del nivel de autonomía del robot y se diferencian dos grandes tipos: los ROVs y los AUVs. Posteriormente, se han diseñado otro tipo de robots que nacen de la fusión de ambos, son los llamados IAUVs.

1.2.1 ROVs (Robots Submarinos Operados Remotamente)

Los robots submarinos operados remotamente, conocidos como ROVs de las siglas en ingles de *Remotely Operated Vehicle* [10], son aquellos robots que son controlados por un operario o piloto de forma remota mediante un sistema de cableado (cordón umbilical) donde recibe y envía información al usuario así como también se provee de energía. A través de una interfaz gráfica del sistema de control situado en la superficie, el usuario ejecuta los comandos que se deberían ejecutar en el robot.

Las estructuras submarinas como pueden ser instalaciones petroleras o de gas requieren una inspección frecuente que es realizada en su mayoría mediante ROVs. El uso de ROVs aumenta notablemente en instalaciones más alejadas de la costa debido a la fuerza de arrastre del agua, que hace que el vehículo sea menos manejable a pesar del aumento en el diámetro del cable. Por tanto, las operaciones con ROVs (ver figura 1.4) aumentan cuanto mayor es la profundidad donde es necesaria realizar una tarea de control o inspección a pesar del tiempo de respuesta del equipo y de las corrientes de arrastre. Cabe destacar que estos robots pueden fabricarse en dimensiones más reducidas que otros ya que el sistema de control está fuera de la estructura del robot pudiendo así reducirse su tamaño.

Figura 1.4: Robot ROV estudiando las profundidades del océano.
Fuente: Picssr

1.2.2 AUVs (Robots Submarinos Autónomos)

Una evolución del ROV ha sido el vehículo autónomo submarino, AUV(*Autonomous Underwater Vehicle*)[10]. Los robots submarinos autónomos son vehículos capaces de funcionar de manera independiente ya que poseen un sistema de control incorporado así como baterías para proveerse de energía. Debido a esto, el robot puede ser programado para realizar una trayectoria o tomar decisiones por sí mismo usando inteligencia artificial en función de los sensores sin la presencia de un operario, facilitando la respuesta y el trabajo a mayores profundidades. Los avances en sistemas de propulsión y fuentes de energía le dan a estos robots mayor duración en cuanto a distancias y tiempos de operación. Al no requerir de un operario, se reduce el riesgo de fatiga y se provee de una mayor repetibilidad experimental. Se usan principalmente para experimentos científicos y para realizar muestreos y toma de datos como se muestra en la figura 1.5.

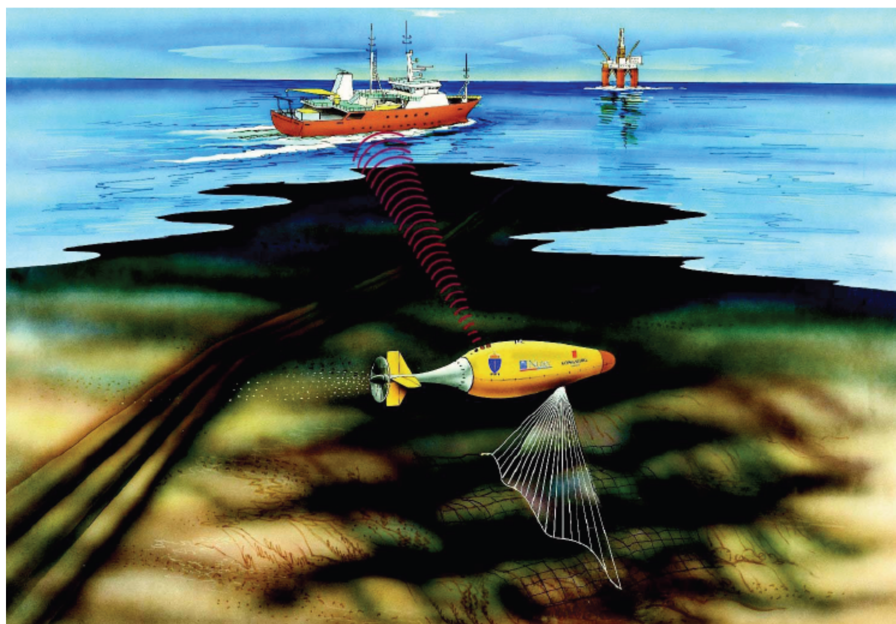

Figura 1.5: Robot AUV explorando el terreno mediante un sensor multihaz o multibeam sensor.
Fuente: Semantic Scholar

1.2.3 IAUVs (Intervention Autonomous Underwater Vehicle)

Una fusión de ambas tecnologías serían los robots IAUV (*Intervention Autonomous Underwater Vehicle*). Los IAUV son robots desarrollados para realizar una intervención en el medio de forma autónoma. Aunque en una primera instancia, estos robots realizan un muestreo de forma autónoma del terreno, es el usuario u operario que una vez recabada la información, decide el punto donde se realizaría la intervención supervisada.

1.3 Aplicaciones de los UUVs

Como hemos comentado en apartados anteriores, el océano posee gran cantidad de recursos por tanto es evidente que se pretende su explotación debido a esto existen diferentes aplicaciones para los vehículos submarinos. Entre las cuales destacan:

- **Aplicaciones de mantenimiento e puesta en marcha de estructuras marinas.** Como pueden ser las instalaciones petroleras submarinas, que requieren un mantenimiento e instalación de cables y tuberías necesarias para realizar correctamente su función.
- **Aplicaciones científicas.** Son aquellas cuya finalidad es profundizar en el conocimiento biológico, geológico y arqueológico de los mares y océanos.
- **Conservación y control de la fauna marina.** Mediante un AUV se puede en gran medida controlar de manera más exacta las poblaciones de organismos que viven debajo del mar. Además, se puede controlar la polución oceánica para la conservación del medio subacuático.
- **Asistencia a operaciones.** Estos vehículos autónomos son capaces de explorar zonas donde las cuales para un ser humano es peligroso realizar cualquier tarea. Su utilización es valiosa en tareas de salvamento y en naufragios.

1.4 Objetivo

El objetivo de este trabajo es la simulación de un sónar de imagen para una posterior implementación de un algoritmo de detección de obstáculos junto con un algoritmo de evitación de los mismos para un vehículo submarino autónomo (AUV), en nuestro caso basado en el Sparus II AUV diseñado por la Universidad de Girona. Con los datos recopilados del sensor simulado e incorporado a la simulación del Sparus II se generará un mapa del entorno de tal forma que se logrará el movimiento del vehículo de forma autónoma.

1.5 Motivación

La motivación de este TFM nace de la dificultad de realizar pruebas de detección y evitación de obstáculos usando robots reales en un entorno no controlado como es el océano. La implementación de un simulador de sónar de imagen de barrido mecánico (MSIS), junto con dos algoritmos, uno de detección y otro de evitación de obstáculos, permitirán analizar el comportamiento de la plataforma en entornos simulados de complejidad variable y controlada. A partir de los resultados obtenidos, se podrá reutilizar todo lo aprendido en detección y evitación de obstáculos en plataformas reales, con un mayor conocimiento de cómo debería reaccionar la plataforma a los entornos desconocidos.

1.6 Estructura del documento

Este documento se estructura en diferentes capítulos. A continuación se muestra una breve descripción de cada uno de ellos:

- Capítulo 1: Introducción, presenta algunos conceptos básicos sobre la robótica submarina así como los objetivos y motivación del trabajo.
- Capítulo 2: Fundamentos teóricos, en este punto se describen los principios fundamentales de los sensores, especialmente de un sensor MSIS como también los algoritmos teóricos de las técnicas de detección y evitación de objetos.
- Capítulo 3: Herramientas software utilizadas, en este apartado se describe el entorno de desarrollo y simulación. El entorno de desarrollo se explican los conceptos básicos de ROS y Cola2. Mientras que para la simulación se utiliza Uwsim y Rviz.
- Capítulo 4: Implementación, describe el procedimiento requerido para la simulación del MSIS. Además de la descripción de algoritmo de detección de obstáculos diseñado junto con la simulación de evitación de obstáculos basada en el paquete *move_base*.
- Capítulo 5: Resultados, se presentan dos simulaciones realizadas en dos escenarios diferentes, uno es un entorno que simula el fondo marino con rocas a modo de obstáculos y el otro es una escena con columnas que representa un rompeolas.
- Capítulo 6: Conclusiones y líneas de trabajo futuro, se enumeran los resultados de los experimentos obtenidos como también una conclusión final del trabajo junto con posibles líneas futuras de trabajo.

Capítulo 2

Fundamentos teóricos

En este capítulo de fundamentos teóricos se explican los conceptos básicos de los sensores y tipos. También se expone una descripción de algunos algoritmos de detección y evitación de obstáculos.

2.1 Definición de sónar

La palabra **SONAR** significa ***SO**und **NA**vigation and **R**anging* y por ella se entiende el método y/o el equipo necesario para determinar por medio del sonido la presencia, localización o naturaleza de objetos en las proximidades del sensor. El sónar es básicamente un sistema de navegación y localización parecido al radar, pero en este caso en vez de emitir señales de radiofrecuencia emite impulsos ultrasónicos. Se utilizan ultrasonidos porque bajo el agua las ondas electromagnéticas se atenúan muy rápidamente, mientras que la propagación de las ondas acústicas tiene un mayor alcance. El sónar está compuesto por un transmisor, un emisor, un receptor y un indicador. El transmisor emite un haz de impulsos ultrasónicos a través del emisor. Cuando las ondas acústicas chocan con un objeto se reflejan y forman una señal de eco que es captada por el receptor. El receptor amplifica la energía de las ondas del eco y genera una señal que es enviada al indicador, constituido normalmente por una pantalla en la que se ve el objeto en el que han rebotado las ondas(ver figura 2.1).

El alcance de las ondas acústicas está limitado por un gran número de factores siendo los más importantes la frecuencia de la onda y la efectividad del medio en el que se propaga la energía. Cuanto más baja es la frecuencia, mayor es el alcance que se obtiene. En medios acuáticos el alcance de las ondas acústicas puede variar entre las decenas y las centenas de metros. En algunos casos incluso pueden conseguirse alcances de kilómetros. Por su parte las ondas electromagnéticas raramente superan la decena de metros.

En primer lugar debemos diferenciar los conceptos de *ping* y *pulso*. Se conoce como *ping* a la señal acústica que envía el sónar de la cual se espera recibir el eco. Un *ping* puede estar formado por varios pulsos . Otro concepto importante es el de *scanline*. Se puede definir como el conjunto de intensidades registradas por el sensor en una dirección concreta. Algunos parámetros

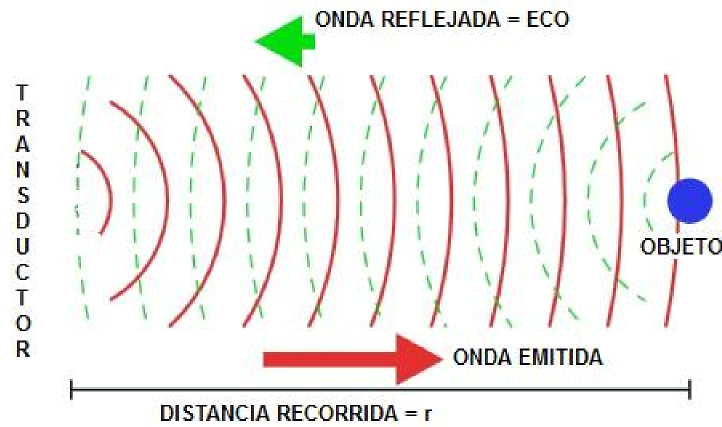

Figura 2.1: Principio de funcionamiento de un sonar.
Fuente: comofunciona.org

configurables a través del protocolo de comunicación del sónar son el rango, que se define como la distancia máxima a la que el dispositivo espera detectar obstáculos y el número de *bins*. La idea es discretizar el rango en un número determinado de pequeñas unidades que representen una distancia concreta. Estos intervalos son denominados *bins* por Tritech. El valor de la intensidad del eco para cada *bin* es devuelto en una trama de información que contiene todas las muestras correspondientes a una *scanline*. Se calcula el máximo tiempo de vuelo del pulso sabiendo el rango y la velocidad aproximada del sonido en el agua. Esta velocidad depende de diferentes factores del medio, como la presión y la temperatura. En condiciones generales, se utiliza 1550 m/s. Se obtiene la expresión siguiente 2.1 donde t_f es el tiempo que tarda un pulso en ir y volver la distancia correspondiente al rango e entre la velocidad del agua v:

$$t_f = e/v \tag{2.1}$$

El problema es que el tiempo t_f es una variable contínua y no es representable digitalmente por lo que es necesaria su discretizacion (ecuación 2.2). Según el número de bins:

$$t_b = t_b/n \tag{2.2}$$

donde n es el número de bins y t_b es el tiempo que transcurre en ir y volver en un mismo *bin*. De esta manera se consigue subdividir el tiempo de vuelo en intervalos. Sin embargo, para rangos muy elevados, es posible que este tb tenga un valor alto, siendo poco precisa la localización del obstáculo.

2.2 Tipos de sónares

Existen dos tipos de sónar: el activo y el pasivo. Un sónar pasivo cuenta con una base de datos sónica con la que compara las señales acústicas que recibe. Se utilizan para detectar tipos de barcos, acciones (velocidad de un barco, armamento) o incluso identificar algún barco en particular. Estos sensores no son usados en la ciencia oceanográfica dado que no pueden medir cómo es el entorno o cuál es su orografía. Por otra parte, un sónar activo no sólo se dedica a escuchar los sonidos bajo el agua, sino que emite pulsos acústicos y espera recibir su propio eco. Conocida la intensidad y modulación del pulso emitido, el eco retornado permite inferir medidas del fondo marino como la intensidad de eco y el tiempo que éste ha tardado en ir y volver. Dentro de la clasificación de sónares activos podemos encontrar tres clases: sónares multihaz (Multibeam Sonar o MBS), sónares de un haz (Single Beam Sonar o SBS) y sónares de barrido lateral (Side Scan Sonar o SSS).

En la figura 2.2 se muestran los sensores de MBS y SBS. Mientras que en la figura 2.3 se muestra el sensor de barrido lateral.

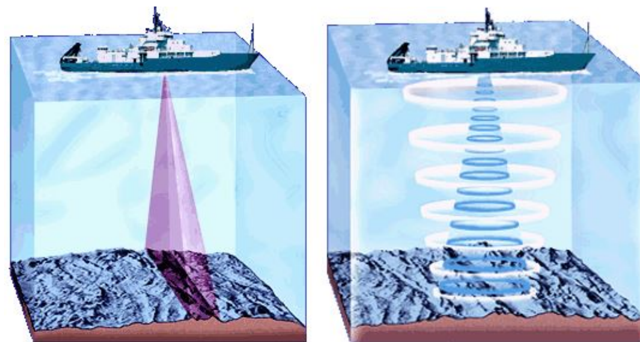

Figura 2.2: A la izquierda un sensor MBS y a la derecha un sensor SBS.
Fuente: comofunciona.org

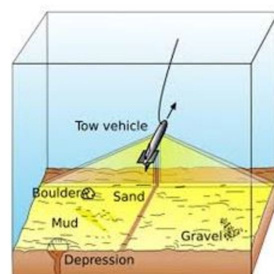

Figura 2.3: Sensor de barrido lateral (SSS).
Fuente: comofunciona.org

En todos estos casos, dependiendo de la apertura trasversal a la trayectoria pueden pasar a llamarse sónares de imagen o *Imaging Sonars*, ya que son capaces de crear una proyección del eco recibido en forma de imagen. Un ejemplo de éstos es el sónar de la casa Tritech llamado Miniking. Éste es un sónar de imagen de barrido mecánico o *Mechacnically Scanned Imaging*

Sonar (MSIS). El transductor está montado encima de una plataforma giratoria controlada por ordenador. Para conseguir una imagen del entorno debe enviar un pulso, esperar su eco y girar un paso de vuelta hasta realizar una vuelta completa.

La imagen de la figura 2.4 se representa el barrido mecánico del terreno por el sensor en el vehículo Sparus II AUV.

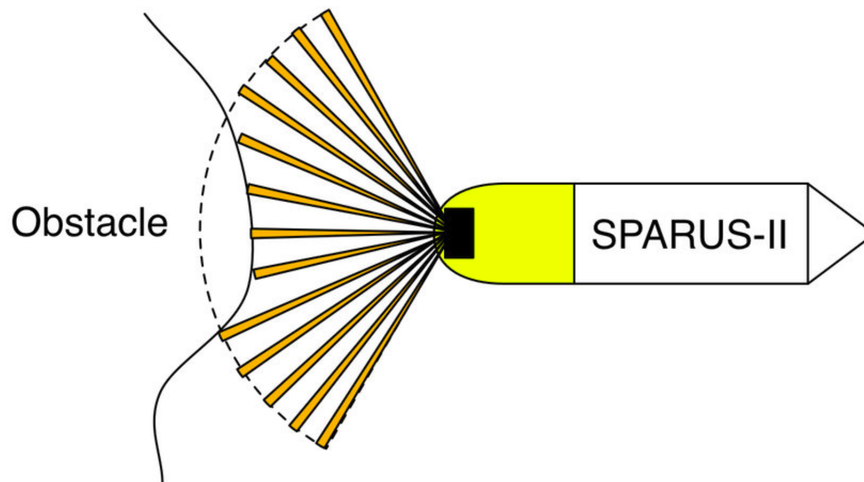

Figura 2.4: Representación del sónar MSIS en el robot Sparus II AUV.
Fuente: researchgate

2.2.1 Miniking Imaging Sonar

Este sónar es un dispositivo que trae un nuevo concepto en los productos de detección debido a su alto rendimiento y su pequeño tamaño le permite poder acoplarse a una gran variedad de portadores acuáticos (véase la figura 2.5).

Figura 2.5: Sónar Miniking de Tritech.
Fuente: Tritech

Este dispositivo se controla a través de un PC, de un ROV o de un AUV , una de las características a destacar es la gran cantidad de funcionalidades que ofrece el software. Éste permite configurar el escáner de diferentes formas y realizar exploraciones 360º o restringidas a un sector angular

configurable gracias a la incorporación de un motor al que está adosado el transductor dentro del cabezal. Se utiliza en sistema genéricos de medida de distancia, para aplicaciones de evitación de obstáculos y reconocimiento de objetos. A pesar de tener una gran cantidad de funciones es de fácil manejo y de una gran fiabilidad. El funcionamiento ya se ha comentado anteriormente [5], se basa en enviar pulsos y escuchar los ecos producidos a raíz de su interacción con el entorno. Estos ecos llegan al sensor en forma de señal ultrasónica, a partir de esta medida se podrá determinar la distancia a la que se encuentra un obstáculo. La clave es que emite el ultrasonido y registra las intensidades de los ecos recibidos a intervalos regulares. Así se tienen intensidades ordenadas por tiempo o, lo que es lo mismo, por distancia. Y el proceso se repite girando el cabezal un paso cada vez.

Las características del sónar Miniking se muestran en la tabla 2.1

Frecuencia de operación	675 kHz
Amplitud vertical del haz	40º
Amplitud horizontal del haz	3.0º
Rango de operación (alcance)	60 metros
Nivel de fuente	210 dB
Ancho de banda del sistema	35 kHz
Rango	360º continuos o 180º frontales con movimiento de izquierda a derecha
Velocidades	Normal, rápido y súper rápido
Fuente de alimentación	12, 24, 48 voltios a 6 VA
Peso en el aire	1,1 kilogramos
Peso en el agua:	0,5 kilogramos
Material	Aluminio
Profundidad máxima soportada	1000 metros
Temperatura de operación:	-10ºC a 35ºC
Temperatura de almacenamiento	-20ºC a 50ºC

Tabla 2.1: Tabla de especificaciones del sónar Miniking.

2.3 Detección y evitación de obstáculos

2.3.1 Algoritmos tipo Bug

Este tipo de algoritmos son de una ejecución bastante sencilla ya que se basan en el sistema que tienen los insectos para detectar obstáculos y superarlos. Consisten en calcular la ruta más rápida y directa para dirigirse al objetivo y, una vez detectado un obstáculo, ir siguiendo su contorno hasta que se pueda seguir el camino fijado anteriormente. En el caso de que se recorra todo el contorno del obstáculo, se determinará que es imposible llegar al objetivo. Existen varios tipos de algoritmos Bug englobados en una misma categoría llamada familia, ya que todos se basan en algoritmos anterior aplicando ciertas mejoras u otras decisiones.

Bug 1

Esta primera versión de los algoritmes Bug fue descrita por primera vez por V. Lumelsky y A. Stephanov [6]. Consiste en encaminar el robot hacia el objetivo y en caso de detectar un obstáculo, el robot almacena esa posición en su memoria interna. Luego, el robot va siguiendo el contorno del obstáculo evaluando cada posición para determinar cuál es la más cercana al objetivo. Una vez recorrido el contorno del obstáculo, el robot retorna a la posición detectada como más cercana al objetivo y una vez ahí, vuelve a encaminarse hacia el objetivo. A continuación, definimos su funcionamiento en pseudocódigo:

Algoritmo 1 Tipo Bug 1

1: $q_0 = q_{inicio}$
2: $i = 0$
3: **para** $q! = q_{goal}$ o $q_h! = obstculo$ **hacer**
4: Ir hacia $q_{goa}l$
5: **fin para**
6: **si** $q == q_{goal}$ **entonces**
7: Salir algoritmo
8: **fin si**
9: **para** $q! = q_{goal}$ o $q! = qh$ **hacer**
10: Seguir el contorno del obstáculo, almacenando la posición más corta al objetivo.
11: **fin para**
12: **si** $q == q_{goal}$ **entonces**
13: Salir algoritmo
14: **fin si**
15: Ir hacia objetivo
16: **si** llegar a q_goal es imposible **entonces**
17: Salir algoritmo con error
18: **sino**
19: $i = i + 1$
20: continuar
21: **fin si**

En la siguiente imagen de la figura 2.6 podemos ver un ejemplo de este algoritmo.

El algoritmo Bug 1 es completo y nos proporciona una ruta hacia objetivo evitando los obstáculos pero el camino resultante puede que sea muy largo.

Bug 2

Algoritmo creado también por V. Lumelsky y A. Stephanov [6] incorporando algunas mejoras respecto al algoritmo Bug 1. En este caso se calcula una línea imaginaria entre la posición inicial del robot y del objetivo (m-line) y se hace que el robot la siga hasta que encuentra un obstáculo. En ese momento el robot empieza a navegar siguiendo el contorno del obstáculo hasta que re-encuentra la línea (m-line) y la vuelve a seguir hasta llegar a objetivo.

A continuación el pseudo-código:

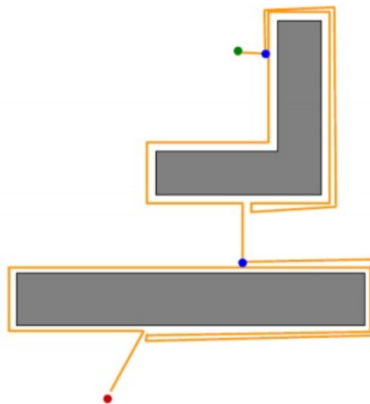

Figura 2.6: Representación gráfica del Bug 1.
Fuente: Ampliación Robótica Industrial UIB

Algoritmo 2 Tipo Bug 2

1: $q_0 = q_{inicio}$
2: $i = 0$
3: **para** $q! = q_{goal}$ o $q_h! = obstculo$ **hacer**
4: Ir hacia $q_{goa}l$ a través de la $m - line$
5: **fin para**
6: **si** $q == q_{goal}$ **entonces**
7: Salir algoritmo
8: **fin si**
9: **para** $q! = q_{goal}$ o $q! = qh$ o se encuentre $m - line$ o el camino a q_{goal} sea posible **hacer**
10: Seguir el contorno del obstáculo.
11: **fin para**
12: **si** $q == q_{goal}$ **entonces**
13: Salir algoritmo
14: **fin si**
15: Ir hacia objetivo
16: **si** q_h es alcanzado **entonces**
17: Error
18: **sino**
19: $q_i = m$
20: $i = i + 1$
21: continuar
22: **fin si**

En la imagen de la figura 2.7 podemos ver un ejemplo de este algoritmo. El algoritmo Bug 2 mejora los tiempos y reduce el camino que tiene que seguir el robot para llegar a objetivo, comparándolo con el algoritmo Bug 1 aunque existen casos que no siempre es así.

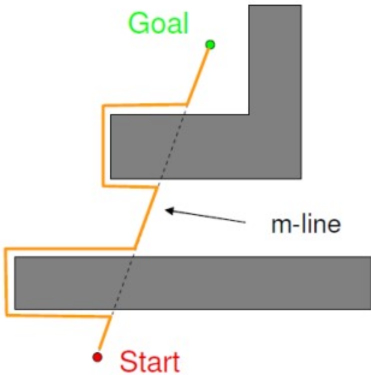

Figura 2.7: Representación gráfica del Bug 2.
Fuente: Ampliación Robótica Industrial UIB

En la imagen de la figura 2.8 se representan los dos algoritmos Bug. En color naranja se dibuja la trayectoria seguida por el Bug 1 y en color rojo el camino del Bug 2. Se observa claramente como el Bug 2 es mucho mejor que Bug 1.

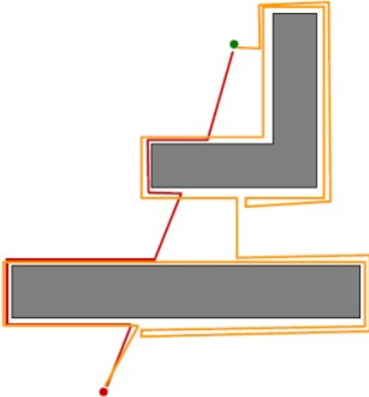

Figura 2.8: Representación gráfica del Bug 1 vs Bug 2.
Fuente: Ampliación Robótica Industrial UIB

2.3.2 Algoritmo VFH

El algoritmo VFH fue diseñado por J. Borenstein y Y. Koren [1] para poder detectar los obstáculos mientras el robot se dirige hacia el objetivo, todo en tiempo real.

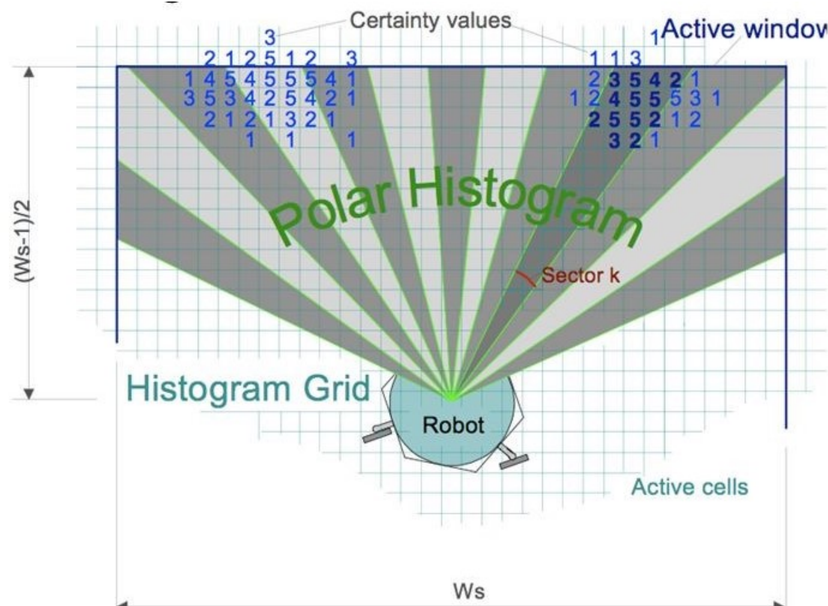

Figura 2.9: Histograma polar con división de celdas.
Fuente: Ampliación Robótica Industrial UIB

Este algoritmo se basa en la creación de un histograma bidimensional a partir de las lecturas de sus sensores para representar los obstáculos (ver figura 2.10). Este histograma se divide en celdas a las cuales se les asigna un valor que representa la probabilidad de encontrar el objeto en esa posición a partir de las mediciones realizadas por los sensores (ver imagen 2.9). Luego se transforma el histograma a un histograma polar de una sola dimensión (ver figura 2.10) representando así la probabilidad de encontrar un obstáculo según la orientación del robot. Por último se le aplica una función de coste (ecuación 2.3) a las diferentes rutas posibles para llegar al objetivo seleccionando la de menor coste:

$$G = a \cdot direccion_objetivo + b \cdot orientacion_robot + c \cdot direccion_anterior \tag{2.3}$$

Siendo a, b y c parámetros internos del robot para poder seleccionar la mejor ruta posible. El inconveniente de este algoritmo es que no tiene en cuenta las restricciones mecánicas del robot y puede llegar a posiciones donde le será imposible salir y encontrar el camino al objetivo (mínimos locales)

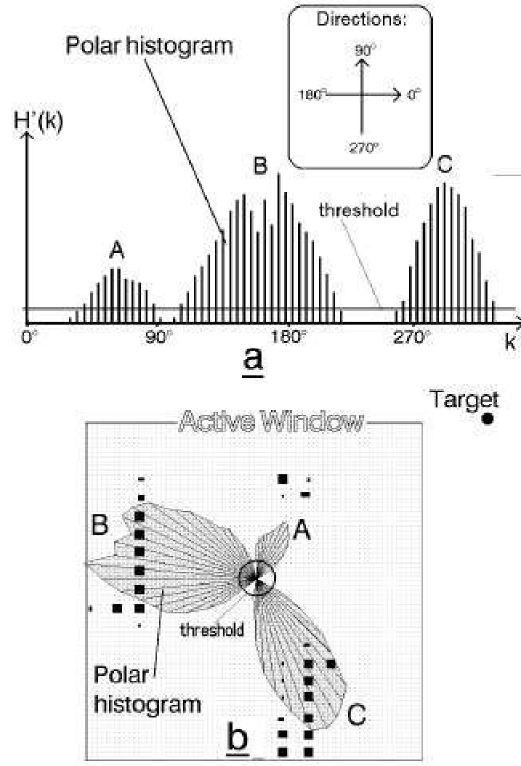

Figura 2.10: a) Representación de un histograma polar, b) Representación de un histograma bidimensional. Fuente: Imagen extraída de [2]

2.3.3 Algoritmo de campos de potencial

Para la realización del algoritmo de campos de potencial, se modela el robot como una partícula sometida a una serie determinada de fuerzas (ver imagen figura 2.12). De forma que el objetivo aplica una fuerza atractiva hacia el robot y los obstáculos una fuerza repulsiva. Por lo tanto el AUV se moverá en función de la dirección que resulta de la suma de estas fuerzas. La trayectoria del robot vendrá determinada por (2.4 y 2.5):

$$F(q) = \nabla U(q) = \nabla U_{att}(q) - \nabla U_{rep}(q) \tag{2.4}$$

Dónde:

$$U_{att} = \frac{1}{2}k_{att} \cdot \rho^2_{goal}(q) = \frac{1}{2}k_{att} \cdot (q - q_{goal})^2$$
$$F_{att} = -\nabla U_{att}(q) = k_{att} \cdot (q - q_{goal}$$
$$U_{rep}(q) = \frac{1}{2}k_{rep}(\frac{1}{\rho(q)} - \frac{1}{\rho_0})^2$$

$$F_{rep}(q) = \begin{cases} k_{rep} \cdot (\frac{1}{\rho(q)} - \frac{1}{\rho_0})\frac{1}{\rho^2(q)} \cdot \frac{q-q_{goal}}{\rho(q)}, & \text{if } \rho(q) \leq \rho_0 \\ 0, & \text{if } \rho(q) \geq \rho_0 \end{cases} \tag{2.5}$$

Considerando que ρ es la distancia al objeto y ρ_0 la distancia de la cual no se considera un obstáculo. Y las variables k son asignadas por el usuario para realizar un correcto modelaje del robot . En la figura 2.11 se muestra la trayectoria seguida por un robot, en la cual se encuentra

un obstáculo por el camino representado como un punto negro, se observa como se generan unas fuerzas de atracción y de repulsión. Estas fuerzas se observan en la figura 2.12, donde el obstáculo genera unas fuerzas de atracción mientras que el robot genera fuerzas de repulsión siendo la fuerza resultante necesaria para evitar el obstáculo.

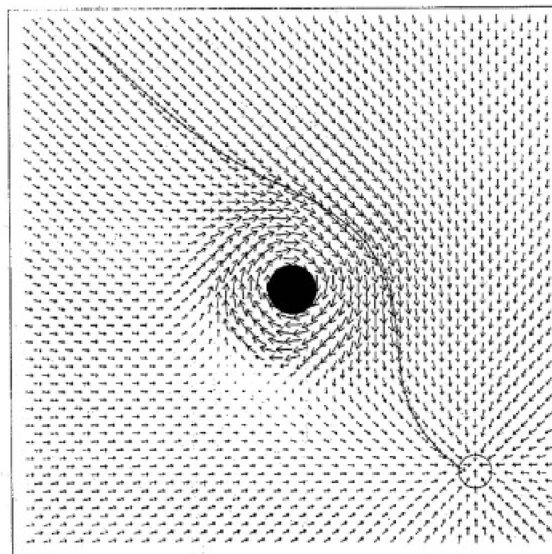

Figura 2.11: Trayectoria seguida por una partícula.
Fuente: Ampliación Robótica Industrial UIB

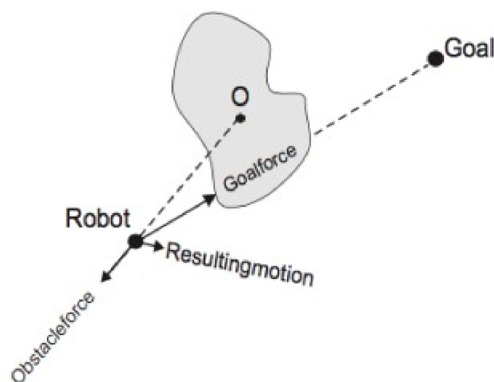

Figura 2.12: Fuerzas de atracción y repulsión de un objeto respecto al robot.
Fuente: Ampliación Robótica Industrial UIB

2.3.4 Algoritmo DWA

Este algoritmo es un planificador local basado que solo tiene en cuenta los parámetros dinámicos de nuestro robot para alcanzar un objetivo. Utiliza los ejes cartesianos (x,y) y los convierte en velocidades (v,w) para el robot, eso quiere decir que solo trabajamos en el espacio de velocidades.

El algoritmo DWA (*Dynamic Window Approach*) cumple dos condiciones para la búsqueda: calcular un espacio válido de velocidades y elegir la velocidad óptima. Para calcular el espacio de velocidades para la toma de la decisión del camino a elegir, el algoritmo DWA tiene en cuenta tres factores:

- Solo se consideran trayectorias circulares únicamente definidas por la velocidad tangencial y angular (v,w).
- Las velocidades admisibles serán aquellas que generen una trayectoria segura, es decir, permitiendo al robot detenerse ante un obstáculo.
- Dada este conjunto de velocidades, el robot las reduce en una ventana dónde las velocidades sean alcanzables en un intervalo pequeño de tiempo.

Para elegir la velocidad óptima, se maximiza el valor de la siguiente función (2.6):

$$G(v,w) = \sigma(\alpha \cdot heading(v,w) + \beta \cdot dist(v,w) + \gamma \cdot vel(v,w)) \tag{2.6}$$

De manera que la velocidad elegida será aquella que maximice la orientación (*heading*) hacia el objetivo, que la distancia (*dist*) entre la trayectoria del robot y el obstáculo sea mayor ya que indicaría que no hay obstáculos cerca; y por último, se preferirá que la velocidad (*vel*) para avanzar sea la más elevada posible.

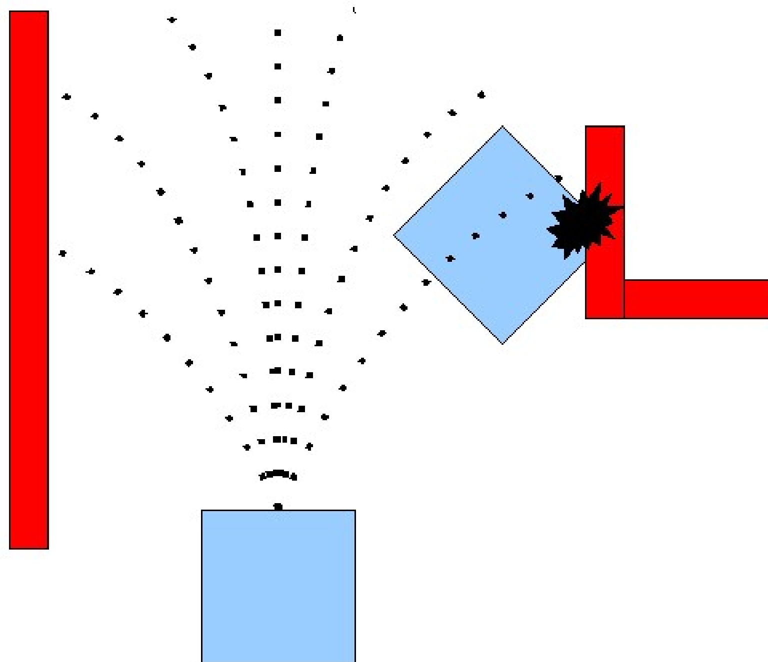

Figura 2.13: Comportamiento del algoritmo DWA.
Fuente: Ampliación Robótica Industrial UIB

El algoritmo en pseudocódigo es el siguiente:

Algoritmo 3 Algoritmo DWA

1: Obtenemos la velocidad ideal entre la posición de nuestro robot y el objetivo.
2: $Allowable_V$ = ventana de velocidades generadas a partir del modelo dinámico.
3: $Allowable_W$ = ventana de velocidades generadas a partir del modelo dinámico.
4: **para** *w en Allowable_W* **hacer**
5:   **para** *w en Allowable_V* **hacer**
6:     Dist = distancia al obstáculo más próximo.
7:     Heading = orientación hacia el objetivo.
8:     Vel = Velocidad ideal - Velocidad actual.
9:     Calculamos G
10:     **si** $G > G_0$ **entonces**
11:       Asignamos la trayectoria dada por la V y W actual.
12:       $G_0 = G$
13:     **fin si**
14:   **fin para**
15: **fin para**

Cabe destacar que este es el algoritmo que usaremos en nuestro proyecto ya que nos permite evitar obstáculos de una forma segura para nuestro AUV.

Capítulo 3

Herramientas Software Utilizadas

En el presente capítulo se describen todas las herramientas software empleadas en el trabajo, tanto el entorno de simulación como el entorno de desarrollo.

3.1 Entornos de desarrollo

3.1.1 ROS

ROS (*Robot Operating System*) es una plataforma de desarrollo abierto que provee toda una serie de librerías y servicios que ayudan significativamente a la creación de software para robótica. Aún siendo similar a otras estructuras de desarrollo, ROS es una de las más útiles gracias a la creciente comunidad de desarrollo, la facilidad de intercambio de software y su capacidad de dar soporte a robots como los que existen actualmente. Para facilitar la comprensión y la generación de software con ROS, se ha utilizado una estructura visual. Se utilizan grafos para visualizar la conexión entre diferentes módulos del proyecto:

- *Nodos*
  Los nodos son programas que ejecuta el computador para cumplir con una función. Cada nodo que se ejecuta tiene un nombre único que los identifica. Un sistema de control puede tener varios nodos. Un beneficio de utilizar nodos es que son individuales y los fallos que ocurren son fácilmente contenibles y detectables. Cabe destacar la facilidad con la que se pueden ejecutar, hasta tal punto de que con un archivo especial (archivo *launch*) es posible ejecutar un gran número de nodos en el mismo instante. En este archivo también es posible dar valor a variables que usaremos dentro de los nodos sin tener que compilarlos, esta función es muy útil cuando se tienen que probar diferentes valores en un mismo problema.

- *Mensajes*
  Los nodos utilizan mensajes para comunicarse entre ellos. Un mensaje es una estructura de datos. Son soportados tanto los tipos primitivos standard (entero, booleano ...) como los arrays y constantes.

- *Topics*
  Es el mecanismo mediante el cual los nodos intercambian mensajes. Los mensajes se enrutan de dos formas, mediante un publisher o un subscriber. Un nodo envía un mensaje mediante su publicación a un topic determinado. Si un nodo necesita un tipo de dato en particular debe suscribirse al topic en cuestión para recibirlo. Puede haber varios editores y suscriptores simultáneos en cada topic. El topic es un nombre que se utiliza para identificar el contenido del mensaje.

- *Servicios*
  Este mecanismo permite implementar la comunicación punto a punto entre nodos. Los servicios están conformados en dos partes, el mensaje de petición y el de respuesta y se basan en un funcionamiento de cliente y servidor.

- *Master*
  Es el nodo principal de ROS, encargado de hacer las conexiones entre los nodos que requieran comunicarse. La figura 3.1 muestra un ejemplo de comunicación entre nodos. Los nodos están representados por los círculos, mientras que la flecha indica el topic que los comunica.

Figura 3.1: Ejemplo de comunicación entre nodos
Fuente: Repositorio de paquetes ROS

3.1.2 Component Oriented Layer-based Architecture for Autonomy, COLA2

COLA2 es un sistema de control para vehículos autónomos submarinos, su nombre viene de *Component Oriented Layer-based Architecture for Autonomy*[8]. Desarrollada por la Universidad de Girona, esta arquitectura se estructura en tres campos: de reacción, de ejecución y de misión. COLA2 se puede usar en conjunto con UWSim y RVIZ, permitiéndonos ver una representación en directo de toda la misión durante su ejecución. El paquete incluye los modelos de los robots Sparus II y Girona 500.

3.2 Entorno de simulación: UWSIM y RVIZ

3.2.1 UWSIM

UWSim (*UnderWater SIMulator*)[9] es el software base del trabajo de código libre. Nació bajo la necesidad de probar y simular diferentes algoritmos de control y percepción antes de su implementación en el mundo real, desarrollado por la universidad Jaume I. Este simulador nos ofrece utilidades de gran importancia a la hora de simular nuestros proyectos:

- *Entorno configurable:* posibilidad de creación de escenarios propios de tal forma que se asemejen lo máximo posible a nuestro entorno real. La definición de nuestro escenario se puede realizar fácilmente por medio de un fichero .xml, donde se define, aparte de nuestro modelo

3D, los sensores a utilizar y la interfaz que se necesite de ROS para la comunicación con el simulador.. Cargar modelos 3D propios siempre que se carguen en un formato valido para OSG (*OpenSceneGraph*). Además, nos permite modificar estados físicos como la superficie del océano, oleaje, transparencia del agua, entre otros parámetros.

- *Soporta diferentes robots:* incluye ciertas características independientes del robot en uso, de forma que podamos añadirlas a nuevos modelos de robots. Ademas, UWSim nos permite trabajar con más de un robot a la vez.
- *Diferentes sensores:* incluye un total de doce sensores distintos (cámara, GPS, Imu, DVL, Multibeam, sensor de fuerza... entre otros).

UWSim (ver imagen Figura 3.2) está integrado dentro de ROS. Gracias a ello, podemos establecer un mapa que nos permita crear una red de comunicación tanto interna para la ejecución del propio proyecto como externa, para poder comunicarnos con el simulador. COLA2 se apoya en el paquete *auv_msgs* de ROS.

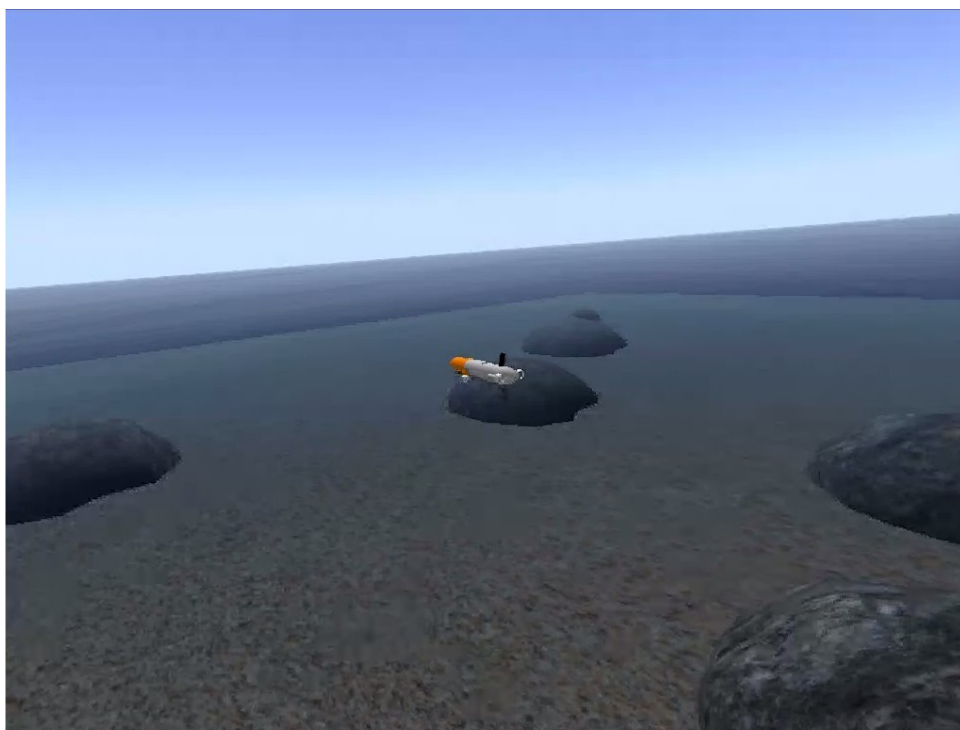

Figura 3.2: Visualización de un entorno de simulación empleado en las pruebas con UWSim
Fuente: Simulador UWSIM

Los distintos tipos de interfaz que UWSim ofrece son los siguientes:

- *ROSOdomToPAT* :lee mensajes de odometría de ROS y mueve el vehículo según corresponda.
- *PATToROSOdom* :publica mensajes de odometría a partir de los datos de posición y velocidad del vehículo.
- *WorldToROSTF:* publica la posición y velocidad de cada objeto presente en el escenario.
- *ArmToROSTF:* interfaz de brazo robótico que publica los estados de las uniones.

- *ROSJointStateToArm:* lee los mensajes de ROS de los estados de las uniones y mueve el brazo según corresponda.
- *VirtualCameraToROSImage:* crea un emisor ROS de imágenes de la cámara que se especifique.
- *ROSImageToHUD:* se subscribe a un emisor de imágenes y genera la cámara dentro de UWSim.
- *ROSTwistToPAT:* lee mensajes de ROS de tipo Twist para establecer la velocidad del vehículo.
- *RangeSensorToROSRange:* emite las distancias le??das por el sensor virtual.
- *ROSPoseToPAT:* lee mensajes de posición de ROS y mueve el vehículo según corresponda.
- *ImuToROSImu:* publica lecturas de la IMU.
- *PressureSensorToROS:* emisor para los sensores de presión.
- *GPSSensorToROS:* lee la información de los sensores GPS y los publica.
- *DVLSensorToROS:* crea un emisor para los dispositivos DVL.
- *RangeImageSensorToROSImage:* utiliza el emisor de imágenes de ROS para publicar imágenes de profundidad
- *multibeamSensorToLaserScan:* transforma las lecturas del sensor multihaz en mensajes tipo LaserScan y los publica.
- *contactSensorToROS:* publica el estado de colisión.
- *ForceSensorToROS:* publica la información del sensor de fuerza.
- *ROSPointCloudLoader:* lee mensajes de nubes de puntos de ROS y los representa en 3D.

3.2.2 RVIZ

RVIZ es una herramienta de propósito general que permite visualizar en un espacio tridimensional muchos de los elementos de un robot como las imágenes de la cámara, nubes de puntos tridimensionales, y barridos láser (ver imagen figura 3.3). Muestra los datos de los sensores en un sistema de coordenadas común a la del robot implementado, con lo que permite visualizar con facilidad lo que ve el robot y puede identificar cualquier desajuste de sensores.

RVIZ también puede interactuar con la información de la biblioteca tf para mostrar toda la información del sistema de coordenadas de los sensores desde diferentes perspectivas. Destaca como ventaja que se trata de una herramienta muy importante ya que permite visualizar que ocurre en el robot en cada momento en busca de errores o en nuestro caso en simulaciones del entorno.

Como se ha comentado anteriormente, se usará un modelo basado en el Sparus II AUV, dicho modelo se basa en la arquitectura de control Cola2 que junto con el simulador UWSim y Rviz forman el entorno ideal de trabajo para la experimentación del sensor y de los algoritmos de evitación.

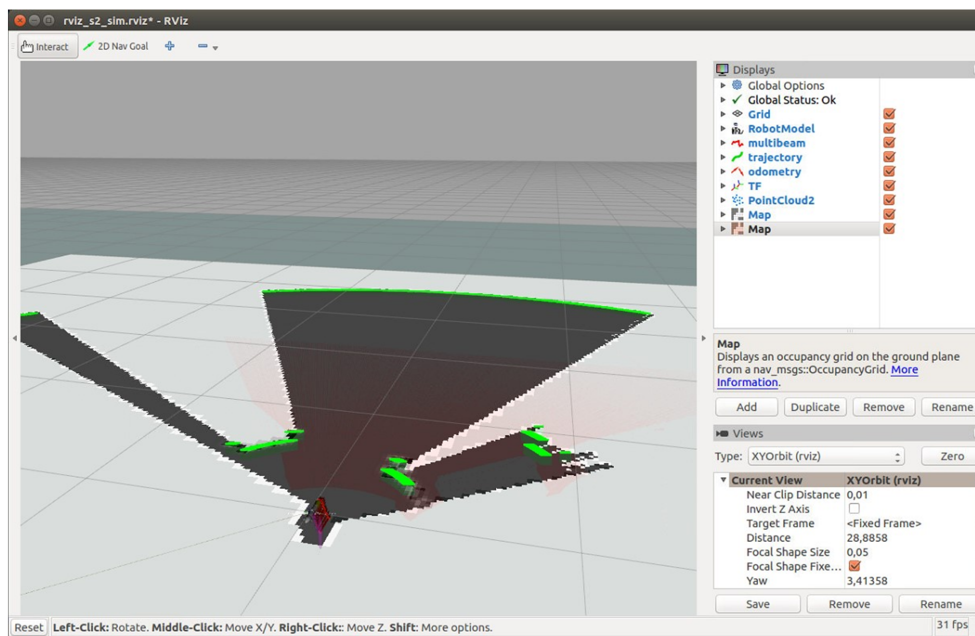

Figura 3.3: Simulación del entorno con Rviz se observa como el AUV realiza un escaneo de la zona detectando tres obstáculos en el escenario

Fuente: Simulador RVIZ

Capítulo 4

Implementación

En este capítulo se describe la implementación de un sensor de barrido mecánico (MSIS) incorporado en un vehículo submarino autónomo Sparus II. Se explica el diseño de un algoritmo de detección de obstáculos creado a partir de las muestras tomadas por el escaneado el sensor, además de la posterior descripción de un algoritmo de evitación basado en un paquete de ROS llamado *move_ base*. En la figura 4.1 se representa un diagrama de bloques que explica las relaciones que se usan para la implementación de todo el conjunto del TFM realizado. Es decir, se muestran los variables empleadas como los pasos a realizar para el diseño del algoritmo de evitación de obstáculos.

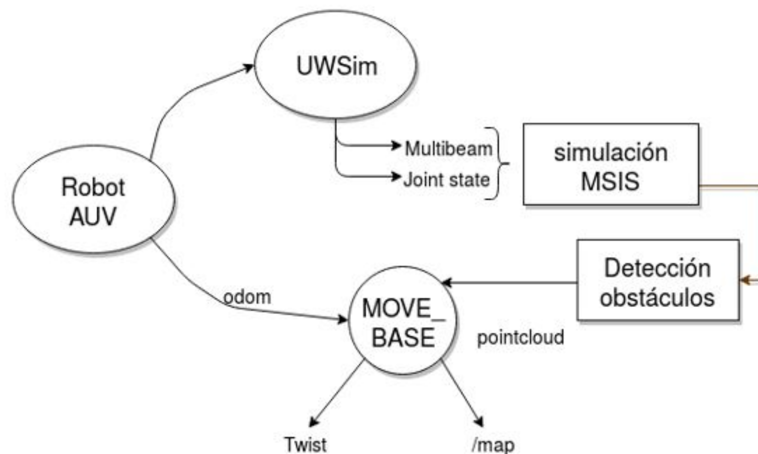

Figura 4.1: Diagrama de bloques del funcionamiento global del AUV.
Fuente: elaboración propia

4.1 Sparus II AUV

El Vehículo Submarino Autónomo SPARUS II (ver figura 4.2) fue concebido en el Centro de Investigación en Robótica Subacuática (CIRS) de la Universidad de Girona. La primera versión se diseñó en 2010 para participar en un concurso europeo, organizado por CMRE en La Spezia (Italia). El robot ganó la competición y, desde entonces, ha colaborado en varios proyectos de investigación. En 2013, se terminó una nueva versión del robot, SPARUS II, que en la actualidad se está comercializando.

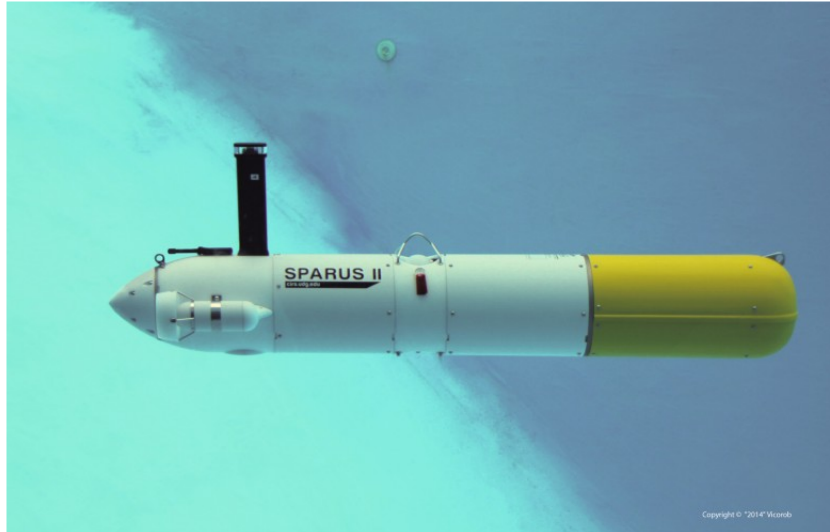

Figura 4.2: SPARUS II AUV.
Fuente: CIRS

SPARUS II AUV es un vehículo ligero e hidrodinámicamente eficiente con una autonomía prolongada en aguas poco profundas (200 metros). Combina el rendimiento en forma de torpedo con capacidad de flotación estacionaria. Es fácil de implementar y operar ya que el usuario final puede personalizar y utilizar una arquitectura de software abierta, basada en ROS, para la programación de diferentes misiones. Su flexibilidad y facilidad de uso hacen que el vehículo sea una plataforma multidisciplinaria con la capacidad de adaptación a aplicaciones industriales, científicas y académicas. SPARUS II implementa COLA2 [8] como arquitectura de control.

4.2 Simulación de un MSIS usando ROS, UWSIM y COLA2

En este apartado se explica la simulación de un *Mechacnically Scanned Imaging Sonar* (MSIS) basado en los parámetros disponibles que ofrece UWSim. Este simulador posee diferentes tipos de interfaces, comentadas en el apartado 3.2.1. a partir de las cuales se consigue implementar un MSIS. La idea de la simulación de este sensor ocurre debido a la falta de una estructura ya implementada para el AUV. Por este motivo, el primer paso en este TFM es la creación de un sensor acoplado al vehículo viable para el escaneo en barrido para su posterior adquisición de datos.

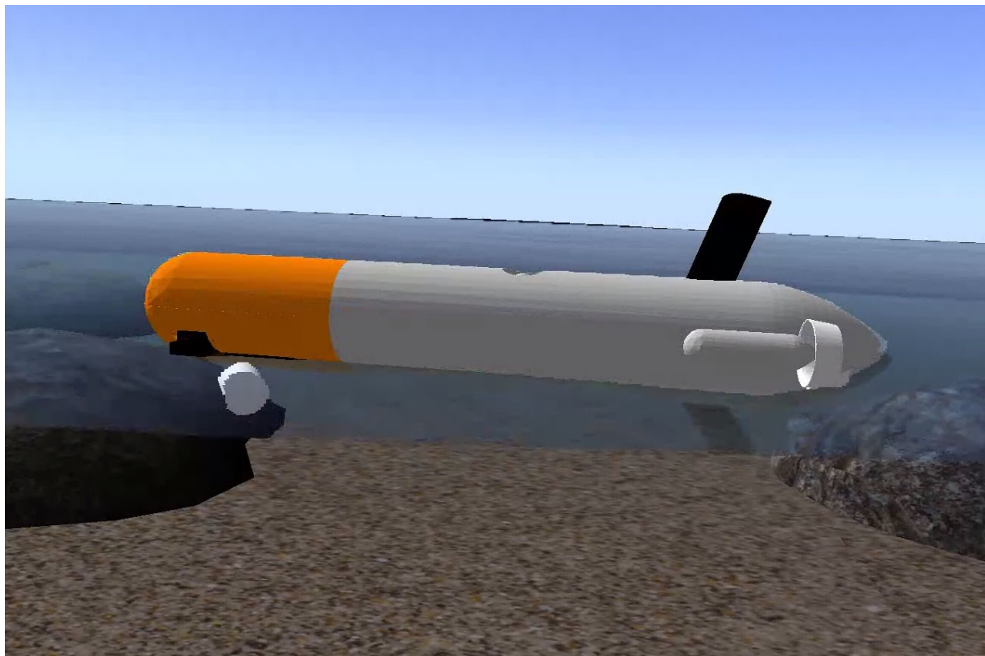

Figura 4.3: Imagen generada por UWSim donde se muestra el sensor de barrido en la parte inferior delantera del robot Sparus II.

Fuente: elaboración propia

Para ello como se puede ver en la figura 4.3, a partir de la estructura base del Sparus II se crea un sensor que se acopla en la parte delantera inferior del vehículo. Esta tarea se consigue a partir del archivo de tipo URDF (*Unified Robot Description Format*), es un archivo de tipo XML (Extensible Markup Language) que representa el modelo del robot, en nuestro caso representa el modelo del Sparus II AUV. En este archivo llamado sparus.urdf (ver código en Anexo D) se añade la representación gráfica del sensor de barrido. Este sensor es representado por un cilindro, el cual se une a la base del robot como se ha visto en la figura 4.3.

Una vez diseñado el MSIS a partir de los enlaces y uniones modulares se permite reproducir la rotación del sensor para que trabaje como un sensor de barrido. Esto se consigue cambiando el ángulo de rotación de la unión o *joint* para ir escaneando el terreno incrementalmente con un ángulo de apertura definido.

Para este cometido se debe crear una función que simule el barrido mecánico del MSIS a partir de los topics (nodos que intercambian mensajes mutuamente) de UWSim: *ROSJointStateToArm* y *multibeamSensorToLaserScan*.

El pseudocódigo es el siguiente:

Algoritmo 4 Algoritmo de barrido mecánico del MSIS

1: Ordenar distancias del láser.
2: **si** dirección = *true* **entonces**
3: Incrementar ángulo de escaneado.
4: **si** posición ≥ ángulo max escaneado **entonces**
5: dirección = false
6: **fin si**
7: **sino**
8: Incrementar ángulo de escaneado.
9: **si** posición ≤ ángulo min escaneado **entonces**
10: dirección = false
11: **fin si**
12: **fin si**

Los robots se basan en un sistema de coordenadas y ángulos de Euler como el mostrado en la figura 4.4. Siendo φ roll (rotación en el eje X) , θ pitch (rotación en el eje Y) y ψ yaw (rotación en el eje Z). Nuestro sensor escanea en el plano XY mientras rota en el plano Z, esto se muestra en la figura 4.5.

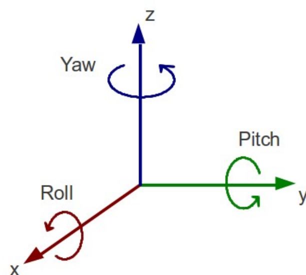

Figura 4.4: Ejes de coordenadas y ángulos de Euler.
Fuente: elaboración propia

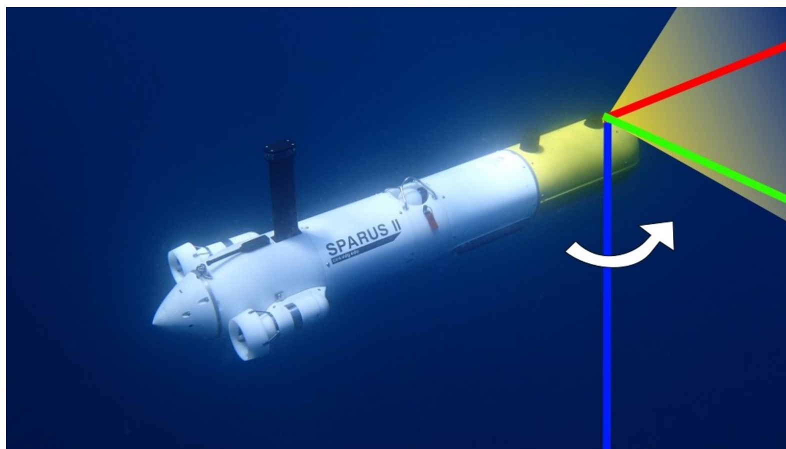

Figura 4.5: Representación del sistema de coordenadas del sensor.
Fuente: elaboración propia

Como se ha explicado en el Capítulo 2, el sensor escanea el entorno emitiendo pulsos ultrasónicos que proporcionan perfiles de intensidad del eco del área escaneada. Pero el problema que se haya en este trabajo es que el sensor devuelve las distancias en las cuales se produce cada eco y no devuelve las intensidades del eco.

Por tanto, a partir de las distancias obtenidas del MSIS, se crean dos funciones para conseguir la detección de obstáculos en el entorno. Se entiende que se detecta un objeto cuando la intensidad del eco es mayor, es decir, cuánto mayor es el ángulo de incidencia mayor es el eco de la intensidad y por tanto, más probabilidad de que se encuentre un obstáculo. Como dato, la sombra producida tras de un obstáculo se detecta como un valor nulo de intensidad y puede confundirse con el fondo. Del tamaño de la sombra se puede obtener una aproximación de cómo de grande es el obstáculo encontrado.

La primera función es la utilizada para el cálculo del ángulo de incidencia θ que se obtiene a partir de las distancias obtenidas del sensor que son las distancias a las que se produce un eco. Se calcula el ángulo de incidencia para cada una de las distancias obtenidas. Para entender el procedimiento de cálculo del ángulo de incidencia θ, se muestran los parámetros del sónar en la figura 4.6.

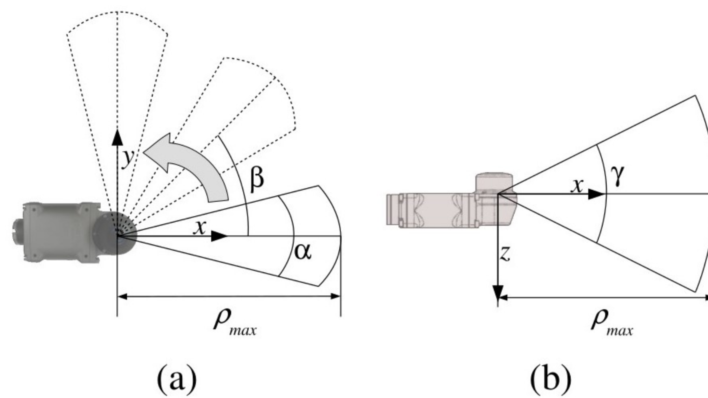

Figura 4.6: . Parámetros del MSIS. a) Plano XY. b) Plano XZ
Fuente: [3]

Los valores de estos parámetros son los mostrados en la Tabla 4.1:

ρ_{max}	α	β	γ
50 m	3º	1,8º	40º

Tabla 4.1: Características específicas del Miniking.

Como se puede ver en la figura 4.7, se debe tener en cuenta que cuando se recibe un eco en la parte superior se puede obtiene el rango máximo del sensor (ya que no detecta el fondo terrestre). A no ser, que se encuentre un obstáculo flotante como puede ser una boya. En cambio, la parte inferior se empiezan a detectar los valores de eco resultantes del fondo y de los obstáculos que se encuentren por el terreno.

En base a la figura 4.7, se observa con más claridad cómo se ha procedido a calcular el ángulo de incidencia y la definición de los parámetros del sensor.

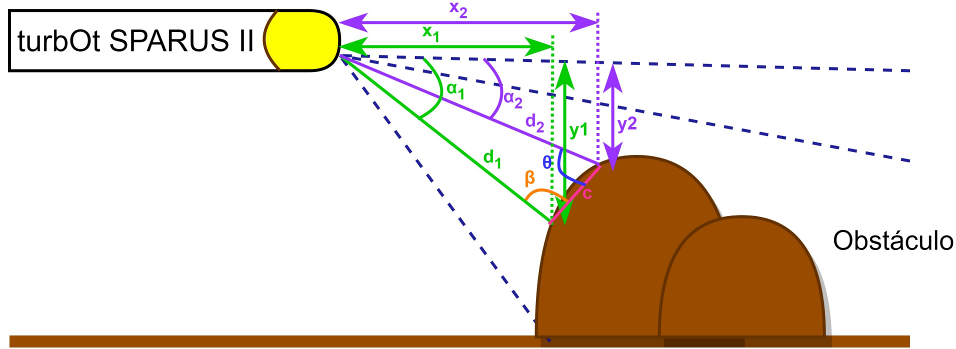

Figura 4.7: Representación gráfica del entorno con los parámetros necesarios para el cálculo del ángulo de incidencia θ

Fuente: elaboración propia.

Donde definimos como:

- ángulo de escaneado α
- d_1 y d_2 las distancias del robot al objeto
- el ángulo de incidencia θ
- c es la distancia de unión de los valores d_1 y d_2
- x distancia en el eje de abscisas del robot al objeto.
- y distancia en el eje de ordenadas del robot al objeto.

Se calcula el ángulo de incidencia θ aplicando el Teorema del Coseno, donde se obtiene la ecuación 4.2:

$$\frac{c}{\sin(\alpha)} = \frac{d_1}{\sin(\theta)} = \frac{d_2}{\sin(\beta)}$$
$$\alpha = \alpha_1 - \alpha_2 \tag{4.2}$$

despejando el ángulo de incidencia θ se tiene la ecuación 4.3

$$\theta = \arcsin(\sin(\alpha) \cdot \frac{d_1}{c}) \tag{4.3}$$

donde la distancia c se puede calcular como se define en la ecuación 4.4:

$$c = \sqrt{(x_1 - x_2)^2 - (y_1 - y_2)^2} \tag{4.4}$$

donde x e y se calculan a partir de geometría básica (ecuación 4.5):

$$x = d \cdot \sin(\alpha)$$
$$y = d \cdot \cos(\alpha) \tag{4.5}$$

Se describe el pseudocódigo del algoritmo de la función del Cálculo del ángulo de incidencia θ.

Algoritmo 5 Algoritmo de Cálculo del ángulo de incidencia

1: Dimensionar parámetros (x, y, c, θ)
2: Inicialización de ángulo de escaneado $\gamma = -20^o$
3: Ordenar distancias (vector Laserscan.ranges)
4: **para** $i\ in\ range(longitud(Laserscan.ranges))$ **hacer**
5: Incremento ángulo de escaneado
6: Cálculo de x e y
7: **fin para**
8: **para** $i\ in\ range(longitud(Laserscan.ranges) - 1)$ **hacer**
9: Cálculo de c
10: Cálculo de ángulo de incidencia θ
11: **fin para**

Seguidamente de obtener el ángulo de incidencia θ, se aplica la función creada para calcular la intensidad de cada eco para contar el námero de haces en cada paso. El paso es la distancia de rango máxima del sensor entre el número de haces deseado. Al menos, es recomendable conseguir un haz para un paso.

El pseudocódigo para el cálculo de haces en cada rango de distancia como la normalización de los valores de la intensidad resultante se ven reflejados a continuación:

Algoritmo 6 Algoritmo de Contador del beams

1: Inicialización parámetros $(reg_beams, reg_beams_n orm, cont_beamsxdist)$
2: $num_sep = (scan.range_max\ ?\ scan.range_min)/beam_footprint$
3: **para** $i\ inrange(longitud(Laserscan.ranges) - 1)$ **hacer**
4: Cálculo de intensidades para cada bloque de distancias x_dist
5: **si** $x < num_sep - 1$ **entonces**
6: Incrementar $cont_beamsxdist[x_dist]$
7: $reg_beams =$ Sumatorio del seno del angulo de incidencia[i]
8: **fin si**
9: **fin para**
10: **para** $i\ inrange(num_sep)$ **hacer**
11: Normalizar reg_beams a reg_beams_norm de 0 a 255
12: **fin para**

Para calcular la intensidad se suman los senos del ángulo de incidencia para cada paso definido. Es decir, en nuestro caso hemos definido la variable $beam_footprint$ con valor de 1 metro. Al menos, es recomendable conseguir un haz para cada metro. Por tanto, se suman todos los senos del ángulo de incidencia que se encuentran en el mismo paso y se obtiene como resultado

un registro con los valores de las intensidades de cada haz de ultrasónido. Este valor no está normalizado así que se normaliza de 0 a 255.

4.3 Detección de obstáculos

El siguiente objetivo en este TFM es la detección de obstáculos a partir de la simulación del MSIS. Para la detección de obstáculos se ha diseñado una función encargada de su localización. Este procedimiento se ha dividido en dos pasos: uno el cálculo rangos de intensidades nulas y otro la comprobación de obstáculo así como el cálculo del máximo pico de intensidad.

En el cálculo de rango de ceros, como bien sabemos la detección de un objeto genera un pico de intensidad a menor distancia del robot. Es decir, la intensidad es mayor si el ángulo de incidencia es más cercano a 90º.

Algoritmo de Cálculo de Ceros: se crea un vector que detecte rangos de ceros. Es decir, cuando se detecta un 0 se rellena el vector con el valor 1 mientras el resto de valores encontrados sustituyen por 0. De esta manera, se calcula del rango de distancias en las que el sensor no detecta un valor nulo.

Algoritmo de Verificación de Obstáculo: obtenidos los rangos de valores en que se detecta un obstáculo se comprueba que el valor del pico máximo se encuentra entre valores ceros, es decir, entre sombras y por tanto es un obstáculo. Se genera un vector que devuelve en número de obstáculos que se encuentran por cada paso del sensor. Así como la posición a la que se encuentran.

También se ha probado la detección de picos con una librería implementada por [4]. Esta función detecta valores de picos, máximos en base a unos parámetros que se definen. Estos son la altura mínima definida para ser considerado pico (mph) y la separación entre picos (mpd).

Además, se ha diseñado un visor de obstáculos (ver Anexo C). En este visor se representa un histograma polar en una dimensión de las intensidades y se marca si se detecta un pico de obstáculo. Esta gráfica se generá de forma automática para cada paso del sensor. En la imagen de la figura 4.8 se muestra una gráfica de los valores de intensidad que se obtienen en el momento inicial de escaneo para una escena determinada, en este caso la escena representa una cueva submarina. Se puede observar para este caso como los picos pronunciados son obstáculos en el entorno, en este caso el pico máximo representa el punto marcado en rojo en el UWSim. Mientras que el punto marcado en azul se detecta como un pico de menor intensidad donde se produce una sombra muy alargada hasta el límite de la escena

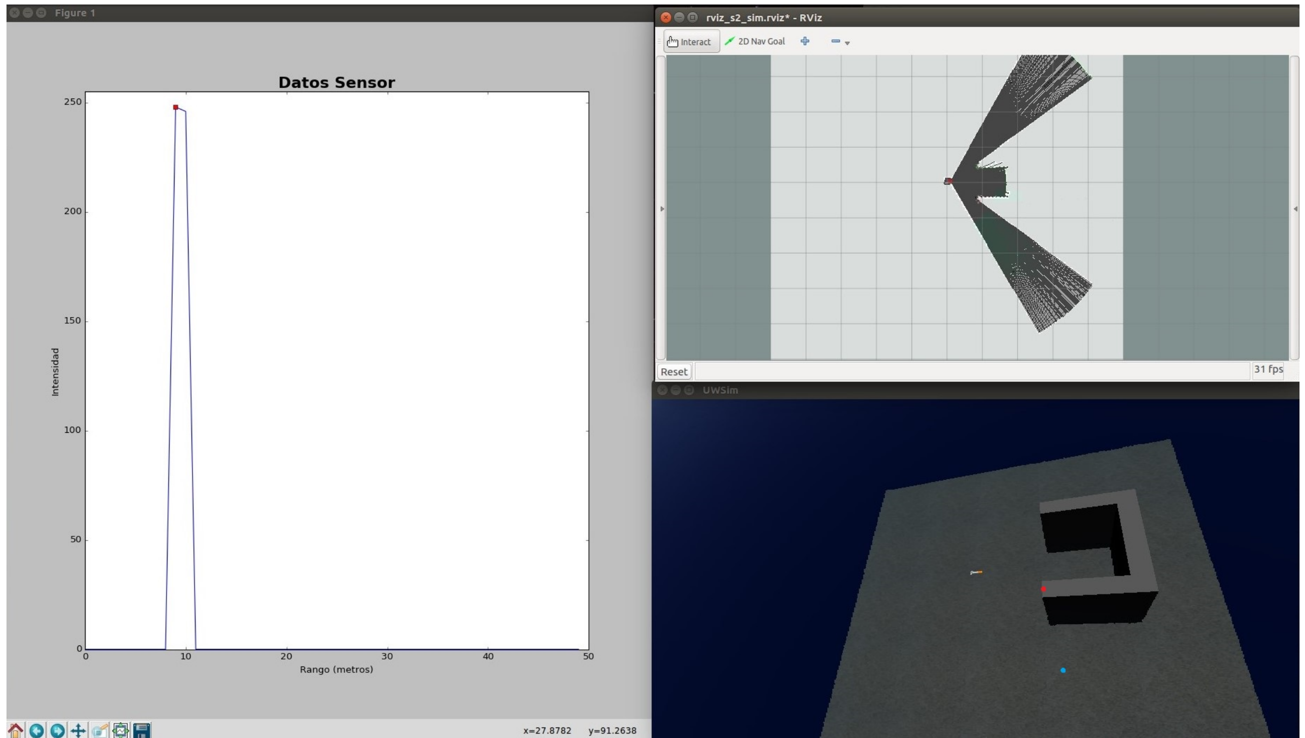

Figura 4.8: Se muestra la simulación del entorno, a la izquierda se ve el histograma polar. A la derecha, las imágenes de Rviz y UWSim.

Fuente: elaboración propia.

En la gráfica de la figura 4.8 se puede observar para este caso como los picos pronunciados son obstáculos en el entorno, en este caso el pico representa el punto marcado en rojo en el UWSim. Mientras que el punto marcado en azul se detecta como un pico de menor intensidad donde se produce una sombra muy alargada hasta el límite de la escena.

4.4 Evitación de obstáculos usando *move_base*

Para cumplir el objetivo de evitar los obstáculos es necesario, por un lado, que el robot sea capaz de reconstruir un escenario de forma que sea comprensible para el vehículo. Por tanto, primero debe ser capaz de detectar los obstáculos, como se ha comentado en el apartado anterior 4.2 se emplean una serie de funciones para su detección. Una vez detectados los objetos en el mapa, se puede proceder a aplicar alguno de los algoritmos de evitación que se han explicado en el capítulo 2, para este TFM se selecciona el paquete *move_base* que se basa en un algoritmo DWA.

Para poder utilizar el paquete *move_base*, se ha tenido que adaptar el código generando un nodo que publique un mensaje de tipo Pointcloud2. Este objetivo se consigue transformando el mensaje de tipo Laserscan a un mensaje tipo Acousticbeam y de este tipo a Pointcoud2. A partir de la nube de puntos guardada con el valor de la posición de los obstáculos, es necesario crear un nuevo nodo llamado twist2bvr.py (ver en Anexo B) para el control de la velocidad del AUV en la simulación y así poder suscribirse al nodo "*/cmd_vel*"que se encarga de transmitir

las velocidades de movimiento al vehículo. Las pruebas realizadas para la comprobación de este algoritmo se describen en el siguiente capítulo.

Capítulo 5

Resultados

En este capítulo se muestran los resultados obtenidos de las simulaciones generadas a partir de dos escenas distintas. Estas escenas representan diferentes entornos que se pueden encontrar en el medio marino, como puede ser un entorno con rocas en el fondo de la superficie o como puede ser un rompeolas, el cual en nuestro caso es representado por una serie de columnas alineadas. Se obtienen distintas gráficas que representan el movimiento de forma autónoma del AUV en el mapa reconstruido a partir de los datos tomados del MSIS.

Estas pruebas engloban todo el proceso especificado en la figura 4.1. Por tanto, si estos experimentos son satisfactorios se deduce que la simulación del MSIS ha sido correcta junto con los algoritmos de detección de obstáculos estudiados.

Cabe destacar que las gráficas resultantes se han generado a partir del programa Matlab, donde mediante la extracción de datos a un fichero .csv y la lectura de los nodos de odometría del robot y la ruta calculada del punto inicial al punto final se dibujan diversas gráficas para analizar los resultados y mostrar las conclusiones del TFM, en el siguiente capítulo.

5.1 Prueba 1. Escenario con piedras

La primera prueba se basa en la simulación de un entorno en el cual los objetos/ obstáculos son las rocas que se hayan en el fondo de la superficie. A partir del nodo del planificador del mapa local se obtiene un mapa de ocupación de celdas, donde las celdas marcadas en oscuro son los obstáculos que detecta el MSIS del AUV. Sobre este mapa, se pintan las trayectorias seguidas por el planificador DWA del move_base y la odometría del AUV desde un punto inicial hasta un punto final. El control del selección del punto final se marca sobre el mapa generado por Rviz mediante la herramienta de navegación 2D. Y el punto inicial es el punto de origen del robot. Después para ver la evolución en el tiempo se crea otra gráfica con la comparación del movimiento en los ejes de coordenadas por separado, es decir, en el eje de abscisas y en el de ordenadas.

En la figura 5.1 se observa como se han detectado varias piedras en el fondo. Si procedemos a señalar el punto final en el mapa el robot debe seguir la trayectoria calculado por el planificador (rojo). Se aprecia la brusquedad de la ruta calculada debido a que el planificador va al momento obteniendo cuál es el mejor camino en cada momento. En cambio, la odometría del robot es más suave ya que en la navegación no se pueden realizar movimientos tan bruscos. Se ve como pasa alrededor de un obstáculo, ya que es la ruta más corta para llegar su destino. Otro dato a comentar es la posición final del vehículo, se puede ver como no están situados en el mismo punto exacto sino que hay un margen de deriva. Este parámetro ha sido ajustado a máximo 2 metros de distancia.

Figura 5.1: Mapa del entorno con la ruta calculada por el planificador y seguida por el robot.
Fuente: elaboración propia

En las figuras 5.2 y 5.3 se muestra la comparación en el tiempo del movimiento. El valor del tiempo de encuentra en epoch, es un sistema para la descripción de instantes de tiempo que se define como la cantidad de segundos transcurridos desde la medianoche del 1 de enero de 1970.

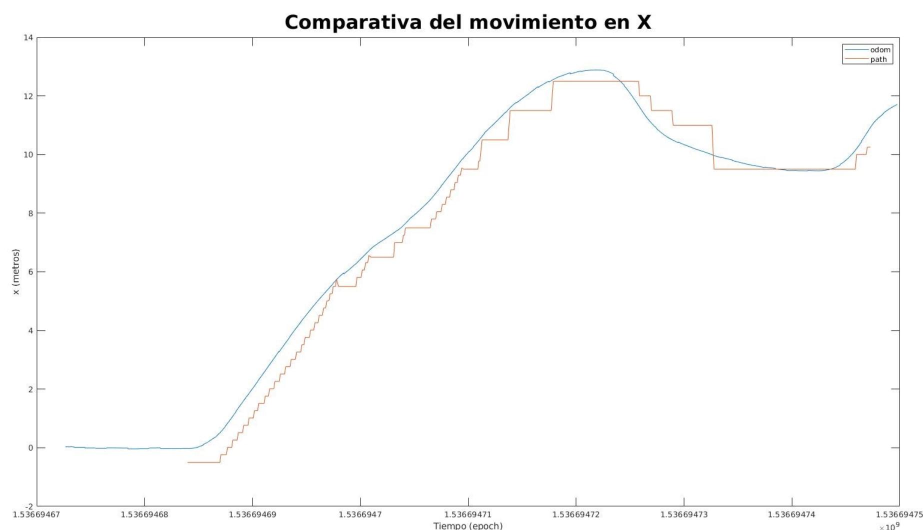

Figura 5.2: Comparación entre el movimiento real del robot y calculado por el planificador en el eje X.
Fuente: elaboración propia

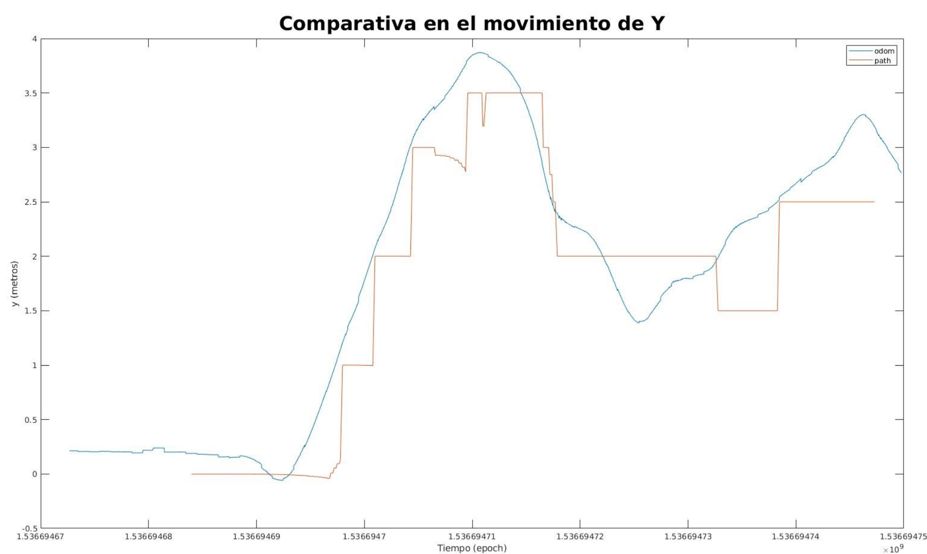

Figura 5.3: Comparación entre el movimiento real del robot y calculado por el planificador en el eje Y.
Fuente: elaboración propia

En conclusión, analizadas todas las figuras de este apartado se concluye que el vehículo es capaz de desplazarse de forma autónoma evitando obstáculos, sin desviarse mucho de la trayectoria originalmente calculada por el planificador DWA de move_base.

5.2 Prueba 2. Escenario con columnas

La segunda prueba se basa en la simulación de un rompeolas, los obstáculos son las columnas que se colocan alineadas como se muestra en la imagen de la figura 5.4. Para esta escena, ha sido necesaria la creación de un archivo XML, donde se ha diseñado un objeto columna en formato OSG (OpenSceneGraph).

Figura 5.4: Rompeolas situado en el Puerto de Sant Feliu de Guixols.
Fuente: Google Maps

La figura 5.5 representa el mapa de ocupación de los obstáculos detectados, se observa como se detectan cuatro columnas. Estas columnas no son perfectas ya que mientras el robot sigue su ruta al punto final va funcionando a la vez el MSIS pero no le da tiempo a generar el barrido completo de lateral a lateral detectando así solo una parte cada vez. Por lo demás, se aprecia como se ha seguido la ruta evitando las columnas encontradas.

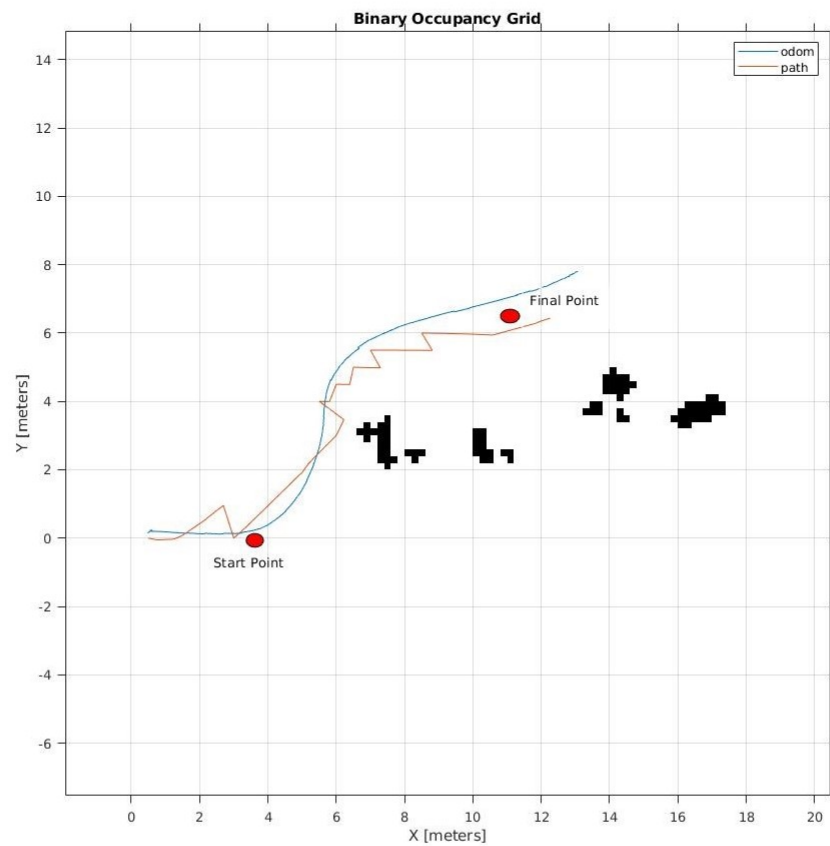

Figura 5.5: Mapa del entorno con la ruta calculada por el planificador y seguida por el robot.
Fuente: elaboración propia

En las figuras 5.6 y 5.7 se observa el correcto funcionamiento del AUV ya que tiene comportamientos muy parecidos en ambos ejes de coordenadas al comportamiento descrito por el algoritmo de evitación.

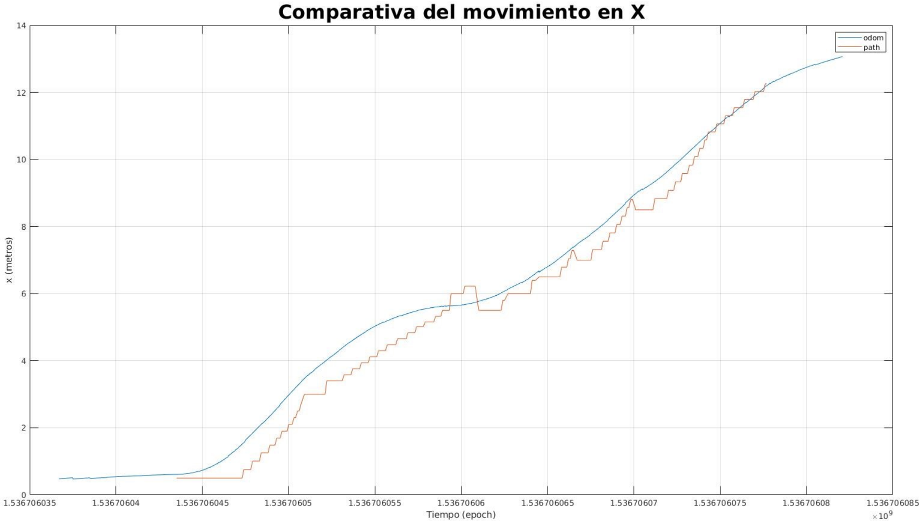

Figura 5.6: Comparación entre el movimiento real del robot y calculado por el planificador en el eje X.
Fuente: elaboración propia

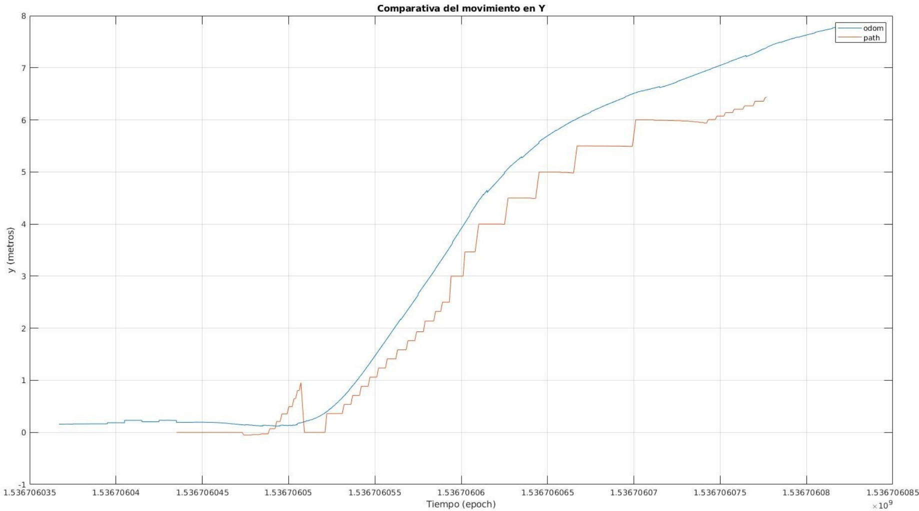

Figura 5.7: Comparación entre el movimiento real del robot y calculado por el planificador en el eje Y.
Fuente: elaboración propia

Capítulo 6

Conclusión

6.1 Conclusiones y líneas de trabajo futuro

A lo largo del presente TFM se ha implementado el diseño de un sensor de barrido mecánico o también descrito en el documento como MSIS a partir de las lecturas ofrecidas por un sensor multihaz. Como se ha explicado se crea una función que se encarga de generar el barrido del sensor a través del movimiento de una unión de la base del robot al sensor. La idea de la simulación de este sensor ocurre debido a la falta de una estructura disponible. Su creación no es inmediata, ya que se deben plantear distintas funciones como el cálculo del ángulo de incidencia, el cálculo de la intensidad del eco contando el número de haces que se encuentran en cada muestreo y la normalización del cálculo de las intensidad.

A partir de este punto, se puede pasar a la implementación de las técnicas de detección y evitación de obstáculos explicadas en el Capítulo 2. Para lograr la detección de obstáculos se ha diseñado un algoritmo de detección de picos a partir de las sombras que se generan en las lecturas. Sabiendo que un obstáculo es considerado tal cuando se encuentra entre ceros, es la base del algoritmo implementado. Mientras el otro algoritmo de detección de obstáculos tiene el mismo objetivo que es encontrar los picos que se encuentran entre ceros pero aplicando un filtro de ajuste de distintos parámetros como son la altura mínima para ser considerado pico y la separación entre picos. Se han ejecutado estos dos algoritmos con un resultado satisfactorio ya que se ha podido comprobar con el visor de obstáculos creado que se marca el punto máximo del pico.

Una vez comprobados estos pasos, se pasa a la evitación autónoma de obstáculos para este fin se ha seleccionado el uso del paquete move_base de ROS. Después de la creación de un nodo del control de las velocidades del robot, se realizan una serie de pruebas seleccionando un punto objetivo de misión para comprobar si el robot se desplaza correctamente por el mundo evitando colisionar con los objetos que se encuentre. Las pruebas experimentadas confirman el correcto funcionamiento del AUV. Se observan pequeñas desviaciones causadas por la fuerza de arrastre del agua, se podría ajustar la velocidad para que el robot se mueva más lentamente pero con un grado menor de deriva. Pero en general, los resultados en los dos escenarios usados han sido

buenos logrando uno de los objetivos del trabajo que es la evitación de obstáculos a medida que el robot va generando el mapa.

Cabe destacar que a diferencia de otros algoritmos de control de evitación de obstáculos, es necesario un tiempo inicial de análisis del entorno antes de poder moverse al punto objetivo seleccionado. Ya que nuestro AUV sólo navega por las celdas descubiertas del mapa de ocupación generado del entorno.

Para simular el funcionamiento de ambos algoritmos se ha utilizado el sistema operativo ROS junto el simulador UWSim y Rviz. El sistema operativo de ROS incorpora un conjunto de librerías y herramientas ya creadas que facilitan enormemente el desarrollo de aplicaciones para robots. En este trabajo, de todas esas librerías/herramientas, se ha hecho uso de los paquetes move_base y auv_msgs. Para la simulación del SPARUS II se han utilizado las librerías de Cola2_s2, Cola2_core y para la simulación del MSIS la librería miniking_ros del grupo SRV (Systems, Robotics and Vision) de la Universidad de las Islas Baleares.

Se proponen algunas líneas de trabajo que han surgido a lo largo del TFM como son:

- Implementación de un algoritmo de evitacción de obstáculos basado en campos de potencial para así poder comparar con la eficiencia del algoritmo del paquete move_base.
- Reconstrucción 3D del entorno a partir de la generación de diferentes mapas en 2D pero a diferentes cotas, así formando una estructura tridimensional del entorno. Ya que el vehículo al trabajar en un plano paralelo a la superficie, inicialmente desconoce la posición del obstáculo, es decir, no sabe si el obstáculo se encuentra en el fondo o flotando.

En conclusión, este TFM ha servido para introducirse más profundamente en el mundo de la robótica submarina aprendiendo el sistema operativo de ROS junto algunas aplicaciones como el UWSim y Rviz que en conjunto han resultado ser un entorno de trabajo perfecto para el correcto desarrollo de este TFM.

Parte II

Presupuesto

Presupuesto

En este apartado se detallarán los costes de ejecución del proyecto teniendo en cuenta los gastos que ello conlleva, como serían las remuneraciones de sus participantes y los costes del material usado. Primero de todo, se tienen en cuenta los salarios de las personas implicadas en el proyecto. En concreto, el salario del supervisor del proyecto y del técnico del proyecto que se pueden ver en la tabla **6.1**

	Sueldo bruto (€)	Sueldo neto (€)	Coste hora (€)
Sueldo Supervisor	40500	26325	19,18
Sueldo Técnico	33750	21937,50	15,98

Tabla 6.1: Tabla de salarios de participantes del proyecto.

Al haber utilizado mayormente software gratuito, no se ha cargado el coste de ningún programa, pero si el coste del hardware ya que ha sido comprado para la ejecución del proyecto. Como indica la tabla **6.2**.

	Coste (€)
Ordenador Sobremesa	1980

Tabla 6.2: Coste de material.

Una vez definidos los costes, procedemos a la distribución del presupuesto. Para la elaboración de dicho documento, se ha utilizado un programa específico para el control de costes y generación de presupuestos llamado Presto 8.8. A continuación, se adjuntan los archivos pertenecientes al presupuesto, en concreto, el cuadro de precios descompuestos, el estado de mediciones valorado y el resumen del presupuesto donde sean cargado los costes de beneficio industrial, gastos generales e IVA.

CUADRO DE DESCOMPUESTOS

Diseño y implementación de algoritmos de evitación de obstáculos

CÓDIGO	CANTIDAD	UD	RESUMEN	PRECIO	SUBTOTAL	IMPORTE
CAPÍTULO 1 Proposición de proyecto						
1.1		**ud**	**Definición de proyecto a realizar**			
			Definición de proyecto a realizar junto con los detalles principales de las tareas a ejecutar. Se detallará el alcance del proyecto y los materiales a utilizar para la correcta ejecución.			
MOT	10,000	h	Hora de supervisor de proyecto	19,18	191,80	
			TOTAL PARTIDA			**191,80**

Asciende el precio total de la partida a la mencionada cantidad de CIENTO NOVENTA Y UN EUROS con OCHENTA CÉNTIMOS

CÓDIGO	CANTIDAD	UD	RESUMEN	PRECIO	SUBTOTAL	IMPORTE
1.2		**ud**	**Hardware utilizado**			
			Suministro de hardware usado para la ejecución del proyecto.			
PC	1,000		Ordenador Sobremesa	1.980,00	1.980,00	
			TOTAL PARTIDA			**1.980,00**

Asciende el precio total de la partida a la mencionada cantidad de MIL NOVECIENTOS OCHENTA EUROS

CÓDIGO	CANTIDAD	UD	RESUMEN	PRECIO	SUBTOTAL	IMPORTE
CAPÍTULO 2 Tareas previas						
2.1		**ud**	**Configuración y puesta a punto**			
			Configuración del hardware para poder realizar el trabajo requerido así como la instalación de todos los programas necesarios relacionados con el proyecto (ROS, Matlab, etc)			
MOP	16,000	h	Hora de técnico	15,98	255,68	
MOT	3,000	h	Hora de supervisor de proyecto	19,18	57,54	
			TOTAL PARTIDA			**313,22**
Asciende el precio total de la partida a la mencionada cantidad de TRESCIENTOS TRECE EUROS con VEINTIDOS CÉNTIMOS						
2.2.1		**ud**	**Aprendizaje ROS**			
			Aprendizaje del entorno de programación que se usará para la implementación de este proyecto, es decir, ROS.			
MOT	5,000	h	Hora de supervisor de proyecto	19,18	95,90	
MOP	50,000	h	Hora de técnico	15,98	799,00	
			TOTAL PARTIDA			**894,90**
Asciende el precio total de la partida a la mencionada cantidad de OCHOCIENTOS NOVENTA Y CUATRO EUROS con NOVENTA CÉNTIMOS						
2.2.2		**ud**	**Estudio del sensor**			
			Documentación y aprendizaje del funcionamiento del sensor que se usará para el proyecto.			
MOT	1,000	h	Hora de supervisor de proyecto	19,18	19,18	
MOP	15,000	h	Hora de técnico	15,98	239,70	
			TOTAL PARTIDA			**258,88**
Asciende el precio total de la partida a la mencionada cantidad de DOSCIENTOS CINCUENTA Y OCHO EUROS con OCHENTA Y OCHO CÉNTIMOS						

CÓDIGO	CANTIDAD	UD	RESUMEN	PRECIO	SUBTOTAL	IMPORTE
CAPÍTULO 3 Desarrollo de proyecto						
3.1			**Simulación de un imaging sonar**			
			Simulación de un MSIS usando ROS, el simulador UWsim y la arquitectura de control COLA2.			
MOP	275,000	h	Hora de técnico	15,98	4.394,50	
MOT	44,000	h	Hora de supervisor de proyecto	19,18	843,92	
			TOTAL PARTIDA			**5.238,42**

Asciende el precio total de la partida a la mencionada cantidad de CINCO MIL DOSCIENTOS TREINTA Y OCHO EUROS con CUARENTA Y DOS CÉNTIMOS

CÓDIGO	CANTIDAD	UD	RESUMEN	PRECIO	SUBTOTAL	IMPORTE
3.2			**Detección de obstáculos**			
			Detección de obstáculos mediante un imaging sonar como es el MSIS.			
MOP	60,000	h	Hora de técnico	15,98	958,80	
MOT	10,000	h	Hora de supervisor de proyecto	19,18	191,80	
			TOTAL PARTIDA			**1.150,60**

Asciende el precio total de la partida a la mencionada cantidad de MIL CIENTO CINCUENTA EUROS con SESENTA CÉNTIMOS

CÓDIGO	CANTIDAD	UD	RESUMEN	PRECIO	SUBTOTAL	IMPORTE
3.3			**Evitación de obstáculos**			
			Mediante el uso del paquete move_base de ROS, se configurará de forma de que evite los obstáculos encontrados en el mapa.			
MOP	75,000	h	Hora de técnico	15,98	1.198,50	
MOT	15,000	h	Hora de supervisor de proyecto	19,18	287,70	
			TOTAL PARTIDA			**1.486,20**

Asciende el precio total de la partida a la mencionada cantidad de MIL CUATROCIENTOS OCHENTA Y SEIS EUROS con VEINTE CÉNTIMOS

CÓDIGO	CANTIDAD	UD	RESUMEN	PRECIO	SUBTOTAL	IMPORTE
CAPÍTULO 4 Ejecución de la solución						
4.1			**Prueba 1: Escenario con piedras** Ejecución del algoritmo en el simulador en un escenario con piedras para ver como se comporta la solución adoptada. Incluye todas las pruebas necesarias para obtener un buen funcionamiento y una solución correcta así como la graficación de los datos obtenidos.			
MOT	0,500	h	Hora de supervisor de proyecto	19,18	9,59	
MOP	25,000	h	Hora de técnico	15,98	399,50	
			TOTAL PARTIDA			**409,09**

Asciende el precio total de la partida a la mencionada cantidad de CUATROCIENTOS NUEVE EUROS con NUEVE CÉNTIMOS

CÓDIGO	CANTIDAD	UD	RESUMEN	PRECIO	SUBTOTAL	IMPORTE
4.2			**Prueba 2: Escenario con columnas** Ejecución del algoritmo en el simulador en un escenario con columnas para ver como se comporta la solución adoptada. Incluye todas las pruebas necesarias para obtener un buen funcionamiento y una solución correcta así como la graficación de los datos obtenidos.			
MOT	1,000	h	Hora de supervisor de proyecto	19,18	19,18	
MOP	25,000	h	Hora de técnico	15,98	399,50	
			TOTAL PARTIDA			**418,68**

Asciende el precio total de la partida a la mencionada cantidad de CUATROCIENTOS DIECIOCHO EUROS con SESENTA Y OCHO CÉNTIMOS

CÓDIGO	CANTIDAD	UD	RESUMEN	PRECIO	SUBTOTAL	IMPORTE
CAPÍTULO 5 Documentación						
5.1			**Ejecución de memoria**			
			Elaboración de toda la documentación necesaria para poder realizar la entrega del proyecto.			
MOT	10,000	h	Hora de supervisor de proyecto	19,18	191,80	
MOP	180,000	h	Hora de técnico	15,98	2.876,40	
				TOTAL PARTIDA		**3.068,20**

Asciende el precio total de la partida a la mencionada cantidad de TRES MIL SESENTA Y OCHO EUROS con VEINTE CÉNTIMOS

PRESUPUESTO Y MEDICIONES

Diseño y implementación de algoritmos de evitación de obstáculos

CÓDIGO	RESUMEN	UDS	LONGITUD	ANCHURA	ALTURA	PARCIALES	CANTIDAD	PRECIO	IMPORTE
	CAPÍTULO 1 Proposición de proyecto								
1.1	**ud Definición de proyecto a realizar**								
	Definición de proyecto a realizar junto con los detalles principales de las tareas a ejecutar. Se detallará el alcance del proyecto y los materiales a utilizar para la correcta ejecución.								
							1,00	191,80	191,80
1.2	**ud Hardware utilizado**								
	Suministro de hardware usado para la ejecución del proyecto.								
							1,00	1.980,00	1.980,00
	TOTAL CAPÍTULO 1 Proposición de proyecto								**2.171,80**

CÓDIGO	RESUMEN	UDS	LONGITUD	ANCHURA	ALTURA	PARCIALES	CANTIDAD	PRECIO	IMPORTE
	CAPÍTULO 2 Tareas previas								
2.1	**ud Configuración y puesta a punto**								
	Configuración del hardware para poder realizar el trabajo requerido así como la instalación de todos los programas necesarios relacionados con el proyecto (ROS, Matlab, etc)								
							1,00	313,22	313,22
2.2.1	**ud Aprendizaje ROS**								
	Aprendizaje del entorno de programación que se usará para la implementación de este proyecto, es decir, ROS.								
							1,00	894,90	894,90
2.2.2	**ud Estudio del sensor**								
	Documentación y aprendizaje del funcionamiento del sensor que se usará para el proyecto.								
							1,00	258,88	258,88

TOTAL CAPÍTULO 2 Tareas previas **1.467,00**

CÓDIGO	RESUMEN	UDS	LONGITUD	ANCHURA	ALTURA	PARCIALES	CANTIDAD	PRECIO	IMPORTE
	CAPÍTULO 3 Desarrollo de proyecto								
3.1	**Simulación de un imaging sonar**								
	Simulación de un MSIS usando ROS, el simulador UWsim y la arquitectura de control COLA2.								
							1,00	5.238,42	5.238,42
3.2	**Detección de obstáculos**								
	Detección de obstáculos mediante un imaging sonar como es el MSIS.								
							1,00	1.150,60	1.150,60
3.3	**Evitación de obstáculos**								
	Mediante el uso del paquete move_base de ROS, se configurará de forma de que evite los obstáculos encontrados en el mapa.								
							1,00	1.486,20	1.486,20
	TOTAL CAPÍTULO 3 Desarrollo de proyecto								**7.875,22**

CÓDIGO	RESUMEN	UDS	LONGITUD	ANCHURA	ALTURA	PARCIALES	CANTIDAD	PRECIO	IMPORTE
	CAPÍTULO 4 Ejecución de la solución								
4.1	**Prueba 1: Escenario con piedras** Ejecución del algoritmo en el simulador en un escenario con piedras para ver como se comporta la solución adoptada. Incluye todas las pruebas necesarias para obtener un buen funcionamiento y una solución correcta así como la graficación de los datos obtenidos.								
							1,00	409,09	409,09
4.2	**Prueba 2: Escenario con columnas** Ejecución del algoritmo en el simulador en un escenario con columnas para ver como se comporta la solución adoptada. Incluye todas las pruebas necesarias para obtener un buen funcionamiento y una solución correcta así como la graficación de los datos obtenidos.								
							1,00	418,68	418,68
	TOTAL CAPÍTULO 4 Ejecución de la solución								**827,77**

CÓDIGO	RESUMEN	UDS	LONGITUD	ANCHURA	ALTURA	PARCIALES	CANTIDAD	PRECIO	IMPORTE
	CAPÍTULO 5 Documentación								
5.1	**Ejecución de memoria**								
	Elaboración de toda la documentación necesaria para poder realizar la entrega del proyecto.								
							1,00	3.068,20	3.068,20
	TOTAL CAPÍTULO 5 Documentación								**3.068,20**
	TOTAL								**15.409,99**

RESUMEN DE PRESUPUESTO

Diseño y implementación de algoritmos de evitación de obstáculos

CAPITULO	RESUMEN	EUROS	%
1	Proposición de proyecto	2.171,80	14,09
2	Tareas previas	1.467,00	9,52
3	Desarrollo de proyecto	7.875,22	51,10
4	Ejecución de la solución	827,77	5,37
5	Documentación	3.068,20	19,91
	TOTAL EJECUCIÓN MATERIAL	**15.409,99**	
	13,00% Gastos generales 2.003,30		
	6,00% Beneficio industrial 924,60		
	SUMA DE G.G. y B.I.	2.927,90	
	16,00% I.V.A.	2.934,06	
	TOTAL PRESUPUESTO CONTRATA	**21.271,95**	
	TOTAL PRESUPUESTO GENERAL	**21.271,95**	

Asciende el presupuesto general a la expresada cantidad de VEINTIUN MIL DOSCIENTOS SETENTA Y UN EUROS con NOVENTA Y CINCO CÉNTIMOS

, a 12/09/2018.

El promotor **La dirección facultativa**

Parte III

Anexos

Apéndice A

Código de mb_sensor.py

```
#!/usr/bin/env python

# IMPORTACIONES LIBRERIAS
# Importaciones ROS basicas
import rospy

# Importaciones mensajes
from sensor_msgs.msg import JointState
from sensor_msgs.msg import LaserScan
from geometry_msgs.msg import Point
from sensor_msgs.msg import PointCloud2, PointField
import sensor_msgs.point_cloud2 as pcl2
from miniking_ros.msg import AcousticBeam

# Mas importaciones
import math
import numpy as np
import visor_obstaculos as graf


def Calculo_angulo_incidencia(scan):
    # Funcion que a partir de los datos obtenidos por el Laserscan
        calcula el valor del
    # angulo de incidencia para cada distancia.
    # angulo_incidencia = angulo (theta) formado por una distancia d
        y la union de d y d'
    # scan = distancia en la que se produce un eco
    # Cuando escanea la parte superior el valor es el rango max del
        sensor
```

```
    # y cuando escanea la parte inferior (hacia el fondo) se detectan
         valores del fondo
    # y valores de los obstaculos que se encuentren por el terreno

    # Declaraciones e inicializacion variables funcion
    x = [] # Vector de valores del eje x de la distancia del sensor
        scan.ranges
    y = [] # Vector de valores del eje y de la distancia del sensor
        scan.ranges
    c = [] # Vector de distancias entre d y d'
    angulo_incidencia = [] # Angulo opuesto a d'
    angulo_scan = scan.angle_min # Inicializacion del angulo a -20
        grados = -0.349rad
                                 # (se modifica el valor en los datos
                                     del multibeamsensor
                                 # en la escena
                                     uwsim_sparus2_echos_micron.xml)

    scan.ranges = sorted (scan.ranges) # Ordena valores en ascendente

    for i in range(len(scan.ranges)):
        angulo_scan = angulo_scan + scan.angle_increment
        x.append(scan.ranges[i]*math.cos(angulo_scan))
        y.append(scan.ranges[i]*math.sin(angulo_scan))

    """Se crea otro bucle para calcular la distancia c ya que se
        necesita conocer el
    valor anterior y actual de x e y. La longitud es 2001 (-1) ya que
        no nos interesa
    el ultimo valor con el siguiente"""
    for i in range(len(scan.ranges)-1):
        c.append(math.sqrt((x[i]-x[i+1])**2 + (y[i]-y[i+1])**2))
        # aplicamos el teorema del seno para calcular el angulo de
            incidencia
        angulo_incidencia.append(math.asin((math.sin(scan.
            angle_increment)*(x[i+1]/c[i]))))

    return angulo_incidencia #en radianes


def Contador_beams(scan, angulo_incidencia, beam_footprint):
    # Funcion que calcula la intensidad del eco para cada paso del
        angulo
    # del escaneo. Sumando para cada distancia (1m) los senos del
        angulo de
```

```
    # incidencia se obtienen los valores de intensidad.
    # Se debe garantizar al menos un rayo para cada una de las
        distancias.

    num_sep = int((scan.range_max-scan.range_min)/beam_footprint)

    cont_beamsxdist = []
    # vector que almacena los beams registrados para cada distancia

    reg_beams = []
    # vector que registra la intensidad del eco para cada distancia

    reg_beams_norm  = []
    # vector que registra los beams para cada distancia, normalizado
        a escala
    # 0 a 255 como se muestra en los valores reales del sensor

    # Inicializacion de los vectores a cero
    for i in range(0, num_sep):
        cont_beamsxdist.append(0)
        reg_beams.append(0)
        reg_beams_norm.append(0)

    # Bucle que calcula la intensidad de los beams para cada
        distancia sin
    # normalizar los datos
    for i in range(len(scan.ranges)):
        x = int(scan.ranges[i]/beam_footprint)
        # se calcula para cada distancia la intensidad del eco
            respecto al
        # angulo de incidencia
        if x < num_sep-1: # se omite la distancia maxima del sensor
            cont_beamsxdist[x] += 1
            reg_beams[x] = reg_beams[x] + math.sin(angulo_incidencia[
                i])

    # Bucle para calcular el vector reg_beams_norm se ajusta la
        escala de
    # los valores obtenidos en el vector reg_beams
    for i in range(num_sep):
        if reg_beams[i] !=0:
            reg_beams_norm[i] = int((reg_beams[i]/cont_beamsxdist[i])
                *255)

    return reg_beams_norm, num_sep
```

```
class SonarSimulator:
    def __init__(self):
        # Constructor #
        """ Calcula  el angulo de incidencia y cuenta los rayos(beams
            ) para cada distancia
        a medida que se ejecuta el barrido del sensor """

        # Inicializaciones del barrido del sensor
        self.angle_max = math.radians(60) # apertura maxima del
            barrido
        self.angle_min = - self.angle_max # apertura minima del
            barrido
        self.inc_angle = 0.005 # inceremento del angulo de barrido
        self.beam_footprint = 1 # en metros = 10cm
        self.direccion = True #True avanza positivo, False avanza
            negativo

        # Creacion de publishers
        self.pub = rospy.Publisher('/uwsim/joint_state_command',
            JointState, queue_size=10)
        self.acoustic_beam_pub = rospy.Publisher('/sonar',
            AcousticBeam, queue_size=10)
        self.pointcloud2 = rospy.Publisher('/sonar_cloud',PointCloud2
            , queue_size=10)

        # Creacion de subscribers
        self.sub = rospy.Subscriber("/multibeam_scan", LaserScan,
            self.callback)

        # Definicion de los mensajes
        self.scan = LaserScan()
        self.js = JointState()
        self.js.name = ['robot2mb']
        self.js.position = [0]
        self.js.velocity = []
        self.js.effort = []
        self.js.header.frame_id = 'mb_sensor'

        self.js.position[0] = math.radians(35) #para que se encuentre
             el obstaculo antes


    def callback(self, scan):
```

```
angulo_incidencia = Calculo_angulo_incidencia(scan)
reg_beams_norm, num_sep = Contador_beams(scan,
   angulo_incidencia, self.beam_footprint)

self.js.header.stamp = rospy.Time.now()

# Barrido del sensor
if(self.direccion):
     self.js.position[0] += self.inc_angle

     if self.js.position[0] >= self.angle_max:
          self.direccion = False #cambia direccion
else:
     self.js.position[0] -= self.inc_angle
     if self.js.position[0] <= self.angle_min:
          self.direccion = True

self.pub.publish(self.js)

# Declaracion parametros del mensaje tipo acousticbeam
abeam = AcousticBeam()
abeam.header = scan.header
abeam.header.frame_id = 'multibeam'
abeam.range_max = scan.range_max # de UWSim!
abeam.angle_step = math.degrees(self.inc_angle ) # angulo del
     barrido
abeam.left_limit = math.degrees(self.angle_min )
abeam.right_limit = math.degrees(self.angle_max)
abeam.bins = len(reg_beams_norm)
abeam.angle = self.js.position[0]
abeam.intensities = reg_beams_norm

dist_step = abeam.range_max/float(abeam.bins)

self.acoustic_beam_pub.publish(abeam)

x_graf, y_graf = graf.grafica(abeam.intensities, num_sep)
#x_graf es el valor de la posicion de la intensidad max,
   y_graf la intensidad pico

points = []
if len(x_graf) == 0:
     points.append([abeam.range_max, 0, 0])
for i in range(len(x_graf)):
     x = dist_step*x_graf[i]
```

```
                points.append([x, 0, 0])

        pc2 = pcl2.create_cloud_xyz32(scan.header, points)
        self.pointcloud2.publish(pc2)


if __name__ == '__main__':
    rospy.init_node('mb_sensor')
    SonarSimulator()
    rospy.spin()
```

Apéndice B

Código de twist2bvr.py

```
#!/usr/bin/env python

# imports
import rospy
from geometry_msgs.msg import Twist
from auv_msgs.msg import BodyVelocityReq, GoalDescriptor

# Nodo para controlar la velocidad del AUV

class Twist2BVR():
        def __init__(self):
                self.bvr_publisher = rospy.Publisher('/cola2_control/
                    body_velocity_req', BodyVelocityReq, queue_size
                    =10)
                self.bvr_sub = rospy.Subscriber("/cmd_vel", Twist,
                    self.twist_callback)

                # Parametros
                self.bvr = BodyVelocityReq()
                self.bvr.header.frame_id = 'sparus2'
                self.bvr.goal.priority = GoalDescriptor.
                    PRIORITY_NORMAL
                self.bvr.goal.requester = 'Twist2BVR'
                self.bvr.goal.id = 0
                self.bvr.disable_axis.x = False
                self.bvr.disable_axis.y = True
                self.bvr.disable_axis.z = True
                self.bvr.disable_axis.roll = True
                self.bvr.disable_axis.pitch = True
```

```
                self.bvr.disable_axis.yaw = False

        def twist_callback(self, msg):
                self.bvr.twist = msg
                print msg
                self.bvr.header.stamp = rospy.Time.now()
                self.bvr_publisher.publish(self.bvr)

if __name__ == '__main__':
    rospy.init_node('twist2bvr')
    Twist2BVR()
    rospy.spin()
```

Apéndice C

Código de visor_obstaculos.py

```
#!/usr/bin/env python


import math
import matplotlib.pyplot as plt
import numpy as np

from detect_peaks import detect_peaks
#http://nbviewer.jupyter.org/github/demotu/BMC/blob/master/notebooks/
    DetectPeaks.ipynb


def Calculo_rango_ceros(a):
    # Funcion que crea un vector que calcula los rangos en que el
        valor es 0

    # Creacion de un vector que detecta los ceros (con valor 1) y el
        resto (valor 0)
    iszero = np.concatenate(([0], np.equal(a, 0).view(np.int8), [0]))
    absdiff = np.abs(np.diff(iszero))
    # Rango donde empieza y acaba el pico en 0
    ranges = np.where(absdiff == 1)[0].reshape(-1, 2)
    return ranges


def verif_picos(registro_norm): ### REVISAR!
    """codigo para calcular si el pico maximo se encuentra entre
        valores ceros
         es decir sombras y por tanto es un obstaculo"""
```

```
        #Calculamos los intervalos en que los valores del sensor son
            nulos
    if registro_norm!=0:
        ceros = Calculo_rango_ceros(registro_norm)
        ceros=ceros.tolist()

        ipico=list(range(len(ceros)-1))
        ipicoy=list(range(len(ceros)-1))

        #vector que contiene la posicion x de los puntos con
            intesidad
        obstaculos=list(range(len(xpico)))
        """vector que contiene todos los posibles obstaculos que
            luego
        se calculan si lo son"""

        #calcula intervalos de valores los cuales se encuentran entre
             ceros
        for j in range(len(ceros)-1):
            c=ceros[j]
            c1=ceros[j+1]
            interv_picos=list(range(c[1],c1[0]+1))
            ipico[j] = interv_picos #se guardan todos los intervalos
                en x
            u=list(range(len(interv_picos)-1))

            for i in range(len(interv_picos)-1):
                u[i]=registro_norm[interv_picos[i]]
                ipicoy[j] = list([registro_norm[interv_picos[i]]])

        #se inicializa la variable obstaculos a false
        for i in range(len(xpico)):
            obstaculos[i]=False

        #se comprueban si son realmente obstaculos
        for i in range(len(xpico)):
            for j in range(len(ipico)):
                if xpico[i] in ipico[j]:
                    obstaculos[i]=True

        #bucle que cuenta todos los obtaculos que hay
        contador=0
        for i in range(len(obstaculos)):
            if(obstaculos[i]==True):
                contador+=1
```

```
def grafica(registro_norm, num_sep):
    # Funcion que muestra en una grafica, la intensidad del eco del
        sensor para cada una
    # de las distancias que abarca el rango del sensor.

    try:
        picos = detect_peaks(registro_norm , mph=110, mpd=2)
        """Funcion que detecta picos, mph-> detecta picos mayores que
            la
        altura minima definida, mpd-> picos que esten separados esa
           distancia"""

        xpico = picos.tolist() #convierte de np.array a una lista
        # Se guarda en xpico el valor de la POSICION de la INTENSIDAD
            de los ecos maximos, es decir,
        # donde se encuentra un obstaculo mas cercano


        #Para obtener el valor de la intensidad de cada xpico
        ypico = []
        if len(xpico)!=0: #si hay algun pico
            for i in range(len(xpico)):
                idx=xpico[i]
                ypico.append(registro_norm[idx])

        # GRAFICA
        plt.ion()
        plt.plot(registro_norm) #dibuja el plot con la variable
           normalizada

        #dibuja en el plot los picos maximos
        for i in range(len(xpico)):
            plt.plot(xpico[i],ypico[i], marker='s',color='r', label='
               contador picos')

        plt.hold(True)
        plt.axis([0,num_sep, 0,255])
        plt.title('Datos Sensor',fontsize=20, fontweight='bold')
        plt.xlabel('Rango (metros)')
        plt.ylabel('Intensidad')
        #plt.legend()

        plt.show()
        plt.pause(1e-10)
```

```
            plt.clf()

    except KeyboardInterrupt:
        print ""
        sys.exit(0)

    return xpico, ypico
```

Apéndice D

Código de sparus2.urdf

```
<?xml version="1.0"?>
<robot name="Sparus2">

<!-- Links -->

  <link name="sparus2">
    <visual>
      <origin xyz="0.0 0 0" rpy="-1.57 0 0"/>
      <geometry>
        <mesh filename="package://cola2_s2/data/model/sparus2.3ds"
            scale="1 1 1"/>
      </geometry>
    </visual>
  </link>

 <link name="multibeam">
    <visual>
      <origin rpy="0.0 0.0 0.0" xyz="0 0 0"/>
      <geometry>
        <cylinder radius="0.05" length="0.1"/>
      </geometry>
    </visual>
  </link>.


<joint name="robot2mb" type="continuous">
    <parent link="sparus2"/>
    <child link="multibeam"/>
    <axis xyz="0 1 0 "/>
```

```
    <origin rpy="1.57 0 0" xyz="0.7 0.0 0.20"/>
  </joint>
</robot>

</robot>
```

Bibliografía

[1] Romerbots Blog. *Algoritmo VFH*. Universitat de les Illes Balears, 2016 (vid. pág. 19).

[2] Johann Borenstein y Yoram Koren. "The vector field histogram-fast obstacle avoidance for mobile robots". En: *IEEE transactions on robotics and automation* 7.3 (1991), págs. 278-288 (vid. pág. 20).

[3] Antoni Burguera. "A novel approach to register sonar data for underwater robot localization". En: *Intelligent Systems Conference (IntelliSys), 2017*. IEEE. 2017, págs. 1034-1043 (vid. pág. 35).

[4] M. Duarte. *Notes on Scientific Computing for Biomechanics and Motor Control*. `https://github.com/demotu/BMC`. 2015 (vid. pág. 38).

[5] Emilio García Fidalgo. "Implantación de un sensor sónar para el control de la navegación de un vehículo submarino". En: (2007) (vid. pág. 15).

[6] Vladimir J Lumelsky y Alexander A Stepanov. "Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape". En: *Algorithmica* 2.1-4 (1987), págs. 403-430 (vid. pág. 16).

[7] Adeline Marcos. *Explotación de los fondos marinos, ¿un negocio rentable?* 2010 (vid. pág. 3).

[8] Narcis Palomeras y col. "COLA2: A control architecture for AUVs". En: *IEEE Journal of Oceanic Engineering* 37.4 (2012), págs. 695-716 (vid. págs. 26, 32).

[9] Mario Prats y col. "An open source tool for simulation and supervision of underwater intervention missions". En: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 2012, págs. 2577-2582 (vid. pág. 26).

[10] Seaview Systems. *What is an ROV*. 2010 (vid. págs. 6, 7).