Doctoral Thesis

ONLINE PATH PLANNING FOR AUTONOMOUS UNDERWATER VEHICLES UNDER MOTION CONSTRAINTS

JUAN DAVID HERNÁNDEZ VEGA

2017

Doctoral Thesis

ONLINE PATH PLANNING FOR AUTONOMOUS UNDERWATER VEHICLES UNDER MOTION CONSTRAINTS

JUAN DAVID HERNÁNDEZ VEGA

2017

Doctoral Program in Technology

Supervised by:
Dr. Marc Carreras

Thesis submitted to the University of Girona in fulfillment of the requirements for the degree of Doctor of Philosophy

CERTIFICAT DE DIRECCIÓ DE TESI

Dr. Marc Carreras, del Departament d'Arquitectura i Tecnologia de Computadors de la Universitat de Girona,

DECLARO:

Que el treball titulat Online Path Planning for Autonomous Underwater Vehicles under Motion Constraints, que presenta Juan David Hernández Vega per a l'obtenció del títol de doctor, ha estat realitzat sota la meva direcció i que compleix els requisits per poder optar a Menció Internacional.

I, perquè així consti i tingui els efectes oportuns, signo aquest document.

Girona, Setembre 2017

Dr. Marc Carreras

ACKNOWLEDGMENTS

The work presented throughout this manuscript would not have been possible without the help of some incredible people. Their continuous support, from the beginning of this journey, till its end, helped me celebrate even the smallest achievements, as well as overcome the difficult moments any PhD thesis involves. I would like to give my gratitude to all of them.

En primer lugar me gustaría agradecer a mi familia en Armenia. A mi madre, por buscar y querer de mi siempre lo mejor; por su carácter y decisión para afrontar las dificultades de la vida; por sus sacrificios para ofrecerme las mejores oportunidades. A Agustín Escobar, por hacer de mi la persona y el ingeniero que soy, y por todos aquellos consejos que nunca olvidaré. Espero que este logro lo estés disfrutando desde el infinito. A mi hermana Maria Paula, por su alegría y su apoyo incondicional, por estar siempre ahí para mi sin importar la distancia. A Denisa, quien también ya hace parte de esta familia, gracias por apoyarme en los buenos y malos momentos de este doctorado, y por escuchar mis interminables conversaciones sobre mis experimentos y artículos.

También me gustaría agradecer a mi familia en Neiva. A mi padre, por sus llamadas y discusiones sobre la vida académica, por los buenos consejos y por escucharme en los momentos que he tenido que tomar decisiones importantes. A mis hermanos Daniela y Sergio Andrés, así como su madre Sonia Perdomo, gracias por la buena energía, y los buenos deseos. Han sido cortos y pocos los momentos que hemos podido compartir en los últimos años, pero gracias por estar siempre ahí.

Sin lugar a dudas, irme a Girona a hacer mi doctorado fue la mejor decisión profesional y experiencia personal que pude tener. Girona siempre será como mi segunda casa. Por esto me gustaría a agradecer a mi tutor, quien desde un inicio me dio la posibilidad de preparar la propuesta de tesis con la que obtuve la beca de Colciencias, propuesta que se ha convertido ahora en una realidad. Gracias por el tiempo, las discusiones, las charlas de camino a Sant Feliu, y por siempre ser crítico y realista a la hora de definir los objetivos. Al mismo tiempo, me gustaría agradecer a Pere, por siempre estar allí para discutir temas adicionales de la tesis, los experimentos y los artículos. Siempre fue para mi como un co-tutor.

Un agradecimiento muy especial va para mis amigos y compañeros de trabajo en el CIRS y el ViCOROB. Quisiera empezar con aquellos que estuvieron más cerca del desarrollo de esta tesis. Gracias a Enric por ser como un tutor y mentor durante mi primer año, por nuestras discusiones sobre path/motion planning, y por sus consejos para escribir artículos. A Eduard, Èric y Narcís quisiera agradecerles por siempre estar dispuestos a implementar, ajustar, mejorar la arquitectura del robot, o simplemente acompañarme durante los experimentos en el mar. A Carles y a Lluís, por tener los robots siempre a punto, solucionar los problemas técnicos, y por darme siempre consejos prácticos de mis experimentos. Gracias a mis compañeros de doctorado con los que compartí mis "deadlines", viajes y competiciones, Arnau, Guillem, Albert, Jep, y Klemen. Gracias también a

los miembros seniors, Aggelos, Tali (en especial por darle revisiones adicionales a mis papers, abstracts, y pósters), Ricard, David y Nuno. Quisiera agradecer además al área administrativa del ViCOROB. Joseta, Mireia, Anna, Olga y Bego, que siempre solucionaron todos los problemas con la Universidad, las conferencias, los viajes y demás.

I would also like to thank my foreigner friends who have been part of ViCOROB. Francesco, Mojdeh and Guillaume, Sonia, Habib (and Sarah), Konstatin (and Amanda), Shihav, and Sharad- I will never forget our amazing multicultural dinners. I learned so much from all of you. And since the group was ever-changing, also its latest members: Dina, Khadidja, Klemen, Richa, and Patryk.

One of the most important and decisive experiences during my PhD was without a doubt my visit to Rice University in Houston. I would especially like to thank Lydia Kavraki and Mark Moll, my advisors there. Their time, dedication and contributions to this thesis are invaluable. I would also like to thank all the people at the Kavraki Lab: Morteza, Ryan, Didier, Dinler, Jayvee, Sarah, and Keliang. Your advice and comments during our meetings not only helped me in my work but also made my stay there an incredible experience.

Por último me gustaría agradecer a las fuentes de financiamiento de mis estudios, experimentos y publicaciones. Al Departamento Administrativo de Ciencia, Tecnología e Innovación (Colciencias), de Colombia, que me otorgó la beca de estudios doctorales. También me gustaría agradecer a la Unión Europea, a la Generalitat de Catalunya y al Gobierno Español, que por medio de proyectos financiaron parte de mi investigación.

PUBLICATIONS

The work developed in this thesis led to the following publications:

JOURNAL ARTICLES

[JFR'17] **J. D. Hernández**, E. Vidal, K. Istenič, M. Moll, M. Carreras, and L. E. Kavraki, "Online Path Planning for Unexplored Underwater Environments using AUVs that Operate under Motion Constraints," Journal of Field Robotics, (in preparation) 2017.

[SENSORS'16] **J. D. Hernández**, K. Istenič, N. Gracias, N.Palomeras, R. Campos, E. Vidal, R. García, and M. Carreras, "Autonomous Underwater Navigation and Optical Mapping in Unknown Natural Environments," Sensors, vol. 16, no. 8, p. 1174. July 2016.

PEER-REVIEWED CONFERENCES AND WORKSHOPS

[IROS'16] **J. D. Hernández**, M. Moll, E. Vidal, M. Carreras and L. E. Kavraki, "Planning Feasible and Safe Paths Online for Autonomous Underwater Vehicles in Unknown Environments," IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), (Daejeon, Korea), pp. 1313–1320. October 2016.

[ROBOT'15] **J. D. Hernández**, K. Istenič, N. Gracias, R. García, P. Ridao, and M. Carreras, "Autonomous Seabed Inspection for Environmental Monitoring," ROBOT2015: Second Iberian Robotics Conference, (Lisbon, Portugal). Springer International Publishing, pp. 27–39. November 2015.

[ICRA'15] **J. D. Hernández**, E. Vidal, G. Vallicrosa, E. Galceran and M. Carreras, "Online Path Planning for Autonomous Underwater Vehicles in Unknown Environments," IEEE International Conference on Robotics and Automation (ICRA), (Seattle, USA), pp. 1152-1157. May 2015.

[NGCUV'15] **J. D. Hernández**, G. Vallicrosa, E. Vidal, E. Pairet, M. Carreras, and P. Ridao. "Online 3D Path Planning for Close-proximity Surveying with AUVs," IFAC Workshop on Navigation, Guidance and Control of Underwater Vehicles (NGCUV), Girona. April 2015.

OTHER CONFERENCES AND WORKSHOPS

[OCEANS'17] **J. D. Hernández**, E. Vidal, J. Greer, R. Piasco, P. Jaussaud, M. Carreras, and R. García, "AUV Online Mission Replanning for Gap Filling and Target Inspection," MTS/IEEE OCEANS, (Aberdeen, Scotland). (to appear, 2017).

[AUV'16] M. Carreras, **J. D. Hernández**, E. Vidal, N. Palomeras, and P. Ridao, "Online Motion Planning for Underwater Inspection," IEEE/OES Autonomous Underwater Vehicles (AUV), (Tokyo, Japan), pp. 336-341. November 2016.

[OCEANS'15] **J. D. Hernández**, E. Vidal, G. Vallicrosa, E. Pairet, and M. Carreras. "Simultaneous Mapping and Planning for Autonomous Underwater Vehicles in Unknown Environments," IEEE OCEANS, (Genova, Italy). May 2015.

CONTRIBUTIONS TO OTHER WORKS

[RA-LETTERS'17] E. Vidal, **J. D. Hernández**, K. Istenič, and M. Carreras, "Online View Planning for Inspecting Unexplored Underwater Structures," IEEE Robotics and Automation Letters, vol. 2, no. 3, p. 1436 - 1443. February 2017.

[ICRA'18] E. Pairet, **J. D. Hernández**, M. Lahijanian, and M. Carreras, "Uncertainty-based Online Mapping and Motion Planning for Marine Robotics Guidance," IEEE International Conference on Robotics and Automation (ICRA), (in preparation) 2018.

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ACRONYMS

C-Space configuration space

ACA Ariadne's clew algorithm

RPP randomized path planner

PRM probabilistic roadmap

RM roadmap

RRT rapidly-exploring random tree

RRT* asymptotic optimal RRT

T-RRT transition-based RRT

CL-RRT closed-loop RRT

GVD generalized Voronoi diagram

CPP coverage path planning

AFP artificial potential field

FADPRM flexible anytime dynamic PRM

D* dynamic A*

AD* anytime dynamic A*

ARA* anytime repairing A*

2D 2-dimensional

3D 3-dimensional

6D 6-dimensional

EST expansive-spaces tree

FM fast marching

OMPL open motion planning library

B-spline basis spline

HLP high-level planner

LLP low-level planner

DFS depth-first search

DOF degrees of freedom

ROS robot operating system

SLAM simultaneous localization and mapping

UAV unmanned aerial vehicle

GA genetic algorithm

DR dead-reckoning

AUV autonomous underwater vehicle

UUV unmanned underwater vehicle

ROV remotely operated vehicle

DVL doppler velocity log

IMU inertial measurement unit

NED north-east-depth

USBL ultra-short baseline

CIRS Underwater Vision and Robotics Research Center

ViCOROB Computer Vision and Robotics Institute

COLA2 component oriented layer-based architecture for autonomy

UWSim underwater simulator

Ifremer French Research Institute for Exploitation of the Sea

ACE automated control engine

ISE International Submarine Engineering

SDNC splats distance normalized cut

CONTENTS

ABSTRACT

Since their beginning in the late 1950s, the capabilities and applications of autonomous underwater vehicles (AUVs) have continuously evolved. Their most common applications include imaging and inspecting different kinds of structures on the sea floor as well as collecting oceanographic information: biological, chemical, and even archaeological data.

Most of these applications require *a priori* information of the area or structure to be inspected, either to navigate at a safe and conservative altitude or to pre-calculate a survey path. However there are other applications where it's unlikely that such information is available (e.g., exploring confined natural environments like underwater caves). In these scenarios, AUVs must operate in unexplored and cluttered environments, and therefore are more exposed to collisions.

Although these AUV applications share some common requirements with other aerial and terrestrial robots (e.g., localization, mapping, vision, etc.), they are also different in significant ways. Navigating autonomously while conducting these type of tasks in underwater environments demands taking into account factors such as: the presence of external disturbances (currents), low-range visibility and limited navigation accuracy. Dealing with such constraints requires a path planner with online capabilities that can overcome the lack of environment information and the global position inaccuracy, especially when navigating in close proximity to nearby obstacles.

In this respect, this thesis presents an approach that endows an AUV with the capabilities to move through unexplored environments. To do so, it proposes a computational framework for planning feasible and safe paths online. This approach allows the vehicle to incrementally build a map of the surroundings, while simultaneously (re)plan a feasible path to a specified goal. To accomplish this, the framework takes into account motion constraints in planning feasible 2D and 3D paths, i.e., those that meet the vehicle's motion capabilities. It also incorporates a risk function to avoid navigating close to nearby obstacles.

To evaluate the proposed approach in different real-world scenarios, a series of trials were conducted with the Sparus II and the AsterX AUVs, torpedo-shaped vehicles that performed autonomous missions. These experiments include simulated and in-water trials in different environments, such as artificial marine structures, natural marine structures, and confined natural environments.

RESUMEN

Desde sus inicios a finales de los años 50, las capacidades y aplicaciones de los vehículos autónomos submarinos o AUVs (por sus siglas en inglés) han estado bajo un continuo proceso de evolución. Las aplicaciones más comunes incluyen la obtención de imágenes e inspeccción de diferentes tipos de estructuras, tales como cascos de barcos, estructuras naturales en el fondo marino, así como la recolección de información oceanográfica como datos biológicos, químicos e incluso arqueológicos.

Muchas de estas aplicaciones requieren información *a priori* del área o estructura que va a ser inspeccionada, ya sea para navegar a una altitud segura o para precalcular un camino para realizar estudios, el cual puede ser corregido o modificado en tiempo real. Sin embargo, existen aplicaciones similares o nuevas, como la exploración de entornos naturales confinados (e.g., cuevas submarinas), donde dicha información puede no estar disponible. En estos escenarios, los AUVs debe operan en entornos desconocidos, y por lo tanto los AUVs están más expuestos a colisiones.

Aunque estas aplicaciones de AUVs comparten algunos requiremientos comunes con otras aplicaciones de robots aéreos y terrestres (e.g., localización, mapeo, visión, etc.), navegar autónomamente mientras se ejecutan este tipo de tareas en entornos submarinos difiere en ciertos factores, como la presencia de pertubaciones externas (corrientes), bajo rango de visibilidad y limitaciones en la precisión del sistema de navegación. Para poder abordar dichas restricciones se requiere un planificador de movimientos con capacidad de cómputo en tiempo real, el que contribuya a superar las limitaciones en la información del entorno y la falta de precisión de posicionamiento, en especial cuando se navega cerca de los obstáculos de su alrededor.

En este sentido, esta tesis presenta una método para dotar un AUV con la habilidad para moverse a través de entornos no explorados. Para ello, esta tesis propone un método para calcular en tiempo real caminos factibles y seguros. El método propuesto permite al vehículo construir incrementalmente un mapa del entorno, y al mismo tiempo replaninficar el camino factible hacia la meta u objetivo establecido. Para lograr esto, es necesario considerar las restricciones de movimiento para planificar caminos 2D y 3D que sean factibles o realizables.

Para evaluar el método propuesto, se realizaron diferentes experimentos con los AUVs Sparus II y AsterX, ambos vehículos tipo torpedo que realizaron misiones de manera autónoma en diferentes escenarios. Estos experimentos incluyen pruebas en simulación y en el agua en diferentes entornos, tales como estructuras marinas artificiales, estructuras marinas naturales, así entornos naturales confinados.

RESUM

Des dels inicis a finals dels anys 50, les capacitats i aplicacions dels vehicles autònoms submarins o AUVs (per les seves sigles en anglès) han experimentat un procés d'evolució continu. Les aplicacions més comunes són l'obtenció d'imatges i inspecció de diferents tipus d'estructures, com per exemple, cascos de vaixells o estructures naturals en el fons marí, i l'adquisició d'informació oceanogràfica com dades biològiques, químiques i arqueològiques.

Moltes d'aquestes aplicacions requereixen informació *a priori* de l'àrea o estructura que es vol inspeccionar, ja sigui per navegar a una altitud segura o per calcular en avançat un camí que permeti realitzar els estudis, el qual pot ser corregit o modificat a temps real. No obstant, existeixen aplicacions similars o noves, com l'exploració d'entorns naturals confinats (e.g., coves submarines), on aquesta informació pot ser inexistent. En aquests casos, els AUVs han d'operar en entorns desconeguts, pel que estan més exposats a col·lisions.

Tot i que aquestes aplicacions de AUVs comparteixen alguns requeriments comuns amb altres aplicacions de robots aeris i terrestres (e.g., localització, mapeig, visió, etc.), navegar autònomament al mateix temps que s'executen aquest tipus de tasques en entorns submarins difereix en certs factors, com la presencia de pertorbacions externes (corrents), baix rang de visibilitat i limitacions en la precisió del sistema de navegació. Per poder tractar les esmentades restriccions és necessari un planificador de moviments amb capacitat de processament en temps real, el que ajudi a superar les limitacions en la informació de l'entorn i la falta de precisió de posicionament, en especial quan es navega a prop dels obstacles presents en el seu voltant.

En aquest sentit, aquesta tesis presenta una alternativa per dotar un AUV amb l'habilitat de moure's a través d'entorns no explorats. Per aconseguir aquesta fita, aquesta tesis proposa un mètode per calcular en temps real camins factibles i segurs. El mètode proposat permet al vehicle construir de forma incremental un mapa de l'entorn, i al mateix temps replanificar un camí factible cap a l'objectiu establert. Per assolir això, el mètode proposat te en compte les restriccions de moviment del vehicle per planificar camins 2D i 3D que siguin factibles o realitzables.

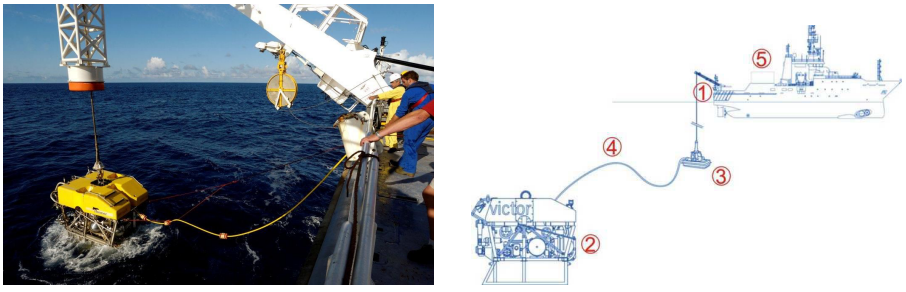
Per avaluar el mètode proposat, s'han realitzat diferents experiments amb els AUVs Sparus II i l'AsterX, els quals són vehicles de tipus torpede que van realitzar missions de forma autònoma en diferents escenaris. Aquests experiments inclouen proves en simulació i a l'aigua en diferents entorns, tal i com estructures marines artificials, estructures marines naturals i entorns naturals confinats.

1 INTRODUCTION

1.1 MOTIVATION

The need to use robotic systems in order to overcome human beings' limitations has been a topic of an ever increasing interest in the last decades. Manipulator arms, for instance, were introduced in the industry to optimize production lines by systematically executing tasks with a bounded margin of error and high repeatability during long working hours. Recently, such an interest in robots has veered towards autonomous vehicles that conduct missions in complex and undiscovered environments, such as deep oceans, volcanoes and even other planets. Consequently, various sectors of the scientific community started to research this area, especially focused on developing robotic vehicles that are capable of conducting tasks where data is gathered autonomously in different environmental media, such as air, soil and water **[Dunbabin2012]**.

Marine scientists were probably the first ones capitalizing on the use of robotic vehicles in exploring unknown environments. Oceanographers, for instance, started using unmanned underwater vehicles (UUVs) to study deep marine environments and the seafloor **[Whitcomb2000]**. Nowadays, UUVs are separated into two categories according to the level of human intervention required for their operation. A first group includes the so-called remotely operated vehicles (ROVs) which, despite being unmanned, have to be tethered to a support surface vessel from which they are powered and controlled by a human operator. Although ROVs are commonly associated with their use in inspecting and maintaining offshore oil and gas platforms **[Khan2002]**, they have also contributed to different scientific applications such as environmental monitoring **[Lam2006]** and marine archeology **[Forney2011]**. Figure 1 shows the widely-known VICTOR 6000 ROV in a typical deployment maneuver from the mother ship, where the umbilical cable used to operate the vehicle can be clearly observed.



(a) (b)

Figure 1: (a) The VICTOR 6000 ROV. (b) The ROV-vessel system: 1) a direct-winding winch, 2) the ROV, 3) the hard ballast, 4) the tether cable, and 5) the power/hydraulic units and control room. Image credit: Ifremer.

The second group gathers the autonomous underwater vehicles (AUVs), or UUVs that do not require being controlled by a human operator. This characteristic eliminates the need of continuously being in contact with a surface mother ship, thus permitting to overcome some major drawbacks of the former group, such as their high operative cost and limited working area. In most of the AUV applications, the vehicle follows a sequence of pre-calculated waypoints and uses its onboard sensors to gather oceanographic information of biological nature [**Grasmueck2006**], chemical composition [**WeiLi2006**], and even archaeological data [**Bingham2010**]. Often, they are also equipped with multibeam and imaging sonars that collect data that is used to build bathymetric maps (i.e., elevation maps of the seabed). These underwater surveys are normally conducted in a previously explored area so that the vehicle navigates at a constant and safe altitude from the seafloor.

Furthermore, AUVs can also be divided into two main subcategories: buoyancy-driven and propeller-driven. The former category corresponds to the underwater gliders, or AUVs that are capable of modifying its buoyancy in order to generate sinusoidal-like vertical motion that, together with their wings, also permit horizontal (forward and lateral) motion. The trajectories followed by these vehicles also include periodic ascents to the surface to obtain GPS fixes, thus improving its navigation estimation (see Fig. 2). This propulsion technique results in a low-consumption but also less maneuverable approach. This is commonly used in long-term studies over open sea areas, in which a single mission can last from days to weeks.

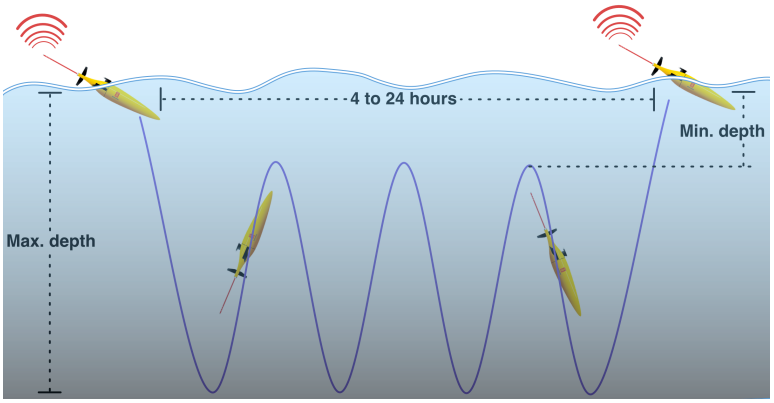


Figure 2: Underwater glider, or AUV with the capability of modifying its buoyancy to generate sinusoidal-like vertical motion. The vehicle dives for several hours to gather data, and surfaces periodically to obtain GPS fixes thus improving its own navigation estimation.

The propeller-driven AUVs, on the other hand, correspond to those that use propellers or thrusters to generate 3-dimensional (3D) motions. The work presented throughout this thesis is dedicated to this latter kind of vehicles. Furthermore, it has been mainly validated with the Sparus II, a torpedo-shaped AUV (see Fig. 3).

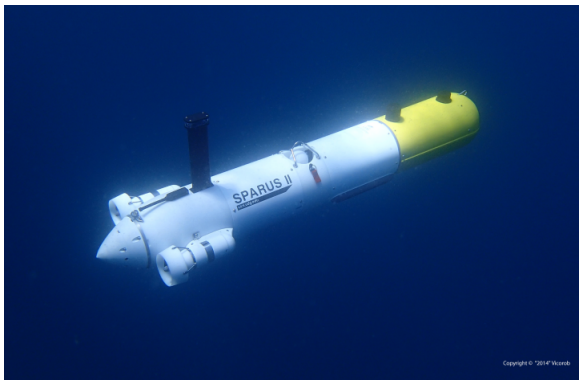


Figure 3: Sparus II AUV, a torpedo-shaped and propeller-driven underwater vehicle, which is equipped with two back thrusters for horizontal motion, and one independent thruster for vertical motion.

1.2 PROBLEM STATEMENT

Recent AUV applications include imaging and inspecting different kinds of structures such as in-water ship hulls [**Hover2012**, **Hurtos2015**] and natural structures on the sea floor [**Galceran2014b**]. These applications share a common characteristic: they require *a priori* information of the area or structure to be inspected. Therefore it is needed to either navigate at a safe and conservative altitude [**Grasmueck2006**, **Bingham2010**] or to pre-calculate a survey path that may be corrected or reshaped online [**Hover2012**, **Galceran2014b**]. However, there are similar or even potentially new applications, such as exploring confined natural environments (e.g., underwater caves) [**Mallios2015**], where such information might not be available. In these scenarios, the AUV must operate in unexplored (unknown), cluttered and dynamic environments, and therefore are more exposed to collisions.

Although the aforementioned AUV applications share some common requirements with aerial and terrestrial robots (e.g., localization, mapping, vision, etc.), navigating autonomously while conducting such type of tasks in an underwater environment presents different challenges. These are caused by factors specific to this environment, such as the presence of external disturbances (currents), low-range visibility and limited navigation accuracy. Dealing with such constraints requires a path planner with online computation capabilities, which contribute in overcoming the lack of surroundings information and the global position inaccuracy, especially when navigating in close proximity to nearby obstacles.

1.2.1 *Navigating in Unexplored Environments with AUVs*

Even with the most recent technological advances, conducting both teleoperated and autonomous missions in underwater environments still represents a challenge with different scientific aspects to be solved. One important restriction faced by UUVs is that they operate in GPS-denied environments, which limits their capability to accurately calculate their position with respect to an inertial reference frame. In order to cope with this, dead-reckoning (DR) techniques are used to estimate the relative po-

sition with respect to an initial position, which is normally defined from GPS fixes obtained when the vehicle is at surface. This approach suffices in many survey AUV applications. However, its limitations rely on highly precise odometry and low-level or negligible disturbances. When these conditions are not met, it can rapidly lead to errors in distance estimation, which make this approach an unsafe alternative when navigating and exploring surroundings in close proximity to nearby obstacles, even when the environment is known and a map is available.

Another alternative is the of use of an ultra-short baseline (USBL), which is a method of underwater acoustic positioning. An USBL system is composed of a transceiver mounted on a surface vessel and a transponder mounted on the AUV. This system permits calculating the range (distance) and the angle of the subsea vehicle with respect to the surface vehicle. Furthermore, given that the transceiver is at surface (i.e., it can get GPS fixes), it is possible to calculate a more accurate absolute position of the AUV. The major drawback of this alternative is the mobility constraint imposed by the transceiver, similar to what occurs with a ROV and its mother ship. This also prevents its use in confined natural environments such as an underwater caves complex, which results in a critical limitation for the new applications intended for AUVs.

Therefore, when planning collision-free AUV paths, the difficulty to pre-explore and build a map of the area or structure of interest is an additional challenge to be considered together with the navigation inaccuracy. In order to tackle these problems, this thesis proposes a framework that endows an AUV with the capability to map an undiscovered environment, while planning collision-free paths to navigate throughout it. Furthermore, although the framework considers specific characteristics of the motion planning problem for AUVs, the proposed approach may also be applied for other types of autonomous vehicles.

1.2.2 *Path/Motion Planning Problem for AUVs*

1.2.2.1 *Overview of the Basic Robot Path/Motion Planning*

In general, path/motion planning methods can be divided into two categories according to their application: coverage planning and start-to-goal planning. The former category, commonly referred to as coverage path planning (CPP), gathers those methods that seek to determine a collision-free path that a robot must follow in order to pass over all points of an area or volume of interest. A detailed review of available techniques that address this problem can be found in [**Galceran2013a**].

The second category, i.e., start-to-goal motion planning, consists in finding valid (collision-free) paths from a start configuration to a goal configuration in the configuration space (C-Space), which is the space of all the possible robot configurations [**Lozano-Perez1983**]. There are different computational algorithms that attempt to solve this task. Some of them find an analytic solution, but their applicability is limited to simple problems. Another group of algorithms, such as Dijkstra's [**Dijkstra1959**], A* [**Hart1968**] and D* [**Stentz1994**], search throughout discretization of the C-Space. How-

ever, in problems involving high-dimensional C-Spaces, these exhaustive search methods suffer from scalability issues.

One alternative in dealing with this is to use the so-called sampling-based methods. These latter ones construct a partial representation by taking samples at random from the C-Space and checking them for collisions [**Choset2005**, **LaValle2006**], instead of fully describing the C-Space. In contrast with exhaustive search methods, these are not complete (i.e., they can not report whether a solution exists), but often find a solution for complex problems where other algorithms fail.

There are additional characteristics or capabilities that should be considered when solving path/motion planning tasks for AUVs. Some of which will be mentioned in the following sections.

1.2.2.2 *Planning under Geometric and Differential (Motion) Constraints*

When calculating a valid path, the planner must deal with different types of constraints, some of which may be related only to the C-Space, while others may also include the vehicle motion capabilities. Depending on which constraints are considered, the planning problem was categorized in one of two possible groups: geometric path planning and motion planning [**Choset2005**]. The former group assumes that the robotic system can move instantaneously in any direction (i.e., system kinematics and dynamics are negligible), thus reducing the problem to geometric constraints that are solved by checking for collisions over the C-Space.

However, the motion constraints associated with the vehicle are often significant, and therefore the robot is not capable of following a simple geometric path. This means that, when calculating collision-free paths, it is necessary to use a vehicle motion model that includes differential constraints, also called kinodynamic constraints. Such a group of planning problems is commonly called motion planning or *kinodynamic motion planning* (this latter term was introduced by Donald et al. in 1993 [**Donald1993**]). An example of such planning problems is the one related to a non-holonomic torpedo-shaped AUV, as the ones used in this thesis (see Figure 3).

As recently the terms path planning and motion planning are used interchangeably [**Sucan2011**], [**LaValle2006**], throughout this thesis, we refer to the either problem as path/motion planning

1.2.2.3 *3D Path/Motion Planning*

In some AUV applications the vehicle either navigates at constant depth or attempts to keep reference values of altitude. In those cases, and specially in areas with no or little relief, it is possible to simplify the motion model to one that assumes the vertical position variation as constant or negligible. Nonetheless, when such conditions are not met, it is required to use more complete motion equations, which in the simplest case implies adding an additional state variable for the vertical motion, or in more complex scenarios may require to use a 6-dimensional (6D) state space, i.e., one that incorporates the position and the orientation of the vehicle in a 3D workspace, where each state or configuration is $q = [x, y, z, \phi, \theta, \psi]$.

1.2.2.4 *Online Path/Motion Planning*

Another important aspect to be considered when planning motions for AUVs, especially when navigating in unexplored environments, is the necessity of calculating the paths (motions) online. To deal with such computation constraints, the planner must be capable of providing a solution at *any time* and of correcting (reshaping) it according to the updated environment information gathered along the mission execution. This characteristic implies that although it is possible that the solution provided at *any time* may not be the most optimal one, any available time left will be spent on improving the existing solution path. This means that the next delivered solution is better than the previous one or at least equally good. Finally, it is also important to correctly balance the computation load to coexist with other functional modules of the AUV such as navigation, control, etc.

1.3 OBJECTIVES OF THE THESIS

Once the motivation and the most relevant aspects of the problem addressed in this thesis have been established, the main objective of this work can be stated as follows:

To endow an autonomous underwater vehicle (AUV) with the capability to incrementally map an unexplored environment, while planning online collision-free paths; such paths should be 3-dimensional (3D), safe, and feasible (doable) according to the vehicle's motion constraints. This will contribute a further step towards better and more reliable AUVs for the new and potential applications.

This objective can be separated into the following specific objectives:

Review of the Path/Motion Planning Literature: To conduct a review of the most relevant path/motion planning methods, which have been used for different robotic systems including terrestrial, aerial, and, especially, marine vehicles. This will allow identifying the most appropriate approaches for meeting the aforementioned constraints for the intended applications.

Planning Feasible Motions for AUVs: To propose a motion planning method that takes into account the AUV's motion constraints for both 2D and 3D movements. This will permit generating more feasible motions according to the AUV's capabilities. Furthermore, such motions will be more likely to be followed by the vehicle, thus minimizing unexpected maneuvers.

Online Mapping and Path/Motion Planning for AUVs: To propose a framework that allows an AUV to safely navigate through initially unexplored environments. This implies that the vehicle must be capable of incrementally mapping the surroundings while, simultaneously and online, planning safe and feasible paths.

Simulation and In-water Validation: To extensively test and validate the proposed approach in different simulated and real-world scenarios.

1.4 THESIS OUTLINE AND CONTRIBUTIONS

This manuscript is organized in a way that presents the incremental and progressive development of this thesis. In order to contextualize and better understand the real impact of such developments, Chapter 2 firstly makes a review of the most relevant methods, techniques, and applications that compose the state of the art, not only in what has to do with path/motion planning, but also in relation with the approaches currently applied with AUVs. Then, based on the new and potential applications intended for AUVs that were mentioned above, and considering their requirements, the main contributions of this thesis are gathered and distributed throughout this document as follows:

1) The incorporation of motion constraints into the AUV path planner, which acts as a mechanism to avoid generating unfeasible paths, thus reducing the number of unexpected maneuvers. This includes those constraints required for constant depth missions that only take into account 2-dimensional (2D) horizontal motions, which are discussed in Chapter 3. But it includes as well more challenging and variable depth missions that require analysing and combining the vehicle turning and ascending/descending limitations, which are addressed in Chapter 4.

2) An efficient approach for approximating the risk associated with a path. There are situations in which considering the vehicle motion constraints does not avoid risky maneuvers, especially when navigating in close proximity to nearby obstacles. In such cases, the AUV path/motion planner must lead the vehicle to navigate at a safe and conservative distance from its surroundings, yet without discarding any possible solution. Different strategies to estimate the risk associated with a vehicle configuration, and therefore with a path, are presented and compared in Chapter **??**.

3) The capability of an AUV to efficiently (re)plan feasible and safe paths, i.e., under motion constraints and taking into consideration the associated risk, in scenarios where the environment is incrementally discovered. This clearly means that the path/motion planner has to operate under online computation constraints, which can only be done by combining existing approaches and newly proposed strategies, such as *opportunistic state checking* and *reuse of the last best known solution*. The complete online planning approach will be presented in Chapter **??**.

4) A framework that allows an AUV to incrementally map an environment undiscovered previously, while simultaneously (re)planning a path that must be followed by the vehicle to safely cross and explore the surroundings. This framework integrates all previous stages and characteristics, in order to finally endow the AUV with the required capabilities for the intended applications, and will also be presented in Chapter **??**.

5) An extensive validation of the proposed framework is presented in Chapter 5. This includes simulated and in-water trials in different real-world scenarios, where an AUV had to navigate through initially unexplored environments, while gathering different kind of information such as acoustic and optical data. These tests were conducted with two different AUVs: the Sparus II from the University of Girona, and the AsterX from the

French Research Institute for Exploitation of the Sea (Ifremer)[1]. The results include planning safe and feasible paths to move through both artificial and natural marine structures, as well as the autonomous survey replanning for gap filling and target inspection.

Finally, concluding remarks and directions for further research are given and discussed in Chapter 6.

1 Technical details about both vehicles can be found in Appendix A.

2

STATE OF THE ART

Although the major research efforts in this thesis were focused on developing, extending and using path/motion planning methods for AUV applications, their validation with an AUV in real-world scenarios involved other fields of study. These included but were not limited to: navigation, perception, mapping and control. An extensive presentation of the state-of-the-art background of all these areas would not only prove to be lengthy and diffuse, but would also be out of the scope of this work. Nonetheless, this chapter is dedicated to contextualizing the contribution of this thesis. In order to do so, it firstly provides a survey of the techniques available for solving specifically path/motion planning problems. Secondly, it discusses the most common extensions used when the vehicle motion capabilities have to be taken into consideration or when dealing with online computation constraints. Finally, it presents a review of the algorithms and extensions that have been used with AUVs particularly.

2.1 PLANNING COLLISION-FREE PATHS OVER THE C-SPACE

Even though first works on path/motion planning appeared in the late 60's **[Nilsson1969]**, it was only until the 80's, when Lozano-Perez introduced the concept of the *configuration space* **[Lozano-Perez1983, Lozano-Perez1984, Lozano-Perez1987]**, that this field of study became active. The configuration space (or C-Space) establishes the set of all possible *configurations* that a robot can adopt when executing tasks in the workspace. While the workspace, $\mathcal{W}$, is typically defined as $\mathbb{R}^n$ (where $n = 2, 3$ for 2D and 3D motion, respectively), the C-Space, $\mathcal{C}$, depends on the robot motion capabilities. For example, a robot considered to be a point that moves in a plane (i.e., $\mathcal{W} = \mathbb{R}^2$) requires two coordinates in order to specify its *configuration*, q, which is equal to $[x, y]^T$, thus $q \in \mathcal{C} = \mathbb{R}^2$.

However, if the robot is considered to be a rigid body that moves in a plane (i.e., in the same $\mathcal{W}$), it requires three coordinates; these would describe not only its position, but also its orientation, therefore its *configuration* is now $q = [x, y, \psi]^T$, which is commonly expressed as $q \in \mathcal{C} = SE(2) = \mathbb{R}^2 \times SO(2) = \mathbb{R}^2 \times \mathcal{S}$. A similar scenario occurs with a single rigid-body robot that operates in 3D workspaces, in which $q = [x, y, z]$, so that $q \in \mathcal{C} = \mathbb{R}^3$ when only the robot´s position is considered, or $q = [x, y, z, \phi, \theta, \psi]$, so that $q \in \mathcal{C} = SE(3) = \mathbb{R}^3 \times SO(3)$ when considering both position and orientation. Finally, when the robot is an articulated rigid body system, for instance a manipulator arm, a given *configuration* is described by the values of a set of n generalized coordinates $q = q_1, \ldots, q_n$, corresponding to each of the robotic arm degrees of freedoms (DOFs).

The solution to a simple path/motion planning problem, which requires connecting a start and a goal configuration, q_{start} and q_{goal}, is a continuous path $p : [0, 1] \rightarrow \mathcal{C}$, such that $p(0) = q_{start}$ and $p(1) = q_{goal}$.

However, a robot generally conducts tasks in environments that contain obstacles, which normally are to be avoided. For this reason, the C-Space is subdivided into *free space* ($\mathcal{C}_{free}$) and the *obstacle region* ($\mathcal{C}_{obs}$), meaning that $\mathcal{C} = \mathcal{C}_{free} \cup \mathcal{C}_{obs}$. Therefore, a collision-free path is defined as a continuous path $p : [0, 1] \rightarrow \mathcal{C}_{free}$. This concept is illustrated in Figure 4.

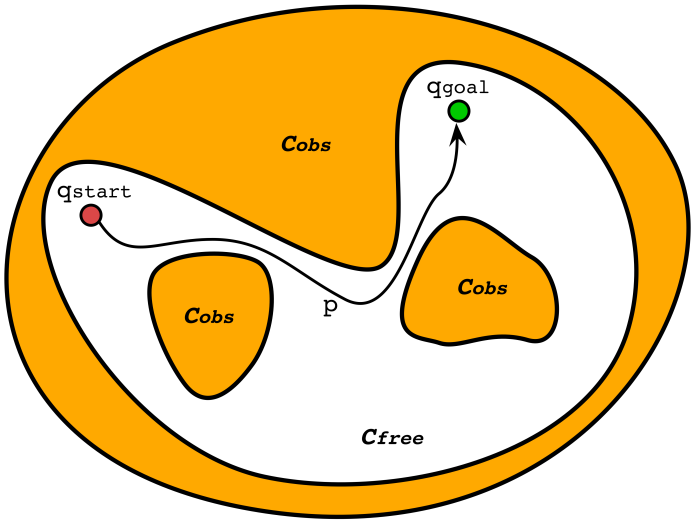


Figure 4: The basic path/motion planning problem seeks to connect a start configuration (q_{start}) and goal configuration (q_{goal}) with a continuous collision-free path $p : [0, 1] \rightarrow \mathcal{C}_{free}$.

2.2 BUG-BASED METHODS

Bug-based methods represent one of the earliest reactive and sensor-based path planning approaches, in which a mobile robot moves in the plane towards a global goal, while having a limited (local) knowledge of the environment. The algorithms included in this group are based on two basic behaviors: go straight towards the goal and follow the obstacles' boundary. Lumelsky and Stepanov presented the first of these algorithms, known as Bug1 and Bug2, that mainly rely on tactile or zero range sensors to perceive the obstacles [**Lumelsky1987**]. Later, Kamon et al. introduced the Tangent Bug algorithm, which is an extension that uses non-zero range sensors [**Kamon1996**]. All of them are classified as complete algorithms, which means that they find a solution when one is possible or, otherwise, they report when there is no solution. Furthermore, they assume the robot is a point that moves in a 2D workspace, and that it is capable of knowing its position and calculating its distance to the goal.

Bug1 [**Lumelsky1987**] is the most basic algorithm that uses the aforementioned behaviors. With this method, the robot moves from the start position (q_{start}) towards the desired goal position (q_{goal}) until it reaches the goal or until it finds an obstacle (detected with its tactile sensors). When the latter situation occurs, the robot stores its current position and marks it as the *hit point* (H_i). Then it changes its motion mode (behavior) to completely circumnavigate the obstacle. While the vehicle follows the obstacle boundary, it continuously calculates the distance to the goal in

order to determine the position that corresponds to the minimum possible distance. This is marked as the *leave point* (L_i). Once the robot reaches the *hit point* again (i.e., after it has traveled all the obstacle contour), it goes back to the *leave point* and there changes its behavior, once again, in order to move towards the goal. This procedure is repeated until reaching the specified goal. Figure 5 depicts an example of this algorithm solving a start-to-goal query.

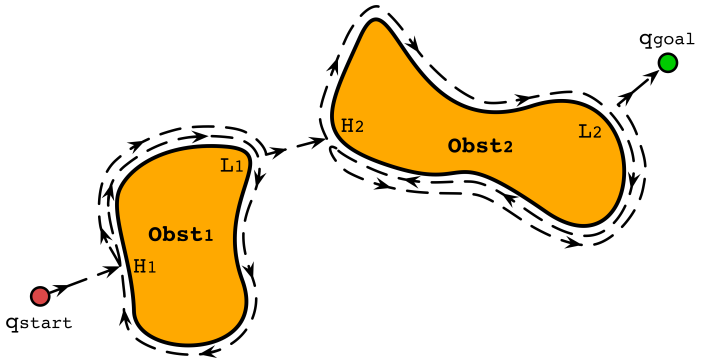
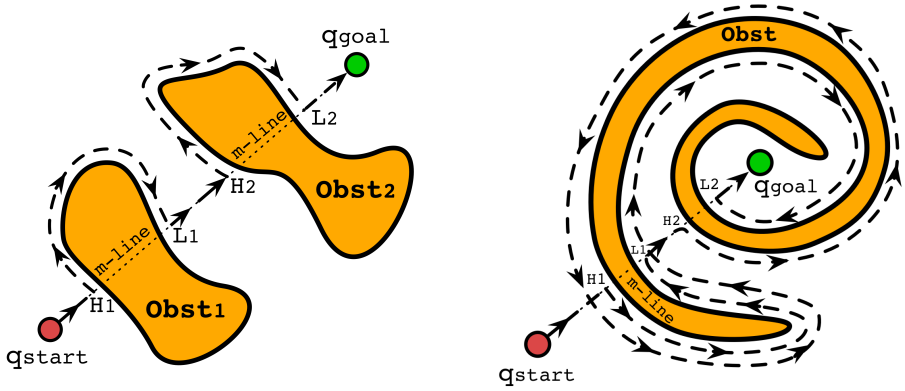


Figure 5: Reactive and sensor-based method Bug1

Bug2 [**Lumelsky1987**] attempts to be an improved version of the previously explained algorithm. It establishes a straight line, sometimes referred as the *m-line* [**Choset2005**], which connects the start and goal positions. With this algorithm, the robot starts moving towards the desired goal by following the *m-line* until it reaches the goal or finds an obstacle. As occurs with Bug1, if the robot is dealing with an obstacle, it first marks that position as the *hit point* (H_i) and then changes its behavior to circumnavigate the obstacle. However, with Bug2 the robot does not necessarily travel the entire obstacle boundary, but instead stops when reaching another point in the *m-line*; if such a point is closer to the goal than the previous *hit point*, the robot marks this position as *leave point* (L_i) and, from there, continues following the *m-line* towards the goal. This procedure is repeated until reaching the specified goal. Even though Bug2 does not require to completely circumnavigate the obstacles, there are environments where the robots do require to travel the entire boundary; in these cases Bug2 may generate longer paths than those calculated by Bug1, and therefore is less efficient. Figure 6 depicts an example of Bug2 solving start-to-goal queries in both simple and complex scenarios.

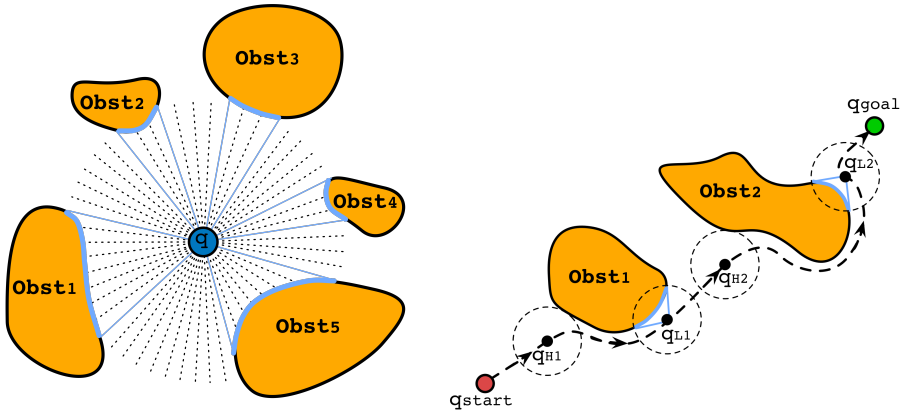
Finally, the Tangent Bug algorithm [**Kamon1996**] was proposed as an improved alternative for Bug1 and Bug2. In this version, the robot is assumed to be equipped with a non-zero range sensor, which permits detecting in advance not only the obstacles, but also their continuous boundaries. This way, when the robot is navigating towards the goal and faces an obstacle, it can determine the discontinuities in the boundaries or the limits of the perceived area, which are marked as *endpoints*. Then, in order to avoid the obstacle, the robot checks which of the *endpoints* minimizes the distance to the goal and starts moving towards it. The procedure is repeated until the obstacle is no longer perceived, at which point the robot can continue mov-



(a) Bug2 simple scenario (b) Bug2 complex scenario

Figure 6: Reactive and sensor-based method Bug2

ing towards the goal following a straight line. Figure 7 depicts an example of Tangent Bug solving a start-to-goal query.



(a) Boundaries detection (b) Execution

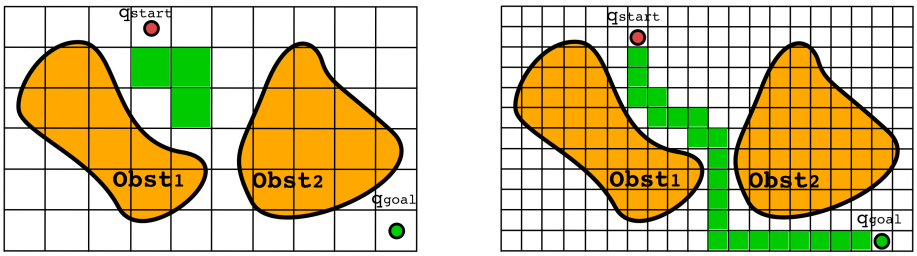
Figure 7: Reactive and sensor-based method Tangent Bug

2.3 SEARCH-BASED (GRID-BASED) PATH/MOTION PLANNING

Search-based planning is a path/motion planning approach that utilizes graph search methods to find collision-free paths over a discrete version of $\mathcal{C}$. For doing so, these methods overlay a grid on $\mathcal{C}$ and assume that each collision-free configuration corresponds to a point on the grid, which is why they are also called grid-based methods. Over that grid, the robot is allowed to move from one point to any other adjacent point as long as the line between them is proved to be collision-free (i.e., is contained within $\mathcal{C}_{free}$). Hence, using this approach to solve, for instance, a start-to-goal query requires coping with two problems: how to correctly discretize $\mathcal{C}$ to establish the grid, and how to search a path from the start point to the goal point (configuration) over such a grid.

For the first problem, it is important to correctly define the grid resolution, which mainly depends on the environment and the problem's

requirements. Coarser grids, for example, will permit faster searches, but may fail to find paths when dealing with narrow passages in $\mathcal{C}_{free}$ (see Figure 8). Finner grids, on the other hand, will allow solving queries in more complex scenarios, but may be computationally too expensive for online applications. Once the grid is established, there are different methods to calculate an optimal path, some of which are described in this section.



(a) Coarse grid (b) Fine grid

Figure 8: Grid-based method solving a start-to-goal query. (a) Although coarser grids decrease the computing time, they may fail when dealing with narrow passages where finer grids could succeed (b).

2.3.1 *Dijkstra's Algorithm*

Presented in 1959, Dijkstra's algorithm [**Dijkstra1959**] is one of the first widely known methods used to find a path in a graph from one node to another. The algorithm uses a weighted graph $\mathcal{Q}$[1] to determine which path that connects two nodes has of the lowest cost. While this method has been used in different areas and applications such as network routing protocols, in robotics it has been applied to path/motion planning problems. There, the associated edge cost may correspond to different optimization metrics such as distance, energy, time, etc., that can be considered when calculating the optimal path for a start-to-goal query.

In order to find the optimal path, the algorithm iteratively executes the following steps until reaching the goal (q_{goal}): 1) Setting an initial cost to all nodes, specifically zero to the initial one (q_{start}) and infinity to all others. 2) Establishing q_{start} as the current node and marking the rest as unvisited. 3) Calculating the cost for the current node's unvisited neighbors (equal to the current node's cost plus the edge to the neighbor cost), as well as updating any previously calculated node cost if the new value is lower. For example, if node B cost was x when current node was A, but the new cost is y when current node is C, and $y < x$, then node B cost will be now y. 4) After calculating the cost of all its unvisited neighbors, current node is marked as visited. 5) If the q_{goal} has been marked as visited, the algorithm has finished, otherwise, the new current node will be the unvisited node with the lowest cost, and will be processed as indicated from step 3). Finally, the least cost path can be obtained with backtracking. Figure 9a presents an example of executing this method.

1 A weighted graph is one in which numerical values, or weights, are assigned to its nodes and edges.

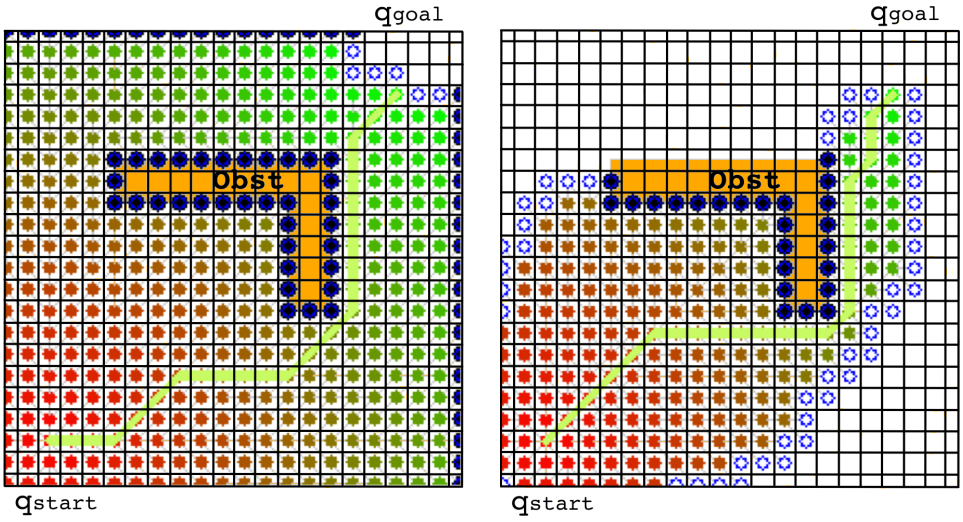
2.3.2 *A**

In 1968, Peter Hart et al. described an extension of Dijkstra's algorithm called A* [**Hart1968**], which incorporates a heuristic that permits estimating the cost of paths from any node of the graph to the goal. Because of this characteristic, A* is considered an informed search algorithm; this means that it always attempts to find the path by firstly evaluating those nodes with the minimum estimated cost according to the heuristic. As occurs with Dijkstra's algorithm, A* starts from a weighted graph $\mathcal{Q}$, in which each node represents a different configuration contained in $\mathcal{C}_{free}$ and the edges correspond to collision-free paths between configurations. Then, it builds a search tree, rooted at q_{start}, by expanding different paths, one step at a time, until one of them ends at the desired q_{goal}. The main difference with respect to Dijkstra's algorithm is the order in which each partial path is expanded. In the case of A*, it expands the node q_i that minimizes the total cost $f(q_i) = g(q_i) + h(q_i)$, which combines the cost required to reach the node from the start configuration $g(q_i)$ with the estimated heuristic cost from the node to the goal $h(q_i)$.

While both Dijkstra's algorithm and A* can generate optimal solutions, the latter can result in a more efficient search by reducing the number of nodes required to be visited in order to determine the solution path (see Figure 9b). However, it is important to know that an incorrect heuristic will also provide a valid solution, but it may be suboptimal. For this reason and in order to produce an optimal path, the heuristic has to be optimistic, or admissible, which means that the estimated cost to the goal has to be lower or equal to the real cost [**Choset2005**]. Finally, it is also important to note that in case no heuristic is provided, i.e., $h(q_i) = 0$, A* behaves as Dijkstra's algorithm. Algorithm 1 presents the pseudocode for A* and also provides a general idea for Dijkstra's algorithm explained in the previous section.

2.3.3 *Dynamic A* (D*)*

The aforementioned search-based algorithms, at least in their original versions, are intended for planning paths in static environments. Nonetheless, there are situations, especially in mobile robotics applications, in which elements of the environment may change. For those cases, Anthony Stentz proposed the dynamic A* (D*) [**Stentz1994**, **Stentz1995**], an incremental search-based algorithm that plans collision-free paths using a similar strategy as A*. The difference is that it also allows to replan according to changes observed in the surroundings while the robot follows the path to the goal. Its most important characteristic is that it locally repairs the path, which is more efficient that invoking multiple times A* to find a new valid path. Contrary to Dijkstra's algorithm and A*, that both search paths from the start to the goal configuration, D* expands nodes by searching backwards from the goal until the node to be expanded coincides with the start configuration, at which time the search is concluded. At the moment, D* and some of its variants are probably the most used search-based meth-



(a) Dijkstra algorithm (b) A*

Figure 9: Grid-based methods solving a start-to-goal query. (a) Dijkstra's algorithm requires an exhaustive search to determine the shortest path to the goal. (b) A* requires exploring less cells since it uses a heuristic to guide the search. Image credit: modified from Wikipedia.

Algorithm 1 : A*

Input :

q_{start}: Start configuration.

q_{goal}: Goal configuration.

$\mathcal{Q}$: Graph of configurations q, such that $q \in \mathcal{C}_{free}$

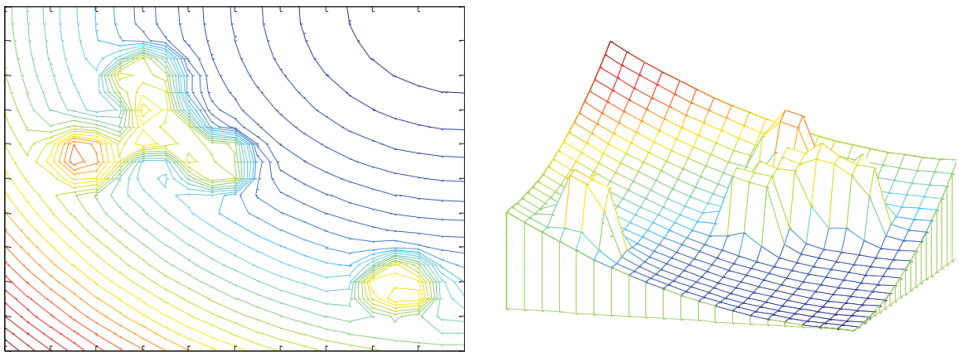
begin
  forall $q \in \mathcal{Q}$ **do**
    $g(q) = \infty$
  $g(q_{start}) = 0$
  UNVISITED = `PriorityQueue()`
  UNVISITED.`insert`$(q_{start}, g(q_{start}) + h(q_{start}, q_{goal}))$
  while $\underset{q \in \text{UNVISITED}}{\arg\min} \; (g(q) + h(q, q_{goal})) \neq q_{goal}$ **do**
    $q \leftarrow$UNVISITED.`popWithMinCost()`
    forall $q' \in \mathcal{Q}$.`Neighbors(`q`)` **do**
      if $g(q') > g(q) + c(q, q')$ **then**
        $g(q') = g(q) + c(q, q')$
        UNVISITED.`insert`$(q', g(q') + h(q', q_{goal}))$

ods in mobile robotics. Some of those extensions and applications will be presented in Section 2.7.

2.4 POTENTIAL FIELDS

Even though search-based methods have proved to be successful in 2D and 3D workspaces for some terrestrial, aerial and even aquatic robotic applications, those exhaustive methods suffer from scalability issues in problems involving high-dimensional configuration spaces. Another important drawback is the necessity of establishing a grid over $\mathcal{C}_{free}$, which means discretizing the C-Space, thus limiting the possible and available solutions according to the chosen resolution. An alternative approach is the use of potential functions.

Originally proposed by Oussama Khatib in 1985 [**Khatib1985**, **Khatib1986**], potential functions, also known as *potential fields*, constitute a reactive approach for path/motion planning, which attempts to guide a robot from an initial configuration to a goal configuration while avoiding obstacles. A *potential field* basically defines a real-valued function $U : \mathbb{R}^m \to \mathbb{R}$, which is composed of an attractive component $U_a(q)$ that pulls the robot towards the goal, and a repulsive component $U_r(q)$ that pushes the robot away from the obstacles. This function can be viewed as the total energy $U(q) = U_a(q) + U_r(q)$, which means that the total force applied by the potential field to the robot is defined as the negative of the vector gradient, i.e., $f(q) = -\nabla U(q)$, where $\nabla U(q) = DU(q)^T = \left[\frac{\partial U}{\partial q_1}(q), \ldots, \frac{\partial U}{\partial q_m}(q)\right]^T$. In other words, the gradient $\nabla U(q)$ establishes the force required at any $q \in \mathcal{C}$, in order to guide the robot throughout a collision-free path towards the goal. Figure 10 presents an example of the two components of a potential field and the total potential field.



(a) Top view (b) Perspective view

Figure 10: Example of potential field. The obstacles can be observed as a repulsive component of the whole potential. Image credit: Liu et al. [**Liu2016**].

Once the potential function and the corresponding gradient have been established, the solution path to a start-to-goal query can be incrementally obtained by using the gradient descent, which is a widely-known algorithm in solving optimization problems. Having q_{start} as the initial configuration, the algorithm iteratively calculates a new configuration by moving in the direction opposite to the gradient until the gradient is zero. The pseudocode for this method is presented in Algorithm 2.

Algorithm 2 : Gradient Descent

Input :
q_{start}: Start configuration.
$\nabla U(q)$: Gradient of the potential field.

```
1 begin
2     q(0) = q_start
3     i = 0
4     while ∇U(q(i)) ≠ 0 do
5         q(i+1) = q(i) − ∇U(q(i))
6         i = i+1
```

However, the gradient descent algorithm using potential functions as explained above does not guarantee finding or converging to a solution for a start-to-goal query. This happens because it may reach a local minimum of $U(q)$ that may not correspond to q_{goal}. To deal with such situations, Barraquand and Latombe proposed the randomized path planner (RPP) [**Barraquand1990**, **Barraquand1991**], which is an algorithm that makes use of the gradient descent together with random walks and backtracking in order to avoid issues related to local minima. However, it is important to note that RPP effectiveness is highly dependent on parameter tuning. Another important aspect to highlight is that RPP was one of the first methods to use a stochastic or random approach for path/motion planning; algorithms with this characteristic will be discussed more in detail in Section 2.6.

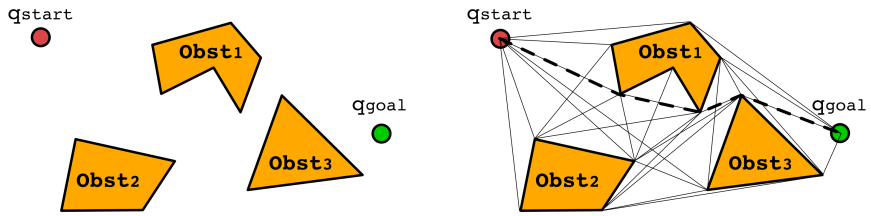
2.5 ROADMAPS

So far, the methods and algorithms presented above attempt to solve a single start-to-goal query, which means they calculate a path that connects a start configuration and a goal configuration. For doing so, they incrementally search a path towards the goal while avoiding, at the same time, collisions with the obstacles. Nonetheless, there are some applications in which the path/motion planner is intended to solve more that one query. For those cases, it makes sense to have a map that contains the information about all feasible routes, and that can also be used more than once to solve multiple start-to-goal queries. There are different alternatives to define a map that can be used for path/motion planning, including topological, geometric, and grid-based representations.

This section reviews methods that use a class of topological maps called *roadmaps* [**Canny1993**, **Latombe1991**]. A roadmap (RM) is defined as a subset of the C-Space that results from the union of curves, in which any q_{start} and q_{goal} contained in $\mathcal{C}_{free}$ can be connected by a path that meets the following properties [**Chosset2005**]: 1) **Accessibility** - there is a path from $q_{start} \in \mathcal{C}_{free}$ to some $q'_{start} \in \mathcal{RM}$. 2) **Departability** - there is a path from some $q'_{goal} \in \mathcal{RM}$ to $q_{goal} \in \mathcal{C}_{free}$. 3) **Connectivity** - there is a path in $\mathcal{RM}$ that connects q'_{start} and q'_{goal}.

2.5.1 *Visibility Graphs*

Visibility graphs are one of the alternatives in defining a roadmap. Assuming a 2D C-Space with polygonal obstacles, the set of the visibility graph nodes (v_i) is composed of q_{start}, q_{goal}, and all the vertices of the obstacles. Its edges, e_{ij}, are straight-line segments that can connect any possible combination of two nodes v_i and v_j, without colliding with the obstacles ($e_{ij} \in \mathcal{C}_{free}$). Once the visibility graph is fully defined, the solution path can be obtained by conducting any graph-based search method, such as those explained in Section 2.3. While Figure 11 depicts an example of a visibility graph and a start-to-goal query solution over it, Algorithm 3 presents the pseudocode to build a visibility graph.



(a) (b)

Figure 11: Example of a visibility graph. (a) A 2D C-Space that contains polygonal obstacles, a start configuration, and a goal configuration. (b) A roadmap is built by connecting any possible combination of two vertices without colliding with the obstacles. The start-to-goal query is solve using a graph search method.

Algorithm 3 : Visibility Graph

Input :
q_{start}: Start configuration.
q_{goal}: Goal configuration.
World: n Obstacles.
Output :
Roadmap ($\mathcal{RM}$): Visibility Graph(VG) = (V, E).

```
1  begin
2      V = {}
3      E = {}
4      for i = 1 : n do
5          V.addNodes(Obst(i).getVertices())
6      for i = 1 :V.getNumNodes() do
7          for j = 1 :V.getNumNodes() and j ≠ i do
8              v_i ← V(i)
9              v_j ← V(j)
10             e_ij ← findStraightLine(v_i, v_j)
11             if isCollisionFree(e_ij) then
12                 E.addEdge(e_ij)
```

2.5.2 *Generalized Voronoi Diagrams*

Another alternative to build a roadmap for path/motion planning is the generalized Voronoi diagram (GVD). The GVD is defined for a set of points called *sites*; given a particular site, the set of points closest to it is called a *Voronoi region*. Finally, the Voronoi diagram is the set of points that are equidistant to at least two sites [**Aurenhammer1991**]. In path/motion planning applications, the GVD defines the sets of points equidistant to at least two obstacles. This means that the sites are the center of the obstacles to be avoided, and the edges correspond to the possible channels that maximize the distance to the obstacles [**Choset2005**]. Figure 12 displays an example of a planar GVD.

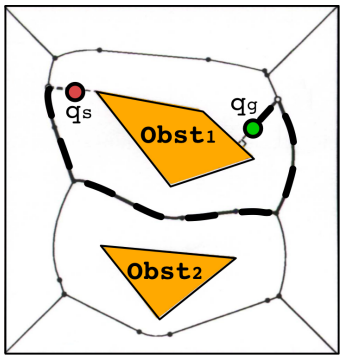


Figure 12: Example of a Voronoi diagram

2.6 SAMPLING-BASED ALGORITHMS

Some of the methods presented in previous sections, such as potential fields, roadmaps, and cell decomposition, require an explicit representation of $\mathcal{C}_{free}$ in order to find a collision-free path from a start configuration to a goal configuration. Nonetheless, there are some situations in which building such a representation is not possible or is computationally intractable, for instance when dealing with high-dimensional configuration spaces. For those cases, sampling-based methods have been demonstrated to be effective.

Firstly developed during the 90's, *sampling-based algorithms* are an alternative approach that aims to create a sampled (discrete) representation of C-Space that captures the connectivity of the regions of $\mathcal{C}_{free}$. These methods exploit the fact that, while explicitly building $\mathcal{C}_{free}$ is expensive, checking a given configuration for collisions can be done quickly. To build such a discrete representation, they firstly generate random samples q_{r_i} from C-Space, which are checked for collision in order to ensure that $q_{r_i} \in \mathcal{C}_{free}$. Secondly, they interconnect the random and collision-free configurations, thus establishing different routes (paths) to solve single or multiple start-to-goal queries. This two-stage (sampling and connecting) approach that uses a collision checking routine to validate samples, also allows generalizing the path/motion planning problem. This is done by separating the algorithm from the specific geometric representation of the environment.

However, there is one important characteristic to be considered when using sampling-based algorithms. Contrary to the methods mentioned previously, sampling-based algorithms weaken the completeness guarantee, which means that they are not capable of notifying if a solution exists. Nonetheless, given that many of these methods are based on random sampling, which is dense with probability one [**LaValle2006**], this also implies that with enough points (samples), if a solution exists then the probability of finding it converges to one. In other words, if the algorithm runs for a sufficient amount of time, it will find a solution if there is one. This property is called *probabilistic completeness* [**Kavraki1996**, **Barraquand1997**, **Kavraki1998**].

These characteristics have led sampling-based algorithms to be considered the state-of-the-art approach for solving various path/motion planning problems. Albeit there are several methods and variants used nowadays, it is possible to identify those that, at the time, were pioneers and the most relevant ones. This section presents such methods and classifies them according to their capability to solve single or multiple start-to-goal queries.

2.6.1 *Multiple-query Methods*

As it was explained in Section 2.5, roadmaps are data structures that contain all feasible routes that can be used more than once to solve multiple start-to-goal queries. Likewise, there is a sampling-based method called probabilistic roadmap (PRM) that creates a graph attempting to represent the connectivity of $\mathcal{C}_{free}$. PRM was developed simultaneously at Stanford [**Kavraki1994**, **Kavraki1994a**] and Utrecht [**Overmars1994**], and was jointly presented in 1996 [**Kavraki1996**].

Nowadays, PRM is one of the most representative sampling-based algorithms. It is mainly composed of a *preprocessing phase* and a *query phase*. During the first phase, the algorithm builds a roadmap, or undirected graph $G = (V, E)$. The set of nodes (V) contains n collision-free samples q_{r_i} (i.e., $q_{r_i} \in \mathcal{C}_{free}$) that are randomly obtained from an uniform distribution[2]. The set of edges (E) corresponds to the collision-free paths from each node q_{r_i} to its k closest nodes q_{r_j}; the connecting paths are calculated by a local planner that checks them for collisions. Algorithm 4 presents the pseudocode for the *preprocessing phase*.

Once the roadmap (graph) has been built, the *query phase* attempts to find a collision-free path between the provided q_{start} and q_{goal}. To do so, PRM tries to connect q_{start} and q_{goal} to their k closest nodes in the graph G. If the connections are successful, a search-based method (e.g., A*, Dijkstra, etc.) attempts to find the shortest path over the graph. If the connections for q_{start} and q_{goal} to the graph are not possible, or if the search-based algorithm fails to find a solution path for the query, it does not necessarily mean that a solution does not exist. As explained above, most sampling-based methods, including PRM, are probabilistic complete,

2 Uniform distribution is the basic form to obtain the random samples, at least in the basic version of the algorithm. Other distributions have been used in some extensions of the original method.

Algorithm 4 : PRM, preprocessing phase

Input :
n: Number of nodes of the roadmap.
k: Number of closest nodes to attempt connection.
$\mathcal{C}$: C-Space
Output :
Probabilistic roadmap (PRM): $G = (V, E)$.

```
1  begin
2      V = {}
3      E = {}
4      while |V| < n do
5          q_rand = C.generateRandomConf()
6          if C.isCollisionFree(q_rand) then
7              V.addNode(q_rand)
8      for q_ri ∈ V.getNodes() do
9          for q_rj ∈ C.getClosestNodes(q_ri, k) do
10             e_ij ← findPath(q_ri, q_rj)
11             if C.isCollisionFree(e_ij) then
12                 E.addEdge(e_ij)
```

which means that more time may be required to find a solution, if there is one. In this case, it would imply that the number of nodes n have to be increased, as that will generate a more dense probabilistic roadmap.

An important number of variants and extensions have been proposed to deal with different situations in which the originally proposed PRM may fail [**Choset2005**], [**LaValle2006**]. A typical example includes a workspace that creates a C-Space with narrow passages. In such a case, a common approach would be oversampling the regions of interest (e.g., narrow passages), thus increasing the probability of finding a solution path. Other extensions that have served as a base for the work developed throughout this thesis, will be discussed in Section 2.7. Finally, Figure 13 depicts PRM solving a start-to-goal query in a 2D scenario for a point-like robot.

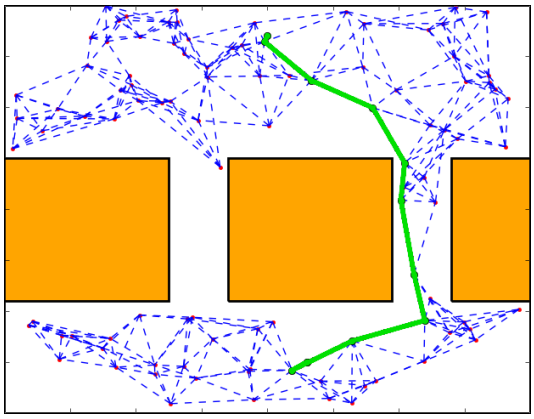


Figure 13: Example of a probabilistic roadmap (PRM) used to solve a start-to-goal query. The PRM was built with 100 random samples, which were attempted to connect to their 6 nearest neighbors.

2.6.2 *Single-query Methods*

There is another group of sampling-based algorithms that is devoted to solving a single start-to-goal query. Such methods conduct an incremental search, which in most cases is achieved by expanding a tree of randomly sampled configurations. The search is generally biased to finding a solution path for one particular start-to-goal query, rather than attempting to completely represent the connectivity of $\mathcal{C}_{free}$. These methods are *unidirectional* when a single tree is expanded from q_{start} until reaching q_{goal} or vice versa. Or, they are *bidirectional* if two trees are expanded, one from q_{start} and other from q_{goal}, until both trees meet at a common point. Choosing q_{start} or q_{goal} as the tree's root depends on the specific problem and scenario, since there can be situations in which expanding from the goal is easier (less constrained) that doing it from the start configuration. There are different sampling-based single-query methods, however, this section presents a brief review of those considered the most relevant.

2.6.2.1 *Randomized Path Planner (RPP)*

Section 2.4 presented an approach for guiding a robot from its current position towards the goal position, by following the negative gradient of an artificial potential field. As mentioned there, one of the main disadvantages of this approach is that the vehicle can get trapped in a local minimum that may not correspond to the specified goal. In order to deal with such situations, Barraquand and Latombe proposed in 1990 the randomized path planner (RPP) [**Barraquand1990**, **Barraquand1991**], which uses the strategy of potential fields, but also incorporates random walks to escape local minima. RPP is commonly acknowledged as the first randomized algorithm. Albeit RPP has proved to be successful in many applications, it may fail when coping with narrow passages.

2.6.2.2 *The Ariadne's Clew Algorithm (ACA)*

Presented in 1993, Ariadne's clew algorithm (ACA) builds a tree from q_{start} by interleaving the *exploration* of the C-Space and the *search* of a connection between the tree and q_{goal} [**Bessiere1993**, **Mazer1998**]. During the exploration, the algorithm places a random collision-free configuration as far as possible from the others, therefore guaranteeing resolution completeness. New configurations are selected by using genetic optimization methods. These correspond to those from which a connection to q_{goal} is attempted. The main drawback of this approach is that the exploration of the C-Space is computationally expensive and also requires some parameter tuning.

2.6.2.3 *Expansive-Spaces Tree (EST)*

Proposed by David Hsu et al. in 1997, expansive-spaces tree (EST) [**Hsu1997**, **Hsu1999**, **Hsu2000**, **Hsu2002**] is a single-query method that incrementally builds a tree over the C-Space by interleaving its *construction* and *expansion*. In contrast to what occurs with PRM that computes a roadmap attempting to represent the whole $\mathcal{C}_{free}$, EST tries to sample the region of $\mathcal{C}$ that is

relevant in order to solve the specific start-to-goal query. To do so, during the *construction* phase the algorithm selects the node q to be extended in a way that prioritizes less explored regions, and then it randomly samples a configuration q_{rand} around q. Then, using a local planner, EST calculates the path that connects q and q_{rand}. In case that both q_{rand} and the calculated path are proved collision-free, they will be added to the tree, thus *expanding* it.

2.6.2.4 *Rapidly-exploring Random Tree (RRT)*

Within the group of sampling-based single-query algorithms, there is one method that is considered the state-of-the-art, which has been extended, modified and applied to a wide range of applications. This method is known as rapidly-exploring random tree (RRT) and it was firstly presented by Steven LaValle in 1998 [**LaValle1998**]. RRT is a tree-based algorithm that has different properties such as rapid exploration of the C-Space, probabilistic completeness, ease of implementation, just to mention some. In 1999, LaValle and Kuffner formally presented the RRT as a path/motion planning method capable of dealing with both geometric and motion constraints. They also proposed a greedy approach to decrease the time required to find a solution by interleaving a random growing of the tree with a biased growing towards the goal [**LaValle1999**, **LaValle2000**, **LaValle2001**]. They later proposed a bidirectional version called RRT-Connect that extends the basic concept by constructing two RRTs towards each other [**Kuffner2000**].

Similar to ACA and EST, the basic RRT algorithm is mainly composed of two main procedures, *sample* and *extend*. Algorithm 5 presents the first of them, where the tree is incrementally built until a stop condition occurs; such a condition can either be finding a feasible path that reaches the goal close enough, or that a maximum number of iterations has been completed. In each iteration, the RRT algorithm attempts to extend the tree towards a randomly sampled configuration q_{rand}. For doing so, the second procedure described in Algorithm 6 finds q_{near}, which is the nearest configuration to q_{rand} in the tree (line 2). Then, a local planner calculates a path of length δ from q_{near} towards q_{rand} (line 3); if the path is proved collision-free, the algorithm generates a new configuration q_{new}, which together with the calculated path are added to the tree (lines 5-6). A typical growth process of an RRT algorithm can be clearly observed in Figure 14a, where no goal has been specified and the tree attempts to explore uniformly the C-Space. Figure 14b depicts the RRT algorithm solving a start-to-goal query.

2.6.3 *Optimal Planning*

At least in their original formulation, sampling-based algorithms do not guarantee that the calculated path is optimal with respect to a specified cost function. However, more recent contributions have attempted to cope with this situation. In 2010, Jaillet et al. presented the transition-based RRT (T-RRT), which calculates a low-cost path that follows valleys and saddle points in a costmap established over the C-Space [**Jaillet2010**]. To achieve this, it verifies the quality of the path by using the minimal work

Algorithm 5 : sampleRRT

Input :

q_{start} : Start configuration.

q_{goal} : Goal configuration.

$\mathcal{C}$: C-Space.

Output :

Rapidly-exploring Random Tree (RRT): $T = (V, E)$.

```
1 begin
2     V = {}
3     E = {}
4     V.addNode(q_start)
5     while not stopCondition(T, goal) do
6         q_rand = C.generateRandomConf()
7         extendRRT(T, q_rand)
```

Algorithm 6 : extendRRT

Input :

T: an RRT.

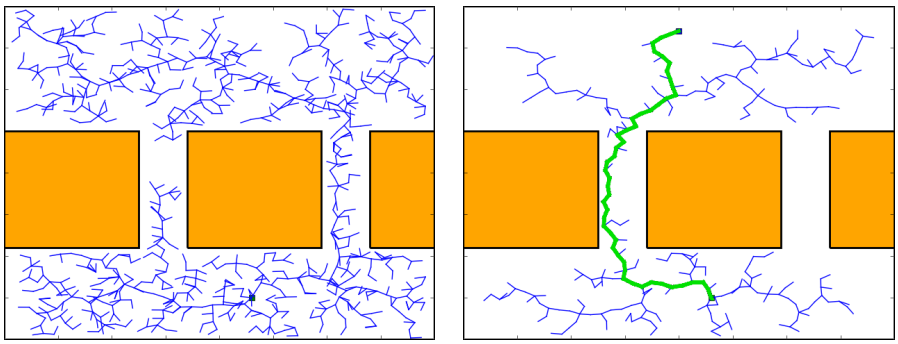
q_{rand}: configuration towards which RRT will be extended.

$\mathcal{C}$: C-Space.

Output :

Result after attempting to extend.

```
1 begin
2     q_near ← T.findNearestNeighbor(q_rand)
3     q_new, collision ←calcNewConf(q_near, q_rand, δ)
4     if collision = FALSE then
5         V.addNewNode(q_new)
6         E.addNewEdge(q_near, q_new)
7         return ADVANCED
8     else
9         return TRAPPED
```



(a) RRT expansion (b) RRT solving a query

Figure 14: Example of a rapidly-exploring random tree (RRT) in a 2D workspace. (a) The uniform and rapid exploration of the C-Space is one of the main characteristics. (b) The expansion of the tree stops once a solution has been found.

criterion. Its key principle is that positive variations of the cost function can be seen as forces acting against motion, and thus producing mechanical work. Contrasted to a standard RRT algorithm implementation, an additional transition validation is performed, which accepts or rejects new potential configurations before adding them into the solution tree.

Nonetheless, the state-of-the-art sampling-based method for calculating optimal paths is the asymptotic optimal RRT (RRT*). In 2010, Karaman and Frazzoli firstly introduced the RRT* algorithm and its concept of asymptotic optimality. This property states that the total cost of the solution, measured by a user-defined function, decreases as the number of samples increases [**Karaman2010a**]. In this approach, new configurations are connected to the closest and best configuration, i.e., the one that guarantees a minimum cost. Furthermore, an additional step of sample reconnection allows improving costs to surrounding configurations (see Algorithm 7). This concept was later extended by the same authors to the PRM in [**Karaman2011**], where they formally presented the RRT* and PRM* algorithms. Similar extensions have been done to other RRT-based algorithms as T-RRT and its respective T-RRT* version [**Devaurs2016**]. A typical growth process of an RRT* algorithm can be clearly observed in Fig. 15.

Algorithm 7 : extendRRT*

Input :
T: tree of collision-free configurations.
q_{rand}: state towards which the tree will be extended.
$\mathcal{C}$: C-Space.
Output :
Result after attempting to extend.

```
1  begin
2      q_near ← T.findNearestNeighbor(q_rand)
3      q_new, collision ←calcNewConf(q_near, q_rand, δ)
4      if collision = FALSE then
5          addNewNode(T, q_new)
6          Q_near ←findNearestNeighbors(T, q_new)
7          q_min_cost ←findMinCost(T, Q_near, q_new)
8          addNewEdge(T, q_min_cost, q_new)
9          reconnectNearNeighbors(T, Q_near, q_new)
10         return ADVANCED
11     else
12         return TRAPPED
```

2.7 EXTENSIONS AND APPLICATIONS

Most of the aforementioned methods have been widely used in real-world applications not only with manipulator arms, but also with different aerial, terrestrial and aquatic robotic systems. For doing so, some of these methods have been extended and adapted according to the specific applications' requirements. Considering the problem and objectives stated for this the-

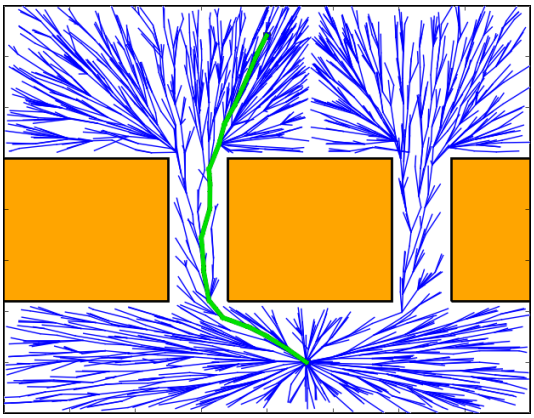


Figure 15: Example of an asymptotic optimal rapidly-exploring random tree (RRT*) in a 2D workspace. The RRT* preserves the uniform and rapid exploration of the C-Space as the standard RRT; however, it can be observed how the additional reconnection step reshapes the tree branches. The expansion of the tree does not stop once a solution has been found. Instead, it keeps improving the solution until either a maximum number of iterations or a maximum time has been reached.

sis in Chapter 1, this section presents a brief review of the most relevant extensions and applications of path/motion planning algorithms, which have permitted conducting autonomous missions under both motion and online computation constraints.

2.7.1 *Planning Feasible Paths*

As explained in Section 1.2.2.2, planning under motion (differential) constraints has to do with considering the limits of the feasible system's maneuvers. These are described by a set of differential equations, generally expressed as $\dot{q} = f(q, u)$ (where q and $\dot{q}$ are the system state and its first derivative, respectively, and u is the control input). A large body of research has been dedicated to the development and improvement of different approaches attempting to solve planning problems that include this kind of constraints.

When Donald et al. formally introduced the problem of kinodynamic motion planning[3], they proposed the use of dynamic programming to find the shortest path in a directed graph using depth-first search (DFS). In this case, the graph represents a discretization of $\mathcal{C}_{free}$; the edges correspond to trajectory segments obtained after applying an acceleration a (bounded according to the system's capabilities) for a period of time τ (determined by the algorithm) [**Donald1993**].

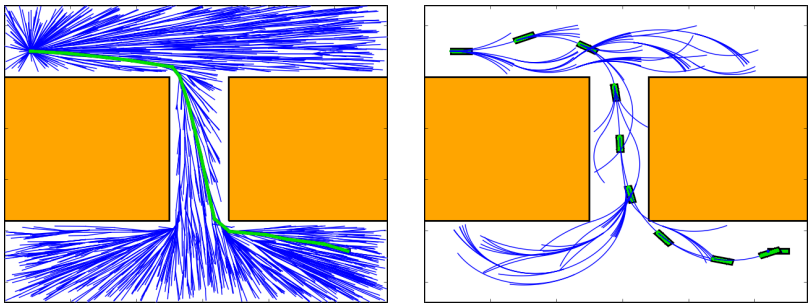
Other variants of grid-based methods, such as A*, have been also used with similar strategies for discretizing $\mathcal{C}_{free}$. In terrestrial vehicles, for example, the most remarkable contributions were a result of the DARPA Grand Challenge [**Thrun2006**][4]. In one of those works, Likhachev and Ferguson used a multi-resolution lattice state space (a $\mathcal{C}_{free}$ discretization),

3 Kinodynamic motion planning alternatively refers to motion planning under motion or differential constraints.

4 The DARPA Grand Challenge is a competition of autonomous vehicles, funded by the Defense Advanced Research Projects Agency (DARPA).

where states represent configurations, and connections between them represent feasible paths (i.e., those that consider kinematic constraints). Then, one A* variant (called AD*) would find paths over the lattice [**Likhachev2009**]. Similarly, Dolgov et al. presented an approach in which $\mathcal{C}_{free}$ is also discretized and paths are found by running another A* variant (Hybrid-State A*). This is guided by two heuristics, one that considers the vehicle's non-holonomic constraints and a second one that computes the Euclidean distance. This approach also connects the states (configurations) by restricting the control inputs according to the feasible (doable) maneuvers of a car-like system, which are expressed by non-holonomic (kinematic) constraints [**Dolgov2008**, **Dolgov2010**].

In what concerns sampling-based methods, EST and RRT algorithms were initially conceived to cope with motion constraints [**Hsu2002**, **LaValle2001**]. In their original versions, the differential equation of motion is used to generate new nodes (states) during the *expansion* of the tree (e.g., in Algorithm 6, line 3)[5]. Figure 16 depicts two sampling-based algorithms: RRT* and RRT in the process of solving a start-to-goal query in a 2D workspace under both geometric and motion constraints.



(a) Geometric Constraints (b) Differential Constraints

Figure 16: Start-to-goal query solution in a 2D workspace, where obstacles appear in orange, free space in white, and the tree of collision-free configurations in blue. The solution (in green) is calculated by (a) an RRT* algorithm under geometric constraints for a point-like system $\left(\mathcal{C} = \mathbb{R}^2\right)$, and (b) an RRT algorithm under differential constraints for a car-like system $\left(\mathcal{C} = \mathbb{R}^2 \times \mathcal{S}\right)$.

There are several successful applications based on this approach. In terrestrial vehicles, one of the most relevant application was presented by Kuwata et al. during The DARPA Urban Challenge. They proposed an approach known as closed-loop RRT (CL-RRT), which not only considers the vehicle's motion model, but also includes the controller's dynamic behavior [**Kuwata2009**]. Another interesting application was done with aerial systems, in which Müller et al. proposed to use an A* for globally finding the collision-free path which, at the same time, guides an RRT algorithm expanded under differential constraints. This latter guarantees finding paths that meet the system's motion constraints [**Muller2011**].

5 While tree and graph nodes are generally referred to as configurations in geometric path planning, in kinodynamic motion planning they are commonly called states. However, this latter term has been indistinctly used by some authors to refer to them in either case.

2.7.2 *Online Path/Motion Planning*

As discussed in Chapter 1, potential and new applications for AUVs involve navigating unknown or undiscovered environments that require endowing the vehicles with the capability of (re)planning online while, at the same time, they explore the environment. There are different extensions that can contribute to achieving this requirement, however this section focuses on two of them. The first is the *anytime* computation, a characteristic incorporated to different kind of algorithms, including both search-based and sampling-based methods. The second refers to *lazy collision checking*, a strategy specifically used with sampling-based methods. These two characteristics have served as a basis for the proposed approach of this thesis.

2.7.2.1 *Anytime Planning Algorithms*

In some situations finding a definite path to solve a start-to-goal query in a finite and deterministic period of time is not possible. It may occur either because of the complexity of the task (i.e., more computation time is required) or because vehicles deal with partially known or dynamic environments. In either case, a common approach is to use an *anytime* algorithm that is capable of providing the best partial solution when the available time is over [**Zilberstein1995**],[**Dean1988**]. The most relevant and well-known planning algorithms have been extended based on this strategy.

Likhachev et al. have studied search-based algorithms, including their extensions for *anytime* computation. They presented the anytime repairing A* (ARA*), a variant that rapidly calculates a suboptimal path to the goal using a loose bound, which is later tightened to progressively improve the path [**Likhachev2003**]. While this approach obtains a fast solution, it also permits improving it if additional time is available. Nonetheless, this approach is useful when full and accurate information about the environment is available, otherwise it may require recalculating the whole path if the environment changes. For those cases, i.e., when coping with dynamic environments, it is better to use an incremental search-based method as D*, which allows locally repairing (replanning) the path when new information of the environment is provided (see Section 2.3.3). However, in its original version, D* lacks the *anytime* property. In order to improve D*'s characteristics, Likhachev et al. also presented the anytime dynamic A* (AD*). This is a variant that not only replans if required, but also improves simultaneously the available solution path [**Likhachev2005**]. A detailed discussion of these *anytime* search-based algorithms is provided in [**Likhachev2008**]. Some applications for vehicles that not only require online/anytime computation, but also navigate under motion constraints are also presented in [**Likhachev2009**].

In the case of sampling-based methods there are also different extensions for *anytime* computation. Belghith et. al, for example, proposed the flexible anytime dynamic PRM (FADPRM), which is an approach that combines both: a standard PRM for constructing the roadmap and an AD* for finding a solution path. This latter one extends the original PRM by permitting not only *anytime* calculation, but also a progressive improvement of

the resulting path. An important characteristic of this approach is the possibility to establish zones with different values of desirability that allow the sampling strategy to be influenced in order to generate less awkward (inefficient and not smooth) paths [**Belghith2006**]. A similar approach presented by van den Berg et al. uses PRM to represent the static portion of C-Space and an AD* to deal with the dynamic elements [**VandenBerg2006**].

On the other hand, and because of its incremental nature, randomized tree-based methods, such as RRT algorithm, are more commonly used for applications in partly known or dynamic environments, where online and *anytime* computation is required. Ferguson et al. proposed an anytime RRT-based algorithm that calculates an initial path and its cost using the standard RRT, thus ensuring that a first solution is found in the shortest time possible. Then, a modified RRT algorithm iteratively generates a series of new solutions that are guaranteed to have a lower cost. The algorithm executes until a stop condition is reached, e.g., when the best available solution path needs to be provided [**Ferguson2006**, **Ferguson2007**]. Other RRT variants as RRT* were also formulated as *anytime* algorithms [**Karaman2011a**].

2.7.2.2 *Lazy Collision Checking*

Even though PRM was originally intended for multi-query applications, Bohlin and Kavraki presented a modified version known as Lazy PRM, which minimizes the execution time by reducing the quantity of collision checking callbacks when solving a specific (single) query [**Bohlin2000**, **Bohlin2001**]. This variant builds a roadmap just as a standard PRM does, but it assumes that all the nodes (configurations) and edges (paths between configurations) are collision-free, leaving the collision detection to the final stage, i.e., when it must find the shortest path between an initial and a final configuration. If a collision is found, the associated nodes and edges are discarded (eliminated) and a new short path is calculated. The authors also suggested an optional *enhancing roadmap* step when collisions are indeed detected. This consists in including more samples around the discarded nodes.

This strategy of delaying the collision detection is known as *lazy collision checking*, and has been used in different applications, especially those with online computation requirements. Bekris and Kavraki, for instance, proposed and validated a tree-based planning framework for terrestrial vehicles, where the states validity is only checked once a path between the start and the goal configuration has been found [**Bekris2007**]. Another example is the one presented by Vahrenkamp et al., where this strategy was used to speed up the motions calculation for humanoid robots (with many degrees of freedom, i.e., high dimensional C-Space) [**Vahrenkamp2007**].

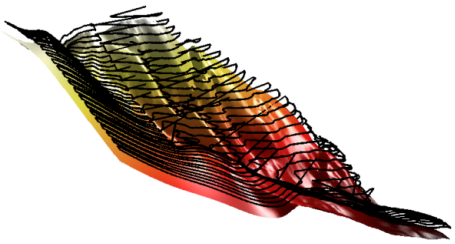
Finally, it is important to note that *lazy collision checking* has served as a base for one of the extensions proposed and used in this thesis, which will be explained in detail in Chapter **??**.

2.8 PATH/MOTION PLANNING FOR AUVS

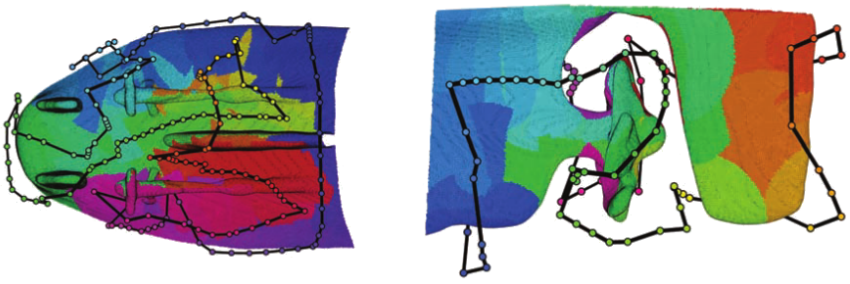
An important aspect to consider when comparing the path/motion planning approaches for AUVs is their application. Based on this, the different

contributions can be classified into two main categories. The first group gathers those applications that require coverage path planning (CPP) techniques, which are commonly applied to guide AUVs over survey tasks. The most common examples within this group include coverage missions used for creating detailed bathymetric maps of the seabed [**Fang2010**, **Galceran2012**, **Galceran2013b**, **Galceran2013d**], detecting potential targets (such as underwater mines [**Stack2003**, **Williams2010**]), and inspecting artificial structures (such as in-water ship hulls [**Englot2010**, **Hover2012**, **Hollinger2013**, **Englot2013**]), as well as natural marine formations [**Galceran2014**, **Galceran2014b**].

A common characteristic in all these approaches is that the planner is provided with preliminary information of the target area or structure (see Fig. 17). This may include its location and shape. Based on this, the CPP algorithm defines a survey path that, in some cases, is reshaped or refined online according to the data obtained during the mission execution. This characteristic implies that most of the computation is done offline, i.e., before conducting the mission. Most recent work presented by Vidal et al. proposes a novel approach to conduct inspection tasks without preliminary information of the target. Results are, however, still limited to 2D motions (at a constant depth) [**Vidal2017**].



(a) Coverage path over of a natural formation. Image credit: Galceran et al. [**Galceran2013b**]



(b) Coverage path of a ship hull. Image credit: Hover et al. [**Hover2012**, **Englot2013**]

Figure 17: Coverage path planning (CPP) algorithms are normally used to plan routes to inspect different kind of structures. In most of these applications the planner has preliminary information. This may include a 2D/3D map of the area or structure to be inspected.

The second group of AUV applications, on the other hand, gathers those that are focused on safely and efficiently guiding the vehicle from one initial position to a specified goal. For doing so, different strategies, such as those explained throughout previous sections, have been applied to under-

water vehicles. In fact, these start-to-goal methods are commonly used as low-level motion planners required for the aforementioned coverage applications. The work presented throughout this thesis seeks to make contributions to this group of start-to-goal applications, therefore it is important to identify the main characteristics between the different approaches used.

2.8.1 *Start-to-goal Path/Motion Planning for AUVs*

One characteristic to consider in a start-to-goal planner for AUVs, and yet not an obvious one, is the capability of conducting 2D and 3D motions. Although AUVs operate in 3D workspaces, in a significant number of applications the vehicles navigate either at a constant altitude or at a constant depth [**Petillot2001**, **Poppinga2011**, **Soulignac2011**, **McMahon2016**], thus simplifying considerably the motion planning problem. There are, however, some contributions that have presented alternatives in modelling and planning 3D (and therefore 2D) AUV motions.

From the approaches that address 3D motions, the available contributions have made use of different approaches such as potential fields [**Warren1990**, **Sequeira1994**], genetic algorithms [**Sugihara1997**, **Alvarez2004**, **Hong-jian2004**], as well as sensor-based [**Ying2000**, **Houts2012**], grid-based [**Carroll1992**, **Sequeira1994**, **Kim2012a**], and sampling-based methods [**Caldwell2010**, **Poppinga2011**, **McMahon2016**]. However, in some of these situations, the vehicle operates in open sea areas, where they do not usually have to deal with obstacles, narrow passages, or high-relief environments. This kind of constraints corresponds to the new and potential AUV applications presented in Chapter 1.

2.8.2 *Online Motion Planning for AUVs through Unexplored Environments*

New AUV applications require a path/motion planner to safely guide the vehicle through unexplored and challenging environments. This implies meeting online computation limitations while, at the same time, considering the vehicle's motion capabilities. Little research on this area, especially for underwater vehicles, has been addressed. In what concerns to generating AUV feasible paths, i.e., those that meet the motion constraints, a first group includes those approaches that use sensor-based methods either to navigate through unknown underwater environments [**Ying2000**], or to follow the terrain shape of a given bathymetric map [**Houts2012**] (see Fig. 18). In both cases, the vehicle is assumed to be equipped with a forward-looking sonar to reactively avoid collisions. Such maneuvers are calculated by a local planner that meets the AUV's kinematics or dynamics. An important characteristic of this kind of approach is the lack of global knowledge of the environment (i.e., a map is not incrementally built), which can cause the vehicle to get trapped in complex scenarios.

A similar reactive approach establishes a set of inequality constraints that describes the obstacles as convex regions contained in the C-Space [**Petillot2001**]. The initial configuration is treated as the starting point of a nonlinear search, where the goal configuration is assumed to be a unique global minimum of the objective function. The start-to-goal query is then solved

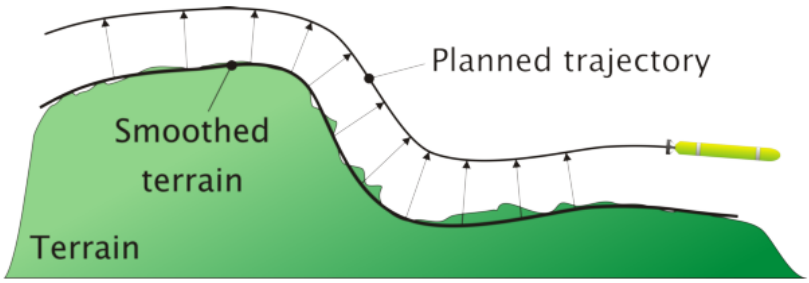


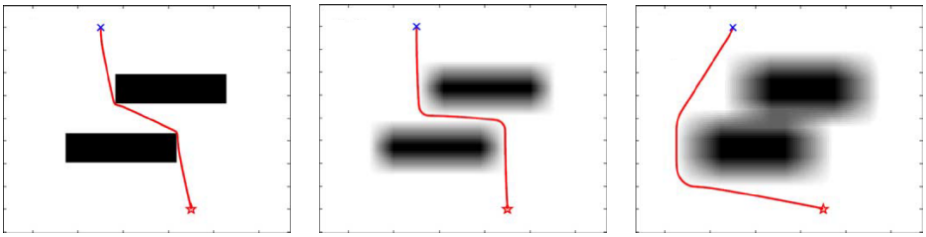
Figure 18: Sensor-based planning for AUVs under motion constraints. The vehicle trajectory can be either preplanned or incrementally calculated to maintain a desired distance from the terrain. The resulting trajectory is fit to satisfy curvature constraints that approximate the AUV motion constraints. Image credit: Houts et al. [**Houts2012**, **Houts2014**].

as an optimization problem, in which a local planner takes into account the vehicles' constraints. This strategy was one of the first online obstacle avoidance approaches for underwater vehicles; it used a real-world dataset of acoustic images obtained by a ROV equipped with a multibeam forward-looking sonar. Its validity was demonstrated by guiding a simulated ROV. However, the capability of simultaneous mapping (detection) and planning online was not established.

A formulation that represents obstacles as convex regions has likewise been used either to represent obstacles detected online, which triggers collision avoidance maneuvers [**Qu2009**], or to approximate the terrain shape that must be followed by the vehicle [**Murthy2010**]. In both cases, the low-level controller attempts to generate feasible trajectories by using the AUV kinematic equations and spline-based interpolation techniques, respectively. However, the main drawback of these approaches is the difficulty in creating a convex representation of complex obstacles.

Another common strategy used in some of the aforementioned methods, is trying to get as close as possible to a path that can be followed by an AUV. Pêtrès et al., for instance, proposed a fast marching (FM)-based approach to find collision-free paths, which are smoothed by a cost function that contains kinematic and curvature constraints [**Petres2007**] (see Fig. 19). Likewise, another example based on genetic algorithms (GAs) finds a valid route to the goal by using basis spline (B-spline) curves, thus seeking to generate more feasible trajectories for AUVs [**Cheng2010**].

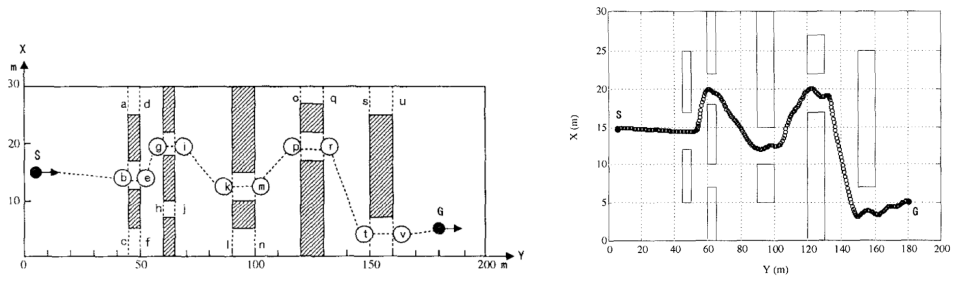
Unlike the previous group of sensor-based approaches, the second group of path planning methods for motion-constrained AUVs uses grid-based approaches. Sequeira and Ribeiro, for example, presented a two-layer framework that is composed of a high-level planner (HLP) and a low-level planner (LLP) [**Sequeira1994**]. The HLP creates a visibility graph using the information of the known obstacles, sea currents, and specified waypoints of the mission. Furthermore, the energy required to move between the graph nodes corresponds to the edge weights. The global and optimal geometric route to the goal is then found by Dijkstra's algorithm. Finally, in order to calculate the vehicle's maneuvers between the different solution segments, the LLP uses an artificial potential field (AFP). This way the total artificial force includes a 3D double integrator that takes into account the AUV motion constraints. There are other similar two-layer approaches, where the



(a) (b) (c)

Figure 19: Start-to-goal query that is solved by a FM-based method. The solution path is smoothed by a cost function to approximate the AUV motion constraints. (a) The initial optimal path. (b) The effect after smoothing the path. (c) The use of the cost function can merge the obstacles, thus discarding possible solutions. Image credit: Pêtrès et al. [**Petres2007**].

global path planning problem is tackled with a grid-based method, and the local motion planning deals with the AUV constraints [**Arinaga1996**] (see Fig. 20). The main disadvantage of line of works is that the grid-based layer generally requires *a priori* information of the environment, e.g., a navigation map.



(a) (b)

Figure 20: (a) Global motion planning solved by a grid-based method, (b) which is adjusted by a local planner that takes into consideration the AUV motion constraints. Image credit: Arinaga et al. [**Arinaga1996**].

A third group of recent contributions includes those that use sampling-based planning algorithms. The most common approach builds a tree of collision-free configurations, which are obtained by integrating the differential equation that describes the AUV dynamic behavior [**Tan2004**, **Caldwell2010**, **He02013**]. This strategy was briefly introduced in Section 2.7.1, but its use with a torpedo-shaped AUV will be explained in more detail in Chapter 3.

Finally, there is another concept for generating feasible paths. It consists in utilizing the Dubins curves [**Dubins1957**], which establishes a set of six maneuvers (RSR, RSL, LSR, LSL, RLR, LRL, where R states for Right, L for Left, and S for straight) to connect two configurations $q_i, q_j \in \mathcal{C} = SE(2)$. These curves correspond to time-optimal trajectories for car-like vehicles. Albeit they have been used for trajectory generation [**Wehbe2014**, **Cao2016**], they are also useful in the context of this thesis.

Another characteristic required for the new AUV applications is the capability of mapping and planning safe paths, simultaneously and online, as the environment is incrementally explored. Apart from the already explained work by Petillot et al., which in fact does not demonstrate on-

line mapping and planning capacity [**Petillot2001**], Maki et al. proposed an online motion planning method that uses landmarks to guide an AUV. Nonetheless, their approach does not permit replanning and, furthermore, results were obtained in a controlled environment (i.e., in a water tank) [**Maki2007**].

2.9 SUMMARY

This chapter has presented an extensive review of the most common path/-motion planning approaches, and those that have been used with underwater vehicles. Table 1 presents a summary of such methods and their characteristics. It is important to bear in mind that the intended applications for AUVs, discussed in Chapter 1, must meet online computation constraints in order to deal with unexplored environments. This implies that time complexity and anytime computation are important requirements. The following chapters will present the extension of some of these approaches, and their successful use in some of the intended applications introduced in Chapter 1.

Method	Completeness	Optimality	Time Complexity	Anytime
		Search-based		
Dijkstra	Yes[a]	Yes	$O(n^2)$[a]	No
A*	Yes	Yes	$O(b^d)$[b]	No[b]
		Potential fields		
Potent. Funct.	No[c]	Locally	$O(1)$	Yes
		Roadmaps		
Visibility Graphs	Yes	Yes[d]	$O(n^2)$[d]	No
		Sampling-based		
PRM + Dijkstra	Prob. Compl.	Asymp. Opt.	$O(n^2)$[e]	No
RRT	Prob. Compl.	No	$O(n \log n)$[e]	Yes
RRT*	Prob. Compl.	Asymp. Opt.	$O(n \log n)$[e]	Yes

Table 1: Path/motion planning methods. [a] Complete for bounded C-Space, where n is the number of nodes in the graph. [b] The number of nodes expanded is exponential in the depth of the solution, where b is the branching factor (the average number of successors per state); ARA* provides a mechanism for anytime computation by reusing previous solutions. [c] requires full knowledge of the C-Space, suffers form local minima. [d] Optimal with respect to the traveled distance; where n is the number of points defining obstacles. [e] Number of sampled configurations.

3

PLANNING CONSTANT-DEPTH PATHS UNDER AUV MOTION CONSTRAINTS

Recent and potential applications for AUVs mentioned in Chapter 1 establish most of the motion planner requirements, such as online computation, motion in 3D workspaces, and navigation along unexplored environments in close-proximity to nearby obstacles. All these constraints can be met with sampling-based planning methods. Particularly for navigating in close-proximity, one desired characteristic is being able to calculate feasible motions that take into account the AUV capabilities. This allows minimizing unexpected vehicle trajectories when attempting to follow the calculated path. In this respect, this chapter firstly explains the equations of motion used to approximate the 2D (at a constant depth) vehicle behavior for motion-planning purposes. Secondly, it explains two different approaches to integrate such equations into the motion planner. Lastly, it presents and discusses the results obtained with both approaches in a simulated environment.

3.1 2D FIRST-ORDER MOTION MODEL

As occurs with any mechanical system, AUV motions can be formulated with kinematic and dynamic models. The former ones describe the geometry of motion by relating the system positions and velocities. Dynamic models, on the other hand, not only include the system kinematics, but also take into account the forces and torques that generate the motions. This latter kind of model is generally employed for designing robust controllers, however, their high computational cost makes them an inappropriate approach for the scenarios proposed in this thesis, which require an online (re)planning behavior. The associated overhead is especially true when using sampling-based planning methods, where all configurations are generated by evolving the system, i.e., performing numerical integration of the equation of motion. Alternatively, a kinematic model provides a less accurate approximation for the vehicle's motion constraints, but it also allows additional computation time that can be dedicated to finding better collision-free paths.

In order to establish which kinematic model must be used, it is necessary to understand the vehicle motion capabilities and the possible test scenarios. In some AUV applications, a first valid approximation is to assume that the vehicle navigates at a constant depth (see Sec. 2.8). Furthermore, if the vehicle is a torpedo-shaped AUV, usually, it will only be able to changing its direction of motion by moving forward/backward. In the case of the Sparus II AUV, the vehicle is equipped with two back thrusters that allow it to spin over. However, it is still not capable of conducting lateral motion. Therefore, it is correct to affirm that the vehicle is subject to a motion constraint along its y-axis (see Fig. 21).

As explained in Chapter 1, motion or differential constraints can be expressed as a set of differential equations in the general form $\dot{q} = f(q, u)$, where q and $\dot{q}$ are the system state and its first derivative, respectively, and u is the control input. Having this in mind, and considering the aforementioned constraints when navigating at a constant depth, a non-holonomic and torpedo-shaped AUV can be represented as a simple car-like vehicle, with a first-order motion model defined as follows:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} v \cos(\psi) \\ v \sin(\psi) \\ w \end{bmatrix}, \tag{1}$$

where $q = [x, y, \psi]^T$ corresponds to the system state that includes its 2D position and orientation with respect to an inertial reference frame, and $\dot{q} = [\dot{x}, \dot{y}, \dot{\psi}]^T$ is the first time derivative that depends on the state itself and the control inputs, i.e., linear/surge speed (v) and turning rate (w). From Eq. (1), it can be concluded that the C-Space of a torpedo-shaped AUV is $\mathcal{C} = SE(2) = \mathbb{R}^2 \times SO(2) = \mathbb{R}^2 \times \mathcal{S}$. Figure 21 depicts the inertial frame, the body-fixed frame, and the different variables (positions and velocities) contained in (1).

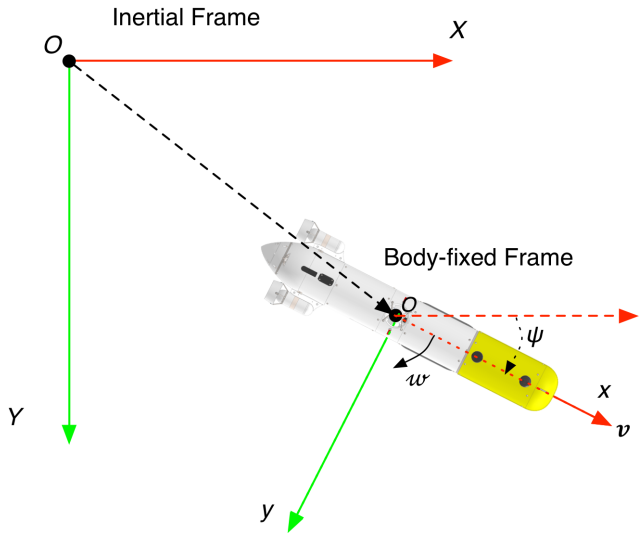


Figure 21: Top view of the Sparus II AUV, including the inertial and body-fixed frames, and the 2D vehicle state and control variables.

Once the constraints have been established through the corresponding differential equations, the next step is to define an appropriate strategy to generate feasible motions that meet such constraints. The following sections present two different approaches, which use sampling-based algorithms to calculate such kind of motions for a torpedo-shaped AUV that navigates at a constant depth.

3.2 MOTION PLANNING BY USING DIFFERENTIAL EQUATIONS

Sampling-based algorithms, especially the RRT [**LaValle2001**] and its variants, have proved to efficiently explore the C-Space while taking all the

motion constraints into account. Furthermore, there are other characteristics such as the incremental search behavior, which allows these methods to continuously (re)plan the solution path as the vehicle moves through unexplored environments (see Chapter **??**). All this together makes an RRT-based algorithm the appropriate approach for the intended AUV applications.

As explained in Section 2.6.2.4, the RRT algorithm builds a single tree that is rooted at an initial state (configuration), which is incrementally expanded towards uniformly randomly *sampled* states[1]. This procedure is the same as the one initially presented in Algorithm 5, being this a particular case where a random state corresponds to $q_{rand} \in SE(2)$, as explained before. In order to *extend* the tree towards q_{rand} while meeting the vehicle's motion constraints, new collision-free motions (states) are obtained by integrating the vehicle's equation of motion, such as Eq. (1). In such a case, instead of using Algorithm 6, the tree expansion can be rewritten as shown in Algorithm 8. This procedure firstly requires finding the state q_{near}, that is the nearest to q_{rand} (line 2). This can be done by using a weighted metric to calculate the distance between configurations. Such a metric combines the translation component, calculated as the Euclidean distance, and the orientation component, calculated as the smallest orientation difference [**Chosset2005**, **LaValle2006**].

Algorithm 8 : extendRRT (when dealing with motion constraints)

Input :
T: tree of collision-free configurations.
q_{rand}: configuration towards which the tree will be extended.
$\mathcal{C}$: C-Space.
Output :
Result after attempting to extend.

```
1 begin
2     q_near ← T.findNearestNeighbor(q_rand)
3     u_near_to_rand ←findInput(T, q_near, q_rand)
4     q_new, collision ←calcNewState(q_near, u_near_to_rand, Δt)
5     if collision = FALSE then
6         V.addNewNode(q_new)
7         E.addNewEdge(q_near, q_new)
```

Once the nearest motion has been found in the tree, the next step is to calculate the control input $u_{near\_to\_rand} = [v, w]$ that has to be applied in order to change the vehicle state from q_{near} towards q_{rand} (line 3). With an RRT algorithm under geometric constraints the tree is iteratively expanded a geometric distance ϵ (see Algorithm 6, line 3), in the case of an RRT algorithm under motion constraints the tree is expanded by applying an input u for a period of time Δt (line 4).

1 There exists another RRT variant called RRT-Connect that grows two trees, one from the start state and another one from the goal state. However, this is not possible when dealing with motion constraints, since merging both trees at a common configuration may become computationally intractable [**Kuffner2000**].

In some AUV applications, a common assumption is that the vehicle navigates at a constant surge speed v, but can vary its turning rate w up to a maximum value w_{max}. This means that calculating the input u is limited to firstly establishing a constant v that will be used along the mission, and then computing a valid value for w that meets the vehicle motion constraints. There are different approaches that permit calculating u. Two alternatives, also presented by LaValle and Kuffner [**LaValle2001**], are either to randomly sample the control space, or to test a number of possible inputs and then select the one that generates the state q_{new} that is closest to q_{rand}. The implementation used in a first stage of this thesis is based on a combination of both approaches.

In such an implementation, instead of testing a large number of different turning rates w, a set of five possible values is established as follows: $w_{control_i} \in \{-w_{max}, -\frac{w_{max}}{2}, 0, \frac{w_{max}}{2}, w_{max}\}$. These values, in turn, define a set of four sub-intervals over the control space: $\{[-w_{max}, -\frac{w_{max}}{2}], [-\frac{w_{max}}{2}, 0], [0, \frac{w_{max}}{2}], [\frac{w_{max}}{2}, w_{max}]\}$. Then, using Eq. (1) with q_{near} as the starting state, each $u_i = [v, w_{control_i}]$ is applied for a period of time Δt to generate five different states $q_{control_i}$. Having them, it is possible to estimate which of the four sub-intervals may contain a control input that leads the tree expansion from q_{near} closer to q_{rand}. The final value of w_{rand} is obtained by sampling a random turning rate over the chosen sub-interval. This procedure avoids discretizing the control space, which would imply the loss of the random exploration of the C-Space, an important property of the original algorithm. Figure 22 presents the main concept of the control input selection.

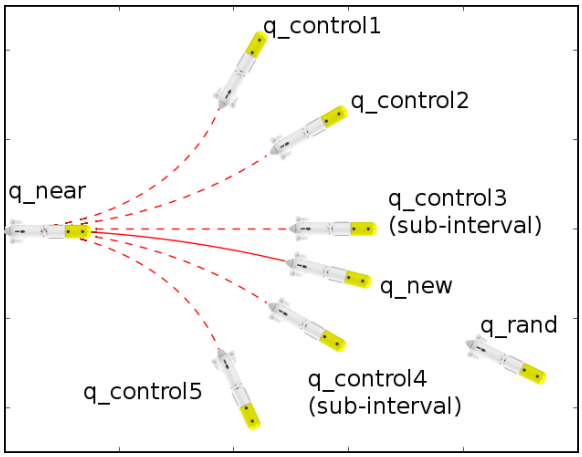


Figure 22: Control input selection. Tree is expanded from q_{near} towards q_{rand}. To do so, an interval of possible turning rates ($q_{control\_i}$) are evaluated. This permits defining a sub-interval from which the control input is sampled randomly, and is applied to generate the new state q_{new}.

The remaining of the *extend* procedure (Algorithm 8) calculates the new state q_{new} (line 4). This is done by integrating Equation (1) while using the final sampled input $u_{near\_to\_rand} = [v, w_{rand}]$. Finally, if q_{new} is proved to be collision-free, it is added and connected to the tree (lines 5-7). Figure 23 depicts an example of an RRT algorithm that solves a start-to-goal query under these motion constraints.

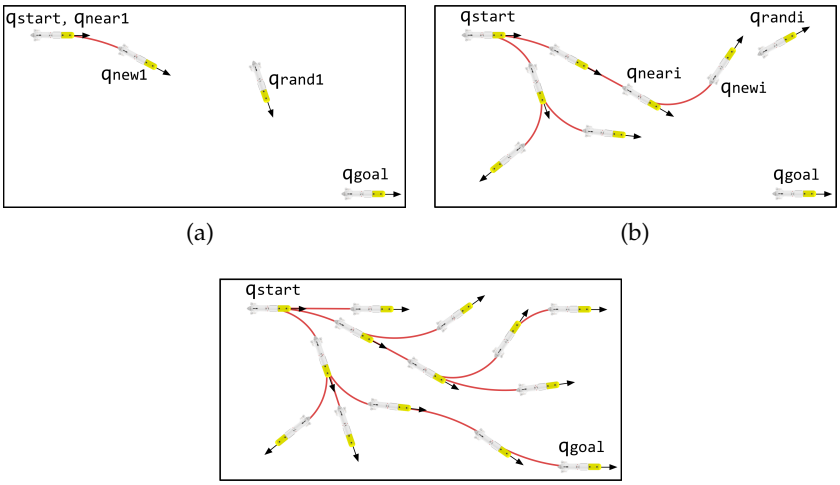


Figure 23: Tree expansion of an RRT algorithm that considers the differential constraints described by Equation (1). (a) Given the start and goal states, a first random sample is used to generate the first expansion of the tree (q_{new}). (b) The expansion i^{th} of the tree. (c) A feasible path has been found from the start to the goal state.

3.3 MOTION PLANNING BY USING DUBINS CURVES

There are several examples where a purely geometric path is first computed, and then it is transformed into a path that is appropriate for the considered vehicle. For example, Yang et al. used an RRT algorithm to find a route of collision-free waypoints, which is interpolated with a cubic Bézier spiral [**Yang2008**]. This seeks to convert the initial route into a smooth and feasible path for an unmanned aerial vehicle (UAV). As another example, Kuwata et al. proposed to expand an RRT by considering not only the vehicle dynamics, but also its controller behavior [**Kuwata2009**].

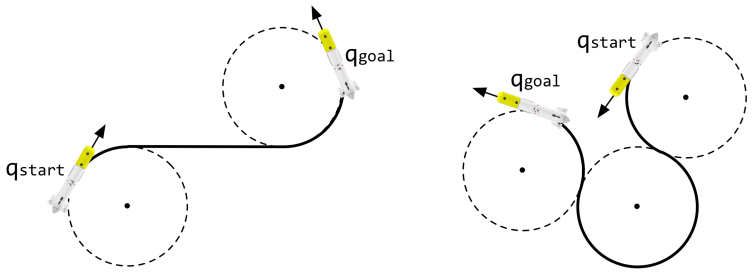
There are other RRT-based approaches that have been proposed to generate feasible paths for aerial and terrestrial vehicles. In one of those approaches, for instance, a geometric RRT algorithm finds a route of collision-free waypoints, which are interpolated with a cubic Bézier spiral. This seeks to convert the initial route into a smooth and feasible path for an UAV [**Yang2008**]. In another approach that is closer to the one presented in the previous section, an RRT algorithm is expanded by considering not only the vehicle dynamics, but also its controller behavior [**Kuwata2009**].

Nonetheless, all those approaches, including the one presented in the previous section, have a major drawback; they do not guarantee optimality for any metric. It is a common characteristic of most sampling-based methods, at least in their original form. This issue could be critical in underwater applications, which may require optimizing different criteria such as visibility (for gathering information), vehicle autonomy, or even the safety associated with a path when navigating in close-proximity to nearby obstacles.

As explained in Section 2.6.3, one option to cope with this situation is to use the RRT* algorithm, which is a variant that incorporates the asymptotic optimality property [**Karaman2011**]. Its main difference with respect

to other RRT-based methods, is a routine that checks if reconnecting new state's nearest nodes improves their associated cost. This implies that the probability of obtaining an optimal path converges to 1 over time. (see Algorithm 7). For doing this, the RRT* algorithm requires a steering function that permits calculating such states (nodes) reconnection. In the case of systems under motion constraints, having such a function implies calculating the required input to dynamically evolve the system from a given state to a desired one. However, defining this function requires solving a two-point boundary value problem, which is in general a very difficult problem.

For the purposes of this thesis, as an alternative to defining a steering function, it is possible to adopt the Dubins vehicle model [**Dubins1957**]. Dubins geometrically demonstrated that, for a system that is only capable of traveling forward and with a constraint on the curvature of the path, the shortest path to connect any two configurations (states) $q_i, q_j \in SE(2)$, when no obstacles are present, can be obtained analytically by the combination of circular arcs and straight lines. Using three possible maneuvers as input, left (L), straight (S) or right (R), Dubins curves define six possible combinations: RSR, RSL, LSR, LSL, RLR, LRL. For a Dubins vehicle, at least one of these characterizes the optimal (shortest) trajectory between two states (see Fig. 24).



(a) RSL (b) LRL

Figure 24: Examples of Dubins curves

These constrained maneuvers are commonly used to describe feasible trajectories for different ground, aerial, and underwater vehicles such as a torpedo-shaped AUV. For this latter case, Equation (1) describes an AUV that navigates at constant depth and with a constant surge speed v, where the maximum turning rate w_{max} establishes the minimum turning radius r_{min} for a Dubins vehicle. For both arcs and straight lines, this means that for any given position (x, y) over the path, its tangent angle corresponds to the AUV heading (ψ), thus completing the state information $q = [x, y, \psi]$.

This alternative formulation for the AUV motion constraints can be used with incremental search methods such as the RRT variants. In such a case, the control input $u_{near\_to\_rand}$, required to evolve the system from q_{near} to q_{rand} (Algorithm 8, line 3), can be replaced with Dubins curves (see Fig. 25). Furthermore, it is important to note that Dubins curves also work as a steering function, thus permitting to generate near-optimal paths with an RRT* algorithm. Figure 26 depicts an example of how this approach is used for reconnecting configurations near to q_{new}, thus im-

proving their associated cost. In this example the cost is assumed to be the path length. Further details about Dubins curves are found in references **[Dubins1957]**,**[Shkel2001]**.

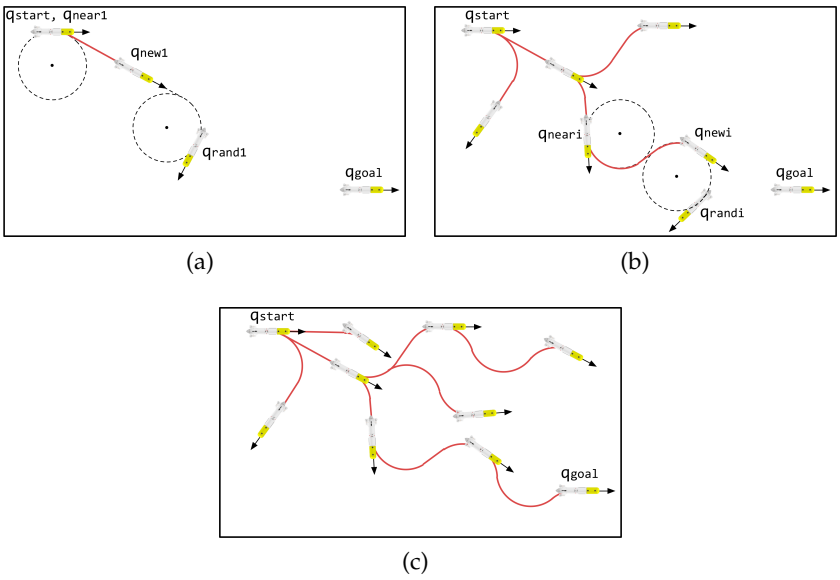


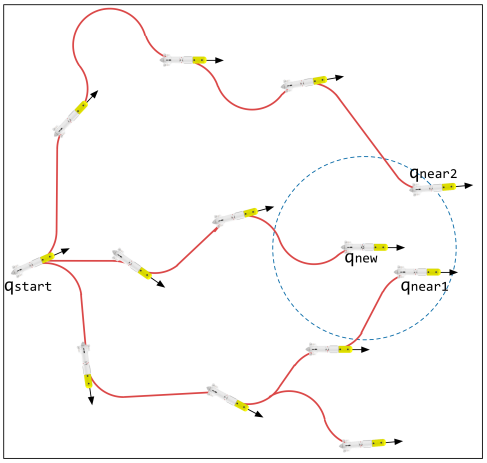
Figure 25: Tree expansion of an RRT algorithm that uses the Dubins curves. (a) Given the start and goal states, a first random sample is used to generate the first expansion of the tree (q_{new}). (b) The expansion i^{th} of the tree. (c) A feasible path has been found from the start to the goal state.

Finally, in order to fully understand the aforementioned approaches, next section presents solutions for different start-to-goal queries that have been obtained using both geometric and motion constraints. Moreover, it also proves the advantages of using Dubins curves instead of using a standard RRT algorithm by integrating Equation (1).

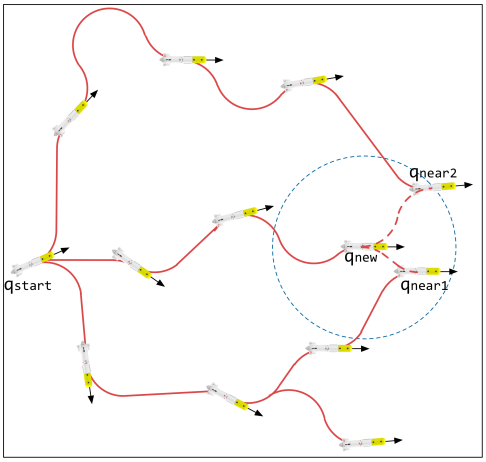
3.4 RESULTS

The main motivation behind the research presented throughout this thesis was the need to plan more accurate motions that permit an AUV to operate in environments that are more complex. As explained in the Introduction, this manuscript is organized in different chapters, each presenting the incremental development of a motion planning framework that endows AUVs with such capabilities. Therefore, the results presented and discussed in this section not only tackled one of the proposed objectives, but largely established the starting point for the rest of this research.

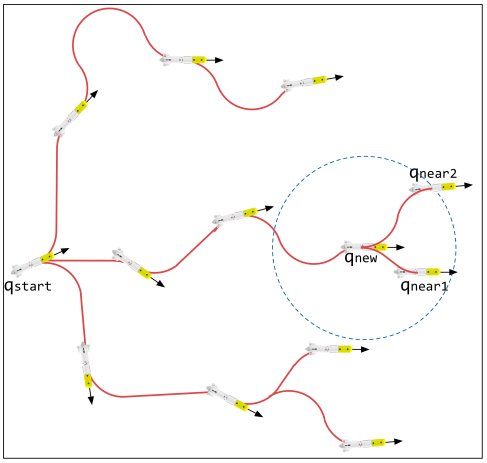
Having said that, the rest of this section seeks to prove the importance of taking motion constraints into consideration when using non-holonomic AUVs. To do so, simulation results present the Sparus II AUV conducting different missions that were planned under both geometric and differential constraints. In all cases, the vehicle navigated at a constant depth in a pre-explored environment, i.e., the obstacles and their locations were known.



(a)



(b)

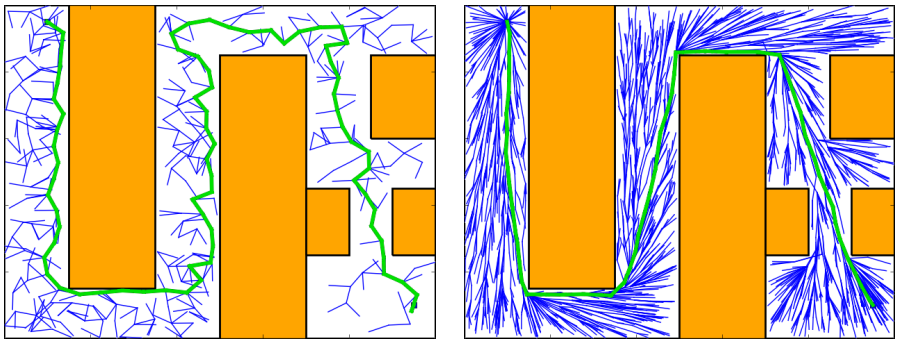


(c)

Figure 26: RRT* algorithm reconnection using Dubins curves as steering function. (a) Once a new configuration q_{new} has been obtained, (b) the RRT* algorithm checks if the nearest configurations to q_{new} can be reconnected. (c) If such a reconnection decreases the cost associated with the nearest configurations, the new paths are created while discarding the previous ones. This allows to progressively improves the solution path.

3.4.1 *Conducting Missions under Geometric Constraints*

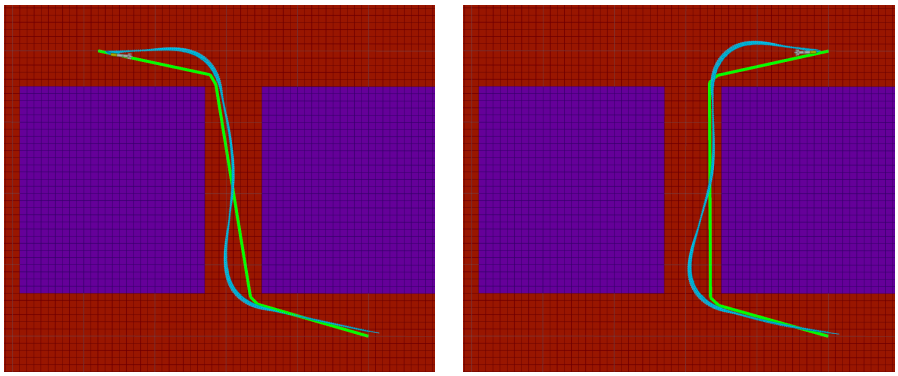
One simple alternative for planning paths for an AUV that operates at constant depth could be to approximate the vehicle to a point-like system, where each configuration $q = [x, y] \in \mathbb{R}^2$. In such a case, an RRT* algorithm could generate (near) optimal paths if enough computing time is provided. Figure 27 depicts the solution path for a start-to-goal query calculated for a point-like system using the RRT and the RRT* algorithms, both under geometric constraints. However, in both cases there are situations in which the path is so close to the obstacles that it could lead to a collision if any real (non-point-like) vehicle would actually try to follow the path. To prevent this, a simple solution is to increase either the size of the obstacle or the size of the vehicle when checking for collisions **[Choset2005]**.



(a) Solution obtained by an RRT (b) Solution obtained by an RRT*

Figure 27: Start-to-goal query solution for a point-like system using both (a) the RRT and (b) the RRT* algorithms. The simulated scenario is composed of a series of polygonal obstacles presented in orange, the safe or collision-free space in red, the tree branches in blue, and the solution path in green.

Nonetheless, even if the shape checked for collision is enlarged, it does not guarantee that a vehicle can accurately follow the path. Figure 28 depicts this situation in a simulated scenario composed of two obstacles that create a narrow passage. In this scenario, two different start-to-goal queries were defined in such a way that required the solution path to be amidst the obstacles. Such paths were calculated by an RRT* algorithm under geometric constraints. In both cases, a simulated Sparus II AUV attempted to follow the path, however new risky situations appeared when conducting turning maneuvers. This occurred because the planner, over $\mathcal{C} = \mathbb{R}^2$, does not consider the vehicle motion limits (constraints) but, instead, it assumes the capability of instantaneously changing the vehicle direction of motion. It is also important to highlight that despite the fact that the executions of the missions were collision-free, the AUV surge speed and turning rate could have been established in a way that would have made the vehicle collide with the obstacles.



(a) Query 1 (b) Query 2

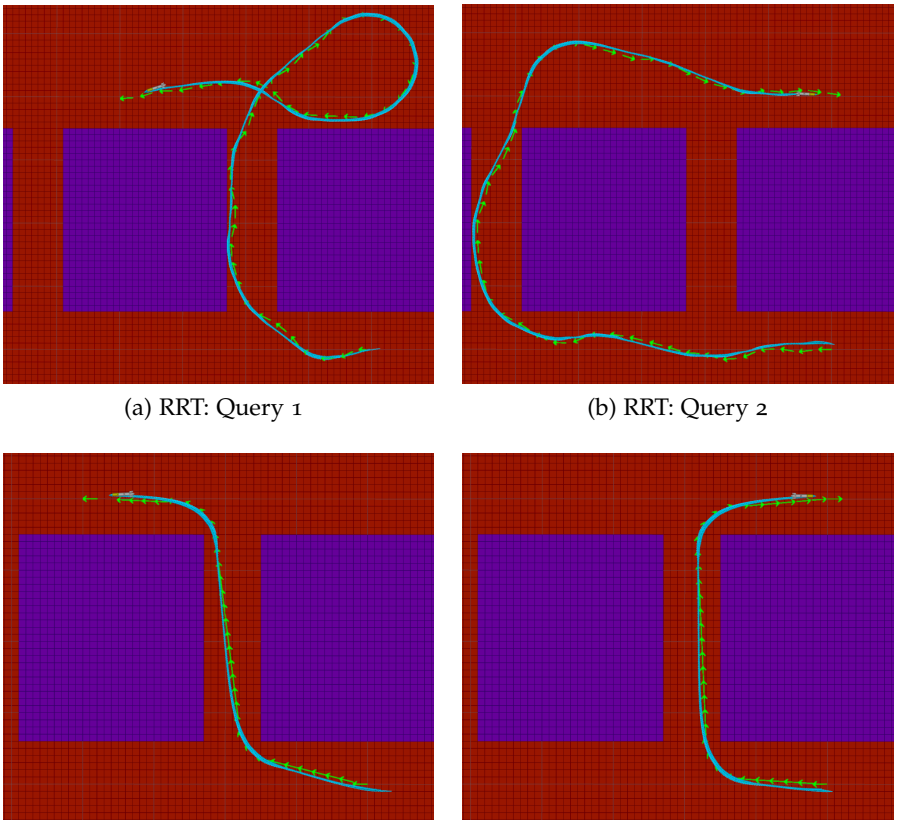
Figure 28: Simulated environment composed of two obstacles that create a narrow passage. Two different start-to-goal query solutions that were calculated by an RRT* algorithm under geometric constraints are shown in green. A simulated Sparus II AUV attempted to follow the paths, however the resulting vehicle trajectory (in light blue) differs from the one calculated, thus leading to risky situations when conducting turning maneuvers.

3.4.2 *Conducting Missions under Motion Constraints*

As explained along previous sections, a more appropriate formulation to limit unexpected vehicle trajectories when following a path is to use a planner that incorporates motion constraints. In such a case, the planner must not only specify the position and orientation, but also use the vehicle equation of motion. For the particular case of a torpedo-shaped AUV, this chapter explained two alternatives to incorporate Eq. (1) into the motion planner. In order to compare the approaches, both were used to solve the same start-to-goal queries presented in Fig. 28. For the first alternative, Figures 29a and 29b depict a simulated Sparus II AUV following solution paths that were calculated by an RRT algorithm that mathematically integrates Eq. (1). In this case, the RRT algorithm does not provide any guarantee for optimality.

The second alternative uses the RRT* algorithm and the Dubins curves, which work as a steering function equivalent to Eq. (1). Figures 29c and 29d depict how, with this approach, the simulated Sparus II AUV follows (near) optimal paths of minimal length. Although this latter approach generates better and more feasible solution paths, there are also some risky states; these were generated because the planner tried to provide the shortest paths, which were close to nearby obstacles. This specific issue about the safety of the resulting path is addressed in detail in Chapter **??**.

A final remark about the content of this chapter is the fact that all cases assume an AUV that navigates at a constant depth. Even though this assumption is valid in several applications, there are others, however, that require to vary the vertical position of the vehicle. For this reason, the next chapter covers the extension of this latter approach (using Dubins curves) for dealing with 3D workspaces.



(a) RRT: Query 1

(b) RRT: Query 2

(c) RRT* with Dubins: Query 1

(d) RRT* with Dubins: Query 2

Figure 29: Simulated environment composed of obstacles that create narrow passages. Sparus II AUV follows the calculated paths. (a), (b) Two different start-to-goal query solutions that were calculated by an RRT algorithm under motion constraints are shown in green. In this case, the RRT algorithm does not provide any guarantee for optimality. (c), (d) The same start-to-goal queries were solved by an RRT* algorithm with Dubins curves as steering function. Solutions are shown in green.

4

PLANNING 3D MOTIONS FOR TORPEDO-SHAPED AND PROPELLER-DRIVEN AUVS

Although there is an important number of AUV applications in which the vehicle navigates at a constant altitude or depth, and therefore a 2D path/-motion planning approach is valid, there are other scenarios in which the vehicle is required to conduct 3D motions (see Chapters 1 and 2). Chapter 3 presented two alternative approaches to plan collision-free paths, which take into account the motion constraints of a torpedo-shaped AUV. While both approaches are limited to plan 2D paths, the one that uses Dubins curves also allows to obtain near-optimal solutions.

In order to define a correct strategy for planning 3D AUV paths, it is necessary to firstly establish the specific vehicle type, and then to understand its main motion characteristics. Underwater gliders, for instance, always require 3D motion planners and controllers, especially if considered their propulsion mechanism and the kind of trajectories they follow (see Chapter 1, Fig. 2). However, given that they operate in open sea areas, these vehicles do not usually have to deal with obstacles, narrow passages, or high-relief environments. Instead, their motion planners are more focused on coping with external perturbations such as ocean currents, which may deviate the glider from the desired trajectory. This makes that, in most cases, motion planning becomes a trajectory planning problem, which must consider the ocean forecast in order to minimize the effect of such perturbations [**Smith2010**, **Thompson2010**].

Propeller-driven AUVs, on the other hand, are equipped with thrusters that increase their maneuverability for 2D and 3D motions. This chapter discusses the 3D first-order model, which approximately describes the kinematic behavior of a torpedo-shaped and propeller-driven AUV. It also analyses the motion constraints involved, and how they can be included into a extended version of the Dubins curves approach presented in the previous chapter.

4.1 3D FIRST-ORDER MOTION MODEL

For 3D motions, six different variables specify the vehicle's position and orientation, which means that the C-Space is $\mathcal{C} = SE(3) = \mathbb{R}^3 \times SO(3)$ (see Fig. 30). Dynamic models, which normally consider all the 6 variables, have also been used in some AUV path-planning applications [**Ying2000**, **Wehbe2014**]. Nonetheless, and as it has been already mentioned, their high computational cost make them an inappropriate alternative for the scenarios proposed in this thesis.

As occurs with 2D AUV motions, there exist different kinematic models that could be used for describing 3D motions. They mainly vary according to the AUV motion mechanism and its corresponding constraints. Two different cases will be explained in the following sections, one of which has been employed for planning collision-free paths for the Sparus II AUV.

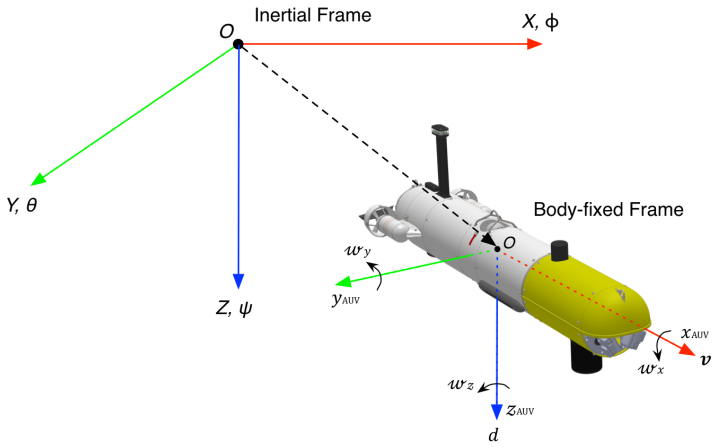


Figure 30: Perspective view of the Sparus II AUV, including the inertial and body-fixed frames, and the 6D vehicle state and control variables.

4.1.1 *Vertical Motion Constraints*

As explained in Section 3.1, a torpedo-shaped and propeller-driven AUV that conducts 2D motions, is generally subject to differential (motion) constraints. These constraints prevent the vehicle from laterally moving by only allowing it to turn while moving forward or backward, or to spin over when it is equipped with two back thrusters, for example as occurs with the Sparus II. When analysing 3D motions for this kind of vehicles, such limitations must be combined with those imposed by the mechanism that permits the vehicle ascending and descending.

Since the AUV volume remains constant during a mission, it is necessary to increase the downward force in order to make the vehicle vary its navigation depth. For doing so, there are different submerging mechanisms that can be classified into two groups. 1) Those vehicles with only one propulsion force in the back, which must be distributed into forward and downward components. This can be done either with inner ballast systems or with external foreplanes. 2) Those vehicles with two propulsion forces, one for forward motion, and an independent one for submerging, which is commonly generated by one or more vertical thrusters.

From the kinematics perspective, both groups generate different kind of motion constraints. The former group's vehicles descend and ascend with a non-zero pitch angle. In this case, the maximum surge speed and the pitch angles establish the maximum descending and ascending speeds. The second group's, on the other hand, are normally designed and trimmed for diving with a near-zero pitch angle. In this case, the maximum descending and ascending speeds are determined by the vertical thruster(s) specifications. In both cases, the vehicles are normally equipped with additional aft planes to generate turning maneuvers, and to minimize the vehicle roll. These two approaches result into different mathematical equations that describe the vehicle kinematics.

4.1.2 *Mathematical Formulation*

A torpedo-shaped AUV with non-zero pitch vertical motion, can be kinematically described with a 6-DOF model as follows **[Wad002010]**:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \cos(\psi)\cos(\theta) & 0 & 0 & 0 \\ \sin(\psi)\cos(\theta) & 0 & 0 & 0 \\ -\sin(\theta) & 0 & 0 & 0 \\ 0 & 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & 0 & \cos(\phi) & -\sin(\phi) \\ 0 & 0 & \sin(\phi)\sec(\theta) & \cos(\phi)\sec(\theta) \end{bmatrix} \begin{bmatrix} v \\ w_x \\ w_y \\ w_z \end{bmatrix}$$
$$= \begin{bmatrix} v\cos(\psi)\cos(\theta) \\ v\sin(\psi)\cos(\theta) \\ -v\sin(\theta) \\ w_x + w_y\sin(\phi)\tan(\theta) + w_z\cos(\phi)\tan(\theta) \\ w_y\cos(\phi) - w_z\sin(\phi) \\ w_y\sin(\phi)\sec(\theta) + w_z\cos(\phi)\sec(\theta) \end{bmatrix}, \tag{2}$$

where $q = [x, y, z, \phi, \theta, \psi]^T$ corresponds to the system state that includes its 3D position and orientation with respect to an inertial reference frame, and $\dot{q} = \left[\dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\theta}, \dot{\psi}\right]^T$ is the first time derivative that depends on the state itself and the control inputs, i.e., linear/surge speed (v) and turning rates around the different vehicle axis (w_x, w_y, w_z). Here the vehicle is subject to two non-holonomic constraints, which are established on the linear velocities along the vehicle y and z directions.This full kinematic model presented in Eq. (2), has been used in trajectory planning applications **[Qu2009]**. However, it can also be simplified if the vehicle is assumed to maintain a near-zero roll angle ($\phi \approx 0$):

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} v\cos(\psi)\cos(\theta) \\ v\sin(\psi)\cos(\theta) \\ -v\sin(\theta) \\ w_y \\ w_z\cos(\phi)\sec(\theta) \end{bmatrix} \tag{3}$$

On the other hand, torpedo-shaped and propeller-driven AUVs with near-zero pitch motion ($\theta \approx 0$), i.e., those with an independent propul-

sion force for vertical motion, must be described with a different kinematic model as follows:

$$\begin{aligned}\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\\ \dot{\phi}\\ \dot{\psi}\end{bmatrix} &= \begin{bmatrix}\cos(\psi) & 0 & 0 & 0 & 0\\ \sin(\psi) & 0 & 0 & 0 & 0\\ 0 & 1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0 & 0\\ 0 & 0 & 0 & \sin(\phi) & \cos(\phi)\end{bmatrix}\begin{bmatrix}v\\ d\\ w_x\\ w_y\\ w_z\end{bmatrix}\\ &= \begin{bmatrix} v\cos(\psi)\\ v\sin(\psi)\\ d\\ w_x\\ w_y\sin(\phi)+w_z\cos(\phi)\end{bmatrix},\end{aligned} \tag{4}$$

where, once again, $q = [x, y, z, \phi, \psi]^T$ corresponds to the system state that includes its position and orientation with respect to an inertial reference frame. Furthermore, $\dot{q} = \left[\dot{x}, \dot{y}, \dot{z}, \dot{\phi}, \dot{\psi}\right]^T$ is the first time derivative that depends on the state itself and the control inputs, which in this case are the linear speeds (surge and heave (v, d)) and the turning rates (w_x, w_z). Here the vehicle is subject to one non-holonomic constraint, which is established on the linear velocity along the vehicle y direction. Equation (4) can also be simplified if the vehicle is assumed to maintain a near-zero roll angle ($\phi \approx 0$):

$$\begin{bmatrix}\dot{x}\\ \dot{y}\\ \dot{z}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix} v\cos(\psi)\\ v\sin(\psi)\\ d\\ w\end{bmatrix}, \tag{5}$$

which can be written in the form $\dot{q} = f(q, u)$, where $q = [x, y, z, \psi]^T$, $\dot{q} = \left[\dot{x}, \dot{y}, \dot{z}, \dot{\psi}\right]^T$, and the control input $u = [v, d, w]^T$ that corresponds to the surge speed (v), the heave speed (d), and the rate of turn (w).

4.2 TREE EXPANSION USING DUBINS CURVES FOR 3D MOTIONS

Different authors have used Eq. (5) to generate 3D motions for torpedo-shaped and propeller-driven AUVs. For instance, Heo et al. presented an approach to build a tree of collision-free and feasible configurations, over which a final path is found with the A* algorithm [**Heo2013**]. The tree expansion is done by directly integrating Eq. (5), similarly as it was explained in Sec. 3.2. However, although A* finds the best solution over the available paths, it requires an estimate of the cost to the goal, and this may be hard to obtain, especially in partially known environments.

An alternative to find better 3D solution paths is to use the RRT* algorithm, however it requires a steering function for improving the configurations connections (see Sec. 2.6.3). Section 3.3 introduced the use of the

Dubins curves as a steering function for 2D paths, which can also be extended to 3D motions. To do so, let us consider the AUV has an independent propulsion force for its vertical motion, and therefore its kinematics can be described with Eq. (5). In this case, it is possible to affirm that the motion in the horizontal plane can be decoupled from the heave (vertical) motion. This allows to directly deal with the former component as stated in Chapter 3, while treating the latter component as an additional constraint over an extended C-Space, $\mathcal{C} = SE(2) \times \mathbb{R}$. This motion separation is not new for underwater vehicles, since it has also been done for dynamic models formulations [**Miskovic2010**].

On this basis, the path between two states for the 3D decoupled kinematics of our vehicle is the one that connects their components in the horizontal plane using Dubins curves, and their vertical components with a linear interpolation. The vertical motion must have a gradient no greater that the maximum ascending/descending AUV speed, thus meeting the vehicle motion capabilities. In order to complete the steering function required by the RRT* algorithm, it is also necessary to establish a metric ρ that allows determining the distance between two states. In this case, one alternative is to combine the lengths of the horizontal and the vertical paths as:

$$\rho\left(q_i, q_j\right) = \rho_h\left(q_i, q_j\right) + \rho_v\left(q_i, q_j\right), \tag{6}$$

where the first term corresponds to sum of the n Dubins sectors lengths, which are required to connect q_i and q_j in the horizontal plane, and the second term corresponds to the distance in the heave (vertical) motion:

$$\rho_v\left(q_i, q_j\right) = \left|q_{iz} - q_{jz}\right| \tag{7}$$

All this together, i.e., the mathematical formulation, the use of the Dubins curves over a 4D space, and the corresponding metrics for calculating distances between different configurations, establish an alternative approach to calculate 3D motions for torpedo-shaped AUVs under motion constraints. Note that in the 3D case, we no longer have an analytical solution for the shortest path between two poses. However, under the assumption that the AUV moves with a constant surge speed, the RRT* algorithm will, with the distance metric defined above, converge on the shortest path that satisfies the descent speed constraint. The next section presents different simulation tests that seeks to prove the validity of this approach.

4.3 RESULTS

This section presents and discusses simulations of the Sparus II AUV conducting 3D start-to-goal missions in different scenarios. Similarly as it was done in the previous chapter, the solution paths have been calculated under both geometric and differential constraints, in order to better understand the importance of considering the AUV motion capabilities when navigating in complex environments. In all cases, the environment was assumed to be pre-explored, i.e., a map of the surroundings was available.

4.3.1 *Conducting 3D Missions under Geometric Constraints*

As it was explained before, Sparus II is a torpedo-shaped AUV equipped with an independent vertical thruster that allows decoupling the horizontal and vertical motions. This means that the vehicle is capable of descending or ascending without needing to move forward or backward. For example, if a start-to-goal query is defined as $q_{start} = [x_0, y_0, z_0, \psi_0]$ and $q_{goal} = [x_0, y_0, z_0 + depth, \psi_0]$, and the planner only takes into account geometric constraints, the solution path should be one that connects both configurations with a straight descent trajectory (see Fig. 31).

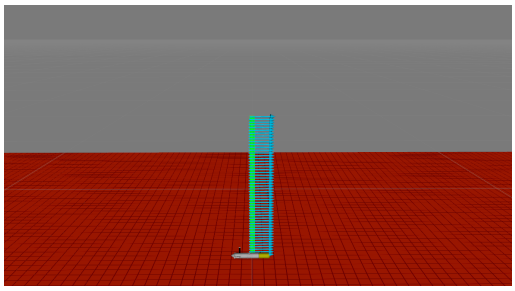
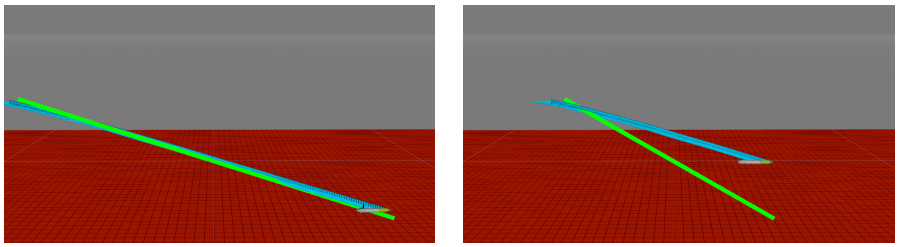


Figure 31: Start-to-goal query that is defined as $q_{start} = [x_0, y_0, z_0, \psi_0]$ and $q_{goal} = [x_0, y_0, z_0 + depth, \psi_0]$. The solution path (in green) is calculated only taking into account geometric constraints, and requires the AUV to follow a straight descent trajectory (in light blue).

Let us define now a start-to-goal query where $q_{start} = [x_0, y_0, z_0, \psi_0]$ and $q_{goal} = [x_0 + distance, y_0, z_0 + depth, \psi_0]$, which means that the vehicle must conduct forward and vertical motions simultaneously. Furthermore, let us assume that the vehicle is required to navigate at a constant surge speed (v), as occurs in several AUV applications. Under such constraints, the time required to travel the horizontal distance can be calculated as $t_h = \frac{distance}{v}$, while the descending speed (d) can be adjusted up to a maximum value (d_{max}), which can be calculated as $d = \frac{depth}{t_h}$. This kind of tasks could also be tackled with geometric constraints, however there may be situations in which the resulting path cannot be followed by the AUV. For example, if t_h allows enough time to descend the desired vertical distance, i.e., $d <= d_{max}$, the vehicle will successfully complete the task, otherwise it may not reach the desired depth (see Fig. 32).

Previous examples did not require the AUV to change its position along the sway direction, and therefore a constant orientation was assumed during the mission. Nonetheless, real missions, especially those intended for this thesis, present more challenging scenarios. In such cases, the vehicle probably needs not only to vary its vertical position, but also to change its orientation in order to conduct turning maneuvers.

Figure 33 depicts a test scenario that is composed of two blocks (12m × 12m × 8m), which are separated by a passage of 4m. Over this simulated environment, a start-to-goal query solution and its execution are presented, where $q_{start} = [x_0, y_0, z_0]$ and $q_{goal} = [x_1, y_1, z_1]$, which means that the orientation was not taken into account. The 3D solution path was calculated using an RRT* algorithm under geometric constraints, and it was followed by the simulated Sparus II. It can be observed that, although the change of depth did not require a descending speed greater than the ve-



(a) Successful query (b) Unsuccessful query

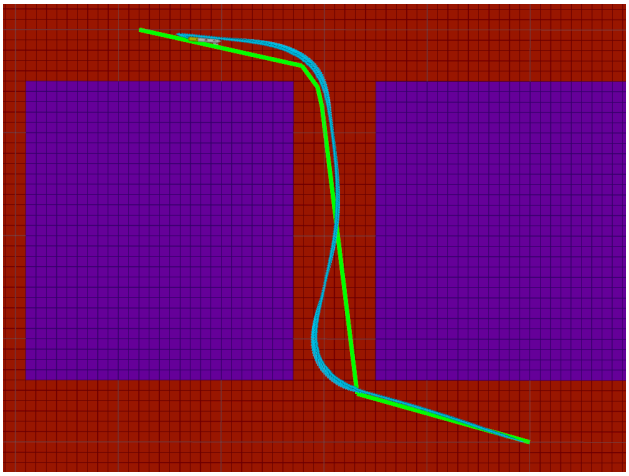
Figure 32: Start-to-goal queries that are defined as $q_{start} = [x_0, y_0, z_0, \psi_0]$ and $q_{goal} = [x_0 + distance, y_0, z_0 + depth, \psi_0]$. The solution path (in green) is calculated only taking into account geometric constraints, and requires the AUV to conduct forward and vertical motion (in light blue). The Sparus II is assumed to have a constant surge speed $v = 0.6m/s$, and a maximum descending speed $d_{max} = 0.2m/s$. (a) Successful query with $distance = 20m$ and $depth = 6m$. (b) Unsuccessful query with $distance = 10m$ and $depth = 6m$.

hicle capabilities (i.e., $d < d_{max}$), the vehicle followed a trajectory that was not initially planned, especially when conducting turning maneuvers. This was mainly due to the fact that the planner did not consider the vehicle's motion constraints. Next sections will demonstrate how the use of the extended 3D Dubins curves approach, explained before, can overcome these situations.

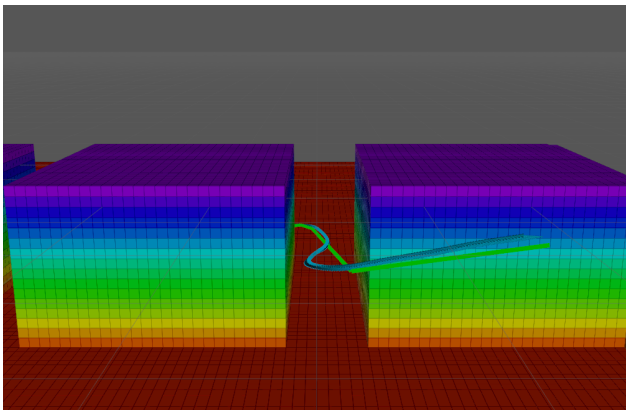
4.3.2 *Conducting 3D Missions under Motion Constraints*

The previous section presented as a first example, a simulated task in which the Sparus II AUV was required to conduct a vertical motion while attempting to maintain its orientation invariant (see Fig. 31). Let us now assume that the vehicle has to solve the same task, i.e., a start-to-goal query defined as $q_{start} = [x_0, y_0, z_0, \psi_0]$ and $q_{goal} = [x_0, y_0, z_0 + depth, \psi_0]$; but this time the vehicle has to navigate with a constant surge speed v and a descending speed up to a maximum value d_{max}. Under such constraints, the motion planner must find a solution that combines turning and descending maneuvers to reach the final position and orientation. This problem can be solved with the extended Dubins curves approach that has been explained throughout previous sections. One possible solution, where the vehicle circumnavigates while descending to reach the desired depth and orientation, can be observed in Fig. 34. Furthermore, this solution is near-optimal under the assumption of a constant surge speed.

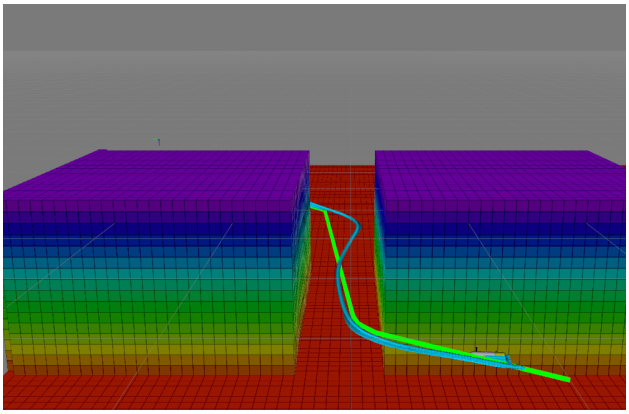
Figure 35 presents the same test scenario composed of two blocks separated by a narrow passage used in the previous section. Similarly as occurred in the simulation presented in Fig. 33, the start-to-goal query requires the vehicle to completely change its 3D position, but this time it also specifies the desired AUV heading. This task, for which the vehicle is also under the surge and descending speeds constraints mentioned before, consists of a $q_{start} = [x_0, y_0, z_0, \psi_0]$ and a $q_{goal} = [x_1, y_1, z_1, \psi_1]$. This time the resulting vehicle trajectory coincides with the planned path, even when conducting turning maneuvers.



(a) Query top view

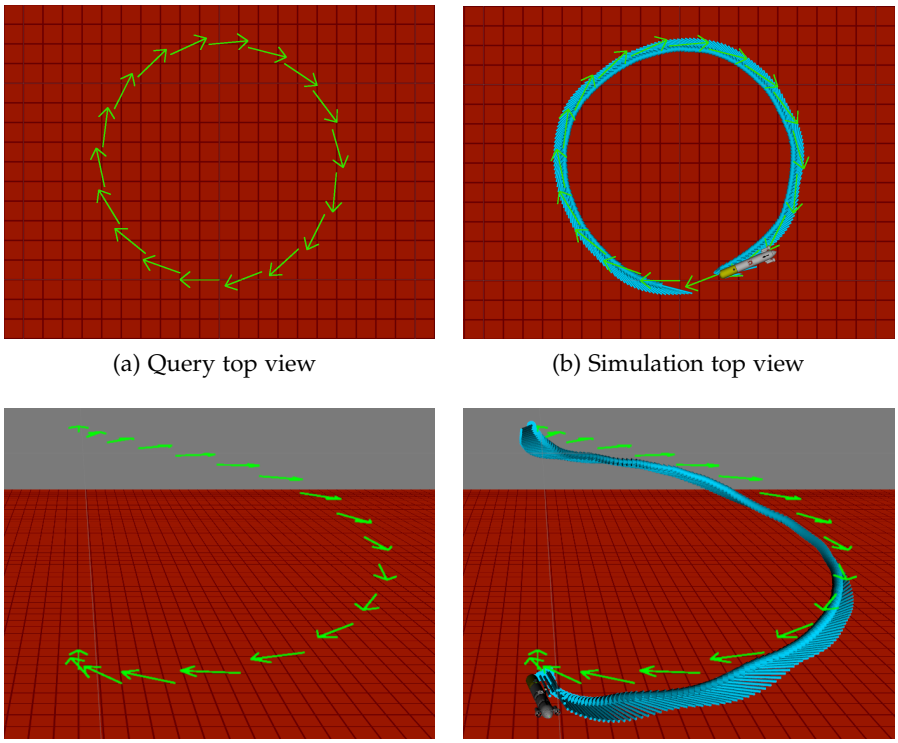


(b) Query front view



(c) Query back view

Figure 33: Simulated environment composed of two blocks (12m $\times$ 12m $\times$ 8m), where a start-to-goal query is defined as $q_{start} = [x_0, y_0, z_0]$ and $q_{goal} = [x_1, y_1, z_1]$. The solution path (in green) is calculated only taking into account geometric constraints. A simulated Sparus II AUV attempted to follow the path, however the resulting trajectory (in light blue) differs from the one calculated, thus leading to risky situations when conducting turning maneuvers.



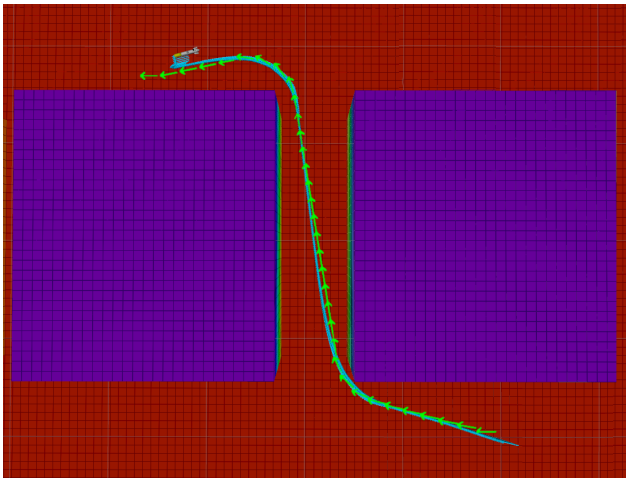
(a) Query top view

(b) Simulation top view

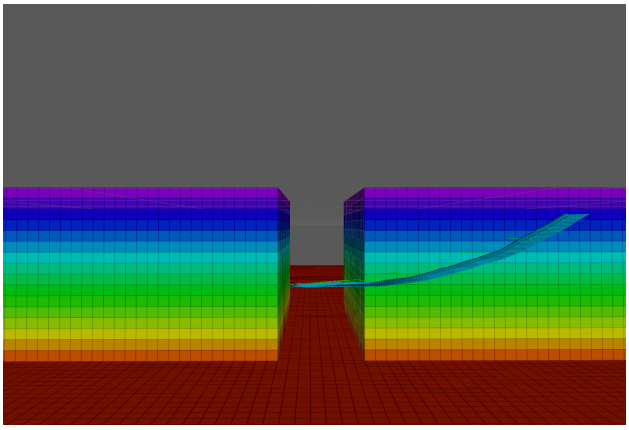
(c) Query perspective view

(d) Simulation perspective view

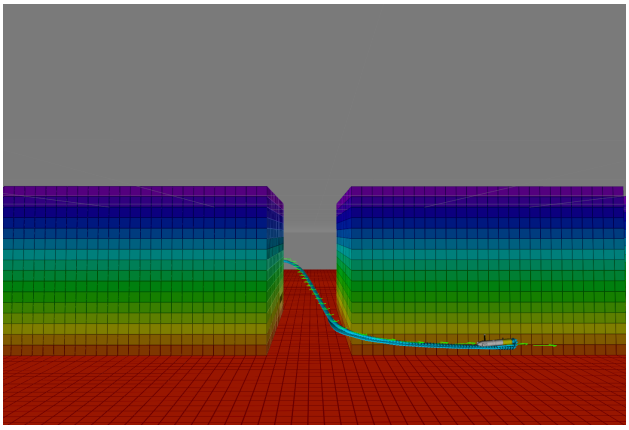
Figure 34: Start-to-goal query that is defined as $q_{start} = [x_0, y_0, z_0, \psi_0]$ and $q_{goal} = [x_0, y_0, z_0 + depth, \psi_0]$. The AUV is assumed to navigate at a constant surge speed v, and a descending speed up to d_{max}. The solution path (in green) is calculated with an RRT* that uses the extended Dubins curves, which incorporate the vehicle 3D motion capabilities. The simulated Sparus II successfully describes a helical trajectory (in light blue) that follows the calculated path.



(a) Query top view



(b) Query front view

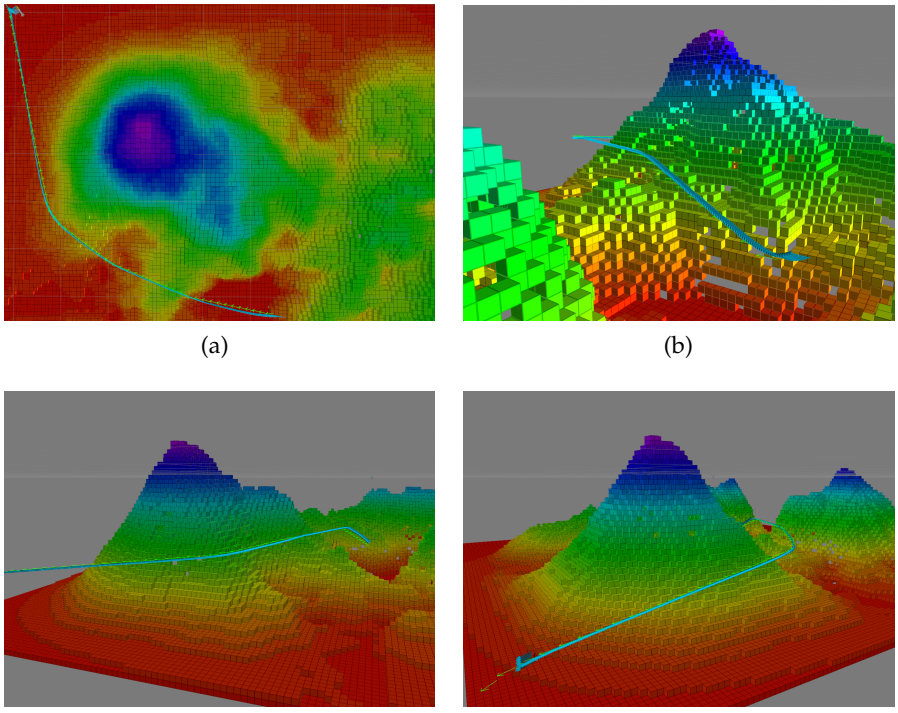


(c) Query back view

Figure 35: Simulated environment composed of two blocks ($12m \times 12m \times 8m$), where a start-to-goal query is defined as $q_{start} = [x_0, y_0, z_0, \psi_0]$ and $q_{goal} = [x_1, y_1, z_1, \psi_1]$. The solution path (in green) is calculated by an RRT* that uses the extended Dubins curves approach. A simulated Sparus II AUV describes a trajectory (in light blue), which does not differ significantly from the one calculated, even when conducting turning maneuvers.

Finally, in order to completely assess the extended Dubins approach for 3D motions, Figure 36 depicts a start-to-goal mission in a high-relief environment. The requested query has been defined in such a way, that any solution path would require the AUV to conduct turning, ascending and descending maneuvers. The resulting path and its corresponding execution by the simulated Sparus II AUV was successful, thus completely proving the presented approach capabilities.

There are, however, some aspects that have to be addressed in order to make this approach really useful for the proposed applications. The first of them is that the intended usage scenarios involve generally unexplored spaces, which means that a map is not available. A second aspect is the safety of the vehicle, which can be compromised when navigating in close proximity to the obstacles. These and other aspects, as well as the different strategies proposed to cope with them, will be discussed in detail in the next chapter.



(a) (b)

(c) (d)

Figure 36: High-relief simulated environment composed of multiple seamounts, where a start-to-goal query is defined as $q_{start} = [x_0, y_0, z_0, \psi_0]$ and $q_{goal} = [x_1, y_1, z_1, \psi_1]$. A simulated Sparus II AUV describes a trajectory (in light blue), which attempts following a solution path that. This path is calculated by an RRT*, which uses the extended Dubins curves approach.

5 RESULTS IN REAL-WORLD SCENARIOS

Previous chapters presented different strategies and extensions that, all together, seek to endow an AUV with the capabilities required to succeed in the applications presented in Chapter 1. The final result is a framework that incrementally builds an Octomap of the surroundings and, simultaneously, plans a collision-free path to move through an initially undiscovered environment. To accomplish this, the framework not only considers the motion constraints involved (thoroughly explained in Chapters 3, 4), but also uses the *risk zones* to establish an optimization function for combining the length and the risk associated with the solution path (see Sec. **??**). Furthermore, while the framework is expanding the tree to find a solution path, it *opportunistically checks* the sampled states and iteratively *reuses the last best known solution*. This allows to efficiently replan (reshape) the path in order to deal with the partial knowledge of the environment (see Secs. **??**, **??**).

This chapter presents an extensive evaluation of the proposed framework. This includes simulation and in-water trials that were carried out in different real-world scenarios. Such experiments were mainly conducted with the Sparus II AUV, but some of them also involved the AsterX AUV[1]. The experiments are separated into four different scenarios: 1) planning constant-depth paths to move through artificial marine structures; 2) planning constant-depth paths to move through natural marine formations; 3) planning variable-depth paths to move through confined marine environments; and 4) the autonomous survey replanning for gap filling and target inspection. The following sections will discuss in detail the results obtained for each of these scenarios.

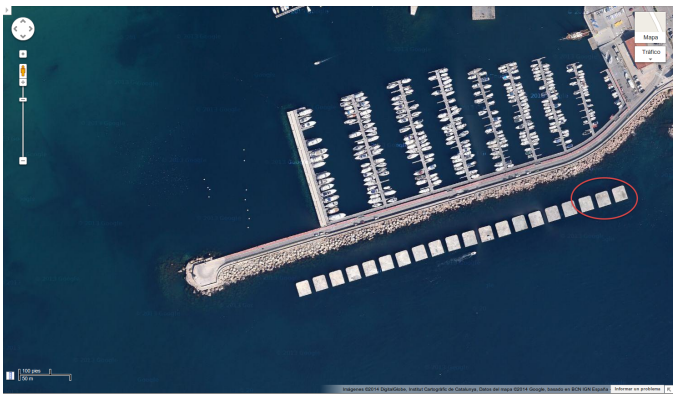
5.1 PLANNING AUV PATHS IN ARTIFICIAL MARINE STRUCTURES

Nowadays, there are a wide range of man-made marine constructions, going from large commercial harbors to offshore platforms. These artificial structures often have a regular and known shape, which could be used to preplan trajectories to travel through them. However, as it was explained in Chapter 1, there are different factors such as low visibility and limited navigation accuracy, which make it difficult to correctly follow such precalculated paths. The online path/motion planning framework proposed in this thesis seeks to contribute overcoming these limitations.

An example of this kind of structures is the breakwater mentioned in previous sections (see Fig. 37c). In this scenario, a test mission could be one that makes an AUV to move amidst the concrete blocks. In order to do so, the origin for the north-east-depth (NED) inertial system is defined by a latitude-longitude coordinate, which can be obtained from Google Maps ©. Likewise, a start-to-goal query can be specified, in such a way

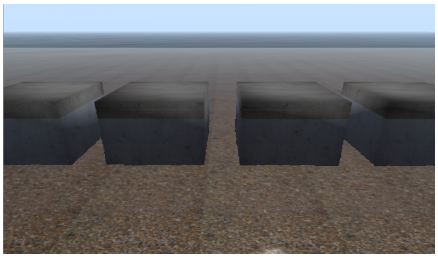
1 Hardware and software details for both vehicles can be found in Appendix A.

that the AUV is required to traverse the blocks from the outer to the inner area, and vice versa. Once the origin, the start configuration and the goal configuration have been selected, it is necessary to define a workspace representation or map, where the collision checking routines can be done. This latter is needed in the case of solving the query with a sampling-based approach, otherwise a discrete or full description of the C-Space may be required instead.



(a) Satellite view. Image credit: Map data ©2017 Google.

(b)



(c)

Figure 37: Test scenario of an artificial marine structure. **??** Harbor of Sant Feliu de Guíxols in Catalonia (Spain), where a breakwater structure composed of concrete blocks is demarcated. (b) Close proximity view of the concrete blocks. (c) Virtual environment for simulations over UWSim.

As mentioned in previous chapters, an Octomap of the area can be used for collision-checking purposes. Assuming the environment as explored, the Octomap could be either one built from real data (see Fig. **??**) or one synthetically created (see Fig. 33). Then, a collision-free, feasible and safe path could be calculated over such a known map. Simulations of this kind, i.e., using an existing map, were presented in Chapters 3 and 4. However, before conducting any real-world autonomous mission, the Sparus II AUV was teleoperated at surface while traversing the concrete blocks. This allowed to verify the map consistency with respect to the GPS data (see Fig. 38).

Figure 38 highlights one of the main limitations of assuming the environment as previously known and mapped. Even if the AUV is teleoperated at surface, which allows to continuously obtain GPS fixes, there is an error associated with the GPS that can range from one to two meters. This magnitude of error is especially critical when navigating in narrow passages, such as the four-meter gaps between the concrete blocks. Furthermore, if

Figure 38: Sparus II AUV was teleoperated at surface in the breakwater structure scenario. Although the Sparus II did not collide during the teleoperated mission, GPS data (in red) shows a vehicle trajectory under collision with and even over the concrete blocks. Satellite view. Image credit: Map data ©2017 Google.

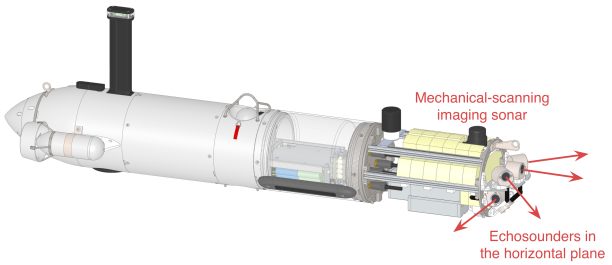
the vehicle submerges, the position estimation error will also accumulate the navigation drift associated with the dead-reckoning (DR) system.

An alternative to conduct the intended mission could be the use of an USBL system, which would contribute to minimize the navigation error. Nonetheless, this option requires a surface vessel that follows the AUV along the mission, something that is not feasible in this scenario. Considering all this, an alternative approach is to use the framework proposed in this thesis.

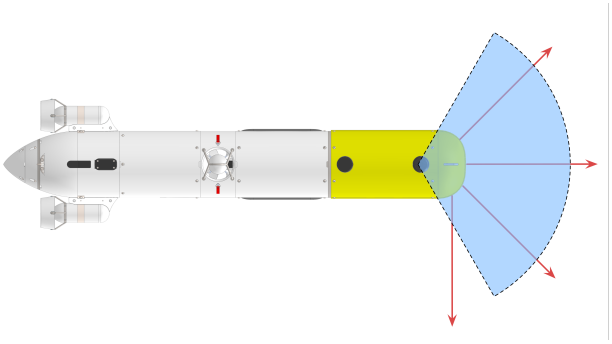
As explained before, the framework requires that the vehicle is equipped with exteroceptive sensors. These sensors allow detecting the surroundings to create a map online. For this scenario, a set of four echosounders and one mechanical-scanning imaging sonar were located within the vehicle payload (front) area, all of them pointing in the horizontal plane (see Fig. 39). Three of the echosounders were separated by 45°, with the central one looking forward and parallel to the vehicle's direction of motion, while the fourth one was perpendicular to the central one (see Fig. 39a). As for the imagining sonar, it was setup to cover a scan sector in the vehicle's direction of motion (see Fig. 39b). Since both sensors cover the horizontal plane, missions with this configuration were executed at a constant depth. For more details about the complete Sparus II's hardware configuration, the interested reader is referred to Appendix A.

During the first simulation and in-water trials conducted over this test scenario, the Sparus II AUV only used the echosounders to perceive the environment. Nevertheless, the four echosounders beams were insufficient to rapidly establish the validity of the path along the direction of motion, which triggered multiple replanning maneuvers. Although the vehicle was capable of finding a solution path, it did required multiple attempts to complete one full successful mission. Furthermore, this limitation generated unexpected and non-desired trajectories. An example of this was already presented in Chapter **??**, Fig. **??**.

In order to avoid the aforementioned situations, the Sparus II used both the echosounders and the mechanical-scanning imaging sonar to incre-



(a)



(b)

Figure 39: Exteroceptive sensors configuration for the Sparus II AUV to conduct autonomous missions in the breakwater structure. (a) The vehicle was equipped with four echosounders and one mechanical-scanning imaging sonar, all of them pointing in the horizontal plane. (b) Top view, echosounders beams direction and imagining sonar scan sector.

mentally build the map. For each beam position within the scan sector, the imaging sonar provided an array of intensities. From these intensity values, those that were over a specified threshold represented the obstacles detected. Once those values had been identified, they were transformed into ranges or distances to the obstacles. However, since the imaging sonar has a vertical beamwidth of 35°, the maximum range was limited to 10m to avoid false-positive detections from the sea bottom. The echosounders, on the other hand, were setup with a maximum range of 20m, since they have a smaller beamwidth of 10° (see Appendix A).

This payload configuration was used in different sea trials, and allowed the vehicle to traverse the breakwater structure with great repeatability. To prove this latter, the Sparus II conducted missions that included multiple and successive start-to-goal queries. Figure 40 depicts one of these missions. For safety reasons, the vehicle was connected to surface with a wireless access point buoy that allowed us to monitor the mission and abort it in case of detecting an unexpected behavior (see Fig. 40a). For this mission, the Sparus II navigated with a constant surge speed $u = 0.5\text{m/s}$, a maximum turning rate $r_{max} = 0.3\text{rad/s}$, and at a constant depth of 2m.

The mission was composed of three different and consecutive start-to-goal queries, and assumed the environment as unexplored. Both the NED reference frame origin and the different goals coordinates when obtained from Google Maps © 2017. For the first query, the vehicle was initially in the inner area of the breakwater structure, and the goal was a coordinate in the outer area. Hence, the only possible solution path required the AUV

to navigate through one of the four-meter gaps (see Fig. 40b). The second query was set back in the inner area but in a different coordinate, thus requiring to move through to one of the gaps once again (see Fig. 40c). Likewise, the third and last query was defined to conclude the mission in the outer area (see Figs. 40d, 40e).

Along the mission, the Sparus II did not surface to obtain GPS fixes. Figure 40f depicts the vehicle trajectory and the Octomap built overlapped with a satellite imagine of the breakwater structure. Although the Octomap is coherent with respect to the real-world structure, there are some differences due to the accumulation of navigation error. Yet this did not prevent the AUV from completing successfully the mission.

To further discuss potential applications, Figure 41 depicts a photo-realistic 3D model of the area traveled by the Sparus II during the mission presented above. In this particular mission, the AUV used a stereo optical camera, which gathered additional data from the surroundings. Such information was used to create a more detailed survey of the area after concluding the mission. Other missions may include sensors to measure different variables of interest, such as biological, chemical, as well as bathymetric information. For more details about these 3D reconstructions, the interested reader is referred to [**Hernandez2016**, **Hernandez2016a**].

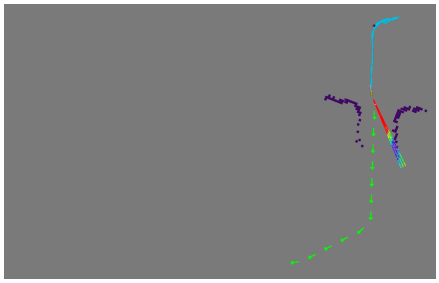
5.2 PLANNING AUV PATHS IN NATURAL MARINE FORMATIONS

Although the mission in the breakwater structure represents a valid application example, it is also true that the obstacles in it, i.e., the concrete blocks, have regular shapes. Their vertical walls, for instance, were correctly detected by a mechanical-scanning imaging sonar, despite its great beamwidth. Natural environments, on the other hand, present less regular and more challenging scenarios. Therefore and in order to prove the proposed framework under different conditions, the Sparus II also conducted autonomous missions over a real-world natural environment. The testing area is also located in Sant Feliu de Guíxols (Spain), and contains rocky formations that create an underwater canyon (see Fig. 42).

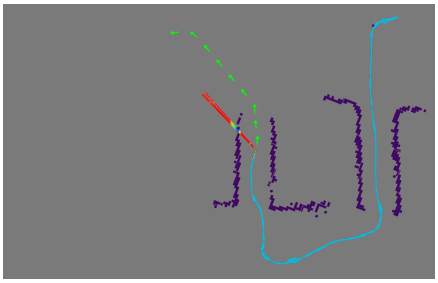
For this scenario, the Sparus II AUV used a mechanic-scanning profiling sonar, which has a smaller beamwidth of $1-2^{o}$ (see Appendix A). This sensor, which covered the horizontal plane, permitted not only to perceive the obstacles shape with more accuracy, but also to set a higher maximum range without being affected by false-positive detections from the bottom. The AUV was also equipped with a set of three GoPro™ Hero 4 Black edition cameras. They were used to gather the optical images required to create a 3D reconstruction of the surroundings. The cameras were positioned systematically to ensure the highest possible coverage, where two of them were placed in a downward configuration at an angle of 20^{o}, while the third camera was positioned forward looking at 40^{o} (see Fig. 43).

In order to inspect the underwater canyon and the surroundings, two different start-to-goal queries were established by extracting GPS coordinates from Google Maps ©. The first query required the Sparus II AUV to traverse the canyon towards the shore. The second query goal was chosen on the outside of the rocky formation in such a way that the vehicle had to

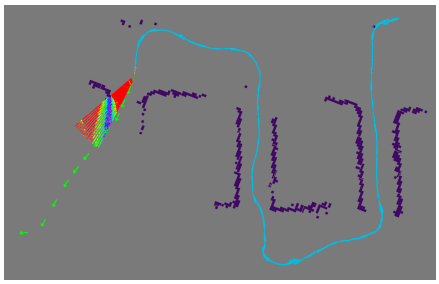
(a)



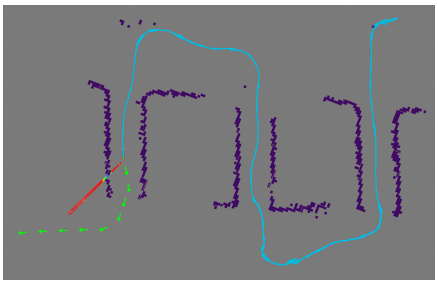
(b)



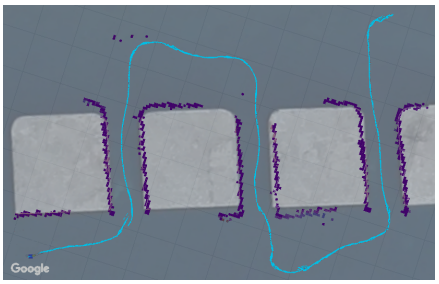
(c)



(d)



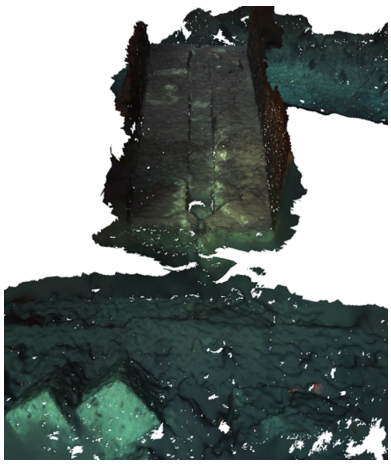
(e)



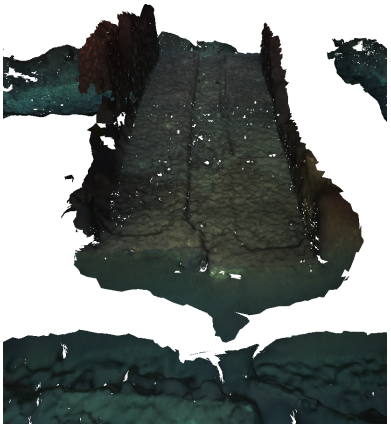
(f)

Figure 40: The Sparus II AUV used the path/motion planning framework to navigate through a breakwater structure without a preliminary map of the surroundings. (a) The vehicle is submerged and is moving autonomously amidst two concrete blocks. (b), (c), (d), (e) The mission required traversing the breakwater structure multiple times by solving successive start-to-goal queries. (f) The vehicle trajectory is drawn in light blue, while the map built with the imaging sonar data is presented in purple. This information is overlapped with a satellite image (Google Maps © 2017).

(a)



(b)



(c)

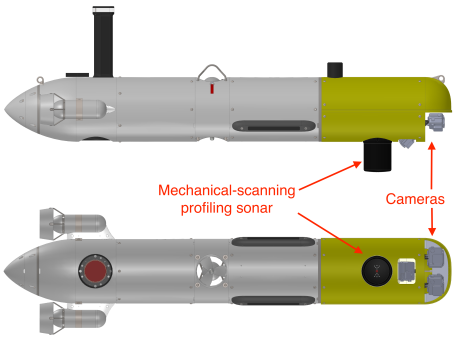
Figure 41: (a) Top-down view of the textured 3D model built with the images gathered along the mission shown in Fig. 40. (b),(c) 3D views of the corridors or gaps between the concrete blocks.

(a) Satellite view. Image credit: Map data ©2017 Google.

(b)

(c)

Figure 42: The test scenario consists of rocky formations. (a) They create an underwater canyon that can be observed in the satellite image. (b) From the inner area (i.e., from the coastline towards the open sea) the canyon can be clearly observed. (c) From the outer area the canyon can be observed in the left.



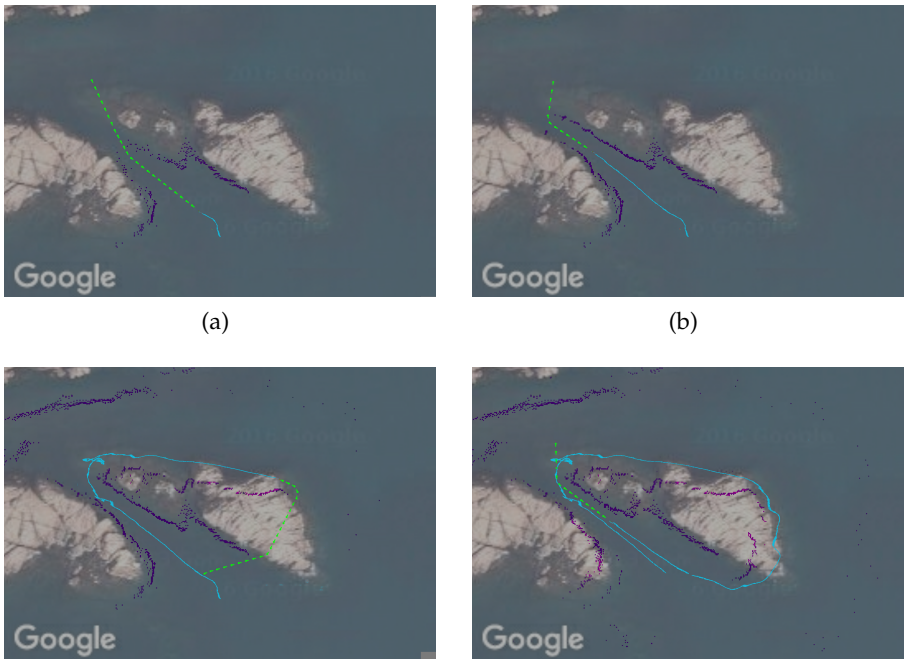
(a)

(b)

Figure 43: Exteroceptive sensors configuration for the Sparus II AUV to conduct autonomous missions in the underwater canyon. (a) The vehicle was equipped with three GoPro™ Hero 4 Black edition cameras and one mechanical-scanning profiling sonar. (b) Real-world vehicle's payload.

circumnavigate the outer rock. Furthermore, after completing the second query, the first query was executed again until the vehicle overlapped its initial trajectory in the canyon. This allowed us to verify the navigation consistency and to close the imaging acquisition loop, thus improving the reconstruction results. As the profiling sonar covered the horizontal plane, the mission had to be conducted at a constant depth.

Figure 44 depicts one of the inspection missions conducted at 3m of depth in the underwater canyon. The Sparus II not only created a map of a complex and unexplored environment, but also planned a collision-free path simultaneously and incrementally. The map and the vehicle trajectory are shown overlapped with a satellite image. In the initial part of the mission, i.e., when the vehicle traverses the canyon for the first time, the map coincides with the satellite image (see Fig. 44b); however, disparities can be clearly observed after some time (see Figs. 44c, d). Such differences are due to the accumulation of error in the navigation system that depends on the doppler velocity log (DVL), which may provide incorrect data when navigating over rocks, as occurred in this test scenario. Despite this situation, the vehicle succeeded in conducting the mission because both the map and the path are created online, which permits correcting or adjusting them even when moving in previously visited areas (see Fig. 44d when accessing the canyon for a second time).

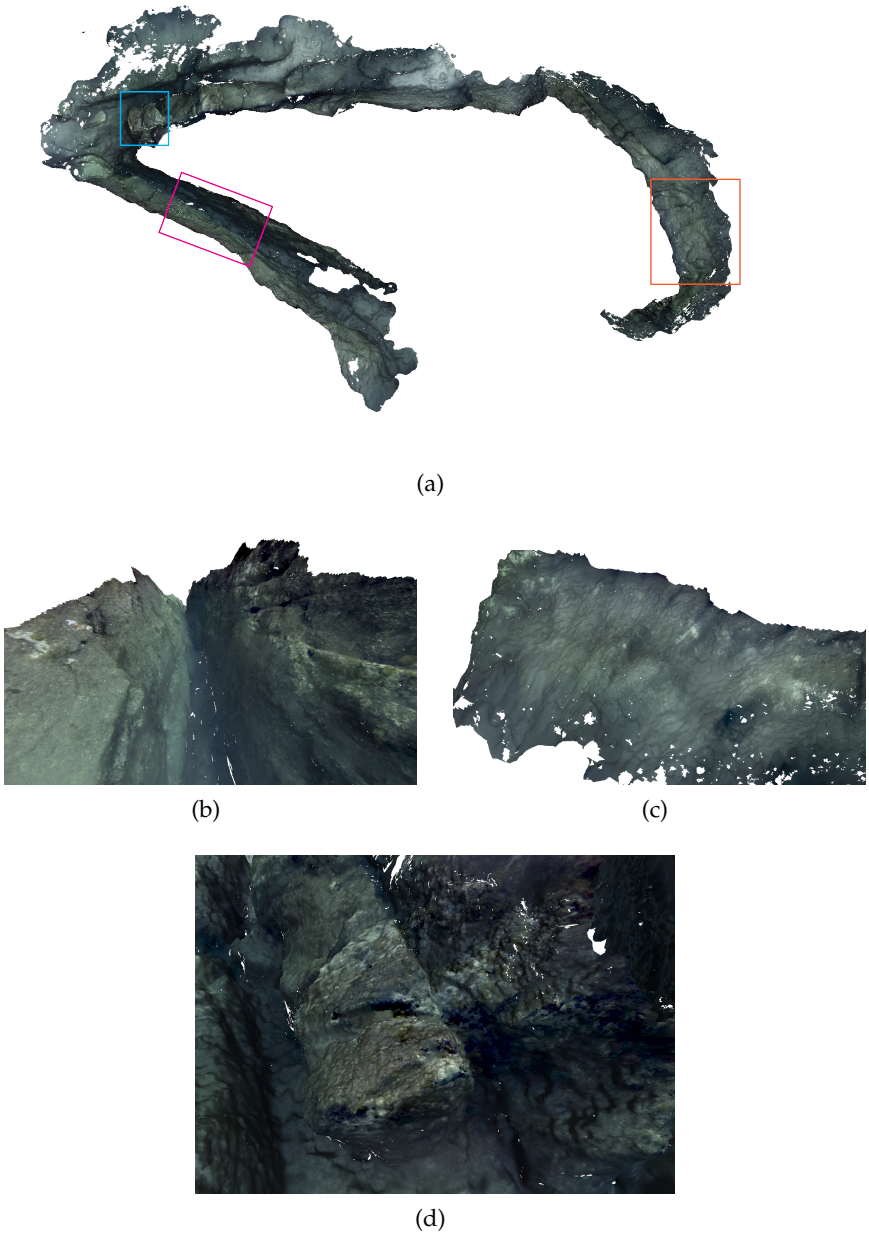


(a) (b)

(c) (d)

Figure 44: The Sparus II AUV used the path/motion planning framework to navigate through an underwater canyon without a preliminary map of the surroundings. (a), (b) The first start-to-goal query required the Sparus II traversing an underwater canyon. (c) For the second query, the vehicle circumnavigated the outer rock. (d) The AUV partially repeated the first start-to-goal query in order to close the loop and to obtain overlapped images. Vehicle trajectory and Octomap overlap a satellite image (Map data ©2017 Google).

In this kind of explorations, the data that is gathered along the mission can be used to create a more detailed survey of the area. In this particular mission, for instance, the images were extracted and were used to build a photo-realistic 3D model, which is depicted in Figure 45a. This 3D reconstruction allows us to better understand complex environments by generating arbitrary user-defined views (see Figs 45b–d). For more details about these 3D reconstructions, the interested reader is referred to [**Hernandez2016**, **Hernandez2016a**].



(a)

(b) (c)

(d)

Figure 45: (a) Top-down view of the textured 3D model with marked areas additionally depicted in magenta, orange and blue, which correspond to generated views of: (b) the underwater canyon; (c) the external side of the underwater rocky formation; (d) underwater rocks.

Finally, in order to understand the complexity associated with this mission, Figure 46 presents a visual comparison between the vehicle's trajectory, estimated by its DR system, and the cameras' trajectory, estimated

during the reconstruction (green and red, respectively). While both trajectories have a similar shape, it can be clearly observed how the one derived from the cameras is more realistic according to the rock observed in the surface (the rocky formation does not create a vertical wall, which means that the vehicle may have moved further from the visible part of the rock when navigating at 3m deep), while the one estimated by the AUV's dead-reckoning system seems to be colliding with the rock. This latter situation is mainly due to the accumulation of errors in the navigation system, as it was already mentioned before.

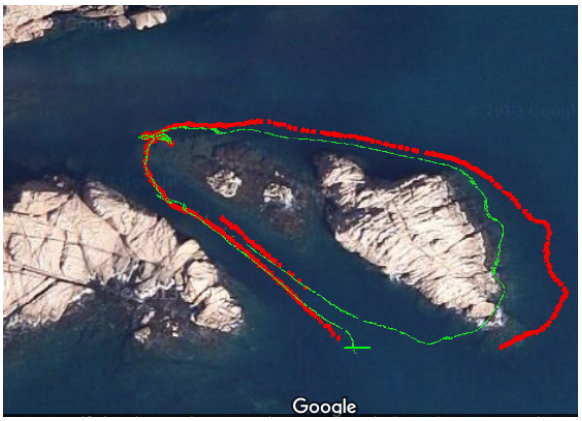


Figure 46: Vehicle trajectory (green) calculated by its dead-reckoning system and the cameras trajectory (red) estimated by the image-based reconstruction. Both trajectories are shown overlapped with a satellite image of the test scenario (Map data ©2017 Google).

5.3 PLANNING AUV PATHS IN CONFINED NATURAL ENVIRONMENTS

The experiments that have been presented until now involved the Sparus II navigating at a constant depth. However, there are other scenarios in which the AUV must modify its vertical position in order to complete the intended mission. In such cases, one of the main limitations from the mapping and path-planning perspective is the capability to build online a 3D map of the surroundings. As explained in Sec. **??**, Octomaps offer an efficient alternative to represent volumetric environment information. Such data can be gathered from a wide range of sensors such as echosounders, profiling/imaging sonars, and multibeam sonars.

One example of a mission that require a 3D path/motion planner is to move through confined natural environments (e.g., underwater caves and tunnels). This kind of scenarios present all sort of limitations, including the impossibility of having either a support surface ship to correct the AUV navigation through an USBL system, or even a wireless access point buoy for monitoring the correct mission execution. Other technical aspects that make more difficult to conduct autonomous missions in these environments, include the navigation error associated with incorrect data provided by the DVL when traveling over rocky surfaces.

In dealing with this latter aspect, Mallios et al. presented the exploration of an underwater caves complex (see Fig. 47), in which a diver guided the Sparus AUV to gather environment acoustic information [**Mallios2015**]. To

do so, the vehicle was equipped with two mechanically scanning imaging sonars to cover the horizontal and vertical planes. Such data was used to prove a scan-matching algorithm over a simultaneous localization and mapping (SLAM) framework, which allows reducing and bounding the AUV navigation error. This survey provides an extensive dataset that includes not only the sonars' raw data, but also a detailed meshed map (see Fig. 48) [**Mallios2015**, **Mallios2017**]. This latter one was obtained by using the splats distance normalized cut (SDNC) algorithm [**Campos2013**]. For more details about this survey, the interested reader is referred to the cited references.

Figure 47: Satellite image of the dataset area. The underwater caves complex "Coves de Cala Viuda" is located in the L'Escala, Spain (Lat: 42.10388, Lon: 3.18255). It consists of three single-branch caves and several tunnels. Their approximate positions are marked with dotted white lines. Image credit: Mallios et al. 2015 [**Mallios2015**, **Mallios2017**]

All this together is nowadays considered a significant step towards the exploration of these kinds of environments. Nonetheless, there are still other aspects that must be tackled in order to conduct this kind of missions fully autonomous. This thesis contributes to coping with some of those aspects. The proposed framework, for instance, seeks to endow AUVs with additional capabilities that will allow to replace the diver's guidance in a near future. To prove these capabilities, this section presents a simulated mission over the meshed map of the cave complex. The mission consisted in solving six consecutive start-to-goal queries, where the Sparus II was not provided with any preliminary environment information, thus requiring to incrementally map and (re)plan the path to the goals. Figure 48 depicts the meshed map and the approximate locations of six different query goals.

In order to conduct this test mission, the meshed map was added into UWSim as a 3D simulation environment (see Fig. 49). To perceive the surroundings, the AUV's payload was assumed to be equipped with an additional link capable of rotating 120° around the vehicle's z axis. Over this link, a forward-looking multibeam sonar was mounted. The sonar had 240 beams distributed over a total aperture of 120° around the vehicle's x axis. With this payload configuration, the simulated Sparus II was capable of gathering 3D range data of the environment along its direction of motion.

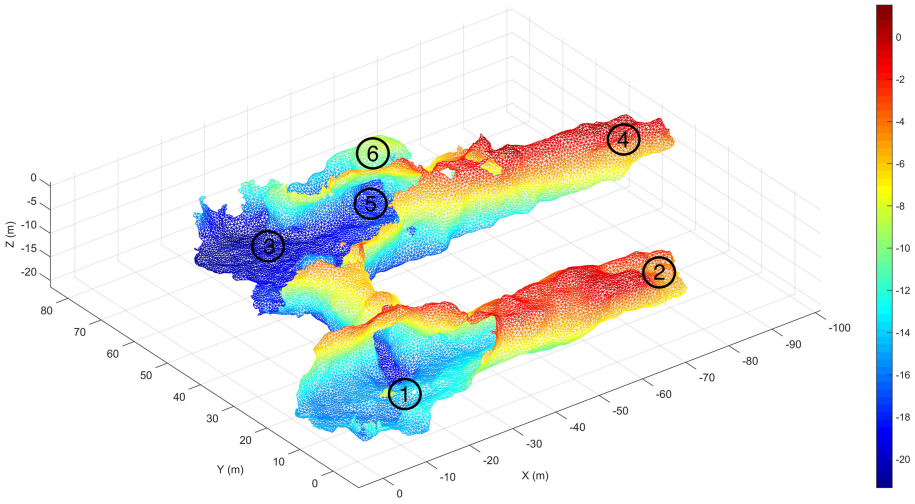


Figure 48: The underwater caves complex "Coves de Cala Viuda". Meshed map using the SNDC algorithm **[Campos2013]**. Image credit: Mallios et al. 2015 **[Mallios2015]**. Over the map, the approximate locations of six different goals have been marked in black.

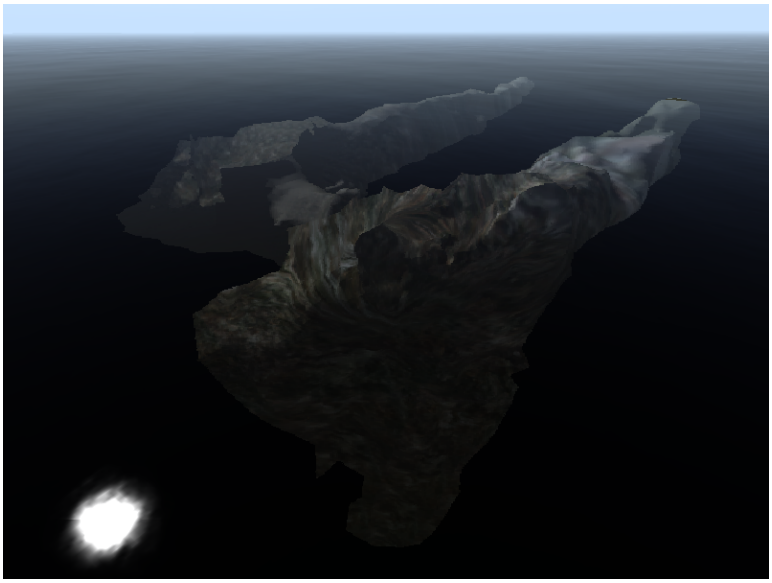
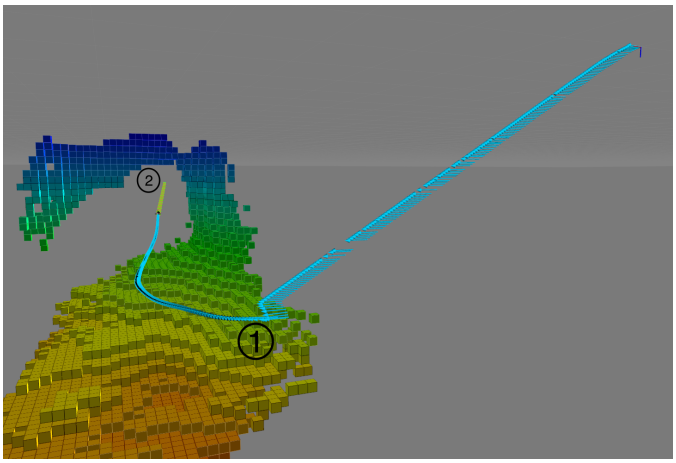
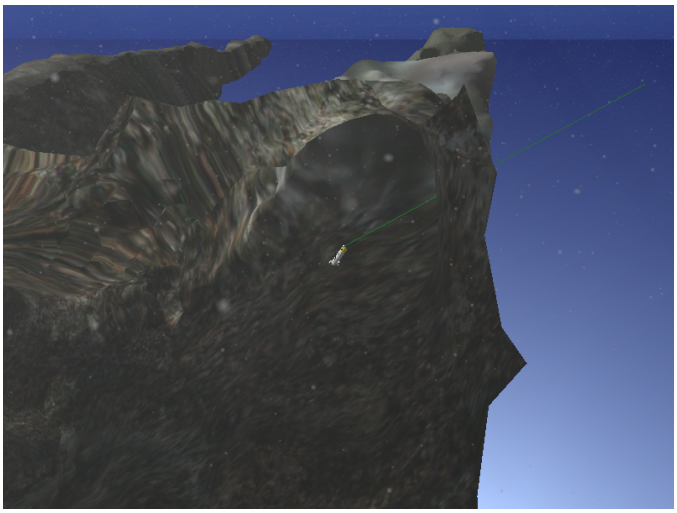


Figure 49: The underwater caves complex "Coves de Cala Viuda" added into UWSim as a 3D simulation environment.

At the beginning of the mission, the simulated Sparus II was located at the origin of the NED reference system and oriented towards the north, i.e., $q_{start} = [0.0, 0.0, 0.0, 0.0]$. The first start-to-goal query was defined to guide the AUV closer to the caves complex location. This required setting $q_{goal_1} = [20.0, 0.0, 12.0, 0.0]$. From this latter position, the second query sought to explore the first and biggest single-branch cave. This meant navigating through and until the end of the cave, which required setting $q_{goal_2} = [18.0, 55.0, 8.0, 1.57]$. For these queries, the planner limited the vehicle to navigate at a constant surge speed $v = 0.5m/s$, a maximum ascending speed $d_{ascend} = 0.2m/s$, and a maximum descending speed $d_{descend} = 0.18m/s$. Figure 50 depicts part of the execution of these two queries, including the AUV trajectory, the calculated path, and the goals.



(a) Vehicle trajectory, calculated path, goals, and Octomap

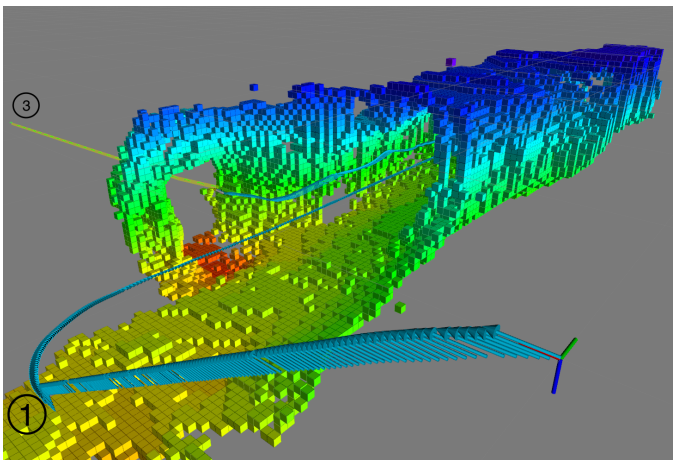


(b) Sparus II approaching the entrance of the first cave

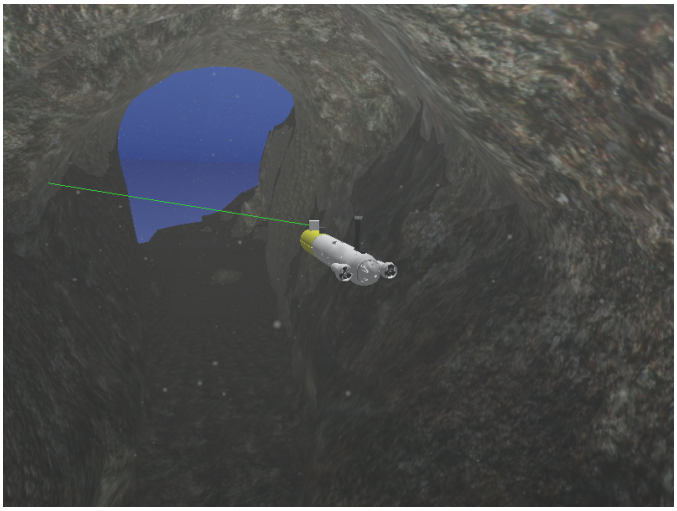
Figure 50: Simulated mission in the caves complex. The first start-to-goal query guided the AUV closer to the complex location. The second query required the vehicle to navigate through the first single-branch cave. In (a) The vehicle trajectory appears in light blue, and the remaining of the calculated path in yellow.

From the second query's goal configuration, a third query was defined to take the Sparus II closer to the second single-branch cave's entrance, i.e., $q_{goal_3} = [80.0, 25.0, 16.0, 0.23]$. To accomplish this part of the mission,

the AUV not only had to find a way out of the first cave, but also had to traverse a tunnel that connects the entrances of both single-branch caves. For this query, the planner limited the vehicle to navigate at a constant surge speed $v = 0.3\text{m/s}$, while keeping the previous maximum ascending/descending speeds ($d_{ascend} = 0.2\text{m/s}$ and $d_{descend} = 0.18\text{m/s}$). This decrease of v permitted the vehicle to conduct maneuvers with a smaller turning radius, especially required to leave the first cave. Figure 51 depicts part of the execution of this query.



(a) Vehicle trajectory, calculated path, goals, and Octomap

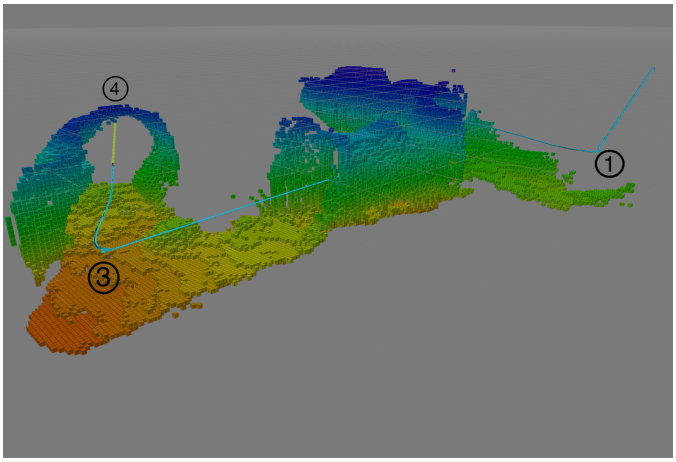


(b) Sparus II traversing the underwater tunnel

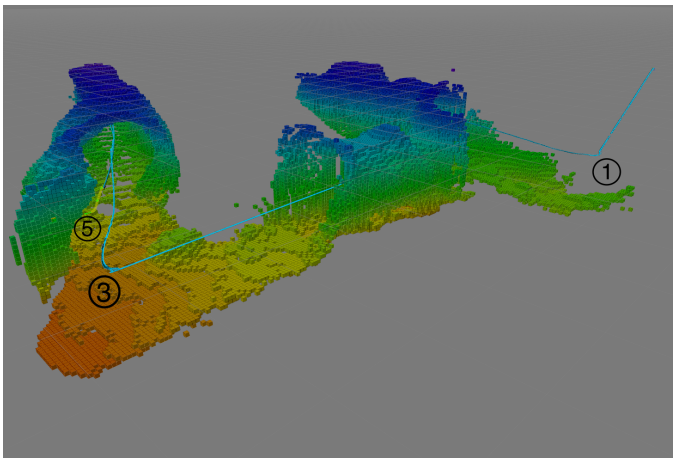
Figure 51: Simulated mission in the caves complex. The third start-to-goal query guided the AUV to the second single-branch cave's entrance. In (a) the vehicle trajectory appears in light blue, and the remaining of the calculated path in yellow. This query required the vehicle not only to find a way out of the first cave, but also to traverse a tunnel that connects the entrances of both single-branch caves.

Once the Sparus II AUV had crossed the tunnel, the fourth query was defined to navigate through and until the end of the second single-branch cave, i.e., $q_{goal_4} = [60.0, 87.0, 9.0, 1.82]$. To do so, the speed constrains were kept as established for the previous query. Once this was accomplished, the fifth query was set to guide the AUV close to a third cave's entrance; this meant $q_{goal_5} = [82.0, 47.0, 17.0, 0.0]$. This latter query required the planner

to find a way out of the second cave, which is considerably narrower than the first one. To cope with this latter situation, the planner used a lower surge speed, $v = 0.1\mathrm{m/s}$, which allowed the vehicle to turn back with a smaller turning radius. Figure 52 depicts part of the execution of these two queries.



(a) Sparus II traversing the second cave

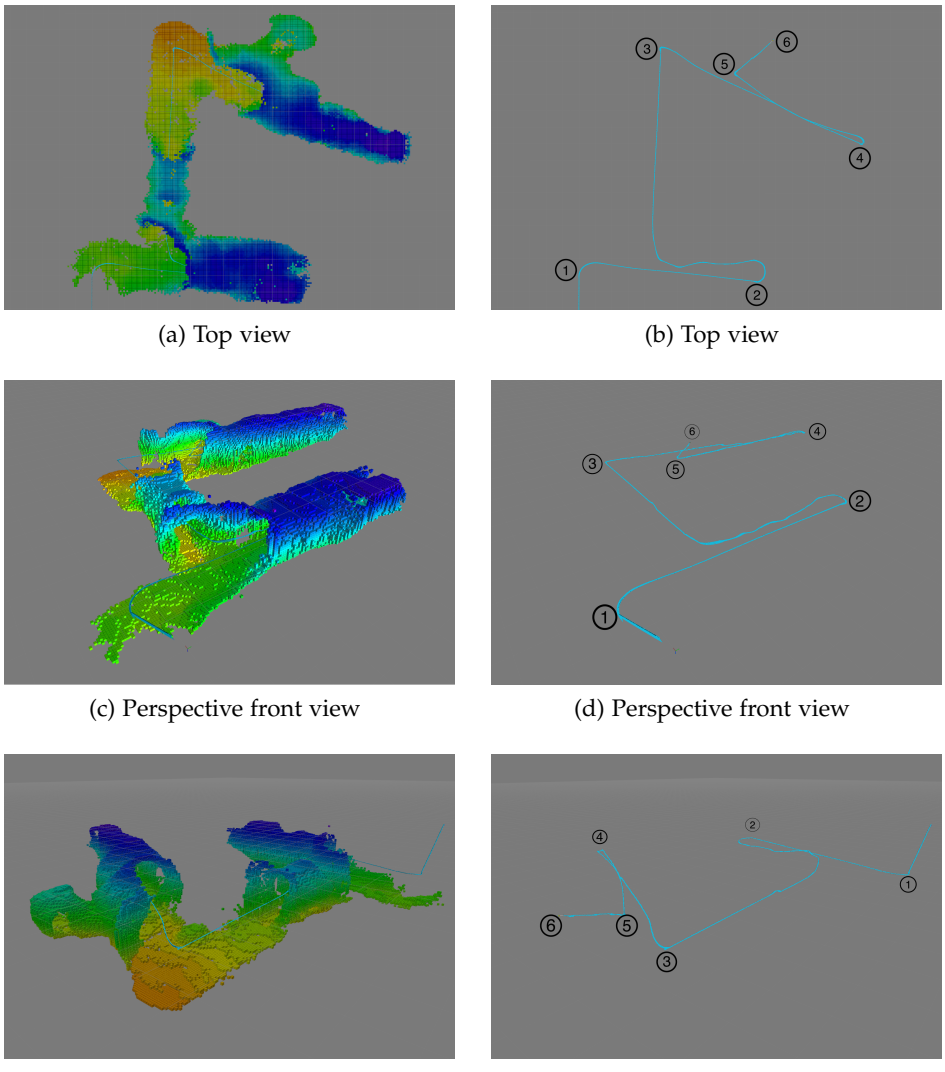


(b) Sparus II on its way back to the third cave's entrance

Figure 52: Simulated mission in the caves complex. (a) The fourth start-to-goal query required the vehicle to navigate through the second single-branch cave. (b) The fifth query was defined to guide the vehicle close to a third cave's entrance. This required the planner to find a way out of the cave. In both images, the vehicle trajectory appears in light blue, and the remaining of the calculated path in yellow.

Finally, Figure 53 presents different views of the whole mission execution, where both the vehicle trajectory and the Octomap built along the mission can be observed.

Figure 53 presented a mission in an environment that has been created from a real-world dataset. This test not only evaluates all the aspects that have been covered throughout this thesis, but also represents a clear advance towards fully autonomous inspections. However and as it was mentioned before, there are still some technical aspects that have to be further developed in order to successfully conduct in-water trials. Some of these aspects will be analysed and discussed in the next chapter.



(a) Top view

(b) Top view

(c) Perspective front view

(d) Perspective front view

(e) Perspective back view

(f) Perspective back view

Figure 53: Simulated mission in the caves complex. The whole mission was composed of six start-to-goal queries that were executed consecutively.

5.4 GAP FILLING AND POTENTIAL TARGET INSPECTION

The experiments presented in the previous sections have attempted to prove the capabilities of the proposed path/motion planning framework. But as it was also mentioned before, in order to make some of these kind of missions a reality, there are still additional technical aspects that have to be tackled. For this reason, some of the framework's validation tests have been limited to conduct simulations over real-world datasets. One example of this is the capability to plan feasible 3D paths, which was evaluated with the exploration of confined natural environments. In order to complement the validation of this feature, this section presents another experiment that has been carried out with the AsterX AUV.

In most of the current operational and commercial AUVs, a mission plan is generally defined in advanced by an operator. Such a mission seeks to cover a predefined area from a constant and safe altitude. To do so, the mission commonly consists of the path to be followed and a series of payload commands. These commands establish when and where sensors must acquire different data along the specific route. Therefore, the nature of the data mainly depends on the mission objectives, as well as the sensors installed in the vehicle, e.g., sonars, optical cameras and even physico-chemical sensors.

In this approach of covering a predefined area, the preplanned route is generally not modified during the survey. Instead, the objective is to sweep the area in order to detect and localize potential targets, which can include geological formations [**Galceran2012**], shipwrecks [**Gracias2013**], and other artificial or natural underwater structures [**Escartin2013**]. A common approach in this kind of missions is to conduct a first exhaustive survey, and then extract and analyse the gathered data to program a second and more specific survey. This latter one attempts to obtain more details about the potential targets, as well as to cover possible gaps resulted during the first exploration. However, this two-survey strategy can be inefficient, since it requires establishing a communication link between the AUV and its operator (located in the mother ship) for retrieving the data and reprogramming a new mission. This is particularly unnecessary in modern AUVs with long-term autonomy.

Aiming to overcome some of the aforementioned limitations, one alternative is to endow an AUV with a mission planner, or high-level controller, that extends its decision-making capabilities. This additional control layer must allow the vehicle not only to conduct predefined surveys, but also to autonomously detect and inspect potential subareas of interest without the need of resurfacing. To do so, the AUV is assumed to be equipped with a looking-downward multibeam sonar, which gathers data from the sea bottom along the initial mission. This data can be then automatically processed onboard in order to detect anomalies and gaps, which are marked as regions that require further inspection, thus enlarging the original survey. In the case of anomalies (i.e., objects that protrude above the surrounding seafloor), the vehicle must conduct closer explorations; while in the case of gaps, it must complete covering the area that was initially defined.

To implement this approach over a particular AUV, it is necessary to first understand its control architecture. In what concerns the AsterX, it is com-

posed of three main functional blocks (see Appendix A, Fig. 62): 1) the low-level controller, or *frontseat*, which runs over QNX; 2) the high-level controller, or *backseat*, which runs over robot operating system (ROS); and 3) the *payload* controller[2]. Having this in mind, the proposed mission planner can be developed within the *backseat*, which allows including different specialized processes, or nodes as referred to in ROS. Figure 54 depicts the *backseat* controller together with the required nodes. It is also important to notice that these nodes not only are capable of exchanging information between each other, but also can externally communicate with the *frontseat* and payload controllers.

The first of these nodes is called the *target/gap detector*, and it uses the multibeam data gathered along the initial survey to detect targets (anomalies) and gaps. The second and central node is called *mission handler*, and it receives a list of the targets and gaps' positions. Around these positions, this node establishes subareas of interest. The third node is the *path planner*, and it is based on the *planning* module presented in Chapter **??**. This node calculates 3D feasible paths to approach and conduct further exploration or coverage of the subareas of interest. Finally, once the planner has provided the paths, the *mission handler* send them to the *path-tracking controller*. This latter node generates the corresponding *frontseat* controller setpoints.

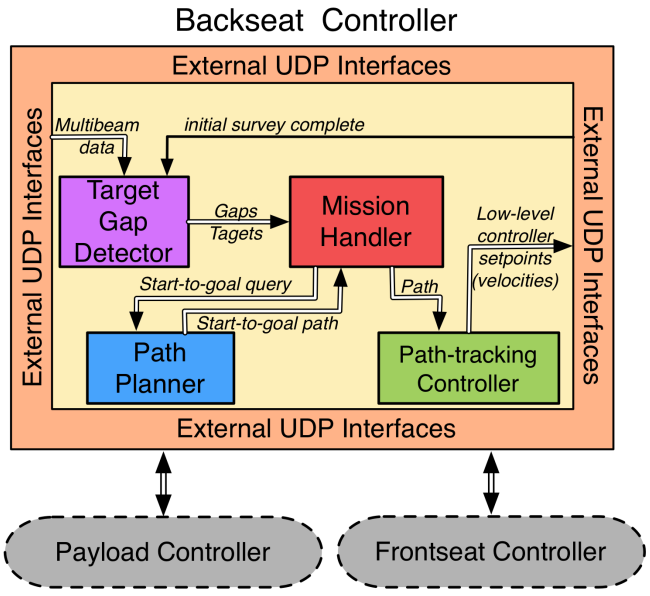
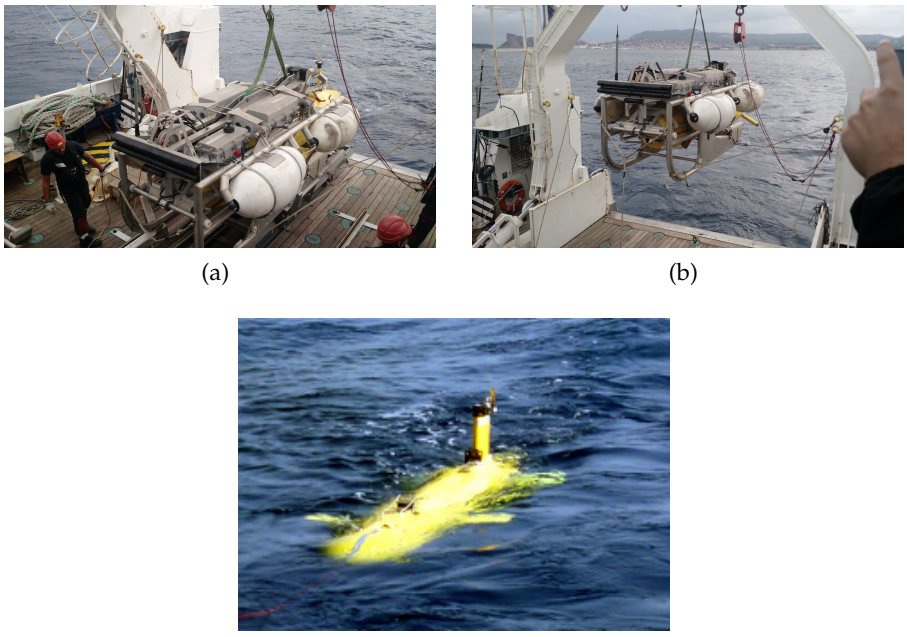


Figure 54: Main modules of the *backseat* controller

An important aspect of the *path planner* used for this experiment is the necessity to take into account the vehicle motion constraints involved. Apart from those constraints associated with a torpedo-shaped AUV, the AsterX can also be limited to navigate at a constant surge speed. This leaves the turning rate as the variable that determines the AUV turning radius. This formulation coincides with the approach presented in the Chapter 3, thus allowing the use of the *planning* module presented in Chapter **??**. In order to validate this new one-survey approach, the AsterX AUV conducted a real-world inspection by using its *backseat* controller with the

2 For more details, the interest readers are encouraged to look into Appendix A

functional nodes mentioned before. Figure 55 depicts the deployment of the AUV before starting the mission.



(a) (b)

(c)

Figure 55: AsterX AUV deployment from its mother ship L'Europe

The mission consisted in detecting and navigating in close proximity to a plane wreck that is located in the bay of La Ciotat, France. Although the approximate location was known, it was not considered accurate enough to send the vehicle closer. Before starting the mission, the AsterX was programmed to follow a predefined coverage survey of the area of interest. To do so, the AUV navigated at 10m deep and with a constant surge speed of 1.5m/s. This initial survey was defined within the *frontseat* controller, which, once completed this first part, handed over the control of the AUV to the *backseat*. Once the *backseat* was informed, it guided the vehicle over a straight line trajectory while keeping the same speed and safe altitude. This maneuver was conducted while the *target/gap detector* processed the multibeam data to detect the potential targets and gaps, if any of them exist. Figure 56 depicts the area of interest, the initial survey, and the potential target and gaps detected from the multibeam data.

After the *detector* provided the locations of the potential targets and gaps, the *path planner* calculated a complementary 3D trajectory to both explore the potential targets and cover the gaps. For the former case, the planner calculated a path 15m over than highest point detected in the target. For the latter case, the planner defined paths to cover the gaps at the deep of the initial survey, i.e., 10m for this particular mission. As the extended mission required navigating at different altitudes, the *planner* calculated 3D paths that met the ascending and descending motion constraints, as it was explained in Chapter 4. Once this calculation was complete, the whole path was sent to *path-tracking controller*. Figure 57 depicts the extended mission to navigate over the potential target and to cover the gaps.

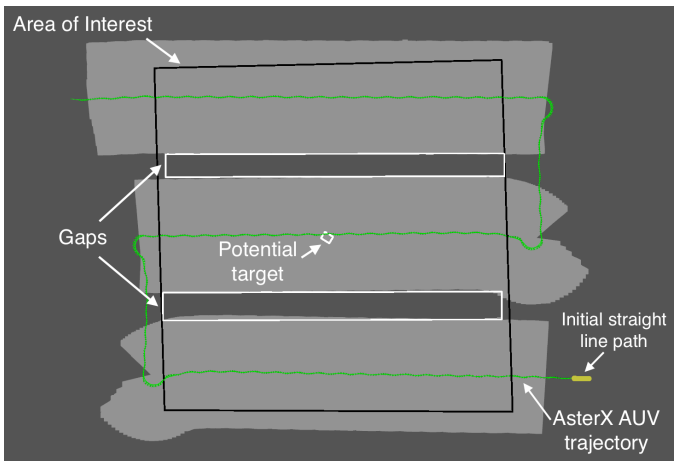


Figure 56: Potential target and gaps detected after completing the initial survey with the AsterX AUV. It can be observed the area of interest in black, the AUV trajectory in green, the area covered with multibeam sonar with light grey, and two gaps and one potential target in white.

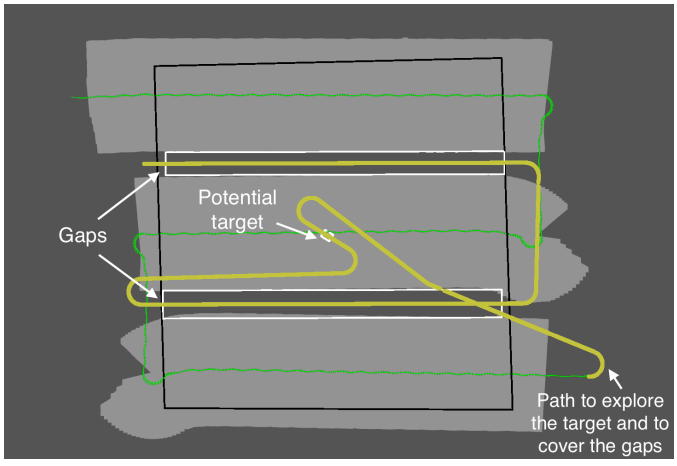


Figure 57: Path planned to further cover the potential target and gaps with the AsterX AUV. It can be observed the new extended mission in yellow. Although the view is from the top, the path is 3D since it requires different altitudes to inspect targets and to cover gaps.

As mentioned in the beginning of this section, this experiment allows further proving the capability of planning 3D AUV paths by using the approach proposed in this thesis. Figure 58 depicts how the AsterX AUV further inspected the target by following a calculated path. To do so, the vehicle descended to a specify altitude with respect to the detected anomaly (target), while conducting turning maneuvers. Both horizontal and vertical motions met the vehicle motion capabilities. Finally, Figure 59 shows the final AUV trajectory after completing the whole mission. It can be observed how the vehicle not only traveled over the target, but also covered the gaps.

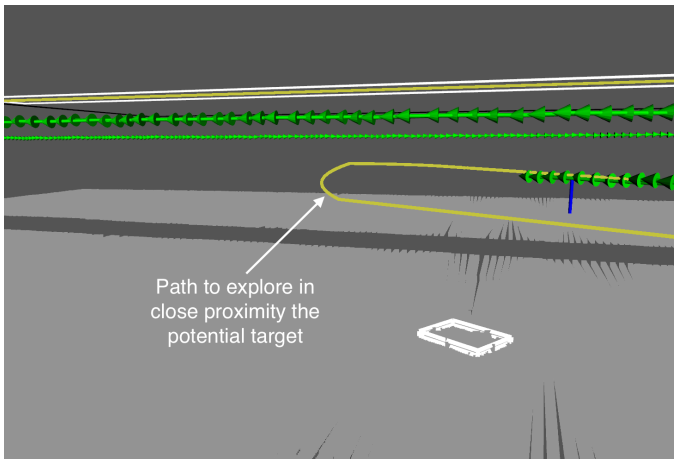


Figure 58: The AsterX AUV approaches to further inspect a potential target. The vehicle trajectory is presented in green, while the calculated path appears in yellow.

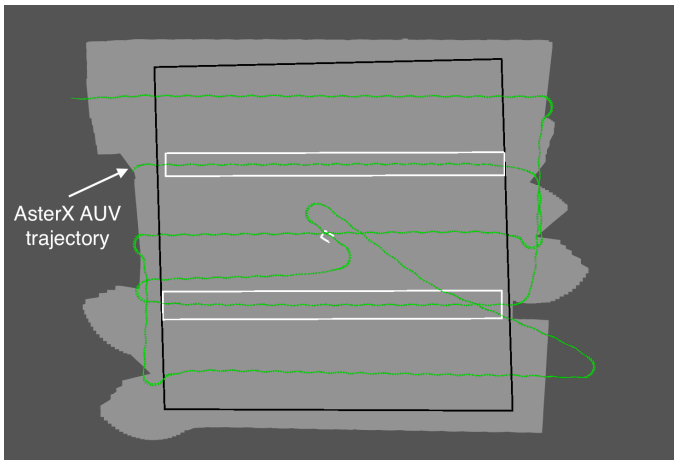


Figure 59: The AsterX AUV completed the mission by inspecting the potential target and covering the gaps from the initial survey. It can be observed the area of interest in black, the AUV trajectory in green, the area covered with multibeam sonar with light grey.

6 CONCLUSIONS

6.1 SUMMARY OF COMPLETED WORK

This thesis has addressed the problem of planning paths online for an autonomous underwater vehicle (AUV), which navigates under motion constraints through unexplored environments. This kind of missions require the vehicle to incrementally map the surroundings, while simultaneously replanning a feasible and safe path. After providing an overview of the general problem and the proposed objectives to tackle it, Chapter 2 presented the state of the art of path/motion planning for robotic systems. This review started with discussing the general planing problem, then it classified the available techniques, and finally it discussed their most common extensions. For this latter part, the review put special emphasis on those methods that have been used with underwater vehicles.

Chapter 3 presented the use of sampling-based methods to plan feasible and constant-depth paths for a torpedo-shaped AUV. Here the term feasible refers to paths that take into consideration the vehicle's limitations, either kinematic or dynamic ones. In the case analysed in this thesis, torpedo-shaped AUVs are kinematically constrained, since they cannot conduct pure lateral motion. Instead, they are required to move forward or backward while conducting turning maneuvers. Two different approaches to consider such limitations were presented and discussed. The first one directly uses the AUV kinematic equation of motion. The second one employs Dubins curves to characterize the AUV motion with straight line segments and circular arcs. Both approaches were evaluated and compared in different simulations. This allowed identifying their advantages and drawbacks, which led to establish the one using Dubins curves as the best approach.

Chapter 4 proposed an alternative formulation to plan feasible and variable-depth paths for a torpedo-shaped AUV. This formulation extended the Dubins curves by including the AUV vertical motion, and it was also used with a sampling-based planning method. This sought to define a general strategy to plan 3D paths, which took into account both the vehicle's lateral and vertical motion constraints. This approach was evaluated in different simulated scenarios, where it was compared with another approach that does not include the vehicle's motion constraints involved. Results proved that the proposed approach allows an AUV to follow more accurately 3D paths.

While simulation tests in Chapters 3 and 4 assumed fully mapped environments, the main objective of this thesis was to endow an AUV with the capability to move through unexplored environments. In order to do so, Chapter **??** introduced an online mapping and path/motion planning framework for AUVs. The framework uses an Octomap to incrementally build a representation of the surroundings. To calculate the AUV path, it includes a sampling-based planner that employs the extended Dubins curves formulation for dealing with 2D and 3D missions. Furthermore, the plan-

ner not only computes a feasible path, but also attempts to obtain a safe one by minimizing its associated risk. Finally, the framework incorporates new strategies that seek to reduce the running time, which is a critical requirement for the intended applications and their online computation limitations. The framework was evaluated with simulations in different scenarios.

In order to completely validate the proposed framework and its properties, Chapter 5 presented different experiments that were conducted by torpedo-shaped AUVs. This included four different real-world scenarios:

1) *Planning constant-depth paths to move through artificial marine structures*. In this case, the Sparus II AUV traversed multiple times a breakwater structure, which is composed of a series of concrete blocks separated by four-meter gaps. At the beginning of the mission, the vehicle was not provided with a map of the surroundings. This required the vehicle to incrementally build a map, while planning a feasible and safe path to the different specified goals. Furthermore, the vehicle was equipped with optical cameras to gather images, which were used to create a 3D reconstruction of the traveled area.

2) *Planning constant-depth paths to move through natural marine structures.* In this case, the Sparus II navigated through a natural underwater canyon made by rocky formations. This experiment sought to prove the framework's capabilities in a more challenging scenario with non-regular shape obstacles. As occurred in the previous experiment, the vehicle was not provided with an initial map of the area, and it was also equipped with optical cameras. The result of the mission also included a photo-realistic 3D reconstruction of the natural scenario.

3) *Planning variable-depth paths to move through confined marine environments.* In the previously mentioned experiments, the vehicle navigated at a constant depth due to a sensor limitation. However, in order to validate the capability to plan 3D paths, a simulated Sparus II conducted missions over a real-world dataset of a cave complex. This scenario required solving consecutive start-to-goal queries of different depths. As occurred with the previous experiments, the vehicle was not provided with an initial map of the surroundings.

4) *The autonomous survey replanning for gap filling and target inspection.* In this case, the AsterX AUV conducted an inspection of an area of interest. The mission was composed of two phases. The first one required the AUV to follow a preplanned coverage path. The second one guided the vehicle to further inspect either potential targets or gaps (i.e., areas not correctly covered). For this latter phase, the extended Dubins curves formulation was used to plan 3D paths to guide the AUV closer from the potential targets. This experiment sought to complement the validation of planning 3D paths with a real-world vehicle.

These tests allowed proving the viability of the proposed approach under real-world conditions. This included the assessment of the computational efficiency, in which a single embedded computer was capable of conducting simultaneously mapping, planning, and control tasks. Furthermore, along the development of the proposed framework, the success rate increased from $10-20\%$ during the initial tests, to $80-90\%$ when using

the final version of the framework. However, it is also important to mention that those failing cases were mainly due to sensor limitations.

6.2 REVIEW OF CONTRIBUTIONS

Aiming to endow AUVs with the capabilities required to operate in unexplored environments, this thesis has proposed an online mapping and path/motion planning framework. This led to extend and develop different strategies that contribute to the current state-of-the-art for underwater vehicles. Such contributions have been presented and peer-reviewed along different Publications, and they can be gathered in four main aspects:

PATH/MOTION PLANNING ONLINE This thesis established that in order to move through unexplored environments, an AUV has to be capable of incrementally mapping the surroundings, while simultaneously planning collision-free paths. However, an initial approach in this thesis was to have two vehicles to conduct missions in unexplored environments. The first vehicle had to navigate at a safe altitude, and it was assumed to be equipped with a downward-looking multibeam sonar that allowed to build online a map. Such a map was used to simultaneously plan collision-free paths for a second vehicle that navigated in close-proximity to the sea bottom [NGCUV'15]. Although the previous approach was limited to simulation tests, it did contribute to define a first mechanism for simultaneous mapping and planning [OCEANS'15]. This mechanism later became the framework thoroughly explained in Chapter **??**. In its first version, the framework used an anytime tree-pruning strategy that *opportunistically check* states for collision [ICRA'15], but it was later improved by *reusing the last best known solution* [IROS'16].

PLANNING FEASIBLE MOTION FOR AUVS Although a simultaneous mapping and planning mechanism allowed an AUV to move through an unexplored environment [ICRA'15], results also showed multiple replanning maneuvers. The main reason was that the AUV was not capable of accurately following the provided paths. This thesis identified the necessity to plan not only collision-free paths, but also feasible ones. This means that the calculated paths must take into consideration the vehicle's motion constraints, thus minimizing unexpected vehicle's trajectories when attempting to follow the calculated path. This thesis proposed and validated the use of Dubins curves to characterize the constant-depth AUV motions [IROS'16], which was later extended to consider full 3D trajectories [JFR'17].

PLANNING SAFE PATHS AUVs operate in complex environments, where they can be affected by external perturbations such as waves and currents. Furthermore, AUVs' navigation system has a position error that cannot be corrected while submerged, especially in environments where a mother ship cannot be used with an USBL system. These operation conditions can lead the vehicle to risky situations, especially when moving in close-proximity to nearby obstacles. This thesis evaluated different alternatives to plan not only collision-free

and feasible paths, but also ones that attempt to minimize the risk of collision. A first alternative was to maintain a safe distance from the surroundings [NGCUV'15]. However, it was later replaced with an optimization function that combines the length and the risk associated with a calculated path [IROS'16].

EXPERIMENTAL EVALUATION This thesis has extensively proved the proposed framework and its newly introduced capabilities. This validation mainly involved the Sparus II AUV. The experiments included simulated and in-water trials in different scenarios, such as artificial marine structures (breakwater) [ICRA'15, IROS'16], natural marine structures (underwater canyon) [SENSORS'16], confined natural environments (caves complex) [JFR'17]. Furthermore, the capability of planning feasible 3D paths was validated with the AsterX AUV [OCEANS'17].

6.3 FUTURE WORK

This thesis cannot be considered a final and definitive solution for the problem of planning feasible and safe paths for AUVs. However, it does contribute a further step towards better and more reliable underwater vehicles. In doing so, this thesis has established the basis for challenging future work that will continue extending the AUVs' capabilities.

IN-WATER TRIALS FOR 3D MAPPING AND PLANNING Even though the framework proposed in this thesis supports both 2D and 3D missions, its capability of mapping 3D environments was only evaluated with simulations over virtual and real-world datasets. This was mainly due to technical limitations, such as the absence of a mechanism that allowed the AUV to rotate a forward-looking multibeam sonar. Therefore, the next and immediate step to continue this work could be to conduct in-water trials in 3D scenarios, such as the caves complex mentioned in Section 5.3.

PLANNING VARYING SPEED MOTIONS The framework proposed in this thesis uses a sampling-based method with Dubins curves as a steering function. This formulation assumes that the AUV moves with a constant surge speed and a maximum turning rate. In this way, the AUV paths can be parameterized with straight line segments and circular arcs of constant radius. However, there are situations in which the vehicle should be capable of conducting tight turns. Some of these situations where presented in Section 5.3, where the AUV surge speed was reduced to allow the planner finding a way out of narrow tunnels. Therefore, another possible extension for future work is to analyse alternatives to plan feasible and safe paths with varying speeds, thus permitting maneuvers with different turning radius.

VIEW/INSPECTION PLANNING As it was mentioned when discussing the experiments presented in Chapter 5, this thesis did not have as an objective to provide a strategy to inspect an area or structure of interest. Another possible future work from this thesis can be the

development of methods to efficiently conduct such inspections. In this case, the proposed framework can be used as a low-level layer for a high-level control pipeline. This latter will have to establish the trajectory that AUV should follow in order to fully inspect an area or structure. As an example of this possibility, some of the characteristics of the proposed framework have been already used in a view planning framework for AUVs [RA-LETTERS'17].

PATH/MOTION PLANNING UNDER UNCERTAINTY One of the requirements identified for moving through unexplored environments was to attempt minimizing the risk of collision. Having this mind, this thesis presented an optimization function that allows combining the length and the risk associated with the path. This approach, however, is heuristically established. Another possible future work is to propose an alternative methodology to calculate safe paths. Such a new approach should consider the different sources of uncertainty to establish the validity of the vehicle's states. In this case, the uncertainty might include the one associated with the model, the controller, the navigation system, and/or the exteroceptive sensors that detect nearby objects. Most of the current work on this matter is still computationally expensive, although there are some promising formulations that could be extended for AUVs. Some these alternatives were analysed during the development of this work, and they are currently under studied as an ongoing work at University of Girona [ICRA'18].

APPENDIX

A

EXPERIMENTAL PLATFORMS

The experimental validation of the framework proposed in this thesis has involved two AUVs. While most of those tests were conducted with the Sparus II AUV, the experiments that proved one of the framework capabilities were done with the AsterX AUV. The following sections explain the main software and hardware aspects of both vehicles.

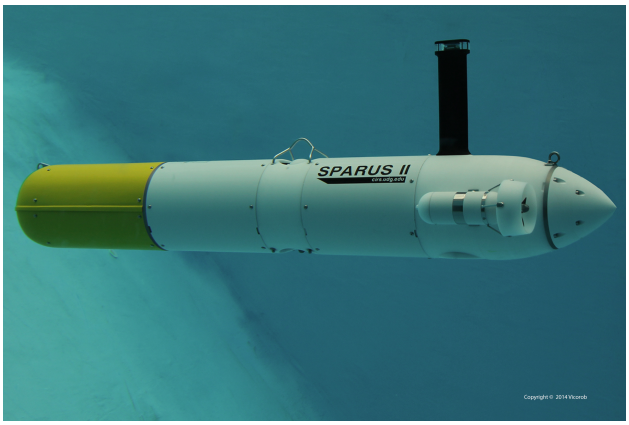
A.1 SPARUS II AUV

The Sparus II is a torpedo-shaped AUV with hovering capabilities, which has been designed and developed at the Underwater Vision and Robotics Research Center (CIRS)[1]. The vehicle is rated for depths up to 200m, and is equipped with three thrusters; two of them are located in the back, and are used for motion on the horizontal plane; the third one is located in the middle, and is dedicated to vertical motion. This implies that the AUV can be actuated in surge, heave and yaw DOF. Furthermore, the Sparus II is equipped with a navigation sensor suite that includes a pressure sensor, a doppler velocity log (DVL), an inertial measurement unit (IMU) and a GPS to receive position fixes while at surface.

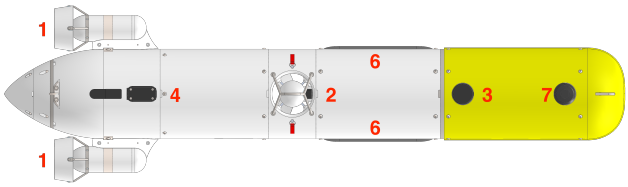
The Sparus II AUV also has communication devices such as an acoustic modem for underwater communication with other vehicles or surface stations (e.g., by using an USBL system), and a Wi-Fi antenna that can be used when the AUV is at surface. Moreover, the vehicle includes a configurable payload area in the front, which contains a set of exteroceptive sensors to perceive and detect the surroundings. This latter group of sensors can be modified according to the mission's requirements, and may include optical cameras, single-beam echosounders, mechanical-scanning (profiler and imaging) sonars, multibeam sonars, etc. Figure 60 depicts different views of the Sparus II AUV, including one where a possible payload configuration can be observed.

In what software concerns, the Sparus II AUV is controlled through the component oriented layer-based architecture for autonomy (COLA2) [**Palomeras2012**], which is a control architecture that is completely integrated with the robot operating system (ROS). Besides operating aboard real robots, COLA2 can interact with the underwater simulator (UWSim) [**Prats2012**], which can import 3D environment models and simulate the vehicle's sensors and dynamics with high fidelity. Furthermore, the use of ROS allows to easily integrate third party tools, such as the open motion planning library (OMPL) that offers a convenient framework that can be adapted to specific path/-motion planning problems [**Sucan2012**].

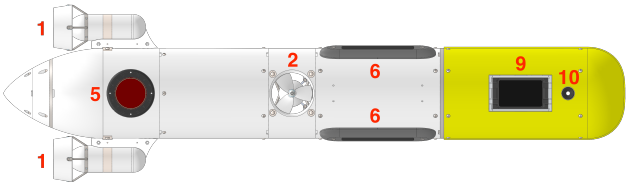
1 CIRS is part of the Computer Vision and Robotics Institute (ViCOROB) in Girona (Spain)



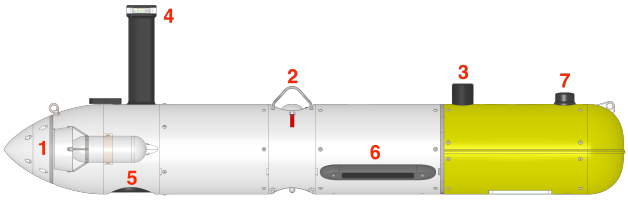
(a) Sparus II in a water tank at CIRS



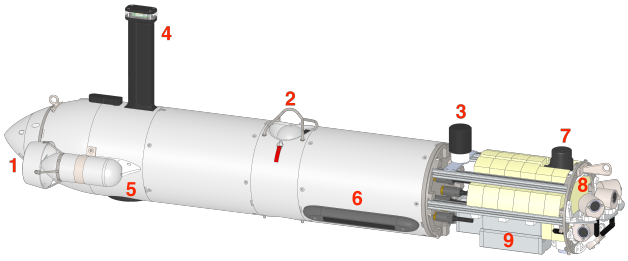
(b) Top view



(c) Bottom view



(d) Lateral view

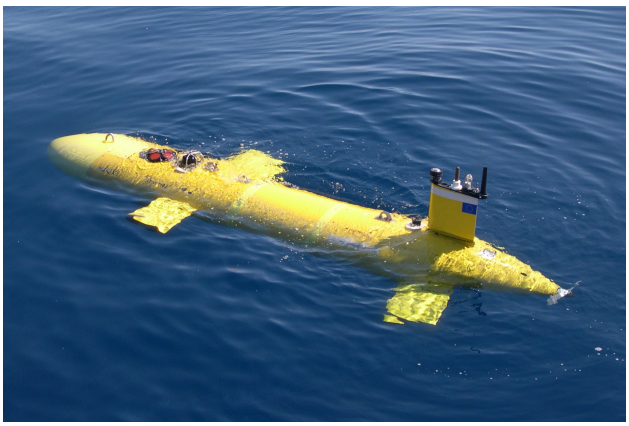


(e) 3D view

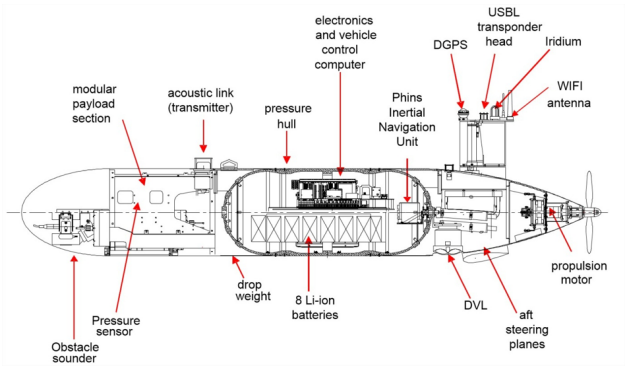
Figure 60: Sparus II AUV: top, bottom, lateral and 3D views. Different hardware parts can be observed, including the thrusters (1,2), the acoustic modem (3), the Wi-Fi and GPS (4), as well as interoceptive and exteroceptive sensors such as the DVL (5), side-scan sonar (6), mechanically-scanning imaging sonar (7), single-beam echosounders (8), multibeam sonar (9), and an optical camera (10).

A.2 ASTERX AUV

The AsterX is a torpedo-shaped AUV from the French Research Institute for Exploitation of the Sea (Ifremer). This vehicle is based on the Explorer 3000 AUV, which is built by International Submarine Engineering (ISE), from Canada. The vehicle is rated for depths up to 3000m, and is equipped with one back propulsion motor, three aft steering planes, and two fore planes. Similarly as the Sparus II, AsterX is equipped with a navigation sensor suite that includes a pressure sensor, a DVL, an IMU and a GPS to receive position fixes while at surface. It also has an acoustic modem, and a Wi-Fi antenna. This AUV also includes a modular payload area in the front, which may carry different sensors, e.g., a multibeam sonar. Figure 61 depicts the AsterX AUV at sea surface, and a schematic with its main inner hardware devices distribution.



(a) AsterX AUV at sea surface during in-water trials



(b) Lateral view

Figure 61: AsterX AUV at sea surface and its lateral view with a description of the different hardware elements.

On the other hand, the AsterX's software architecture is composed of three main functional blocks (see Fig. 62). Firstly, the AUV low-level controller, also referred as *frontseat*, guides the vehicle using the automated control engine (ACE) middleware. This controller is executed over QNX operating system, thus guaranteeing real-time computation constraints. Furthermore, this functional block also handles the vehicle's navigation and safety routines. Secondly, the *backseat* controller extends the vehicle's capabilities and applications by easing the implementation of high-level

routines, such as algorithms for path/motion planning, path-tracking, and docking. These routines can directly send low-level control setpoints. This functional block has its own dedicated computer that works under the ROS (over Linux). Finally, the *payload* controller acts as a bidirectional interface between the interoceptive and exteroceptive sensors, and, on the other hand, both the *frontseat* and *backseat* controllers.

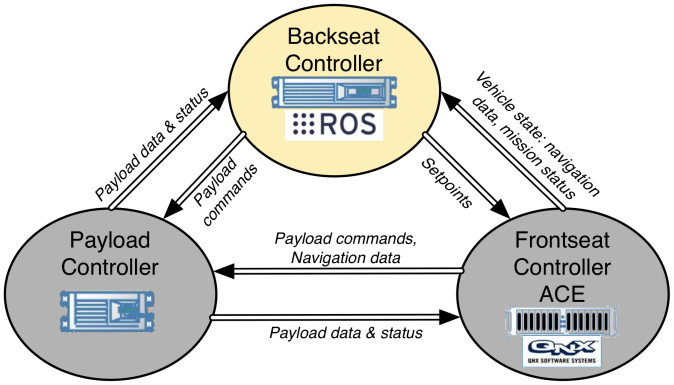


Figure 62: AsterX AUV software architecture