Universitat de Girona

PATH PLANNING WITH HOMOTOPIC CONSTRAINTS FOR AUTONOMOUS UNDERWATER VEHICLES

Emili HERNÀNDEZ BES

Dipòsit legal: Gl. 1193-2012
http://hdl.handle.net/10803/83568

Ph.D. thesis

PATH PLANNING WITH HOMOTOPIC CONSTRAINTS FOR AUTONOMOUS UNDERWATER VEHICLES

EMILI HERNÀNDEZ BES

2012

Universitat de Girona

Ph.D. thesis

PATH PLANNING WITH HOMOTOPIC CONSTRAINTS FOR AUTONOMOUS UNDERWATER VEHICLES

EMILI HERNÀNDEZ BES

2012

Doctoral Programme in Technology

Supervisors
Dr. Pere Ridao Rodríguez and Dr. Marc Carreras Pérez

Work submitted to the University of Girona in fulfilment of the requirements for the degree of Doctor of Philosophy

Dedicated to my family and especially to Sara.

ACKNOWLEDGMENTS

I would like to express my gratitude to those who helped me during the development of this thesis. First of all, I would like to thank my supervisors Marc Carreras and Pere Ridao for believing in me and encouraging me to realize this research work. Without their advices and optimism in those moments when things did not work, this work would not have been possible.

I also thank the members of the Underwater Robotics Lab. for the help whenever it was necessary. I want to thank my colleagues Narcís, David, Andrés, Aggelos, Tali, Lluís, Enric, Arnau, Chee Sing, Simone and Carles for always being there. I would also like to extend my gratitude to the rest of the members of the Computer Vision and Robotics Group of the Department of Computer Engineering, in particular to Xevi, Jordi, Xavi, Rafa, Miki, Quintana, Tudor, Sergio, Sik, Quim and Aulinas.

I am also grateful to the people in the Systems, Robotics & Vision Group at the Mathematics and Computer Science Department of the University of the Balearic Islands for their warm reception and hospitality during my stay. I especially want to thank Javier Antich for his useful advices and those interesting discussions, and Alberto Ortiz for his help during the stay. Thank you Francesc, Toni, Biel, Julian, Oscar and Xisco for making me feel like a member of the group.

I also wish to extend my gratitude to all my friends, in particular to Ivan for his enormous patience and those wonderful dives that helped me to break with the routine of the last period of the Ph.D.

I give my special gratitude to my parents, Rosa and Emili and my brother Edu. They deserve special gratitude for their comprehension and support of my decision to start a Ph.D. Thank you Joan for helping me with those obstacle's panels. Thank you Maribel, Adriana and Carla for being there.

Finally, I want to express my gratitude to Sara for her love, patience and comprehension at every moment.

ABSTRACT

The work presented in this thesis addresses the path planning problem for Autonomous Underwater Vehicles (AUVs). Our method proposes the utilization of homotopy classes to provide topological information on how paths avoid obstacles. Looking for a path within a homotopy class constrains the search into a specific area of the search space, speeding up the computation of the path. The method starts by generating the homotopy classes that connect the starting point with the ending point in a workspace with obstacles. Those classes which most probably contain lower cost solutions are determined by means of a lower bound criterion before computing a path. Finally, a path planner uses the topological information of homotopy classes to generate a few probable good solutions very quickly. Three path planners from different approaches have been proposed to generate paths for the homotopy classes obtained. The first is the Homotopic A* (HA*), a graph-search based algorithm that computes the shortest homotopic path according to an input homotopy class. The second is a probabilistic sampled-based approach called Homotopic RRT (HRRT). The last is the Homotopic Bug (HBug), a Bug-based algorithm that combines following the lower bound path with the surroundings of the obstacle boundaries.

This thesis also proposes a local map building approach to generate Occupancy Grid Maps (OGMs) where path planning is performed. The method first improves the dead-reckoning navigation of an AUV through a sonar scan matching technique to generate a more feasible OGM according to the information provided by the onboard sensors.

The local map building approach has been tested in a dataset gathered with an AUV. Results of our path planning approach in synthetic environments have shown that the HA* should be used when generating optimal solutions at the expense of low performance is required, whereas the HBug is suitable for applications where the time to perform the path planning is highly constrained. The map building approach and the path planning method have been tested together in real experimentation with the Sparus AUV in a controlled unknown environment.

RESUM

Aquesta tesi aborda el problema de la planificació de camins per a Vehicles Submarins Autònoms (AUVs). El nostre mètode proposa la utilització de classes d'homotopia per a proporcionar informació topològica de com els camins eviten els obstacles. Calcular un camí dins d'una classe d'homotopia permet limitar l'espai de cerca accelerant-ne el càlcul de la solució. El mètode proposat comença amb la generació de les classes d'homotopia que connecten el punt inicial amb el punt final d'un *workspace* amb obstacles. Aquelles classes que probablement contenen les solucions de menor cost s'identifiquen per mitjà d'un criteri de *lower bound* sense haver de calcular el camí al *workspace*. Finalment, un planificador de camins utilitza la informació topològica de les classes d'homotopia per generar solucions segons les classes seleccionades molt ràpidament. Els camins de les diferents classes d'homotopia obtingudes es generen per mitjà de tres planificadors de camins que segueixen propostes ben diferenciades. El primer és l'Homotopic A* (HA*), un algoritme de cerca en grafs que calcula el camí òptim per a una classe d'homotopia utilitzada com a paràmetre d'entrada. El segon és un algoritme probabilístic basat en la generació de mostres aleatòries anomenat Homotopic RRT (HRRT). L'últim és l'Homotopic Bug (HBug), un algoritme Bug que combina el seguiment del *lower bound* amb el seguiment del contorn dels obstacles.

En aquesta tesi també es proposa un mètode per generar mapes locals basats en Occupancy Grid Maps (OGMs), on posteriorment es realitza la planificació de camins. El mètode primer millora la navegació basada en *dead-reckoning* de l'AUV mitjançant una tècnica de *scan matching* que permet generar OGMs més fidedignes d'acord amb la informació proporcionada pels sensors a bord del vehicle.

La proposta presentada per a la construcció de mapes locals s'ha provat en un dataset adquirit amb un AUV. Els resultats obtinguts amb el mètode de planificació de camins en escenaris sintètics han posat de manifest que l'HA* s'hauria d'utilitzar quan es requereix generar solucions òptimes sempre i quan es pugui assumir els seus alts temps d'execució, mentre que l'HBug és un bon candidat per aplicacions on el temps per a planificar camins està altament restringit. La proposta de construcció de mapes i el mètode de planificació de camins s'han testejat conjuntament en un experiment real utilitzant el Sparus AUV en un entorn desconegut i controlat.

RESUMEN

El trabajo de esta tesis aborda el problema de la planificación de caminos en Vehículos Submarinos Autónomos (AUVs). Nuestro método propone la utilización de clases de homotopía para proporcionar información topológica de cómo los caminos evitan los obstáculos. Realizar una búsqueda del camino siguiendo una clase de homotopía específica permite limitar el espacio de búsqueda, acelerando el cálculo de la solución. El primer paso del método presentado genera las clases de homotopía que conectan el punto inicial con el punto final en un *workspace* con obstáculos. Aquellas clases que probablemente contienen las soluciones de menor coste son identificadas mediante un criterio de *lower bound* sin necesidad de calcular ningún camino en el *workspace*. Por último, un planificador de caminos utiliza la información topológica de las clases de homotopía con el fin de generar soluciones para las clases seleccionadas muy rápidamente. Para generar los caminos de las diferentes clases de homotopía obtenidas, se ha propuesto tres planificadores de caminos que siguen estrategias bien diferenciadas. El primero es el Homotopic A* (HA*), un algoritmo de búsqueda en grafos que calcula el camino óptimo según la clase de homotopía seleccionada como parámetro de entrada. El segundo es un algoritmo probabilista basado en la generación de muestras aleatorias llamado Homotopic RRT (HRRT). El último es el Homotopic Bug (HBug), un algoritmo Bug que alterna el seguimiento del *lower bound* con el del contorno de los obstáculos.

En esta tesis también se presenta un propuesta de un método para generar mapas locales basado en Occupancy Grid Maps (OGMs) donde posteriormente se realiza la planificación de caminos. El método primero mejora la navegación basada en *dead-reckoning* del AUV mediante una técnica de *scan matching* que permite generar OGMs más fidedignos según la información proporcionada por los sensores a bordo del vehículo.

El método propuesto para la construcción de mapas locales ha sido testeado con un dataset adquirido con un AUV. Los resultados obtenidos con el método de planificación de caminos en escenarios sintéticos han puesto de manifiesto que el HA* debería de ser utilizado cuando se requiere generar soluciones óptimas en aplicaciones donde sus altos tiempos de ejecución no sean un problema. Por el contrario, el HBug es un buen candidato para aplicaciones donde el tiempo para obtener los caminos debe ser muy reducido. La propuesta presentada para la construcción del mapa y el método de planificación de caminos han sido probados conjuntamente en un experimento real utilizando el Sparus AUV en un escenario desconocido y controlado.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACRONYMS

AAC	Architecture Abstraction Component.
AD*	Anytime Dynamic A*.
ADRRT	Anytime Dynamic RRT.
AHRS	Attitude and Heading Reference System.
AI	Artificial Intelligence.
ARA*	Anytime Repairing A*.
ARRT	Anytime-RRT.
ASC	Autonomous Surface Craft.
ASEKF	Augmented State EKF.
AUV	Autonomous Underwater Vehicle.
BFS	Breadth-First Search.
C-space	Configuration Space.
CCD	Charge Coupled Device.
CL	Component Labeling.
COLA2	Component Oriented Layer-based Architecture for Autonomy.
DES	Discrete Event System.
DFS	Depth-First Search.
DGPS	Differential Global Positioning System.
DoF	Degree of Freedom.
DSTL	Defence Science and Technology Lab.
DVL	Doppler Velocity Log.
EKF	Extended Kalman Filter.
ENU	East-North-Up.
EST	Expansive-Space Tree.
FM	Fast Marching.

GPS	Global Positioning System.
GVD	Generalized Voronoi Diagram.
HA*	Homotopic A*.
HBug	Homotopic Bug.
HIL	Hardware In the Loop.
HRRT	Homotopic RRT.
I-AUV	Intervention-Autonomous Underwater Vehicle.
ICP	Iterative Closest Point.
IFoG	Intelligent Fiber-optic Gyro.
IGC	Intelligent Gyro Compass.
MCL	Mission Control Language.
MCS	Mission Control System.
MPS	Multibeam Profiling Sonar.
MRU	Motion Reference Unit.
MSIS	Mechanical Scanned Imaging Sonar.
O2CA2	Object Oriented Control Architecture for Autonomy.
OGM	Occupancy Grid Map.
pIC	Probabilistic Iterative Correspondance.
PID	Proportional Integral Derivative.
PNP	Petri Net Player.
PRM	Probabilistic RoadMap.
r.g.v	random Gaussian variable.
RL	Reinforcement Learning.
ROV	Remotely Operated Vehicle.
RRT	Rapidly-exploring Random Tree.
SAUC-E	Student Autonomous Underwater Challenge-Europe.
SBL	Single-query Bi-directional Lazy collision-checking.
SLAM	Simultaneous Localization And Mapping.
SVS	Sound Velocity System.

TAM	Thruster Allocation Matrix.
ULV	Ultra Low Voltage.
USBL	Ultra-Short BaseLine.
UUV	Unmanned Underwater Vehicle.
VICOROB	Computer Vision and Robotics.

1 INTRODUCTION

Despite covering almost three quarters of the Earth's surface, seas and oceans represent some of the least known areas of the planet due to the technological challenges that underwater research implies. Unmanned Underwater Vehicles (UUVs) are useful devices to perform exploration, inspection and intervention operations in oceans and internal waters. The development of Remotely Operated Vehicles (ROVs) during the 80s has had a huge impact in various application areas such as the oceanographic field, where ROVs have reduced the need for manned submersibles, increasing the observation time from a few hours to a few days. These robots have also played an important role in industrial field applications for the inspection of submerged structures like cables, pipes or dams, allowing intervention capabilities as well as rescue operations.

In order to overcome the limitations imposed by ROVs'tether cable, which highly restricts the working area of the vehicles, and to increase their autonomy and reduce the human intervention for operating the vehicles, Autonomous Underwater Vehicles (AUVs) have been developed. These vehicles are built with autonomous capabilities in mind. They cover a wide range of potential applications from data sampling to structure inspection. There are several AUV applications being explored by various organizations around the world (Davis, 1996): environmental monitoring, oceanographic research and maintenance/monitoring of underwater structures are just a few examples. AUVs are attractive for use in these areas because of their size and their non-reliance on human operators. Recently, some research institutions have started to develop a new generation of AUVs with one or more manipulators called Intervention-Autonomous Underwater Vehicles (I-AUVs). Their main advantage is the low operational cost, since there is no need for large intervention ships (or oceanographic ships) with dynamic positioning capabilities. I-AUVs are designed to be operated very close to the seabed or in close proximity to industrial structures like those used by offshore industries. Given the potential applications and advantages of AUVs and I-AUVs, academic and commercial organizations around the world are conducting research using these vehicles.

This thesis explores the path planning capabilities of a navigation, mapping and guidance system for an AUV, whose main goal is to achieve a safe guidance of the vehicle in complex scenarios where intervention and surveilling applications are being carried out. After studying the main path planning approaches applied to robotics, this work has focused on generating topological information which is used to constraint the path generation. Consequently, the necessity of developing path planning algorithms that take into account topological constraints arises. In order to demonstrate the feasibility of our

proposal, it has been a priority to carry out experiments with real data from AUVs.

This introduction continues with the main aspects which have conditioned this thesis. First, some background information on the motivations and applicability is provided. Then, a description of the objectives of this thesis and an outline of this document are given.

1.1 MOTIVATIONS

The research presented in this thesis was carried out in the Underwater Robotics Laboratory of the Computer Vision and Robotics (VICOROB) group of the University of Girona. This group has been doing research in underwater robotics since 1992, supported by several National and European programs. The main contribution over the past few years has been the development of several ROV and AUV prototypes. The most recent vehicles, all operative at the present date, are the Ictineu AUV, the Sparus AUV and the Girona 500 I-AUV.

The Ictineu AUV is a small form factor with remarkable sensorial capabilities and easy maintenance which makes it a perfect research platform for testing in both laboratory and real application environments. The Sparus AUV was designed as a small, simple torpedo-shaped vehicle with hovering capabilities, which increases the vehicle's autonomy with respect to the Ictineu AUV when traveling long distances. Both prototypes where conceived to participate in the 2006 and 2010 editions respectively of the Student Autonomous Underwater Challenge-Europe (SAUC-E) competition, but keeping in mind their later use in a wide range of applications such as the inspection of hydroelectric dams, mosaicking of areas with biological interest, harbors, underwater cables and pipes.

In the context of a National program, the lab's latest prototype, the Girona 500, a reconfigurable I-AUV designed for a maximum operating depth of up to 500m was developed. The vehicle is composed of a frame supporting three torpedo-shaped hulls providing a good hydrodynamic performance and a large space for housing equipment while maintaining a compact size which allows the operation of the vehicle from small boats. Concerning the basic configuration, the vehicle is equipped with navigation sensors and basic survey equipment. Moreover, the vehicle has a reserved space for a mission-specific payload such as a stereo imaging system or an electric arm for manipulation tasks.

The research efforts in the Underwater Robotics Laboratory have been oriented to the development of the diverse disciplines related with the operation of autonomous vehicles. An example can be found in the work done in control architectures, which has lead to the creation of the Object Oriented Control Architecture for Autonomy (O2CA2) control architecture (Carreras et al., 2001), recently substituted by a new layer-based control architecture named Component Oriented Layer-based Architecture for Autonomy (COLA2) implemented in the three operative prototypes. As the complexity of the autonomous mis-

sions increased, advances were made towards the development of a Mission Control System (MCS) (Palomeras, 2011). Also Artificial Intelligence (AI) capabilities have been studied by using Reinforcement Learning (RL) techniques (El-Fakdi, 2011) for automatic behavior learning.

Navigation has been addressed putting special efforts into vehicle localization since it constitutes a major underwater problem. Different approaches are currently being explored in the lab. The first is by means of Simultaneous Localization And Mapping (SLAM) techniques (Ribas et al., 2010; Ridao et al., 2011), which require neither previous knowledge of the scenario nor the use of absolute positioning systems. Relative localization techniques based on scan matching, which improve the localization of the vehicle by overlapping successive sonar range scans provided by the sensors onboard the vehicle (Hernández et al., 2009b), have also been studied. Recent achievements rely on the combination of scan matching and SLAM in order to obtain the high quality results of SLAM with the low computation time of scan matching (Mallios et al., 2009, 2010a, 2011).

The principle motivation of this thesis is to expand the autonomous navigation and guidance capabilities of AUVs in inspection, intervention and surveilling applications. This kind of mission usually requires guiding the vehicle at a close distance to the seafloor or to the surface to explore, which makes the robot to carry on the mission in complex scenarios. Following the studies carried out in mobile robotics (Mínguez et al., 2004), developing a navigation and guidance system capable of generating safe trajectories between known positions is required. The vehicle must be able to sense the environment and build a local map of the surroundings to compute a safe path towards the goal. After that, a path following algorithm has to guide the vehicle along the computed path with a simple reactive obstacle avoidance behavior which ensures the safety of the AUV in case of imminent collision.

This thesis focuses on the path planning capabilities of a navigation and guidance system for an AUV. In inspection and surveilling missions, the vehicle is not only required to compute a safe path between a start and a goal position, it must generate paths that avoid obstacles in a specific manner. This dissertation goes a step further and surveys most of the path planning approaches that have offered important advances in robotic applications, paying special attention to those which use homotopy classes since they provide a topological description of how paths avoid obstacles with respect to a set of obstacles and can be used to constrain the path search.

1.2 GOAL OF THE THESIS

As stated earlier, the goal of this thesis is to explore the path planning capabilities in the context of a navigation and guidance system for an AUV. Path planning is a discipline that has been widely studied. However, a small number of algorithms have been applied to underwater robotics since most of the missions are carried out in the open ocean or along the coast where the

presence of obstacles is relatively low and the common strategy to avoid them is to keep a safe distance from the seafloor.

The purpose of the work presented in this thesis is to provide trajectories for other types of missions, such as surveys and/or search tasks where the robot is required to navigate at a fixed altitude in bottom-following mode while acquiring opto-acoustic imagery. For this purpose, the robot has to navigate in environments where high deliberative capabilities are required since the number of obstacles and the complexity of the environment increase. This is the case in applications like benthic habitat mapping, underwater archeology and cable or pipe inspection. Although path planning for AUVs is naturally formulated in 3D, under exposed conditions it can be considered that the robot is only required to generate a 2D map parallel to the sea floor, where any area with a slope greater than a certain threshold behaves as a 2D obstacle. This thesis has considered this fact and the 3D formulation of the problem is beyond its scope.

AUVs are usually designed to maximize their autonomy. Because of this, most of these vehicles have limited computational capabilities, which are commonly shared with online data processing for map building and/or obstacle avoidance purposes. For this reason, it is necessary to keep the computational load for path planning tasks relatively low. Moreover, the applications listed before usually require the generation of safe paths from two known positions that avoid obstacles in a specific manner, allowing the robot to navigate through regions of interest as well as avoiding potentially dangerous zones. In this dissertation both objectives are studied through the utilization of topological description of paths provided by homotopy classes, and how to use them for path planning purposes. Topological information provides information of how paths would avoid obstacles before generating any paths at all. Furthermore, with this information, it is possible to know in which areas of the scenario the solution path will be generated. Thus, path planning algorithms can take advantage of this topological description to accelerate the path generation process. Once a path has been found, the real time guidance of the vehicle is assumed to achieve it, conveniently merging a path following algorithm with a simple reactive obstacle avoidance technique. This system is not within the scope of this dissertation.

1.2.1 *Objectives*

After reviewing the research motivations and describing the problem, the goal of this thesis is stated here as:

Development of path planning techniques for autonomous underwater vehicles on sonar maps built with their onboard sensors.

The objective of this dissertation has been oriented to the autonomous navigation and guidance of AUVs in this lab. The main goal is to develop a method to automatically extract high-level topological knowledge of a given environment by means of homotopy classes and use this information in path

planning algorithms. This dissertation reviews the main path planning approaches and focuses specially on those which use topological information to constrain the path search, which are usually not taken into account by the robotics community.

In most autonomous navigation and guidance applications, the AUV is deployed with limited or no previous information about the environment. Therefore, the vehicle has to be able to generate an internal representation of the scenario where path planning is then performed. For this reason and taking into account the previous work done in the Underwater Robotics Lab., it is also an objective of this thesis to develop a local map building system to improve the dead-reckoning navigation of the vehicles in realtime in order to increase the feasibility of the map.

Although it is necessary to specify the theoretical properties of the algorithms proposed, it is also very important to show their applicability in real conditions. For this reason, carrying out experiments with real data from AUVs in order to demonstrate the achieved research advances has been a priority.

The goal of this thesis can be divided into the following more specific objectives:

TOPOLOGICALLY CONSTRAINED PATH PLANNING. To study the most relevant path planning approaches putting special attention into those techniques that constrain the path search topologically using homotopy classes. The objective is to constrain the generation of paths that avoid obstacles in different manners, which are best suited for surveilling purposes or to avoid certain undesirable zones.

PATH PLANNING FOR AUVS. Once the topological information of how paths avoid obstacles between the start and goal positions is provided by means of homotopy classes, a path planning algorithm has to generate a feasible path to be followed by a robot. The objective is to develop several algorithms based on different approaches since it is a subject of study to highlight the advantages and disadvantages of each proposal in order to specify what type of solution best fits in each context.

LOCAL MAP BUILDING. The applicability of our path planning method is tested on maps generated with the AUV's onboard sonar sensors. The accurateness of the map depends on the localization estimation itself and on the precision of the sensor readings. Although it is not the main goal of this research work, different sensor models are studied and developed in order capture all the information provided by the sensors on our vehicles into the map.

SCAN MATCHING NAVIGATION. In order to generate more feasible maps of the navigated area where the path planning has to be performed, a suitable scan matching technique developed for mobile robots is adapted to work with the sonar sensors which AUVs are commonly equipped with.

TEST AND RESULTS. The evaluation of path planning methods, either in synthetic or in real environments. The goal is to compare the proposed algorithms and extract their strong and weak points. In this work, we put special emphasis on real experimentation. Given an unknown scenario, it is expected that the robot is capable of building an internal representation of the environment, improved with scan matching techniques. The final map is used by the AUV to generate paths according to the set of homotopy classes that allow avoidance of obstacles in different manners.

1.3 OUTLINE OF THE THESIS

This thesis can be divided into three different parts. The first part consists of an state of the art about path planning techniques, paying special attention to those methods that use homotopy classes to constrain the path search (Chapter 2). Based on the methods studied, Chapter 3 presents a new path planning method that uses homotopy classes to constrain the search for the path. The method provides general rules to generate homotopy classes in any 2D workspace that can then be accomplished by path planners. The second part of the thesis focuses on a map building approach for navigation purposes (Chapter 4) which improves the dead-reckoning navigation of AUVs through scan matching techniques to generate more feasible maps with an Occupancy Grid Map (OGM) algorithm. The resultant maps are used to perform path planning with the method developed in part one. Finally, the third part of this dissertation (Chapter 5) details the experimental platforms which are compounded by the Ictineu AUV and the Sparus AUV. The experimental results with the map building and path planning approaches are shown in Chapter 6, whereas Chapter 7 summarizes the contributions and future work. A brief description of each chapter is presented next.

CHAPTER 2 *State of the Art*. This chapter reviews the main path planning techniques for robotic applications. Its main goal is to provide an insight into the different techniques studied. Because of this, a description of the main features of each approach with some representative algorithms shown in detail is provided. The chapter starts with a description of the graph-based search, probabilistic sampled-based algorithms and bug-based algorithms. An insight into anytime path planning techniques is also provided. Further, topological path planning approaches are presented, paying special attention to those techniques based on homotopy classes. Finally, the chapter shows some path planning solutions adopted for AUVs.

CHAPTER 3 *Path Planning with Homotopy Class Constraints*. This chapter describes our proposal to generate homotopy classes whose paths can be followed in any 2D workspace. Then, it describes a heuristic estimator used as a lower bound criterion to set up a preference order when looking for paths of the homotopy classes. The path for each homotopy class can be generated with three different proposals; the first is a graph-based

search algorithm, the second exploits the capabilities of the probabilistic random sampling, and the third focuses the search according to the strategies of the bug-like algorithms. The theoretical properties of each algorithm are also detailed.

CHAPTER 4 *Local Map Building*. This chapter describes a map building approach to generate an internal representation of the environment where the path planning is performed. The chapter is divided into two main parts. The first one focuses on improving the navigation of the AUV through a scan matching technique designed to deal with the noisy data and motion induced distortion introduced by sonar sensors with rotation heads. The second part of the chapter details the utilization of the improved navigation to generate more reliable maps with an OGM technique.

CHAPTER 5 *Experimental Platforms*. This chapter introduces the Ictineu AUV and the Sparus AUV, the two vehicles used in the experimental phase of this research project. Details of the main sensors used for map building and the control architecture that operates the vehicles are also provided.

CHAPTER 6 *Results*. This chapter presents the experimental results of this thesis, which are divided into four main sections. The first shows the results obtained with the map building procedure described in Chapter 4 applied to a dataset gathered in a man-made marina environment. Then, the capabilities of the path planners constrained with homotopy classes proposed in Chapter 3 are shown in different synthetic environments. A comparison of well known path planners is also included. In the third section, the map building procedure and the path planning method are merged into one single experiment performed with the Sparus AUV. Finally, at the end of the chapter, the applicability of our path planning method on a bathymetric map is shown.

CHAPTER 7 *Conclusions*. This chapter concludes the thesis with a summary of all the work done, pointing out contributions and future work. It also comments on the research evolution and the publications accomplished during this research project.

2 STATE OF THE ART

The work presented in this thesis concerns a system for an AUV that first builds an internal map according to sonar sensor readings. This map is then used to perform deliberative path planning to generate safe and feasible paths towards a goal position. Since most of the work has been focused on the path planner, this chapter contains a survey of different path planning techniques that have been studied in the development of our proposal.

After an introduction that briefly describes the path planning problem in Section 2.1, the survey explores graph search-based, probabilistic sample-based and bug-based path planning approaches in sections 2.2, 2.3 and 2.4 respectively. Then, Section 2.5 exposes anytime path planning techniques, which assume that in real world applications, robots have a limited time in which to operate. Section 2.6 introduces topological path planning and Section 2.7 focuses on solutions that explicitly take into account homotopy classes. Section 2.8 reports path planning solutions for AUVs. Finally, a discussion of the survey is given in Section 2.9.

2.1 THE PATH PLANNING PROBLEM

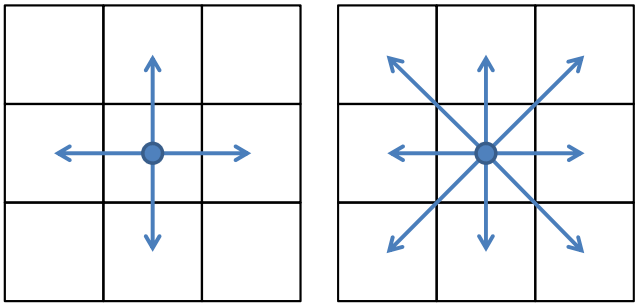
In robotics, the path planning problem consists of computing a safe path in the workspace that can be followed by a vehicle without collisions. Computing the path directly in the workspace is a very complex task since it requires taking into account the specific constraints of the robot such as size, shape and Degrees of Freedom (DoFs). Because of this, path planning is performed on the Configuration Space (C-space), where the robot is represented as a single reference point called *configuration* q. Every configuration is a vector of all the parameters that represent the robot in the environment in a given position, orientation, energy, etc. The dimension of the C-space is determined by the number of parameters of the configuration.

2.1.1 *Overview*

The set of configurations q of a C-space Q that are free of obstacles is the *free configuration space* Q_{free} and the *configuration space obstacles* Q_{obst} groups all the configurations occupied by obstacles in the workspace.

$$Q_{free} = Q \setminus Q_{obst}$$

Then, the path planning problem looks for a continuous mapping p in the Q_{free}



a) A4 connectivity b) A8 connectivity

Figure 1: Two discrete motion models.

$$p : [0, 1] \rightarrow Q_{free}$$

where [0, 1] is the parametrization interval, $p(0)$ corresponds to the start configuration q_{start} and $p(1)$ to the goal configuration q_{goal}.

A path planning algorithm is said to be *optimal* when it computes a path that minimizes a set of constrains such as distance, time or energy. A path planner is also called *complete* when the algorithm finds a path from the start configuration to the goal configuration when there is a solution and reports no solution when there is not.

2.2 GRAPH-BASED SEARCH PATH PLANNING

The graph-based search is one of the most popular ways to perform path planning since it is theoretically well-grounded, has been extensively studied and has been applied in many different domains. A graph can be organized as a grid, where the nodes, usually called cells in this configuration, and the edges are distributed regularly. In robotics, grids are widely used to represent environments where the robots have to perform path planning. Every cell of the grid corresponds to a physical region of the scenario. The value of a cell can represent, for instance, whether the cell is occupied by an obstacle or not (see Chapter 4 for further explanation). When the C-space is represented by a Cartesian grid, graph search algorithms commonly use an A4 or A8 connectivity discrete motion model of the robot, see Figure 1.

The Depth-First Search (DFS) and the Breadth-First Search (BFS) are two well-known algorithms to perform graph exploration. The DFS carries out an uninformed search based on the expansion of the first child node, which then directs the search towards deeper nodes. When there are no more children nodes to explore, the search backtracks, returning to the most recent node that has not been explored and repeats the procedure until the goal is reached. On the other hand, the BFS expands and examines all nodes of the graph by systematically searching through every possible solution. The algorithm performs an exhaustive search throughout the entire graph or sequence of nodes without considering the goal until it finds it.

The Wavefront algorithm (Barraquand et al., 1992), also called NF1, is a particular implementation of the BFS that can only be applied to grid environments. The algorithm explores the C-space from the start according to the selected connectivity which follows a *wave* pattern. Once the goal is reached by the wave, the path is computed with a gradient descent technique. The NF1 has been applied in many situations with success. For instance, (Mínguez, 2002) uses it in a sensor-based motion system for a mobile robot to avoid trap situations that could arise using only an obstacle avoidance algorithm. (Parker et al., 2003) implement a dual wavefront path planner in a mobile robot to intercept targets in an indoor environment using information from a distributed acoustic sensor network.

Dijkstra's algorithm (Dijkstra, 1959) is an efficient search algorithm for finding the optimal path in a graph when no other information besides the graph is given. Every time a node is explored, it is updated with the cost to reach the node from the start. Once the goal is found, the least-cost solution can be obtained with backtracking. The A* algorithm (Hart et al., 1968; Nilsson, 1982) uses a BFS to find the least-cost path from an initial node to one goal node. It extends the Dijkstra's algorithm by incorporating a *heuristic* to the cost estimation of paths from each node of the graph to the goal.

2.2.1 *Heuristic Functions Overview*

Path planning algorithms often use heuristic functions in order to estimate the path cost from a specific node to the goal before computing it. This information is commonly used together with the cost of traversing the graph from start to the specific node to set up a preference order when choosing the candidate node to be explored.

A heuristic is called *optimistic* (or *admissible*) when it returns a value that is less or equal to the cost of the shortest path from the current node to the goal. Formally, given a heuristic function for the a node n to the goal node n_{goal}, $h(n, n_{goal})$ and a function $c(n, n_{goal})$ that returns the cost of the shortest path from n to the goal node n_{goal} an optimistic heuristic accomplishes:

$$\forall n,\ h(n, n_{goal}) \leqslant c(n, n_{goal})$$

In this case the heuristic search is efficient. On the other hand, using a *pessimistic* heuristic, a path is also found but it takes more time than probably required and there is the possibility of finding a suboptimal solution. A heuristic function is also *consistent* (or *monotonic*) when the search approaches the solution incrementally without any backtracking, which improves the performance. Formally, for every node n and every successor of n, n', the estimated cost of reaching the goal from n is no less than or equal to the step cost of getting to n' plus the estimated cost of reaching the goal from n':

$$\forall n,\ h(n, n_{goal}) \leqslant c(n, n') + h(n', n_{goal})$$

and

$$h(n_{goal}, n_{goal}) = 0$$

2.2.2 *The A* algorithm*

As stated before, the A* extends the Dijkstra's algorithm by incorporating a heuristic to the cost estimation of paths from each node of the graph to the goal. Each node is ordered according to the sum of its current path cost from the start and a heuristic estimation of its path cost to the goal. The node with the minimum value is evaluated first, since it is the most promising to belong to an optimal path from the start node to the goal node. When applied to path planning, the algorithm uses an optimistic heuristic to ensure that the shortest path is found. Usually, the heuristic function is also consistent to improve the search by ignoring those nodes already explored. However, some recent research (Zhang et al., 2009) points out that using inconsistent heuristics can improve the performance of A* and similar algorithms.

During the computation of the path, the A* generates a search tree which has no cycles. In a bounded world, the number of nodes is limited, which means the number of acyclic paths that can be generated is bounded. In the worst possible case, the A* has to explore all the possible acyclic paths to find the solution, which means the algorithm will always finish. Therefore, the A* is complete.

The A* is written in pseudocode in Algorithm 1. The nodes of the algorithm are configurations of the robot q and processed according to their position in the OPEN priority queue. Each node in this queue is ordered according to the sum of its current path cost from the start, $g(n)$, and a heuristic estimation of its path cost to the goal, $h(n, n_{goal})$. The node with the smallest sum is at the top of the priority queue. Assuming that the heuristic function is consistent, once a node is processed, it is added to the CLOSED set.

The algorithm receives as input the start configuration q_{start} and the goal configuration q_{goal}, which, inside the algorithm, are equivalent to start and goal nodes (n_{start} and n_{goal}) respectively. It starts by adding n_{start} into the OPEN queue. While node n with the minimum $g(n) + h(n, n_{goal})$ is different from the cost to reach the goal node $g(n_{goal})$, the algorithm pops n to the top of the queue and puts it in the CLOSED set. For all the nodes n' reachable from n that are not in the CLOSED set, the algorithm checks whether they are in the OPEN queue or not. If not, n' is added to the priority queue with a priority $g(n')$ plus the heuristic $h(n', q_{goal})$ (line 12). If it is, and the cost $g(n)$ plus the cost of traversing from n to n', $c(n, n')$ is less than its current cost (line 13), $g(n')$ is set to this new lower value. This process is repeated until the n_{goal} is found or OPEN has no more nodes to be expanded.

Figure 2 depicts an example of an A* execution in a grid cell environment using A4 connectivity and Manhattan distance as a heuristic estimator. The neighbor cells are explored clockwise starting with the upper neighbor. Black cells represent obstacles and cannot be traversed. The numbers on each cell represent their priority value in the OPEN queue, which is the sum of the

Algorithm 1 The A* algorithm

$A^*(q_{start}, q_{goal})$

1: $n_{start} \leftarrow q_{start}$; $n_{goal} \leftarrow q_{goal}$
2: $g(n_{start}) \leftarrow 0$; $g(n_{goal}) \leftarrow \infty$
3: OPEN $\leftarrow \emptyset$; CLOSED $\leftarrow \emptyset$
4: OPEN.push(q_{start})
5: **while** $\min_{n \in OPEN}(g(n) + h(n, n_{goal})) \neq g(n_{goal}))$ **do**
6: $n \leftarrow$ OPEN.top()
7: OPEN.pop()
8: CLOSED $\leftarrow$ CLOSED $\cup\, n$
9: **for all** $n' \in Succ(n) \backslash n' \notin$ CLOSED **do**
10: **if** $n' \notin$ OPEN **then**
11: $g(n') \leftarrow g(n) + c(n, n')$
12: OPEN.push(n') with $g(n') + h(n', n_{goal})$
13: **else if** $g(n') > g(n) + c(n, n')$ **then**
14: $g(n') \leftarrow g(n) + c(n, n')$
15: **end if**
16: **end for**
17: **end while**
18: **if** $\min_{n \in OPEN}(g(n) + h(n, n_{goal})) = g(n_{goal}))$ **then**
19: publish solution
20: **end if**

cost to reach the cell from the start node and its heuristic value toward the goal. Black dots represent nodes already considered. Those nodes with priority values that do not have a dot depicted are stored in the OPEN queue. Figures 2.a,b,c depict different moments during the execution. Figure 2.d shows the final path obtained with backtracking from the goal to the start.

2.2.3 *Replanning Algorithms*

In real world applications, robots do not have complete knowledge of the environment and usually carry onboard sensors that provide updated information while navigating. Therefore, the graph that models the environment may change as new information arrives, which can make a computed solution suboptimal or unreachable.

In order to avoid recomputing the solution from scratch, the D* (Stentz, 1995) and D* Lite (Koening and Lickhachev, 2002) algorithms extend the A* by taking into account the changes in the environment in order to update the computed solution. In robot path planning, these algorithms are used in grids to compute discrete paths which are unnatural and constrain the robot's movement. (Ferguson and Stentz, 2005) propose the Field D* algorithm to generate a smooth solution which not only uses the center of the cells for generating the paths, but the edges as well. The E* algorithm, developed by (Philippsen and Siegwart, 2005), computes continuous cost paths by using a Fast Marching (FM) method (Sethian, 1996), which generates a set of continuous propagation curves from the start that are traversed by a gradient descent method. (Dolgov et al., 2010) propose the Hybrid-state A* to perform path

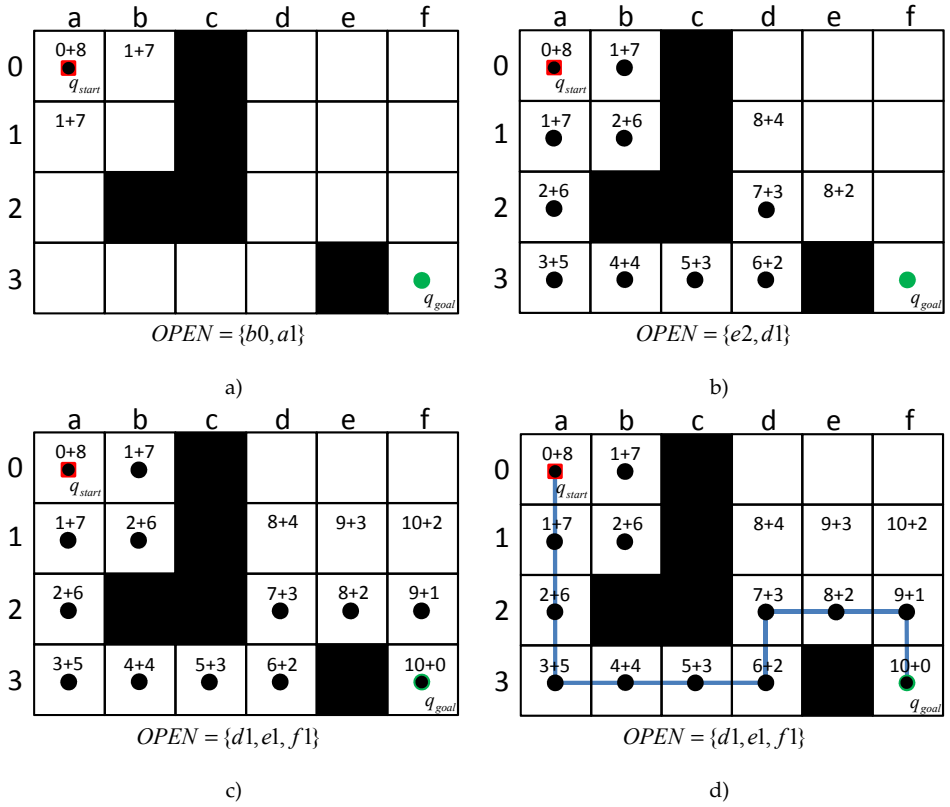
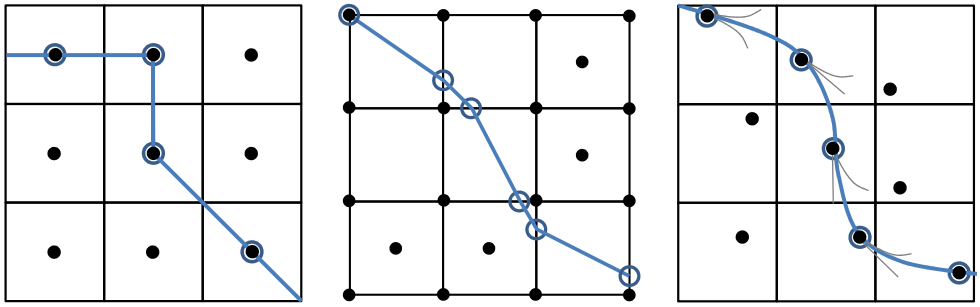


Figure 2: A* execution example. a), b) and c) depict the environment exploration at three different moments. d) Shows the shortest path computed with backtracking from the goal once it has been found.



a) A* b) Field-D c) Hybrid-A*

Figure 3: Graphical comparison of graph-based search algorithms. A* (A8 connectivity) only assumes the cost of the centre of the discrete cells. Field-D associates costs with cell corners and allows linear paths between cells. Hybrid-A* associates a continuous state with each cell. Image extracted from (Dolgov et al., 2010).

planning for autonomous vehicles in unknown semi-structured environments represented by grid cells. Essentially, it is an extension of the A*, similar to the Field D*, that expands every state with several continuous steering actions computed by a kinematic model. Although its global optimality is not ensured, the algorithm generates continuous paths that can be achieved by the vehicles. Figure 3 depicts a graphical comparison of the solutions obtained with the A*, Field D* and Hybrid A* algorithms.

2.3 PROBABILISTIC SAMPLE-BASED PATH PLANNING

Probabilistic sampling-based path planners include multiple-query and single-query methods. Both strategies generate samples of the free space which connect the start to the goal by means of a graph or a tree data structure. Every sample is a node that represents a configuration of the robot. These nodes are connected with edges which represent free paths through which the robot can navigate.

Probabilistic RoadMap (PRM) methods (Kavraki et al., 1996; Svestka and Overmars, 1998) were originally presented as multiple-query planners: the goal was to create a roadmap that captures the connectivity of the free space and then answer multiple user-defined queries very quickly. Although the connectivity of the free space is usually performed before any path planning query, both phases can be done at the same time.

On the other hand, single-query path planners attempt to solve a query as fast as possible and do not focus on the exploration of the entire free space. The Expansive-Space Tree (EST) (Hsu, 2000; Hsu et al., 2002) and the Rapidly-exploring Random Tree (RRT) (LaValle, 2006) build a tree rooted at q_{start} that grows towards q_{goal}. The EST generates new samples in the free space around a configuration selected in a low density sampled area, whereas the RRT attempts to expand the sampling into the regions of the free space away from the tree. In both algorithms, the growing of the tree is highly dependant

on the sampling method. For geometric problems, it is possible to bias the configuration sampling by using a bi-directional strategy, which is maintaining two trees, rooted at q_{start} and q_{goal} respectively, and making them grow toward each other until they are merged into a single one. Another single-query path planner is the Single-query Bi-directional Lazy collision-checking (SBL) (Sanchez and Latombe, 2002), a bi-directional EST that improves the performance of the original bi-directional EST by using a lazy evaluation in the node connection strategy which performs the collision checking between nodes only when it is absolutely necessary.

2.3.1 *The Rapidly-exploring Random Tree Approach*

The RRT (LaValle, 2006) has become the most popular single-query planner. It has been shown to be effective for solving path planning tasks in many kinds of problems (Kim and Ostrowski, 2003; Alcázar et al., 2011), even with a lot of DoFs (Kuffner and LaValle, 2000; Yang and Sacks, 2006). Algorithm 2 shows the pseudocode of the RRT. It begins with the initial robot configuration q_{start} as the root node and incrementally grows a tree T until the goal configuration q_{goal} is reached. To grow the tree, first a target configuration q_{rand} is randomly selected from the C-space using the function RandomConfiguration. Then, a NearestNeighbor (line 2) function selects the node $q_{nearest}$ in the tree closest to q_{rand}. Finally, a new node q_{new} is created by growing the tree some distance from $q_{nearest}$ towards q_{rand}. If extending the tree towards q_{rand} requires growing through an obstacle, no extension occurs. This process is repeated until the tree grows to within some distThreshold from the goal (line 7). Figure 4 depicts this process and Figure 5.a depicts an RRT in a simple scenario. A property that follows from this method of construction is that the tree growth is strongly biased towards unexplored areas of the C-space. Consequently, exploration occurs very quickly.

Algorithm 2 Rapidly-exploring Random Tree

Extend(T, q_{goal})

1: $q_{rand} \leftarrow$ RandomConfiguration()
2: $q_{nearest} \leftarrow$ NearestNeighbor(T, q_{rand})
3: $q_{new} \leftarrow$ ComputeQNew$(q_{nearest}, q_{rand})$
4: T.Add(q_{new})
5: **return** q_{new}

RRT(q_{start}, q_{goal})

5: $q_{new} \leftarrow q_{start}$
6: T.Add(q_{new})
7: **while** Distance$(q_{new}, q_{goal}) >$ distThreshold **do**
8: $\quad q_{new} \leftarrow$ Extend(T, q_{goal})
9: **end while**

Although RRTs offer very good performance, it is a common practice to use a biased sampling towards the goal which makes a certain percentage of random samples coincident with the goal, making the tree expand in that direction

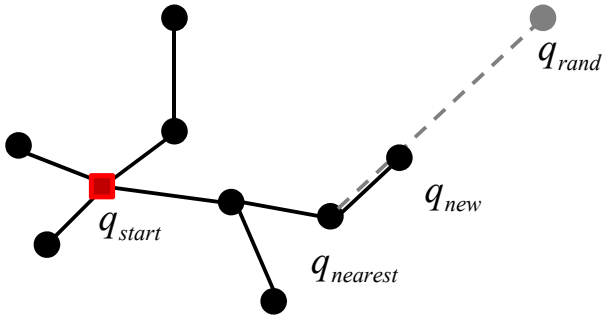


Figure 4: The RRT uses the q_{start} as the root node of a tree until the goal configuration q_{goal} is reached. At each step, a configuration q_{rand} is selected using a random sampling distribution. Then, the nearest node in the tree to q_{rand} is labeled as $q_{nearest}$. Finally, a new node q_{new} is added to the tree at a certain distance from $q_{nearest}$ towards q_{rand}.

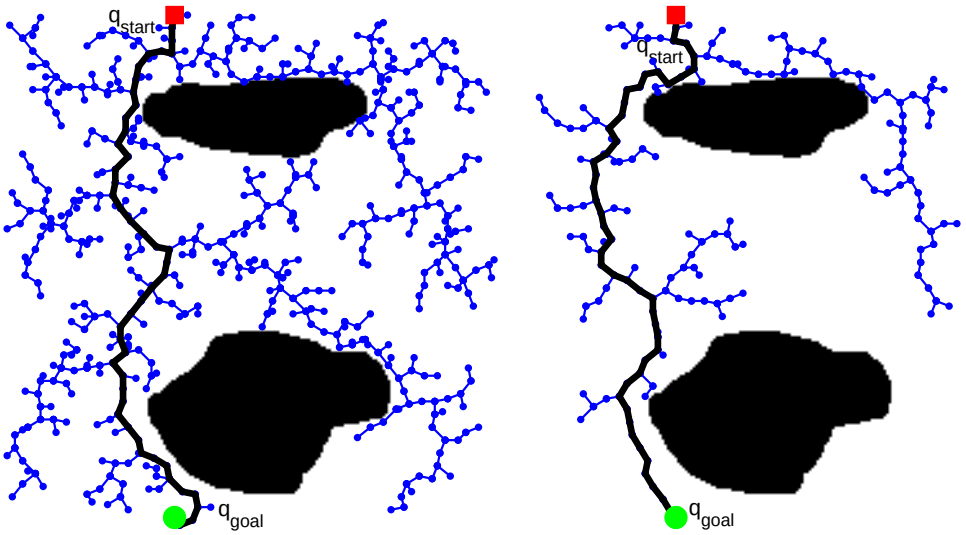
(Frazzoli et al., 2000; Hsu et al., 2003; Kuwata et al., 2008) (see Figure 5.b). (Branicky et al., 2001; Lacevic and Rocco, 2010; Thomas and Iser, 2010) improve the original RRT by using a quasi-random sampling distribution such as a Halton sequence or a Hammersley sequence since they offer lower discrepancy[1] than random distributions, which means that the samples perform a coverage of the space more uniformly.

There are other strategies to improve RRTs. For instance, (Kuwata et al., 2008; Borrajo and Veloso, 2012) incorporate a heuristic estimator in the building tree process in order to obtain better solutions at the expense of reducing the performance. As stated before, it is possible to use a bi-directional strategy (Kuffner and LaValle, 2000) which generates two trees, one from the start and other from the goal. Both explore the respective space around them and advance towards each other by using a simple greedy heuristic. The path is found when both trees are merged after a collision between them is detected. (Ferguson et al., 2006) present a replanning algorithm for repairing RRTs when changes are made on the C-space. In such a situation, those parts of the search tree that become invalid are pruned and the remaining tree is then grown to the goal configuration again.

Sampling-based path planners are *probabilistically complete* (LaValle, 2001), because the probability of finding a solution increases as more samples are used. However, in practical situations, these methods may not find a solution in a finite time. A narrow passage is a typical problem where sampling-based path planners can get stuck. In this situation, a common practice is to fix the number of times the tree can be expanded. If then no solution has been found, it is assumed the goal is not reachable.

Although many authors incorporate kinematic and dynamic constraints in the C-space (Kuwata et al., 2008; Shkolnik et al., 2009) to generate feasible paths and trajectories for real robots, sampling-based path planners do not compute optimal solutions. Thus, a common practice is to post-process the path in order to improve the solution (Kuffner and LaValle, 2000; Thomas

1 Discrepancy: A measure of how uniformly points are distributed over the space.



a) RRT b) Goal-biased RRT

Figure 5: Example of an RRT execution in a simple environment: a) Exploration tree with no biased sampling; b) Exploration tree making 5% of the samples coincident with the goal.

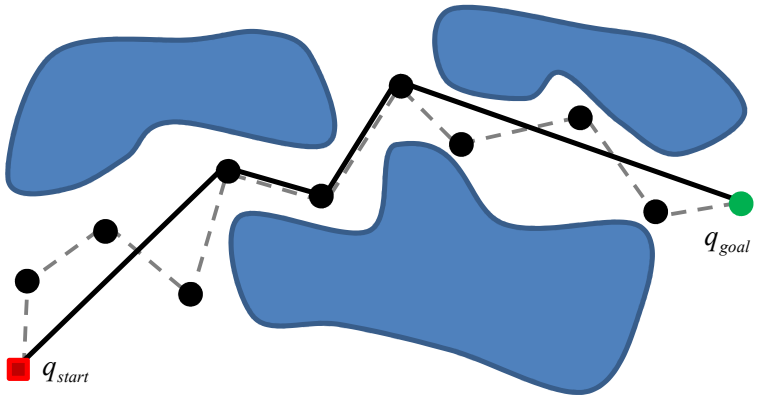


Figure 6: Improving a sampled-based path with the greedy approach.

and Iser, 2010). For instance, a greedy approach starts by trying to connect q_{start} directly with q_{goal}. If this step fails, it starts from the configuration after q_{start} and tries again. This process is repeated until a configuration q can be connected to q_{goal}. Next, the target is set to q and the process begins again. Figure 6 illustrates this process, which can also be applied backwards. Notice that using postprocessing steps to improve the path can impose a significant overhead on the absolute computation time of the path.

2.4 BUG-BASED PATH PLANNING

Bug-based algorithms were one of the first planners developed. Although they are reactive algorithms designed for online robot navigation that can be applied to robots with low computational capabilities, it is possible to use

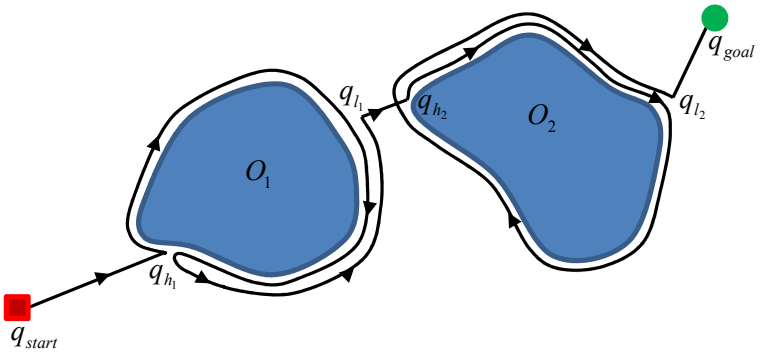


Figure 7: An example of the execution of the Bug1 algorithm in a simple scenario.

them to perform deliberative path planning on a C-space, which is the main goal of this dissertation.

Bug-based path planners assume the robot is a point with a contact sensor or a zero-range sensor to detect obstacles. Essentially, the Bug-like algorithms are compounded of two different behaviors: go straight and follow the obstacle boundary. Bug1 and Bug2 (Lumelsky and Stepanov, 1987) are two well known path planners that have been used in many applications (Chien and Xue, 1992; Sarid et al., 2007; Mastrogiovanni et al., 2009; Zhu et al., 2010). These path planning algorithms try to go from the start configuration q_{start} to the goal configuration q_{goal} in a straight line until they find an obstacle. In such case, they surround the boundary of the obstacle until a condition is satisfied.

The Bug1 represents the most basic idea when looking for the goal while surrounding obstacles. The algorithm tries to follow an imaginary line, called the *m-line*, which connects the q_{start} with the q_{goal} until an obstacle is found. The contact point with the obstacle is marked as a *hit point* q_h. Then, the robot completely surrounds the obstacle reaching the q_h again, which allows the computation of the *leave point* q_l. That is the closest point of the obstacle's boundary coinciding with the new *m-line* which goes from q_l to q_{goal} and does not intersect with the surrounded obstacle. Next, the robot has to reach the *leave point* in order to leave the obstacle. This procedure is repeated until q_{goal} is reached. Figure 7 depicts an example of the generated path in a simple scenario.

In a worst case scenario, the robot has to navigate half of the perimeter of the current ith obstacle p_i. Assuming this situation can arise for all the n obstacles of the environment encountered, the upper bound of a Bug1 is defined as

$$UB_{Bug1} \leqslant dist(q_{start}, q_{goal}) + 1.5 \sum_{i=1}^{n} p_i \tag{2.1}$$

where $dist(q_{start}, q_{goal})$ is de distance between the q_{start} and q_{goal}.

The Bug2 algorithm tries to improve the Bug1 by fixing the *m-line* from q_{start} to q_{goal} at the beginning. The robot starts by following the *m-line* until the first *hit point* q_{h_1} is reached. Then, the boundary following behavior surrounds the obstacle until it reaches a point in the *m-line* which is closer

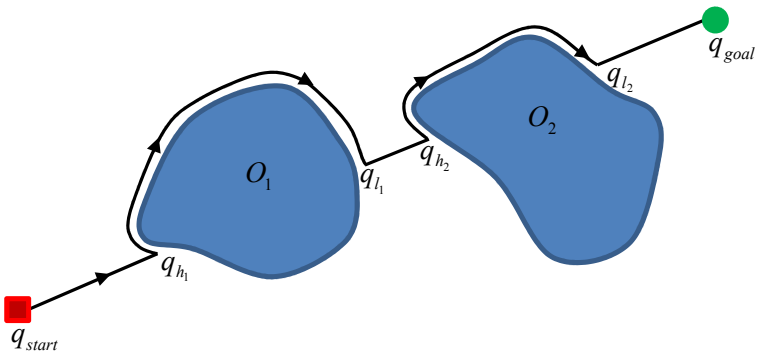


Figure 8: Example of execution of Bug2 algorithm in a simple scenario.

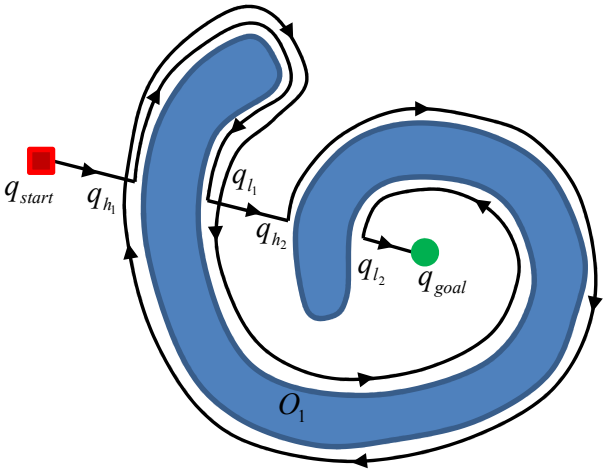


Figure 9: Bug2 path in a scenario with a complex obstacle.

to the goal than the previous *hit point*, thus becoming the first *leave point* q_{l_1}. The same process is repeated until q_{goal} is found. Figure 8 depicts a path generated by the Bug2 algorithm. Note that the direction to circumnavigate the obstacles can be fixed at the beginning of the execution or chosen randomly.

Although the Bug2 algorithm is designed to avoid the complete circumnavigation of the obstacle, depending on the obstacle's shape, this is not always the case. Figure 9 shows the path generated by the Bug2 algorithm in a scenario with a complex obstacle. The Bug2 *m-line* intersects with O_1 obstacle several times. Assuming that the circumnavigation direction does not change for the whole obstacle, the algorithm generates a longer solution than the one computed with the Bug1.

Formally, the ith obstacle is intersected by the *m-line* n_i times, which means there are n_i *leave points*. However, half of these points are not valid since they lie on the side of the obstacle that moving towards the goal would generate a collision. As reported in (Choset et al., 2005), in the worst case, the robot

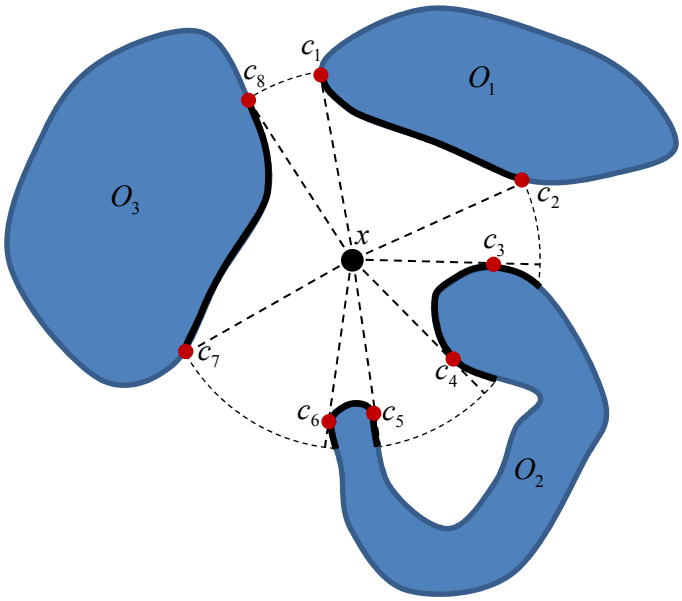


Figure 10: Interval of continuities found by the range sensor at a fixed position.

would traverse nearly the entire perimeter of the obstacle for each leave point. Therefore, the upper bound of the algorithm is

$$UB_{Bug2} \leqslant dist(q_{start}, q_{goal}) + 0.5 \sum_{i=1}^{n} n_i p_i \tag{2.2}$$

In order to improve the Bug2 algorithm, (Kamon et al., 1998) developed the Tangent Bug. This algorithm supposes the robot has a finite range sensor used to detect the obstacles in the workspace. It is assumed that the sensor is able to detect the continuous boundary of the obstacles within its range, which are called *intervals of continuity*. As depicted in Figure 10, every time that a discontinuity appears in the boundaries gathered by the sensor, an endpoint c_i tangent to the obstacle arises.

Figure 11 depicts an execution of the Tangent Bug. Like the other Bug algorithms, the robot initially goes in a straight line towards the goal until hit point q_{h_1}, where it senses the first obstacle O_1 in the goal direction. At that point, an interval of continuity limited by two endpoints $C = \{c_1, c_2\}$ is created. Notice that the interval of continuity intersects with the segment between the robot and the goal. Then the robot moves to the right since c_2 minimizes a heuristic distance such as $dist(x, c_i) + dist(c_i, q_{goal})$, where x represents the robot's position. At q_{l_1} the updated interval of continuity $C = \{c_{1'}, c_{2'}\}$ no longer intersects with the line from the robot to the goal, therefore, the robot moves towards the goal following a straight line. At q_{h_2} obstacle O_2 is detected and the robot moves to the left since the heuristics decreases until q_{m_2}, where point m_2 that minimizes the heuristic distance is located. At this point, the algorithm follows the boundary of the obstacle until its next leave point q_{l_2}, which is set when the distance from $c_{3''}$ to q_{goal} is smaller than

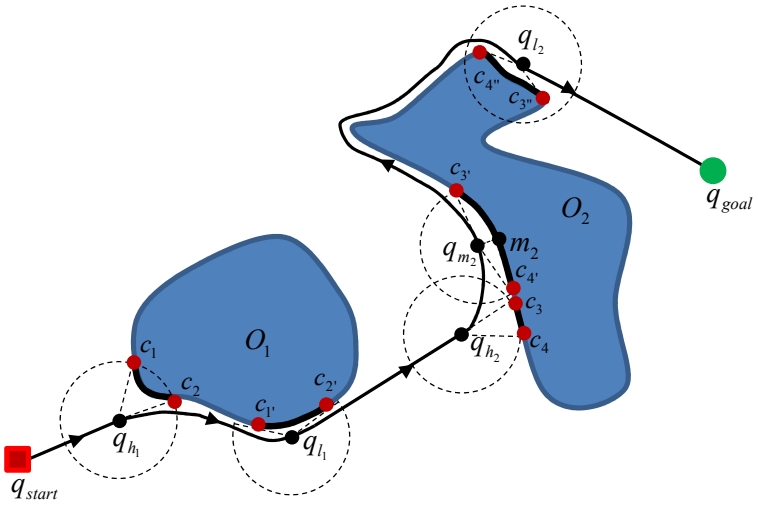
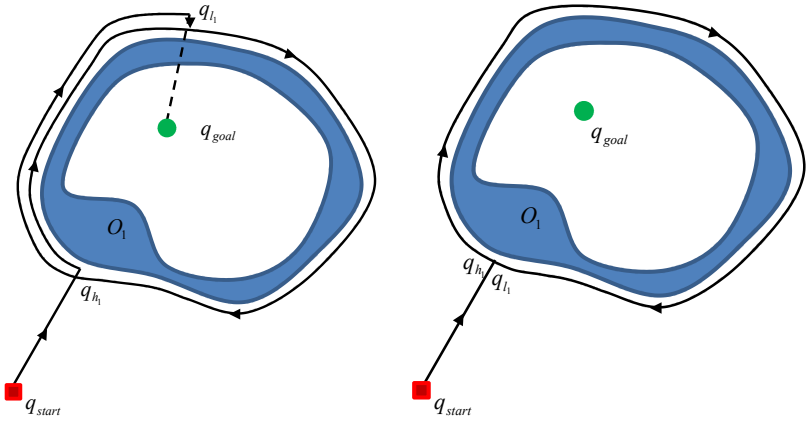


Figure 11: A Tangent Bug algorithm execution.



a) Bug1 algorithm execution b) Bug2 algorithm execution

Figure 12: Completeness of Bug1 and Bug2 algorithms.

the distance between m_2 and q_{goal}. Finally, the robot moves straight towards q_{goal}.

Bug algorithms are complete regardless of the strategy followed because they find a path from the start to the goal when there is a solution, and report when there is not. For instance, Bug1 reports that there is no solution after the robot reaches the leave point and intersects with an obstacle when trying to navigate towards the goal (Figure 12.a). On the other hand, Bug2 reports that there is no solution when it circumnavigates the entire obstacle, which means that a leave point has not been found (Figure 12.b). The Tangent Bug also navigates through the whole obstacle perimeter before reporting there is no solution.

There are many recent Bug algorithms that try to improve the performance or the quality of the solution or both (Ng and Braunl, 2007). For instance, (Lumelsky and Skewis, 1990) propose an improvement of the Bug2 incorporating range sensor information. The algorithm, called VisBug, enhances the

condition that the robot uses to stop surrounding an obstacle and resumes the movement to the goal, which results in short cuts in the path. A similar idea is used with the DistBug (Kamon and Rivlin, 1997), which proposes another alteration in the leaving condition that allows the robot to abandon obstacle boundaries as soon as global convergence is guaranteed. The K-Bug (Langer et al., 2007) changes the paradigm of contouring all the obstacles towards the goal to the idea of contouring only the essential ones to reach the destiny point. (Antich and Ortiz, 2009) propose the Bug2+, an improved version of the Bug2 algorithm, which only takes into account the valid leave points when the *m-line* intersects a number of times with the same obstacle. However, to the best of the author's knowledge, none of the Bug-based algorithms are optimal since following the boundary of the obstacles encountered does not generate the shortest path to the goal.

2.5 ANYTIME PATH PLANNING

In real world applications, robots have to perform path planning in a limited amount of time with the computers they carry onboard. Algorithms that look for the optimal solution are not usually feasible due to their computation time. Instead, the path planner has to find the best solution within a limited time. Therefore, anytime path planners (Ferguson et al., 2005; Likhachev et al., 2008) are the most suitable as they find a first solution, possibly highly suboptimal, very quickly and then refine it until time runs out.

Most of the anytime path planning literature is about deterministic approaches. They all share the strategy of using a multiplication factor to inflate the heuristics (Likhachev et al., 2004), which is usually referred to as weighted heuristic search. As a result, the nodes of the C-space to be explored first are those which are closer to the goal (Ferguson and Stentz, 2007), generating solutions very quickly at expense of their quality. On the other hand, there are also some probabilistic anytime proposals, the vast majority of them based on RRTs (Ferguson and Stentz, 2006). These algorithms generate an initial RRT solution and, within the time allotted, a new RRT that improves the previous one is computed. Essentially, this new solution is generated by an enhanced sampling strategy and using a heuristics that only lets the tree expand into those areas of the C-space that would possibly improve the solution. Further details on deterministic and probabilistic anytime approaches are given in the next sections.

Although it is not common, there is also an anytime approach inspired by bug algorithms (Antich et al., 2009). This algorithm assumes that the scenario is known. Every time the path that follows the *m-line* hits an obstacle, an improved version of the Bug2 computes left and right boundaries of the obstacle until the *m-line* is reached again. Each hit point of the path with the obstacles is stored in the nodes of a binary search tree which is explored using the A* algorithm with an inflated heuristic to speed up the path search. Once computed, the path is post processed in order to obtain the optimal solution while maintaining the manner in which the path avoids the obstacles.

2.5.1 *The Deterministic Anytime Approach*

As stated before, most of the deterministic anytime path planning algorithms use a weighted heuristic search to speed up the path search (Ferguson and Stentz, 2007). The weight value ϵ ($\epsilon \geqslant 1$) is a multiplication factor used to control the cost of the generated solution. Some algorithms compute different solutions using the same ϵ (Hansen et al., 1997; Zhou and Hansen, 2002; Hansen and Zhou, 2007). Other approaches use a decrement factor in order to decrease the ϵ value at each iteration until $\epsilon = 1$, if the time does not expire before (Likhachev et al., 2005). For instance, the Anytime Repairing A* (ARA*) proposed by (Likhachev et al., 2004) is a reference within the class of deterministic/heuristic based anytime path planners. It inflates the heuristic function using the ϵ value to guide the search towards those states closer to the solution whose final cost is ensured not to be more than ϵ times the cost of the optimal path (Pearl, 1984).

Algorithm 3 shows a simplified version of the ARA*. This algorithm starts by doing an A* search (line 24) with an inflation factor ϵ expanding each node just once. As in the A*, the nodes in OPEN are processed according to their priority, which in this case is the sum of its cost $g(n)$ and its inflated heuristic value $\epsilon \cdot h(n, n_{goal})$. CLOSED contains the nodes already expanded in the current search. Once expanded, if a node becomes inconsistent during the search ($g(n) \neq min_{n' \in Pred(n)}(g(n') + c(n', n))$) due to a cost change associated with a neighboring node, it is inserted into the INCONS list (line 14), which contains all inconsistent nodes already expanded. When the current path search is finished, the nodes in the INCONS list are moved into the OPEN queue with the priorities updated according to the new inflation factor (lines 28– 29) which is going to be used in the next search.

Figure 13 shows three consecutive iterations of an ARA* execution in a simple grid scenario. In this example, an A8 connectivity grid is used with black cells representing obstacles. The cost of moving from one cell to its neighbor is one. The heuristic is the largest of the x and y distances from the cell to the goal. Expanded cells are shown in grey and those which are inconsistent at the end of each iteration are shown with an asterisk. The initial solution is computed with an inflation factor of 2.5 expanding 13 cells. The next iteration computes the solution using $\epsilon = 1.5$ and requires only one cell to be expand. Finally, the third iteration generates the path using $\epsilon = 1.5$ which computes the optimal solution by expanding 9 cells. The total number of cells expanded in this example are 22.

Although at each iteration the solution is intended to be improved by decreasing ϵ, the generation of a new/better path is not ensured (the same path as in the previous iteration of the algorithm can be obtained), which means a waste of computation time in a critical context where the time to perform the path planning is very limited. The algorithm also proposes the reutilization of the inconsistent states between iterations, updating only the heuristic values of these states with the new inflation factor, which can only be used if the current path search passes through the previously visited states.

Algorithm 3 The Anytime Repairing A*

Key(n)

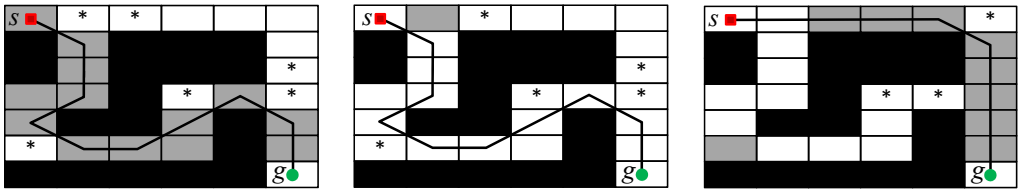
1: **return** $g(n) + \epsilon \cdot h(n, n_{goal})$

ImprovePath$()$

2: **while** $\min_{n \in OPEN}(Key(n)) < Key(n_{goal})$ **do**
3: $n \leftarrow$ OPEN.top(); OPEN.pop()
4: CLOSED $\leftarrow$ CLOSED $\cup\, n$
5: **for all** $n' \in Succ(n)$ **do**
6: **if** n' was not visited **then**
7: $g(n') \leftarrow \infty$
8: **end if**
9: **if** $g(n') > g(n) + c(n, n')$ **then**
10: $g(n') \leftarrow g(n) + c(n, n')$
11: **if** $n' \in$ CLOSED **then**
12: OPEN.push(n') with $Key(n')$
13: **else**
14: INCOS $\leftarrow$ INCONS $\cup\, n'$
15: **end if**
16: **end if**
17: **end for**
18: **end while**

ARA*(q_{start}, q_{goal})

19: OPEN $\leftarrow \emptyset$; CLOSED $\leftarrow \emptyset$; INCONS $\leftarrow \emptyset$
20: $n_{start} \leftarrow q_{start}$; $n_{goal} \leftarrow q_{goal}$
21: $g(n_{start}) \leftarrow 0$; $g(n_{goal}) \leftarrow \infty$
22: OPEN.push(n_{start})
23: **repeat**
24: ImprovePath()
25: publish solution
26: decrease ϵ
27: **if** $\epsilon > 1$ **then**
28: OPEN $\leftarrow$ INCONS; INCONS $\leftarrow \emptyset$
29: $\forall n \in$ OPEN update priorities with $Key(n)$
30: CLOSED $\leftarrow \emptyset$
31: **end if**
32: **until** $\epsilon > 1$



a) $\epsilon = 2.5$ b) $\epsilon = 1.5$ c) $\epsilon = 1$

Figure 13: An ARA* search. Images extracted from (Likhachev et al., 2004)

Otherwise, no reutilization can be done, which means that in some scenarios the total number of states explored during the execution can be larger than using the standard A*. Generally, ϵ and the decrement factor values are critical depending on the complexity of the scenario. However, authors do not provide a general rule to set them, hence the setting of these parameters strongly depends on the expertise of the user.

There are anytime approaches that instead of increasing the weight of the heuristics, select a number of the most promising nodes during the search for further expansion while the remaining nodes are pruned (Zhou and Hansen, 2005; Furcy, 2006; Aine et al., 2007). Other algorithms, such as Anytime Dynamic A* (AD*) (Likhachev et al., 2005) combine anytime performance with replanning capabilities.

2.5.2 *The Probabilistic Anytime Approach*

There are some probabilistic anytime path planners. For instance, (Belghith et al., 2006) is based on the PRM approach. This algorithm applies the AD* to the PRMs generated with an improved sample strategy. However, most of these approaches are based on RRTs since they offer very good performance. The Anytime-RRT (ARRT) developed by (Ferguson and Stentz, 2006) is a well-known anytime sampling-based planner that works by generating a series of RRTs where each new tree reuses the cost information from the previous tree to control its growth and thus improve the quality of the resultant path. Unlike the ARA*, the ARRT scarcely reutilizes data between iterations at all because each new tree is almost built from scratch.

Algorithm 4 shows a simplified version of the ARRT. Essentially, the algorithm computes a set of solutions based on RRTs while the time does not expire. Like the RRT, each solution is obtained by expanding a tree T (with the function ExpandRRT) until the tree grows to within some distThreshold. At each iteration, the cost of the tree is ensured to be less than or equal to $(1 - \epsilon_f)$ times the previous cost of the tree, where $\epsilon_f \in [0..1)$ is a fixed improvement factor (line 27),

The extension of the tree starts with function ComputeQRand which generates a sample q_{rand}. This function improves the sampling process by introducing a bias towards the goal (line 1) and constraining it into those areas of the C-space where the heuristics from q_{start} to q_{goal} through q_{rand} are lower than an upper bound (line 6). Then, the kNearestNeigbor function computes the k nearest nodes in tree T to the sample q_{rand} and sorts them according to their cost in order to obtain the cheaper solutions (Urmson and Simmons, 2003). Next, the first q_{near} node from Q_{near} is selected to compute a set of extensions (line 11). The one with the lowest cost is chosen as the q_{new}. If the cost of the tree from q_{start} to q_{new} through the selected q_{near} (line 13) is lower than the upper bound (line 14), the extension of the tree is performed and the function returns. Otherwise, the process is repeated for the next q_{near}. Figure 14 depicts a simple example of an execution.

Algorithm 4 The Anytime RRT

ComputeQRand(q_{goal})

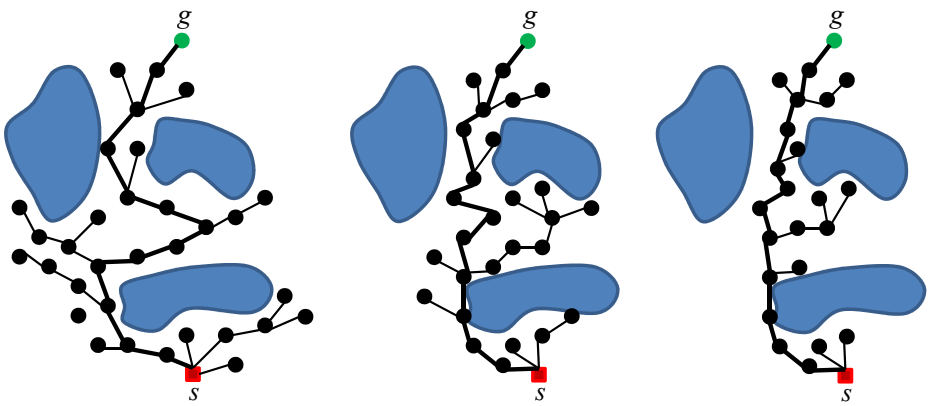
1: **if** RandomReal($[0.0, 1.0]$) $<$ probThreshold **then**
2: **return** q_{goal}
3: **end if**
4: **repeat**
5: $q_{rand} \leftarrow$ RandomConfiguration()
6: **until** $h(q_{start}, q_{new}) + h(q_{new}, q_{goal}) \leqslant T.c_s$
7: **return** q_{rand}

ExtendRRT(T, q_{goal})

8: $q_{rand} \leftarrow$ ComputeQRand()
9: $Q_{near} \leftarrow$ kNearestNeighbor(T, q_{rand}, k)
10: **for all** $q_{near} \in Q_{near}$ **do**
11: $Q_{ext} \leftarrow$ GenerateExtension(q_{near}, q_{rand})
12: $q_{new} \leftarrow \operatorname{argmin}_{q \in Q_{ext}} c(q_{near}, q)$
13: $T.c_{new} \leftarrow c(q_{start}, q_{near}) + c(q_{near}, q_{new})$
14: **if** $T.c_{new} + h(q_{new}, q_{goal}) < T.c_s$ **then**
15: T.add(q_{new})
16: **return** $\{q_{new}, T.c_{new}\}$
17: **end if**
18: **end for**

ARRT(q_{start}, q_{goal})

19: **while** there is time **do**
20: $T \leftarrow \emptyset$; $T.c_{new} \leftarrow \infty$
21: $q_{new} \leftarrow q_{start}$
22: T.Add(q_{new})
23: **while** Distance(q_{new}, q_{goal}) $>$ distThreshold **do**
24: $\{q_{new}, T.c_{new}\} \leftarrow$ ExtendRRT(T, q_{goal})
25: **end while**
26: publish solution
27: $T.c_s \leftarrow (1 - \epsilon_f) \cdot T.c_{new}$
28: **end while**



a) Initial RRT solution b) *k*-iteration c) Final solution

Figure 14: An ARRT example.

The Anytime Dynamic RRT (ADRRT) (Ferguson and Stentz, 2007) improves the ARRT by adding repairing capabilities. This algorithm generates a solution that is continually improved while the time does not expire, as ARRT does, and, at the same time, repairs the solution when changes that invalidate it are detected in the C-space. Some other proposals, like (Abbasi-Yadkori et al., 2010; Karaman et al., 2011), improve the ARRT with RRT-based algorithms that apart from generating initial solutions very quickly and improving them continuously while there is time, they also compute final solutions close to the optimal.

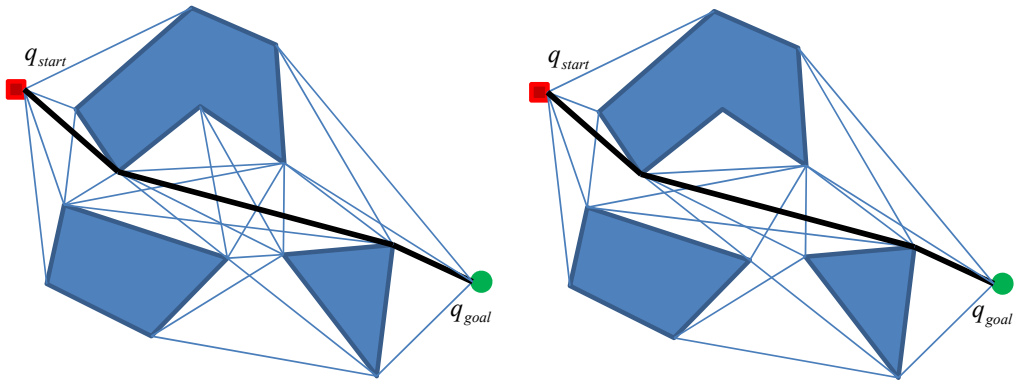
2.6 TOPOLOGICAL PATH PLANNING

Topological approaches are another way to tackle the path planning problem. This kind of solution works with an abstraction of the C-space, a topological-based map usually represented by a graph. These graphs represent the structure of the free space in the C-space in terms of the basic topological notions of connectivity and adjacency (Fabrizi and Saffiotti, 2000), which represents the environment with a reduced number of potential states (Dudek et al., 1991).

Topological-based maps are quite robust with respect to sensor noise and small environmental changes. Moreover, graph representation yields the possibility of exploring them with graph-search algorithms. Visibility maps (Latombe, 1991) is a well-known topological-based technique widely used in robotics. The simplest visibility map, called *visibility graph*, consists of a set of nodes in the free C-space that are within the line of sight of at least one node of the visibility graph. Although it has been generalized to be used with curved obstacles (Liu and Arimoto, 1992), the original version was intended to work in polygonal C-spaces, where each edge of the obstacles is a node in the graph. Assuming that the start and goal configurations are also nodes in the graph, the visibility graph is built by trying to connect each node with all the other nodes with a straight line. The connection is performed only when the line does not intersect with an obstacle. Once the graph is generated, it is traversed with a graph-based search algorithm to compute the shortest path according to the graph (Hans and Rohnert, 1986). Figure 15.a depicts an example.

In order to improve the search in the shortest-path problem, it is a common practice to work with a *reduced visibility graph* which performs the connection between two nodes only when the line that connects them is tangent to the two obstacles they belong to (see Figure 15.b).

Using the *Voronoi diagram* to perform topological path planning is another technique widely extended (Takahashi and Schilling, 1989; Acar et al., 2006). Essentially, the *Voronoi diagram* is defined for a set of points called *sites* (Aurenhammer, 1991). Then, the Voronoi diagram is a set of points equidistant to the two closest sites at the same time. Figure 16 depicts an example of a Voronoi diagram for a low number of sites. When applied to path planning, the sites are the centre of the obstacles to be avoided and the edges of the diagram define the possible channels that maximize the distance to the obstacles. When



a) Visibility grah b) Reduced visibility graph

Figure 15: Example of computing the shortest path using a visibility graph.

applied to motion planning, it is a common practice to use the Generalized Voronoi Diagram (GVD) since it assumes than obstacles are more complex than single points, allowing the computation of a diagram which is closer to reality (Bhattacharya and Gavrilova, 2007). Nevertheless, both options allow the computation of safe paths according to the distance to obstacles.

Another way to perform path planning with topological information is by means of homotopy classes, which are detailed in the following section.

2.7 HOMOTOPY CLASSES

Homotopy is a topological concept related with path deformation with respect to a set of obstacles (Munkres, 1974; Seifert et al., 1980). A *homotopy class* is the set of all possible trajectories from a start to an end point that avoid obstacles in the same manner. Given two paths, they are said to be *homotopic* if one can be deformed into the other without encroaching any obstacle. Figure 17.a shows an example where p_1 and p_2 are homotopic and hence, belong to the same homotopy class. Figure 17.b depicts two paths that do not belong to the same homotopy class since they encroach on O_1 when p_1 is deformed to be coincident with p_2.

Path homotopy is an equivalence relation, which means that it is possible to identify a homotopy class by giving a representative path, whatever the definition chosen to represent a path is. It is common practice to use the *canonical representation* to describe a homotopy class since it is the simplest representation of a class without changing its topology. Moreover, two paths belong the same homotopy class if they have the same canonical sequence (Cabello et al., 2002).

The homotopy concept has applications in many different areas such as VLSI routing (Leiserson and Maley, 1985; Gao et al., 1988; Yang, 1997) or image analysis (Kalitzin et al., 2001). In robotics, homotopy classes are intrinsically taken into account; for instance, into the map discovering problem with multiple vehicles (Williams et al., 2002), for surveillance purposes (Tang and Ozguner, 2005) or for locally optimal navigation in unknown scenarios (Tovar et al.,

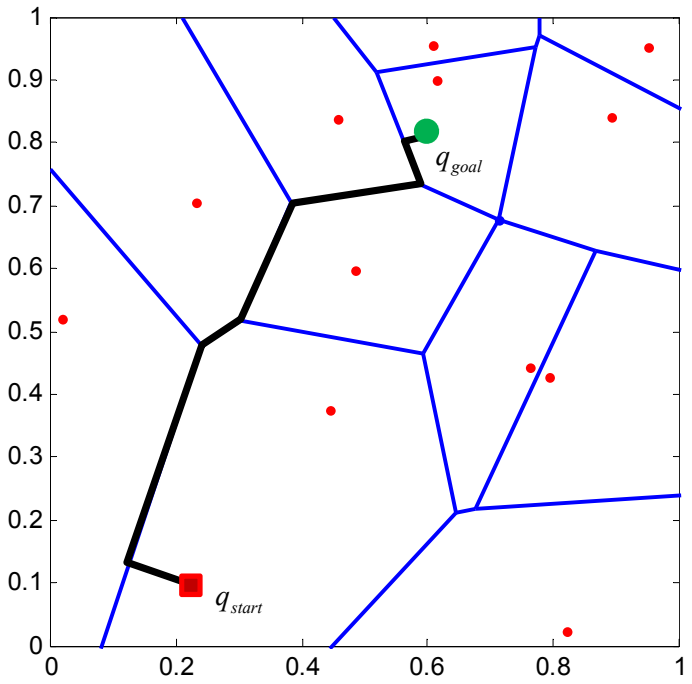
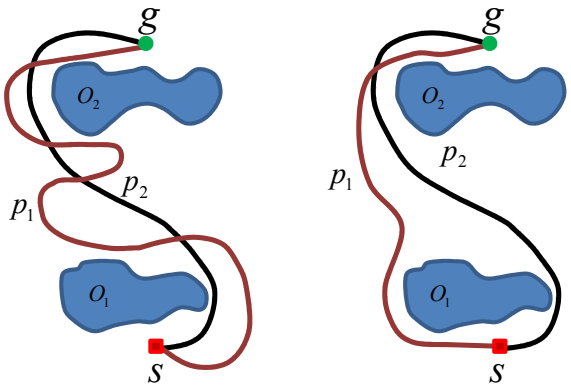


Figure 16: Computing the shortest path using a Voronoi diagram in a scenario with 12 sites.



a) p_1 and p_2 are homotopic b) p_1 and p_2 are not homotopic

Figure 17: Example of homotopic paths.

2007). Focusing on path planning, (Antich et al., 2009) use homotopy classes to classify all the different paths that can be generated for a scenario to ensure that their topology is not affected after an optimization process. Along the same line, (Laumond et al., 1994) use homotopy classes to maintain the topology when optimizing trajectories for a non-holonomic mobile robot. (Asama et al., 1991; Bourgault et al., 2002) address the motion planning for multiple vehicles keeping communication between them, which means that the vehicles are constrained to certain homotopy classes. (Fenwick and Estivill-Castro, 2005) compute the optimal paths in the mutually visible path problem assuming that agents are represented by points which move and pause along the paths to achieve the goal.

In this dissertation the application of homotopy classes from the path planning point of view is studied, since they provide topological information about how paths are generated with respect to a set of obstacles. The following sections present an overview of the most relevant contributions related with homotopy classes that can be applied to path planning. The aim of this overview is not to be an exhaustive enumeration of all the publications, but a description detailed enough to provide an insight into understanding the proposals which have been grouped according to the problem they try to solve.

2.7.1 *The Shortest Homotopic Path Problem*

This section describes proposals that address the computation of the shortest path when the homotopy is already specified, usually with an input path or with a constrained area through which the path has to go. This is a computational geometry problem that has direct application to VLSI, network routing, gaming or path optimization.

2.7.1.1 *Triangulated environments*

(Chazelle, 1982) proposed *the funnel algorithm* to compute the shortest path within a simple polygon called *channel*. A channel is assumed to be triangulated and hence the shortest path has to cross all the interior edges of the polygon. Notice that the start and goal points belong to the set of vertices of the channel. Although it was not explicitly designed to be used with homotopy classes, the funnel algorithm intrinsically constrains the path to a specific homotopy class described by the channel. Figure 18.a depicts a polygon with the interior edges to be traversed by the shortest path. The algorithm considers a *path* as the sequence of line segments that belong to the shortest path known within the channel at some point of the execution of the algorithm. There is a structure called *funnel*, which consists of two series of segments that represent the area where all shortest paths can be found. The two line series of the funnel are connected by a point called *apex*.

The algorithm starts by placing the initial apex at the start point. The funnel is constructed as the segments connect the start point with the first interior edge. At each step, the algorithm checks whether the vertices of the new interior edge are inside the funnel or not. If they are, the funnel is updated

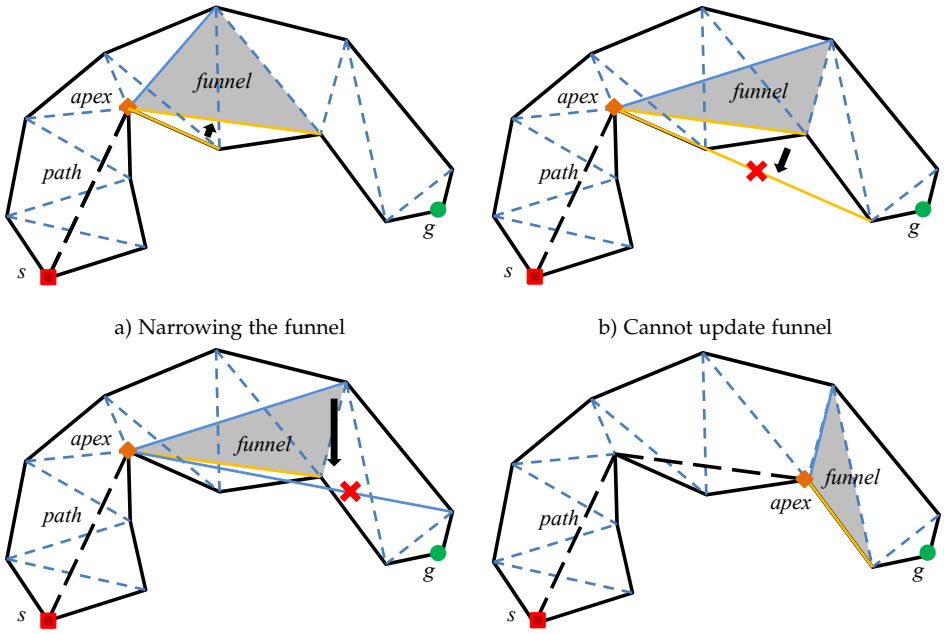


Figure 18: Different moments during the computation of the shortest path using the funnel algorithm. The dashed blue lines represent the interior edges of the channel.

and narrowed (see Figure 18.a). There are two situations that do no allow the funnel to narrow. The first is when the vertex of the next interior edge is outside the current funnel (Figure 18.b). The second is when adding the vertex of the new interior edge makes the current line segments of the funnel go over to its opposite line segments. In such case a new apex is set to the vertex that belongs to the opposite line segments that avoids such a situation (see Figure 18.c and Figure 18.d). The segment line between the current apex and the old apex is ensured to belong to the shortest path, and hence, is added to the *path*. This procedure is repeated until the channel has been completely explored.

Following the same line, (Hershberger and Snoeyink, 1991) compute the shortest path while keeping the homotopy of an input path in a simple polygonal environment. This method can be applied to more complex paths such as closed curves and does not require a channel to constrain the search. Essentially, it triangulates the environment and splits it into simplified subpolygons. Each one of these subpolygons contains a part of the input path, which is then optimized with a funnel algorithm (Chazelle, 1982) before being compounded back to generate the global optimal path.

(Cheng et al., 2010) propose a method to compute an approximate shortest homotopic path in a polygonal triangulated environment, where each region has a specific weight. Obstacles are represented by simple polygons. The method is aimed at encoding the homotopy class of a given input path according to the crossed edges of a spanning tree of obstacles. The tree checks whether an edge is crossed to the left or right obtaining the first representation

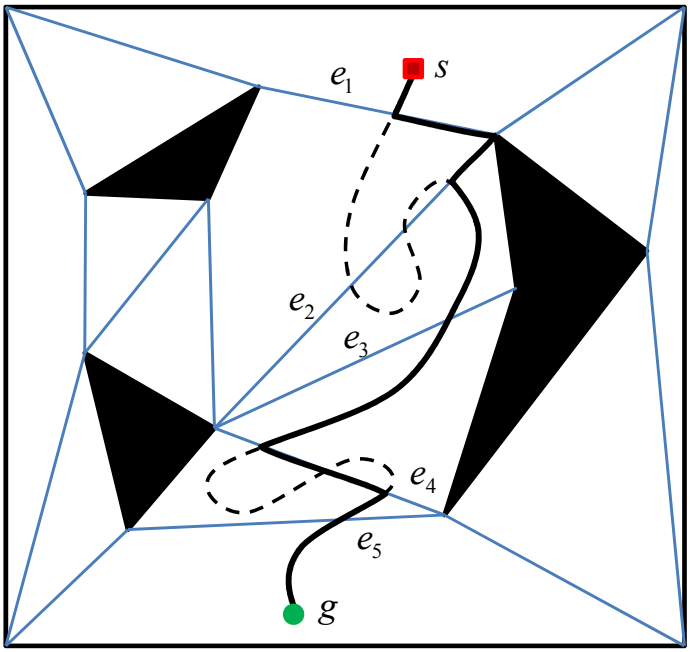


Figure 19: According to (Cheng et al., 2010), the input path (dashed line) is encoded according to the edges traversed on the triangulated environment differentiating whether an edge is crossed to the left or right: $\overleftarrow{e_1}\overrightarrow{e_2}\overleftarrow{e_2}\overrightarrow{e_2}\overrightarrow{e_3}\overleftarrow{e_4}\overrightarrow{e_4}\overleftarrow{e_4}\overrightarrow{e_5}$. Then, it is possible to obtain its canonical representation $\overleftarrow{e_1}\overrightarrow{e_2}\overrightarrow{e_3}\overleftarrow{e_4}\overrightarrow{e_5}$ which shortens the path (solid line). The next step would expand the triangulation to improve the path while keeping the canonical sequence of the homotopy class.

of the path, which is then improved until its canonical representation (see Figure 19). A graph that expands the triangulations of the tree is computed to generate a shorter path without changing its homotopy.

2.7.1.2 *Non-triangulated environments*

(Efrat et al., 2002) propose an algorithm that efficiently computes the shortest homotopic path for a set of input paths. The algorithm considers the start and goal points of each path and the obstacles as *terminal* points. It begins by applying vertical shortcuts to reduce each input path to a sequence of essential x-monotone path sections. A vertical shortcut is a vertical line segment that joins a path p at two or more points with the property that the subpath of p which follows the segment is homotopic to the line segment. The algorithm applies the maximum number of vertical shortcuts ensuring that, at each step, the length of the subpath is not increased. The second step of the algorithm bundles the essential x-monotone paths obtained in the previous step according to their homotopy class and chooses a representative path. Notice that if the paths in each bundle are homotopic, they have the same shortest path. Two x-monotone paths are homotopic if they share the same start and goal points and there are no other terminals between them. For each bundle of x-monotonic paths, the shortest homotopic path is computed using an improved funnel

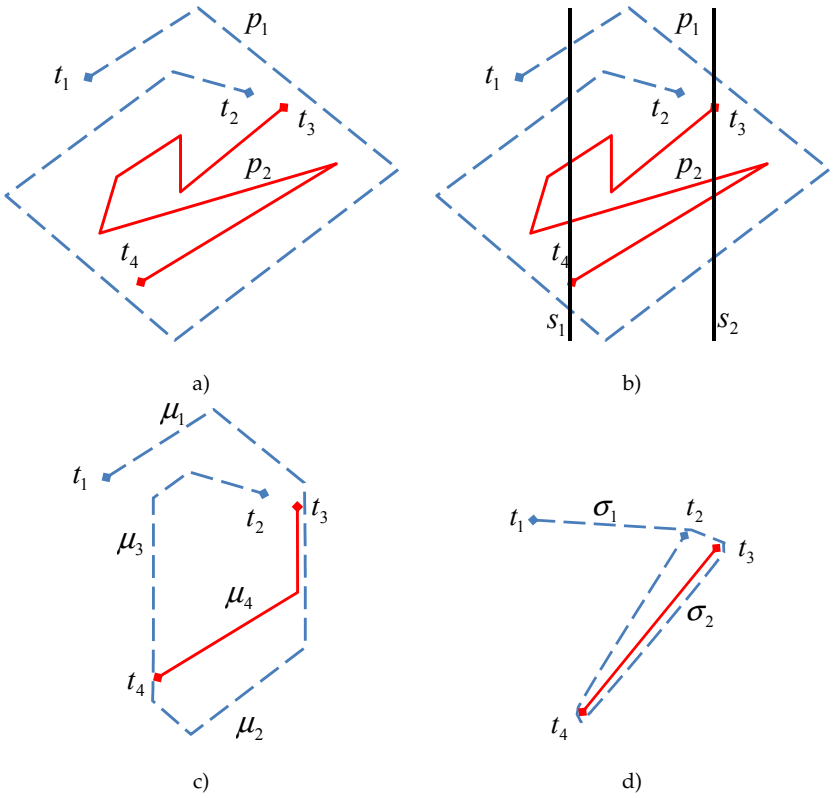


Figure 20: Computing the shortest homotopic paths: a) Input paths p_1 and p_2 with their respective terminal points (t_1 and t_2 for p_1, and t_3 and t_4 for p_2). b) Minimum necessary shortcuts s_1 and s_2 to obtain x-monotone paths. c) Monotone pieces after applying shortcuts: p_1 is divided into 3 monotone pieces: μ_1 from t_1 to t_3, μ_2 from t_3 to t_4, and μ_3 from t_4 to t_2; p_2 is represented by one monotone piece μ_4. μ_2 and μ_4 belong to the same bundle since they are homotopic because they share terminal points and there are no more terminal points between them. d) Paths after computing the shortest path for each bundle: σ_1 and σ_2 are the shortest homotopic paths of p_1 and p_2 respectively. Figure extracted from (Efrat et al., 2002).

technique (Chazelle, 1982) that reuses the information gained when each path is computed. Finally, the paths are unbundled to recover final paths. Figure 20 illustrates this process and Algorithm 5 summarizes the steps of the algorithm.

(Bespamyatnikh, 2003) uses a similar idea to Efrat et al. by first turning the paths into a set of x-monotone subpaths. Then, the subpaths that belong to the same homotopy class are bundled. For each bundle, the shortest path is found with a technique that assumes that the path is bound into a simplified polygon, which allows the efficiency of the method to be improved. Bespamyatnikh also extends the Efrat et al. proposal by allowing paths that self-intersect to be used as an input .

In the same vein, (Grigoriev and Slissenko, 1998) propose a method to construct the shortest path for a given homotopy class that does not intersect in a scenario with semi-algebraic obstacles. For each obstacle in the scenario a representative point is selected. From this point a curve called a *cut* is launched

Algorithm 5 Efrat's homotopic shortest path

1: Shortcut Paths
2: Bundle homotopically equivalent pieces
3: Compute the shortest path for each bundle
4: Recover final paths

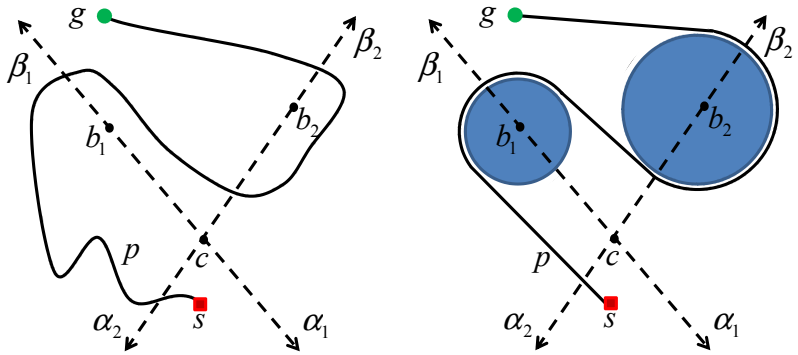
and goes to infinity. Then, assigning a letter to each cut and differentiating whether the cut is crossed clockwise or counterclockwise (e.g. α and α^1 respectively), any path can be represented by a *word* generated with its consecutive intersections with the cuts. The computation of the shortest path for any path consists of reducing the number of cuts –and the word– until the canonical sequence is found.

Given a set of input paths and rectangular obstacles, (Speckmann and Verbeek, 2010) compute the non-crossing thick minimum-link rectilinear homotopic paths. This is a two step process. The first consists of computing the shortest homotopic paths for the input paths using a modified version of the algorithm proposed by (Efrat et al., 2002). Then the paths are untangled according to the length of their shortest version while ensuring that the number of links is no more than doubled. Although this solution is strongly focused on VLSI routing, it is also interesting for path planning purposes since the thickness of the path is a constraint that can make the solution unfeasible and this algorithm is capable of detecting it, adapting the path, and, if it is not possible, reporting that there is no solution.

2.7.2 *Homotopy Classes Generation Approaches*

There are some approaches that do not require specifing the homotopy of the path to compute. Usually, these are two step methods that first generate the homotopy classes according to data structures that characterize the topology of the environment and then compute a path for each class.

For instance, (Jenkins, 1991) proposes a method to generate homotopy classes for any 2D workspace with obstacles. Assuming that any obstacle can be represented by a single point, the method first constructs a reference frame which is a geometric structure compounded of a set of semi-finite rays that establish the topological relationships between the obstacles in the metric space. The reference frame allows the description of any path in the workspace as a topological sequence according to the semi-rays traversed. Then, a topological graph, whose construction is based on the reference frame, allows the systematic generation of homotopy classes using graph-search algorithms. Once generated, (Cuerington, 1991) proposes a method to compute the shortest path in the workspace for each homotopy class with circular obstacles. Essentially, the algorithm looks for the direct straight path in the workspace that follows the homotopy class. If it does not exist, it is computed by following the tangent lines of the corresponding obstacles that accomplish the input homotopy class. The lines can be connected with the boundary of the obstacles to obtain the homotopic shortest path. Figure 21 depicts a simple



a) A path p in a reference frame b) The shortest homotopic path of p

Figure 21: A (Jenkins, 1991) and (Cuerington, 1991) example: a) In the reference frame, the obstacles are represented as single points b_k and a path p is described topologically according to the semi-rays traversed . b) The homotopic shortest path of p crosses the semi-rays of the reference frame in the same manner.

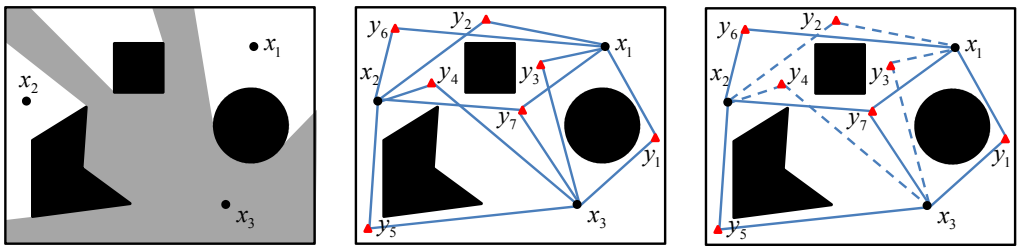
example of a homotopy class in the reference frame and its corresponding shortest path. Since the method proposed in this thesis is based on the Jenkins approach, more details are given in Chapter 3.

An example applied to robotics is proposed by (Milgram and Kaufman, 2000), who characterize topologically fixed routes such as energized wires embedded in a factory floor that are traversed by automatically guided vehicles. The goal is to avoid collisions between the vehicles while maintaining the homotopy of their original paths.

(Schmitzberger et al., 2002) propose to perform motion planning on a PRM which takes into account topological constraints. The method assumes the coverage of the C-space by a set of visibility domains. A visibility domain is the field of view of a point placed in the C-space. Figure 22.a depicts a scenario with tree points and the field of view of x_3. Then, a set of points that belong to at least two different visibility domains are added in order to generate a PRM that covers the full exploration of the C-space (see Figure 22.b). The added points may generate redundant loops that can be identified since the paths belong to the same homotopy class and therefore do not encroach on any obstacles. The final step consists of removing those points that generate redundant loops. The final result is a reduced PRM that allows the generation of trajectories which avoid obstacles in all possible manners (see Figure 22.c). However, the trajectories generated are far from optimal and postprocessing is required to achieve better results.

2.7.3 *Constraining Path Search Topologically*

This section groups a set of representative algorithms that starts by computing an initial path whose homotopy class is then obtained in order to prevent the algorithm from generating another path with the same topology. This



a) Field of view of x_3 b) Redundant PRM c) PRM without redundant loops

Figure 22: Schmitzberger et al. places a minimum number of x points that cover the whole C-Space with their fields of view (a), and others support points y in order to build a redundant PRM (b), which is simplified in c). The final PRM allows performing a suboptimal motion planning with any homotopy class. Images extracted from (Schmitzberger et al., 2002).

methodology achieves the generation of a path for different homotopy classes by blocking the ones previously explored.

(Fujita et al., 2003) propose a two step method to compute local minimum paths for homotopy classes that do not cross. Given a start and goal configuration in a C-space represented as a graph, the first step consists of computing, for each node, the shortest path between the start and the goal that passes through that node. This is done by running the Dijkstra's algorithm from the start to all the nodes in the graph, and running it again from the goal to all the other nodes. The cost of both searches is added to create a cost map, which is used to obtain the shortest path that traverses every node in the graph. Hence, the number of computed paths is the same as the number of nodes in the graph. The second step consists of pruning the number of paths in order to obtain the shortest path for each homotopy class. The procedure explores all the nodes in the graph in increasing order according to their value in the cost map. For each node, the shortest path that passes through that node is generated, and the nodes of the corresponding path are labeled. When the shortest path for the next node according to the cost map is generated, two possible situations can arise: when all path nodes adjoin at least one labeled node, the path has the same topology as a path previously computed and is discarded; otherwise, when the path has a node that does not adjoin any labeled node from the previous paths, it belongs to a different homotopy class. The process is repeated for all the computed paths. (Shiller et al., 2004) improve the second step of this method by identifying the homotopy classes according to their canonical sequence as (Cabello et al., 2002) propose. Therefore, the shortest path for each node of the cost map does not have to be completely traversed to look for a node with no junction with previous paths. Only the computation of the canonical sequence of the path is required to check whether it has been already found.

(Banerjee and Chandrasekaran, 2006) propose to perform path planning for military applications. Essentially, it generates a Voronoi diagram assuming the enemy zones to be avoided are the sites in the diagram. Based on the Voronoi diagram, a graph that keeps the homotopy is then constructed. The

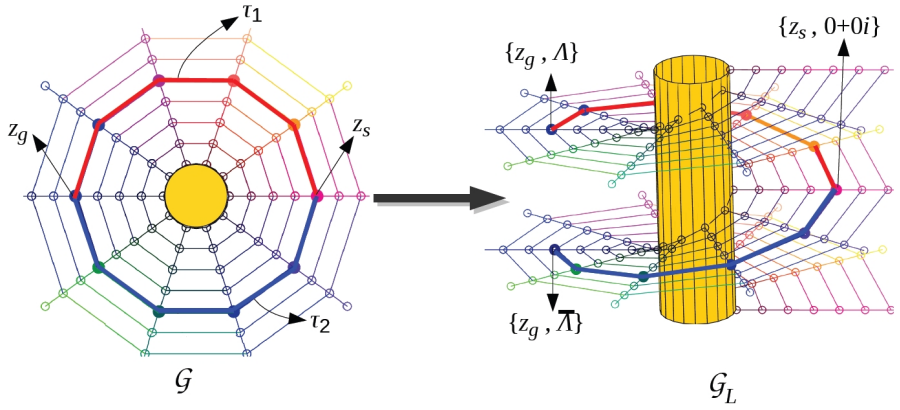


Figure 23: The original graph G is expanded by the *L-value* to generate the G_L graph. In the example, trajectories τ_1 and τ_2 are obtained from the start point in the complex domain z_s with an *L-value* of 0+0i, to the goal z_g which has an *L-value* Λ or $\bar{\Lambda}$ depending on the state of G_L. Image extracted from (Bhattacharya et al., 2010).

graph is traversed with graph-search algorithms based on the DFS which compute paths that have a direct equivalence to the Voronoi diagram. Notice that this method does not compute optimal paths since it focuses on avoiding undesirable zones.

(Bhattacharya et al., 2010) propose a method to perform path planning with homotopy class constraints using graph-search algorithms. The graph that represents the environment is expanded with Complex Analysis values to characterize homotopy classes while computing the path. Assuming that each obstacle is represented by a single point, Bhattacharya et al. define a function in the complex domain that allows the characterization of any node in the search space according to its relative position with respect to the obstacles. Since any trajectory is a sequence of nodes traversed in order, the line integration of the function generates a value that represents the trajectory according to its relative position between the obstacles. This value is called *L-value* and is used to identify the homotopy classes. Therefore, two trajectories that share the same start and end points with an equal *L-value* belong to the same homotopy class, whereas different *L-values* means that the two trajectories do not share the same homotopy class. In order to perform search-based path planning with homotopy class constrains, the search space is expanded with *L-values* while computing the path. Once the first path is computed, its homotopy class is identified through the *L-value*. Then, to obtain the path for a different homotopy class, the previous *L-value* is used to constrain the search. Figure 23 depicts a simple example. However, it is difficult to configure the path planner to follow a specific homotopy class without finding its path and hence, its *L-value*.

2.7.4 *Summary*

Computing the shortest homotopic path for an input path/homotopy class is a problem that has been studied since the 80's. Therefore, there is a large set of proposals that performs the computation efficiently for any possible path. However, when there is no input homotopy class or path, the problem becomes intractable for these solutions making it difficult to apply them to robot path planning. On the other hand, some of these algorithms are suitable for the optimization process, when a path has been already generated by a path planner.

There is also a group of methods that compute the shortest path and then identify its homotopy class. This process can be done during the path search or once the whole path has been computed. Then, the topology of the path is encoded in order to restrict the next path search, which ensures that the new shortest path will have a different topology and hence, belong to another homotopy class. By repeating this process, it is possible to obtain the k-shortest paths of k-homotopy classes. These methods have the clear advantage of starting with the computation of the global shortest path. However, if we are interested in the solution of the k path, computing all the previous paths is required.

On the other hand, there is a minor group of methods that first compute the homotopy classes and then search for a path that follows them. In order to generate the homotopy classes, a data structure able to codify the topology of the environment as a graph is required. Then, the homotopy classes can be systematically generated by exploring the graph with a graph-search algorithm. These methods offer the flexibility of computing a path that does not belong to the homotopy class of the global optimal path without having to compute any previous paths. However, generating the homotopy classes systematically can make the problem intractable depending on the number of classes generated. This problem can be overcome by using some restriction criteria during the generation of the classes such as allowing the generation of homotopy classes in their canonical form, and avoiding those classes that self-cross or repeat cycles.

A summary of the reviewed methods that takes into account the homotopy of the paths can be seen in Table 1. The outlined features are:

- A short description of the problem to be solved.
- The dimensionality of the problem.
- The required input when it is necessary to specify the homotopy class of the solution.
- The types of obstacles the methods are able to deal with.
- The additional data structure required to name the homotopy classes.
- The homotopy class representation.
- Whether the method can deal with paths that self-cross.

2.8 PATH PLANNING FOR AUVS

Although the literature on path planning applied to AUVs is not extensive, several proposals that follow a wide range of different approaches have been reported. (Warren, 1990) presents an algorithm that uses artificial potential fields to aid in the path planning. It is first required to generate a trial path and then modify the entire path under the influence of the potential fields. A path is a set of connected straight line segments, only those endpoints of the segments that remain at the start and end positions are fixed, while the others are modified according to the potential field in order to generate a safe path. (Carroll et al., 1992) apply the A* algorithm in a graph-based environment built according to a previously gathered bathymetry, water currents and exclusion zones. The A* has also been adopted by (Garau et al., 2005) to compute optimal paths in terms of energy cost in underwater environments with eddies. Recently, (Fernandez-Perdomo et al., 2011) have adapted the A* to perform a constant time surfacing algorithm that computes suitable trajectories to be followed by a Glider.

(Vasudevan and Ganesan, 1994) propose to perform path planning with a case-based algorithm which is part of a larger planning system. For a given set of routes to be followed, the planner relies on past experience to adapt the old routes to achieve the goal. Further, it can also build new routes by adopting the same strategies for those cases that approximately match the current situation such as connecting a new segment of a route with a previously existing one. (Petillot et al., 2001) propose a path planning strategy based on nonlinear programming techniques using acoustic images gathered with a multi-beam forward looking sonar in an obstacle avoidance system for a ROV

(Alvarez et al., 2004) use a genetic algorithm to perform path planning for an AUV in an ocean environment characterized by strong currents, minimizing the energy cost. (Kruger et al., 2007) proposes a technique to perform path planning in rivers based on analytic models, which enables an AUV to plan optimally exploiting useful currents, avoiding adverse currents, maximizing speed, minimizing energy and avoiding obstacles. Water currents are also taken into account by (Petres et al., 2007), who developed a path planning algorithm called E* that computes a continuous path from a discrete representation of the environment based on the FM approach.

Some authors tackle the coverage path planning problem for AUVs, since this kind of survey has a wide range of applications such as map building purposes, mosaicking and target localization. For instance, (Maki et al., 2009) propose a path planning method that generates a set of waypoints for an AUV to follow the surface of the target at a constant distance, based on the previously given information about the configurations of both the target and the vehicle. The target application is the support leg inspection of on-water platforms consisting of cylindrical rods. The method starts by turning the 3D workspace into a set of 2D slices at different depths parallel to the surface. The result is a set of 2D workspaces with cylindrical obstacles where paths are generated following their boundaries at a safe distance. (Williams, 2010)

addressed the problem of designing the optimal survey route for an AUV in mine countermeasure operations according to the area perceived by its sonar sensors. (Paull et al., 2010) propose an adaptive path planning method that modifies a global precomputed path according to the AUV's measurements of the environment with side-looking sensors to achieve complete coverage of a specific area.

Over the last few years, data collection using AUVs has been increasing in importance within the oceanographic research community. For this purpose, (Binney et al., 2010) adapt an informative path planner based on a recursive greedy algorithm focused on collecting data within a certain area using Gliders. The algorithm uses previous knowledge of the environment gathered before releasing the robot to compute the path. Despite receiving updated information from the onboard sensors, the algorithm only uses it as a complement to the previous data, but not to improve the path. On the other hand, (Smith et al., 2010) present an algorithm designed to generate paths for AUVs to high-value location for data acquisition. The algorithm focuses on tracking dynamic ocean features according to ocean model predictions and predicted current velocities. Hence, the predefined path has to be modified according to the features of interest perceived.

2.9 DISCUSSION

In this survey we have explored a set of different proposals to perform path planning. The graph-based search algorithms look for the global shortest path in the C-space, which is obtained with an exhaustive exploration of the search space. Most of these algorithms use a heuristic function in order to speed up the exploration process by first selecting the most promising states according to the heuristics. On the other hand, probabilistic sample-based path planners, most of them based on the RRT, perform the exploration by growing a tree incrementally until the goal is reached. These algorithms do not perform an exhaustive exploration in C-space, therefore, they provide a solution very quickly at the expense of its quality. Because of this, it is a common practice to improve the computed path with a post-processing step.

Anytime approaches have been shown suitable to be used with robots that have a limited amount of time to perform path planning. These algorithms compute an initial solution highly suboptimal very quickly and improve it until time runs out. They speed up the path generation by inflating the heuristics to force the exploration of those configurations that are closer to the goal according to their heuristics. During successive iterations, the inflation factor is decreased in order to generate better solutions. However, it is difficult to set up the correct inflation value since it is highly dependent on the complexity of the scenario. Moreover, even with decreasing the heuristics inflation, the generation of a better path is not ensured.

In the survey, Bug-based approaches, initially developed to perform motion planning, have also been considered for path planning purposes since their navigation strategies are suitable to generate paths in a C-space. These

Cite	Description	Dimension	Input	Obstacles type	Add. data structure	Homotopy class representation	Self-crossing allowed
Chazelle (1982)	Shortest path	2D	Triangulated simple polygon	–	–	Triangulated simple polygon	No
Hershberger and Snoeyink (1991)	Shortest homotopic path	2D	Path	Polygonal	–	Edge sequence	Yes
Cheng et al. (2010)	Approximate shortest homotopic path	2D	Path	Polygonal	Spanning tree	Edge label seq.	Yes
Grigoriev and Slissenko (1998)	Shortest homotopic path	2D	Hc/ Path	Semi-algebraic	Labeled curves	Word (label seq.)	No
Efrat et al. (2002)	Shortest homotopic path	2D	Path	Points	–	Edge sequence	No
Bespamyatnikh (2003)	Shortest homotopic path	2D	Path	Points	–	Edge sequence	Yes
Speckmann and Verbeek (2010)	Rectilinear homotopic thick path	2D	Paths	Rectangular	–	Edge seq	No
Jenkins (1991)	Homotopy classes computation	2D	–	Any kind (points)	Reference frame Topological graph	Edge label seq.	–
Cuerington (1991)	Shortest homotopic path	2D	Hc	Circles	Reference frame	Edge label seq.	Yes
Milgram and Kaufman (2000)	Topological environment characterization	2D	Paths	–	–	Seq. nodes	Yes
Schmitzberger et al. (2002)	Motion planning with homotopy classes	2D	–	Geometric	–	Edge sequence	Yes
Fujita et al. (2003)	Shortest path with homotopy constraints	2D	–	Any kind	–	Point sequence	No
Shiller et al. (2004)	Shortest path for each homotopy class	2D	–	Any kind	–	Point sequence	No
Banerjee and Chandrasekaran (2006)	N-shortest paths	2D	–	Points	Voronoi diagram	Seq. traversed edges	Yes
Bhattacharya et al. (2010)	Shortest path with homotopy constraints	2D	–	Any kind	Expanded graph	Complex number	Yes

Table 1: Summary of selected methods that explicitly deal with homotopy classes.

algorithms look for a path to the goal by following a straight line until an obstacle on the path is detected. At this point, the obstacle is surrounded until a stopping criterion makes the path go straight towards the goal again. Therefore, only the search space around the boundary of the obstacles towards the goal have to be explored.

The topological approaches described in this survey mainly focus on homotopy classes, which provide information on how paths avoid obstacles. In order to perform path planning for an AUV, methods that compute the shortest homotopic path are not suitable because it is first necessary to generate a path. Neither are the methods that restrict the path search according to the topology of the paths previously computed. Hence, if the path we are interested in does not belong to the homotopy class of the global optimal solution, it is required to compute a path for each homotopy class that has a shorter path. Finally, those methods that first generate the homotopy classes are the most suitable to achieve success in our path planning for an AUV, since they allow us to look for a path that accomplishes other criteria than its shortest length, such as surveilling an area or avoiding dangerous zones.

Despite the range of the solutions surveyed to perform path planning for AUVs, most of which are highly specific to the problem they want to solve and hence, are difficult to apply in a different context. Some of these solutions take into account water currents, which implies previous knowledge of the scenario that is not provided in our target applications. Further, none of them take into account the homotopy classes of the generated paths. Because of this, our proposal focuses on the topological approach presented by (Jenkins, 1991), which is a generic method to compute the homotopy classes independently from the path planning approach selected to compute the final path in the C-space. However, it is necessary to establish a set of restriction criteria in order to generate only those homotopy classes that are interesting for the problem we need to solve.

3

PATH PLANNING WITH HOMOTOPY CLASS CONSTRAINTS

This chapter concerns the path planning method developed for this research project. It is assumed that the computation of the path has to be performed on a local map built for navigation purposes with data provided by the onboard sonar sensors of the vehicle. In order to achieve the realtime requirements of the robot, the computation of the path has to be done in a short time.

Our method proposes the utilization of homotopy classes in order to constrain the path generation topologically. As pointed out in Chapter 2, a homotopy class of a set of trajectories from the start to the end point describes how these trajectories avoid obstacles. From the different approaches that deal with homotopy classes, those methods that first generate the classes neither compute previous path to obtain its class nor require giving any input homotopy class specified by the user. Therefore, they are suitable for the automatic generation of homotopy classes for surveilling or zone avoiding purposes, since they may not follow the homotopy class of the global optimal path. However, these methods require establishing a set of restriction criteria in order to allow the generation of only those homotopy classes that are interesting for the problem to be solved.

Given any 2D environment, our method first generates the set of all the homotopy classes that connect the start point with the end point with an extension of the (Jenkins, 1991) proposal we have developed. Using the topological information, path planners do not have to explore the whole space but the space confined in a homotopy class. Then, using a lower bound criterion that estimates the cost of the homotopy classes in the workspace, it is possible to set up a preference order when choosing a homotopy class to generate its path. Finally, three different path planning algorithms have been developed in order to compute paths in the C-space that accomplish the homotopy classes.

Section 3.1 gives an overview of the method to generate homotopy classes proposed by (Jenkins, 1991). In Section 3.2 we propose an extension of the method that allows the generation of homotopy classes that can be followed in any 2D workspace. Section 3.3 shows the heuristic estimator used as a lower bound criterion to set up a preference order when looking for paths of the homotopy classes. Section 3.4 describes three different path planning algorithms designed to follow the homotopy classes previously computed. Finally, in Section 3.5 a summary is provided.

3.1 OVERVIEW

The method to generate homotopy classes that we propose is based on the (Jenkins, 1991) approach. Given a 2D workspace, this method first constructs a

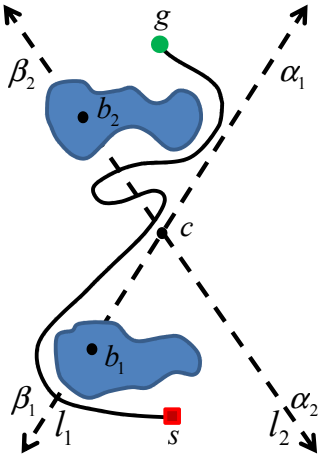


Figure 24: A topological path represented in the reference frame as $p = \beta_1\alpha_2\alpha_2\alpha_2$ with the transitions labeled.

reference frame that allows the representation of any path in the workspace as a topological sequence. The reference frame is also used to build a topological graph which is explored with graph-search algorithms to generate topological sequences that represent homotopy classes.

More in detail, the reference frame determines, in the metric space, the topological relationships between obstacles and is used to name the homotopy classes. In a workspace with n obstacles, construction starts by representing each obstacle with a single point b_k, where $k = 1..n$. Then, a central point c is selected. This point cannot be inside an obstacle nor inside the $n(n-1)/2$ lines determined by the pairwise choices of distinct b_k. Finally, the n lines l_k joining c with each b_k are constructed. Each line is partitioned into two directed semifinite rays: the ray emanating from b_k and away from c is labeled β_k and the ray from b_k that contains c is labeled α_k. Figure 24 shows an example of a reference frame in a scenario with two obstacles with a path described by the rays of the reference frame it crosses.

The topological graph, whose construction is based on the reference frame, provides a model to describe the topological relationships between regions of the metric space. The reference frame divides the metric space into regions called *wedges*. Each wedge represents a node in the graph which is connected to its neighbors through labeled edges according to the semifinite rays shared with its contiguous wedges in the reference frame.

Figure 24 depicts an example of a reference frame of a simple workspace with two obstacles and Figure 25 shows its correspondent topological graph. In the reference frame, lines l_1 and l_2 split the workspace into four different wedges, hence, the topological graph has four nodes. The wedges in the reference frame and their corresponding nodes in the topological graph are labeled in increasing order in a counter clockwise direction. As stated earlier, nodes of the topological graph are connected with their neighbors according to the segments shared between wedges in the reference frame. For instance, wedges 1 and 2 share a single semifinite ray α_2 in the reference frame, thus, their nodes in the topological graph are connected with the edge labeled α_2.

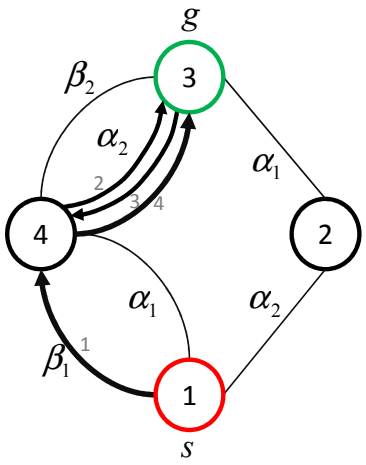


Figure 25: Path $p = \beta_1\alpha_2\alpha_2\alpha_2$ represented in the topological graph with bold arrows.

The wedges of nodes 1 and 4 share the α_1 and β_1 rays, so they are connected with the two edges with these labels. In the reference frame, a path is defined according to the segments it crosses, whereas, in the topological graph, it turns into traversing the graph from the start node to the end node. Start and end nodes are those wedges in the reference frame, nodes in the topological graph, where the start and end points are located.

Once the topological graph is constructed, Jenkins proposes traversing it using a modified version of the BFS. The algorithm begins at the start node and new homotopy class candidates are systematically generated according to the edges traversed. In order to reduce the potential number of classes generated, the author provides a set of restriction criteria to avoid the generation of the homotopy classes that self-cross or are duplicated according to their canonical sequences. The algorithm does not stop when the goal node has been reached, it continues generating potential solutions until all the homotopy classes have been computed or the last candidate homotopy class generated is larger than a given threshold. Table 2 shows a small part of the execution of the BFS algorithm and Table 3 summarizes the homotopy classes generated with the method proposed by Jenkins. It is worth noting that this method achieves the generation of homotopy classes in any 2D workspace assuming that obstacles are represented by single points. Despite that it can be applied to any kind of obstacle, depending on their shape/size, resultant classes might not be followed back in the workspace with the original obstacles. The next section details this problem.

3.1.1 *Applicability to the Path Planning Problem*

Our work proposes using the Jenkins method to guide a path planning algorithm following a topological path. Thus, the topological information of the homotopy classes has to be turned into metric paths in the workspace by using the reference frame as a link between the topological graph and the workspace. In order to find a path in the workspace that follows a specific homotopy class,

Homotopy class
α_1
β_1
α_2
$\alpha_1\beta_1$
$\alpha_1\alpha_2$
$\alpha_1\beta_2$
$\beta_1\alpha_1$
$\beta_1\alpha_2$
$\beta_1\beta_2$
$\alpha_2\alpha_1$
$\alpha_1\beta_1\alpha_1$
$\alpha_1\beta_1\alpha_2$
...

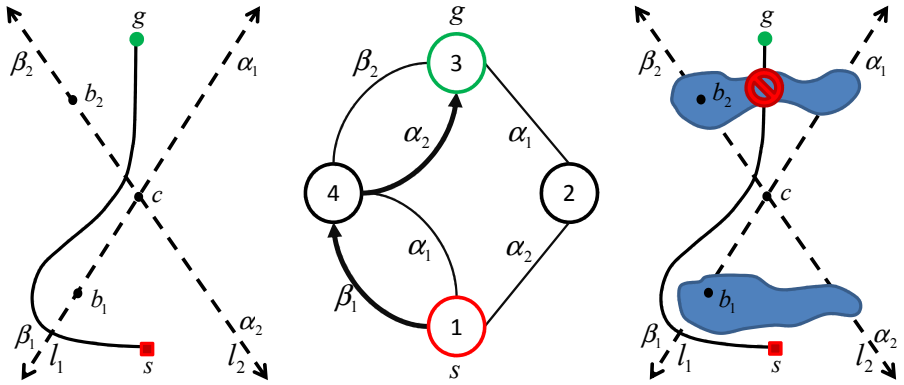
Table 2: A systematic generation of homotopy class candidates with the BFS algorithm.

Idx	Homotopy class
1	$\alpha_1\alpha_2$
2	$\alpha_1\beta_2$
3	$\beta_1\alpha_2$
4	$\beta_1\beta_2$
5	$\alpha_1\alpha_2\beta_2\alpha_1\beta_1\beta_2$
6	$\beta_1\beta_2\alpha_2\beta_1\alpha_1\alpha_2$

Table 3: Homotopy classes with their index of generation obtained with the topological graph in Figure 25.

it is required for a path planning algorithm to be modified to look for the intersections with the desired segments in the reference frame.

During the reference frame construction, each obstacle in the workspace is represented by a b_k point without area in order to ensure that each line in the reference frame crosses only one obstacle; and the topological graph is built under this assumption. However, depending on the reference frame construction and the particular shape of the obstacles, it is possible that a line in the reference frame intersects with more than one obstacle. In some cases, there will be homotopy classes that cannot be followed in the workspace. Figure 26 depicts this problem: the homotopy class to follow is $\beta_1\alpha_2$, which is shown as a path (Figure 26.a) in the reference frame with its equivalence in the topological graph (Figure 26.b). However, in Figure 26.c the metric path cannot be followed because obstacle 2 crosses l_2 and l_1. Notice that this problem would not arise if points c and b_1 were more carefully selected to avoid the



a) Reference frame b) Topological graph c) Workspace

Figure 26: Example of a valid homotopy class ($\beta_1\alpha_2$) in the reference frame (a) and in the topological graph (b) that cannot be followed in the workspace (c) because at least one line (l_1 or l_2) in the reference frame intersects more than one obstacle.

intersection of obstacle 2 with l_1. However, this solution cannot be applied in complex scenarios with more obstacles.

3.2 HOMOTOPY CLASSES GENERATION

This section describes the generation of homotopy classes for any 2D workspace. The approach presented here is an extension of the method proposed by (Jenkins, 1991) which ensures that all the homotopy classes computed can be followed in the workspace. As stated before, this method first builds a reference frame which determines the topological relationships between obstacles in the workspace and is used to name the homotopy classes. The reference frame is then used to build the topological graph which allows the computation of homotopy classes systematically.

3.2.1 *Reference Frame*

Given a workspace with n obstacles, the reference frame determines, in the metric space, the topological relationships between obstacles and is used to name the homotopy classes. The whole construction process is summarized in three steps:

1. Select a random point inside each obstacle and label it b_k, where $k = 1..n$.
2. Select the central point c of the reference frame. This point cannot be inside an obstacle nor inside the $n(n-1)/2$ lines determined by the pairwise choices of distinct b_k.
3. Construct n lines l_k joining c with each b_k. Each line is partitioned into $m+1$ segments, where m is the number of obstacles that intersect with l_k in the workspace. The segments from b_k and away from c are labeled

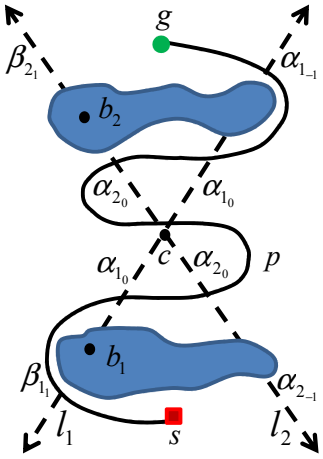


Figure 27: Topological path represented in the reference frame as $p = \beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_0}\alpha_{2_0}\alpha_{2_0}\alpha_{1_0}\alpha_{1_{-1}}$

with β_{k_s}, and the segments in the opposite direction are labeled α_{k_s}, where $s = 0..u$ with $u \in \mathbb{Z}^+$ for the segments of l_k from c that pass through b_k and $s = 0..v$ with $v \in \mathbb{Z}^-$ for the segments in the opposite direction.

Using the reference frame, any path p can be defined topologically by the sequence of labels of the segments crossed in order from the start to the end point. For instance, Figure 27 depicts a reference frame for a scenario with two obstacles. The path p is labeled $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_0}\alpha_{2_0}\alpha_{2_0}\alpha_{1_0}\alpha_{1_{-1}}$. There are two special cases when defining paths in the reference frame: when p does not cross any rays then $p = \emptyset$; and when p crosses through c meaning that all the α_{k_0}'s are simultaneously crossed, all α_{k_0} are added in subindex order to the sequence.

Notice that the start and goal points cannot be in lines l_k it the reference frame since it would not be possible to determine whether the affected l_k lines are crossed or not.

3.2.2 *Computation of the Canonical Sequence*

As stated in Chapter 2, it is possible to know if the paths that do not follow the same crossing-ray order in the reference frame are homotopic through their canonical sequence. The canonical sequence is the simplest representation of a path without changing its topology. With the extension of the notation used in the reference frame, it is computed according to Algorithm 6. First, the α_{k_0}'s substrings are sorted according to the subindex of the path in non-decreasing order. Then, all the elements of the sequence that have the same character by pairs are removed. This process is repeated until no changes are made in the sequence. For instance, once path $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_0}\alpha_{2_0}\alpha_{2_0}\alpha_{1_0}\alpha_{1_{-1}}$ gets its α_{k_0}s substring sorted, it becomes $\beta_{1_1}\alpha_{1_0}\alpha_{1_0}\alpha_{1_0}\alpha_{2_0}\alpha_{2_0}\alpha_{2_0}\alpha_{1_{-1}}$. At this point a α_{1_0} and a α_{2_0} pairs can be canceled $\beta_{1_1}\cancel{\alpha_{1_0}\alpha_{1_0}}\alpha_{1_0}\cancel{\alpha_{2_0}\alpha_{2_0}}\alpha_{2_0}\alpha_{1_{-1}}$ obtaining $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$. Since it cannot be shortened, it represents the canonical se-

quence of the path. Figure 28 depicts one possible solution in the workspace of this canonical sequence.

Algorithm 6 Canonical representation

CanonicalRepresentation(p)

1: **repeat**
2: $p_{sorted} \leftarrow Sort\alpha_{k_0}'sSubstrings(p)$
3: $p_{canceled} \leftarrow CancelEqualPairs(p_{sorted})$
4: $p \leftarrow p_{canceled}$
5: **until** $(p_{canceled} = p_{sorted})$
6: **return** p

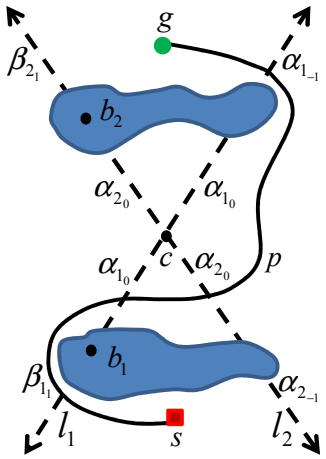


Figure 28: A possible path in the reference frame of the canonical sequence $p = \beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$ obtained with the sequence $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_0}\alpha_{2_0}\alpha_{2_0}\alpha_{1_0}\alpha_{1_{-1}}$.

3.2.3 *Topological Graph*

The reference frame is used to compute a topological graph G, providing a model to describe the relationships between regions of the metric space. Its construction can be divided into three steps:

1. The lines in the reference frame divide the metric space into regions or *wedges* and the obstacles that intersect with more than one line at the same time split these wedges into *sub-wedges*. Each sub-wedge represents a node of G.

2. Each node of G is labeled according to the wedge w and sub-wedge sw using the notation $w.sw$. $w \in \mathbb{N}$ is numbered counterclockwise. For each w, its corresponding $sw \in \mathbb{N}$ is numbered sequentially starting by 1 for the one closest to c.

3. Two nodes of G are interconnected according to the number of segments they share in the reference frame. Each edge of G is labeled with the same label of the segment that crosses it in the reference frame.

In the reference frame, a path is defined according to the segments it crosses, whereas in G it turns into traversing the graph from the start node to the end node. Notice that the start and end nodes of G are those sub-wedges in the reference frame where the start and end points are located. Figure 29 depicts the canonical sequence $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$ in the topological graph.

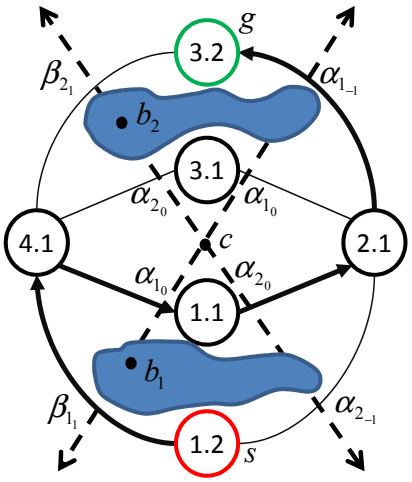


Figure 29: The path $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$ represented in the topological graph.

3.2.4 *Systematic Homotopy Classes Computation*

Once the topological graph is constructed, it is traversed using a modified version of the BFS algorithm. As stated in Chapter 2, the BFS is a graph search algorithm that begins at the root node and explores all the neighboring nodes. Then, for each of those nearest nodes, it explores their unexplored neighbors. The process is repeated until the goal is found. Unlike the standard BFS, which stops when all vertexes have been visited, the modified algorithm continues until there are no more homotopy class candidates to explore or the length of the last homotopy class candidate is larger than a given threshold.

3.2.4.1 *Restriction Criteria*

During the BFS execution, several restriction criteria are applied to avoid the generation of any homotopy classes which either self-intersect or whose canonical sequence is duplicated and has already been considered. All classes that accomplish any of the following restriction criteria are ignored to avoid using them as a root for future homotopy classes:

SIMPLE WRAP. Any string that contains a substring of the form $\alpha_{k_s}...\chi_{k_t}...\alpha_{k_u}$ or $\beta_{k_s}...\chi_{k_t}...\beta_{k_u}$ where $\chi = (\alpha, \beta)$ with $s = u$ represents a class that wraps around an obstacle and is self-crossing. Figure 30 shows an example of a path that accomplishes the simple wrap criterion.

WRAP. Any string that contains a substring of the form $\chi_{k_s}...\chi_{k_t}...\chi_{k_u}$ where $\chi = (\alpha, \beta)$ with $s, t, u \geqslant 0$ and $s > t < u$ or with $s, t, u \leqslant 0$ and $s < t > u$

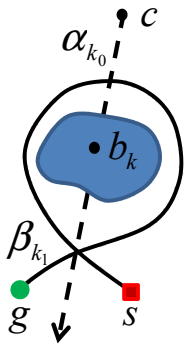


Figure 30: A simple wrap in path $\beta_{k_1}\alpha_{k_0}\beta_{k_1}$.

represents a class that wraps around an obstacle and is self-crossing. Figure 31 shows an example of a path that accomplishes the wrap criterion.

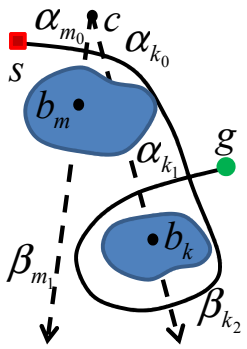


Figure 31: A wrap in path $\alpha_{m_0}\alpha_{k_0}\beta_{k_2}\alpha_{k_1}$.

SELF-CROSSING. Any string that contains a substring of the form $\chi_{k_s}...\beta_{m_t}...\alpha_{m_u}...\chi_{k_v}$ where $\chi = (\alpha, \beta)$ with $s, v \geqslant 0$ and $s < v$ or with $s, v \leqslant 0$ and $s > v$ represents a class that self-crosses. The reversed substring $\chi_{k_s}...\alpha_{m_t}...\beta_{m_u}...\chi_{k_v}$ with $s, v \geqslant 0$ and $s > v$ or with $s, v \leqslant 0$ and $s < v$ also represents a class that self-crosses. Figure 32 shows an example of a path that accomplishes the self-crossing criterion.

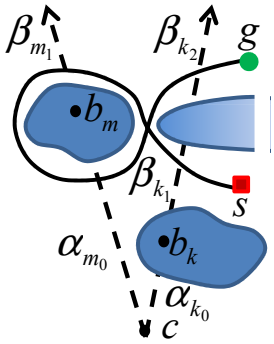


Figure 32: Self-crossing in path $\beta_{k_1}\beta_{m_1}\alpha_{m_0}\beta_{k_2}$.

DUPLICATED. Duplicated strings are not allowed in the list of homotopy class candidates. If a string is not in its canonical form, it can be simplified without modifying its topology. Then, it is ensured that the resultant string has already been computed by the BFS algorithm because it would be shorter than the input string. Finally, the algorithm cannot traverse the same edge on two consecutive occasions. By doing that, a string with a repeated pair would be generated. Consequently, the pair would be simplified and the string discarded for being duplicated.

Table 4 shows the homotopy classes computed for the workspace depicted in Figure 28 applying all the restriction criteria described in this section. Notice that all the homotopy classes can be followed in the workspace with the original obstacles. For a detailed execution of the homotopy class generation in this example see Appendix A.

Idx	Homotopy class
1	$\beta_{1_1}\beta_{2_1}$
2	$\alpha_{2_{-1}}\alpha_{1_{-1}}$
3	$\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$
4	$\alpha_{2_{-1}}\alpha_{1_0}\alpha_{2_0}\beta_{2_1}$

Table 4: Homotopy classes obtained with the extension of the Jenkins method we propose. The first column shows their index of generation.

3.3 LOWER BOUND ESTIMATOR

The number of homotopy classes generated by the BFS algorithm highly depends on the number of the nodes in the topological graph. Therefore, in most scenarios it is not possible to compute all the correspondent paths of the homotopy classes in the workspace in real-time. In order to set up a preference order when choosing the homotopy classes to compute their paths, a modified version of the funnel algorithm (Chazelle, 1982) is used to obtain a quantitative measure for each homotopy class estimating its quality. This algorithm computes the shortest path within a *channel*, which is a polygon formed by the vertexes of the segments in the reference frame that are traversed in the topological graph. The modification consists of accumulating the Euclidean distance between the points while they are being added to the channel's shortest path. Hence, the result of the funnel algorithm is a lower bound of the optimal path in the workspace of the selected homotopy class.

The lower bound estimator is used to set up a preference order to compute the homotopy classes path in the workspace when operating under time restrictions. Notice that the segments of the reference frame constrain the region where the paths can go through, but do not take into account the whole shape of the obstacles. For that reason, a homotopy class with a smaller lower bound may have a longer path in the workspace than an another homotopy class with a higher lower bound.

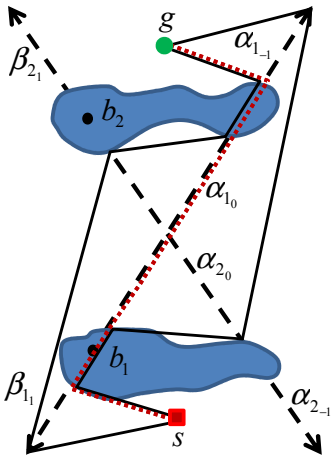


Figure 33: The channel and lower bound path for the homotopy class $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$.

Idx	Homotopy class	Lower bound
1	$\beta_{1_1}\beta_{2_1}$	0.94
2	$\alpha_{2_{-1}}\alpha_{1_{-1}}$	0.95
3	$\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$	1.04
4	$\alpha_{2_{-1}}\alpha_{1_0}\alpha_{2_0}\beta_{2_1}$	1.08

Table 5: Lower bounds for each homotopy class normalized according to the cost of the A* solution.

Figure 33 depicts an example where the funnel algorithm computes the lower bound for the homotopy class $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$. The solid lines represent the channel and the dashed red line is the path after applying the funnel algorithm. It is worth noting that the modified algorithm takes into account that some subsegments may self-intersect when creating the channel as can be seen with the α_{1_0} and α_{2_0} segments in the figure. Table 5 shows the lower bound for each homotopy class obtained in this example. All lower bounds have been normalized according to the cost of the path obtained with the A*, which belongs to class 2 ($\alpha_{2_{-1}}\alpha_{1_{-1}}$), using an A8 connectivity in a discretized representation of the environment. Some values are lower than 1 since the A* solution takes into account the shape of the obstacles while the computation of the lower bound does not.

3.4 HOMOTOPIC PATH PLANNING ALGORITHMS

Once the homotopy classes are computed and sorted according to their lower bound, a path planning algorithm has to find a path in the workspace that follows a given homotopy class which essentially implies turning a topological path into a metric path. The only link between the workspace and the topological space is the reference frame. It allows checking whether a metric path in the workspace is following a topological path by following the intersections in order from the initial configuration to the current configuration.

This section presents three well known algorithms that have been adapted to compute paths for a single homotopy class. The first algorithm is the homotopic version of the A* called Homotopic A* (HA*). It allows the computation of the optimal path according to a homotopy class to follow. The second algorithm is a probabilistic approach based on the RRT called Homotopic RRT (HRRT). Finally, the third algorithm is a Bug-based approach called Homotopic Bug (HBug) which tries to follow the lower bound path until it collides with an obstacle. Then the algorithm surrounds the boundary of the obstacle while the homotopy class is being followed till the point where the lower bound path leaves the obstacle.

3.4.1 *Homotopic A**

The Homotopic A* (HA*) works like the A* algorithm, but instead of exploring the entire search space, it only explores the zones in the workspace that satisfy a given homotopy class by checking the intersections with the reference frame before taking into consideration the node as a candidate to be explored.

Given a node n, the algorithm uses a heuristic function $f(n)$ based on the distance to the goal and the cost from the start to set up the order in which the search visits the nodes. The heuristic function is $f(n) = g(n) + h(n, n_{goal})$ where:

- $g(n)$ is the path-cost function, which is the cost from the start node to the current node.
- $h(n, n_{goal})$ is the heuristic estimation of the distance from the current node to the goal which does not overestimate the distance to the goal.

3.4.1.1 *Implementation*

The HA* is written in pseudocode in Algorithm 7. The nodes in the algorithm are tuples that contain the configuration of robot q and the topological path from q_{start} to q. These values are accessible through the functions Q and P respectively. Just like the A*, open nodes are processed according to their position in a priority queue OPEN. Each node in this queue is ordered according to the sum of its current path cost from the start, $g(n)$, and a heuristic estimation of its path cost to the goal, $h(n, n_{goal})$. The node with the minimum sum is at the top of the priority queue.

The algorithm receives as input the start configuration q_{start}, the goal configuration q_{goal}, a candidate homotopy class to follow H and the reference frame F. The configurations q_{start} and q_{goal} are used to set up the initial node n_{start} and the goal node n_{goal} (line 27).

The function ComputePath computes the shortest path that follows H. It starts by adding the n_{start} into the OPEN queue. As node n with the minimum $g(n) + h(n, n_{goal})$ is different from n_{goal}, the algorithm pops n to the top of the queue. For all the configurations q' reachable from $Q(n)$, the function FindIntersections (line 15) returns the intersections of the segment

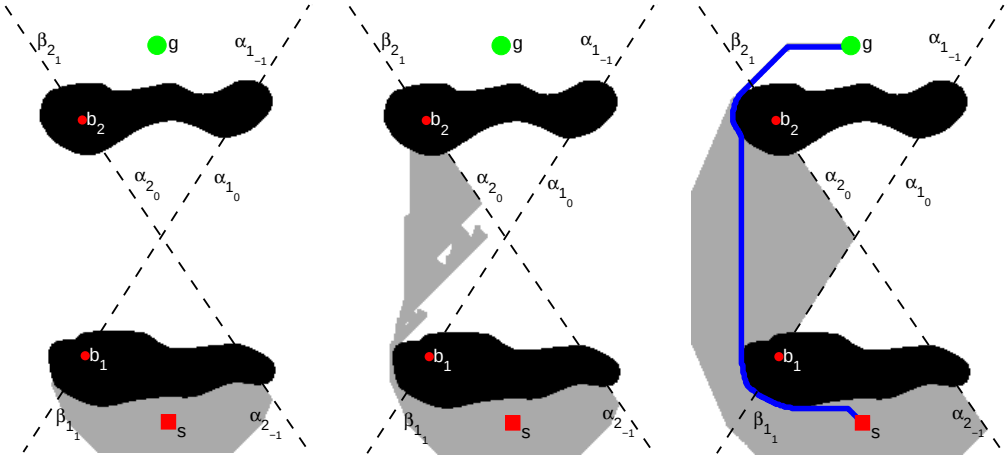


Figure 34: An HA* execution example in a simple workspace with two obstacles for homotopy class $\beta_{1_1}\beta_{2_1}$.

$[Q(n), q']$ with F sorted by distance[1]. Then the UpdateH (line 16) generates the new topological path according to the intersections. No intersections with F means that the explored configuration is in the same sub-wedge of the C-space and the function returns $P(n)$. If there are intersections and these intersections follow H, the function returns $P(n) \cup I$ in order to create a new node candidate n'. Whether $n'$is in the OPEN queue or not is then checked. If not, it is added to the queue with a priority $g(n')$ plus the heuristic $h(n', q_{goal})$ (line 20). If it is, and the cost $g(n)$ plus the cost of traversing from n to n', $c(n, n')$ is less than its current cost (line 21), $g(n')$ is set to this new lower value. This process is repeated until the n_{goal} is found or OPEN has no more nodes to be expanded.

Figure 34 depicts an example of an HA* execution in a discrete version of the workspace shown during the generation of the homotopy classes. The input homotopy class is $\beta_{1_1}\beta_{2_1}$. The algorithm explores the environment using A8 connectivity and the Euclidian distance as heuristic estimator. Figures 2.a,b depict the explored states in grey at two different moments of execution. Figure 2.c shows the final path obtained with backtracking from the goal to the start with the total number of cells explored. The solution is the shortest homotopic path for the input homotopy class. It is worth noting that cells are explored in only those areas that accomplish the class.

3.4.1.2 *Theoretical properties*

The HA* inherits several properties from the A* algorithm:

OPTIMAL. The algorithm computes the least cost path for a given homotopy class which can be used as a ground truth for other path planning algorithms that follow homotopy classes.

1 Notice that it is possible to intersect more than one segment of the reference frame depending on how close $Q(n)$ and q' are to the c point.

Algorithm 7 Homotopic A*

FindIntersections$([q, q'], F)$

1: $r \leftarrow \emptyset$
2: **for** $i \leftarrow 1$ **to** $|F|$ **do**
3: **if** $x \leftarrow Intersection([q, q'], F[i]) \neq null$ **then**
4: $r \leftarrow r \cup \{Edge(i), Distance(q, x)\}$
5: **end if**
6: **end for**
7: $r \leftarrow SortByDistance(r)$
8: **return** r

ComputePath(n_{start}, n_{goal}, F)

9: $OPEN \leftarrow \emptyset$; $V \leftarrow \emptyset$
10: $OPEN.push(n_{start})$
11: **while** $min_{n \in OPEN}(n \neq n_{goal})$ **do**
12: $n \leftarrow OPEN.top()$
13: $OPEN.pop()$
14: **for all** $q' \in Succ(Q(n))$ **do**
15: $I \leftarrow FindIntersections([Q(n), q'], F)$
16: **if** $H' \leftarrow UpdateH(P(n), I)$ **then**
17: $n' \leftarrow \{q', H'\}$
18: **if** $n' \notin OPEN$ **then**
19: $g(n') \leftarrow g(n) + c(n, n')$
20: $OPEN.push(n')$ with $g(n') + h(n', n_{goal})$
21: **else if** $g(n') > g(n) + c(n, n')$ **then**
22: $g(n') \leftarrow g(n) + c(n, n')$
23: **end if**
24: **end if**
25: **end for**
26: **end while**

HA*$(q_{start}, q_{goal}, H, F)$

27: $n_{start} \leftarrow \{q_{start}, \emptyset\}$; $n_{goal} \leftarrow \{q_{goal}, H\}$
28: $ComputePath(n_{start}, n_{goal}, F)$
29: **if** $min_{n \in OPEN}(n = n_{goal})$ **then**
30: publish solution
31: **end if**

COMPLETE. The completeness of the algorithm is ensured because when n_{goal} is not reachable, the algorithm explores all the nodes in the OPEN priority queue before returning that no path has been found. On the other hand, when n_{goal} can be reached, the HA* finds the node that accomplishes the homotopy class to follow in OPEN.

3.4.2 *Homotopic Rapidly-exploring Random Tree*

The Homotopic RRT (HRRT) is based on the goal-biased RRT algorithm which has been shown to be very efficient in time, even in complex workspaces (LaValle, 1999; Kim and Ostrowski, 2003). The algorithm allows a constrained growing of the tree only in those directions that satisfy a given homotopy

class. Before adding a new node into the tree, the topological path traversed is checked to ensure that it belongs to the homotopy class by computing the intersections of the path with the reference frame.

3.4.2.1 *Implementation*

The HRRT is detailed in Algorithm 8. It receives as input the start configuration q_{start}, the goal configuration q_{goal}, a candidate homotopy class to follow H and the reference frame F. The nodes on tree T are tuples that contain the configuration of the robot q and the topological path from q_{start} to q. These values are accessible through the functions Q and P respectively. Just like the RRT, the function `Extend` (line 31) iteratively extends tree T until the distance between the configuration of n_{new} ($Q(n_{new})$) and n_{goal} ($Q(n_{goal})$) is lower than a `distThreshold`.

The extension of the tree starts by selecting a random configuration q_{rand} from the C-space with the function `ComputeQRand`. Then, the `Nearest–Neighbor` function returns the nearest node $n_{nearest}$ regarding a random configuration q_{rand} by looking for the node whose topological path is closer to $P(n_{goal})$ (line 4). If there is more than one candidate, the node selected is the closest to the goal according to the Euclidean distance. After q_{new} is computed using the function `ComputeQNew`, `FindIntersections` (line 21) checks whether the segment $[Q(n_{nearest}), q_{new}]$ intersects with any segment in reference frame F[2]. The function returns the intersected edges sorted by distance from $Q(n_{nearest})$. Then, the function `UpdateH` (line 22) generates the new topological path H' according to the intersections. No intersections with F means that the tree grows in the $n_{nearest}$ sub-wedge and hence, the function returns $P(n_{nearest})$. If there are intersections and these intersections follow the topological path, the function returns $P(n_{nearest}) \cup I$ in order to create a new candidate node n_{new} to be added to the tree; otherwise a `null` path is returned and no node is added to the tree.

Figure 35 depicts an execution example of the HRRT with an exploration tree and the solution obtained with no post-processing. The homotopy class to follow is $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$. Notice that the algorithm constrains the expansion of the tree only into those zones that accomplish the input homotopy class, thus, in this example, segments $\alpha_{1_{-1}}$ or β_{2_1} in the reference frame do not have to be crossed by the tree.

3.4.2.2 *Theoretical Properties*

The HRRT has two important theoretical properties:

PERFORMANCE. The HRRT constrains the growing a tree only into those directions that satisfy a given homotopy class. With respect to the RRT, every time a new node is generated, it is checked whether the branch that connects the tree with the new node intersects with any segment of the

2 Notice that it is possible to intersect with more than one segment in the reference frame depending on the step between $n_{nearest}$ and q_{new} and how close these nodes are to the c point.

Algorithm 8 Homotopic RRT

NearestNeighbor(T, q_{rand})
1: $n \leftarrow T$; $d \leftarrow Distance(Q(n), q_{rand})$
2: **for all** $c \leftarrow T.Children()$ **do**
3: $[n', d'] \leftarrow NearestNeigbor(T, q_{rand})$
4: **if** $(|P(n')| > |P(n)|)$ **or** $(|P(n')| = |P(n)|$ **and** $d' < d)$ **then**
5: $n \leftarrow n'$; $d \leftarrow d'$
6: **end if**
7: **end for**
8: **return** $\{n, d\}$

FindIntersections$([q_{nearest}, q_{new}], F)$
9: $r \leftarrow \emptyset$
10: **for** $i \leftarrow 1$ **to** $|F|$ **do**
11: **if** $p \leftarrow Intersection([q_{nearest}, q_{new}], F[i]) \neq null$ **then**
12: $r \leftarrow r \cup \{Edge(i), Distance(q_{nearest}, p)\}$
13: **end if**
14: **end for**
15: $r \leftarrow SortByDistance(r)$
16: **return** r

Extend(T, n_{goal}, F)
17: $n_{new} \leftarrow \{\infty, null\}$
18: $q_{rand} \leftarrow ComputeQRand()$
19: $n_{nearest} \leftarrow NearestNeighbor(T, q_{rand})$
20: $q_{new} \leftarrow ComputeQNew(Q(n_{nearest}), q_{rand})$
21: $I \leftarrow FindIntersections([Q(n_{nearest}), q_{new}], F)$
22: $H' \leftarrow UpdateH(P(n_{nearest}), I\}$
23: **if** $(H' \neq null)$ **then**
24: $n_{new} \leftarrow \{q_{new}, H'\}$
25: $n_{nearest}.Add(n_{new})$
26: **end if**
27: **return** n_{new}

HRRT$(q_{start}, q_{goal}, H, F)$
28: $n_{new} \leftarrow \{q_{start}, \emptyset\}$; $n_{goal} \leftarrow \{q_{goal}, H\}$
29: $T.Add(n_{new})$
30: **while** $Distance(Q(n_{new}), Q(n_{goal})) > distThreshold$ **do**
31: $n_{new} \leftarrow Extend(T, n_{goal}, F)$
32: **end while**

reference frame. This extra computational load, assumed by the function FindIntersections, makes the HRRT no to reach the performance of the RRT, but ensures growth only into those regions that accomplish the homotopy class.

COMPLETENESS. When the HRRT is used together with the automated generation of the homotopy classes, the algorithm is complete because if the goal is not reachable, no homotopy classes will exist and, consequently, no paths will be generated. On the other hand, when the HRRT is executed to find a path of an arbitrary topological sequence, the algorithm

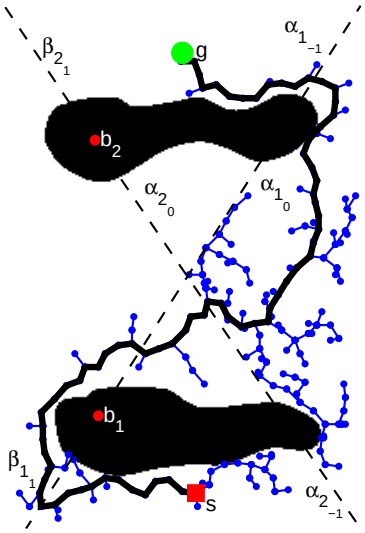


Figure 35: A path computed with the HRRT path planner for the homotopy class $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$.

is probabilistically complete because it is not ensured that a path will be found.

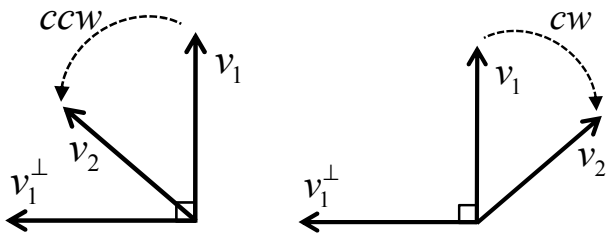
3.4.3 *Homotopic Bug*

The Homotopic Bug (HBug) is the third and last approach of this dissertation to generate paths according to a homotopy class. The algorithm is based on the Bug2. Essentially, it tries to follow directly the lower bound path obtained with the modified funnel algorithm which ensures that the homotopy class is being accomplished. However, as mentioned in Section 3.3, the segments in the reference frame constrain the regions the paths can go through, but do not take into account the shape of the obstacles. For this reason, the lower bound path may intersect with the obstacles. In such cases, the obstacle boundary is followed in a clockwise or counterclockwise direction according to the homotopy class until the lower bound path leaves the obstacle. This process is repeated for all the obstacles intersected by the lower bound path.

3.4.3.1 *Perpendicular Dot Product*

The perpendicular dot product is used to compute the direction to surround an obstacle while keeping the homotopy class. Formally, the perpendicular dot product, referred to as *perp dot product*, $\mathbf{v_1}^{\perp} \cdot \mathbf{v_2}$ for $\mathbf{v_1}$ and $\mathbf{v_2}$ vectors in the plane is a modification of the two-dimensional dot product in which $\mathbf{v_1}$ is replaced by the perpendicular vector rotated 90 degrees to the left, as defined by (Hill, 1994). It satisfies the identities:

$$\mathbf{v_1}^{\perp} \cdot \mathbf{v_2} = \|\mathbf{v_1}\| \, \|\mathbf{v_2}\| \sin(\theta) \tag{3.1}$$



a) Counterclockwise direction b) Clockwise direction

Figure 36: Perpendicular dot product.

$$(\boldsymbol{v_1^\perp} \cdot \boldsymbol{v_2})^2 + (\boldsymbol{v_1} \cdot \boldsymbol{v_2})^2 = \|\boldsymbol{v_1}\|^2 \|\boldsymbol{v_2}\|^2 \tag{3.2}$$

where θ is the angle from vector $\boldsymbol{v_1}$ to vector $\boldsymbol{v_2}$.

Based on Equation 3.1, the sign of the perpendicular dot product indicates the turning direction from $\boldsymbol{v_1}$ to $\boldsymbol{v_2}$: if $\boldsymbol{v_1^\perp} \cdot \boldsymbol{v_2}$ is greater than 0, the direction is counterclockwise (Figure 36.a); if it is less than 0, it is clockwise (Figure 36.b) and if it is 0, $\boldsymbol{v_1}$ and $\boldsymbol{v_2}$ are parallel.

3.4.3.2 *Implementation*

The HBug detailed in Algorithm 9 receives as input parameters the lower bound path P, a candidate homotopy class to follow H and the reference frame F. Notice that the first and last elements of P are the start (s) and goal (g) nodes respectively.

The algorithm is a three step process. First, the function BoundaryNodes checks the intersections of P with the obstacles in the C-space. Every time that P hits or leaves an obstacle, a boundary node is created. Each node contains the contact point c and the obstacle label k, which is the subindex of the point b_k that represents the obstacle in the reference frame. These parameters are accessible through the functions Q and Obst respectively. Then, ObstacleNodes computes the nodes O based on the boundary nodes N previously computed. Each obstacle node contains the first boundary node that hits obstacle n_h, the last node in its boundary without changing the obstacle n_l, and the direction d to surround the obstacle while following H (line 23). Finally, the function BuildPath creates the path P′ in the workspace by joining the boundary of each obstacle $o_i \in O$ from n_h to n_l with the direction d.

The direction d to surround an obstacle is set according to the direction of a hit node n_h towards its successor n_{h+1}[3] with respect to point b_k that represents the obstacle in the workspace in the reference frame. Notice that n_h and n_{h+1} are ensured to belong to the same obstacle since for any point that hits an obstacle there has to be another that releases it. The perpendicular dot product between vectors $(Q(n_h) - b_k)$ and $(Q(n_{h+1}) - b_k)$ computes the boundary following the direction (line 15). If the result is less than 0, the

3 When the lower bound path intersects with an obstacle only once, the n_{h+1} node is also the n_l node.

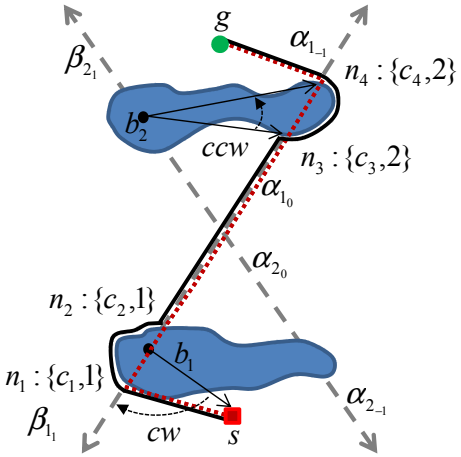


Figure 37: Path computed with the HBug path planner for the homotopy class $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$.

direction from n_h to n_{h+1} is counterclockwise; if it is greater than 0, the direction is clockwise.

The result of the perpendicular dot product can be 0 if the vectors $(Q(n_h) - b_k)$ and $(Q(n_{h+1}) - b_k)$ are parallel, which means that n_h, n_{h+1} and b_k belong to the same l_k in the reference frame (line 16). In such cases, d is obtained according to two conditions: the initial direction selected to cross l_k from the start point, and the number of times that l_k is crossed until α_k or β_k, denoted by χ_k, of the homotopy class, on which n_{h+1} relies, is reached. The initial direction is obtained with the dot product from the start s to the first χ_k with the same subindex as l_k[4] (line 17). The number of times that l_k is crossed depends on the number of χ_k found in the homotopy class from the beginning to the index i_k, which indicates the position of the χ_k that contains n_{h+1} (line 19).

Figure 37 depicts an example scenario where the HBug is applied. The homotopy class to follow is $\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$. The dashed line represents its lower bound path, which intersects the first obstacle generating two boundary nodes, n_1 and n_2, both located on line l_1 in the reference frame. The point that represents the obstacle is b_1, also on l_1, which makes the perpendicular dot product between $(Q(n_1) - b_1)$ and $(Q(n_2) - b_1)$ unable to set the direction ($d = 0$). Therefore, using the start point s and a point of the edge β_{1_1}, the initial direction is set clockwise (cw). The last edge involved in this situation is α_{1_0}, located in the second position in the homotopy class. The number of edges with subindex 1 up to this position is 2, thus, the direction is not changed. Then, the lower bound path intersects the second obstacle in n_3 and n_4. Using the base point b_2, the perpendicular dot product sets the direction as counterclockwise (ccw). Finally, the path is composed from s to g with the boundaries of obstacle 1 (from n_1 to n_2) and obstacle 2 (from n_3 to n_4) joined by straight lines.

4 Notice that the start point cannot be in line l_k in the reference frame since the perpendicular dot product would also be 0.

Algorithm 9 Homotopic Bug

BoundaryNodes(P)

1: $N \leftarrow \emptyset$
2: **for** $i \leftarrow 1$ **to** $|P|-1, p_i \in P$ **do**
3: $C \leftarrow$ ContourPoints(p_i, p_{i+1})
4: **for all** $j \leftarrow 1$ **to** $|C|, c_j \in C$ **do**
5: $k \leftarrow$ Label(c_j)
6: $N \leftarrow N \cup \{c_j, k\}$
7: **end for**
8: **end for**
9: **return** N

ObstacleNodes(N, H, F)

10: $O \leftarrow \emptyset$
11: $h \leftarrow 1$
12: **while** $n_h \in N/h < |N|-1$ **do**
13: $n_l \leftarrow$ last $n_j \in N/j > h$ without changing $Obst(n_h)$
14: $b_k \leftarrow$ point of $Obst(n_h)$ in F
15: $d \leftarrow (Q(n_h) - b_k)^{\perp} \cdot (Q(n_{h+1}) - b_k)$
16: **if** $d = 0$ **then** {parallel}
17: $d \leftarrow (s - b_k)^{\perp} \cdot$ (point of 1^{st} $\chi_k \in H - b_k$)
18: $i_k \leftarrow$ index of $\chi_k \in H$ where n_{h+1} relies on
19: **if** $|\chi_k| \in H_{1..i_k}$ is *even* **then**
20: switch d
21: **end if**
22: **end if**
23: $O \leftarrow O \cup \{n_h, n_l, d\}$
24: $h \leftarrow l + 1$
25: **end while**
26: **return** O

BuildPath(O)

27: $P' \leftarrow \emptyset$
28: **for** $i \leftarrow 1$ **to** $|O|, o_i \in O$ **do**
29: $P' \leftarrow P' \cup$ Boundary(o_i)
30: **end for**
31: **return** P'

HBug(P, H, F)

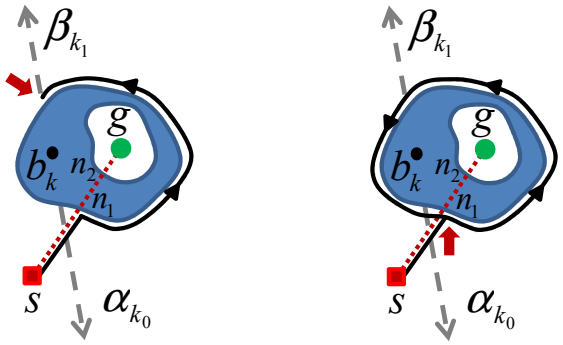
32: N ← BoundaryNodes(P)
33: O ← ObstacleNodes(N, H, F)
34: $P' \leftarrow$ BuildPath(O)

3.4.3.3 *Theoretical properties*

The HBug algorithm has been designed to be used together with the automated computation of the homotopy classes. Nevertheless, the algorithm itself has several properties:

COMPLETENESS. As a Bug-based algorithm, when there is a solution path the HBug finds it. On the other hand, when there is no solution, the function BuildPath tries to follow the boundary of the obstacle that does not allow the solution to exists. However, by doing this, it crosses segments

in the reference frame that do not describe the specific homotopy class that has to be followed. To avoid checking the intersections with the reference frame, it is also possible to find that no solution exists when the algorithm again reaches the hit point after surrounding the whole obstacle. Figure 38 depicts both situations for the homotopy class α_{k_0}.



a) Intersection of the reference frame b) Findind a visited contact point

Figure 38: Two ways to check completeness with HBug algorithm for the homotopy class α_{k_0}.

NO OPTIMAL. The HBug algorithm does not compute the optimal solution. The cost of the solution for a specific homotopy class depends on how far the subsegments in the reference frame are from the boundary of the obstacles.

PERFORMANCE. During the path computation, the HBug only considers those nodes of the C-space that belong to the lower bound path, which is already computed, as long as it does not traverse an obstacle. When it does, the algorithm only looks for the free space around the intersected obstacle. Therefore, the number of explored nodes in the C-space is drastically reduced when compared with the HA* and HRRT algorithms. This property has been reported in experimental results shown in Chapter 6.

UPPER BOUND. As with the Bug1 and Bug2 algorithms described in Chapter 2, it is possible to establish an upper bound for the path length. The ideal path would be the lower bound path length LB, which can only be achieved when it does not intersect with an obstacle in the C-space. However, most of the homotopy classes require the circumnavigation of the n obstacles in the environment that cross with the lower bound path. Depending on the shape of the obstacles, the worst possible case would be to follow almost the whole perimeter p of the obstacles involved. Thus, the upper bound is

$$UB_{HBug} \leqslant LB + \sum_{i=1}^{n} p_i \tag{3.3}$$

3.5 SUMMARY

Given an environment with obstacles, in this chapter we have proposed a method to generate homotopy classes from a start to an end position that can be followed in the C-space. The method starts by constructing a reference frame which establishes the topological relationships between the obstacles in the metric space. This frame allows us to describe any path according to a sequence of intersected segments. The reference frame is used to set up a topological graph that allows the systematic generation of homotopy classes by means of a graph-based search algorithm. In order to contain the number of classes generated, only those that neither self-intersect nor are duplicated are allowed to be generated. The computed homotopy classes are sorted according to a lower bound criterion which estimates their quality before generating a path.

The next step consists of turning the topological information provided by the homotopy classes into metric paths in the C-space. Three path planners from three different approaches have been proposed to achieve this goal. The first is the HA*, a graph-based search algorithm based on the A*. This algorithm performs an exhaustive exploration of the C-space according to a heuristic function. When the exploration of a node implies an intersection with the reference frame, it is checked whether the segment has to be crossed according to the input homotopy class. In such cases, it is explored, otherwise it is discarded. The result of the HA* is the optimal path for the input homotopy class.

The second path planner is the HRRT based on the goal-biased RRT algorithm to perform a fast exploration of the C-space. Following the same idea as the HA*, the HRRT constrains the growing of the tree only in those directions that satisfy an input homotopy class. Before adding a new node into the tree, the topological path traversed is checked to ensure that it belongs to the homotopy class by computing the intersections in the path with the reference frame.

The last proposed path planner is a Bug-based algorithm called HBug. This algorithm tries to follow the lower bound path, ensuring the accomplishment of the homotopy class. Since the segments in the reference frame constrain the regions the paths can go through but do not take into account the shape of the obstacles, the lower bound path may intersect with obstacles. In this situation, the obstacle boundary is followed in the direction that accomplishes the homotopy class until the lower bound path leaves the obstacle. This process is repeated for all the obstacles intersected by the lower bound path. As a result, the HBug generates a suboptimal path that follows the homotopy class very quickly because the explored C-space is constrained between the nodes of the lower bound path (already explored) and the partial boundary of the obstacles intersected. Moreover, an intersection check with the reference frame is not required every time a node is explored.

4

LOCAL MAP BUILDING

As stated in Chapter 1, the main goal of our map building algorithm is to compute a map with which the robot can perform path planning to generate safe trajectories while avoiding obstacles. Nevertheless, robot localization and map building is a research discipline by itself. Over the few last years, great efforts have been made to provide a coupled solution to the Simultaneous Localization And Mapping (SLAM) problem (Bailey and Durrant-Whyte, 2006; Durrant-Whyte and Bailey, 2006). In underwater applications, autonomous robot localization and map building is an even harder problem to solve and is beyond the scope of this work. Since this research project focuses on path planning, a simple approach for local map building is adopted as a support for the proposed algorithms. Hence, for the sake of simplicity, an uncoupled approach for the localization and the map building problems is adopted. First, the localization problem is addressed through sonar scan matching (Section 4.1), and then the map, with which the robot can perform path planning is generated using an Occupancy Grid Map (OGM) algorithm (Section 4.2).

4.1 SCAN MATCHING

In order to generate a local map to perform path planning, the robot has to be localized, which can be easily done by means of dead-reckoning. Then, the exteroceptive robot measurements can be compounded with the robot pose to iteratively map the robot's surroundings. Even though a local map works as a short term memory, if the dead-reckoning drifts very fast, the map may become distorted even in a very short period of time. Hence, inspired in mobile robotics domain (Mínguez et al., 2004), it is proposed an approach that uses scan matching to improve the dead-reckoning estimate. Scan matching is an interesting method since it does not need an a priori map of the environment, and is able to work in both structured and unstructured environments, the latter being the most common underwater.

Although a large literature exists reporting successful applications of scan matching with mobile robots (Lu and Milios, 1994; Mínguez et al., 2005; Montesano et al., 2005; Burguera et al., 2007), very few attempts have been done to use sonar scan matching in underwater applications. In (Castellani et al., 2004) a non-probabilistic variation of Iterative Closest Point (ICP) (Besl and McKay, 1992) is proposed to achieve on-line performance for registering multiple views captured with a 3D acoustic camera. (Silver et al., 2004) proposed the use of a particle filter to deal with the sonar noisy data. In (Hernández et al., 2009b) we proposed the MSISpIC to deal with data gathered by an AUV using an Mechanical Scanned Imaging Sonar (MSIS). This algorithm is our scan matching proposal and will be detailed in this chapter. Essentially, it is an extension of

the Probabilistic Iterative Correspondance (pIC) algorithm (Montesano et al., 2005) that takes into account the distortions in the acoustic image due to the vehicle's motion. In (Bülow et al., 2010) a 2D scan matching method based on spectral registration of rendered scan data is presented, which has been recently extended to work with 3D sonar images (Bülow and Birk, 2011).

Scan matching can delay the drift of dead-reckoning estimation. However, it is not able to bound it over time. For this reason, recent studies have been developed to combine SLAM and sonar scan matching underwater. In (Roman and Singh, 2005), an ICP variant is used to register bathymetric sub-maps gathered with a multibeam sonar profiler. (Mallios et al., 2010b) used the MSISpIC to develop an algorithm that incorporates point-to-point scan matching in an Augmented State EKF (ASEKF) to bound the drift. The algorithm was tested in a dataset gathered in a man-made marina environment. With the same dataset, (Burguera et al., 2010) proposed an iterated Extended Kalman Filter (EKF) to estimate the trajectory of the AUV.

This section is organized as follows. First, Section 4.1.1 formally describes the scan matching problem. Next, Section 4.1.2 details the scan matching algorithm adopted from the mobile robotics domain to improve AUV localization. Then, Section 4.1.3 descibes the method we propose to gather full scans of the environment with the sonar sensors our vehicles are equipped with. Finally, Section 4.1.4 provides a full description of the MSISpIC, the scan matching algorithm we propose, which takes into account the uncertainties of the sensor and the dead-reckoning estimation.

4.1.1 *Problem Definition*

Assuming we have a robot equipped with sensors that provide an estimation of displacement through dead-reckoning as well as with a range and bearing sensor to perceive the environment, the scan matching problem can be defined as follows:

Let

- $S_{ref} = \{\mathbf{r}_1, \mathbf{r}_2, ..., \mathbf{r}_n\}$ be the reference scan, a set of n measurements, represented by Cartesian points, gathered with a range and bearing sensor at the reference frame $\{R\}$.
- $S_{new} = \{\mathbf{n}_1, \mathbf{n}_2, ..., \mathbf{n}_m\}$ be the new scan, a set of m measurements, represented by Cartesian points, gathered with a range and bearing sensor at the new reference frame $\{N\}$.
- $\mathbf{q}_0 = (x_0, y_0, \theta_0)$ be an initial guess of the robot's relative displacement between the two consecutive gathered scans S_{ref} and S_{new}.

The goal of scan matching algorithms is to estimate the robot's motion that maximizes the overlap between S_{ref} and S_{new}, using as the initial motion $\mathbf{q}_0$ in order to to obtain a better estimation of the real displacement $\mathbf{q} = (x, y, \theta)$. Figure 39 illustrates this process.

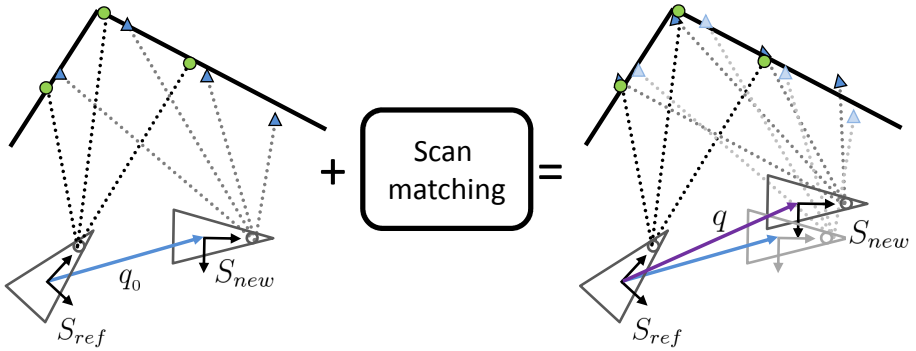


Figure 39: The scan matching problem.

4.1.2 *Related Work*

This section describes the ICP (Besl and McKay, 1992) and the pIC (Montesano et al., 2005) algorithms. The ICP is a well-known scan matching reference which is the base of the pIC, whereas our scan matching algorithm is based on the pIC. Both techniques have been applied with success to mobile robots which are usually equipped with laser-based range sensors that provide fast and accurate measurements. However, underwater robots use sonar-based sensors which usually gather data slower and are not as accurate as laser-based ones. The solution we propose to this problem will be detailed in Section 4.1.3

4.1.2.1 *Iterative Closest Point*

The Iterative Closest Point (ICP) (Besl and McKay, 1992) is an algorithm originally developed for computer vision purposes. It addresses scan matching with a two step iterative process. First, at each iteration, there is an association step in which the correspondences between scans are computed using a rough displacement estimation in terms of the Euclidian distance. Next, a minimization error process is applied in order to compute a new estimation of this displacement. The process is repeated using the prior displacement estimation until convergence.

The algorithm receives as an input a reference scan S_{ref} with points $\mathbf{r}_i$ (where $i = 1..n$), a new scan S_{new} with points $\mathbf{n}_j$ (where $j = 1..m$) and a relative displacement estimation $\mathbf{q} = (x, y, \theta)$ between them. The goal of the ICP is to compute the corrected motion between the two scans $\mathbf{q}_{ICP}$ by performing the following steps at each iteration k:

1. Transform each point of S_{new} to the coordinate system of S_{ref} using the current estimation $\mathbf{q}_k$. For each $\mathbf{n}_j$, compute its correspondence point $\mathbf{c}_j$ of S_{ref}, which is the closest point among those with a distance lower than a given threshold d_{min}.

$$\mathbf{c}_j = arg\ min_{\mathbf{r}_i \in S_{ref}} \{\|\mathbf{r}_i - \mathbf{q}_k \oplus \mathbf{n}_j\| \leqslant d_{min}\}$$

Point $\mathbf{c}_j$ is the pairing of $\mathbf{n}_j$ if, and only if, $\|\mathbf{r}_i - \mathbf{q}_k \oplus \mathbf{n}_j\| \leqslant d_{min}$. Otherwise $\mathbf{c}_j$ does not have a pairing in S_{new}. At the end of this step, the correspondence set C, compounded of tuples $\{\mathbf{c}_j, \mathbf{n}_j\}$, is obtained. C is formally defined as:

$$C = \{\{\mathbf{c}_j, \mathbf{n}_j\} \in S_{ref} \times S_{new} / \mathbf{c}_j = arg\ min_{\mathbf{r}_i \in S_{ref}} \{\|\mathbf{r}_i - \mathbf{q}_k \oplus \mathbf{n}_j\| \leqslant d_{min}\}\}$$

2. Compute the displacement $\mathbf{q}_{min}$ which minimizes the mean square error between C elements:

$$\mathbf{q}_{min} = arg\ min_{\mathbf{q}_k} \sum_{i=1}^{|C|} \|\mathbf{c}_i - \mathbf{q}_k \oplus \mathbf{n}_i\|^2$$

Convergence is reached when the global alignment error between S_{ref} and S_{new} for the estimated global displacement is less than a given threshold. Another convergence condition appears when the difference between the displacement estimations in two consecutive iterations is smaller than a certain threshold. If there is convergence, the estimated value $\mathbf{q}_{min}$ is returned as $\mathbf{q}_{ICP}$. Otherwise, another iteration with $\mathbf{q}_{k+1} = \mathbf{q}_{min}$ is required.

Although the ICP is a reference within scan matching techniques, it uses the Euclidian distance in the association step and in the minimization process. This represents a limitation since it does not take into account the sensor's rotation.

4.1.2.2 *Probabilistic Iterative Correspondence*

The Probabilistic Iterative Correspondance (pIC) algorithm (Montesano et al., 2005) is a statistical extension of the ICP. The ICP does not model the uncertainty of the sensor measurements. Because of this, if the scan data is very noisy, two statistically compatible points could appear far enough, in terms of the Euclidean distance. This situation might prevent a possible association or even generate a wrong one. The pIC algorithm models the relative displacement $\mathbf{q}$ as well as the observed points in both scans $\mathbf{r}_i$ and $\mathbf{n}_i$ as random Gaussian variables (r.g.vs). Whereas the geometric ICP algorithm uses the closest point rule to find the correspondence for a point in the new scan, the pIC algorithm first computes the set of compatible points in terms of the Mahalanobis distance, and then the virtual *expected* compatible point to be used as the correspondence (Figure. 40).

The pIC algorithm is described in pseudocode in Algorithm 10. The inputs are the reference scan S_{ref} with points $\mathbf{r}_i$ (where $i = 1..n$), the new scan S_{new} with points $\mathbf{n}_j$ (where $j = 1..m$) and the initial relative displacement estimation $\mathbf{q}$ with its covariance $\mathbf{P}_\mathbf{q}$. The following procedure is iteratively executed until convergence. First, the points of the new scan $\mathbf{n}_j$ are compounded with the

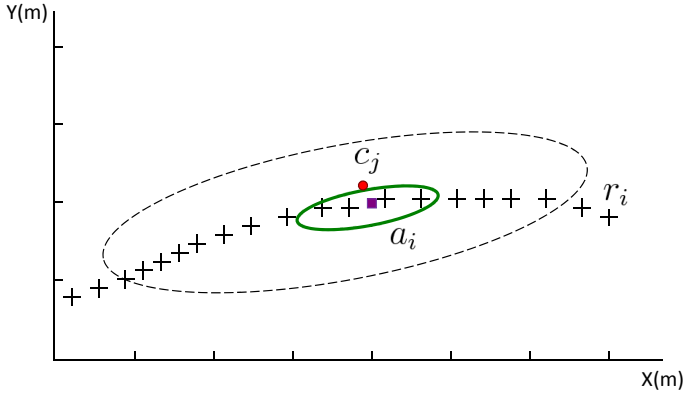


Figure 40: pIC correspondence computation. The large ellipse contains all the statistically compatible points and the squared point represents the correspondence with its uncertainty (small ellipse).

current estimation of the robot's displacement $\mathbf{q}_k$ (line 5). The result $\mathbf{c}_j$ are the points of the new scan referenced to the reference frame. Then, for each point $\mathbf{c}_j$, a set A_j of all the compatible points in the reference scan S_{ref} is established using a compatibility test over the squared Mahalanobis distance:

$$D_M^2 = (\mathbf{r}_i - \mathbf{c}_j)\mathbf{P}_{ij}^{-1}(\mathbf{r}_i - \mathbf{c}_j)^T \tag{4.1}$$

The next step consists of computing the virtual association point $\mathbf{a}_j$ as the expectancy over the random variable defined by the set A_j (line 7). To do so, it is necessary to evaluate the probability $p(r_i = c_j)$ of each r_i, for being the correct pairing for c_j, whose error is defined as a r.g.v.

$$\boldsymbol{e}_{ij} = \mathbf{r}_i - \mathbf{q}_k \oplus \mathbf{n}_j$$

$$\boldsymbol{e}_{ij} \cong N(\hat{\mathbf{r}}_i - \hat{\mathbf{q}}_k \oplus \hat{\mathbf{n}}_j, \mathbf{P}_{\boldsymbol{e}_{ij}})$$

$$\mathbf{P}_{\boldsymbol{e}_{ij}} = \mathbf{P}_{\mathbf{r}_i} + \mathbf{J}_{1\oplus}\mathbf{P}_{\mathbf{q}}\mathbf{J}_{1\oplus}^T + \mathbf{J}_{2\oplus}\mathbf{P}_{\mathbf{n}_j}\mathbf{J}_{2\oplus}^T$$

where:

$$\mathbf{q}_k \equiv N(\hat{\mathbf{q}}_k, \mathbf{P}_{\mathbf{q}})$$

$$\mathbf{n}_j \equiv N(\hat{\mathbf{n}}_j, \mathbf{P}_{\mathbf{n}_j})$$

$$\mathbf{r}_i \equiv N(\hat{\mathbf{r}}_i, \mathbf{P}_{\mathbf{r}_i})$$

$\mathbf{J}_{1\oplus}$ and $\mathbf{J}_{2\oplus}$ are the well-known Jacobian matrixes of the compounding operation with respect to the first and second arguments respectively (Smith et al., 1990).

then $p(\mathbf{r}_i = \mathbf{c}_j) = p(\boldsymbol{e}_{ij} = 0)$ can be computed as follows:

$$p(\boldsymbol{e}_{ij} = 0) = \frac{f_{\boldsymbol{e}_{ij}}(\mathbf{r}_i - \mathbf{c}_j)}{\sum_{\mathbf{r}_i \in A_j} f_{\boldsymbol{e}_{ij}}(\mathbf{r}_i - \mathbf{c}_j)} \tag{4.2}$$

where $f_{\boldsymbol{e}_{ij}}$ is the probability density function of $\boldsymbol{e}_{ij}$ r.g.v. Once $\hat{\mathbf{a}}_j$ has been computed as the expectance of r_i over A_j, a similar procedure can be used to estimate its uncertainty $\mathbf{P}_{\mathbf{a}_j}$ before computing the error covariance $\mathbf{P}_{\boldsymbol{e}_j}$ of the matching error $(\hat{\mathbf{a}}_j - \hat{\mathbf{c}}_j)$. Then, it is possible to estimate the displacement $\hat{\mathbf{q}}_{min}$ which minimizes the mean square error of the Mahalanobis distance (Bar-Shalom and Fortman, 1998) between $\hat{\mathbf{a}}_j$ and $\hat{\mathbf{c}}_j$ (line 11). This is done through the Least Squares minimization method. If there is convergence, the function returns, otherwise another iteration is required.

Algorithm 10 Probabilistic Iterative Correspondance

pIC$(S_{ref}, S_{new}, \hat{\mathbf{q}}, \mathbf{P}_{\mathbf{q}})$

1: $k \leftarrow 0$
2: $\hat{\mathbf{q}}_k \leftarrow \hat{\mathbf{q}}$
3: **repeat**
4: **for** $j \leftarrow 1$ **to** $|S_{new}|$ **do**
5: $\hat{\mathbf{c}}_j \leftarrow \hat{\mathbf{q}}_k \oplus \hat{\mathbf{n}}_j$
6: $A_j \leftarrow \{\mathbf{r}_i \in S_{ref} / D^2_M(\mathbf{r}_i, \mathbf{c}_j) \leqslant \chi^2_{2,\alpha}\}$
7: $\hat{\mathbf{a}}_j \leftarrow \sum_{\mathbf{r}_i \in A_j} \hat{\mathbf{r}}_i p(\mathbf{r}_i = \mathbf{c}_j)$
8: $\mathbf{P}_{\mathbf{a}_j} \leftarrow \sum_{\mathbf{r}_i \in A_j} [(\hat{\mathbf{r}}_i - \hat{\mathbf{a}}_j)(\hat{\mathbf{r}}_i - \hat{\mathbf{a}}_j)^T] p(\mathbf{r}_i = \mathbf{c}_j)$
9: $\mathbf{P}_{\boldsymbol{e}_j} \leftarrow \mathbf{P}_{\mathbf{a}_j} + \mathbf{J}_{1\oplus} \mathbf{P}_{\mathbf{q}} \mathbf{J}^T_{1\oplus} + \mathbf{J}_{2\oplus} \mathbf{P}_{\mathbf{n}_j} \mathbf{J}^T_{2\oplus}$
10: **end for**
11: $\hat{\mathbf{q}}_{min} \leftarrow \arg\min_{\mathbf{q}} \left\{ \sum_j \left((\hat{\mathbf{a}}_j - \hat{\mathbf{c}}_j)^T \mathbf{P}^{-1}_{\boldsymbol{e}_j} (\hat{\mathbf{a}}_j - \hat{\mathbf{c}}_j) \right) \right\}$
12: **if** Convergence() **then**
13: $\hat{\mathbf{q}}_{pIC} \leftarrow \hat{\mathbf{q}}_{min}$
14: **else**
15: $\hat{\mathbf{q}}_{k+1} \leftarrow \hat{\mathbf{q}}_{min}$
16: $k++$
17: **end if**
18: **until** Convergence() **or** $k \geqslant maxIterations$
19: **return** $\hat{\mathbf{q}}_{pIC}$

4.1.3 *Scans Generation using an MSIS*

The pIC algorithm was conceived to be used with a laser range finder in structured environments. These sensors gather a full range scan of the environment almost instantaneously and thus, the displacement of the vehicle is negligible. However, in underwater robotics, commercially available scan sensors are based on acoustics. Most of these sensors have a mechanical head that rotates at fixed angular steps. At each step, a beam is emitted and received, measuring ranges to the obstacles found in its trajectory. Because of this, getting a complete scan lasts few seconds while the vehicle is moving, generating deformed

scans. Therefore, it is necessary to correct them taking into account the vehicle pose when the beam was grabbed.

This section describes the process of gathering a full scan, as the one expected by the pIC algorithm as the input, using the sensors of the experimental platforms used in this work. To perceive the environment an MSIS is used, and a combination of Doppler Velocity Log (DVL) and Motion Reference Unit (MRU) readings to estimate the vehicle's motion. For a detailed explanation of the vehicles and their sensors see Chapter 5.

4.1.3.1 *Perceiving the environment with an MSIS*

An MSIS returns a polar acoustic image composed of beams. Each beam has a particular bearing angle value and a set of intensity measurements. The angle corresponds to the orientation of the sensor head when the beam was emitted. The beam itself contains an acoustic linear image which corresponds to the intensity levels of beam's echoes perceived by the sensor at a certain distance. The beam information is returned as an array of acoustic intensities. Figure 41 illustrates an example of the echoes strength intensity levels of one single beam according to the obstacles in its trajectory and Figure 42 shows a full example scan gathered in a water tank environment.

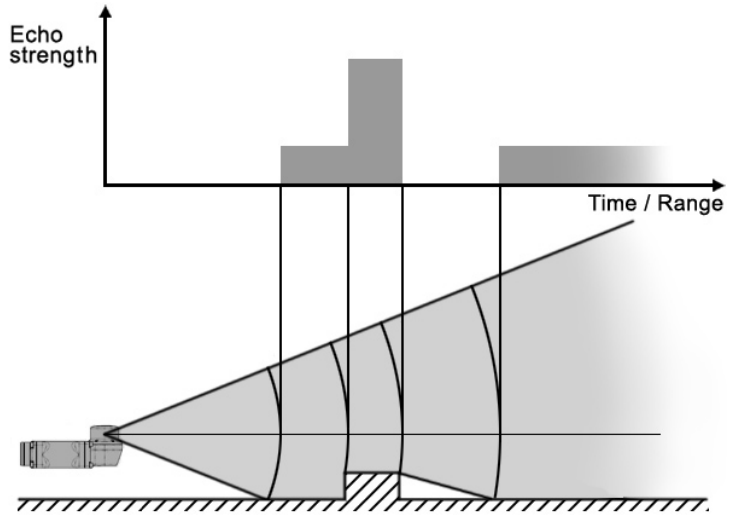


Figure 41: Generation of an acoustic beam. Extracted from (Ribas et al., 2010)

Since the pIC algorithm is intended to work with range scans, a segmentation process to obtain range and bearing data is required. Each beam gathered is segmented using a predefined threshold to compute the intensity peaks. Due to the noisy nature of the acoustic data, a minimum distance between peaks criteria is also applied. Therefore, the positions finally considered are those corresponding to high intensity values above the threshold with a minimum *distance* between each other. Figure 43 illustrates this process.

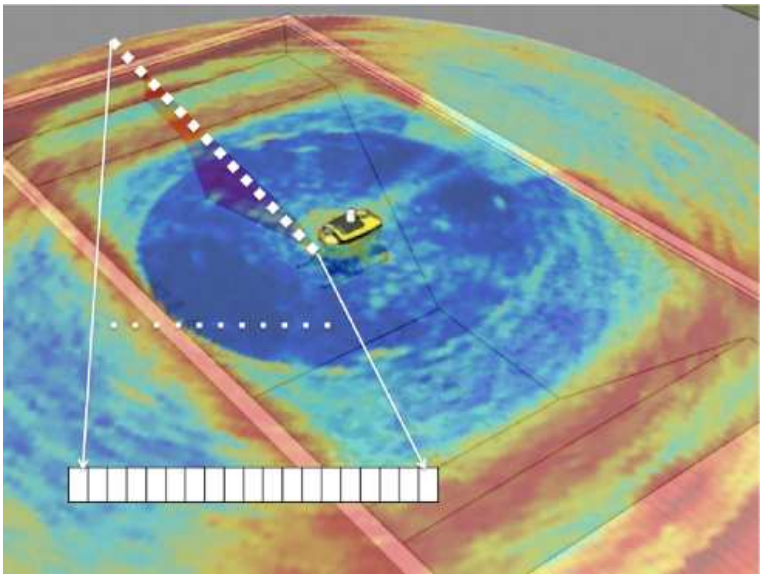


Figure 42: Interpretation of a polar image gathered with an MSIS. The current beam is detailed.

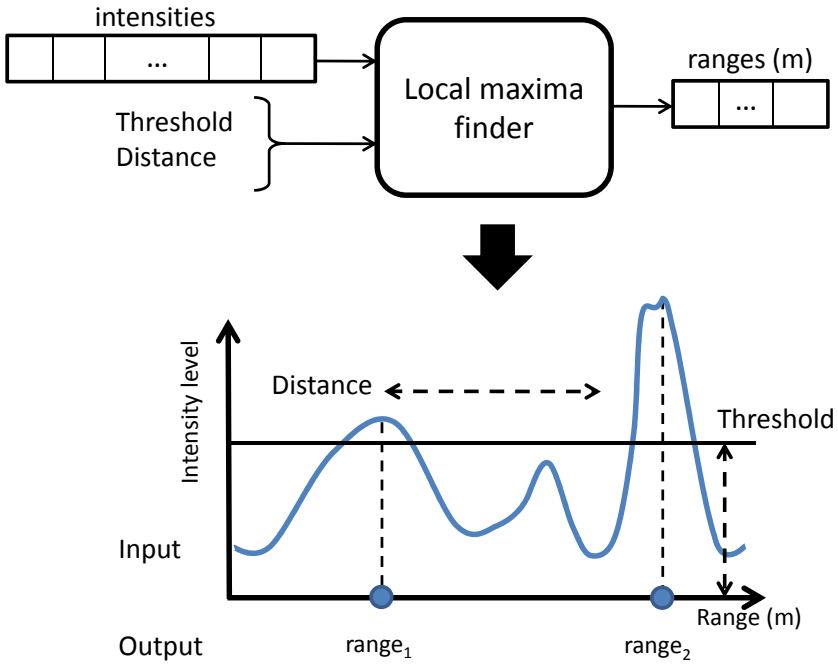


Figure 43: A peaks detector for an MSIS beam.

4.1.3.2 *Relative Vehicle Localization*

The pIC algorithm needs a complete scan to be registered with the previous one in order to estimate the vehicle's displacement. Since an MSIS needs a considerable period of time to obtain a complete scan, if the vehicle does not remain static, the vehicle's motion induces a distortion in the acoustic image. To deal with this problem, it is necessary to know the vehicle's pose at the beam reception time with a relative localization system. All the range measurements belonging to the same scan have to be referenced to the same coordinate system. Figure 44 depicts an example of a distorted acoustic image due to the motion of the vehicle while perceiving the environment and the resultant corrected image using a relative localization system.

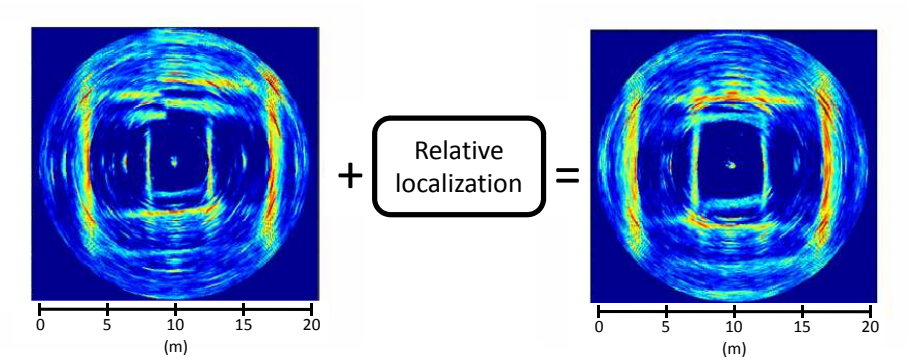


Figure 44: The distortion produced by the displacement of the robot in the scenario in Figure 42 while acquiring data can be corrected using the relative localization system.

The localization system for gathering scans is a slight modification of the navigation system described in (Ribas et al., 2006). In this system, an MRU provides heading measurements and a DVL unit which includes attitude, heading and depth sensors is used to estimate the vehicle's pose while the scan is being gathered. On one hand, the beams acquisition of the MSIS depends on the fire up range (the larger the distance the slower the acquisition). On the other hand, the DVL readings arrive at a frequency from 1.5Hz to 3Hz, depending on the sensor model, while the MRU returns data at 10Hz. Since all data is gathered asynchronously, an EKF is used to estimate the vehicle's 6DoF pose whenever a sonar beam is read. DVL and MRU readings are used asynchronously to update the filter. In order to predict the robot motion between the DVL measurements, a simple 6DoF constant velocity kinematics model is used.

The information of the system at step k is stored in the state vector $\mathbf{x}_k$ with estimated mean $\hat{\mathbf{x}}_k$ and covariance $\mathbf{P}_k$ defined as follows:

$$\hat{\mathbf{x}}_k = \left[\hat{\boldsymbol{\eta}}^B, \hat{\boldsymbol{\nu}}^R\right]^T \quad \mathbf{P}_k = E\left[\left(\mathbf{x}_k - \hat{\mathbf{x}}_k\right)\left(\mathbf{x}_k - \hat{\mathbf{x}}_k\right)^T\right] \tag{4.3}$$

where:

$$\boldsymbol{\eta}^B = [x, y, z, \phi, \theta, \psi]^T, \quad \boldsymbol{\nu}^R = [u, v, w, p, q, r]^T \tag{4.4}$$

As defined in (Fossen, 1994), η^B is the position and attitude vector referenced to a base frame B, and ν^R is the linear and angular velocity vector referenced to the robot's coordinate frame R. The coordinate frame B is chosen coincident with the initial reference frame I but oriented to the north, hence the compass measurements can be integrated in a straight forward manner.

The vehicle's movement prediction is performed using the 6DoF kinematic model:

$$\mathbf{x}_k = f(\mathbf{x}_{k-1}, \boldsymbol{w}_k) = \begin{bmatrix} \eta_k^B \\ \nu_k^R \end{bmatrix} = \begin{bmatrix} \eta_{k-1}^B + J(\eta_{k-1}^B) \left[\nu_{k-1}^R \Delta T + \frac{1}{2} w_k \Delta T^2 \right] \\ \nu_{k-1}^R + w_k \Delta T \end{bmatrix} \tag{4.5}$$

$$J(\eta) = \begin{bmatrix} c\psi c\theta & c\psi s\theta s\phi - s\psi c\phi & c\psi s\theta c\phi + s\psi s\phi & 0 & 0 & 0 \\ s\psi c\theta & s\phi s\psi s\theta + c\psi c\phi & s\psi s\theta c\phi - s\phi c\psi & 0 & 0 & 0 \\ -s\theta & c\theta s\phi & c\theta c\phi & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & s\phi t\theta & c\phi t\theta \\ 0 & 0 & 0 & 0 & c\phi & -s\phi \\ 0 & 0 & 0 & 0 & s\phi/c\theta & c\phi/c\theta \end{bmatrix} \tag{4.6}$$

Although in this model the velocity is considered to be constant, in order to allow for slight changes, a velocity perturbation modeled as the integral of a stationary white noise w_k is introduced. The covariance matrix $\mathbf{Q}_k$ of this acceleration noise is diagonal and in the order of magnitude of the maximum acceleration increment that the vehicle may experience over a sample period.

$$E[w_k] = 0, \quad E[w_k w_j^T] = \delta_{k_j} \mathbf{Q} \tag{4.7}$$

Hence, w_k is the acceleration noise which is integrated and added in velocity, being nonlinearly propagated to the position. Finally, the model prediction and update is carried out as detailed below:

PREDICTION. The estimate of the state is obtained through function f, described in Equation 4.3:

$$\hat{\mathbf{x}}_k = f(\hat{\mathbf{x}}_{k-1}) \tag{4.8}$$

and its covariance matrix as:

$$\mathbf{P}_k = \mathbf{F}_k \mathbf{P}_{k-1} \mathbf{F}_k^T + \mathbf{G}_k \mathbf{Q}_k \mathbf{G}_k^T \tag{4.9}$$

where $\mathbf{F}_k$ and $\mathbf{G}_k$ are the Jacobian matrices of partial derivatives of the non-linear model function f with respect to the state $\mathbf{x}_k$ and the noise w_k, respectively.

UPDATE USING DVL MEASUREMENTS. The model prediction is updated by the standard Kalman filter equations each time a new DVL measurement arrives:

$$z_{DVL,k} = [u_b, v_b, w_b, u_w, v_w, w_w, \phi_a, \theta_a, \psi_c, z_{depth}]^T \tag{4.10}$$

where subindex b stands for the bottom tracking velocity, w for the through water velocity, a for attitude and c represents the compass. The measurement model is:

$$z_{DVL,k} = \mathbf{H}_{DVL,k} x_{k|k-1} + v_k \tag{4.11}$$

$$\mathbf{H}_{DVL} = \begin{bmatrix} 0_{3\times3} & 0_{3\times3} & \mathbf{I}_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & \mathbf{I}_{3\times3} & 0_{3\times3} \\ 0_{3\times3} & 0_{3\times3} & 0_{3\times3} & 0_{3\times3} \\ 0\ 0\ 1 & 0_{1\times3} & 0_{1\times3} & 0_{1\times3} \end{bmatrix} \tag{4.12}$$

Notice that the DVL has a pressure sensor which provides precise depth measurements (z_{depth}). The measurement noise v_k is a gaussian zero-mean white noise:

$$E[v_k] = 0, \quad E[v_k v_j^T] = \delta_{k_j} \mathbf{R}_{DVL,k} \tag{4.13}$$

where the correlated covariance matrix $\mathbf{R}_{DVL,k}$ is computed with the same method described in (Ribas et al., 2010) using the σ values provided by the manufacturer's specifications:

$$\mathbf{R}_{DVL,k} = \begin{bmatrix} \sigma_u^2 & \sigma_{uv} & \sigma_{uw} \\ \sigma_{vu} & \sigma_v^2 & \sigma_{vw} \\ \sigma_{wu} & \sigma_{wv} & \sigma_w^2 \end{bmatrix} \tag{4.14}$$

Since the DVL sensor provides a status measurement for the bottom tracking and water velocities, depending on the quality of the measurements, different versions of the $\mathbf{H}$ matrix are used to fuse one (removing row 2), the other (removing row 1), or both readings (using the full matrix).

UPDATE USING MRU MEASUREMENTS. Whenever a new attitude measurement is available from the MRU sensor, the model prediction is updated using the standard EKF equations:

$$z_{MRU,k} = [\phi, \theta, \psi]^T, \quad z_{MRU,k} = \mathbf{H}_{MRU,k} x_{k|k-1} + v_k \tag{4.15}$$

$$\mathbf{H}_{MRU} = \begin{bmatrix} \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} & \mathbf{0}_{3\times6} \end{bmatrix} \tag{4.16}$$

where the measurement noise v_k is a gaussian zero-mean white noise:

$$E[v_k] = 0, \quad E[v_k, v_j^T] = \delta_{k_j} \mathbf{R}_{MRU,k} \tag{4.17}$$

where the covariance matrix $\mathbf{R}_{MRU,k}$ is non-correlated since the MRU is mounted aligned with the vehicle's axes. σ values are set according to the manufacturer's specifications:

$$\mathbf{R}_{MRU,k} = \begin{bmatrix} \sigma_\phi^2 & 0 & 0 \\ 0 & \sigma_\theta^2 & 0 \\ 0 & 0 & \sigma_\psi^2 \end{bmatrix} \tag{4.18}$$

4.1.3.3 *Scan Forming*

The presented navigation system is able to estimate the vehicle's pose, but the uncertainty will grow without limit due to its dead-reckoning nature. Moreover, the pIC algorithm only requires the robot's relative pose and uncertainty with respect to the initial reference frame I at the beginning of the scan. Hence, a slight modification to the filter is introduced making a reset in position (setting x, y, z to 0 in the vector state) whenever a new scan is started. Therefore, while the filter is running, the estimated position is always relative to the position where the first beam of the scan was gathered. Note that it is important to keep the ψ value since it represents an absolute angle with respect to the magnetic north. For this reason, a reset would mean an unreal high rotation during the scan. The same thing applies to ϕ and θ. Since it is only required the uncertainty accumulated during the scan, the reset process also affects the x, y, and z terms of the covariance matrix $\mathbf{P}$.

The modified EKF provides the vehicle's relative pose where the beams were gathered including its uncertainty accumulated during the scan. Therefore, using a similar procedure than in (Ribas et al., 2008), it is possible to reference all the ranges computed from the beams to the initial frame I, removing the distortion induced by the robot's motion by using the following method.

The mathematical method needed to compute the undistorted points of the scan and their uncertainty is detailed below while a geometric representation of the transformations involved is provided in Figure 45.

Let

- $\boldsymbol{\rho} \equiv N(\hat{\boldsymbol{\rho}}, \mathbf{P}_\rho)$ be a r.g.v corresponding to the polar measurement where $\hat{\boldsymbol{\rho}} = (\beta, r)$ is the observed measurement and $\mathbf{P}_{\boldsymbol{\rho}}$ its corresponding uncertainty.

- $\mathbf{x}_R^B \equiv N(\hat{\mathbf{x}}_R^B, \mathbf{P}_{BR})$ be a r.g.v corresponding to the vehicle's uncertain position where the ρ beam was gathered. This value is estimated by the EKF and is represented in the northern referenced frame B.
- $\mathbf{x}_B^I \equiv N(\hat{\mathbf{x}}_B^I, \mathbf{P}_{IB})$ be a r.g.v corresponding to the transformation needed to map B frame to I frame. In this particular case, it is a null translation followed by a rotation used to align B with I.
- $\mathbf{x}_S^R$ be a deterministic vector that describes the position and attitude of the sensor frame S with respect to the robot's frame R. Note that this is a non-random rigid body transformation.

Then, it is possible to compute the pose and uncertainty of any observed point referenced to the initial frame I as follows:

1. $\mathbf{p}^S = P2C(\rho) \Rightarrow \mathbf{p}^S = N(\underbrace{P2C(\hat{\boldsymbol{\rho}})}_{\hat{\mathbf{p}}^S}, \underbrace{\mathbf{J}_S \mathbf{P}_\rho \mathbf{J}_S^T}_{\mathbf{P}_S})$
   where $P2C(\boldsymbol{\rho})$ turns polar coordinates into Cartesian coordinates and
   $$\mathbf{J}_S = \left.\frac{\partial\, P2C(\boldsymbol{\rho})}{\partial \boldsymbol{\rho}}\right|_{\hat{\boldsymbol{\rho}}}$$
2. $\mathbf{p}^R = \mathbf{x}_S^R \oplus \mathbf{p}^S \Rightarrow \mathbf{p}^R = N(\underbrace{\mathbf{x}_S^R \oplus \hat{\mathbf{p}}^S}_{\hat{\mathbf{p}}^R}, \underbrace{\mathbf{J}_{2\oplus} \mathbf{P}_S \mathbf{J}_{2\oplus}^T}_{\mathbf{P}_R})$
3. $\mathbf{p}^B = \mathbf{x}_R^B \oplus \mathbf{p}^R \Rightarrow \mathbf{p}^B = N(\underbrace{\hat{\mathbf{x}}_R^B \oplus \hat{\mathbf{p}}^R}_{\hat{\mathbf{p}}^B}, \underbrace{\mathbf{J}_{1\oplus} \mathbf{P}_{BR} \mathbf{J}_{1\oplus}^T + \mathbf{J}_{2\oplus} \mathbf{P}_R \mathbf{J}_{2\oplus}^T}_{\mathbf{P}_B})$
4. $\mathbf{p}^I = \mathbf{x}_B^I \oplus \mathbf{p}^B \Rightarrow \mathbf{p}^I = N(\underbrace{\hat{\mathbf{x}}_B^I \oplus \hat{\mathbf{p}}^B}_{\hat{\mathbf{p}}^I}, \underbrace{\mathbf{J}_{1\oplus} \mathbf{P}_{IB} \mathbf{J}_{1\oplus}^T + \mathbf{J}_{2\oplus} \mathbf{P}_B \mathbf{J}_{2\oplus}^T}_{\mathbf{P}_I})$

First, function P2C transforms the bearing and range data $\boldsymbol{\rho} = (\beta, r)^T$ from Polar space to Cartesian space. The result is the observed point $\mathbf{p}^S$ referenced to the sensor frame S. As stated, $\mathbf{p}^S$ is an r.g.v whose mean $\hat{\mathbf{p}}^S$ and covariance $\mathbf{P}_S$ can be easily computed. Then, by means of a rigid body transformation, the point is referenced to the robot's frame R. Again, the new representation $\mathbf{p}^R$ is an r.g.v with mean $\hat{\mathbf{p}}^R$ and covariance $\mathbf{P}_R$. Now, the robot's relative position $\mathbf{x}_R^B$ computed with the EKF is compounded with the robot's referenced point $\mathbf{p}^R$ to get the r.g.v $\mathbf{p}^B$ with mean $\hat{\mathbf{p}}^B$ and covariance $\mathbf{P}_B$. Finally, the last compounding operation rotates the point to reference it to the initial frame I. As in the previous cases, $\mathbf{p}^I$ is an r.g.v with a known mean $\hat{\mathbf{p}}^I$ and covariance $\mathbf{P}_I$. Algorithm 11 describes this process in algorithmic notation. In order to show an illustrated example, the full process is applied to a scan depicted in Figure 46, where each beam is gathered in a different vehicle position. As a result, the algorithm references the scan measurements in the position of the robot where the first beam of the scan was gathered, which is depicted in Figure 47.

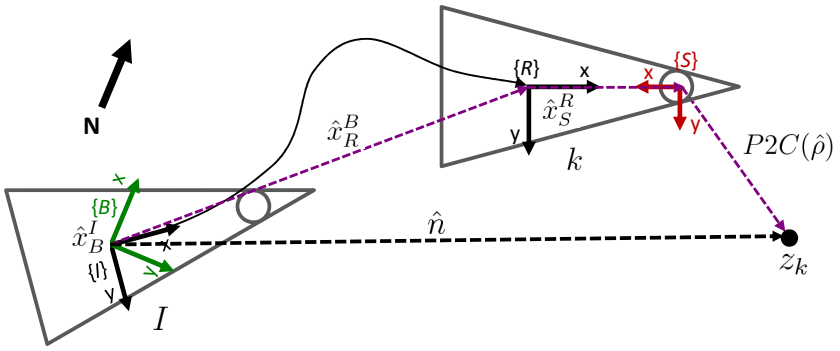


Figure 45: A scan forming process: any beam k of the scan is represented with respect to the pose of the robot when the first beam I was gathered.

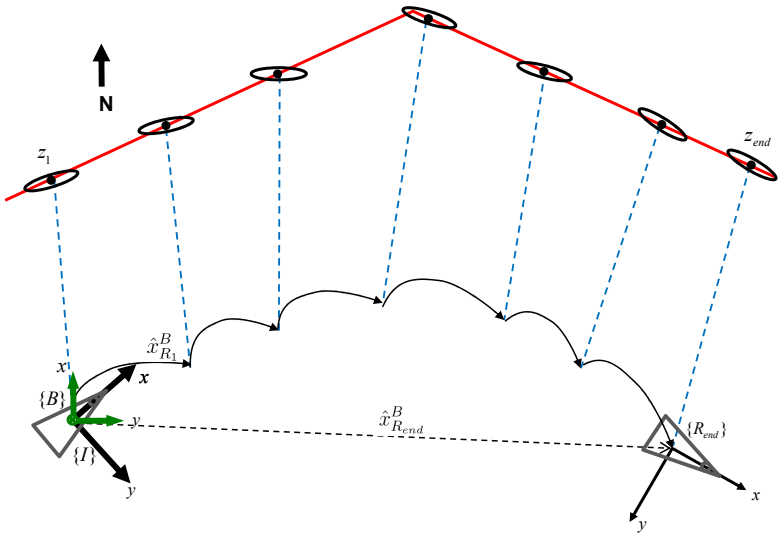


Figure 46: Initially each beam is gathered at different vehicle positions.

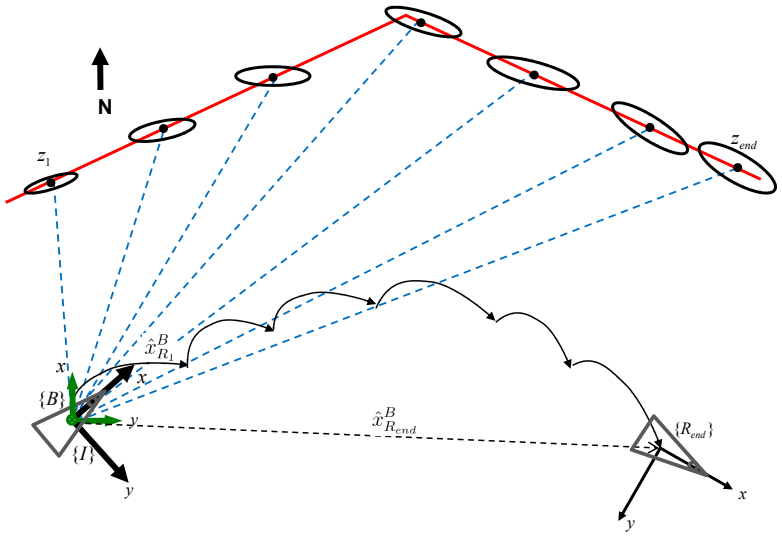


Figure 47: The scan grabbing process references all the beams of the scan at the position of the robot when the first beam was gathered. The uncertainty of the motion has been propagated to the scan points.

Algorithm 11 Scan grabbing

ScanGrabbing()

1: ResetDeadReckoningXYZ()
2: $\{\hat{\mathbf{x}}_B^I, \mathbf{P}_{IB}\} \leftarrow$ GetDeadReckoning()
3: $S \leftarrow \emptyset$
4: **while** $|S| <$ beamsPerScan **do**
5: $\quad$ beam $\leftarrow$ GetBeam()
6: $\quad$ beam $\leftarrow$ Segment(beam)
7: $\quad \{\hat{\rho}, \mathbf{P}_\rho\} \leftarrow$ LocalMaximaFinder(beam)
8: $\quad \{\hat{\mathbf{x}}_R^B, \mathbf{P}_{BR}\} \leftarrow$ GetDeadReckoning()
9: $\quad \hat{\mathbf{n}} \leftarrow \hat{\mathbf{x}}_B^I \oplus \hat{\mathbf{x}}_R^B \oplus \mathbf{x}_S^R \oplus$ P2C($\hat{\rho}$) {$\hat{\rho}$ from the local frame I}
10: $\quad \mathbf{P}_B \leftarrow [\mathbf{J}_{1\oplus}]_{\hat{\mathbf{x}}_R^B} \mathbf{P}_{BR} [\mathbf{J}_{1\oplus}^T]_{\hat{\mathbf{x}}_R^B} + [\mathbf{J}_{2\oplus}]_{\hat{\mathbf{p}}^R} [\mathbf{J}_{2\oplus}]_{\hat{\mathbf{p}}^S} [\mathbf{J}_S]_{\hat{\rho}} \mathbf{P}_\rho [\mathbf{J}_S^T]_{\hat{\rho}} [\mathbf{J}_{2\oplus}^T]_{\hat{\mathbf{p}}^S} [\mathbf{J}_{2\oplus}^T]_{\hat{\mathbf{p}}^R}$
11: $\quad \mathbf{P}_n \leftarrow [\mathbf{J}_{1\oplus}]_{\hat{\mathbf{x}}_B^I} \mathbf{P}_{IB} [\mathbf{J}_{1\oplus}^T]_{\hat{\mathbf{x}}_B^I} + [\mathbf{J}_{2\oplus}]_{\hat{\mathbf{p}}^B} \mathbf{P}_B [\mathbf{J}_{2\oplus}^T]_{\hat{\mathbf{p}}^B}$ {$\mathbf{P}_\rho$ from the local frame I}
12: $\quad S \leftarrow S \cup \{\hat{\mathbf{n}}, \mathbf{P}_n\}$
13: **end while**
14: $\hat{\mathbf{q}} \leftarrow \hat{\mathbf{x}}_R^B$
15: $\mathbf{P}_q \leftarrow \mathbf{P}_{BR}$
16: **return** $\{S, \hat{\mathbf{q}}, \mathbf{P}_q\}$

4.1.4 *The MSISpIC algorithm*

The MSISpIC is an extension of the pIC algorithm that deals with motion induced distortions involved in the scan grabbing process when using imaging or profiler sonar sensors. The algorithm, described in pseudocode in Algorithm 12, iteratively grabs two scans and register them using the pIC algorithm (line 5). It is worth noting that the pIC takes as input two consecutive scans (S_{ref} and S_{new}) and its relative displacement which coincides with the pose of the vehicle at the end of the first scan ($\mathbf{q}_{ref}$). The output is an improved estimation of the robot's displacement $\mathbf{q}_{new}$. The iterative compounding of the relative displacement allows us to track the robot's global position (line 6).

Algorithm 12 MSISpIC

MSISpIC()

1: $\{S_{ref}, \hat{\mathbf{q}}_{ref}, \mathbf{P}_{q_{ref}}\} \leftarrow$ ScanGrabbing()
2: $\hat{\mathbf{q}}_{global} \leftarrow \mathbf{0}$
3: **loop**
4: $\quad \{S_{new}, \hat{\mathbf{q}}_{new}, \mathbf{P}_{q_{new}}\} \leftarrow$ ScanGrabbing()
5: $\quad \hat{\mathbf{q}}_{pIC} \leftarrow$ pIC($S_{ref}, S_{new}, \hat{\mathbf{q}}_{ref}, \mathbf{P}_{q_{ref}}$)
6: $\quad \hat{\mathbf{q}}_{global} \leftarrow \hat{\mathbf{q}}_{global} \oplus \hat{\mathbf{q}}_{pIC}$
7: $\quad S_{ref} \leftarrow S_{new}$
8: $\quad \hat{\mathbf{q}}_{ref} \leftarrow \hat{\mathbf{q}}_{new}$
9: **end loop**

4.2 OCCUPANCY GRID MAPPING

This section concerns about Occupancy Grid Map (OGM) techniques which group a set of algorithms used to build a representation of the environment

based on the data gathered by the vehicle while navigating. Scan matching techniques improve the dead-reckoning estimation of the vehicle, but do not compute its exact displacement due to the intrinsic noisy data gathered with the sonar sensors. Therefore, most of the time raw data is not suitable to build a robust representation of the environment where the robot has to perform path planning.

OGM algorithms address the problem of generating consistent maps from noisy and uncertain measurement data under the assumption that the vehicle pose is known. The basic idea of occupancy grids is to represent the map as a field of random variables arranged in an evenly space grid. Each random variable is binary and corresponds to the occupancy of the location it covers. OGM algorithms implement approximate posterior estimation for those random variables (Thrun et al., 2005).

OGM techniques were originally developed by (Moravec and Elfes, 1985) and have been used in mobile robotics since its early origins in the middle eighties as a robust map representation for SLAM (Hähnel et al., 2003), as well as for obstacle avoidance (Borenstein and Koren, 1991) and path planning (Elfes, 1989). In underwater robotics, grid mapping has been used in many applications such as searching hydrothermal vent fields (Jakuba and Yoerger, 2008) as well as for SLAM applications (Fairfield et al., 2007).

4.2.1 *Problem Definition*

The standard OGM problem consists of computing the grid histogram related to the density function $p(m|z_1, ..., z_T)$, where m is the map and $z_1, ..z_T$ are the robot's measurements for a time t from 1 to T. The problem assumes that the robot's trajectory $(x_1, ..., x_T)$ is known. Since maps are defined over a high dimensional space, the following assumption is commonly applied in order to keep the problem tractable:

$$p(m|z_1, ..., z_T) = \prod_{x,y} p(m_{x,y}|z_1, ..., z_T)$$

where it has been assumed that each individual grid cell is conditional independent of the rest converting the multidimensional mapping problem into a set of one dimensional mapping problems (one for each cell). Then, the Bayes rule can be used to estimate the individual probability of each cell at each time step t:

$$p(m_{x,y}|z_1, .., z_t) = \frac{p(z_t|z_1, .., z_{t-1}, m_{x,y})p(m_{x,y}|z_1, .., z_{t-1})}{p(z_t|z_1, ..., z_{t-1})}$$

Under the *static world assumption*, given a certain map m, the conditional dependence between the last and the past measurements disappears:

$$p(m_{x,y}|z_1, ..., z_t) = \frac{p(z_t|m_{x,y})p(m_{x,y}|z_1, ..., z_{t-1})}{p(z_t|z_1, ..., z_{t-1})}$$

Then, the Bayes rule can also be applied to $p(z_t|m_{x,y})$ to obtain the following recursive formulation:

$$p(m_{x,y}|z_1,...,z_t) = \frac{p(m_{x,y}|z_t)p(z_t)p(m_{x,y}|z_1,...,z_{t-1})}{p(m_{x,y})p(z_t|z_1,...,z_{t-1})} \tag{4.19}$$

By analogy, for the opposite situation $\neg m$:

$$p(\neg m_{x,y}|z_1,...,z_t) = \frac{p(\neg m_{x,y}|z_t)p(z_t)p(\neg m_{x,y}|z_1,...,z_{t-1})}{p(\neg m_{x,y})p(z_t|z_1,...,z_{t-1})} \tag{4.20}$$

Dividing (4.19) by (4.20) leads to cancelation of various probabilities difficult to calculate:

$$\begin{aligned}\frac{p(m_{x,y}|z_1,...,z_t)}{p(\neg m_{x,y}|z_1,...,z_t)} &= \frac{p(m_{x,y}|z_t)}{p(\neg m_{x,y}|z_t)}\frac{p(m_{x,y}|z_1,...,z_{t-1})}{p(\neg m_{x,y}|z_1,...,z_{t-1})}\frac{p(\neg m_{x,y})}{p(m_{x,y})} \\ &= \frac{p(m_{x,y}|z_t)}{1-p(m_{x,y}|z_t)}\frac{p(m_{x,y}|z_1,...,z_{t-1})}{1-p(m_{x,y}|z_1,...,z_{t-1})}\frac{1-p(m_{x,y})}{p(m_{x,y})}\end{aligned} \tag{4.21}$$

(4.21) is commonly expressed using log odds to avoid numerical instabilities for probabilities near zero or one:

$$l^t_{x,y} = l^{t-1}_{x,y} + m^t_{x,y} - l^0_{x,y}$$

where:

$$l^t_{x,y} = \log\frac{p(m^t_{x,y})}{1-p(m^t_{x,y})}$$

$$l^{t-1}_{x,y} = \log\frac{p(m^{t-1}_{x,y})}{1-p(m^{t-1}_{x,y})}$$

$$m^t_{x,y} = \log\frac{p(m^t_{x,y}|z_t)}{1-p(m^t_{x,y}|z_t)}$$

$$l^0_{x,y} = \log\frac{p(m^0_{x,y})}{1-p(m^0_{x,y})}$$

Notice that the probabilities can be easily recovered from the log odds ratio using for example $l^t_{x,y}$:

$$p(m^t_{x,y}) = 1 - \frac{1}{1+\exp\{l^t_{x,y}\}}$$

From the computational point of view, to compute the new log odds value of a grid cell probability $l^t_{x,y}$, its previous value $l^{t-1}_{x,y}$ has to be incremented to a certain value $m^t_{x,y}$ as stated by the inverse sensor model corresponding to the last measurement z_t and the initial probability of the grid cell $l^0_{x,y}$ must be subtracted.

4.2.2 *Inverse Sensor Model*

A conventional sensor model used in probabilistic localization algorithms is described by a density function $p(z|x)$ which allows the computation of the observation probabilities for a given robot position. For mapping applications, the inverse sensor model $p(m|z)$ is used instead, providing a mean to compute the map occupancy probability related to a certain observation. In this section, two inverse sensor models are provided, a straight forward classical one applicable to a sonar profiler and a proposed new one more suitable for imaging sonar sensors.

4.2.2.1 *Profiler Sonar*

A sonar profiler is a range and bearing sensor which behaves as common time-of-flight air ultrasonic sensors do. The main difference between them is that air ultrasonic sensors normally have a very wide angle β (e.g. 30°) in their beam pattern while underwater pencil-beam type sonar profilers have overture angles of 3° or even 1°. For this sort of sensor, the classical cone model broadly used in mobile robotics can be used. Given a certain range measurement, this model provides the probability of being occupied in the map for each grid cell that belongs to the sector related to the beam pattern. The model establishes three probabilities in their log odd form, which are depicted in Figure 48:

- l_{occ}. The probability of being occupied, which is assigned to all the grid cells within the sensor cone range close to the range distance. The width of this area is set by an α parameter.
- l_{free}. The probability of being free, which is assigned to all the cells between the origin of the beam and the observed range minus the $\alpha/2$ interval.
- $l_{unknown}$. The probability of being unknown, assigned to all the grid cells further than the observed range plus the $\alpha/2$ interval.

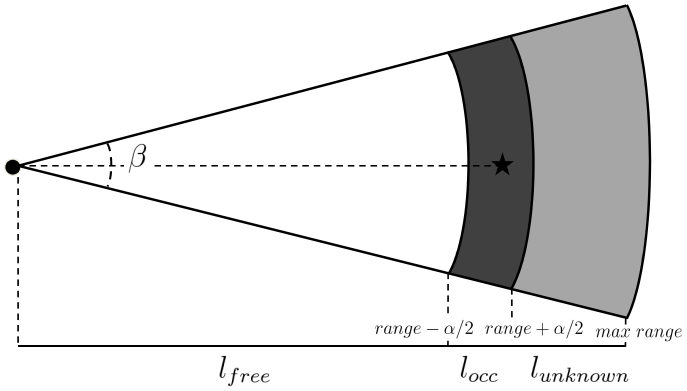


Figure 48: A profiler sonar inverse sensor model.

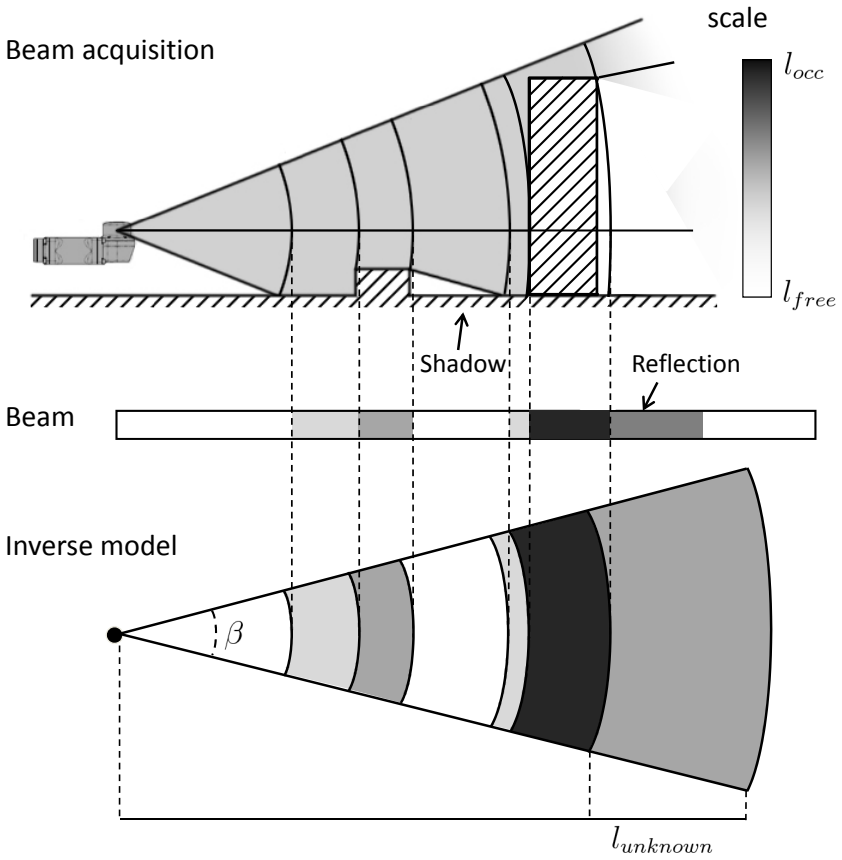


Figure 49: Imaging sonar inverse sensor model.

4.2.2.2 *Imaging Sonar*

A range finder sensor is usually modeled as a cone, whereas an imaging sonar beam pattern is often represented with a fan shape. This beam pattern has two overture angles, a narrow horizontal angle (e.g. 3°) and a wide vertical one (e.g. 40°). Since the work presented in this document is conceived in 2D, the imaging sonar is represented as a sector with an overture of $\beta = 3°$. As stated earlier, there is a relationship between the amount of acoustic energy reflected at a certain distance and its occupancy probability within the beam cone. Under this hypothesis, an inverted beam model like the one shown in Figure 49 can be established, where for each bin intensity, a related log odd probability value is assigned to all cells that have the same range within the sensor cone. These log odd probabilities are computed using a linear transformation whose limits are set according to the intensity bounds of the sensor.

As stated in the profiler sonar model, if a bin has a higher intensity value than a predefined threshold, it is assumed that an obstacle is found and hence, a $l_{unknown}$ value is assigned from the next bin to the end of the beam because it is not possible to know the information behind it. However, small obstacles that do not return high intensity levels create shadows in the beam's projection that become *unknown* areas. These are projected into the map according to their intensity values.

4.3 SUMMARY

In this chapter we have proposed a local map building method that achieves the computation of an OGM in realtime while the robot is navigating. The method first improves the dead-reckoning navigation through scan matching. We have proposed the MSISpIC, an extension of the pIC algorithm that deals with the data gathered with an MSIS. The algorithm removes the motion induced distortion of the scan and predicts its uncertainty by using an EKF with a constant velocity model and acceleration noise, updated with velocity and attitude measurements obtained from a DVL and an MRU respectively. Then, the standard pIC improves the relative displacement between them.

In order to generate an OGM, two inverse sensor models have been proposed: the commonly used range finder, modeled as a sector where the area far from the sensor is assumed to be occupied by a detected obstacle, whereas the rest of the sector is a free zone; and an imaging sonar sensor, which takes into account the probability of occupancy at each discretized part of the beam cone according to the amount of acoustic energy reflected.

5

EXPERIMENTAL PLATFORM

This chapter reports the main features of the experimental platforms used in this dissertation, which are Ictineu AUV and Sparus AUV, including their design principles, the actuators and the on board sensors. Next, a detailed description of the sensors used for map building purposes is provided. Finally, an insight of the control architecture used in the robots is given.

5.1 VEHICLE EXPERIMENTAL PLATFORMS

The VICOROB group of the University of Girona started the development of underwater vehicles in 1995 with the construction of Garbi (Amat et al., 1996) which was conceived as a ROV. In 2005, the vehicle was rebuilt and converted into the Garbi AUV which was equipped with four thrusters, a simple sensor suite and two battery packs. The Garbi's dimensions were 1.3m long, 0.9m high and 0.7m wide with a maximum speed of 1knot and a weight of approximately 150Kg. In 2001, Uris (Batlle et al., 2004) was developed, a light weight (35Kg) low cost AUV. It was a spherical shaped vehicle (35cm diameter) used as a research platform in a water tank testing facility. The experience obtained with the development of these vehicles by the group made it possible to build two more low-cost vehicles: the Ictineu AUV and the Sparus AUV, which have been used in some of the experiments in this thesis.

5.1.1 *Ictineu AUV*

The Ictineu AUV, depicted in Figure 50, is the result of a project started in 2006. During the summer of that year, the Defence Science and Technology Lab (DSTL), the Heriot-Watt University and the National Oceanographic Center of Southampton organized the first SAUC-E, a European wide competition for students to promote research and development in underwater technology. The Ictineu AUV was originally conceived as an entry for the SAUC-E competition by a team of students collaborating with the VICOROB group at the University of Girona (Ribas et al., 2007). Although the competition determined many of the vehicle's specifications, Ictineu AUV was also designed keeping its posterior usage as an experimental platform for various research projects in our laboratory in mind. The experience obtained from the development of previous vehicles by the group made it possible to build a low-cost vehicle of reduced weight (52Kg) and dimensions (0.74x0.465x0.524m) with remarkable sensorial capabilities and easy maintenance.

The Ictineu AUV was built using a typical open frame design. This configuration has been widely adopted by commercial ROVs because of its simplicity, toughness and reduced cost. Although the hydrodynamics of open frame

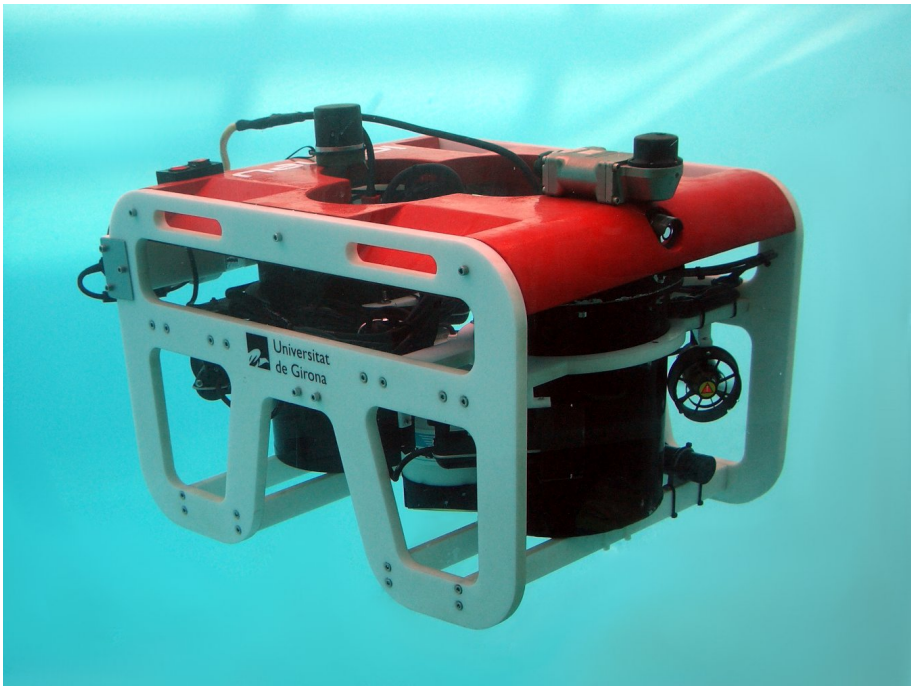


Figure 50: The Ictineu AUV.

vehicles is known to be less efficient than that of closed hull type vehicles, they are suitable for applications requiring neither high velocity movements nor traveling long distances. The robot's chassis is made of Delrin, a plastic engineering material which is lightweight, durable and resistant to liquids. Another aspect of the design is the modular conception of its components which simplifies upgrading the vehicle and makes it easier to carry out maintenance tasks.

The thrusters and most of the sensors are watertight and mounted directly on the vehicle's chassis. On the other hand, two cylindrical aluminum pressure vessels house the power and computer modules while a smaller one made of Delrin contains an MRU. Their end-caps are sealed with conventional O-ring closures while the electrical connections with other hulls or external sensors are made with plastic cable glands sealed with epoxy resin. The Ictineu AUV is propelled by six thrusters which allow it to be fully actuated in Surge (movement along X axis), Sway (movement along Y axis), Heave (movement along Z axis) and Yaw (rotation around Z axis) achieving maximum speeds of 3knots. It is passively stable in both Pitch and Roll DoFs as its meta-centre is above the centre of gravity. This stability is the result of an accurate distribution of the heavier elements in the lower part of the frame combined with the effect of technical foam placed in the top, which provides a slightly positive buoyancy to the vehicle.

One of the main objectives of the laboratory was to provide the underwater robot with a complete sensor suite. The robot includes a Tritech Miniking MSIS designed for use in underwater applications such as obstacle avoidance and target recognition. The robot is also equipped with a SonTek Argonaut DVL

which measures ocean currents, vehicle speed over ground and altitude using its 3 acoustic beams. The particular spatial distribution chosen to place the acoustic sensors within the vehicle's frame avoids dead zones, improving their overall performance. Moreover, the Ictineu AUV has a compass which outputs the sensor heading (angle with respect to the magnetic North), a pressure sensor for water column pressure measurements and an Xsens MTi low cost miniature MRU which provides 3D orientation (attitude and heading), a 3D rate of turn as well as 3D acceleration measurements. Finally, the robot is also equipped with two cameras; a forward-looking color camera, mounted on the front of the vehicle and a downward-looking Tritech Super SeaSpy color Charge Coupled Device (CCD) Underwater Camera, located in the lower part of the vehicle. The latter is mainly used to capture images of the seabed for research on image mosaicking while the former is intended for target detection and tracking, inspection of underwater structures and to provide visual feedback when operating the vehicle in the ROV mode. At present, the Ictineu AUV is being used as a research platform for various underwater inspection projects which include dams (Ridao et al., 2010), harbors, shallow waters and cable/pipeline inspections (El-Fakdi et al., 2010).

5.1.2 *Sparus AUV*

The Sparus AUV, depicted in Figure 51, is the second experimental platform used in this thesis. It was built by a group of students in the Univerisity of Girona to face the SAUC-E competition 2010 edition (Hurtos et al., 2010). The main goal of the Sparus design was to build a small, simple torpedo-shaped vehicle with hovering capabilities.

The Sparus AUV is equipped with three Seabotix thrusters integrated in a classical torpedo shape. Two are placed horizontally at the back of the vehicle for the Surge and Yaw DoFs, and the third is placed vertically for the Heave DoF. The horizontal propellers at the back of the vehicle are separated from its longitudinal axis to generate a torque for the Yaw DoF. The vertical thruster is placed in the centre of the vehicle, where the centre of gravity and buoyancy are located. Therefore, the mechanical structure and components are organized around this configuration.

All the sensors of the vehicle are placed at the front. The robot also has two cylindrical aluminum pressure vessels. The first one, placed at the front, houses the battery. The second one, which houses the electronics, computer and inertial navigation system, is placed at the back. Having the batteries in a separate housing increases the weight, length and consumption of the vehicle. However, it minimizes the downtime between missions by allowing battery packs to be quickly swapped. Moreover, potentially explosive gases that can be released from the batteries do not interfere with sparking and high temperature electronics.

The main structure is made of aluminum profiles and stainless steel clamps that hold the two pressure vessels. The battery housing is held with only one clamp in the top, to allow easy and fast replacement for a second battery

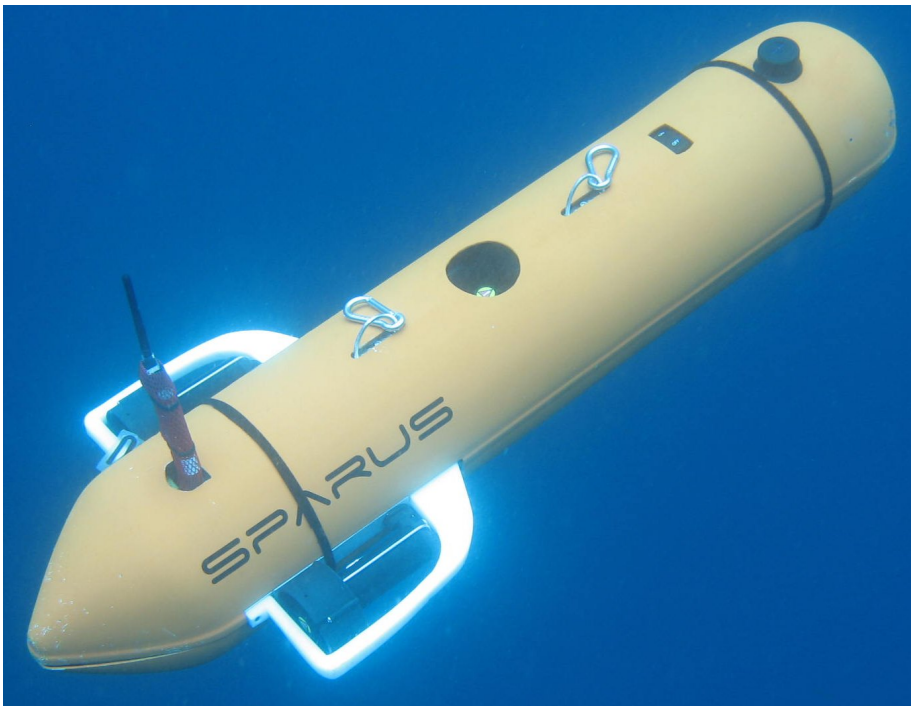


Figure 51: The Sparus AUV.

housing. The housings are designed to resist water pressure to a 100m depth. They are made of aluminum, which efficiently transmits internal heat to the environment, it is easy to machine and is lighter than stainless steel. In order to eliminate screws, a 1.8mm nylon thread before the O-ring secures the end-caps. The electronics housing can be easily opened from the back of the vehicle, allowing access to the electronics and computer. The dimensions of the vehicle are 1.22m length by 0.23m diameter, and its weight is around 30kg.

To provide the vehicle with the required buoyancy, there is technical foam distributed all over the top part of the robot. It is strategically located to place the buoyancy centre above the gravity centre, at the same longitudinal position, ensuring Pitch and Roll stability. The vehicle is trimmed with lead weights to bring the gravity centre in line with the vertical thruster. Small zinc anodes are added to eliminate corrosion due to sea water and different metals on the vehicle. Finally, a two-part ABS skin covers the AUV to reduce water drag and to protect the components inside.

Two types of underwater connectors were used to connect all the external components on the vehicle: Subconn® for high current connections and Lumberg® for low current parts. Another Subconn® connector is used for the umbilical cable. The main battery switch is IP68 rated and covered with resin. The robot has a WiFi adapter, also covered with resin, placed on the top. The embedded on-board computer was chosen as a trade-off between processing power, size and power consumption. An Ultra Low Voltage (ULV) Intel® Core™ Duo processor with the 3.5" small form factor was selected. The vehicle is also equipped with a complete sensor suite composed of two

color video cameras (forward-looking and downward-looking), an MRU MTi from XSens Technologies, a Micron MSIS from Tritech, an echo-sounder, a pressure sensor and a DVL from LinkQuest which also includes a compass/tilt sensor. Temperature, voltage and pressure sensors as well as water leakage detectors are installed into the pressure vessels for safety purposes. The on-board computer, the sensors and the three thrusters are powered by two battery packs. The first one, at 12V, powers the computer, the electronics and the sensors while the second one, at 24V, provides power to the thrusters. Each battery pack has a 10Ah capacity, which allows for an autonomy of 2.5 hours. Until the present, the Sparus AUV has been used as a research platform to obtain seabed photo mosaics covering areas of biological interest (Schmiing et al., 2009), dataset acquisition to apply scan matching and SLAM techniques (Mallios et al., 2011).

5.2 MAP BUILDING HARDWARE

This section details the sensors used for map building purposes in this research project.

5.2.1 *Doppler Velocity Log*

DVL sensors are specially designed for ROV/AUV applications which measure vehicle speed over water, vehicle speed over ground and altimetry using a precise 3-axis measurement system based on the Doppler shift effect.

On one hand, the Ictineu AUV is equipped with a SonTek Argonaut DVL (see Figure 52.a). This system operates at a frequency of 1500kHz and has a range of about 15m. Its three acoustic transducers are slanted 25° off the housing vertical axis and equally spaced at 120° relative azimuth angles. This beam geometry permits measuring velocities in 3D maintaining an optimal balance with total measurement range and near-boundary operation. The velocities can be obtained either in the beam coordinate system, in which the velocities are reported along the beams, or in the XYZ coordinate system, in which velocity measurements are stored using a right-handed Cartesian coordinate system relative to the robot. Its depth rating is 200m.

On the other hand, the Sparus AUV is equipped with a LinkQuest NavQuest 600 Micro DVL (see Figure 52.b). This system operates at a frequency of 600kHz and has a range of about 110m. The minimum altitude is 0.3m, which allows operation of the vehicle close to the bottom. It has four acoustic transducers slanted 22° off the housing vertical axis. The LinkQuest DVL also offers the possibility of obtaining the velocities in the beam coordinate system or in the XYZ coordinate system. Its standard depth rating is 800m.

Alternatively, both DVLs are equipped with a compass/tilt sensor which allows transforming the velocities to an East-North-Up (ENU) coordinate system so the data can be reported independently of instrument orientation. It can also record the distance to the bottom for each beam independently, estimate depth by means of a pressure sensor and measure water temperature

a) SonTek Argonaut DVL b) LinkQuest NavQuest 600 Micro DVL.

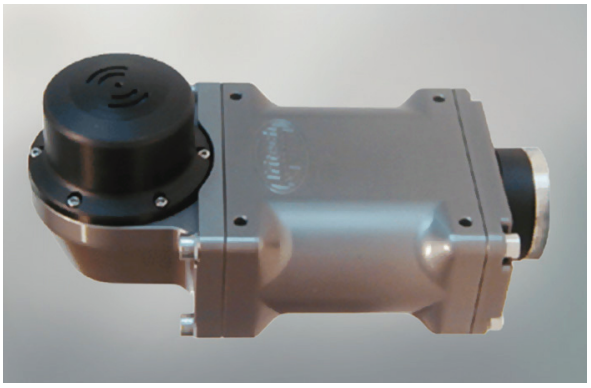
Figure 52: DVLs used in this research project.

Figure 53: Xsens MTi MRU.

for sound speed calculations. These two devices are versatile sensors, which together with their compact size, low power consumption and depth ratings, make them well suited for underwater vehicle positioning applications.

5.2.2 *Motion Reference Unit*

The Xsens MTi sensor, depicted in Figure 53, is a gyro-enhanced low cost miniature MRU which provides 3D orientation (attitude and heading), 3D rate of turn (rate gyro) as well as 3D acceleration measurements. Although the sensor is able to provide data at higher rates, the system gathers measurements from the MTi at a rate of 10 Hz. In order to produce drift-free angular measurements, the sensor also measures the directions of gravity and magnetic north. Our particular device configuration has $17 m/s^2$ full scale in acceleration measurements, which is far from the low accelerations that the Ictineu AUV and the Sparus AUV experience nowadays. For this reason, we do not usually rely on its acceleration estimates. On the other hand, the angular measurements are much more reliable and, as they are output at a higher rate than the data from the DVL sensor, they are mainly used for the estimation of attitude.



a) Miniking MSIS

b) Micron MSIS .

Figure 54: Models of the MSISs used in this research project.

5.2.3 *Mechanical Scanned Imaging Sonar*

Tritech MiniKing and Micron devices are small compact MSISs designed for use in underwater applications such as obstacle avoidance and target recognition for ROVs and AUVs. This sonar can perform scans in a 2D plane by rotating a fan-shaped sonar beam through a series of small angle steps. It can be programmed to cover variable length sectors from a few degrees to full 360° scans. A fan-shaped beam with a vertical aperture angle of 40° and a narrow horizontal aperture of 3° allows forming a sonar image with enough information about the surrounding environment to recognize the size and shape of a target at distances of up to 100 meters. Miniking and Micron sensors are mounted on the upper front part of the Ictineu AUV and Sparus AUV respectively to provide a clear view and avoid occlusions in the resulting data. Figure 54 depicts both MSISs. The principles of sonar theory are described in (technology corp., 2002) and (Urick, 1983) provides a deeper explanation.

5.2.4 *Multibeam Profiling Sonar*

The Multibeam Profiling Sonar (MPS) Model 837B “Delta T" 1000 from Imagenex is a high resolution multiple receiver sonar system designed to provide video-like imaging using sonar technology. The acoustic array is designed to provide detection over a 120x2.5° field of view covered by 480 beams for bottom profiling applications. The 2.5° field of view is also applicable to obstacle avoidance in near-bottom applications where the narrow vertical extent of the beam reduces false obstacle detection. This sensor operates at a 120kHz frequency to provide detection up to 300m range. The beam rate frequency is between 5-10Hz depending on the depth of the area scanned. The MPS has an MRU sensor to capture roll, pitch and heading. The stainless steel housing provides a 1000m depth rating, which makes it suitable for ROV and AUV applications.

Figure 55: Imagenex Multibeam.

5.3 COLA2 ARCHITECTURE

The architecture used in all the underwater vehicles available at University of Girona was the O2CA2 first proposed in (Ridao et al., 2002). It was a behavior-based control architecture (Brooks, 1986) that implemented a reactive layer in which a set of behaviors were able to perform some specific tasks. With the inclusion of a component with the ability to merge the behaviors responses (Carreras et al., 2001), it was possible to enable several of these behaviors simultaneously in order to perform a more complex task. However, traditional behavior-based architecture limitations appeared when trying to undertake long-range missions. Recently, this reactive architecture was turned into a layer-based architecture (Ribas et al., 2012) by adding an MCS, which adds the functionalities of a deliberative and an execution layer. The MCS is an independent set of components that can be connected with the O2CA2 reactive architecture but also with any other reactive architecture/layer implemented by an autonomous vehicle or manipulator (Palomeras et al., 2010). The new layer-based control architecture, named COLA2, was implemented in the Ictineu AUV and the Sparus AUV. It is based on software modules that encapsulate a set of related functions or data called components. Components may exist autonomously from other components in a network node having the ability to communicate with each other. The architecture, depicted in Figure 56, is organized in three layers following the hybrid model (Arkin and Balch, 1997; Firby, 1989): the mission layer, the execution layer and the reactive layer. The mission layer obtains a mission plan by means of an on-board automatic planning algorithm or by compiling a high-level mission description given by a human operator. The mission plan is then interpreted by the execution layer. It is executed by means of enabling/disabling the vehicle's behaviors or primitives contained in the reactive layer.

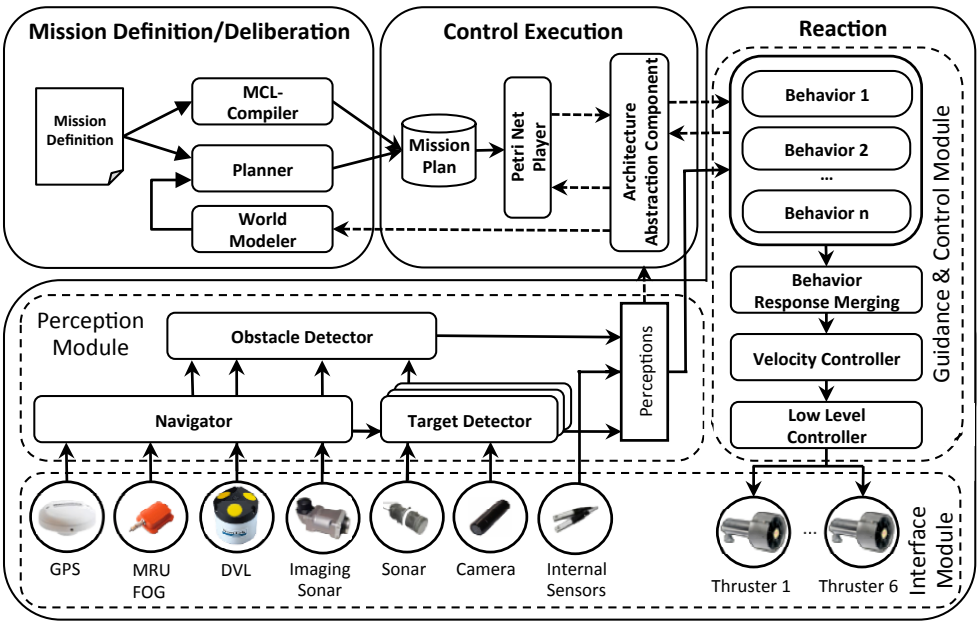


Figure 56: COLA2 architecture

5.3.1 *Reactive Layer*

The reactive layer implemented in our vehicles is based on the O2CA2 reactive architecture presented in (Ridao et al., 2002). Its goal is to execute basic primitives in order to fulfill the missions defined at the mission layer. Primitives are basic robot functionalities offered by the robot's control architecture. For an AUV, a primitive can range from a basic component that checks the battery level to a complex component that navigates toward a 3D way point. Primitives have a goal to achieve. For instance, the goal of an *achieve altitude* primitive would be to drive the robot at a constant altitude.

The reactive layer is highly dependant on the sensors and actuators of the robot. It is divided into three modules: the vehicle's interface module, the perception module and the guidance and control module.

- The vehicle's interface module contains components called *drivers* which interact with the hardware. It includes sensor drivers used to read data from sensors and actuator drivers to send commands to the actuators. An additional function provided by the drivers is to convert all the data to the same units as well as to reference all the gathered data to the vehicle's fixed body frame. Optionally, a Hardware In the Loop (HIL) simulator called Neptune (Ridao et al., 2004) can replace the drivers, allowing the execution of the architecture in simulation mode without modifying the rest of the components.

- The perception module receives the data gathered by the vehicle's interface module. Perception module components are called *processing units* and the main ones are: the navigator, the obstacle detector and the target

detectors. The navigator processing unit estimates the vehicle's position and velocity merging the data obtained from the navigation sensors by means of an EKF (Ribas et al., 2010). The obstacle detector measures the distance from the robot to the obstacles, mainly detected using acoustic sensors. Target detectors process acoustic or visual images to extract the most relevant features. Multiple target detectors have been programmed in order to detect different objects (El-Fakdi et al., 2010; Ribas et al., 2007).

- The guidance and control module includes a set of behaviors, a component to merge the behavior responses if necessary, a velocity controller and a low level controller. Behaviors receive data from the vehicle's interface or perception modules, remaining independent of the physical sensors and actuators used. Simple behaviors can be programmed as simple controllers, however, when the number of parameters to tune increases, they may be difficult to adjust. Then, RL techniques can be used to improve the adaptability of the vehicle's behaviors to the environment (Carreras et al., 2003; El-Fakdi et al., 2010). The second component is a coordinator used to combine all the responses generated by the behaviors into a single one. In general, a behavior response is a velocity request specified in the vehicle's body frame, which usually affects some particular DoFs at the same time. The coordinator generates a combined response for each DoF according to the behaviors priority. The last component is a velocity controller, which takes the merged velocity request and turns it into a force request. A simple Proportional Integral Derivative (PID) for each DoF is used for this task. The low level controller takes as input the force request provided by the velocity controller and computes a set-point for each thruster to achieve the desired force. A Thruster Allocation Matrix (TAM) plus a function for each thruster relating its input setpoint to the force it generates are used in this last step.

5.3.2 *Execution Layer*

The execution layer acts as the interface between the reactive layer and the mission layer, translating high-level plans into low-level commands. Additionally, the execution layer monitors the primitives being executed in the reactive layer. The execution layer of the COLA2 is composed of two main components: the Architecture Abstraction Component (AAC) and the Petri Net Player (PNP).

The AAC is the lower level component of the execution layer. Its function is to keep the mission and execution layers vehicle-independent, making the reactive layer the only one tied to the vehicle's hardware. The AAC provides an interface to the reactive layer based on three types of signals: actions, events and perceptions.

- Actions enable or disable basic primitives within the vehicle's reactive layer. For instance, an action can enable a primitive that controls the

vehicle's depth, pointing to a desired setpoint and the maximum time to reach it.

- Events are triggered in the reactive layer to announce changes in the state of its primitives. Following the last example, an event can announce that the desired depth has been reached within the required time or that the time has run out.
- Perceptions, meaning specific sensor or processing unit values, are transmitted from the reactive layer to the mission layer in order to be used to extract relevant information about the current world state when an on-board planner is used. Therefore, the execution layer is not using the perceptions, just transferring them from the reactive to the mission layer.

The second module included in the execution layer is the PNP, which executes mission plans using the Petri net formalism (Murata, 1989) by sending actions and receiving events through the AAC. The execution layer behaves as a Discrete Event System (DES) which connects high-level discrete plans, given by the mission layer, with low-level continuous primitives, in the reactive layer. The PNP controls all the timers associated with timed transitions, fires enabled transitions, sends actions from the execution layer to the reactive layer and, if necessary, fires enabled transitions in the mission layer when events are received.

5.3.3 *Mission Layer*

Nowadays, predefined plans are the state of the art for AUV missions. However, offline plans may fail during execution when the assumptions upon which they were based are violated (Turner, 2005). On the other hand, the use of online generated plans may result in unpredictable vehicle behaviors. Therefore, it is worth finding a compromise between predefined offline plans and automatically generated online plans. The COLA2 architecture introduces a high-level language, called Mission Control Language (MCL), for easily describing offline plans that are then automatically compiled into a formal Petri net (Palomeras et al., 2009). A mission can be either predefined by a user by means of a high-level language, the MCL, or using this same language, predefine some planning operators and let an onboard planner automatically execute, at each moment, the ones which are most appropriate to fulfill the mission.

In summary, the COLA2 is a layered architecture that combines the components in each layer. First of all, the reactive layer can be seen as a behavior-based architecture in which a set of behaviors are coordinated to fulfill a goal. However, instead of using the subsumption approach (Brooks, 1986) to coordinate them, the execution layer is in charge of enabling, prioritizing, and configuring behaviors following the plans described in the mission layer. This approach offers a good response without being limited to simple missions, as occurs in pure behavior-based architectures. Other advantages of the COLA2 architecture are the inclusion of an AAC, which makes the execution and mission

layers independent of hardware changes, and the use of a formal DES (Petri nets), which allows us to systematically verify and execute mission plans without complicating their description thanks to the MCL.

6

RESULTS

This chapter presents the results of this thesis, which are organized over four experiments. The first experiment (Section 6.1) consists of applying the map building procedure detailed in Chapter 4 to a dataset gathered with the Ictineu AUV in a man-made environment. The experiment runs the MSISpIC to improve a dead-reckoning trajectory and then computes an OGM with the two different inverse sensor models proposed in Chapter 4. Section 6.2 reports the second experiment, which shows the results obtained with the path planning method described in Chapter 3 in two different bitmap scenarios with irregular obstacles. The first one is a small cluttered environment which allows listing and tracking all the generated homotopy classes in the reference frame. The path for each homotopy class has been computed with the HA*, the HRRT and the HBug algorithms. The second one is a large scenario which accentuates the differences between the path planning algorithms. This environment is also used for two comparisons: the first one compares the homotopic path planners with their respective non-homotopic versions, the second one compares the HRRT and HBug with respect to the HA*, since it computes the shortest path for each homotopy class. Section 6.3 reports the next experiment, which was carried out using the Sparus AUV in a scenario built in the water tank of the Underwater Robotics Lab. of the University of Girona, and offers a controlled unknown environment to test the map building procedure together with the path planning method proposed in this thesis. Finally, in order to show the applicability of our path planning method in the context of the TRIDENT European project (EU FP7 ICT-248497), the last experiment (Section 6.4) presents the results obtained with the HA* algorithm in a bathymetry gathered in the Formigues Islands.

6.1 MAP BUILDING IN A MAN-MADE MARINA ENVIRONMENT

The goal of this experiment is to test the map building procedure described in Chapter 4 with a dataset obtained in an abandoned marina located in Sant Pere Pescador, on the Catalan coast (Ribas et al., 2008). The experiment was carried out using the Ictineu AUV (Ribas et al., 2007), which is described in Chapter 5, teleoperated along a 600m path.

In order to obtain a ground truth of the trajectory, a surface buoy equipped with a Differential Global Positioning System (DGPS) receiver was attached to the robot in order to gather the ground truth trajectory (see Figure 57). Since there is no DGPS signal when the device is submerged, the vehicle was teleoperated at a constant depth during the whole experiment.

The MSIS was configured to scan the whole 360° sector and was set to fire up to a 50m range with a 0.1m resolution and a 1.8° angular step. Dead-reckoning

Figure 57: The Ictineu AUV with a surface buoy equipped with a DGPS.

was computed using the velocity readings coming from the DVL and the heading data obtained from the MRU sensor, both merged using the EKF described in Chapter 4. Standard deviation for the MSIS sensor was set as is specified by the manufacturer: 0.1m in range and 1.8° in angular measurements. The whole dataset was acquired in 53min and the off-line execution of the algorithm implemented in MATLAB in a Intel® CoreDuo™@1,83GHz laptop took less than 14min. Therefore, it would be possible to execute the algorithm online in the vehicle while navigating.

6.1.1 *Scan Matching*

Figure 58 depicts the DGPS trajectory and the raw data plotted on an orthophotomap. The range data obtained from the MSIS beams through the segmentation process described in Section 4.1.3 almost perfectly match with the walls of the marina environment. Therefore, the DGPS trajectory can be used as a ground truth. Notice that the range measurements that appear in the middle of the water correspond to different objects laying on the seafloor, which returns an intensity level higher than the segmentation threshold.

Figure 59 shows the range data plotted with respect to the dead-reckoning trajectory. It can be clearly appreciated that dead-reckoning suffers from an important drift that makes the range data appear sparser. This issue is accentuated in those areas where the robot navigates closing loops, like the triangle shaped channel at the beginning of the experiment or the last part of the trajectory, after the corridor.

Using the MSISpIC, the drift is considerably reduced, as can be appreciated in Figure 60. Most of the error in the trajectory estimated by the MSISpIC

Figure 58: DGPS trajectory with range data plotted on the orthophotomap.

appears when the robot is traversing an area where the scan only observes one or two walls parallel to the robot's path, being able to correct the lateral and rotational displacement but still drifting in the forward direction. It is worth noting that, even in the presence of structures in all directions, scan matching algorithms are expected to drift due to their iterative formulation. However, the trajectory obtained with the MSISpIC shows an important improvement with respect to the dead-reckoning trajectory. Figure 61 depicts the absolute error of the displacement accumulated in both trajectories with respect to the DGPS. After traveling 600m, the dead-reckoning trajectory differs around 40m whereas the MSISpIC maximum difference is set around 12m.

6.1.2 *Occupancy Grid Mapping*

In order to generate the map, the OGM technique described in Chapter 4 was applied with the different trajectories obtained. The cell resolution for all the maps is 0.4m. The probability range was set up experimentally to be $p(m_{x,y} = free) = 0.3$, $p(m_{x,y} = occupied) = 0.9$ and $p(m_{x,y} = unknown) = 0.5$. These values are fixed for the sonar profiler model and linearly interpolated for the imaging sonar model (except the $p(m_{x,y} = unknown)$ value).

Figure 62.a and Figure 62.d depict the OGMs generated with the trajectory based on DGPS and the corrected MRU measurements, computed through SLAM (Ribas et al., 2008), used as a ground truth. Since the DGPS frequency rate was 1Hz, the trajectory of the vehicle through successive sensor measurements was interpolated by means of a spline in order to obtain a position and heading at the time each MSIS beam was received.

Using the dead-reckoning trajectory, most of the map obtained with the sonar profiler model (Figure 62.b) shows better defined walls when compared with the raw range measurement, except in the last part (after the corridor)

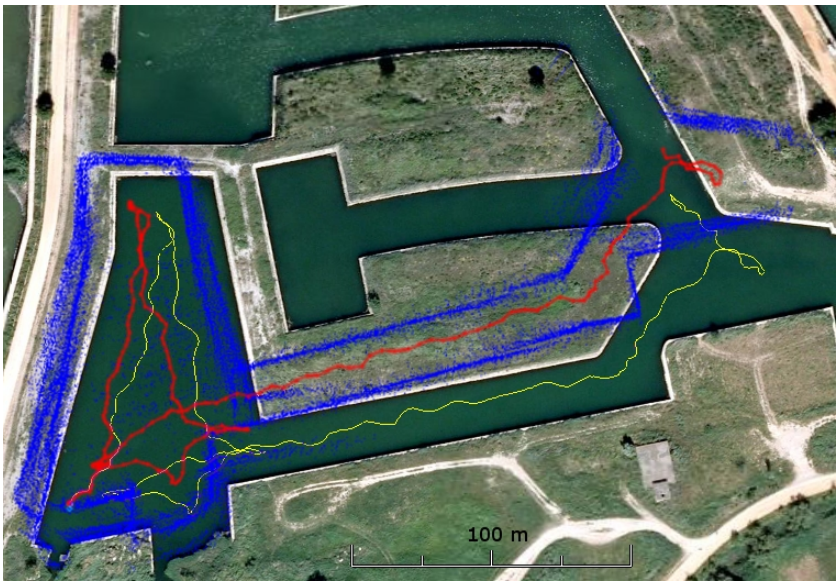


Figure 59: Dead-reckoning trajectory (in red) with range data plotted on the orthophotomap.

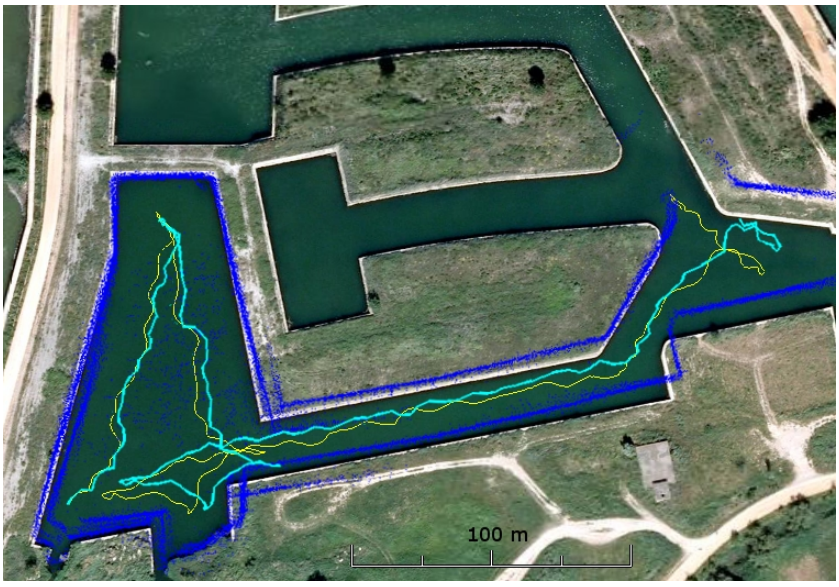


Figure 60: The MSISpIC trajectory (in cyan) with range data plotted on the orthophotomap.

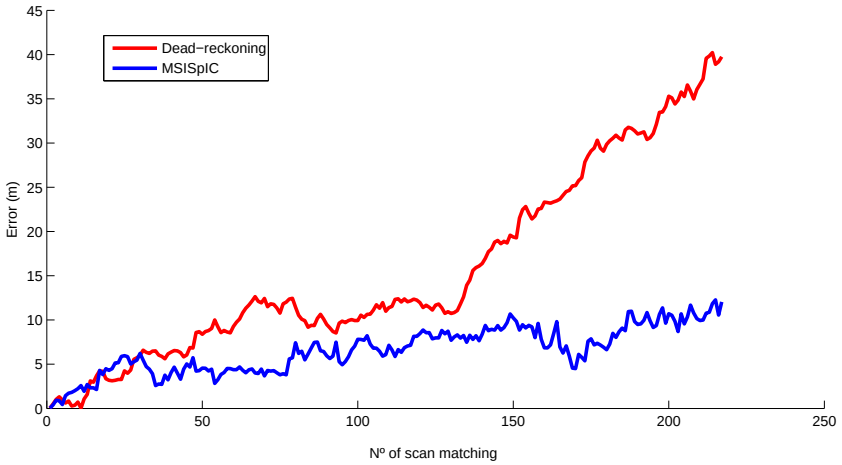


Figure 61: Dead-reckoning and MSISpIC trajectories absolute error with respect to the DGPS.

where the left wall is practically undistinguishable, as can be seen in Figure 59. Figure 62.e depicts the map generated with the imaging sonar model. The thickness of the walls increases because the neighbor bins of the one corresponding to the maximum return also have a remarkable intensity.

The maps generated with the scan matching trajectory are depicted in Figure 62.c and Figure 62.f. They show a substantial improvement with respect to the one generated using dead-reckoning. When comparing the maps built using raw data (see Figure 60), the one generated with scan matching is more accurate and the walls are less sparse. It can be appreciated that in the grid map, using the sonar profiler model, an *almost unknown* area appears within the corridor. This is due to the lack of range detection for the beams parallel to the walls of the corridor. When no range is detected, the entire sensor cone model is mapped to the unknown probability. In general, noisy measurements are clearly removed with the imaging sonar model since all the bins are mapped to a certain probability depending on the acoustic intensity.

6.2 PATH PLANNING WITH HOMOTOPY CLASS CONSTRAINTS

This section shows the simulations performed with the topological path search and the path planning algorithms described in Chapter 3. The environments are represented by bitmap images, where each pixel depicts a cell of an OGM. Black pixels represent occupied cells and white pixels represent free cells. The OGMs are also used as C-spaces, assuming that the vehicle is a single point without area.

The tests have been carried out in two different scenarios with irregular obstacles, the most common to be found in unstructured underwater environments. In order to identify the obstacles in the scenarios, a modified version of the Component Labeling (CL) algorithm (Chang et al., 2004) has been applied, which efficiently labels connected cells and their contours in greyscale images at the same time. For the construction of the reference frame, the c point has

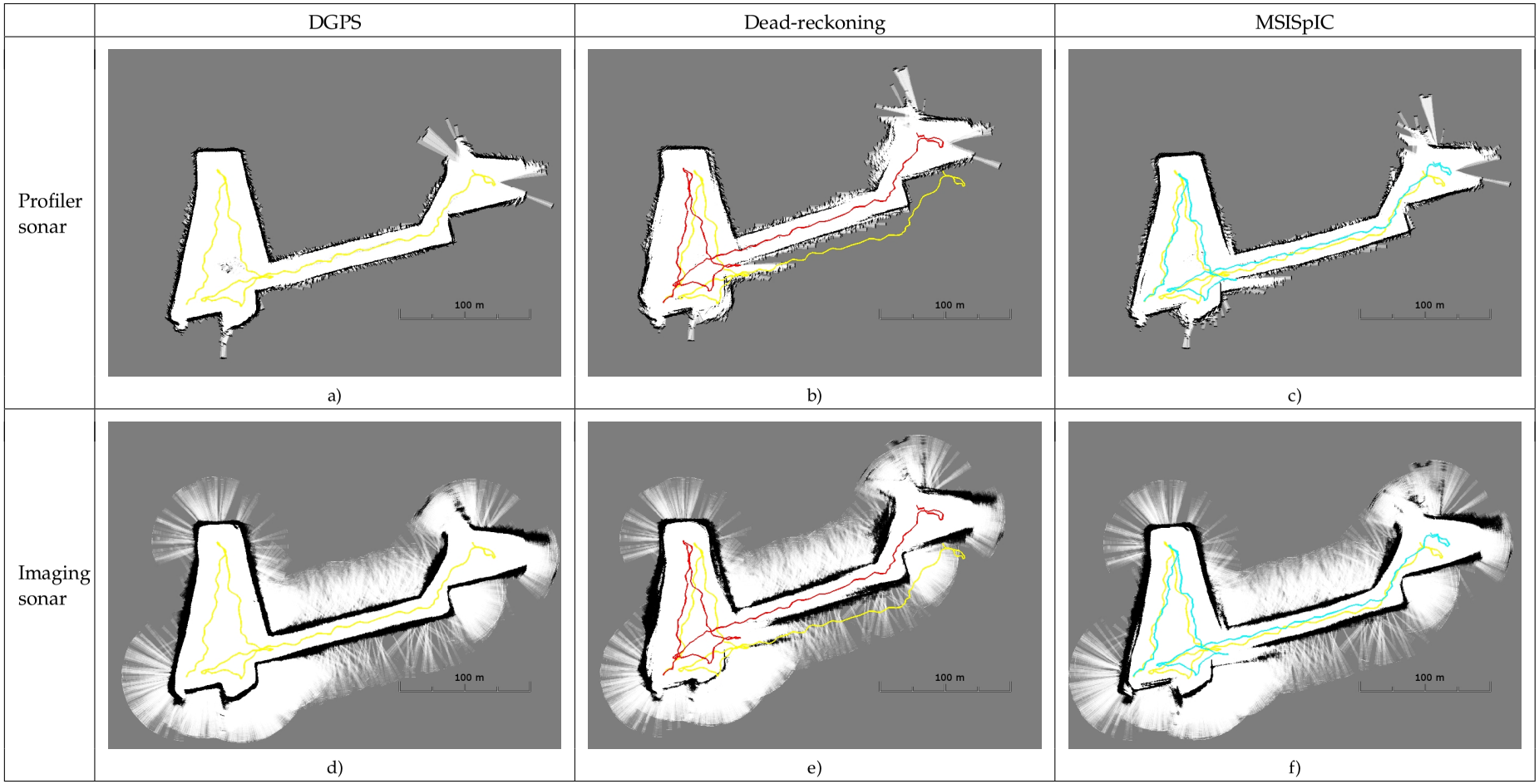


Figure 62: OGMs based on DGPS, dead-reckoning and MSISpIC trajectories. Each trajectory is represented using a profiler sonar model and an imaging sonar model.

been set at a fixed position in order to ensure the same topological graph construction, and homotopy classes generation, through different executions. The homotopy classes have been set at a maximum of 20 character length. In order to show all the possible results, no time restrictions have been taken into consideration.

6.2.1 *Cluttered Scenario*

This experiment shows the results obtained with the path planning method proposed in this dissertation in a cluttered environment with a low number of obstacles. The aim of this experiment is to show the whole set of the generated homotopy classes and to test the homotopic path planners in a tractable environment

The environment is represented by a 200x200 pixel bitmap with five irregular obstacles (see Figure 63). It has been strategically designed to avoid ideal conditions for the path planners. The obstacles present concave regions, hence the HA* can be temporally trapped exploring states which will not be in the optimal path cost for the homotopy class. The obstacles are relatively large in order to represent a challenging environment for the generation of the HRRT tree, whose growth depends on the chosen *step*. The step is empirically selected according to the environment. In this particular scenario, a step too large would require exploring fewer configurations in the C-space but more attempts to expand the tree would have to be done because the candidate nodes of the tree could be generated inside an obstacle; a step too small would make the tree expand easily, but more configurations would have to be explored. The HBug is based on the Bug2 algorithm which follows an obstacle boundary from a hit point to a leave point. The HBug works with the same principle but the hit and leave points are set according to the intersections of the obstacles with the lower bound path, whose quality depends on the number of the reference frame segments that are required to traverse. When a lower bound path is constrained by a high number of segments, it intrinsically contains more restrictions of the reference frame and therefore its quality is closer to the final solution. Based on this fact, the c point of the reference frame has been placed in a corner to minimize the intersections in the reference frame with the obstacles.

The construction of the reference frame, the topological graph and the generation of the homotopy classes with their lower bound computation took 7.9ms. Table 6 lists the homotopy classes sorted by their lower bound with the path cost for HA*, the HRRT and the HBug algorithms, which are also compared in Figure 64. The homotopy classes have been sorted according to their lower bound. To ensure the stability of the results, the path cost and time for the HRRT are the average of 100 executions using a *step* of 10 cells. Path costs and lower bounds have been normalized with respect to the A* path cost. Despite that most of the solutions obtained with the HBug have a lower cost than the HRRT solutions, the difference is very small. When compared with the HA*, the solution obtained for eighth class (index 13) with

Idx	Homotopy class	Lower bound	HA*	HRRT	HBug
2	$\alpha_{2_1}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\alpha_{5_1}$	0.89	**1**	**1.32**	**1.35**
8	$\beta_{2_2}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\alpha_{5_1}$	0.90	**1.03**	**1.37**	**1.32**
3	$\alpha_{2_1}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\beta_{5_2}$	0.97	**1.28**	**1.64**	**1.62**
9	$\beta_{2_2}\alpha_{1_1}\beta_{4_1}\alpha_{3_1}\beta_{5_2}$	0.99	**1.30**	**1.69**	**1.58**
1	$\alpha_{5_0}\alpha_{3_0}\alpha_{4_0}\alpha_{1_0}\alpha_{2_0}$	1.05	1.11	**1.42**	**1.31**
11	$\beta_{2_2}\beta_{1_2}\beta_{4_2}\alpha_{3_2}\beta_{5_2}$	1.06	1.36	**1.74**	**1.63**
10	$\beta_{2_2}\beta_{1_2}\beta_{4_2}\alpha_{3_2}\alpha_{5_1}$	1.08	1.26	**1.61**	**1.55**
13	$\beta_{2_2}\beta_{1_2}\beta_{4_3}\beta_{3_3}\beta_{5_2}$	1.12	1.18	**1.45**	**1.36**
5	$\alpha_{2_1}\beta_{1_2}\beta_{4_2}\alpha_{3_2}\beta_{5_2}$	1.24	1.71	**2.31**	**2.23**
4	$\alpha_{2_1}\beta_{1_2}\beta_{4_2}\alpha_{3_2}\alpha_{5_1}$	1.26	1.61	**2.18**	**2.13**
12	$\beta_{2_2}\beta_{1_2}\beta_{4_3}\beta_{3_3}\alpha_{5_1}$	1.27	1.45	**1.93**	**1.81**
7	$\alpha_{2_1}\beta_{1_2}\beta_{4_3}\beta_{3_3}\beta_{5_2}$	1.33	1.53	2.03	1.95
6	$\alpha_{2_1}\beta_{1_2}\beta_{4_3}\beta_{3_3}\alpha_{5_1}$	1.45	1.80	2.49	2.40

Table 6: Homotopy classes of the cluttered environment sorted by their lower bound with the cost of the paths computed using the HA*, HRRT and HBug algorithms. Costs in bold show the paths that would be computed when operating under realtime constraints.

the HBug is the closest one to the optimal path (1.15 times the HA* cost) and the solution obtained for thirteenth class (index 6) with the HRRT shows the largest difference (1.38 times the HA* cost).

Figure 63 depicts the paths of the four homotopy classes with smaller lower bound in Table 6 generated with the HA*, HRRT and HBug path planners. Figure 65 depicts the accumulated computation time for each path, which takes into account the computation of the reference frame, the topological graph, the homotopy classes with their lower bound and the paths generation. The HA* took almost 4.5s to generate paths for the whole set of homotopy classes being the class with index 1 the fastest to be generated (94.4ms) and the class with index 12 the slowest (603.8ms). The total computation time with the HRRT was reduced to 0.6s with the fifth class (index 1) being the fastest one to be generated (12.2ms) and the ninth class (index 5) the slowest (88.1ms). Finally, the total computation time using the HBug was only 8.5ms. The fastest path (index 10 class) was generated in $3.2\text{x}10^{-5}$s and the index 6 class path was the slowest one to be generated with only $7.6\text{x}10^{-5}$s. These low values justify the almost flat line for this path planner when compared with the HA* and the HRRT in Figure 65.

When operating under time restrictions, it is possible to stop the path search when the lower bound of the next homotopy class, whose path is going to be computed, is higher than the minimum cost of the paths already computed. In such cases, it is ensured that the best path has already been computed because it is not possible to obtain a path with a lower cost than its lower bound. For instance, in Table 6 the HA* would stop before computing a path for the

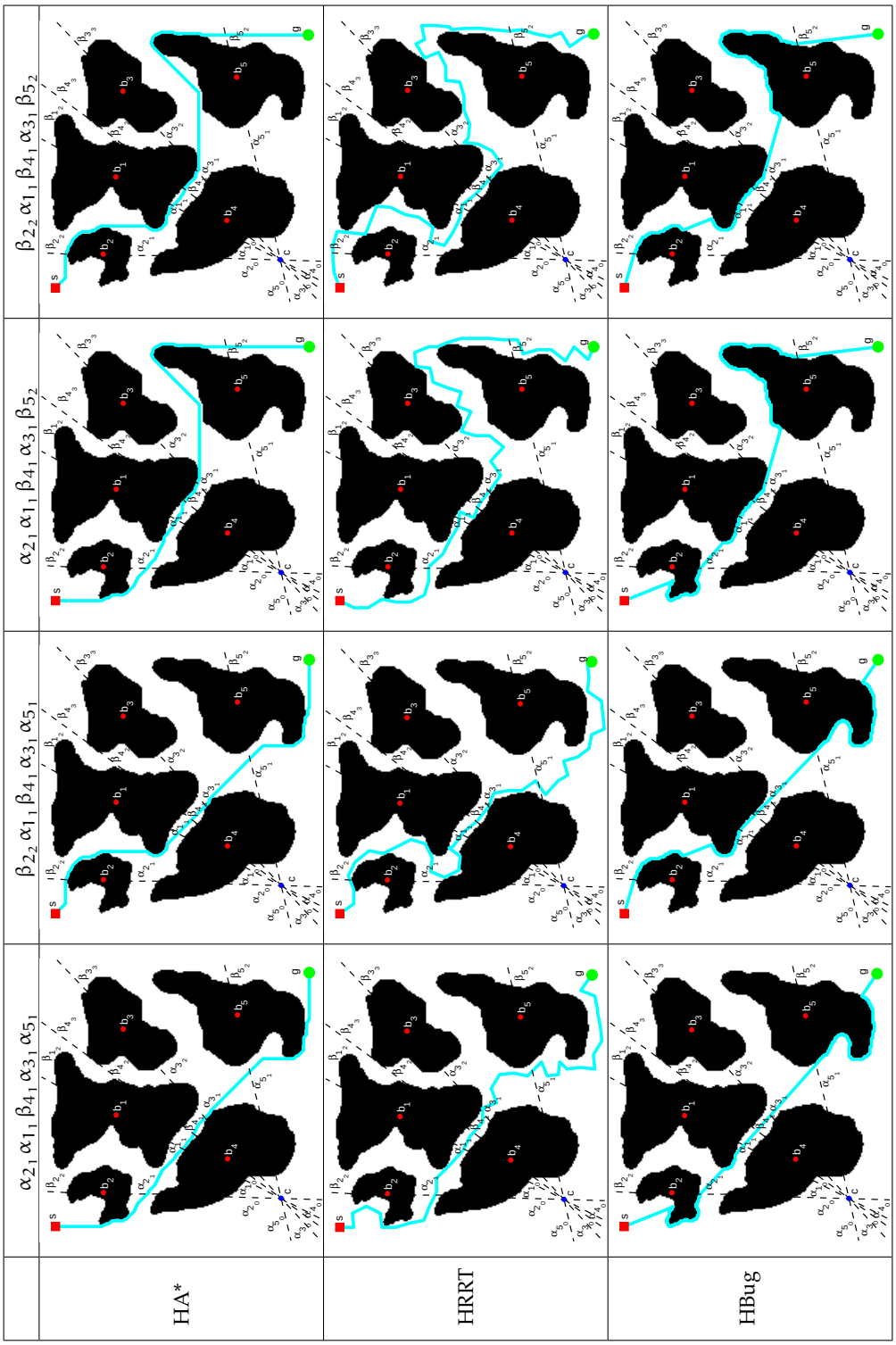


Figure 63: Paths generated with the HA*, HRRT and HBug algorithms for the four homotopy classes with the smaller lower bound in the cluttered environment.

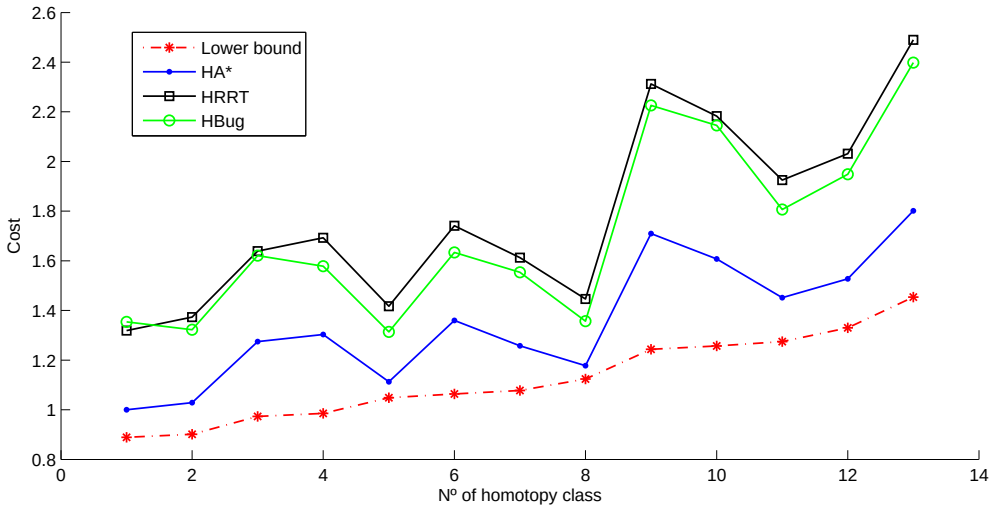


Figure 64: Cost of the paths computed with the HA*, HRRT and HBug algorithms for each homotopy class in Table 6 sorted according their lower bound.

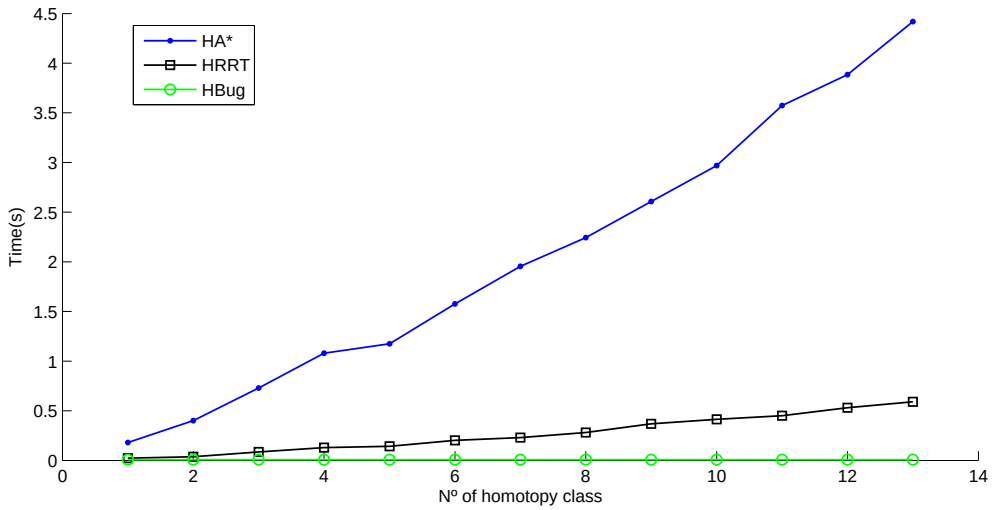


Figure 65: Accumulated computation time of the paths using the HA*, HRRT and HBug algorithms for each homotopy class in Table 6 sorted according their lower bound.

Idx	Homotopy class
25	$\alpha 15_{-1}\alpha 9_0\alpha 12_0\alpha 1_0\alpha 4_0\alpha 6_0\alpha 5_0\alpha 8_1\alpha 3_{-1}\alpha 11_2\alpha 13_2\beta 7_2\alpha 2_{-2}\alpha 14_2\beta 10_2$
26	$\alpha 15_{-1}\alpha 9_0\alpha 12_0\alpha 1_0\alpha 4_0\alpha 6_0\alpha 5_0\alpha 8_1\alpha 3_{-1}\alpha 11_2\alpha 13_2\beta 7_2\alpha 2_{-2}\beta 14_3\beta 10_2$
5	$\alpha 10_{-1}\alpha 14_{-1}\beta 2_1\alpha 7_{-1}\alpha 13_{-1}\alpha 11_{-1}\alpha 3_0\alpha 8_0\beta 5_1\alpha 6_1\alpha 4_{-1}\alpha 1_{-2}\alpha 12_2\beta 9_2\alpha 15_2$
1	$\alpha 10_{-1}\alpha 14_{-1}\beta 2_1\alpha 7_{-1}\alpha 13_{-1}\alpha 11_{-1}\alpha 3_0\alpha 8_0\alpha 5_0\alpha 6_0\alpha 4_0\alpha 1_{-2}\alpha 12_2\beta 9_2\alpha 15_2$
41	$\alpha 15_{-1}\alpha 9_0\alpha 12_0\alpha 1_0\beta 4_1\alpha 6_{-1}\alpha 5_{-1}\alpha 8_1\alpha 3_{-1}\alpha 11_2\alpha 13_2\beta 7_2\alpha 2_{-2}\alpha 14_2\beta 10_2$

Table 7: The five homotopy classes of the large environment with the smaller lower bound and their generation index.

fifth homotopy class (index 1) since its lower bound (1.05) is higher than the path length obtained with the first class (index 2, cost 1). The accumulated computation time would be 1.54s (see Figure 65). On the other hand, the HRRT and the HBug algorithms would stop before computing the path of the two last classes with a lower computation time than the HA* (0.53s for the HRRT and $8.4\text{x}10^{-3}$s for the HBug) at the expense of higher path costs.

6.2.2 *Large Scenario*

This experiment tests the scalability of the path planning proposed in this thesis with a bitmap of 1000x1000 pixels with 15 irregular obstacles, depicted in Figure 66. The construction of the reference frame, the topological graph and the generation of 112 homotopy classes with their lower bound computation took 0.304s.

The next three sections present the results obtained with the HA*, HRRT and HBug algorithms respectively. In each section, the paths obtained with the corresponding path planner for the five homotopy classes with smaller lower bound are also shown and listed in Table 7. The last section makes two comparisons: the first one compares the homotopic path planners with their non-homotopic versions: the A* and the ARA* for the HA*, the RRT and the ARRT for the HRRT, and the Bug2 for the HBug. The A*, the RRT and the Bug2 are designed to compute only one path towards the goal. The ARA* and the ARRT compute several paths but without taking into account their homotopy. Because of this, only the solutions generated with the homotopy classes in Table 7 are used in the comparison with these well-known path planners. The second comparison shows the results of the HRRT and the HBug against the HA*, since it computes the shortest path for a given homotopy class.

6.2.2.1 *Results Obtained with the HA**

This section shows the results obtained with the HA* algorithm. The path planner has been used to compute the path for all 112 homotopy classes in the scenario. The paths of the homotopy classes with the smaller lower bound, shown in Table 7, are depicted in Figure 66. Figure 67 shows the cost and computation time for each homotopy class path computed with the HA*. The

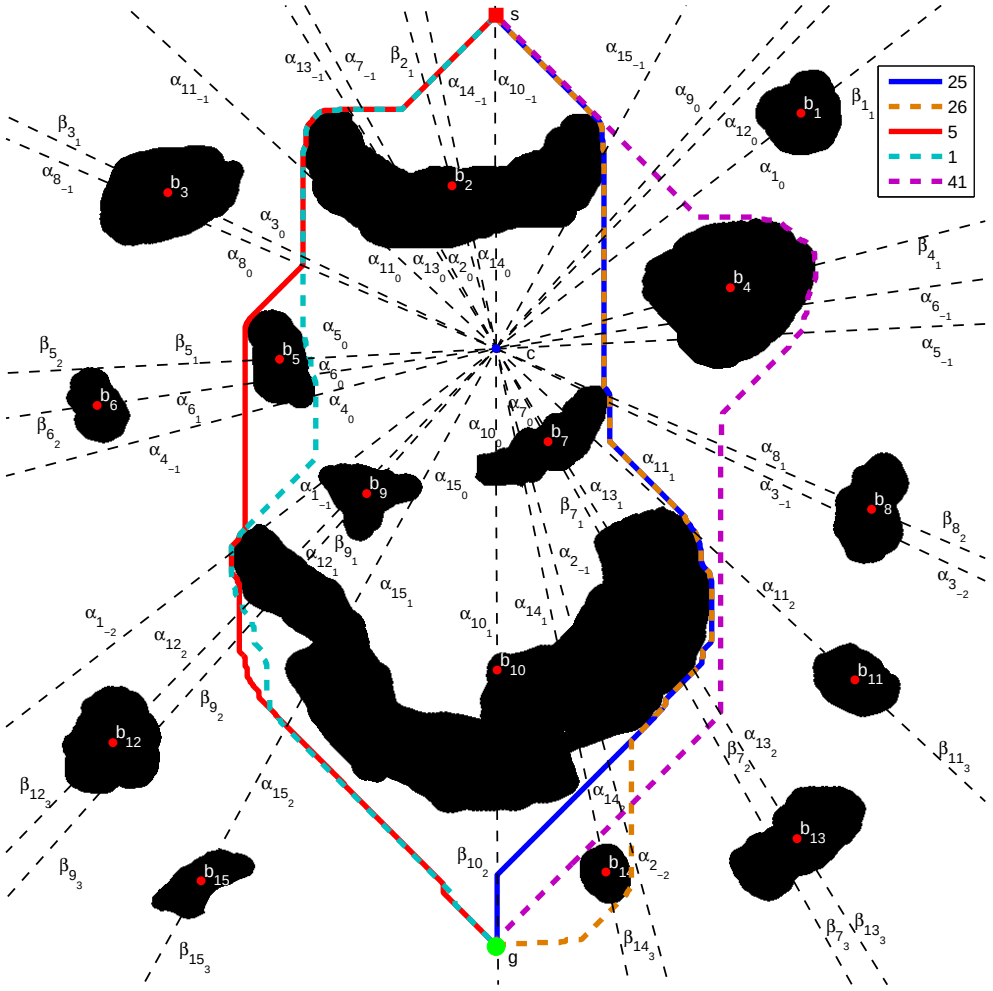


Figure 66: Paths of the five homotopy classes with the smaller lower bound computed with the HA*. The class associated to the index can be found in Table 7.

homotopy classes have been sorted according to their lower bound. Notice that the path cost and the lower bound have been normalized with respect to the A* cost.

The fastest path was generated in 7.49s and corresponds to the first class (index 25), which coincides with the global optimal solution. The slowest took 53.69s (class 75) to be generated with a cost 1.6 times of the global optimal solution. The computation time between different homotopy classes may change substantially depending on the number of states that the HA* has to consider. The mean computation time was 30.99s. On one hand, it cannot be accepted for a realtime application since generating the optimal path for the whole set of homotopy classes would take around 58min. On the other hand, when operating under time restrictions, it would be possible to stop the path generation process before computing the path of the third homotopy class (with index 5 in Table 7) since its lower bound is higher than the cost obtained with the first homotopy class (index 25). In such cases, the accumulated computation time for the path generation would be 30.94s (31.25s

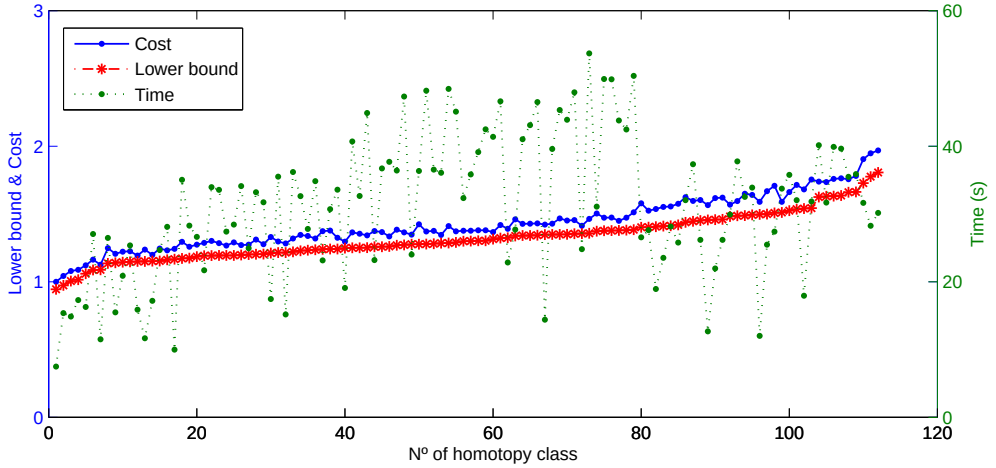


Figure 67: Normalized cost, normalized lower bound and computation time for paths generated with the HA* for each homotopy class.

taking into account the computation of the reference frame, the topological graph and the generation of the homotopy classes with their lower bound).

6.2.2.2 *Results Obtained with the HRRT*

This section shows the results obtained with the HRRT algorithm. The path planner has been used to compute the path for all the homotopy classes in the scenario. Figure 68 depicts the path of the homotopy classes with the smaller lower bound, shown in Table 7, computed with the HRRT. The algorithm has been empirically configured with a *step* and a *distThreshold* of 15 cells. In order to stabilize the results, Figure 69 shows the average cost and computation time of 100 executions for each homotopy class path computed with the path planner. The homotopy classes have been sorted according to their lower bound. The path cost and the lower bound have been normalized with respect to the A* cost.

The best solution computed with the HRRT was obtained in 0.069s, again with the first class (index 25), with a cost 1.4 times the global optimal solution. However, the fastest path was generated in 0.038s (class 14, index 93) with a path cost 1.83 times the global optimal solution. On the other hand, the slowest (class 70, index 19) took 1.248s with a cost of 2.19 times the global optimal solution. The mean computation time for each path was only 0.239s, almost 130 times faster than the HA* at the expense of computing higher cost solutions.

In this scenario, the paths for the whole set of homotopy classes were computed in 26.84s, which is four seconds faster than the HA* assuming it is operating under time restrictions. However, none of the paths generated was the optimal solution for its homotopy class. When operating under time restrictions, it would be possible to stop the path computation with the HRRT before computing the eightieth homotopy class (index 35) since its lower bound is higher than the cost obtained with first homotopy class (index 25). In such cases, accumulated computation time for the path generation would be 18.863s

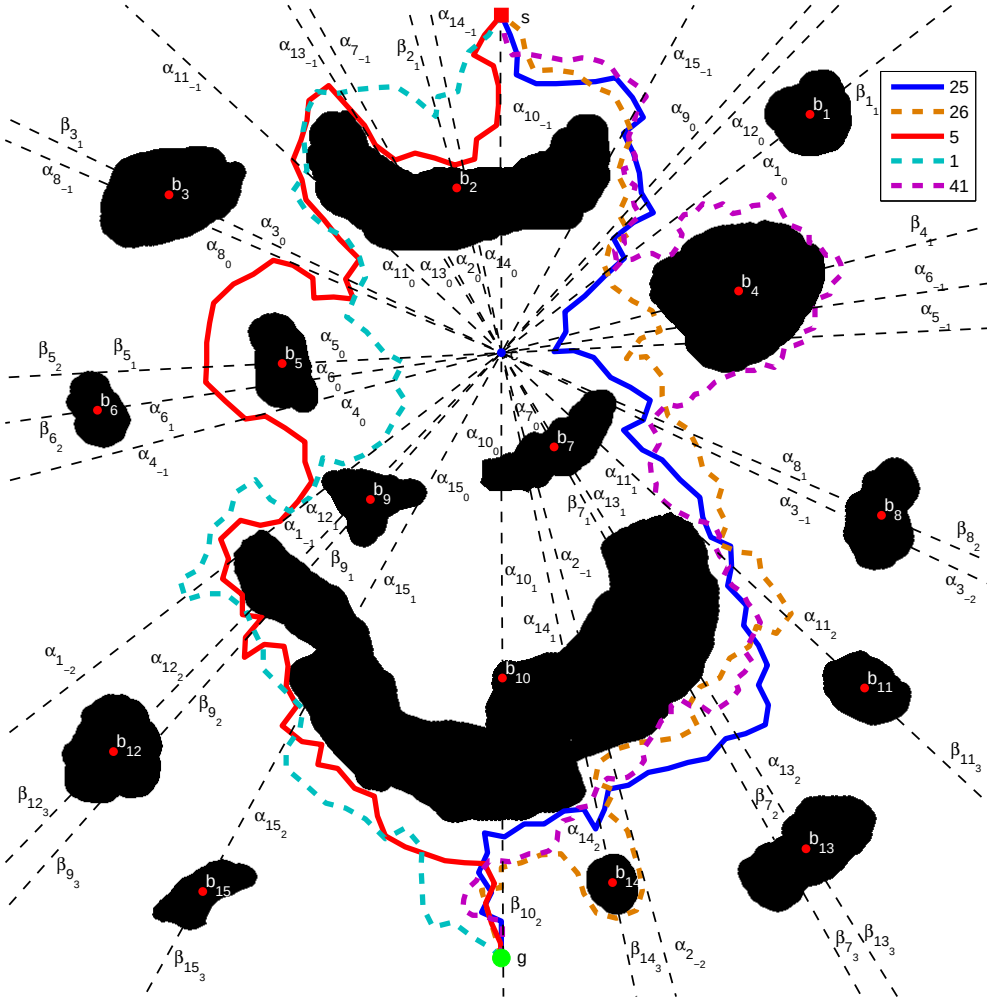


Figure 68: Paths of the five homotopy classes with the smaller lower bound computed with the HRRT. The class associated to the index can be found in Table 7.

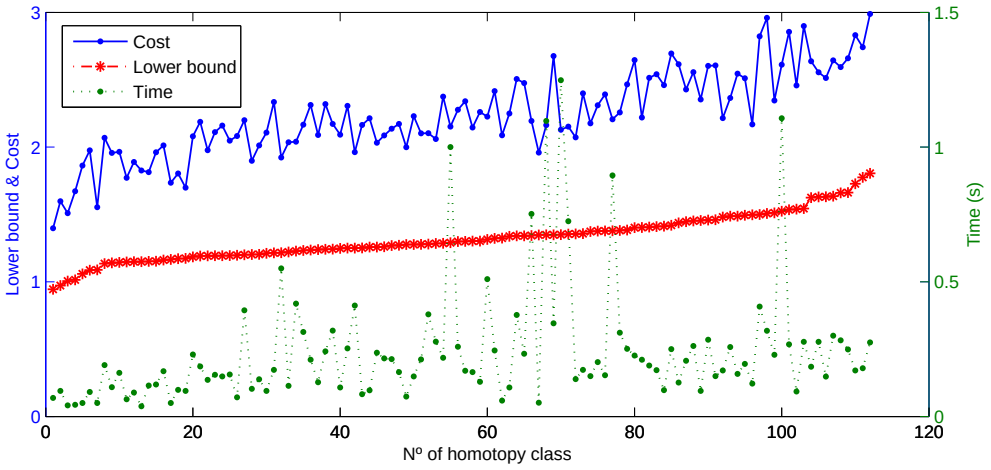


Figure 69: Normalized cost, normalized lower bound and computation time for paths generated with the HRRT for each homotopy class.

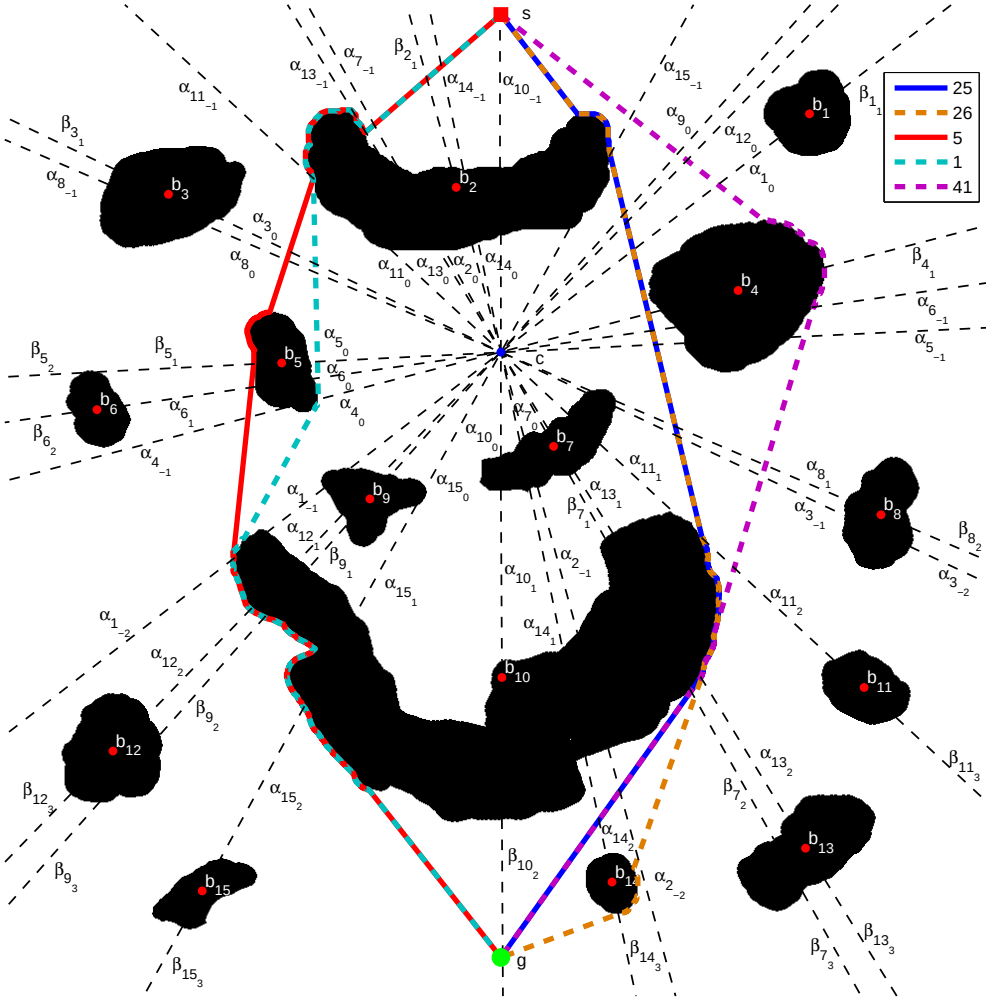


Figure 70: Paths of the five homotopy classes with the smaller lower bound computed with the HBug. The class associated to the index can be found in Table 7.

(19.167s taking into account all the steps in the method), which is 12s faster than the HA*.

6.2.2.3 *Results Obtained with the HBug*

This section shows the results obtained with the HBug algorithm. The path planner has been used to compute the path for all the homotopy classes in the scenario. The HBug paths for the homotopy classes with the smaller lower bound, shown in Table 7, are depicted in Figure 70. Figure 71 shows the cost and computation time for each homotopy class path computed with the HBug. The homotopy classes have been sorted according to their lower bound. As with other path planner solutions, the path cost and the lower bound have been normalized with respect to the A* path cost.

The fastest path was generated in $1.12 \text{x} 10^{-4}$s. It corresponds to the first class (index 25) with a path cost only 1.05 times the optimal solution. The HBug is not intended to provide the optimal solution, however, placing the c point of the reference frame in a central position combined with a relatively high

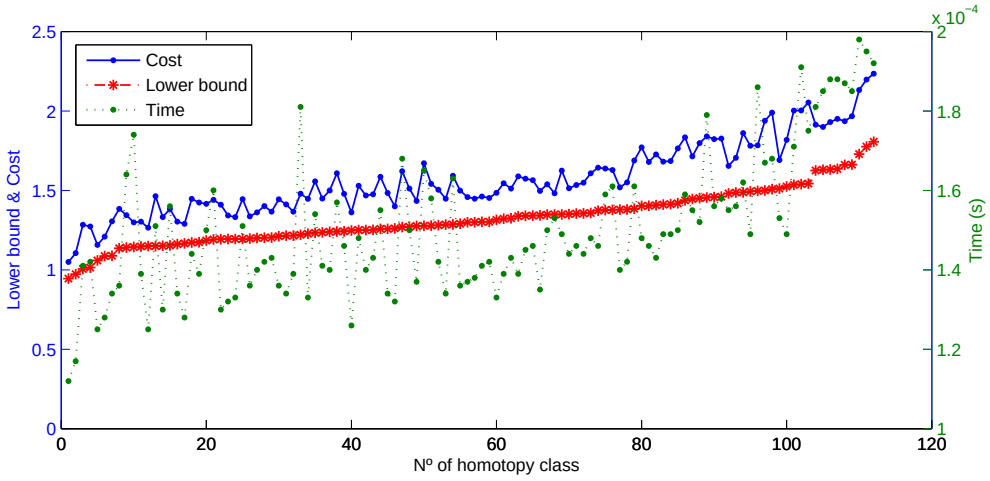


Figure 71: Normalized cost, normalized lower bound and computation time for paths generated with the HBug for each homotopy class.

number of obstacles, makes each reference frame line intersect with several obstacles at the same time. Hence, the number of segments that constrain the solution is relatively high, generating a lower bound path, in which the HBug solution is based, very close to the global optimal path.

The homotopy class 110 (index 102) was the one that took the most time to compute ($1.98x10^{-4}$s) with a cost 2.24 times above the global optimal solution. The mean computation per path was only $1.50x10^{-4}$s, almost 1600 times faster than the HRRT and $2x10^{5}$ times faster than the HA*. In this environment, the paths for the whole set of homotopy classes were computed in 16.8ms, which is almost negligible when compared with the 304ms spent in the generation of the reference frame, topological graph and the homotopy classes with their lower bound. Moreover, when operating under time restrictions, it is possible to stop the path computation before computing the fifth homotopy class (with index 41 in Table 7) since its lower bound is higher than the cost obtained with the first homotopy class (index 25). In such cases, the accumulated computation time for the path generation would be only $5.12x10^{-4}$s (304.5ms taking into account all the steps in the method).

The HBug shows a very good performance because part of its solution is implicitly generated in the lower bound computation. When the lower bound path intersects with an obstacle in the scenario, its boundary is followed in a clockwise or counterclockwise direction according to the homotopy class until the lower bound path leaves the obstacle. Notice that the whole boundary of each obstacle is already computed by the CL algorithm.

6.2.2.4 *Comparative Results*

In this section a comparison of the proposed path planners against the A*, the RRT, their respective anytime versions (ARA* and ARRT) and the Bug2 algorithm is presented. The A*, the RRT and the Bug2 algorithms are designed to compute only one path towards the goal; the ARA* and ARRT compute

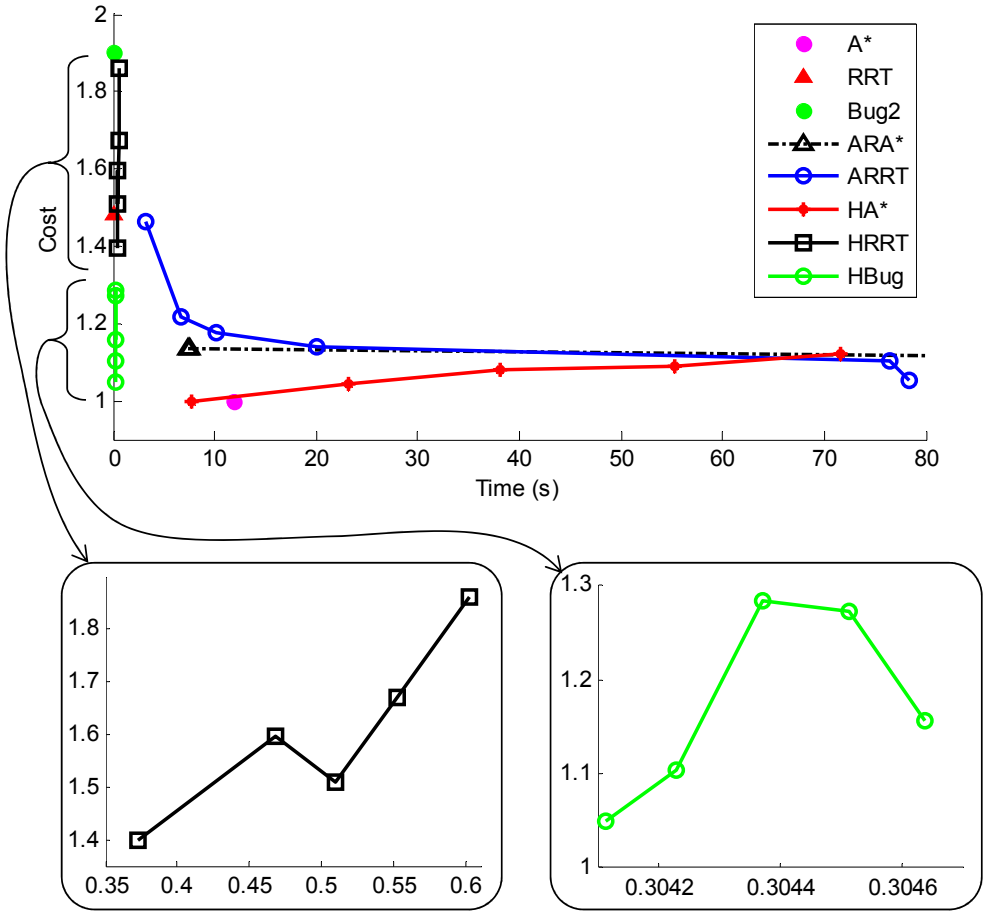


Figure 72: Comparison of the HA*, HRRT and HBug paths of the five homotopy classes with the smaller lower bound vs A*, RRT, ARA*, ARRT and Bug2 algorithms.

several paths but without taking into account their homotopy. Because of this, the comparison with these well-known path planners is against the five homotopy classes with the smaller lower bound. These classes are shown in Table 7. Figure 72 depicts the comparison. In order to ensure the stabilization of the results of the probabilistic path planners, the data obtained with the RRT and the ARRT are the average of 100 executions. Notice that all the time values of the HA*, the HRRT and the HBug include the computation time of the reference frame and topological graph construction, and the generation of the homotopy classes with their lower bound.

The A* returned the optimal path in 11.90s. The ARA* generated the first solution in 7.40s and found the optimal solution after 301s. The RRT algorithm took 0.012s to compute a path with a cost 1.48 times the global optimal solution. The ARRT took 3.13s to obtain the first solution and 78.30s to compute all of them, ensuring that any new solution generated is closer to the optimal one. The Bug2 algorithm computed the path in 0.044s with a cost 1.90 times the optimal solution. In order to obtain the best possible path with this algorithm, we have manually chosen the directions to surround the obstacles: the m-line, which connects the start with the goal, intersects with the obstacles labeled b_1,

b_7 and b_{10}; the directions are clockwise for b_1, counter-clockwise for b_7 and clockwise for b_{10}.

The HA* computed the optimal path (index 25) in 7.79s and obtained the path for the five selected homotopy classes in 71.61s. The computation of the optimal path of the best homotopy class with the HA*, which corresponds to the A* solution, was 1.53 times faster than the A*. The computation of the first path took 0.390s longer than the first solution of the ARA*, but the ARA* needed more iterations to refine the path to obtain the same solution as the HA*.

Using the HRRT, the best solution (index 25) was computed in 0.373s with a cost 1.40 times the optimal one, and obtained the path for the five homotopy classes with the smaller lower bound in 0.603s. The HRRT computation time was better than the A* and the ARRT. The reduced computation time of the RRT could not be reached due to the computation of the homotopy classes and the extra load of checking the topological restrictions of the HRRT. Despite that the class with index 25 had a lower cost than the RRT solution, the difference was small.

Using the HBug, our path planning method computed the best solution (index 25) in 0.304s with a cost 1.05 times the optimal one, and obtained the path for the whole set of homotopy classes in 0.321s. As stated in the previous section, the computation of the paths with the HBug for each homotopy class offers a very good performance. Only the RRT and Bug2 had lower computation times at the expense of finding higher cost solutions. Notice that most of the time was spent in the computation of the reference frame, the topological graph and the generation of the homotopy classes with their lower bound.

Until now, the quality of all the solutions has been compared with the A* cost, which generates the global optimal path. However, the standard A* is not able to take into account any topology during the path generation. On the other hand, the HA* computes the shortest path for each homotopy class, which is the optimal solution according to the topological constraints. Therefore, in Figure 73, a comparison of the solutions generated with the HRRT and the HBug against the HA* cost for each specific homotopy class is depicted.

The HRRT generates solutions between 1.35 (class 15, index 73) and 1.82 (class 83, index 61), with a mean of 1.57 times with respect to the optimal path cost for the specific homotopy class generated with the HA*. As stated before, the c point in the reference frame is placed amid the obstacles in the scenario, which lets each line in the reference frame intersect with several obstacles, allowing the computation of lower bound paths closer to the final solution when using the HBug. This path planner generates solutions between 1.03 (class 5, index 41) and 1.19 (class 100, index 101), with a mean of 1.1 times with respect to the optimal path cost computed with the HA*.

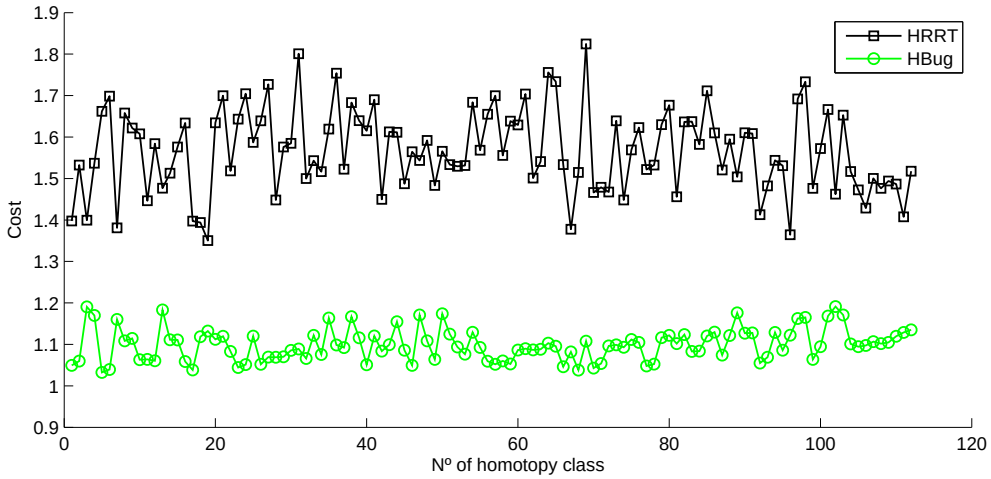


Figure 73: The HRRT and HBug paths cost with respect to the HA* cost for each homotopy class.

6.2.3 *Discussion*

The method of generating the homotopy classes was initially combined with the HRRT (Hernández et al., 2011a,c) because RRT-based solutions have been used with success in contexts where high performance is required. Although it is complete when used together with the homotopy classes generation method, and probabilistically complete when not, it obtains suboptimal solutions.

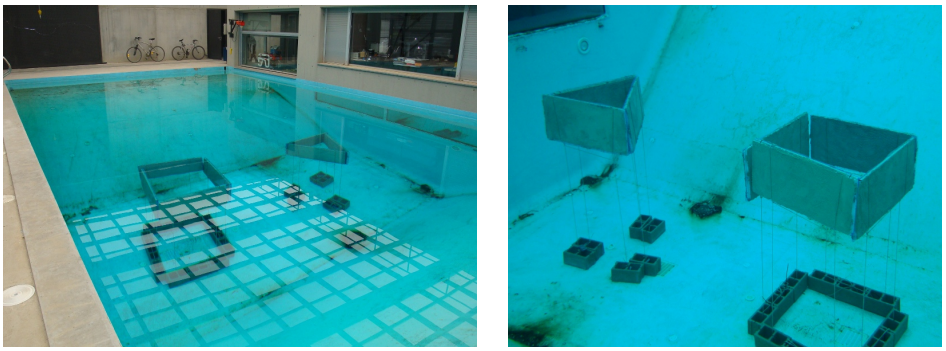
The HA* (Hernández et al., 2011b) computes the optimal path cost for an input homotopy class. It is a graph-based search algorithm that exhaustively explores the search space. Because of this, it is the slowest of the solutions proposed, but it can be used to provide a ground truth.

The HBug (Hernández et al., 2012) was proposed to take the maximum advantage of the homotopy classes generation process, which includes the computation of the lower bound. This algorithm tries to follow the lower bound path which is already computed. When it intersects an obstacle, the algorithm follows the obstacle's boundary in the direction which accomplishes the homotopy class until the lower bound path is reached again. With this simple process it is possible to generate a path according to a homotopy class with a very low computational load, which implies a high performance.

As a conclusion, the HA* should be used when generating optimal solutions at the expense of low performance is required. On the other hand, the HBug is suitable for applications where the time in which to perform path planning is highly constrained. Finally, the HRRT should be avoided since the HBug generates, most of the times, lower cost solutions with a portion of the computation time.

6.3 A WATER TANK ENVIRONMENT TEST

The aim of this experiment was to test the map building procedure together with the path planning algorithms proposed in this dissertation. The experi-



a) Underwater Robotics Lab. water tank

b) Detail of the two obstacles

Figure 74: Water tank environment set up.

ment was carried out using the Sparus AUV (see Chapter 5) in the Underwater Robotics Lab. of the University of Girona. In order to stack obstacles of different shapes and sizes in the water tank to create a controlled unknown scenario, we built a set of plates, each made of a 1.20x0.60x0.04m roof insulator panel covered with plastic mesh and concrete on both sides. The insulator is a cheap alternative to wood and offers buoyancy while the concrete adds weight and allows simulating a harbor's texture and acoustic reflection. Notice that none of the materials in the panels are ferromagnetic, hence the vehicle's compasses are not affected while navigating at close distance.

For the experiment, triangular and square obstacles where set up. Each panel was stacked at a 3m depth using weights (Figure 74). The MSIS was configured to scan the whole 360° sector and was set to fire up to a 5m range with a 0.1m resolution and a 1.8° angular step. The robot's trajectory is based on dead-reckoning, computed using the velocity readings coming from the DVL and the heading data obtained from the MRU sensor, both merged with an EKF. To avoid perturbations in the DVL measurements, the test zone was limited to the maximum depth area, which is at the centre of the water tank.

6.3.1 *Map Building*

During the experiment, the robot was teleoperated through the obstacles at a 3m depth. Figure 75 depicts the range data obtained with the MSIS according to the dead-reckoning trajectory estimated with an EKF. Figure 76 shows the raw map obtained with the trajectory computed with the MSISpIC, which has been used to improve the dead-reckoning (Hernández et al., 2009a). Since the Global Positioning System (GPS) is not accurate inside the Underwater Robotics Lab., there is no ground truth trajectory to compare with. Nevertheless, it can be appreciated that the raw map is less sparse when using the trajectory estimated with the MSISpIC algorithm.

The corrected trajectory was used to compute an OGM with a 0.1m resolution, depicted in Figure 77.a. The inverse sensor model of the MSIS was implemented as a profiler sonar sensor with an overture of 3° as explained in Chapter 4. Since the OGM was also used as a C-space for the path planners,

every obstacle perceived was expanded according to the AUV's size in order to generate paths that would be feasible to follow with the vehicle.

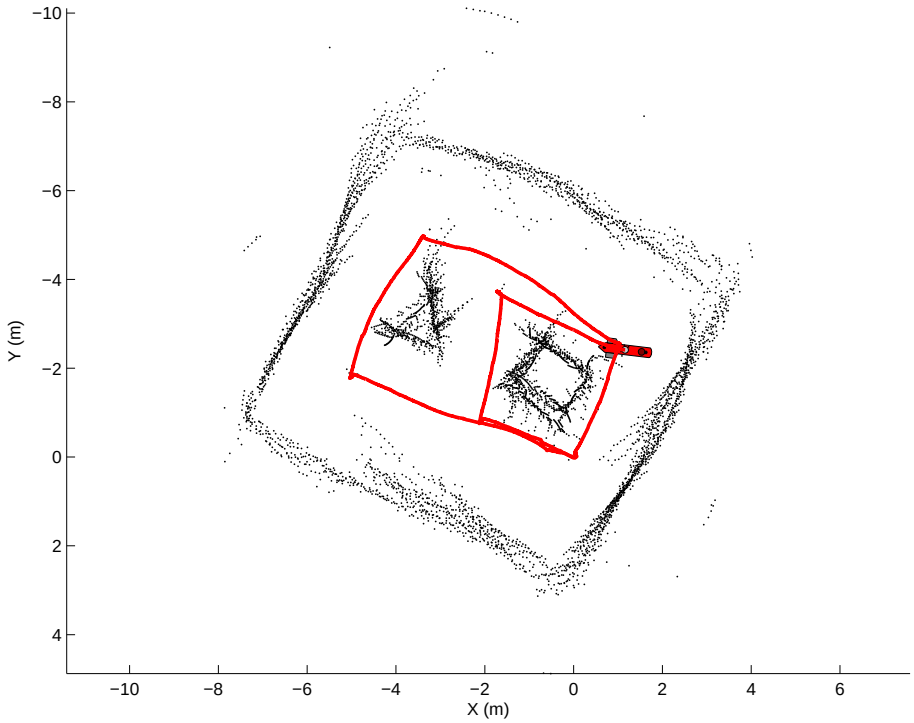


Figure 75: A dead-reckoning trajectory with range data.

6.3.2 *Path Planning with Homotopy Constraints*

Figure 77.a depicts the OGM computed with MSISpIC trajectory. The environment is represented by an 8-bit greyscale image in which each pixel depicts a cell of the OGM. Since the proposed path planning algorithms only distinguish between free cells and occupied cells, a pixel value between 0 and 127 represents an occupied cell, whereas a value between 128 and 255 represents a free cell. As stated before, the OGM is also used as a C-space assuming that the vehicle is a single point without area. The figure also depicts the obstacles identified by the CL algorithm and the reference frame. Its topological graph is shown in Figure 77.b.

With the method described in Chapter 3, four homotopy classes were generated. The computation time to apply the CL algorithm, construct the reference frame with its topological graph and generate the homotopy classes with their lower bound took 157.8ms. Table 8 shows the homotopy classes classified according to their lower bound and the paths'length obtained with the HA*, the HRRT and the HBug algorithms. Table 9 shows the computation time for each single path with the proposed path planners that accomplish the homotopy classes. To ensure the stabilization of the results, the HRRT cost and computation time are the average of 100 executions using a *step* and a *distThreshold* of 0.5m. Finally, Figure 78 depicts the path generated for each

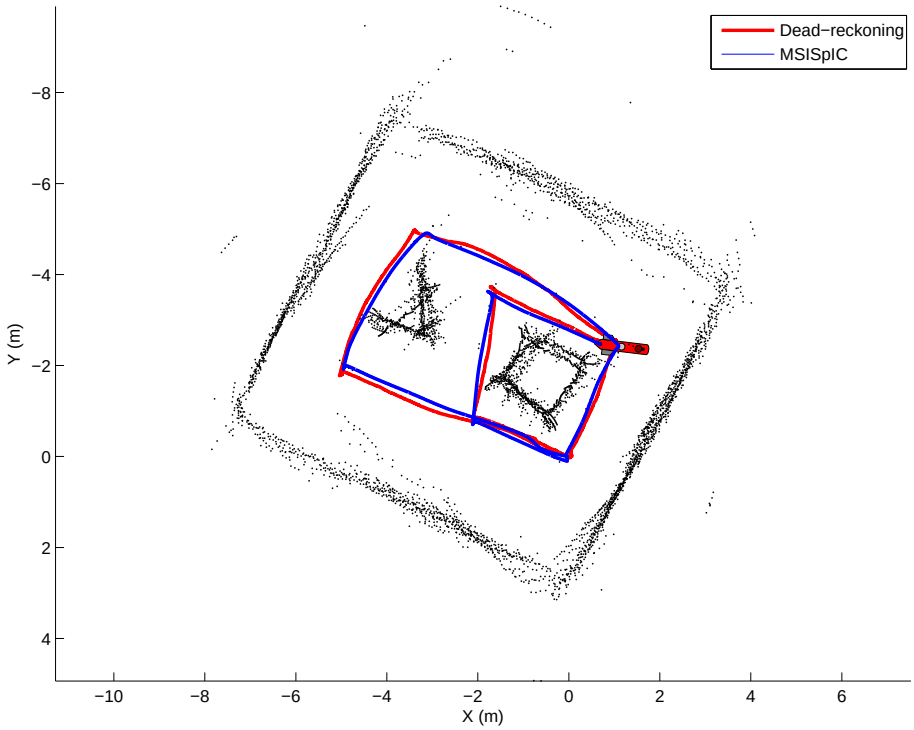


Figure 76: A trajectory estimated with scan matching against dead-reckoning. The range data is plotted according to the MSISpIC trajectory.

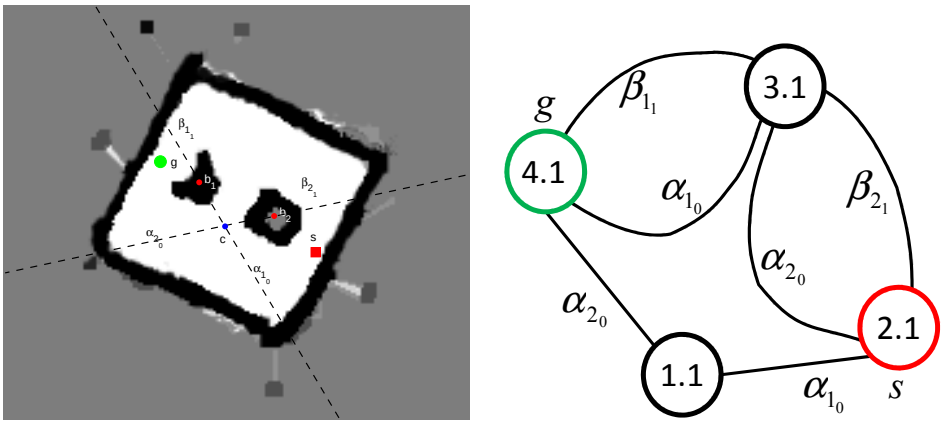
Idx	Homotopy class	Lower bound(m)	HA*(m)	HRRT(m)	HBug(m)
1	$\alpha_{2_0}\alpha_{1_0}$	7.00	7.88	11.20	9.89
2	$\alpha_{2_0}\beta_{1_1}$	7.11	8.76	13.77	10.81
4	$\beta_{2_1}\beta_{1_1}$	7.50	8.52	11.94	10.53
3	$\beta_{2_1}\alpha_{1_0}$	7.75	9.16	14.17	11.14

Table 8: Homotopy classes generated for the Underwater Robotics Lab. environment with their length sorted according to the lower bound.

homotopy class using the HA*, the HRRT and the HBug. In this scenario the best results are achieved with the HA* since it computes the optimal path for each homotopy class taking only 218.168ms to accomplish all the steps of the proposed method. The HRRT and HBug compute the paths faster at the expense of higher path costs. However, for this scenario the time differences between the homotopic path planners are not significative: around 58ms and 60ms more than the HRRT and the HBug respectively.

6.4 EXPERIMENT IN THE FORMIGUES ISLANDS

The path planning proposal presented in this thesis has also been considered as a part of the TRIDENT European project (EU FP7 ICT-248497). This project is focused on providing a new methodology for multipurpose underwater intervention tasks with diverse potential applications like underwater archaeology,



a) C-space with reference frame

b) Topological graph

Figure 77: An 18x16m OGM with 0.1m resolution used as a C-space and its topological graph.

Idx	Homotopy class	Lower bound(m)	HA*(ms)	HRRT(ms)	HBug(ms)
1	$\alpha_{2_0}\alpha_{1_0}$	7.00	10.006	0.253	0.013
2	$\alpha_{2_0}\beta_{1_1}$	7.11	10.689	0.863	0.012
4	$\beta_{2_1}\beta_{1_1}$	7.50	12.882	0.424	0.013
3	$\beta_{2_1}\alpha_{1_0}$	7.75	26.791	1.312	0.011
		Total paths	60.368	2.852	0.049
		Total process	218.168	160.382	157.849

Table 9: Homotopy classes generated for the Underwater Robotics Lab. environment with their computation time using the homotopic path planners. The homotopy classes have been sorted according to their lower bound.

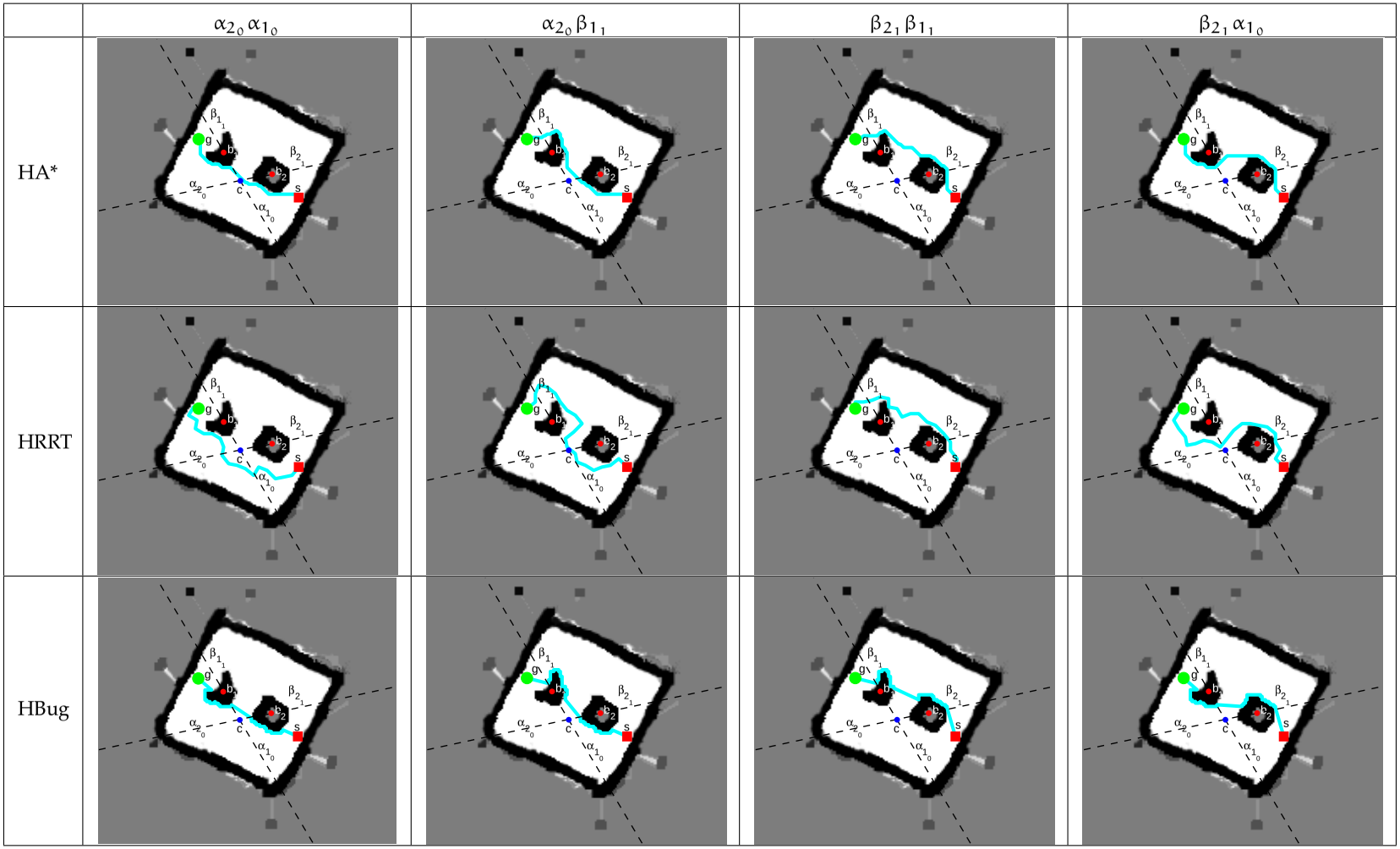


Figure 78: Paths generated with the HA*, the HRRT and the HBug algorithms for each homotopy class generated in the Underwater Robotics Lab. environment.

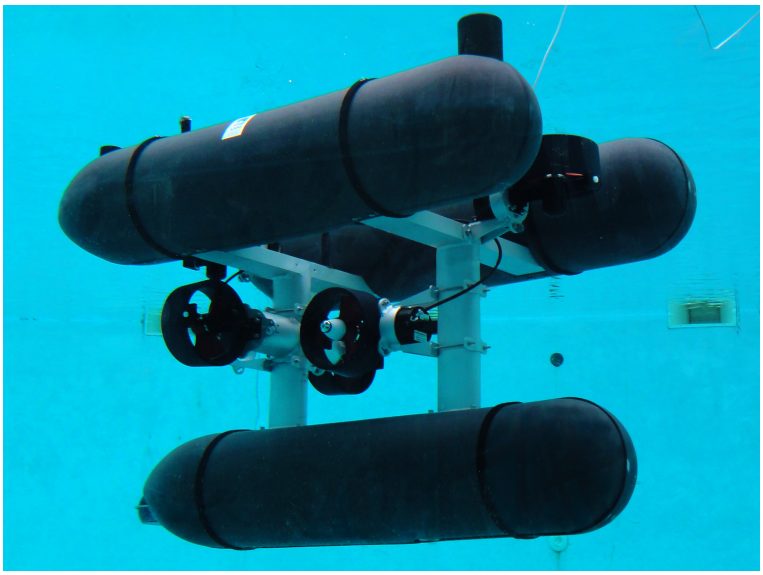


Figure 79: Girona 500 I-AUV

oceanography and offshore industries, going beyond present-day methods typically based on manned and/or purpose built systems. A team of two cooperative heterogeneous robots with complementary skills, an Autonomous Surface Craft (ASC) and an I-AUV endowed with a dexterous manipulator will be used to perform underwater manipulation tasks.

The experiments of the project consist of two steps. In the first step, the I-AUV is deployed from the ASC to perform a path following survey, in which it gathers optical/acoustic data from the seafloor to do an accurate terrain tracking. After the survey, the I-AUV docks in the ASC and sends the data back to a ground station where a map is set up and a target object is identified by the end user. In the second step, the ASC navigates towards a waypoint near the intervention area where the I-AUV is launched to search for the object. When the object has been found, the I-AUV switches to free floating navigation mode to start the manipulation process.

The I-AUV that will be used to carry out the experiments is the Girona 500 (Ribas et al., 2011), depicted in Figure 79. The vehicle is equipped with a phased array DVL Explorer from RDI, an Attitude and Heading Reference System (AHRS) from Tritech composed of an Intelligent Gyro Compass (IGC) for attitude and an Intelligent Fiber-optic Gyro (IFoG) for heading, a Linquest Ultra-Short BaseLine (USBL) 1500HA with modem, a Super Seaking dual frequency profiling sonar from Tritech, a Sound Velocity System (SVS) with a pressure sensor from Valeport, a GPS sensor, an Imaginex Sidescan sonar, a Tritech SeaSpy CCD camera and an Imagenex MPS.

The results presented in this dissertation focus on the path planning approach to be used in the second step of the experiments when the I-AUV has to compute safe paths for the intervention based on the generated map. The method has been applied to a bathymetric map. The path planning method is in charge of generating the different homotopy classes for the environment. After that, the HA* algorithm computes the shortest path for each class. Since

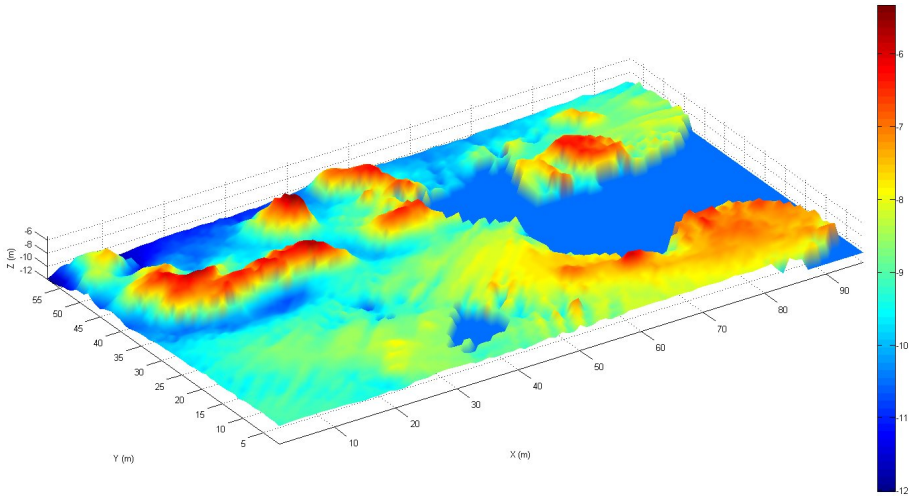


Figure 80: Bathymetric map obtained in the Formigues Islands.

the target has to be selected by the user once the vehicle has performed the survey and reached the surface, time is not a constraint.

The bathymetry was gathered with a MPS Model 837B "Delta T" 1000 from Imagenex. This sensor is a multiple receiver sonar system designed to provide video-like imaging using sonar technology. The MPS has 480 beams spread in a 120° swath overture, the beam rate frequency is between 5-10Hz, depending on the depth of the scanned area. The sensor has an MRU sensor to capture roll, pitch and heading.

The experiment took place in the Formigues Islands, on the Catalan Coast. The MPS was fastened to a mast with a DGPS sensor. The mast was attached to a boat to perform a survey mission in an area of 100x58m. The datasets gathered with the MPS and the DGPS were merged with the commercial software provided by Imagenex to generate the bathymetric map, depicted in Figure 80 with a 0.2m resolution.

In the experiment, it was assumed that the vehicle would navigate at a 7.5m depth. Therefore, using an OGM technique (Hernández et al., 2009a), the cells in the bathymetric map with a lower depth were mapped as occupied and the cells with a higher depth were mapped as free. Figure 81 depicts the resultant C-space as a 500x290 bitmap. The construction of the reference frame, the topological graph and the generation of 45 homotopy classes with their lower bound computation took 0.273s. Figure 82 depicts the normalized cost with respect to the optimal path cost computed with the A* algorithm and the computation time for each homotopy class sorted by their normalized lower bound. Table 10 shows the five best homotopy classes according to their lower bound and their paths are depicted in Figure 81. The whole process was computed in 99.2s. When operating under realtime constraints, it is possible to stop before computing the class 11 path. In such cases, the process would take 34.396s.

Changing the start point and/or the goal point may change the number of homotopy classes generated. For instance, Figure 83 depicts the five best

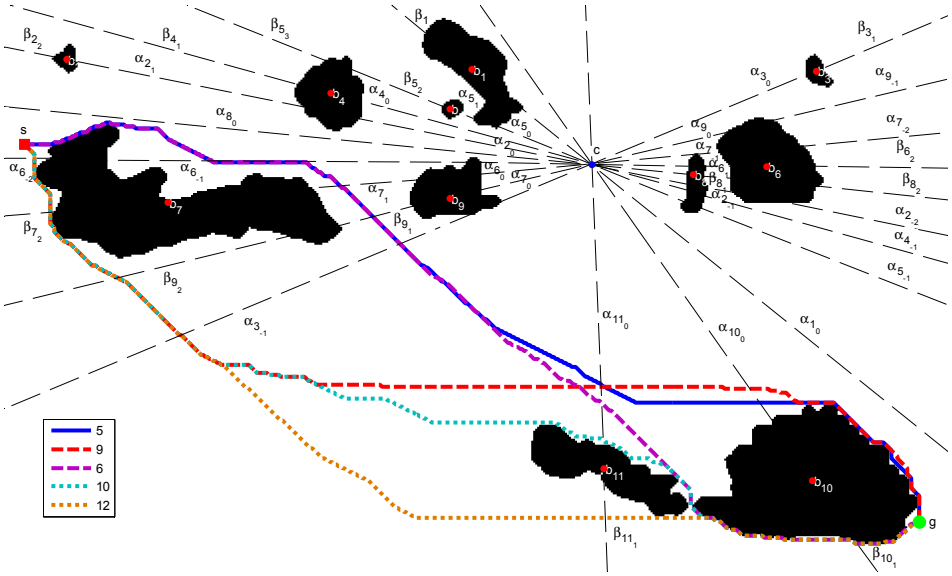


Figure 81: The paths of the five homotopy classes with the smaller lower bound. The class associated to the index can be found in Table 10.

homotopy classes according to their lower bound when a different goal point is selected (Table 11). Our method computed 75 homotopy classes in 0.292s. The cost for each homotopy class with its lower bound and computation time is shown in Figure 84. The paths for all the homotopy classes were computed in 226.4s. When operating under realtime constraints, it is possible to stop before computing the class 7 path. In such cases, the process would take 3.581s.

Idx	Homotopy class
5	$\alpha_{6_{-1}}\alpha_{7_1}\beta_{9_1}\alpha_{3_{-1}}\alpha_{11_0}\alpha_{10_0}$
9	$\alpha_{6_{-2}}\beta_{7_2}\beta_{9_2}\alpha_{3_{-1}}\alpha_{11_0}\alpha_{10_0}$
6	$\alpha_{6_{-1}}\alpha_{7_1}\beta_{9_1}\alpha_{3_{-1}}\alpha_{11_0}\beta_{10_1}$
10	$\alpha_{6_{-2}}\beta_{7_2}\beta_{9_2}\alpha_{3_{-1}}\alpha_{11_0}\beta_{10_1}$
12	$\alpha_{6_{-2}}\beta_{7_2}\beta_{9_2}\alpha_{3_{-1}}\beta_{11_1}\beta_{10_1}$

Table 10: The five homotopy classes with the smaller lower bound in Figure 81 scenario.

6.5 SUMMARY

This chapter has grouped the results obtained with the map building and path planning methods proposed in this thesis. The local map building procedure has been applied to a dataset gathered with the Ictineu AUV in a man-made environment. The experiment first applied the MSISpIC to improve the dead-reckoning trajectory and then generated an OGM with two different inverse sensor models.

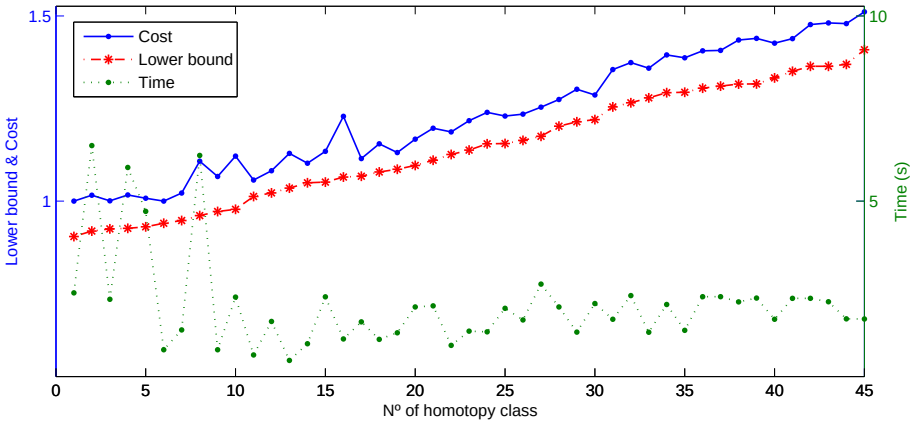


Figure 82: Normalized cost, normalized lower bound and computation time for paths generated with the HA* for the 45 homotopy classes in Figure 81.

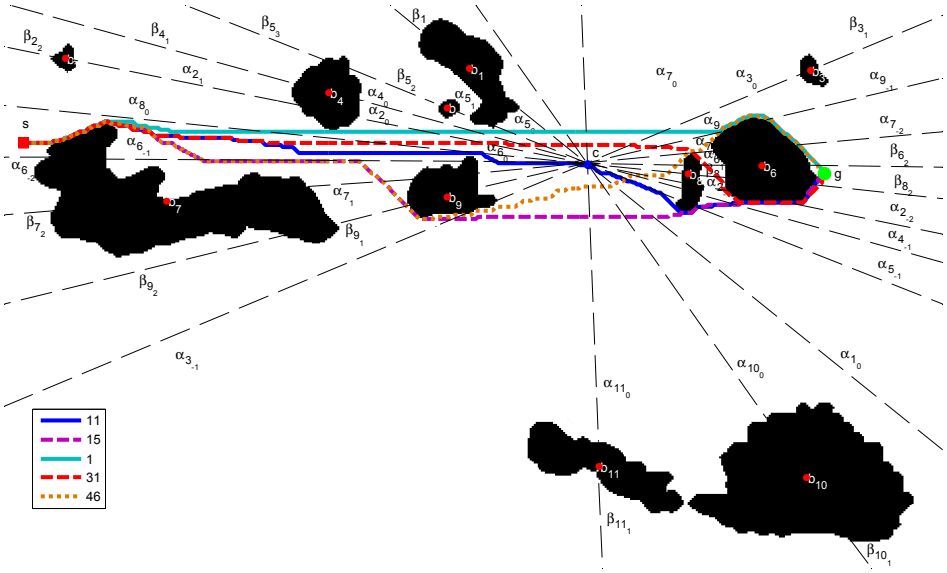


Figure 83: The paths of the five homotopy classes with the smaller lower bound. The class associated to the index can be found in Table 11.

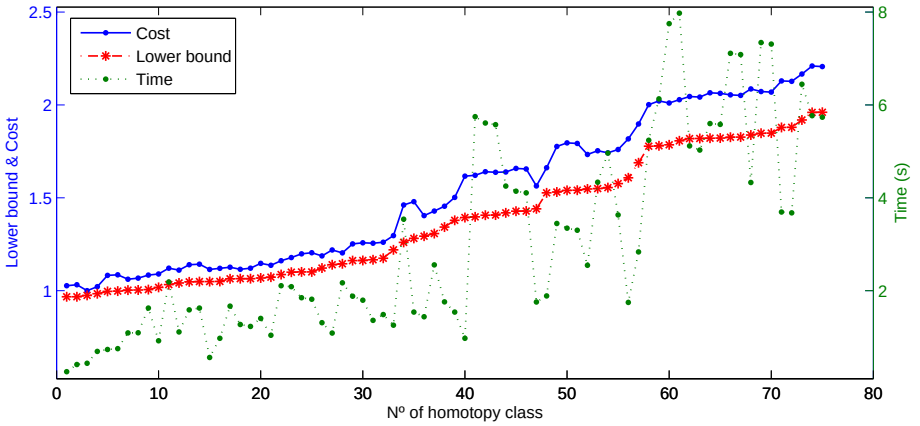


Figure 84: Normalized cost, normalized lower bound and computation time for paths generated with the HA* for the 75 homotopy classes in Figure 83.

Idx	Homotopy class
11	$\alpha_{6_0}\alpha_{7_0}\alpha_{9_0}\alpha_{3_0}\alpha_{11_0}\alpha_{10_0}\alpha_{1_0}\alpha_{5_{-1}}\alpha_{4_{-1}}\alpha_{2_{-2}}\beta_{8_2}$
15	$\alpha_{6_{-1}}\alpha_{7_1}\beta_{9_1}\alpha_{3_{-1}}\alpha_{11_0}\alpha_{10_0}\alpha_{1_0}\alpha_{5_{-1}}\alpha_{4_{-1}}\alpha_{2_{-2}}\beta_{8_2}$
1	$\alpha_{8_0}\alpha_{2_0}\alpha_{4_0}\alpha_{5_0}\alpha_{1_0}\alpha_{10_0}\alpha_{11_0}\alpha_{3_0}\alpha_{9_{-1}}\alpha_{7_{-2}}\beta_{6_2}$
31	$\alpha_{8_0}\alpha_{2_0}\alpha_{4_0}\alpha_{5_0}\alpha_{1_0}\alpha_{10_0}\alpha_{11_0}\alpha_{3_0}\alpha_{9_0}\alpha_{7_{-1}}\alpha_{6_1}\beta_{8_1}\alpha_{2_{-1}}\alpha_{2_{-2}}\beta_{8_2}$
46	$\alpha_{6_{-1}}\alpha_{7_1}\beta_{9_1}\alpha_{3_{-1}}\alpha_{11_0}\alpha_{10_0}\alpha_{1_0}\alpha_{5_0}\alpha_{4_0}\alpha_{2_0}\alpha_{8_0}\alpha_{6_0}\alpha_{7_0}\alpha_{9_0}\alpha_{9_{-1}}\alpha_{7_{-2}}\beta_{6_2}$

Table 11: The five homotopy classes with the smaller lower bound in Figure 83 scenario.

In the second experiment, our path planning approach has been applied to two different bitmap scenarios with irregular obstacles. The first one is a small cluttered environment which allows detailing the homotopy classes generation process. The path for each homotopy class has been computed with the HA*, the HRRT and the HBug algorithms. The second scenario represents a large environment that has shown up the differences between our path planners. Different comparisons have been made between the proposed path planners with their respective non-homotopic versions and with themselves. This experiment has been used to conclude that the HA* should be used when generating optimal solutions at the expense of higher computation time is required, whereas the HBug is suitable for applications in which the time to perform the path planning is highly constrained. The HRRT should be avoided since the HBug generates lower cost solutions much faster.

The map building approach and the path planning method have been tested together in a real experiment using the Sparus AUV in a scenario built in the water tank of the Underwater Robotics Lab. of the University of Girona. Finally, the path planning method has been tested in the context of the TRIDENT European project, where the HA* has been applied in a bathymetry gathered in the Formigues Islands.

7

CONCLUSIONS

This chapter concludes the work presented throughout this dissertation. It first summarizes the thesis by reviewing the contents described in each chapter. Then, it points out the research contributions extracted from the proposals and the experiments. In addition, all aspects which have not been accomplished as well as some interesting future research issues are commented on in the future work section. Then, the research framework in which this thesis was achieved is described. Finally, publications related to this work are listed.

7.1 SUMMARY

This thesis addresses the path planning problem for Autonomous Underwater Vehicles (AUVs). The method proposes the utilization of homotopy classes to provide topological information on how paths avoid obstacles. Therefore, looking for a path within a homotopy class constrains the search into a specific area of the Configuration Space (C-space), speeding up the generation of the path. In addition to the path planning method, a local map building approach has also been proposed. Essentially, it first improves the dead-reckoning navigation of the AUV through a sonar scan matching technique to generate a more feasible Occupancy Grid Map (OGM) where the path planning is carried out. The local map building results of this research project have been presented in a dataset gathered with the Ictineu AUV, one of the experimental platforms available to the Computer Vision and Robotics (VICOROB) group at the University of Girona. The results of our path planning approach have been obtained in synthetic scenarios designed to stress our method, and with the Sparus AUV in a controlled unknown scenario built in the Underwater Robotics Lab. of the University of Girona.

After a brief description of the path planning problem, Chapter 2 has given an overview of the main approaches applied to robotics. The chapter starts with the graph-based search algorithms, which compute the global shortest path in the C-space by means of an exhaustive exploration of the search space, normally improved through a heuristic estimator. On the other hand, most probabilistic sample-based path planners perform the exploration by incrementally growing a tree until the goal is reached. Since the C-space is not explored in detail, the generation of the solution occurs very quickly at the expense of its quality. Anytime path planners obtain a highly suboptimal first solution very quickly which is improved through successive iterations while the time does not expire. Although, the solution is intended to be improved at each iteration, the generation of a new/better path is not ensured. In the survey, Bug-based approaches have also been considered for path planning purposes. Following the same strategies described for motion planning, these

algorithms can generate suboptimal solutions very quickly. An insight into topological path planning has also been provided, paying special attention to those solutions that deal with homotopy classes. Representative path planning approaches for AUVs have been described at the end of the chapter.

Chapter 3 has presented the path planning proposal of this research work, which uses homotopy classes to guide the path planners topologically. The method starts by generating the homotopy classes that connect the start point with the end point in a workspace with obstacles. Then, a lower bound criterion estimates their cost. Hence, those classes which most-probably contain the lower cost solutions are known without computing any path at all. Finally, a path planner uses the topological information of the homotopy classes to generate some good solutions very quickly. Three path planners have been proposed to generate paths for the homotopy classes obtained: the Homotopic A* (HA*), a graph-search based algorithm that computes the shortest homotopic path according to an input homotopy class; a probabilistic approach based on the Rapidly-exploring Random Tree (RRT) called Homotopic RRT (HRRT); and the Homotopic Bug (HBug), a Bug-based algorithm that combines the lower bound path with the surrounding of obstacle's boundaries.

Chapter 4 has described our local map building proposal. The chapter is divided into two main parts. The first one focuses on improving the dead-reckoning navigation of AUVs through scan matching. After providing a definition of the problem, an extension of the Probabilistic Iterative Correspondance (pIC) algorithm, called MSISpIC, has been introduced. This algorithm deals with uncertainties and motion induced distortions introduced by sonar sensors with a mechanical rotating head emitter/receiver that AUVs are usually equipped with. The improved navigation is used to build an OGM in which the path planning is performed. For the construction of the map, two inverse sensor models have been shown: the standard range sonar sensor and an imaging sonar sensor, which takes into account the probability of occupancy at each discretized part of the beam cone according to the amount of acoustic energy reflected.

Chapter 5 has reported the main features of the experimental platforms used in this dissertation, the Ictineu AUV and the Sparus AUV, including the design principles, the actuators and the on board sensors. An insight into the sensors used for map building purposes and the control architecture used in the robots have also been provided.

The results, presented in Chapter 6, have been organized in several sections according to whether they belong to map building, path planning or both. The first experiment has tested the map building approach in a dataset gathered with the Ictineu AUV in a man-made environment. The experiment improved the dead-reckoning trajectory with the MSISpIC and then computed its OGM with the two inverse sensor models proposed. The second experiment groups a set of several tests using our path planning method in different simulated scenarios. After the computation of the homotopy classes, their correspondent paths have been generated in the C-space by means of the HA*, the HRRT and the HBug algorithms. In these experiments, the proposed path planners have

been compared with their respective non-homotopic versions, and a second comparison has put the results obtained with HRRT and HBug in front of the HA* solutions, since it computes the shortest path for each homotopy class. The third experiment has tested the map building and the path planning methods together with the Sparus AUV in a scenario built in the water tank of the Underwater Robotics Lab. of the University of Girona. The last set of experiments have shown the applicability of our path planning method on a bathymetric map in the context of the TRIDENT European project.

7.2 CONTRIBUTIONS

The thesis work has accomplished the proposed goal of developing a path planning method for AUVs that takes into account topological constraints addressed through homotopy classes in sonar-based maps improved with scan matching. In the development of this goal, various research contributions were achieved. These contributions are listed below.

HOMOTOPY CLASSES FOR PATH PLANNING. Some of the most relevant path planning approaches have been studied, specially those which take into account homotopy classes. With these methods, it is possible to generate paths that avoid obstacles in different manners, which is interesting for surveilling purposes and to avoid certain undesirable zones. In this research project we have extended an existing method in order to generate only those homotopy classes which can then be followed in the C-space. Furthermore, we have proposed a lower bound criterion to set up a preference order when choosing a homotopy class to compute its path.

PATH PLANNING FOR AUVS. In this research work we have developed three path planners that generate solutions according to an input homotopy class. Each one is based on a different approach: the HA* generates the optimal path for a given input homotopy class, which makes the algorithm suitable to be used as a ground truth; the HRRT computes the solution through probabilistic random sampling, which speeds up the exploration process of the search space. Finally, the HBug efficiently looks for a path by following obstacle boundaries according to an input homotopy class.

With the method proposed, which assumes that homotopy classes are sorted according to their lower bound, it is possible to stop the path search when running under time constraints when the lower bound of the next homotopy class is equal to or higher than the lowest cost of the previous solutions. At this point, it is ensured that the global optimal solution has been found.

SCAN MATCHING FOR AUVS. In this research project it has been proposed the MSISpIC, a scan matching algorithm that works with mechanically scanned imaging/profiler sonars. These sensors are commonly available in AUVs nowadays. The MSISpIC deals with the motion induced distor-

tion introduced during the scan acquisition process by estimating the relative displacement with an Extended Kalman Filter (EKF) while the scan is being gathered.

LOCAL MAP BUILDING. The improved localization obtained from the scan matching algorithm is used to build an OGM to perform the path planning. The accurateness of the map depends on the localization estimation itself and on the accuracy of the sensor readings. Therefore, apart from the commonly applied inverse range sensor model, an inverse imaging sonar sensor model has also been proposed, which takes into account the amount of acoustic energy reflected at each point in the perception area.

TESTS AND RESULTS. The applicability of the map building approach has been tested on a dataset gathered with the Ictineu AUV in a man-made marina environment. The path planning proposal has been extensively tested in synthetic scenarios. As a final result, it has also been tested with the Sparus AUV in a controlled unknown environment, where the local map building method provided the map with which the paths for each homotopy class were computed using the path planners proposed.

7.3 FUTURE WORK

During the development of this thesis, new problems and topics of interest for future research have arisen. The following points are considered the most logical lines to continue this research.

TRAVERSE THE COMPUTED PATHS WITH AUVS. The next logical step to be performed in this research work is to turn the computed path into a feasible trajectory to be followed by the vehicles. For this reason, it is necessary that path planners take into account the kinodynamic constraints of the vehicles. For safety purposes, it would also be required to set up a simple obstacle avoidance to adapt the trajectory or stop the vehicle in case of imminent collision.

EXTEND THE LOWER BOUND ESTIMATOR. In order to improve the accuracy of the whole path planning method, the lower bound estimator should take into account more restriction criteria than the shortest distance such as dangerous zones.

HOMOTOPY CLASSES IN CHANGING ENVIRONMENTS. The method proposed in this thesis starts the generation of the homotopy classes according to a fixed map. Although the method is robust to small changes in the environment such as updated size or shape of an obstacle already detected, when obstacles are added/removed according to the sensorial information, the topological environment changes. Consequently, the homotopy classes have to be recomputed from scratch. Therefore, it is important to study how classes can be adapted to changing environments and how they evolve when new obstacles are added.

EXTENSION OF THE HOMOTOPY CLASSES. The natural extension of the work performed in this research project is to compute homotopy classes in 3D environments. However, while the homotopy class of a set of trajectories a is a clear concept well defined in 2D, a deeper study is required to clarify whether they are detailed enough to constrain the path search in 3D. Notice that with the current definition of a homotopy class any obstacle placed on the seafloor could be avoided from the left, the right or from above without changing its class.

7.4 RESEARCH FRAMEWORK

The results and conclusions presented in this thesis have been possible after the realization of countless tests and experiments, which were the fruit of numerous efforts made during the development of the different research robots and the necessary software and equipment. All the work done during the evolution of this thesis is summarized here with references to the most relevant research publications made by the author. A complete list of publications can be consulted in the next section.

At the beginning of this thesis, in 2006, the Ictineu AUV was built to face the first edition of the Student Autonomous Underwater Challenge-Europe (SAUC-E). To confront this competition, not only a vehicle had to be built, but the Object Oriented Control Architecture for Autonomy (O2CA2), developed for older vehicle platforms, also had to be remodeled and a Mission Control System (MCS) developed. The experience acquired with the Ictineu AUV was reflected in [CCIA'06, ICRA'07]. In parallel with this competition, the Underwater Robotics Lab. was starting a series of experiments to automatically survey the wall of a dam. For these experiments, it was also necessary to have a functional MCS [IROS'06]. Relative localization techniques with Mechanical Scanned Imaging Sonars (MSISs) were also studied [ICRA'09].

In order to generate paths for the AUV, it was necessary to build a map with the data gathered by the vehicle's onboard sensors. Despite the fact that a Simultaneous Localization And Mapping (SLAM) proposal was being developed in our laboratory, this technique was intended to build a map as close to reality as possible but it was not suitable for online computation. Therefore, the solution to build a map fast enough for online generation while navigating was addressed through sonar scan matching techniques. A first approach to the algorithm which only corrected the displacement but not the rotation was presented in [WAF'08, JoPhA'09]. The final version was presented in [IROS'09]. Both papers presented the results obtained with a dataset gathered with the Ictineu AUV in a man-made marina environment. Once the dead-reckoning navigation was improved, in [MCMC'09] we generated an OGM, in which the path planning could be performed. In order to generate a map that takes into account the data gathered with an MSIS, an imaging sonar inverse sensor model was proposed.

The path planning was addressed through the utilization of homotopy classes. The extension of an existing method to efficiently generate only those

classes that can be reached in any 2D workspace was presented with the HRRT in [ICRA'11]. This paper also contains the first results of the whole path planning method on the Sparus AUV. In [ICAPS'11], the lower bound estimator to establish a preference order to compute the paths for the homotopy classes and a set of improved restriction criteria during the homotopy classes generation, which would reduce the potential class candidates was presented. The HA* was developed as a simple graph-search approach, based on the A*, adapted to compute paths according to specific homotopy classes. Since the algorithm computes the optimal path for any homotopy class, it can be used as a ground truth. The results with this algorithm were presented in [IWC'11]. At the same time, the applicability of this method was tested on bathymetric maps in the context of the TRIDENT European Project [OCEANS'11b]. In order to take the maximum advantage of the lower bound computation, we developed the HBug. This algorithm tries to follow the lower bound path, and when an intersection with an obstacles in the workspace occurs, its boundary is surrounded in the direction that accomplishes the input homotopy class. The HBug with the path planning method working together with the local map building approach has appeared in [NGCUV'12], while exhaustive HBug tests in synthetic scenarios and a comparison with the other homotopic path planners will be presented in [ICINCO'12].

7.5 RELATED PUBLICATIONS

Sonar Scan Matching and Map building

WAF'08 **E. Hernández**, P. Ridao, D. Ribas and J. Batlle. Probabilistic Scan Matching with a Mechanical Scan Imaging Sonar. IX Workshop on Physical Agents (WAF). Vigo, Spain. 2008.

JOPHA'09 **E. Hernández**, P. Ridao, D. Ribas and J. Batlle. MSISpIC: A Probabilistic Scan Matching Algorithm Using a Mechanical Scanned Imaging Sonar. Journal of Physical Agents, 2009, Vol.3, N.1, pages 3–11. ISSN: 1888-0258.

IROS'09 **E. Hernández**, P. Ridao, D. Ribas and A. Mallios. Probabilistic sonar scan matching for an AUV. In Proceedings of the Intelligent IEEE/RSJ International Conference on Robots and Systems (IROS), pages 255–260. St. Louis, MO, USA. 2009.

OCEANS'09 A. Mallios, P. Ridao, **E. Hernández**, D. Ribas, F. Maurelli and Y. Petillot. Pose-Based SLAM with Probabilistic Scan Matching Algorithm using a Mechanical Scanned Imaging Sonar. Oceans IEEE-EUROPE. Bremen, Germany. 2009.

MCMC'10 **E. Hernández**, P. Ridao, A. Mallios and M. Carreras. Occupancy Grid Mapping in an Underwater Structured Environment. In Proceedings of the 8th IFAC International Conference on Manoeuvring and Control of Marine Craft (MCMC). Sao Paulo, Brazil. 2009.

OCEANS'10 A. Mallios, P. Ridao, D. Ribas and **E. Hernández**. Probabilistic Sonar Scan Matching SLAM for Underwater Environment. OCEANS 2010 IEEE - Sydney, pages 1–8. Sidney, Australia. 2010.

OCEANS'11A A. Mallios, P. Ridao, M. Carreras and **E. Hernández**. Navigating and Mapping with the Sparus AUV in a Natural and Unstructured Underwater Environment. OCEANS 2011, pages 1–7. Kona, Hawai. 2011.

AR'12 A. Mallios, P. Ridao, D. Ribas and **E. Hernández**. Scan Matching SLAM in Underwater Environment. Autonomous Robots Journal. 2012. (Submitted).

Path Planning Methods with Homotopy Constrains

OCEANS'11B **E. Hernández**, M. Carreras, E. Galceran and P. Ridao. Path planning with homotopy class constraints on bathymetric maps. OCEANS, 2011 IEEE - Spain, pages 1–6. Santander, Spain. 2011.

ICRA'11 **E. Hernández**, M. Carreras, J. Antich, P. Ridao and A. Ortiz. A Topologically Guided Path Planner for an AUV Using Homotopy Classes. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pages 2337–2343. Shanghai, China. 2011.

IWC'11 **E. Hernández**, M. Carreras, J. Antich, P. Ridao and A. Ortiz. A Search-based Path Planning Algorithm with Topological Constraints. Application to an AUV. In Proceedings of the 18th IFAC World Congress. Milan, Italy. 2011.

ICAPS'11 **E. Hernández**, M. Carreras and P. Ridao. A Path Planning Algorithm for an AUV Guided with Homotopy Classes. In Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS). Freiburg, Germany. 2011.

NGCUV'12 **E. Hernández**, M. Carreras, P. Ridao and A. Mallios. Homotopic Path Planning for an AUV on Maps Improved with Scan Matching. In Proceedings of IFAC Workshop on Navigation, Guidance and Control of Underwater Vehicles (NGCUV). Porto, Portugal. 2012.

ICINCO'12 **E. Hernández**, M. Carreras and P. Ridao. A Bug-based Path Planner Guided with Homotopy Classes. In Proceedings of the 9th International Conference on Informatics and Control, Automation and Robotics (ICINCO). Rome, Italy. 2012. To appear.

Other Works Related with Underwater Robotics

CCIA'06 **E. Hernández**, P. Ridao, M. Carreras, D. Ribas, N. Palomeras, A. El-fakdi, F. Chung, T. Almohaya, X. Ribas, G. García, J. Massich and N. Hurtós. Ictineu AUV, un Robot per a Competir. 9th Congrés Català d'Intel.ligència Artificial (CCIA). Perpignan, France. 2006.

IROS'06 N. Palomeras, M. Carreras, P. Ridao and **E. Hernández**. Mission control system for dam inspection with an AUV. In Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 2551-2556. Beijing, China. 2006.

ICRA'07 D. Ribas, N. Palomeras, P. Ridao, M. Carreras and **E. Hernández**. ICTINEU AUV Wins the first SAUC-E competition. In Proceedings of IEEE International Conference on Robotics and Automation (ICRA), pages 151–156. Rome, Italy. 2007.

ECAI'08 A. El-Fakdi, M. Carreras and **E. Hernández**. Gradient-based Reinforcement Learning for Autonomous Underwater Cable Tracking. In Proceedings of the 18th European Conference on Artificial Intelligence (ECAI). Patras, Greece. 2008.

ICRA'09 W. Kazmi, P. Ridao, D. Ribas and **E. Hernández**. Dam wall detection and tracking using a Mechanically Scanned Imaging Sonar. In Proceedings of IEEE International Conference on Robotics and Automation (ICRA), pages 3595–3600. Kobe, Japan. 2009.

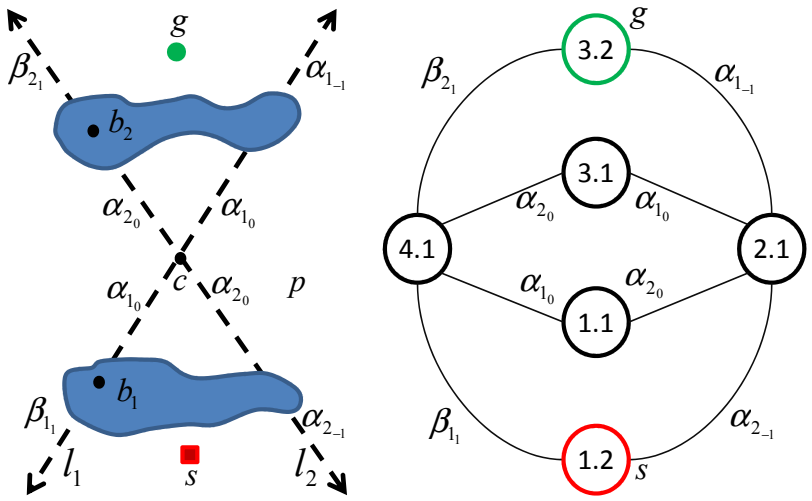
ICRA'11 P. Ridao, D. Ribas, **E. Hernández** and A. Rusu. USBL/DVL navigation through delayed position fixes. In Proceedings of IEEE International Conference on Robotics and Automation (ICRA), pages 2344–2349. Shanghai, China. 2011.

A

AN EXAMPLE OF HOMOTOPY CLASSES GENERATION

This appendix shows a full example of the generation of the homotopy classes with the method developed in this dissertation in a simple environment with two obstacles. The scenario is depicted in Figure 85.a with its reference frame. Figure 85.b shows its topological graph.

Table 12 shows the full execution of the modified Breadth-First Search (BFS) algorithm. When a candidate homotopy class accomplishes one of the restriction criteria, it is described and the class is discarded from being used as a root for future homotopy classes. In this example, the execution of the BFS is stopped when, due to the restriction criteria, there are no more class candidates to be generated. Those homotopy class candidates that pass all the restriction criteria and finish at node 3.2 in the graph are marked with an asterisk. Table 13 summarizes the homotopy classes obtained with their index of generation according to the BFS.



a) Reference frame b) Topological graph

Figure 85: The reference frame of a simple scenario with two obstacles with its correspondent topological graph.

Homotopy class	Comment	Homotopy class	Comment
β_{1_1}		$\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}\beta_{2_1}$	
$\alpha_{2_{-1}}$			
$\beta_{1_1}\alpha_{1_0}$		$\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{2_{-1}}\beta_{1_1}$	Simple wrap
$\beta_{1_1}\alpha_{2_0}$			
$\beta_{1_1}\beta_{2_1}$	*	$\beta_{1_1}\beta_{2_1}\alpha_{1_{-1}}\alpha_{1_0}\alpha_{2_0}$	
$\alpha_{2_{-1}}\alpha_{1_0}$		$\beta_{1_1}\beta_{2_1}\alpha_{1_{-1}}\alpha_{2_0}\alpha_{1_0}$	Duplicated
$\alpha_{2_{-1}}\alpha_{1_{-1}}$	*		
$\alpha_{2_{-1}}\alpha_{2_0}$		$\beta_{1_1}\beta_{2_1}\alpha_{1_{-1}}\alpha_{2_{-1}}\beta_{1_1}$	Simple wrap
$\beta_{1_1}\alpha_{1_0}\alpha_{2_0}$		$\alpha_{2_{-1}}\alpha_{1_0}\alpha_{2_0}\beta_{1_1}\alpha_{2_{-1}}$	Simple wrap
$\beta_{1_1}\alpha_{2_0}\alpha_{1_0}$	Duplicated	$\alpha_{2_{-1}}\alpha_{1_0}\alpha_{2_0}\beta_{2_1}\alpha_{1_{-1}}$	
$\beta_{1_1}\beta_{2_1}\alpha_{1_{-1}}$		$\alpha_{2_{-1}}\alpha_{1_{-1}}\beta_{2_1}\beta_{1_1}\alpha_{2_{-1}}$	Simple wrap
$\alpha_{2_{-1}}\alpha_{1_0}\alpha_{2_0}$		$\alpha_{2_{-1}}\alpha_{1_{-1}}\beta_{2_1}\alpha_{2_0}\alpha_{1_0}$	Duplicated
$\alpha_{2_{-1}}\alpha_{1_{-1}}\beta_{2_1}$		$\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}\beta_{2_1}\alpha_{1_0}$	Simple wrap
$\alpha_{2_{-1}}\alpha_{2_0}\alpha_{1_0}$	Duplicated	$\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}\beta_{2_1}\beta_{1_1}$	Simple wrap
		$\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}\beta_{2_1}\alpha_{2_0}$	Simple wrap
$\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_0}$	Duplicated		
$\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$	*	$\beta_{1_1}\beta_{2_1}\alpha_{1_{-1}}\alpha_{1_0}\alpha_{2_0}\alpha_{1_0}$	Duplicated
$\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{2_{-1}}$		$\beta_{1_1}\beta_{2_1}\alpha_{1_{-1}}\alpha_{1_0}\alpha_{2_0}\beta_{1_1}$	Simple wrap
		$\beta_{1_1}\beta_{2_1}\alpha_{1_{-1}}\alpha_{1_0}\alpha_{2_0}\beta_{2_1}$	Simple wrap
$\beta_{1_1}\beta_{2_1}\alpha_{1_{-1}}\alpha_{1_0}$			
$\beta_{1_1}\beta_{2_1}\alpha_{1_{-1}}\alpha_{2_0}$		$\alpha_{2_{-1}}\alpha_{1_0}\alpha_{2_0}\beta_{2_1}\alpha_{1_{-1}}\alpha_{1_0}$	Simple wrap
$\beta_{1_1}\beta_{2_1}\alpha_{1_{-1}}\alpha_{2_{-1}}$			
		$\alpha_{2_{-1}}\alpha_{1_0}\alpha_{2_0}\beta_{2_1}\alpha_{1_{-1}}\alpha_{2_0}$	Simple wrap
$\alpha_{2_{-1}}\alpha_{1_0}\alpha_{2_0}\alpha_{1_0}$	Duplicated		
$\alpha_{2_{-1}}\alpha_{1_0}\alpha_{2_0}\beta_{1_1}$		$\alpha_{2_{-1}}\alpha_{1_0}\alpha_{2_0}\beta_{2_1}\alpha_{1_{-1}}\alpha_{2_{-1}}$	Simple wrap
$\alpha_{2_{-1}}\alpha_{1_0}\alpha_{2_0}\beta_{2_1}$	*		
		$\alpha_{2_{-1}}\alpha_{1_{-1}}\beta_{2_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_0}$	Duplicated
$\alpha_{2_{-1}}\alpha_{1_{-1}}\beta_{2_1}\alpha_{1_0}$		$\alpha_{2_{-1}}\alpha_{1_{-1}}\beta_{2_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$	Simple wrap
$\alpha_{2_{-1}}\alpha_{1_{-1}}\beta_{2_1}\beta_{1_1}$		$\alpha_{2_{-1}}\alpha_{1_{-1}}\beta_{2_1}\alpha_{1_0}\alpha_{2_0}\alpha_{2_{-1}}$	Simple wrap
$\alpha_{2_{-1}}\alpha_{1_{-1}}\beta_{2_1}\alpha_{2_0}$			

Table 12: Modified BFS execution.

Idx	Homotopy class
1	$\beta_{1_1}\beta_{2_1}$
2	$\alpha_{2_{-1}}\alpha_{1_{-1}}$
3	$\beta_{1_1}\alpha_{1_0}\alpha_{2_0}\alpha_{1_{-1}}$
4	$\alpha_{2_{-1}}\alpha_{1_0}\alpha_{2_0}\beta_{2_1}$

Table 13: Homotopy classes obtained for the example.

B

TRANSFORMATIONS IN 2D

In (Smith et al., 1990) two operations representing the most frequently encountered spatial relationships in stochastic mapping applications were presented. One of these operations is the composition transformation, represented by the operators $\oplus$:

$$\mathbf{x}_C^A = \mathbf{x}_B^A \oplus \mathbf{x}_C^B$$

B.1 COMPOSITION

Given two spatial transformations (reference B relative to reference A and reference C relative to reference B):

$$\mathbf{x}_B^A = \begin{bmatrix} x_1 \\ y_1 \\ \phi_1 \end{bmatrix} \qquad \mathbf{x}_C^B = \begin{bmatrix} x_2 \\ y_2 \\ \phi_2 \end{bmatrix}$$

The location of C relative to A can be described by the composition transformation:

$$\mathbf{x}_C^A = \mathbf{x}_B^A \oplus \mathbf{x}_C^B = \begin{bmatrix} x_1 + x_2 \cos\phi_1 - y_2 \sin\phi_1 \\ y_1 + x_2 \sin\phi_1 + y_2 \cos\phi_1 \\ \phi_1 + \phi_2 \end{bmatrix}$$

The estimated location of C relative to A can be described as the composition transformation:

$$\mathbf{x}_C^A = \mathbf{x}_B^A \oplus \mathbf{x}_C^B$$

B.2 POINT FEATURES

Given the location of a point feature P relative to reference B:

$$\mathbf{x}_P^B = \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}$$

In a similar manner, as mentioned before, the location of P relative to reference A can be described by the composition transformation:

$$\mathbf{x}_P^A = \mathbf{x}_B^A \oplus \mathbf{x}_P^B = \begin{bmatrix} x_1 + x_2 \cos\phi_1 - y_2 \sin\phi_1 \\ y_1 + x_2 \sin\phi_1 + y_2 \cos\phi_1 \end{bmatrix}$$

The estimated location of P relative to A can be described as the composition transformation:

$$\mathbf{x}_P^A = \mathbf{x}_B^A \oplus \mathbf{x}_P^B$$

BIBLIOGRAPHY

Abbasi-Yadkori, Y., Modayil, J., and Szepesvari, C. (2010). Extending rapidly-exploring random trees for asymptotically optimal anytime motion planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 127 –132.

Acar, E., Choset, H., and Lee, J. Y. (2006). Sensor-based coverage with extended range detectors. *IEEE Transactions on Robotics*, 22(1):189 – 198.

Aine, S., Chakrabarti, P. P., and Kumar, R. (2007). AWA* - a window constrained anytime heuristic search algorithm. In *Proceedings of the 20th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 2250–2255.

Alcázar, V., Veloso, M., and Borrajo, D. (2011). Adapting an rrt for automated planning. In Borrajo, D., López, C. L., and Likhachev, M., editors, *Proceedings of the Fourth International Symposium on Combinatorial Search (SoCS)*, pages 2–9, Cardona, Spain. AAAI Press.

Alvarez, A., Caiti, A., and Onken, R. (2004). Evolutionary path planning for autonomous underwater vehicles in a variable ocean. *IEEE Journal of Oceanic Engineering*, 29(2):418–429.

Amat, J., Batlle, J., Casals, A., and Forest, J. (1996). Garbi: a low cost ROV, constrains and solutions. In *6e Seminaire IARP en robotique sous-marine*, pages 1–22, Toulon-La Seyne, France.

Antich, J. and Ortiz, A. (2009). Bug2+: Details and formal proofs. Technical Report Report A-1-2009, University of the Balearic Islands.

Antich, J., Ortiz, A., and Minguez, J. (2009). A bug-inspired algorithm for efficient anytime path planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5407 –5413.

Arkin, R. and Balch, T. (1997). AuRA: Principles and practice in review. *Journal of Experimental and Theoretical Artificial Intelligence*, 9:175–189.

Asama, H., Ozaki, K., Itakura, H., Matsumoto, A., Ishida, Y., and Endo, I. (1991). Collision avoidance among multiple mobile robots based on rules and communication. In *Proceedings of the IEEE/RSJ Intelligent Robots and Systems (IROS). International Workshop on Intelligence for Mechanical Systems*, volume 3, pages 1215–1220.

Aurenhammer, F. (1991). Voronoi diagrams – a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405.

Bailey, T. and Durrant-Whyte, H. F. (2006). Simultaneous localization and mapping (SLAM): Part II, state of the art. *IEEE Robotics and Automation Magazine*, 13(3):108–117.

Banerjee, B. and Chandrasekaran, B. (2006). A framework for planning multiple paths in free space. In *Proceedings of 25th Army Science Conference*, Orlando, FL.

Bar-Shalom, Y. and Fortman, T. (1998). *Tracking and Data Association*. Academic Press.

Barraquand, J., Langlois, B., and Latombe, J.-C. (1992). Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man and Cybernetics*, 22(2):224 –241.

Batlle, J., Ridao, P., Garcia, R., Carreras, M., Cufi, X., El-Fakdi, A., Ribas, D., Nicosevici, T., and Batlle, E. (2004). *URIS: Underwater Robotic Intelligent System*, chapter 11, pages 177 – 203. Instituto de Automatica Industrial, Consejo Superior de Investigaciones Cientificas, 1st edition.

Belghith, K., Kabanza, F., Hartman, L., and Nkambou, R. (2006). Anytime dynamic path-planning with flexible probabilistic roadmaps. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2372–2377, Orlando, Florida, USA.

Besl, P. J. and McKay, N. D. (1992). A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):239–256.

Bespamyatnikh, S. (2003). Computing homotopic shortest paths in the plane. *Journal of Algorithms*, 49:284–303.

Bhattacharya, P. and Gavrilova, M. (2007). Voronoi diagram in optimal path planning. In *4th International Symposium on Voronoi Diagrams in Science and Engineering (ISVD)*, pages 38 –47.

Bhattacharya, S., Kumar, V., and Likhachev., M. (2010). Search-based path planning with homotopy class constraints. In *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, volume 2, pages 1230–1237, Atlanta, Georgia, USA.

Binney, J., Krause, A., and Sukhatme, G. (2010). Informative path planning for an autonomous underwater vehicle. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4791 –4796.

Borenstein, J. and Koren, Y. (1991). The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278–288.

Borrajo, D. and Veloso, M. (2012). Probabilistically reusing plans in deterministic planning. In *Proceedings of ICAPS'12 workshop on Heuristics and Search for Domain-Independent Planning*. AAAI Press.

Bourgault, F., Makarenko, A., Williams, S., Grocholsky, B., and Durrant-Whyte, H. (2002). Information based adaptive robotic exploration. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 540–545.

Branicky, M., LaValle, S., Olson, K., and Yang, L. (2001). Quasi-randomized path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 1481–1487, Seoul, Korea.

Brooks, R. A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(3):14–23.

Bülow, H. and Birk, A. (2011). Spectral registration of noisy sonar data for underwater 3d mapping. *Autonomous Robots*, 30:307–331. 10.1007/s10514-011-9221-8.

Bülow, H., Pfingsthorn, M., and Birk, A. (2010). Using robust spectral registration for scan matching of sonar range data. In *7th Symposium on Intelligent Autonomous Vehicles (IAV)*, volume 7, Lecce, Italy. IFAC.

Burguera, A., Gonzalez, Y., and Oliver, G. (2007). Probabilistic sonar scan matching for robust localization. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3154–3160, Roma, Italy.

Burguera, A., Oliver, G., and González, Y. (2010). Scan-based SLAM with trajectory correction in underwater environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2546 –2551, Taipei, Taiwan.

Cabello, S., Liu, Y., Manler, A., and Snoeyink, J. (2002). Testing homotopy for paths in the plane. In *Proceedings of the Symposium on Computational Geometry (SoCG)*, pages 160–169, Barcelona, Spain.

Carreras, M., Batlle, J., and Ridao, P. (2001). Hybrid coordination of reinforcement learning-based behaviors for AUV control. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 1410 – 1415, Wailea, Hawaii, USA.

Carreras, M., Ridao, P., and El-Fakdi, A. (2003). Semi-online neural-qlearning for real-time robot learning. In *IEEE/RSJ Conference on Intelligent Robots and Systems (IROS)*, pages 662 – 667.

Carroll, K., McClaran, S., Nelson, E., Barnett, D., Friesen, D., and William, G. (1992). AUV path planning: an A* approach to path planning with consideration of variable vehicle speeds and multiple, overlapping, time-dependent exclusion zones. In *Proceedings of the 1992 Symposium on Autonomous Underwater Vehicle Technology (AUV)*, pages 79 –84.

Castellani, U., Fusiello, A., Murino, V., Papaleo, L., Puppo, E., Repetto, S., and Pittore, M. (2004). Efficient on-line mosaicing from 3D acoustical images. *OCEANS '04. MTTS/IEEE TECHNO-OCEAN '04*, 2:670–677.

Chang, F., jen Chen, C., and Lu, C.-J. (2004). A linear-time component-labeling algorithm using contour tracing technique. *Computer Vision and Image Understanding*, 93:206–220.

Chazelle, B. (1982). A theorem on polygon cutting with applications. In *23rd Annual Symposium on Foundations of Computer Science (SFCS)*, pages 339 –349.

Cheng, S.-W., Jin, J., Vigneron, A., and Wang, Y. (2010). Approximate shortest homotopic paths in weighted regions. In Cheong, O., Chwa, K.-Y., and Park, K., editors, *Algorithms and Computation*, volume 6507 of *Lecture Notes in Computer Science*, pages 109–120. Springer Berlin / Heidelberg.

Chien, Y.-P. and Xue, Q. (1992). Path planning for two planar robots moving in unknown environment. *IEEE Transactions on Systems, Man and Cybernetics*, 22(2):307 –317.

Choset, H., Lynch, K. M., Hutchinson, S., Kantor, G. A., Burgard, W., Kavraki, L. E., and Thrun, S. (2005). *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, Cambridge, MA.

Cuerington, A. (1991). The shortest path problem in the plane with obstacles: Bounds on path lenghts and shortest paths within homotopy classes. Master's thesis, Naval Postgraduate School, Monterey, California.

Davis, D. (1996). *Precision maneuvering and control of the Phoenix Autonomous Underwater Vehicle for entering a recovery tube*. PhD thesis, Naval Postgraduate School, Monterey, California.

Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271. 10.1007/BF01386390.

Dolgov, D., Thrun, S., Montemerlo, M., and Diebel, J. (2010). Path planning for autonomous vehicles in unknown semi-structured environments. *International Journal of Robotics Research*, 29(5):485–501.

Dudek, G., Jenkin, M., Milios, E., and Wilkes, D. (1991). Robotic exploration as graph construction. *IEEE Transactions on Robotics and Automation*, 7(6):859–865.

Durrant-Whyte, H. F. and Bailey, T. (2006). Simultaneous localization and mapping (SLAM): Part I, the essential algorithms. *IEEE Robotics and Automation Magazine*, 13(2):99–108.

Efrat, A., Kobourov, S., and Lubiw, A. (2002). Computing homotopic shortest paths efficiently. In Möhring, R. and Raman, R., editors, *Algorithms – ESA 2002*, volume 2461 of *Lecture Notes in Computer Science*, pages 277–288. Springer Berlin / Heidelberg.

El-Fakdi, A. (2011). *Gradient-Based Reinforcement Learning Techniques for Underwater Robotics Behavior Learning*. PhD thesis, University of Girona.

El-Fakdi, A., Carreras, M., and Galceran, E. (2010). Two steps natural actor critic learning for underwater cable tracking. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2267–2272, Anchorage, Alaska, USA.

Elfes, A. (1989). Using occupancy grids for mobile robot perception and navigation. *Computer*, 22(6):46–57.

Fabrizi, E. and Saffiotti, A. (2000). Extracting topology-based maps from gridmaps. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2972–2978.

Fairfield, N., Kantor, G., and Wettergreen, D. (2007). Real-time SLAM with octree evidence grids for exploration in underwater tunnels. *Journal of Field Robotics*, 24:3–21.

Fenwick, J. and Estivill-Castro, V. (2005). Optimal paths for mutually visible agents. In Deng, X. and Du, D.-Z., editors, *Algorithms and Computation*, volume 3827 of *Lecture Notes in Computer Science*, pages 869–881. Springer Berlin / Heidelberg.

Ferguson, D., Kalra, N., and Stentz, A. (2006). Replanning with RRTs. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1243 – 1248.

Ferguson, D., Likhachev, M., and Stentz, A. (2005). A guide to heuristic-based path planning. In *Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling (ICAPS)*.

Ferguson, D. and Stentz, A. (2005). Field D*: An interpolation-based path planner and replanner. In *Proceedings of the International Symposium on Robotics Research (ISRR)*, pages 1926–1931.

Ferguson, D. and Stentz, A. (2006). Anytime RRTs. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5369 – 5375.

Ferguson, D. and Stentz, A. (2007). Anytime, dynamic planning in high-dimensional search spaces. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1310–1315.

Fernandez-Perdomo, E., Cabrera-Gamez, J., Hernandez-Sosa, D., Isern-Gonzalez, J., Dominguez-Brito, A., Prieto-Maranon, V., and Ramos, A. (2011). Adaptive bearing sampling for a constant-time surfacing A* path planning algorithm for gliders. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2350 –2355.

Firby, R. (1989). *Adaptive Execution in Complex Dynamic Worlds*. PhD thesis, Yale University, New Haven, Connecticut.

Fossen, T. (1994). *Guidance and Control of Ocean Vehicles*. Wiley.

Frazzoli, E., Dahleh, M. A., and Feron, E. (2000). Real-time motion planning for agile autonomous vehicles. *AIAA Journal of Guidence, Control and Dynamics*, 25:116–129.

Fujita, Y., Nakamura, Y., and Shiller, Z. (2003). Dual dijkstra search for paths with different topologies. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 3359–3364.

Furcy, D. A. (2006). ITSA*: Iterative tunneling search with A*. In *AAAI Workshop of Heuristic Search*.

Gao, S., Jerrum, M., Kaufman, M., Mehlhorn, K., and Rülling, W. (1988). On continuous homotopic one layer routing. In *Proceedings of the 4th annual symposium on Computational geometry*, SCG '88, pages 392–402, New York, NY, USA. ACM.

Garau, B., Alvarez, A., and Oliver, G. (2005). Path planning of autonomous underwater vehicles in current fields with complex spatial variability: an A* approach. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 194 – 198.

Grigoriev, D. and Slissenko, A. (1998). Polytime algorithm for the shortest path in a homotopy class amidst semi-algebraic obstacles in the plane. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation (ISSAC)*, pages 17–24, New York, NY, USA. ACM.

Hähnel, D., Burgard, W., Fox, D., and Thrun, S. (2003). An efficient fastSLAM algorithm for generating maps of large-scale cyclic environments from raw laser range measurements. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 206–211.

Hans and Rohnert (1986). Shortest paths in the plane with convex polygonal obstacles. *Information Processing Letters*, 23(2):71–76.

Hansen, E. and Zhou, R. (2007). Anytime heuristic search. *Journal of Artificial Intelligence Research (JAIR)*, 28:267–297.

Hansen, E. A., Zilberstein, S., and Danilchenko, V. A. (1997). Anytime heuristic search: First results. Technical report, University of Massachusetts.

Hart, P., Nilsson, N., and Raphael, B. (1968). A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107.

Hernández, E., Carreras, M., Antich, J., Ridao, P., and A.Ortiz (2011a). A topologically guided path planner for an AUV using homotopy classes. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 2337–2343, Shanghai, China.

Hernández, E., Carreras, M., Antich, J., Ridao, P., and Ortiz, A. (2011b). A search-based path planning algorithm with topological constraints. Application to an AUV. In *Proceedings of the 18th IFAC World Congress*, Milan, Italy.

Hernández, E., Carreras, M., and Ridao, P. (2011c). A path planning algorithm for an AUV guided with homotopy classes. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, Freiburg, Germany.

Hernández, E., Ridao, P., Mallios, A., and Carreras, M. (2009a). Occupancy grid mapping in an underwater structured environment. In *Proceedings of the 8th IFAC International Conference on Manoeuvring and Control of Marine Craft (MCMC)*, Guaruja, Sao Paulo, Brazil.

Hernández, E., Ridao, P., Ribas, D., and Mallios, A. (2009b). Probabilistic sonar scan matching for an auv. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 255–260, St.Louis,MO,USA.

Hernández, E., Carreras, M., Ridao, P., and Mallios, A. (2012). Homotopic path planning for an AUV on maps improved with scan matching. In *Proceedings of IFAC Workshop on Navigation, Guidance and Control of Underwater Vehicles (NGCUV)*, Porto, Portugal.

Hershberger, J. and Snoeyink, J. (1991). Computing minimum length paths of a given homotopy class. *Computational Geometry: Theory and Applications*, 4:331–342.

Hill, Jr., F. S. (1994). *The pleasures of "perp dot" products*, pages 138–148. Academic Press Professional, Inc., San Diego, CA, USA.

Hsu, D. (2000). *Randomized single-query motion planning in expansive spaces*. PhD thesis, Standford University, Stanford, CA, USA. AAI9986110.

Hsu, D., Jiang, T., Reif, J., and Sun, Z. (2003). The bridge test for sampling narrow passages with probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 4420–4426.

Hsu, D., Kindel, R., and Latombe, J.C.and Rock, S. (2002). Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255.

Hurtos, N., Mallios, A., Carreno, S., Campos, R., Lee, C., Fuster, X., Cusi, S., Galceran, E., Carrera, A., Villanueva, M., Palomeras, N., Ribas, D., and Carreras, M. (2010). Sparus, the University of Girona's entry for SAUC-E 2010. *International Journal of Maritime Engineering*, 4:167–171.

Jakuba, M. and Yoerger, D. (2008). Autonomous search for hydrothermal vent fields with occupancy grid maps. In *Australasian Conference on Robotics and Automation*, Canberra, Australia.

Jenkins, K. D. (1991). The shortest path problem in the plane with obstacles: A graph modeling approach to producing finite search lists of homotopy classes. Master's thesis, Naval Postgraduate School, Monterey, California.

Kalitzin, S., Staal, J., ter Haar Romeny, B., and Viergever, M. (2001). A computational method for segmenting topological point-sets and application to image analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(5):447–459.

Kamon, I., Rimon, E., and Rivlin, E. (1998). TangentBug: A range-sensor-based navigation algorithm. *The International Journal of Robotics Research*, 17(9):934–953.

Kamon, I. and Rivlin, E. (1997). Sensory-based motion planning with global proofs. *IEEE Transactions on Robotics and Automation*, 13(6):814–822.

Karaman, S., Walter, M., Perez, A., Frazzoli, E., and Teller, S. (2011). Anytime motion planning using the RRT*. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 1478–1483.

Kavraki, L., Svestka, P., Latombe, J.-C., and Overmars, M. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation (ICRA)*, 12(4):566–580.

Kim, J. and Ostrowski, J. (2003). Motion planning a aerial robot using rapidly-exploring random trees with dynamic constraints. In *IEEE International Conference on Intelligent Robotics and Automation (ICRA)*, volume 2, pages 2200–2205.

Koening, S. and Lickhachev, M. (2002). D*lite. In *Eighteenth national conference on Artificial intelligence*, pages 476–483, Menlo Park, CA, USA. American Association for Artificial Intelligence (AAAI).

Kruger, D., Stolkin, R., Blum, A., and Briganti, J. (2007). Optimal AUV path planning for extended missions in complex, fast-flowing estuarine environments. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4265–4270.

Kuffner, J.J., J. and LaValle, S. (2000). RRT-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 995–1001.

Kuwata, Y., Fiore, G., Teo, J., Frazzoli, E., and How, J. (2008). Motion planning for urban driving using rrt. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1681–1686.

Lacevic, B. and Rocco, P. (2010). Sampling-based safe path planning for robotic manipulators. In *IEEE Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–7.

Langer, R., Coelho, L., and Oliveira, G. (2007). K-Bug, a new bug approach for mobile robot's path planning. In *IEEE International Conference on Control Applications (CCA)*, pages 403 –408.

Latombe, J. (1991). *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA.

Laumond, J.-P., Jacobs, P., Taix, M., and Murray, R. (1994). A motion planner for nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation*, 10(5):577–593.

LaValle, S. M. Kuffner, J. J. (1999). Randomized kinodynamic planning. In *Proceedings of the IEEE International Conference on In Robotics and Automation (ICRA)*, volume 1, pages 473–479.

LaValle, S. M. Kuffner, J. J. (2001). Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press, Cambridge, U.K. Available at http://planning.cs.uiuc.edu/.

Leiserson, C. E. and Maley, F. M. (1985). Algorithms for routing and testing routability of planar vlsi layouts. In *Proceedings of the seventeenth annual ACM symposium on Theory of computing*, STOC '85, pages 69–78, New York, NY, USA. ACM.

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., and Thrun, S. (2005). Anytime dynamic A*: An anytime, replanning algorithm. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Likhachev, M., Ferguson, D., Gordon, G., Stentz, A., and Thrun, S. (2008). Anytime search in dynamic graphs. *Artificial Intelligence Journal (AIJ)*, 172(14):1613 – 1643.

Likhachev, M., Gordon, G., and Thrun, S. (2004). ARA*: Anytime A* with provable bounds on sub-optimality. In *Proceedings of the advances in Neural Inforamtion Processing Systems 16 (NIPS)*. MIT Press.

Liu, Y.-H. and Arimoto, S. (1992). Path planning using a tangent graph for mobile robots among polygonal and curved obstacles. *International Journal of Robotics Research*, 11:376–382.

Lu, F. and Milios, E. (1994). Robot pose estimation in unknown environments by matching 2D range scans. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 935–938.

Lumelsky, V. and Skewis, T. (1990). Incorporating range sensing in the robot navigation function. *IEEE Transactions on Systems, Man and Cybernetics*, 20(5):1058–1069.

Lumelsky, V. and Stepanov, E. (1987). Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430.

Maki, T., Ura, T., and Sakamaki, T. (2009). Map based path-planning and guidance scheme of an AUV for inspection of artificial structures. In *OCEANS 2009, MTS/IEEE Biloxi - Marine Technology for Our Future: Global and Local Challenges*, pages 1 –7.

Mallios, A., Ridao, P., Carreras, M., and Hernandez, E. (2011). Navigating and mapping with the SPARUS AUV in a natural and unstructured underwater environment. In *OCEANS 2011*, pages 1 –7.

Mallios, A., Ridao, P., Hernandez, E., Ribas, D., Maurelli, F., and Petillot, Y. (2009). Pose-based SLAM with probabilistic scan matching algorithm using a mechanical scanned imaging sonar. In *Oceans IEEE - EUROPE*, pages 1–6.

Mallios, A., Ridao, P., Ribas, D., and Hernández, E. (2010a). Probabilistic sonar scan matching SLAM for underwater environment. In *OCEANS 2010 IEEE - Sydney*, pages 1 –8.

Mallios, A., Ridao, P., Ribas, D., Maurelli, F., and Petillot, Y. (2010b). EKF-SLAM for AUV navigation under probabilistic sonar scan-matching. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4404 –4411. IEEE.

Mastrogiovanni, F., Sgorbissa, A., and Zaccaria, R. (2009). Robust navigation in an unknown environment with minimal sensing and representation. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 39(1):212–229.

Milgram, R. and Kaufman, S. (2000). Topological characterization of safe coordinated vehicle motions. In *IEEE International Conference onRobotics and Automation (ICRA)*, volume 3, pages 2039–2045.

Mínguez, J. (2002). *Robot Shape, Kinematics, and Dynamics in Sensor-Based Motion Planning*. PhD thesis, Universidad de Zaragoza,.

Mínguez, J., Lamiraux, F., and Montesano, L. (2005). Metric-based scan matching algorithms for mobile robot displacement estimation. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3557–3563.

Mínguez, J., Montesano, L., and Montano, L. (2004). An architecture for sensor-based navigation in realistic dynamic and troublesome scenarios. *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3:2750–2756.

Montesano, L., Mínguez, J., and Montano, L. (2005). Probabilistic scan matching for motion estimation in unstructured environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3499 – 3504.

Moravec, H. and Elfes, A. (1985). High resolution maps from wide angle sonar. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 2, pages 116–121.

Munkres, J. (1974). *Topology; a first course*. Prentice-Hall.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541 – 580.

Ng, J. and Braunl, T. (2007). Performance comparison of bug navigation algorithms. *Journal of Intelligent & Robotic Systems*, 50:73–84. 10.1007/s10846-007-9157-6.

Nilsson, N. J. (1982). *Principles of Artificial Intelligence*. Springer.

Palomeras, N. (2011). *A Mission Control System for an Autonomous Underwater Vehicle*. PhD thesis, University of Girona.

Palomeras, N., Garcia, J., M., P., Fernandez, J., Sanz, P., and Ridao, P. (2010). A distributed architecture for enabling autonomous underwater intervention missions. In *IEEE Systems Conference*, pages 159 – 164.

Palomeras, N., Ridao, P., Carreras, M., and Silvestre, C. (2009). Using petri nets to specify and execute missions for autonomous underwater vehicles. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4439–4444.

Parker, L., Birch, B., and Reardon, C. (2003). Indoor target intercept using an acoustic sensor network and dual wavefront path planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 1, pages 278–283.

Paull, L., Saeedi, S., Li, H., and Myers, V. (2010). An information gain based adaptive path planning method for an autonomous underwater vehicle using sidescan sonar. In *IEEE Conference on Automation Science and Engineering (CASE)*, pages 835–840.

Pearl, J. (1984). *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Petillot, Y., Tena Ruiz, I., and Lane, D. (2001). Underwater vehicle obstacle avoidance and path planning using a multi-beam forward looking sonar. *IEEE Journal of Oceanic Engineering*, 26(2):240–251.

Petres, C., Pailhas, Y., Patron, P., Petillot, Y., Evans, J., and Lane, D. (2007). Path planning for autonomous underwater vehicles. *IEEE Transactions on Robotics*, 23(2):331–341.

Philippsen, R. and Siegwart, R. (2005). An interpolated dynamic navigation function. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 3782–3789.

Ribas, D., Neira, J., Ridao, P., and Tardós, J. (2006). AUV localization in structured underwater environments using an a priori map. In *7th IFAC Conference on Manoeuvring and Control of Marine Crafts (MCMC)*, Lisbon, Portugal.

Ribas, D., Palomer, N., Ridao, P., Carreras, M., and Hernandez, E. (2007). Ictineu AUV wins the first SAUC-E competition. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 151–156, Roma, Italy.

Ribas, D., Palomeras, N., Ridao, P., Carreras, M., and Mallios, A. (2012). Girona 500 AUV, from survey to intervention. *IEEE/ASME Transactions on Mechatronics*, 17(1):46–53.

Ribas, D., Ridao, P., Magí, L., Palomeras, N., and Carreras, M. (2011). The Girona 500, a multipurpose autonomous underwater vehicle. In *Proceedings of the Oceans IEEE*, pages 1–5, Santander, Spain.

Ribas, D., Ridao, P., and Neira, J. (2010). *Underwater SLAM for Structured Environments Using an Imaging Sonar*. Number 65 in Springer Tracts in Advanced Robotics. Springer Verlag, Heidelberg, Germany.

Ribas, D., Ridao, P., Tardós, J., and Neira, J. (2008). Underwater SLAM in man made structured environments. *Journal of Field Robotics*, 25(11-12):898–921.

Ridao, P., Batlle, E., Ribas, D., and Carreras, M. (2004). Neptune: a HIL simulator for multiple UUVs. In *MTTS/IEEE OCEANS*, volume 1, pages 524 – 531.

Ridao, P., Batlle, J., and Carreras, M. (2002). O2CA2, a new object oriented control architecture for autonomy: the reactive layer. *Control Engineering Practice*, 10(8):857–873.

Ridao, P., Carreras, M., Ribas, D., and Garcia, R. (2010). Visual inspection of hydroelectric dams using an autonomous underwater vehicle. *Journal of Field Robotics*, 27(6):759–778.

Ridao, P., Ribas, D., Hernandez, E., and Rusu, A. (2011). USBL/DVL navigation through delayed position fixes. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2344 –2349.

Roman, C. and Singh, H. (2005). Improved vehicle based multibeam bathymetry using sub-maps and SLAM. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3662–3669.

Sanchez, G. and Latombe, J.-C. (2002). On delaying collision checking in prm planning – application to multi-robot coordination. *International Journal of Robotics Research*, 21:5–26.

Sarid, S., Shapiro, A., and Gabriely, Y. (2007). MRBUG: A competitive multi-robot path finding algorithm. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 877 –882.

Schmiing, M., Afonso, P., Tempera, F., and Santos, R. (2009). Integrating recent and future marine technology in the design of marine protected areas - the azores as case study. In *OCEANS-EUROPE*, pages 1 – 7.

Schmitzberger, E., Bouchet, J., Dufaut, M., Wolf, D., and Husson, R. (2002). Capture of homotopy classes with probabilistic road map. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 3, pages 2317–2322.

Seifert, H., Threlfall, W., Birman, J., and Eisner, J. (1980). *Seifert and Threlfall, A textbook of topology*. Pure and applied mathematics. Academic Press.

Sethian, J. (1996). *Level Set Methods. Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge University Press.

Shiller, Z., Fujita, Y., Ophir, D., and Nakamura, Y. (2004). Computing a set of local optimal paths through cluttered environments and over open terrain. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 5, pages 4759 – 4764.

Shkolnik, A., Walter, M., and Tedrake, R. (2009). Reachability-guided sampling for planning under differential constraints. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 2859 –2865.

Silver, D., Bradley, D., and Thayer, S. (2004). Scan matching for flooded subterranean voids. In *IEEE Conference on Robotics Automation and Mechatronics (RAM)*, volume 1, pages 422–427.

Smith, R., Pereira, A., Chao, Y., Li, P., Caron, D., Jones, B., and Sukhatme, G. (2010). Autonomous underwater vehicle trajectory design coupled with predictive ocean models: A case study. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4770 –4777.

Smith, R., Self, M., and Cheeseman, P. (1990). Estimating uncertain spatial relationships in robotics. In *Autonomous robot vehicles*, pages 167–193, New York, NY, USA. Springer-Verlag New York, Inc.

Speckmann, B. and Verbeek, K. (2010). Homotopic rectilinear routing with few links and thick edges. In López-Ortiz, A., editor, *LATIN 2010: Theoretical Informatics*, volume 6034 of *Lecture Notes in Computer Science*, pages 468–479. Springer Berlin / Heidelberg.

Stentz, A. (1995). The focussed D* algorithm for real-time replanning. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Svestka, P. and Overmars, M. (1998). *Robot Motion Planning and Control (LNCIS, 229)*, chapter Probabilistic path planning, pages 255–304. Springer Verlag.

Takahashi, O. and Schilling, R. (1989). Motion planning in a plane using generalized voronoi diagrams. *IEEE Transactions on Robotics and Automation*, 5(2):143 –150.

Tang, Z. and Ozguner, U. (2005). Motion planning for multitarget surveillance with mobile sensor agents. *IEEE Transactions on Robotics*, 21(5):898 – 908.

technology corp., I. (2002). Sonar theory and applications. `http://www.imagenex.com/sonar_theory.pdf`. [Online; accessed 12-September-2007].

Thomas, U. and Iser, R. (2010). A new probabilistic path planning algorithm for (dis)assembly tasks. *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, pages 1 –6.

Thrun, S., Burgard, W., and Dieter, F. (2005). *Probabistic Robotics*. The MIT Press.

Tovar, B., Murrieta-Cid, R., and LaValle, S. (2007). Distance-optimal navigation in an unknown environment without sensing distances. *IEEE Transactions on Robotics*, 23(3):506–518.

Turner, R. (2005). Intelligent mission planning and control of autonomous underwater vehicles. In *International Conference on Automated Planning and Scheduling (ICAPS)*, Monterey, California, USA.

Urick, R. (1983). *Principles of underwater sound*. Peninsula Publishing, Los Altos, California, 3rd edition edition.

Urmson, C. and Simmons, R. (2003). Approaches for heuristically biasing RRT growth. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, volume 2, pages 1178–1183.

Vasudevan, C. and Ganesan, K. (1994). Case-based path planning for autonomous underwater vehicles. In *IEEE International Symposium on Intelligent Control*, pages 160–165.

Warren, C. (1990). A technique for autonomous underwater vehicle route planning. *IEEE Journal of Oceanic Engineering*, 15(3):199–204.

Williams, D. (2010). On optimal auv track-spacing for underwater mine detection. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4755 –4762.

Williams, S., Dissanayake, G., and Durrant-Whyte, H. (2002). Towards multi-vehicle simultaneous localisation and mapping. In *IEEE International Conference on Robotics and Automation (ICRA)*, volume 3, pages 2743–2748.

Yang, C. (1997). The smallest pair of noncrossing paths in a rectilinear polygon. *IEEE Transactions on Computers*, 46(8):930–941.

Yang, J. and Sacks, E. (2006). RRT path planner with 3dof local planner. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 145 –149.

Zhang, Z., Sturtevant, N., Holte, R., Schaeffer, J., and Felner, A. (2009). A* search with inconsistent heuristics. In *Proceedings of the 21st international jont conference on Artifical intelligence*, IJCAI'09, pages 634–639, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Zhou, R. and Hansen, E. A. (2002). Multiple sequence alignment using anytime A*. In *Eighteenth national conference on Artificial intelligence*, pages 975–976, Menlo Park, CA, USA. American Association for Artificial Intelligence (AAAI).

Zhou, R. and Hansen, E. A. (2005). Beam-stack search: Integrating backtracking with beam search. In *Proceedings of the 15th International Conference on Automated Planning and Scheduling (ICAPS)*, pages 90–98.

Zhu, Y., Zhang, T., Song, J., and Li, X. (2010). A new bug-type navigation algorithm considering practical implementation issues for mobile robots. In *IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 531–536.