Uncertainty-based online mapping and motion planning for marine robotics guidance

Èric Pairet Artau

Supervisors:

Dr. Marc Carreras[§]
Dr. Morteza Lahijanian[†]
Sr. Juan David Hernández[§]

[§]University of Girona — [†]University of Oxford

A Thesis Submitted for the Degree of
MSc Erasmus Mundus in Vision and Robotics (VIBOT)

· 2017 ·

Abstract

In real-world robotics, path planning remains to be an open challenge; not only robots are asked to move through unexplored environments, but also the motion of robots is constrained by their dynamics. At the same time, such dynamics typically suffer from uncertainties, which should be taken into account for completely ensuring the feasibility of the path and the robot's safety.

The state-of-the-art usually addresses those issues separately. Planning online requires being able to quickly update the path according to the incremental knowledge of the environment. Such prescription is hard to be satisfied when considering the system dynamics and its uncertainty because a policy over the entire belief space must be constructed.

This work proposes an incremental mapping-planning framework that jointly addresses these challenges for achieving fast replanning. The framework is threefold: (1) the environment is represented as a collection of local maps, for each of which the system has a relative uncertainty so (2) the probability of colliding with the environment can be probabilistically checked and (3) the feasibility of the path is ensured by considering the kinodynamic constraints of the system.

The proposed framework is evaluated with the Sparus II AUV, a torpedo-shaped vehicle suffering from nonholonomic constraints. The experiments are conducted in simulated and real-world environments, such as a breakwater structure and a natural passage. Results show the potential of the method for planning under motion and probabilistic constraints in uncertain environments while being suitable for systems with limited computational power.

Contents

List of Figures

List of Tables

Acronyms

2D two-dimensional

3D three-dimensional

6D six-dimensional

AUV autonomous underwater vehicle

B-Space belief space

BRM belief roadmap

C-Space configuration space

CIRS Centre Investigació en Robòtica Submarina

CC-RRT chance constrained RRT

COLA2 component oriented layer-based architecture for autonomy

CL-RRT closed-loop RRT

DoF degree of freedom

DVL doppler velocity logger

EST expansive-spaces tree

FIRM feedback-based information roadmap

FoV field of view

GPS global positioning system

IMU inertial measurement unit

LoS line-of-sight

NOAA National Oceanic and Atmospheric Administration

MDP markov decision process

MSIS mechanical scanned imaging sonar

OMPL open motion planning library

PD proportional derivative

POMDP partially observable markov decision process

PRM probabilistic roadmap

RHC receding horizon control

ROS robot operating system

ROV remotely operated vehicle

RRT rapidly-exploring random tree

RViz ROS visualizer

SMR stochastic motion roadmap

SST stable sparse RRT

U-Space control space

UdG Universitat de Girona

UWSim underwater simulator

VICOROB VIsió per COmputador i ROBòtica

X-Space state space

Acknowledgments

This master thesis becomes a reality with the priceless support and help of many individuals. I would like to extend my sincere thanks to all of them.

Foremost, my deepest gratitude goes to all my advisors, especially to Sr. Juan David Hernández, for his time, the countless fruitful discussions and his technical guidance whenever it was needed. Also, to Dr. Morteza Lahijanian for his theoretical insights, for being an insatiable source of ideas and his artistic advice in the writings. Last but not least, to Dr. Marc Carreras for our morning meeting-runs and ensuring that this work was coming to fruition.

This work would neither have been possible without all the people from CIRS; they all have made the path through this thesis smoother. Among them, I am deeply indebted to Dr. Pere Ridao for his ideas about mapping, to all the people in charge of the robot's hardware and software maintenance and to all those with whom I have exchanged some ideas about this work during the coffee breaks.

Finally, the warmest gratitude to my family and friends. Their daily encouragement and moral support have always pushed me to give my best.

Chapter 1

Introduction

Since the dawn of time, human being has had the need of deeply understanding everything that happens around us. This has propelled the research and development of different technologies in order to achieve a better understanding in various areas of knowledge. When it comes to the surroundings exploration, most of the effort has been focused on the Earth's crust. Also, some researchers have been captivated by the mysteries hidden beyond the sky. However, even though oceans cover nearly the 71% of the Earth, according to the National Oceanic and Atmospheric Administration (NOAA), the 90% of the seafloor remains unexplored.

1.1 Context

The late exploration of maritime environments has been mainly due to their challenging conditions; high pressures, currents, and reduced visibility, just to name a few, have made underwater environments an unexplorable place for the human being. In order to cope with these rough conditions, different mechanisms have been developed in the last years. Among them, and probably the most relevant nowadays, are the underwater robots. Their use seeks not only to avoid exposing human lives in dangerous conditions, but also to provide higher autonomy to perform complex tasks for longer periods of time.

Underwater robots are usually divided into two groups: remotely operated vehicles (ROVs) and autonomous underwater vehicles (AUVs). The former group consists of tethered robots, which are remotely controlled by an operator. Their major drawback is the necessity of a continuous connection to a vessel, which not only limits the working area, but also leads to high operating costs. On the other hand, AUVs are robots which can cover larger areas at a lower cost because they do not require being teleoperated from a vessel. Because of that, the usage of AUVs has been a hot topic in research in the last decades.

The Centre Investigació en Robòtica Submarina (CIRS) is a laboratory from the VIsió per COmputador i ROBòtica (VICOROB) institute at the Universitat de Girona (UdG). CIRS has been actively doing research on underwater vision and robotics, while developing their own AUVs since 1995. Nowadays, CIRS has two operative vehicles, the Girona 500 [2] and the Sparus II [3], which are used for evaluating new hardware and software developments for underwater domains.

Although these robots have proved to be capable of conducting complex tasks, e.g. structure inspection, underwater manipulation and seafloor mapping, there are other capabilities such as the path planning that still need to be further improved.

1.2 Motivation

Path planning is the branch of robotics aiming for a path that connects two configurations while satisfying different requirements such as safety, efficiency and feasibility of the path. Although the path planning problem has been studied for terrestrial, aerial and, to a lesser extent, underwater vehicles, there are new applications that require robust planners to deal with more challenging scenarios. As the complexity of the system and the intricacy of the environment increases, the harder planning a path becomes. Thus, planning feasible and secure paths in underwater environments where the prior information of the environment is null still supposes a great challenge for the path planning community.

Many real-world systems have limited mobility due to its mechanical constitution. Because of that, traditional approaches of considering a point mass object capable of freely moving in the space turns to be completely unrealistic when planning a path for those systems. Therefore, the planner should consider the system's motion capabilities to generate a path that is more likely to be feasible for the vehicle. This idea is commonly referred to planning under kinodynamic or differential constraints or, in general, motion planning.

Kinematics and dynamics usually suffer from uncertainties. When a system lacks a precise localisation system, those uncertainties become meaningful, thus increasing the error associated with the state estimation. Although many efforts have been made to improve the system localisation, considering those uncertainties when planning a path is still indispensable for ensuring the safety of the system.

Some of the aforementioned difficulties have already been studied in the robotics community, but current solutions do not tackle all the challenges at the same time. Moreover, some of the proposed algorithms are computationally expensive, thus avoiding their use for real-time applications, especially when the environment is not known beforehand. This master thesis proposes an incremental mapping and planning framework to jointly address the challenge of planning a path under kinodynamic and probabilistic constraints along undiscovered environments.

1.3 Objectives

The main goal of this thesis is to develop a framework which plans and executes a two-dimensional (2D) path over unexplored environments while ensuring the feasibility of the path and probabilistically guaranteeing the robot's safety. At the same time, the proposed approach must accomplish the following requirements:

- To cope with real environments, where most of the obstacles generally adopt non-convex shapes.
- To be computationally feasible for real-time and real-world robotic applications.

- To not be an ad-hoc framework for a specific system, so it can be extended to other robotic systems.
- To be oriented to cope, in the future, with three-dimensional (3D) environments.

In order to meet these properties, the main objective has been broken down into the following tasks:

- To review the state-of-the-art of path planning under kinodynamic constraints, under uncertainty, and along undiscovered environments.
- To integrate the kinodynamic and probabilistic constraints into a framework that meets online computation constraints.
- To evaluate the proposed framework in both simulated and real environments.

1.4 Outline

The remainder of this thesis is organised as follows:

- **Chapter 2** presents the state-of-the-art on path planning algorithms which either deal with kinodynamic constraints, uncertainties or unknown environments.
- **Chapter 3** introduces the methodology for considering both motion and probabilistic constraints in the path planning problem.
- **Chapter 4** details how the safety of the path is probabilistic guaranteed even with an incremental knowledge of the environment.
- **Chapter 5** explains the proposed framework for incrementally mapping the surrounding, while simultaneously planning paths that meet motion and probabilistic constraints.
- **Chapter 6** reports and evaluates the results obtained in both simulated and real-world scenarios.
- **Chapter 7** revisits the work done in this master thesis and suggests future work.

Chapter 2

State-of-the-art

Although path planning is the main research area behind the work presented throughout this thesis, the validation of the proposed approach in real-world scenarios has involved other fields of study such as perception, mapping, navigation, and control. Since reviewing the state-of-the-art for all those areas is out of the scope of this manuscript, this chapter mainly focuses on reviewing the most relevant path planning techniques.

2.1 Background

A basic path planning strategy can be adapted and expanded to address a wide range of requirements and constraints. Because of that, this section reviews not only some of the most relevant path planning approaches, but also different contributions that cope with more specific challenges such as path feasibility, obstacle avoidance, and online planning.

2.1.1 The path planning problem

Probably one of the oldest areas of research in robotics is path planning, dating back to the 60's [4]. At that time, the path planning problem was addressed and solved over the workspace $\mathcal{W} = \mathbb{R}^{n_d}$, being $n_d = 2$ for systems moving on a plane (2D space) and $n_d = 3$ for systems performing their tasks in a volumetric environment (3D space). However, this approach is not scalable for high-dimensional systems, i.e. systems involving a high number of DoFs. It was not until the 80's, when Lozano-Pérez introduced the concept of the configuration space (C-Space) [5], [6], a set of all possible configurations q that the system can adopt, that path planning became an active area of research. The C-Space can be subdivided into the free space $\mathcal{C}_{free}$ and the obstacle region $\mathcal{C}_{obs}$, which correspond to those configurations that are collision-free and under collision, respectively. Thus, path planning aims to find a continuous path $p : [0,\ 1] \rightarrow \mathcal{C}_{free}$, such that $p(0) = q_{start}$ and $p(1) = q_{goal}$.

Many algorithms have been proposed to address the path planning problem. Probably the earliest ones are the Bug methods, which are reactive and sensor-based approaches inspired by insects. These algorithms assume that the system can only have a local knowledge of the environment, which is used for circumnavigating the obstacles until reaching the goal [7], [8]. Another approach

is to use artificial potential fields, where the goal is represented by an attractive field, while the obstacles are represented with repulsive fields [9]. The total field results in an optimisation problem that can guide the robot towards the goal [10].

The information in the C-Space has been widely exploited with theory of graphs. Grid-based algorithms discretise the C-Space to create a graph, which is later explored using a graph search method to find a collision-free path. Some search methods are A* [11], D* [12] and the Dijkstra's [13] algorithm, from which many variants have been proposed. Another discrete approach is sampling-based methods, which are further divided into single and multiple start-to-goal-query algorithms. The former builds a graph or tree out of random samples from the C-Space (nodes) and connecting them (edges) when possible, while the later creates a roadmap which contains the information about all feasible routes according to a specific set of properties [14]. The most widely known sampling-based algorithms are the expansive-spaces tree (EST) [15], the rapidly-exploring random tree (RRT) [16] and the probabilistic roadmap (PRM) [17]. Based on them, there have been proposed different extensions such as the RRT* and PRM*, which incorporate reconnection strategies to obtain asymptotic optimal solutions [18].

2.1.2 Towards path feasibility

Real-world robotics usually suffer from limited manoeuvrability. This fact has to be contemplated by the planner to ensure feasible paths for a specific system. However, traditional path planning approaches cannot cope with this requirement. Thus, Donald et al. coined back the concept of kinodynamic constraints in 1993. It refers to considering kinematic and/or dynamic aspects of the system [19]. When such constraints are integrated into the path planning problem, it seeks to find a feasible path in state space (X-Space) or, in other words, to find a set of states z or motions which are feasible according to control space (U-Space). Then, the motion planning problem aims to find a continuous path $p : [0,\ T] \rightarrow$ U-Space, such that $p(0) = z_{start}$ and $p(T) = z_{goal}$.

The motion planning problem is usually addressed with either grid-based or sampling-based methods [20], [21]. However, motion planning can be also addressed with pure mathematical-based approaches, which describe both the $\mathcal{W}$ and the system in a mathematical form. Even though this threefold classification, some motion planning problems require a combination of algorithms to find a solution path [20].

One of the earliest works in motion planning used a grid-based approach for building a graph from the X-Space. The obtained cells (nodes) were connected (edges) by applying feasible system inputs [19]. Later proposals inspired by this work weighted the connection between nodes according to the environmental information while the system constraints were considered by an upper layer of the framework [22], [23]. An alternative to the traditional grid-based approach was presented in [24], where a multi-resolution lattice was built by considering the system capabilities and constraints. Once obtained a graph that includes the system constraints, any search method can be used to find a solution from the starting state to the goal state. Alternatively, other search-based methods have been proposed to include the system constraints during the search, such as AD* and Hybrid-State A* [25].

Sampling-based methods have also been widely used for motion planning. In fact, the original RRT [16] was already taking the system capabilities into account. In this case, the tree was

expanded using feasible control inputs, which were obtained from uniformly sampling the U-Space. Nowadays, there can be found several RRT variants. One of the most outstanding variants is the closed-loop RRT (CL-RRT), which considers not only the vehicle's motion model, but also the controller's dynamic behaviour [26]. Among the RRT-based algorithms that can be found in the literature, none of them provides optimal paths. Actually, achieving this property when considering kinodynamic constraints remains to be a formidable challenge, because a steering function is required [27].

A kinodynamic RRT* for linear systems was presented in [28]. However, this approach is limited because most of the systems under kinodynamic constraints have a nonlinear mathematical model. In order to simplify the model and to overcome this lack of optimality, there are different options: (i) to linearise the system [29], [30], [31], (ii) to approximate the kinodynamic constraints by using geometric solutions such as Dubins curves [32] or Red Shepp curves [33], and (iii) to dodge the need of steering functions by using a shooting approach [34]. This later option has been recently adopted in a tree-based method, conceiving the stable sparse RRT (SST) algorithm [35].

2.1.3 Planning over the $\mathcal{X}_{\text{free}}$

Determining if a state x belongs to the $\mathcal{X}_{\text{free}}$ turns to be an indispensable ability of any motion planning algorithm. This task has been widely addressed considering point-mass systems, so a state x is said to be valid if it belongs to the $\mathcal{X}_{\text{free}}$. The same approach has been used when planning for rigid bodies, but the $\mathcal{X}_{\text{free}}$ is shrunk accordingly to the dimensions of the system. Many other heuristics have been proposed in the literature, such as the so-called clearance, which aims to maximise the distance to all the obstacles with the shortcoming of being computationally expensive [36]. Recently, another heuristic has been proposed, which defines a set of risk zones around the state to evaluate its associated risk, so it could be accepted or rejected according to a defined threshold [37].

Those heuristics might not guarantee a high level of robustness in real-world robotics, where systems and the environment usually suffer from uncertainties. In these scenarios, decision-making is usually conducted in the belief space (B-Space). This kind of planning problems is commonly formulated as a partially observable markov decision process (POMDP) [38]. Alterovitz et al. introduced the stochastic motion roadmap (SMR) algorithm, which uses a sampling-based roadmap representation of the C-Space to formulate a markov decision process (MDP) [39]. Without addressing this challenge using MDP-based approaches, Censi et al. propagated both the state and the uncertainty on a grid-based representation of the extended space of poses $\times$ covariances [40]. Similarly, the belief roadmap (BRM) algorithm is a variant of the PRM algorithm which is evolved in the B-Space [41], and the feedback-based information roadmap (FIRM) builds a graph in the B-Space, where nodes are beliefs and edges are local controllers [42]. Blackmore et al. used linear chance constraints together with disjunctive linear programming to efficiently perform probabilistic convex obstacle avoidance [43]. This formulation later led to the chance constrained RRT (CC-RRT) [44]. In contrast to the previous works, a mathematical approach was presented in [45], where planning is conducted in the continuous B-Space by finding the best path using nonlinear optimisation methods.

As stated in [46], considering while planning the cumulative nature of configuration uncertainty

to reach distant goals is essential. Thus, real-world robotic applications need the presence of landmarks to bound and reduce their location uncertainty. In fact, this challenge turns to be a completely different field of research from the planning one.

2.1.4 Online planning

Planning in unexplored environments requires incrementally mapping the surroundings while simultaneously planning feasible motions. This can only be achieved either if the planning area is small enough, or the planning horizon is short [45]. In order to deal with these constraints, motion planning algorithms must be computationally efficient by making use of a reasonable amount of memory and being able to cope with partially known environments. All those requirements are a must because online planning algorithms are intended to work on mobile robots, where the computational power is usually limited and the system safety might depend on being able to take decisions in a short period of time.

Probably the first online motion planning works were the aforementioned Bug methods [7], [8]. Those algorithms proved to be capable of conducting online navigation with a finite amount of memory, but without any kind of optimality guarantee, especially due to their limited and local knowledge of the surroundings [47]. From these initial approaches, several extensions have been presented to make the motion planning algorithms more suitable for applications with online computation constraints. Some of them proposed updating the exploration of the X-Space whenever a new portion of the environment was discovered. Another common approach is to endow the planner with anytime computation capabilities, which means that the planner can provide the best partial solution when required [48], [49]. Hernández et al. considered using the last best-known solution as a starting point for the next planning cycle. This approach, however, uses a geometric formulation to define the motion constraints of AUVs [37]. Many other works have assumed finite planning horizons, so they have proposed receding horizon control (RHC) approaches, which imply an increase of the computational costs [50], [51].

Establishing the validity of the states associated with the solution path can be computationally expensive. This is especially critical for sampling-based approaches, where a complete description of the C-Space or the X-Space is not available, thus requiring to conduct state validity checks several times. One alternative to partially deal with this is the lazy collision checking strategy [52]. It was initially used with the PRM, where the roadmap is built without checking the validity of the random states and their connections. Instead, this validation is delayed until the start-to-goal query has to be solved, so only those connections under collision in the solution path are discarded, while new alternatives are evaluated from the originally built roadmap. This method considerably improves the overall computation time. There are other approaches which are also based on this strategy. Bekris et al., for instance, proposed a motion planning framework for terrestrial vehicles, which only validates the states once a solution path has been found [53]. The same approach was adopted for the high dimensional C-Space of a humanoid robot [54]. A similar approach is called opportunistic collision checking which assumes that any state is valid if is located in unexplored areas [37].

2.2 Discussion

After reviewing the state-of-the-art, the main challenges identified for this work are (i) dealing with a high-dimensional C-Space due to the insertion of kinodynamic constraints into a path planning algorithm, (ii) lacking of an optimal solution path because there is not any optimal path planning algorithm for nonlinear models, (iii) performing probabilistic state validity checking, which becomes a computationally expensive task and (iv) planning online because it requires a fast algorithm able to cope with partial and variant knowledge of the environment. All those factors have been jointly considered to propose the work presented in this dissertation.

The use of potential fields does not fulfil the requirements of this work because they need an explicit representation of the environment and they suffer from local minima. Regarding grid-based approaches, they also need an explicit representation of the environment and, on top of that, they do not entirely exploit the dynamic range of the system because of the X-Space discretisation, thus reducing the set of possible manoeuvres. Apart from that, neither potential fields, grid-based approaches nor the roadmaps perform efficiently when the dimensionality of the system grows.

To the best of the authors' knowledge, sampling-based algorithms might be a suitable option to address this thesis goal. They provide a fast exploration of a high dimensional X-Space, making them suitable for (i) integrating kinodynamic and probabilistic constraints in the planning problem, and (ii) adopting an anytime replanning strategy to overcome the online planning requirement. Moreover, sampling-based methods are highly adaptable, which might allow integrating all those factors in the same algorithm. Sampling-based algorithms have clearly demonstrated their advantages in complex tasks where other approaches had failed. However, these algorithms also have weaknesses that must be always taken into account. It has been previously stated that those algorithms neither provide optimal paths with nonlinear systems suffering from kinodynamic constraints nor are able to report when a motion planning problem has no solution. Furthermore, their implementation is usually reduced at a single thread, i.e. it can not be parallelised, and they generally do not perform well in bug trap like scenarios.

This dissertation proposes a framework suitable to jointly address these problems in real-robotics applications. Specifically, a tree-based algorithm builds a tree in the B-Space, where nodes represent uncertain states (beliefs), and edges are local controllers that meet the kinodynamic constraints to evolve from one state to another. All the nodes and edges are probabilistically guaranteed to be below a predefined threshold of collision. In order to provide online planning capabilities to the framework, an anytime policy is adopted.

Chapter 3

Planning under motion and probabilistic constraints

Mechanical systems generally operate under motion constraints that limit their manoeuvrability. Therefore, in order to guarantee the feasibility and robustness of such systems when following desired motions, those constraints must be included into the motion planner. In doing so, a mathematical model that approximates the system behaviour is required. In real robots, simplified models are preferred even though they introduce some uncertainty. This chapter firstly details the sampling-based methodology that has been used to deal with both motion and probabilistic constraints. Then, it introduces the mathematical model of a nonholonomic system, specifically a car-like system which moves in a 2D workspace, and its modifications to make it suitable for such a sampling-based method.

3.1 Tree-based methods for dealing with constraints

As mentioned in Chapter 2, sampling-based algorithms mainly use two types of data structures: graphs, in the case of the PRM, and trees, in the case of the EST and RRT. Although both approaches share most of the main characteristics, such as the use of random configurations from the C-Space, apart from some exceptions, tree-based algorithms are commonly used to deal with motion constraints. These methods basically build a tree of feasible (valid and doable) states. To do so, the algorithm expands a tree towards randomly sampled states by using the mathematical model in Algorithm 1. An additional step that is implicitly included in this expansion is the state validity checking. This validation seeks to guarantee that the different states, randomly sampled or resulted from the tree expansion, are valid under certain constraints. While in the simplest case the states are only geometrically verified, in more elaborated approaches, such as the one developed in this work, the state must meet both motion and probabilistic constraints.

In Algorithm 1, a tree $\mathcal{T}$ is rooted at z_{init} in the X-Space. Then, an iterative procedure that includes a *selection*, a *propagation* and a *validation* routine, is done until the tree $\mathcal{T}$ reaches a goal region, i.e. the z_{goal} with some tolerance. The *selection* process starts by uniformly sampling the X-Space to obtain z_{rand} (line 3) which, at the same time, is used to select the nearest state z_{near} of

Algorithm 1: EXPLORATION_TREE(z_{init}, z_{goal}, X-Space, U-Space) [27]

1 $\mathcal{T} = \{\mathbb{V} \leftarrow \{z_{init}\}, \mathbb{E} \leftarrow \varnothing\}$

2 **while** *not* STOP_CONDITION(z_{goal}, $\mathcal{T}$) **do**
3 $z_{rand} \leftarrow$ RANDOM_STATE(X-Space)
4 $z_{near} \leftarrow$ NEAREST_NEIGHBOUR(z_{rand}, $\mathbb{V}$)
5 u_{new}, $T_{prop} \leftarrow$ RANDOM_CONTROL(U-Space)
6 $z_{new} \leftarrow$ PROPAGATE_STATE(z_{near}, u_{new}, T_{prop})
7 **if** VALID($\overline{z_{near} \rightarrow z_{new}}$) **then**
8 $\mathbb{V} \leftarrow \mathbb{V} \cup \{z_{new}\}$
9 $\mathbb{E} \leftarrow \mathbb{E} \cup \{u_{new}\}$
10 **return** $\mathcal{T}(\mathbb{V}, \mathbb{E})$

the tree (line 4). Such a node (state) is the selected one to expand the tree during the *propagation* routine. In the original formulation of the RRT, a feasible control input u_{new} is obtained by uniformly sampling the U-Space (line 5). Then, u_{new} is applied to dynamically evolve from z_{near} for a period of time T_{prop} to generate z_{new} (line 6). Finally, the *validation* step is conducted. If z_{new} and the motion $\overline{z_{near} \rightarrow z_{new}}$ are valid, they are both added to the tree (line 7).

Expanding the tree by applying a set of feasible control inputs to the system's model allows the planner to incorporate the system's capabilities. However, this approach does not guarantee that the tree is expanded towards z_{rand}. To overcome this, some works propose directing the growth of the tree towards the random samples using a closed-loop model of the system. In what concerns the validation of probabilistic state constraints in a sampling-based method, Luders et al. proposed the chance constrained RRT (CC-RRT) [44]. This algorithm considers a state to be valid when the intersection between the Normal distribution that defines the state's belief and the obstacles in the map is below a predefined threshold.

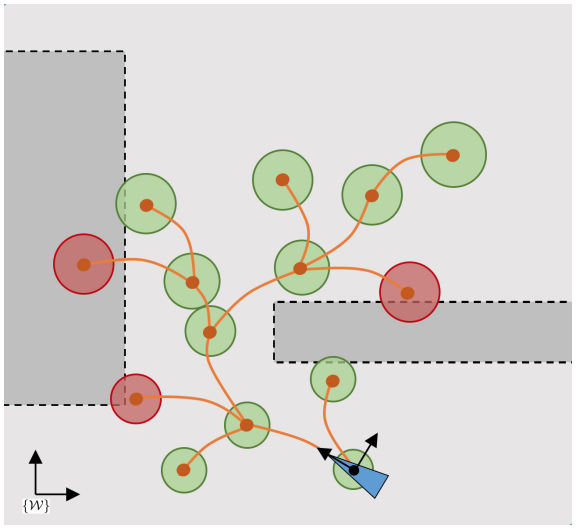


Figure 3.1: Tree expansion under motion and probabilistic constraints.

Figure 3.1 depicts the expected behaviour of a tree-based algorithm that considers both the motion and the probabilistic constraints. In this case, the edges of the tree are guaranteed to be feasible for a modelled system. Regarding the nodes, only those involving both a motion (states along the edge) and a final state with minimum safety guarantees are finally included in the tree. This is indicated with a surrounding green sphere, the size of which is proportional to the uncertainty of the state on the node.

3.2 Nonholonomic system

The state of a nonholonomic system depends on the path taken in order to achieve it. This fact frequently occurs in underactuated systems, i.e. vehicles which have a lower number of actuators than DoFs. In its most generic form, such a constraint is usually modelled as:

$$\dot{\mathrm{s}} = \mathrm{f(s)} + \mathrm{g(s)u}, \tag{3.1}$$

where the state $\mathrm{s} \in \mathbb{R}^{\mathrm{n_s}} \subset \mathcal{X}$ and the control $\mathrm{u} \in \mathbb{R}^{\mathrm{n_u}} \subset \mathcal{U}$.

In the scope of this project, Equation 3.1 has to be accommodated for marine vehicles operating in a 2D workspace, i.e. $\mathcal{W} = \mathbb{R}^2$. Generally, those systems either (i) move forward with one propeller and turn using fins, or (ii) move forward and turn applying a differential control strategy on two propellers. In both cases, the system state is defined as $\mathrm{s} = [\mathrm{x},\ \mathrm{y},\ \vartheta]$, i.e. C-Space $= \mathcal{C} = \mathrm{SE(2)} = \mathbb{R}^2 \times \mathrm{SO(2)} = \mathbb{R}^2 \times \mathcal{S}^1$, where [x, y] correspond to the Cartesian coordinates of the system with respect to its reference frame, and ϑ is the orientation with respect to the x-axis, while the system control is defined as $\mathrm{u} = [\mathrm{v},\ \omega]$, i.e. U-Space $= \mathcal{U} = \mathbb{R}^2$, where v is the forward velocity and ω is the turning rate.

From this characterisation of the C-Space and the U-Space, it is important to note that those systems are not able to move sideways so they movement on a 2D plane is constrained by the relation of their control inputs. This fact makes the model of the aforementioned marine vehicles similar to the model of a car-like system (Figure 3.2).

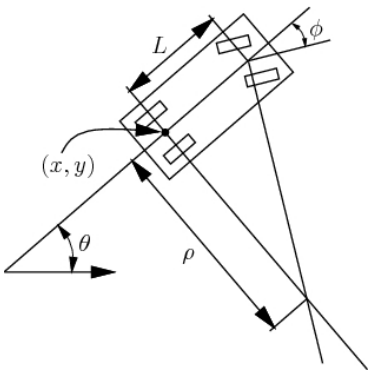


Figure 3.2: Typical car-like system with 3 DoF but with only 2 control inputs. Image credit: LaValle [1].

Apart from acknowledging that a marine vehicle operating in a 2D workspace behaves as a car-like system, this work also assumes that the system kinematics can be initially approximated to a

driftless robotic system, i.e. $f(s) \equiv 0$. This hypothesis lets us temporally reduce the complexity of the model, something that will be reformulated at the end of this section. Thus, the kinematic model of the system with $n_s = 3$ and $n_u = 2$ is given by:

$$\begin{aligned} \dot{x} &= v \cos(\vartheta), \\ \dot{y} &= v \sin(\vartheta), \\ \dot{\vartheta} &= \omega. \end{aligned} \tag{3.2}$$

In order to integrate such a model in the tree expansion procedure stated in Section 3.1, the control inputs $u = [v,\ \omega]$ which drive the system towards the random states z_{rand} must be found. This challenge can be addressed using a controller, but then the system needs to be linearised. Because of that, the remainder of this section first linearises the system described in Equation 3.2, so a controller can be then applied. Finally, the initial driftless system assumption is reformulated.

3.2.1 Dynamic feedback linearisation

Linearisation refers to establishing a linear approximation to a function at a given point. In the case of mechanical systems, linearisation is usually calculated for a specific working or equilibrium point. However, linearisation via a static state feedback does not provide a smooth system response in a multi-working point system and it can never be exact [29]. The linearisation tangent at every equilibrium point is uncontrollable [55]. Taking into account that robotic systems usually operate at different working points, a dynamic feedback linearisation approach has been considered to overcome the aforementioned limitations.

This method adjusts the control $u = [v,\ \omega]$ of the open-loop system defined in Equation 3.1 and Equation 3.2 according to the full system state, i.e. not only the state information contained in s but some extra ξ. The dynamic feedback controller is expressed as:

$$\begin{aligned} \dot{\xi} &= a(s,\ \xi) + b(s,\ \xi)w, \\ u &= c(s,\ \xi) + d(s,\ \xi)w, \end{aligned} \tag{3.3}$$

where the extra state information $\xi \in \mathbb{R}^{n_\xi} \subset \mathcal{X}$ and the input $w \in \mathbb{R}^{n_w} \subset \mathcal{U}$.

According to [56], a system can be exactly linearised via a dynamic feedback of dimension $n_\xi = n_s - 2$. Thus, defining $\xi = v$ and the control input $w = [\ddot{x},\ \ddot{y}]$, the dynamic feedback linearisation yields the controller in Equation 3.4. Note that such a controller has a singularity at $\xi = 0$, which will be addressed in further sections.

$$\begin{aligned} \dot{\xi} &= \cos(\vartheta)w_1 + \sin(\vartheta)w_2, \\ \omega &= \frac{-1}{\xi}\sin(\vartheta)w_1 + \frac{1}{\xi}\cos(\vartheta)w_2. \end{aligned} \tag{3.4}$$

The integration of the designed dynamic feedback controller let us to represent the system with

a closed-loop formulation under the state transformation $z = T(s,\ \xi)$:

$$\begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ z_4 \end{bmatrix} = \begin{bmatrix} s_1 \\ \xi \cos s_3 \\ s_2 \\ \xi \sin s_3 \end{bmatrix} = \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix}, \tag{3.5}$$

which is equivalent to a linear system of the form:

$$\dot{z} = Az + Bw, \tag{3.6}$$

where $z \in \mathbb{R}^{n_z} \subset \mathcal{X}$, $A \in \mathbb{R}^{n_z \times n_z}$, and $B \in \mathbb{R}^{n_z \times n_w}$. Finally, the linearised system is explicitly described as:

$$\begin{bmatrix} \dot{x} \\ \ddot{x} \\ \dot{y} \\ \ddot{y} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x \\ \dot{x} \\ y \\ \dot{y} \end{bmatrix} + \begin{bmatrix} 0 & 0 \\ 1 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{y} \end{bmatrix}. \tag{3.7}$$

3.2.2 Towards system controllability

Once the system has been linearised, the next step is to determine the set of control inputs $w = [\ddot{x},\ \ddot{y}]$ which will evolve the system from an initial state z to a desired state $r = [x_r,\ \dot{x}_r,\ y_r,\ \dot{y}_r]$, where $r \in \mathbb{R}^{n_r} \subset \mathcal{X}$ and $n_r = n_z$. Given the system linearisation, a proportional derivative (PD) controller can be defined as:

$$\begin{aligned} w_1 &= k_{p_x}(x_r - x) + k_{d_x}(\dot{x}_r - \dot{x}), \\ w_2 &= k_{p_y}(y_r - y) + k_{d_y}(\dot{y}_r - \dot{y}), \end{aligned} \tag{3.8}$$

where k_{p_i} and k_{d_i} are the proportional and derivative gains, respectively, for the x-axis and the y-axis. Therefore, the equivalent closed-loop system can be formulated as:

$$\begin{aligned} \dot{z} &= Cz + Dr, \\ C &= A - BK, \\ D &= BK, \end{aligned} \tag{3.9}$$

where $C \in \mathbb{R}^{n_z \times n_z}$, $D \in \mathbb{R}^{n_z \times n_r}$, and the matrix K contains the gains for the PD controller in the form:

$$K = \begin{bmatrix} k_{p_x} & k_{d_x} & 0 & 0 \\ 0 & 0 & k_{p_y} & k_{d_y} \end{bmatrix}. \tag{3.10}$$

At this point, controllability of the initial model in Equation 3.1 has been achieved with a dynamic feedback linearisation combined with a PD controller. Figure 3.3 draws the big picture of all those modifications.

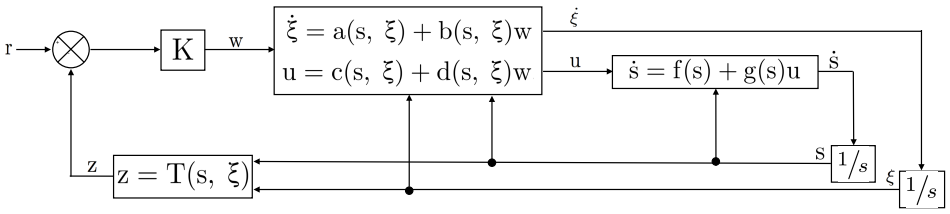


Figure 3.3: Linearisation and control scheme of a nonholonomic system.

3.2.3 Correcting the driftless system assumption

Although the driftless motion model stated in the previous sections is valid as a first approximation, it does not contribute in coping with probabilistic constraints. One alternative is to include a component of noise into the model described in Equation 3.9. In what motion planning concerns, this new component implies that the system is now described in the B-Space, i.e. in an uncertain X-Space.

Before determining the uncertainty associated with the state, it is important to notice that the formulation used until this point has been developed in the continuous time domain. However, in order to implement the linearised and closed-loop model in a real system, it is necessary to discretise such a model for a time step ΔT. Considering the Euler method, this operation can be done for small time steps ΔT by computing:

$$\begin{aligned} z_{t+1} &= C_d z_t + D_d r_t, \\ C_d &= e^{C\Delta T} \approx (I + C\Delta T), \\ D_d &= \int_{\tau=0}^{\Delta T} e^{\tau C} d\tau = C_d^{-1}(C_d - I)D, \end{aligned} \tag{3.11}$$

where C_d and D_d are the discretised versions of C and D, respectively, for a time step ΔT.

At this point, white Gaussian noise $n_t \sim \mathcal{N}(0,\ P_{n_t})$ can be introduced to approximate (i) the original noise of the system, (ii) discrepancies in the linearisation, (iii) inaccuracies of the controller, and (iv) the discretisation error. Then, the system state is now represented by a belief $b_t \sim \mathcal{N}(z_t,\ P_{z_t}) \in$ B-Space, in which the mean and the uncertainty can be independently propagated as:

$$\begin{aligned} z_{t+1} &= C_d z_t + D_d r_t \ \forall\ t \in \mathbb{Z}_{0,N-1}, \\ P_{z_{t+1}} &= C_d P_{z_t} C_d^T + P_{n_t} \ \forall\ t \in \mathbb{Z}_{0,N-1}. \end{aligned} \tag{3.12}$$

It is important to notice that since the introduced noise n_t cannot be mathematically determined, different trials have to be conducted to parametrise P_{n_t} according to how accurate the model in Equation 3.12 is with respect to the real system. Also, in order to simplify the reading of the following chapters, the states and equations are presented in their continuous-time formulation to avoid using subscripts.

Chapter 4

Planning under environment uncertainty

A planner that includes kinodynamic constraints seeks to approximate the system's motion capabilities, thus minimising the risk of collision due to unexpected behaviours when following a calculated path. Additionally, the system's states can be probabilistically represented, as it has been explained in the previous chapter. This allows establishing some minimum safety guarantees when following the desired path. However, this does not verify that the environment awareness is completely accurate. In order to deal with this latter issue, the probabilistic formulation must be also extended to describe the uncertainty of the surroundings knowledge, especially when dealing with initially undiscovered environments. Bearing this in mind, this chapter proposes an alternative to track the system's relative uncertainty with respect to each portion of the map, to finally compute the probability of collision.

4.1 Environment awareness

Environment awareness is essential in autonomous robots. For that, robots gather data to represent their surroundings, however, this information typically suffers from noise. This is especially true in underwater robotics, where the perception of the surroundings is generally done with acoustic sensors. Moreover, more noise is likely to be introduced when projecting the acquired data to the world frame, because those transformations rely on the navigation, which might have its own associated estimation error. In order to enhance the environment awareness, different approaches have been proposed to filter data outliers and to probabilistically fuse different sources of environment information. However, some of those algorithms tend to be unsuitable for applications with real-time computation constraints, especially in robots with limited computing power.

This work does not only seek to address the noise from the input data, but also to build a volumetric grid-based map in which each voxel can adopt any of the following states (Figure 4.1): occupied (red), occluded (green) or free (blue). Considering that the environment is initially undiscovered, the map is initialised as an empty set of voxels; in fact, undiscovered areas of the environment (grey) are not included in the map. Then, those voxels that are observed by the

robot's perception sensors are either marked as free or occupied according to their probability of being occupied. Moreover, since zones that are likely to be occupied are of interest, the environment awareness considers occluded regions. These latter correspond to voxels located behind an occupied voxel within a predefined range.

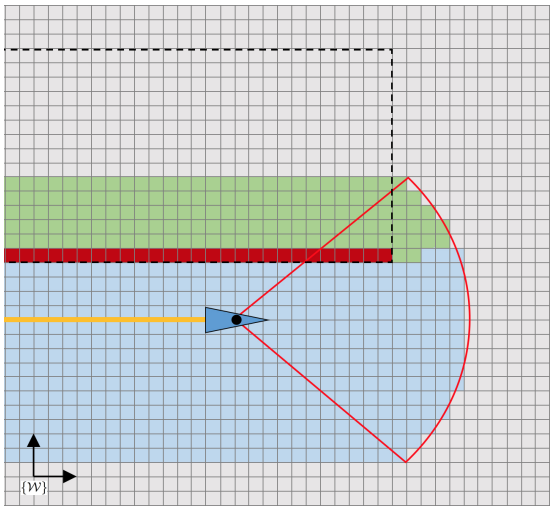


Figure 4.1: System discovering an unknown environment with a fixed range sensor. Free, occluded and occupied voxels are marked as blue, green and red, respectively. The undiscovered environment is indicated in grey.

Each label is associated with a particular probability of being occupied. Free voxels are set to $p_{free} = 0$, occluded voxels with $p_{occluded} = 0.5$ and occupied voxels with $p_{occupied} = 1$. This environment awareness setup has a great advantage when computing the probability of collision with the kernel-based approach proposed at the end of this chapter, because (i) not including the unknown space and setting $p_{free} = 0$ speeds up the computation, (ii) giving some weight to the occluded space let us consider zones that are likely to be occupied, and (iii) rounding the $p_{occupied}$ coming from the sensor fusion process up guarantees steady calculations.

4.2 Relative uncertainty with local maps

Incrementally discovering the surroundings requires having a consistent representation of the entire environment, which is not a trivial problem. Taking into account that this challenge goes beyond the scope of this project, the presented approach considers as an alternative to represent the environment by a sequence of independent local stochastic maps [57], [58], [59]. A local map $\mathcal{LM}$ is a locally consistent portion of the environment awareness, the voxels of which have bounded uncertainty with respect to the reference frame $\{\mathcal{LM}\}$. Because of that, sensor fusion can be performed within each $\mathcal{LM}_i$.

Figure 4.2 depicts an example of the map's incremental awareness. At $t = 0$ the knowledge of the environment is null (Figure 4.2a). As soon as the robot starts mapping (Figure 4.2b), a local map $\mathcal{LM}_1$ is initialised. This means that all the voxels that lie within this map, as well as their corresponding beliefs, are referred to the corresponding reference frame $\{\mathcal{LM}_1\}$. At some point,

because of the inherent uncertainty in the system's dynamics, the uncertainty gets larger than a predefined value. This means that the perception of the environment is considered to be too uncertain with respect $\{\mathcal{LM}_1\}$ to keep performing sensor fusion. Therefore, no more information should be incorporated into $\mathcal{LM}_1$ (Figure 4.2c), so a new independent local map is started $\mathcal{LM}_2$ (Figure 4.2d).

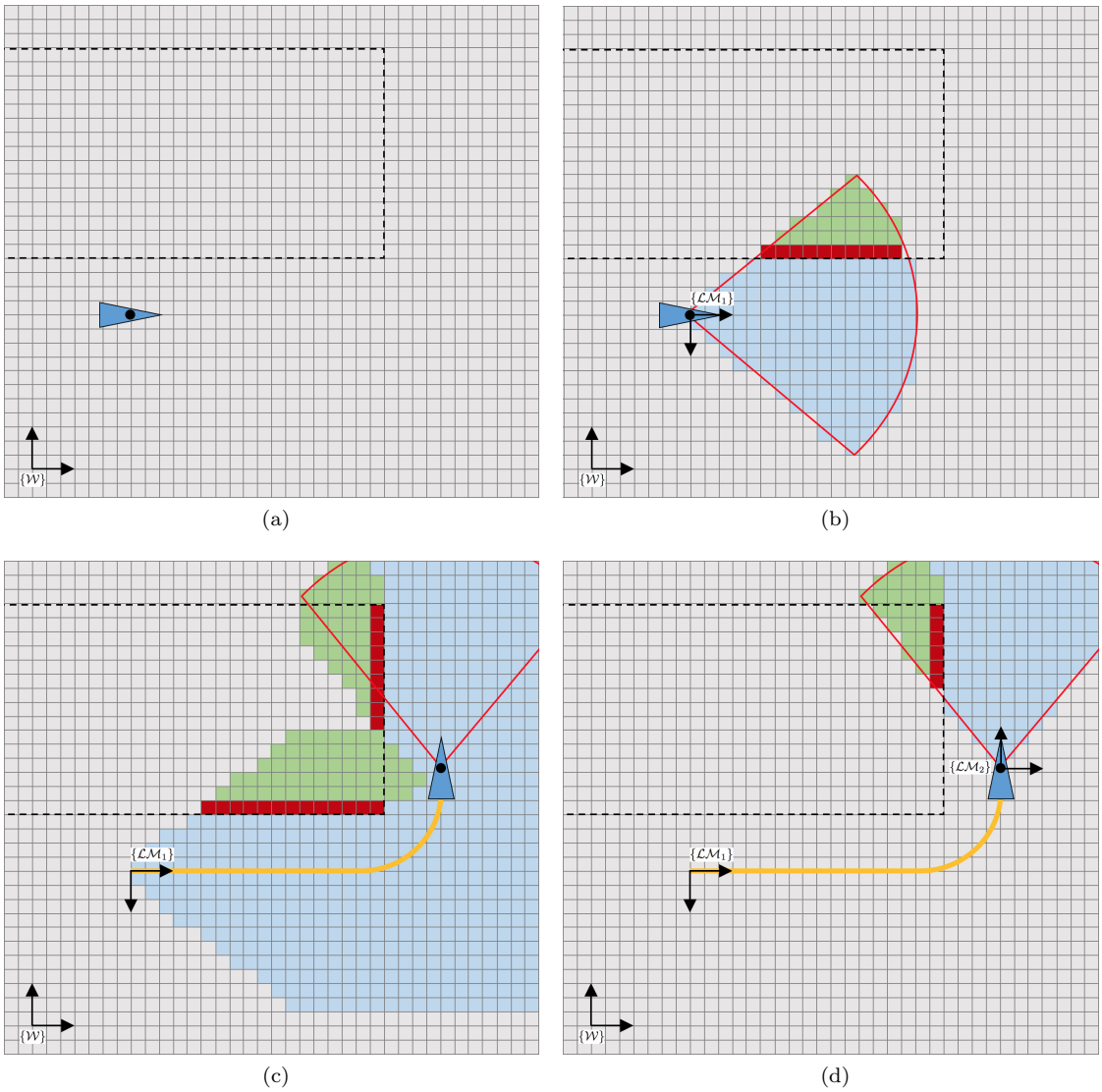


Figure 4.2: Incremental mapping with local maps. (a) Initial environment awareness. (b) Beginning of $\mathcal{LM}_1$. (c) End of $\mathcal{LM}_1$. (d) Beginning of $\mathcal{LM}_2$.

Because of the imprecise modelling of the dynamics, as the system moves, its location becomes more uncertain with respect to each $\mathcal{LM}_\mathrm{i}$, or in other words, more uncertain the location of each $\mathcal{LM}_\mathrm{i}$ is with respect to the local frame. In order to distinguish this source of uncertainty from the one associated with the motion planning problem, this work calls the former one environment uncertainty. To numerically determine this relative uncertainty, the system's belief transformation

between each pair of consecutive reference frames is stored as it incrementally builds $\mathcal{M}$. Then, the environment uncertainty between the system and a specific $\mathcal{LM}_{\mathrm{i}}$ can be determined using back propagation.

Both the motion and the environment uncertainties can be jointly considered in the motion planning problem formulated in Section 3.1. Let us assume that the planner roots a tree in the $\mathcal{LM}_{\mathrm{n}}$, which is the last and current robot's local map. When expanding the tree, the probability of collision associated with a belief (node) b in a specific $\mathcal{LM}_{\mathrm{i}}$ needs to be computed, for what their relative uncertainty has to be determined. This involves considering two sources of uncertainty:

- The associated to the system's motion that accounts for the uncertainty evolution from $\mathcal{LM}_{\mathrm{n}}$ to b along the tree (motion uncertainty).
- The associated to the environment that refers to the uncertainty evolution from $\mathcal{LM}_{\mathrm{i}}$ to $\mathcal{LM}_{\mathrm{n}}$ going through all $\mathcal{LM}_{\mathrm{j}}$, where $\mathrm{j} = \mathrm{i}, ..., \mathrm{n}$ (environment uncertainty).

4.3 Collision checking

Probabilistically checking the validity of a state aims to guarantee a minimum probability $\mathrm{p}_{\mathrm{safe}}$ of being safe. In other words, the probability of a state b being in collision with any obstacle in the $\mathcal{M}$ has to be below $\Delta\mathrm{P} \equiv 1 - \mathrm{p}_{\mathrm{safe}}$, which can be expressed as:

$$\Delta\mathrm{P} < \mathrm{p}_{\mathrm{collision}}(\mathrm{b},\ \mathcal{M}). \tag{4.1}$$

In this work, $\mathcal{M}$ is composed by different local maps, each of them having a particular relative uncertainty with the state b. Thus, the state b is probabilistic valid if

$$\Delta\mathrm{P} < \sum_{\mathrm{i}=1}^{\mathrm{n}} \mathrm{p}_{\mathrm{collision}}(\mathrm{b},\ \mathcal{LM}_{\mathrm{i}}). \tag{4.2}$$

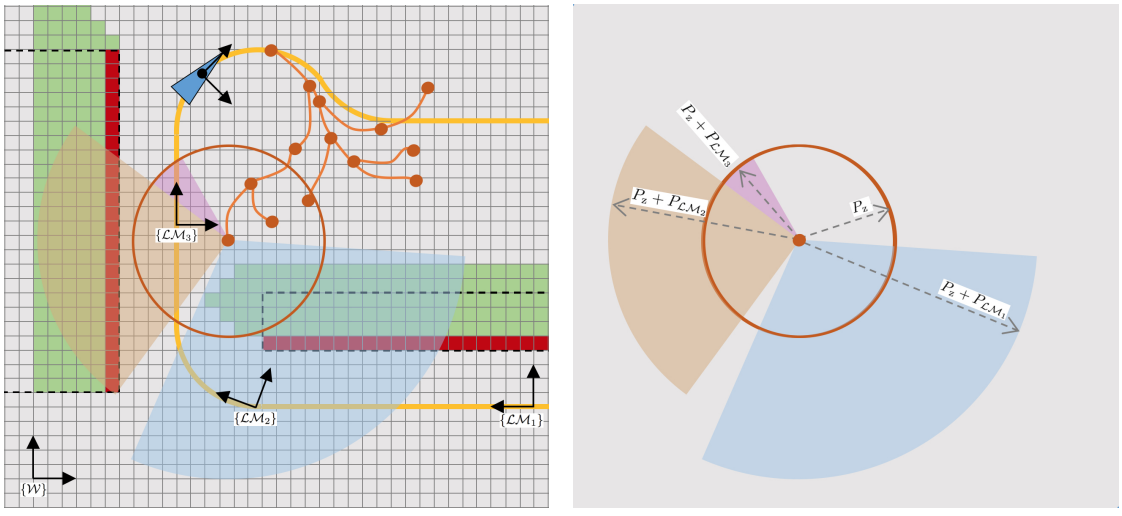
To operate $\mathrm{p}_{\mathrm{collision}}(\mathrm{b},\ \mathcal{LM}_{\mathrm{i}})$, the formulation proposed in [43] and in [44] is computationally forbidden because all the obstacles in each $\mathcal{LM}_{\mathrm{i}}$ must be independently convexified. Moreover, obtaining such a representation is (i) challenging without a prior knowledge of the number of obstacles in the environment, (ii) imprecise with the presence of noise in the map and (iii) computationally expensive for complex environments. To overcome these issues and leveraging the discrete representation of the environment, this work proposes the kernel-based approach detailed in Algorithm 2, which allows validating b according to Equation 4.2.

After extracting the mean z and the motion uncertainty P_{z} of the belief (node) b (line 1), the probability of collision is computed by checking all the local maps in $\mathcal{M}$. Before that, but, the motion uncertainty of the system and the environment uncertainty of the $\mathcal{LM}_{\mathrm{i}}$ have to be merged, obtaining P, which represents the uncertainty of b with respect to the $\{\mathcal{LM}_{\mathrm{i}}\}$ (line 4). Then, only the occluded and occupied voxels in $\mathcal{LM}_{\mathrm{i}}$ (line 5) within a radius of 3.03σ from z (line 6) are weighted by the value at position ($\mathcal{LM}_{\mathrm{i}}$ - z) of a $\mathcal{N}(0,\ \mathrm{P})$ (line 8). At each iteration, the bound in Equation 4.2 is checked to immediately report the invalidy of the state, if necessary (line 9).

Algorithm 2: PROBABILISTIC_VALIDITY_CHECKER(b, $\mathcal{M}$, ΔP)

```
(z, P_z) ← b
p_collision = 0
for all LM in M do
    P, σ ← MERGE(P_z, ENVIRONMENT_UNCERTAINTY(LM, z))
    for all voxels in LM do
        if DISTANCE(voxel, z) > (3.03σ) then
            next
        p_collision += GET_OCCUPANCY(voxel) · GET_GAUSSIAN_AT(voxel - z, P)
        if p_collision ≥ ΔP then
            return False
return True
```

The essence of this algorithm is illustrated in Figure 4.3. Since b has a different uncertainty P with respect to each $\mathcal{LM}_i$, the probability of collision between those elements has to be individually evaluated. For that purpose, a kernel that emulates $\mathcal{N}(0,\ P)$, with a resolution equal to the one of the map, is used. In order to speed up this process, (i) the kernel is bounded at 3.03σ and (ii) neither free nor unseen voxels are considered, which is known as opportunistic collision checking [37]. Finally, the probability of collision with the entire map $\mathcal{M}$ can be obtained by adding up the probability of b colliding with each $\mathcal{LM}_i$. It is important to notice that since the computation of each $p_{collision}(b,\ \mathcal{LM}_i)$ is bounded at 3.03σ, the maximum probability of collision can be 0.99. Thus, the predefined threshold ΔP is correspondingly scaled as 0.99ΔP.



(a) (b)

Figure 4.3: Methodology for computing the probability of collision. (a) The robot has built a map with 3 local maps. From the third one, a tree is expanded and a certain node is checked. (b) Kernel setup for computing the probability of collision with each local map.

Two approaches have been considered to perform the element-by-element multiplication between the kernel and the environment: (i) exploring the kernel row-column wise, and (ii) conducting a spiral-like traversing, i.e. from the centre to the outer part of the kernel. As reported in Section 6.2.1, the latter approach permits considering the highest values of the kernel first, being more efficient than the row-column wise exploration. However, the computational time of the spiral-like traversing strategy is still slightly higher than methods in [43] and [44] but in favour of (i) dealing with non-convex environments, (ii) being suitable for unknown environments and (iii) explicitly computing the probability of collision without conservatism.

Chapter 5

Framework for mapping and planning under uncertainty

In order to efficiently plan paths in undiscovered environments under motion and probabilistic constraints, the methods developed in Chapter 3 and Chapter 4 have to be jointly considered. In doing so, this chapter presents a framework that integrates such approaches as schematised in Figure 5.1. Furthermore, the framework includes a high-level trajectory tracking algorithm to guide the robot through the calculated path so all its properties are kept.

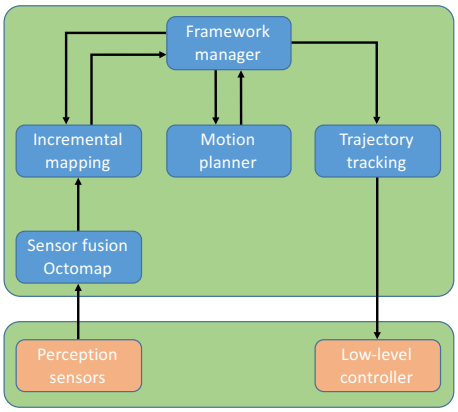


Figure 5.1: Framework for incrementally mapping and planning under motion and uncertainty constraints.

The proposed framework has been fully implemented in C++ over the robot operating system (ROS) [60]. This allows separating the robot's software components such as planners, mappers, controllers, driver, etc., into functional blocks that are called nodes. Such nodes are allowed to communicate each other by using a publish-subscribe protocol. This modular structure makes the framework to be easily integrated within different robotic platforms.

5.1 Environment representation

To represent the environment accordingly to the specifications stated in Section 4.1, a twofold strategy is considered. First, sensor fusion in each $\mathcal{LM}_{\mathrm{i}}$ is done with OctoMaps [61]. Secondly, an additional layer is defined to both include the occluded regions in each $\mathcal{LM}_{\mathrm{i}}$ and to manage the transformations between local maps.

OctoMap efficiently addresses the sensor fusion problem. Data acquired by the sensors is fused into a robust estimation of the true state of the environment. This is done by probabilistically considering the uncertainty of the measurements. The resulting estimations are integrated into a 3D occupancy grid map, which has a resolution that can be freely adjusted. Such a map is stored in an octree data structure, which provides fast access time while, at the same time, optimising the use of memory. A more detailed presentation of the method can be found in [61].

The additional layer included on top of the sensor fusion stage leverages the information provided by OctoMap [61], i.e. the voxel's probability of being occupied, to determine if they are either free or occupied space. This decision is made by thresholding such a probability at 0.5. Then, the environment awareness is expanded with occluded regions, which correspond to those unseen voxels located behind the occupied space. In order to limit the size of the occluded regions, only voxels within a certain radius from the robot's position are considered. In this work, the radius corresponds to the sensor's maximum range.

Apart from that, the upper layer also handles the creation and connection of the local maps. It checks the system's uncertainty evolution to decide if keep performing sensor fusion leads to a consistent local map. As soon as the uncertainty of the system exceeds a predefined value, the additional layer starts a new local map, the reference frame of which is set to the vehicle's state at the end of the previous local map.

5.2 Motion planner

When using sampling-based methods to address the motion planning problem, having a steering function is essential for obtaining optimal paths. However, many systems that operate under kinodynamic constraints do not have such a function. Given this limitation in many systems, Li et al. proposed the SST algorithm to dodge the need of a steering function for getting asymptotically near-optimal paths [35]. As detailed in Algorithm 3, such a planner is a sampling-based method that uses a tree expansion methodology, similar to the one explained in Section 3.1, but with a particular *selection* procedure that is called forward propagation. This work uses the implementation of such algorithm available in the open motion planning library (OMPL) [62].

The SST planner divides the nodes in $\mathcal{T}$ into two subsets, $\mathbb{V}_{\mathrm{active}}$ and $\mathbb{V}_{\mathrm{inactive}}$. The former one contains those nodes that have the best path cost from the root $\mathrm{b}_{\mathrm{init}}$ in a local neighbourhood of radius δ_{S}. $\mathbb{V}_{\mathrm{inactive}}$, on the other hand, stores the nodes that do not have the best local cost, but do have children with good path costs in their local neighbourhood. Thus, when conducting a search along the tree, only nodes in $\mathbb{V}_{\mathrm{active}}$ have to be considered. There exist different optimisation criteria to determine the best cost of a path such as length, time, energy, etc; in this work, the path cost is evaluated with its length.

Algorithm 3: SST($(z_{init},\ P_{z_{init}})$, planning_time, B-Space, U-Space, δ_{BN}, δ_S, $\mathcal{M}$, ΔP)

1 $\mathbb{V}_{active} \leftarrow \{(z_{init},\ P_{z_{init}})\}$
2 $\mathbb{V}_{inactive} \leftarrow \varnothing$
3 $\mathcal{T} = \{\mathbb{V} \leftarrow \mathbb{V}_{active} \cup \mathbb{V}_{inactive}, \mathbb{E} \leftarrow \varnothing\}$
4 **while** *not* STOP_CONDITION(planning_time) **do**
5 $z_{rand} \leftarrow$ RANDOM_STATE(B-Space)
6 $(z_{near}, P_{z_{near}}) \leftarrow$ GET_NEAREST_NEIGHBOURS(z_{rand}, $\mathbb{V}_{active}$, δ_{BN})
7 **if** $(z_{near}, P_{z_{near}}) = \varnothing$ **then**
8 $(z_{selected}, P_{z_{selected}}) \leftarrow$ NEAREST_NEIGHBOUR(z_{rand}, $\mathbb{V}_{active}$)
9 **else**
10 $(z_{selected}, P_{z_{selected}}) \leftarrow \text{argmin}_{z \in (z_{near}, P_{z_{near}})}$ COST(z)
11 $T_{prop} \leftarrow$ RANDOM_CONTROL_DURATION()
12 $(z_{new}, P_{z_{new}}), u_{new} \leftarrow$ PROPAGATE_BELIEF($(z_{selected}, P_{z_{selected}})$, z_{rand}, T_{prop})
13 **if** VALID($\overline{(z_{selected},\ P_{z_{selected}}) \rightarrow (z_{new},\ P_{z_{new}})}$, U-Space, $\mathcal{M}$, ΔP) **then**
14 **if** IS_NODE_LOCALLY_THE_BEST($(z_{new},\ P_{z_{new}})$, $\mathbb{V}_{active}$, δ_S) **then**
15 $\mathbb{V}_{active} \leftarrow \mathbb{V}_{active} \cup \{(z_{new},\ P_{z_{new}})\}$
16 $\mathbb{E} \leftarrow \mathbb{E} \cup \{u_{new}\}$
17 PRUNE_TREE($(z_{new},\ P_{z_{new}})$, $\mathbb{V}_{active}$, $\mathbb{V}_{inactive}$, $\mathbb{E}$)
18 **return** $\mathcal{T}(\mathbb{V},\ \mathbb{E})$

The tree expansion in the SST is presented in Algorithm 3. A tree $\mathcal{T}$ is rooted at b_{init} in the B-Space (line 3). Then, the tree is expanded with the typical iterative *selection-propagation-validation* procedure until a stop condition is met. In this work, the planner stops after a specific amount of time specified in planning_period (line 4). The *selection* process starts by uniformly sampling the B-Space to obtain z_{rand} (line 5) which, at the same time, is used to select, among the nodes in a local neighbourhood of radius δ_{BN}, the node $b_{selected}$ with lowest cost from the root (lines 6 to 10). $b_{selected}$ establishes the node from which the tree is expanded during the *propagation* routine. In this work, a feasible control input u_{new} that grows the tree towards z_{rand} is obtained by applying the linearised closed-loop system formulated in Chapter 3. This control law expands the tree for a period of time T_{prop}, thus generating b_{new} (line 12). Then, the *validation*

Algorithm 4: VALID(segment, U-Space, $\mathcal{M}$, ΔP)

1 **for** *all* motions *in* segment **do**
2 b, u $\leftarrow$ motion
3 **if** u $\notin$ U-Space **then**
4 **return** False
5 **if** *not* PROBABILISTIC_VALIDITY_CHECKER(b, $\mathcal{M}$, ΔP) **then**
6 **return** False
7 **return** True

step firstly determines whether the segment $\overline{b_{selected} \rightarrow b_{new}}$ is feasible and probabilistically valid according to Algorithm 4 (line 13). Furthermore, if b_{new} is locally the best within a radius δ_S (line 14), it is added in the $\mathbb{V}_{active}$ subsection and u_{new} in $\mathbb{E}$. Finally, the tree is traversed to check that its subsections ($\mathbb{V}_{active}$ and $\mathbb{V}_{inactive}$) maintain the aforementioned properties (line 17).

Note that the *validation* routine is done over a complete segment, which includes the involved set of motions. At the same time, each motion is composed by a belief and the corresponding control input. Thus, as detailed in Algorithm 4, to guarantee that a segment is feasible and safe, all its control inputs have to belong to the U-Space (line 3) and all its beliefs need to be probabilistically checked (line 5).

5.3 Path tracking controller

Once a path to the goal has been found, the robot must try to follow it precisely in order to keep its feasibility and safety properties. To do so, the model presented in Section 3.2 has been used altogether with some features of the line-of-sight (LoS) trajectory tracking controller [63]. The combination of those parts leads to the proposed trajectory tracking controller detailed in Algorithm 5.

Algorithm 5: PATH_CONTROLLER(path, R)

```
1  z ← GET_NAVIGATION()
2  elapsed_time ← 0.1

3  while not END_ACHIEVED(z, path) do
4      elapsed_time ← last_time - current_time

5      if elapsed_time ≥ 0.1 then
6          z ← GET_NAVIGATION()
7          r ← COMPUTE_LOOKAHEAD(z, R, path)
8          [v, ϑ̇] ← GET_CONTROL_INPUTS(z, r, elapsed_time)

9          ROBOT_CONTROL_ARCHITECTURE(v, ϑ̇)

10         last_time ← current_time
```

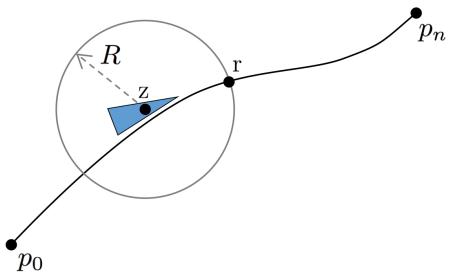


Figure 5.2: Trajectory tracking controller.

The trajectory tracking algorithm iterates until the end of the path p_n is achieved (line 3 to 5). At each iteration, the system's current status z is the centre of a circle with radius R. As shown in Figure 5.2, this circle intersects the path at two points, from which the one closest to the end of the path is selected as lookahead or reference r (line 7). Then, the controller inputs correspond to z and r in order to address the geometric and dynamic tasks involved in any manoeuvring problem, i.e. driving the system to r with a set of feasible velocities (line 8). Finally, the trajectory tracking controller's outputs [v, $\dot{\vartheta}$] are sent to the vehicle's low-level controller (line 9).

5.4 Framework manager

To integrate and coordinate all the previous modules in an online approach, the framework contains a manager that works as a coordinator. Algorithm 6 explains the pipeline execution of this manager and its interaction with the other functional modules.

Algorithm 6: MANAGER(z_{goal}, tolerance, ΔT, planning_time, δ_{BN}, δ_S, main_period, R)

```
1  M ← ∅
2  B-Space ← ∅
3  old_path ← ∅
4  elapsed_time ← main_period

5  while not GOAL_ACHIEVED(z_goal, tolerance) do
6      elapsed_time ← last_time - current_time

7      if elapsed_time ≥ main_period then
8          M ← UPDATE_MAP()

9          if not VALID(old_path, U-Space, M, ΔP) then
10             PATH_CONTROLLER(old_path, R)

11         vs ← GET_NAVIGATION()
12         (z_init, P_z_init) ← PLACE_NEW_ROOT(old_path, vs)
13         T ← SST((z_init, P_z_init), planning_time, B-Space, U-Space, δ_BN, δ_S, M, ΔP)

14         new_path ← GET_SOLUTION(T)
15         if BEST_PATH(old_path, new_path) = new_path then
16             PATH_CONTROLLER(new_path, R)
17             old_path ← new_path

18         last_time ← current_time
```

The framework executes a mapping-planning sequence each period of time main_period until a given goal region, i.e. a goal state z_{goal} with some tolerance, is achieved (line 5 to 7). Defining a goal region instead of a single state is essential when an exact final state might not be reachable, which is especially true when planning under differential constraints. First, the algorithm requests an update of the environment awareness $\mathcal{M}$ (line 8). Then, the previous solution old_path, if any, is probabilistically checked for collision according to the new map $\mathcal{M}$. If the previous path is no

longer valid, the trajectory tracking algorithm detailed in Section 5.3 is updated with the valid segment of the path (line 9 and 10). This approach prevents the vehicle stopping every time that a path gets invalidated.

Once updated the environment awareness, the planner is set to find a path to the goal. First, but, in order to avoid abrupt changes in the vehicle's direction of motion when a replanning manoeuvre is required, the tree root b_{init} is placed on a predicted position along the current valid path (line 12). In that way, initial states of the solution path are prevented from being behind the robot's predicted position in planning_time seconds, thus avoiding undesired manoeuvres (Fig 5.3). After this, the SST algorithm grows a new tree in the B-Space for a specific planning_time (line 13).

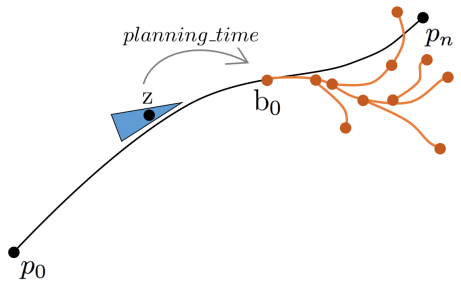


Figure 5.3: Rooting the new tree in the predicted robot's location after a planning period.

After completing the allowed planning_time, the motion planner returns a nearly-optimal path new_path (line 14). Such a solution is only accepted if it is better than the previous one. In this work, the new_path is considered to be better than the old_path if any of the following conditions is met (line 15):

- New_path has a lower cost than the old_path when this is still valid.
- New_path goes near the invalid old_path.

The former condition takes the path with less cost to the goal when the current one is still valid. If the path in execution is not valid, only those new paths that pass close to the state that invalidated the previous solution path are considered. This heuristic seeks to keep a similar trajectory by filtering those paths which reach the goal through a completely different part of the environment. It is important to note that the planner might return a path through unexplored areas, which are easier to explore with a sampling-based approach, if the planning_time is not necessary to converge to a nearly-optimal solution.

Finally, if the new_path is accepted, it is sent to the trajectory tracking algorithm and recorded as old_path (line 16 and 17).

Chapter 6

Results and evaluation

This chapter firstly details the experimental setup, which includes the motion constrained vehicle, the sensors configuration, and the test scenarios. Then, it presents and discusses the results obtained in both simulated and real-world scenarios.

6.1 Experimental platform

The work developed in this dissertation has been designed for any marine vehicle behaving as a car-like system. In order to evaluate the proposed framework, an experimental validation has been carried out with the Sparus II AUV, which is a nonholonomic vehicle. Next, the characteristics of the vehicle, the exteroceptive sensor and the considered environments are detailed.

6.1.1 Sparus II AUV

Sparus II is a torpedo-shaped vehicle with an adaptable payload area, which allows using different sensors configurations according to the specific mission requirements. It is characterised by efficient hydrodynamics for long autonomy and the capability of descending up to 200 metres. Although the vehicle moves in a 3D workspace, which means that it would require a six-dimensional (6D) C-Space, it can only be actuated in surge, heave, and yaw. This kind of constrained motion makes the Sparus II an ideal platform to validate the proposed approach. Furthermore, the Sparus II is

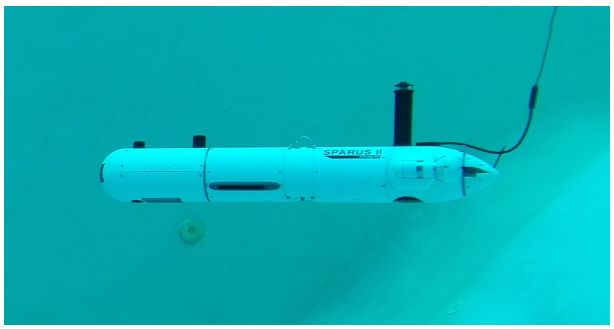


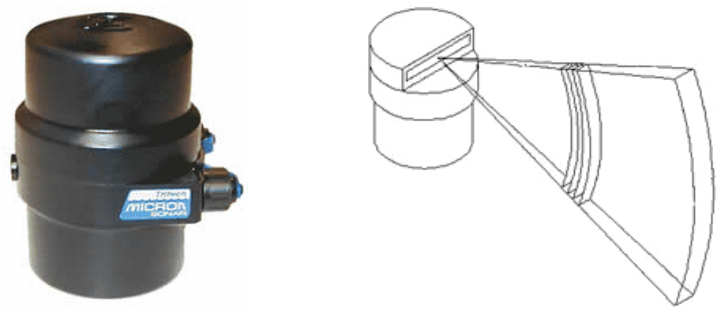
Figure 6.1: Sparus II AUV.

equipped with a set of navigation sensors that includes a global positioning system (GPS), inertial measurement unit (IMU), a pressure sensor and a doppler velocity logger (DVL), which are used to estimate the state of the vehicle.

From the software perspective, the Sparus II AUV is controlled by the component oriented layer-based architecture for autonomy (COLA2) [64], which is a control architecture fully implemented over ROS. This allows to easily integrate new middle and high-level components such as the mapping-planning framework proposed in this work.

6.1.2 Exteroceptive sensor

For conducting the experiments, the Sparus II AUV has been equipped with the mechanical scanned imaging sonar (MSIS) depicted in Figure 6.2a. This sensor has been mounted on the upper front part of payload area. This MSIS is called Micron, from the Tritech company, and it is specially designed for underwater domains, so ROVs and AUVs can gather information from the surrounding environment. This sonar scans the surroundings by rotating a fan-shaped sonar beam through a series of small angle steps. As illustrated in Figure 6.2b, the fan-shaped beam has a vertical aperture angle of 40° and a narrow horizontal aperture of 3°.



(a) (b)

Figure 6.2: Micron from Tritech. (a) Physical appearance. Image credit: www.tritech.co.uk/. (b) Fan-shaped beam. Image credit: www.seaviewsystems.com/.

The sensor can be set to cover variable length sectors, from a few degrees to full 360° scans. For this experiments, it has been configured to scan from -60° to 60° with respect to the vehicle's direction of motion. Even though a wider field of view (FoV) to perceive more information from the environment could be desirable, it would compromise the scanning time, i.e. the amount of time required for doing a full scan. Regarding the range of the beam, it has been set at 10 metres to avoid false-positive detections caused by the high vertical aperture of the sonar.

6.1.3 Test scenarios

A total of three different scenarios have been used to test and evaluate the proposed framework: *harbour*, *blocks* and *canon*. As illustrated in Fig 6.3, all those scenarios are inspired by real areas located in Sant Feliu de Guíxols, Catalonia, Spain. Regarding the scenario called *harbour* (orange), it is an area inside the port which stays protected from strong waves and currents. Contrarily, the

blocks (green) is a breakwater structure in the outer part of the port, and the *canon* (red) is a long natural passage. Both of them areas are exposed to the open sea conditions.

Figure 6.3: Real-world scenarios in Sant Feliu de Guíxols used to test the proposal: *harbour* (orange), *blocks* (green) and *canon* (red). Image credit: Map data ©2017 Google.

All those areas are simulated in the underwater simulator (UWSim), a simulator which is completely integrated with the Sparus II architecture COLA2. This allows testing the algorithms before conducting the experiments in the real scenarios. Figure 6.4 shows Sparus II close to the seabed of a flat region.

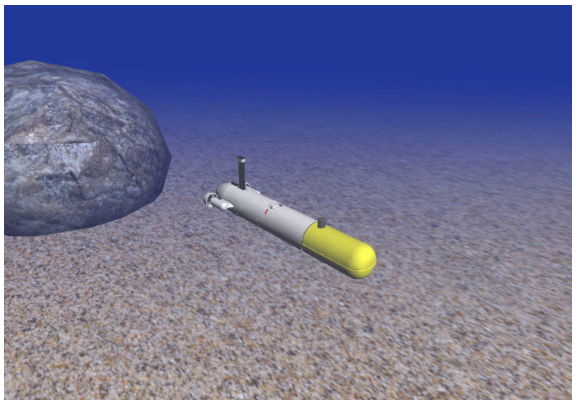


Figure 6.4: Sparus II AUV in the UWSim.

6.2 Experiments in simulated scenarios

Performing experiments in simulated scenarios let us show and evaluate some theoretical aspects of the framework, such as the collision checking method proposed in this framework. Moreover, it lets testing the whole algorithm in controlled environments.

6.2.1 Collision checking

It is well-known that the most expensive part of any planner is the collision checking routine. Thus, the performance of a planning algorithm can be lessened for a slow collision checking method. Even though this work has proposed a kernel-based approach for computing the probability of collision of a certain state, next is shown that its performance, i.e. trade-off between exploration of the B-Space and computation time, is comparable to the state-of-the-art methods which assume known and convexified environments.

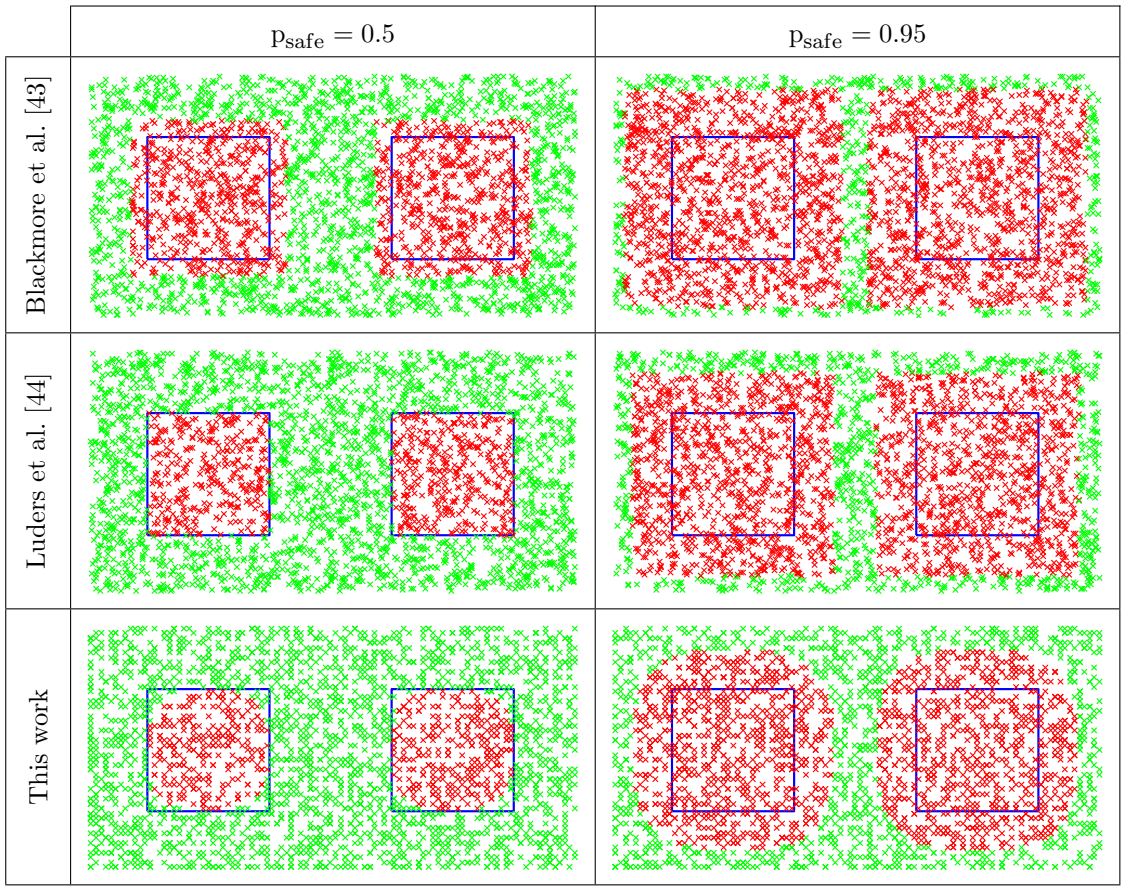


Table 6.1: Workspace exploration according to different probabilistic collision checking methods. The obstacles are represented by blue lines, while the valid and invalid states are marked in green and red, respectively.

First, the map exploration is analysed in comparison with the original proposal of the chance constraints formulation [43] and its less conservationist version [44]. For that purpose, a 2D

environment with two blocks is set, which are defined by the blue lines in all the images in Table 6.1. Then, each method is asked to validate 3000 states uniformly sampled over the B-Space but with fixed uncertainty. The experiment is performed twice but with different thresholds, $p_{safe} = 0.5$ and $p_{safe} = 0.95$.

The obtained results are shown in Table 6.1. The original proposal of the chance contraints [43] is so conservationist that rejects many states which are actually probabilistically correct. This behaviour leads to a shrinkage of the real $\mathcal{B}_{free}$ inside the passage, what would difficult any planner to find a valid path through the blocks. The second row of images proves the method proposed in [44] to be less conservationist than the original formulation. Finally, because of the complete lack of conservatism, the kernel-based approach proposed in this work only rejects the states that are truly in collision according to the selected p_{safe}. Note in the bottom-left image of Table 6.1 that states on the borders of an obstacle are mathematically valid, but this might not be desirable for real-robotic applications. Thus, a heuristic could be set to address this shortcoming of the kernel-based approach.

Secondly, in order to compare the computation time of those methods, they have been executed 10 times each in the same known environment composed of two blocks. Each trial has run until 12000 free and 12000 occupied samples have been obtained to finally report the statistics in Figure 6.5. Note that in these experiments the uncertainty has not been fixed, thus obtaining a representative idea of the methods' behaviour when they are used in a motion planning algorithm. Note that those results uniquely account for the collision checking routine, so formulations in [43] and [44] would require additional time to convexify the environment.

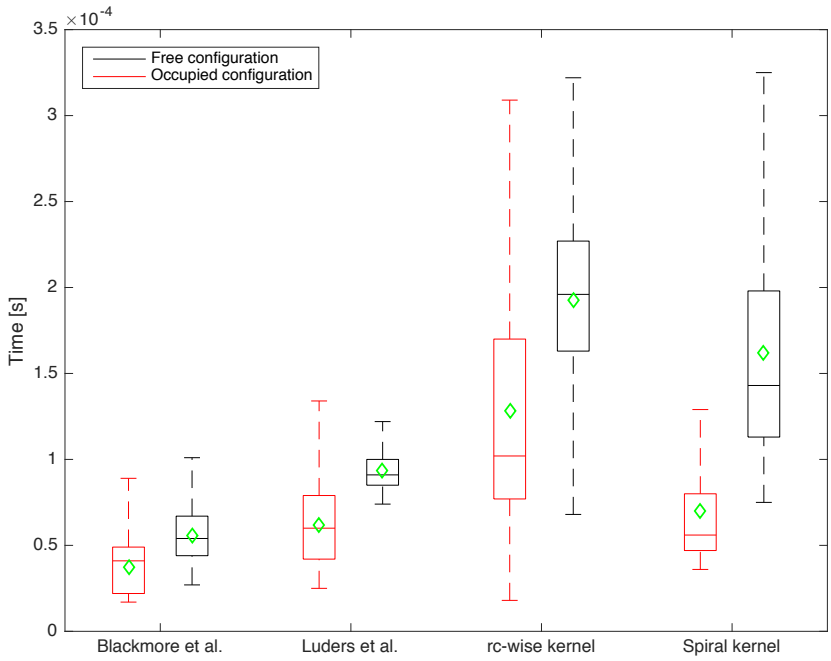


Figure 6.5: Computation time when validating a state with different approaches.

Regarding the method proposed in this work, two different implementations have been tested: checking the kernel row-column wise and in a spiral-like traversing, i.e. from the centre to the

boundaries. This last approach promotes checking the highest values of the Gaussian first, what according to Figure 6.5, turns to be faster than the traditional row-column wise approach.

The most efficient implementation of the proposed collision checking routine, i.e. the spiral-like traversing, is almost twice as slow as the chance constrained formulations when validating a state, but it approximately needs the same time to reject it. In general, the methods with a higher level of conservatism are the fastest ones, because the most exhaustively the B-Space is checked, the more time is needed. Moreover, since the proposal needs to traverse the whole kernel, the time needed to validate a state is related to the state's uncertainty.

Results shown in this section suggests that the trade-off between exploration and computation time depends on the type of environment. For the scenarios where obstacles are distant from the others, the original chance constrained formulation might be the best option to rapidly explore a known map. However, if narrow passages are present in the environment, a method such as the one proposed in this work is preferred; its slightly higher computation time is compensated with an exhaustive exploration of the $\mathcal{B}_{\text{free}}$. Thus, when dealing with unknown environments, the last method is favoured to (i) avoid the need of convexifying the environment, which requires extra computation time, and (ii) be prepared for the most challenging environment.

6.2.2 Undiscovered environment

The overall framework has been tested in two simulated environments, the *blocks* and the *canon*. In both cases, the robot is asked to map the environment with a 0.5 metres resolution and to navigate at a constant depth of 1.5 metres with a maximum velocity of 0.35 m/s. The minimum probability of safeness is set to $\text{p}_{\text{safe}} = 0.8$. The rest of parameters are tuned according to the model of the system in the simulator. To monitor the system's behaviour along the different surveys, the ROS visualizer (RViz) is used.

The first single start-to-goal-query experiment is in the simulated *blocks* scenario. As it has been previously introduced, this environment is inspired by a real breakwater structure in Sant Feliu de Guíxols, Catalonia, Spain, which is depicted in Figure 6.6. In order to bound such an environment, the considered portion of the map in the simulator has dimensions of 34 metres length by 51 metres width. This includes the first 4 blocks starting from the right and its corresponding 3 narrow passages.

Figure 6.6: Breakwater structure in Sant Feliu de Guíxols. Image credit: Map data ©2017 Google.

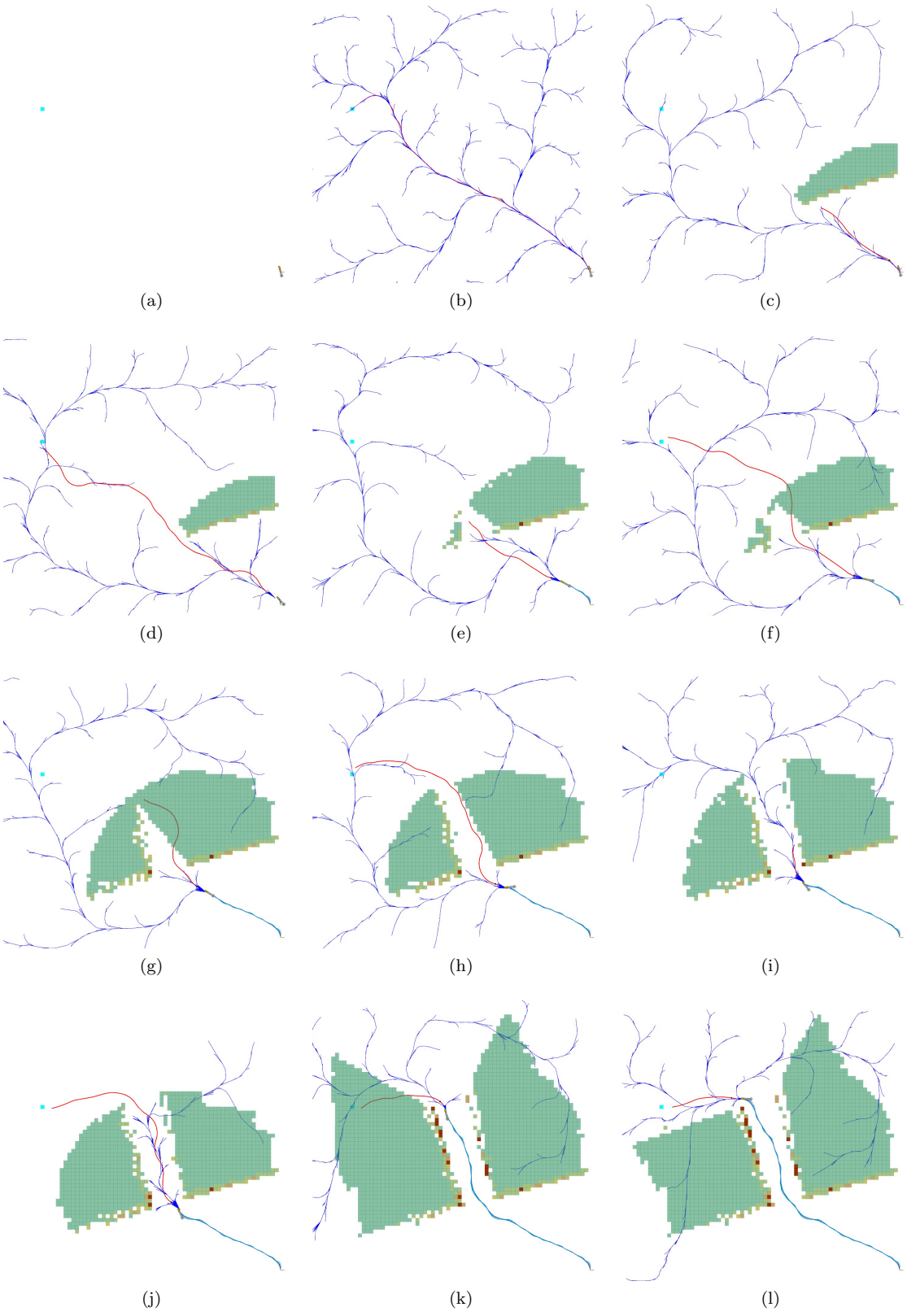
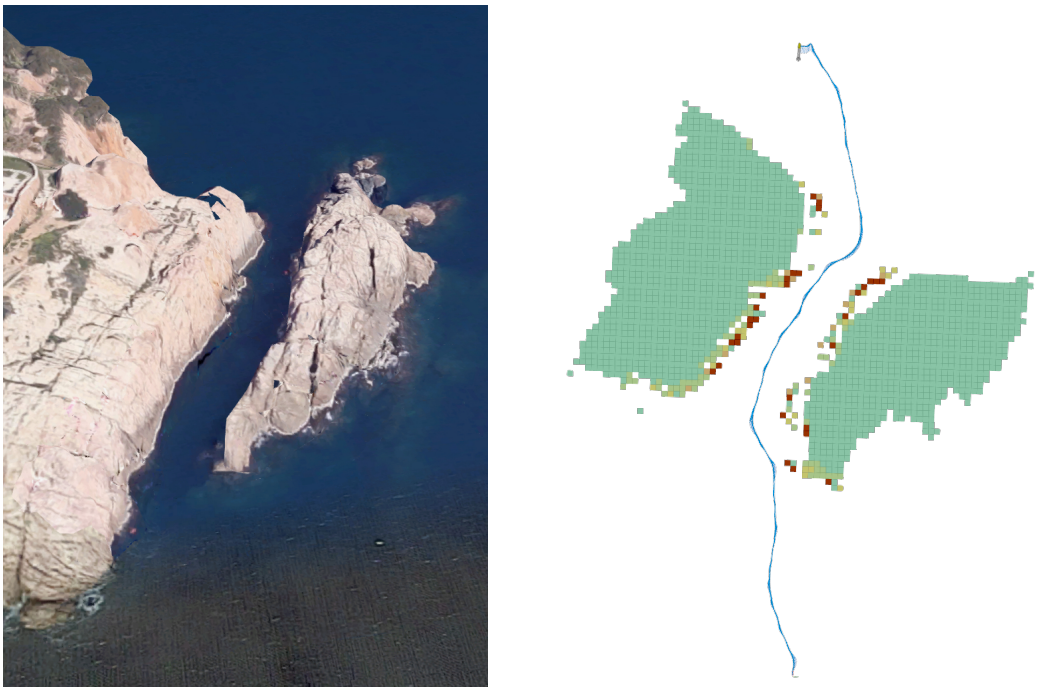


Figure 6.7: Experiments in the simulated *blocks* scenario. Multiple mapping-validating-planning actions make the robot safely get to the goal.

The framework's behaviour during this survey is detailed in Figure 6.7. Initially, the knowledge of the environment is null, the robot's state is at the bottom-right corner and the goal state is on the top-left corner, marked as a blue rectangle (Figure 6.7a). When the framework is initialised, a tree is expanded (blue) and the first solution (red) is almost a straight line towards the goal (Figure 6.7b). The robot starts following the path, and suddenly the current path gets invalidated due to the perceived environment (Figure 6.7c), so the algorithm plans an alternative path which surrounds the obstacle (Figure 6.7d). Then, the framework does an iterative mapping-validation-replanning sequence to find a solution over the continuously updated environment awareness (Figure 6.7e to Figure 6.7j). The solution path illustrated in Figure 6.7j does not get invalidated anymore because it goes through the free space. In fact, it is kept until a path with lower cost is found in Figure 6.7k. Finally, Figure 6.7l shows the 26th and last planned path towards the goal, which leads with an executed trajectory of 45.2 metres length covered in $2'04''$.

The second experiment is conducted in the simulated *canon* scenario. It has been inspired by the natural canon in Sant Feliu de Guíxols, Catalonia, Spain. As illustrated in Figure 6.8a, such a scenario has a narrow passage, the length of which is approximately of 28 metres. Detailing the framework's behaviour in this environment could turn to an extensive sequence of illustrations because of its dimensions (80 metres length by 30 metres width). Thus, Figure 6.8b illustrates the resulting path when performing a single start-to-goal-query in this scenario. After $2'59''$ and 31 replannings, the goal is successfully achieved with a path of 68.4 metres length.



(a) (b)

Figure 6.8: Experiments in the simulated *canon* scenario. (a) 3D view of the natural canon in Sant Feliu de Guíxols. Image credit: Map data ©2017 Google. (b) Environment awareness and path found through the canon.

Figure 6.8b also depicts that a perfect awareness of the environment is not achieved. Some parts of the canon stay unmapped even though the robot could have perceived them, what might be due to either the limited FoV or the scanning time of the sensor. As proposed in future work, these limitations on the perception should also be integrated into the planner to avoid planning paths through those unmapped areas.

6.3 Experiments in real scenarios

The entire framework has also been tested in real scenarios and with the real Sparus II AUV. First, the highest level of the framework, i.e. the trajectory tracking algorithm, has been tunned to secondly conduct a single start-to-goal-query in the *blocks* scenario.

6.3.1 Controller and noise tunning

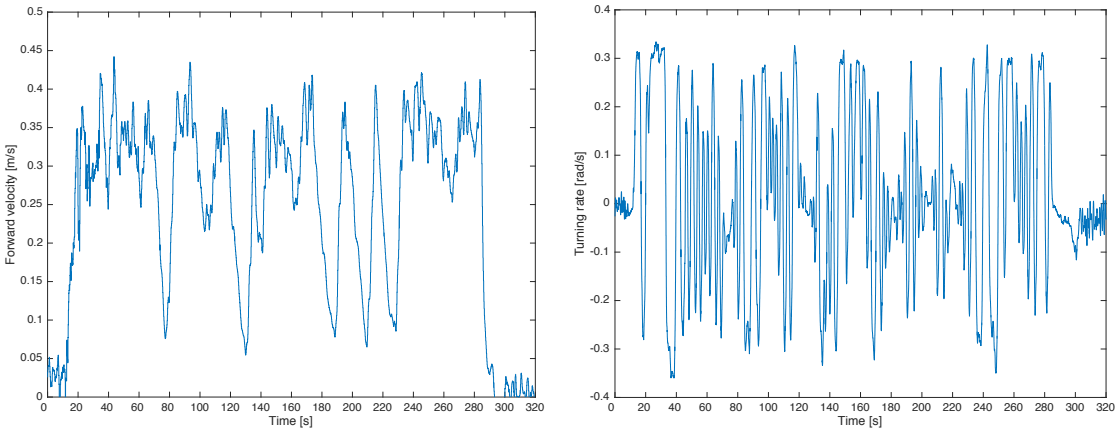
To keep the properties of a given trajectory, the robot has to be able to follow such a path. To ensure this requirement, experiments in the *harbour* have been conducted to tune the controller of the trajectory tracking algorithm introduced in Section 5.3. Specifically, as illustrated in Figure 6.9, the robot is asked to follow a challenging but feasible path of 126.3 metres (red) so its behaviour (blue) can be used to empirically adjust the parameters of the controller.

Figure 6.9: Behaviour of the trajectory tracking algorithm in the Sparus II AUV.

The desired path is expected to be feasible for a maximum velocity of 0.35 m/s, so Sparus II is asked to follow the trajectory with the same maximum velocity. Figure 6.9 depicts two interesting facts. First, that the controller is able to make the system converge to the desired path even if the robot is not placed in the exact starting position of the trajectory. Secondly, that the maximum error committed when following the path is of 0.41 metres. Note that this measurement does

not consider the initial oscillation of the system, which has been expressly caused to show the convergence of the closed-loop system.

Additionally, Figure 6.10 illustrates the forward velocities and turning rates of the system during this survey. Both velocities are variant during the survey, since the proposed path tracking algorithm adjusts both control inputs, without fixing any of them, according to the current system state and the reference. As it has been previously stated in this document, this behaviour is preferable than geometrical approaches such as Dubins [32] and Red-Shepp [33] because the dynamic range of the system is more exploited.



(a) (b)

Figure 6.10: System velocities when following a path. (a) Forward velocities. (b) Turning rates.

Those experiments have also been used to evaluate the error committed when modelling the system. Obtaining ground truth in underwater environments is not always possible and when it is, it requires setting external localisation systems. Since doing an extensive evaluation of the noise in the system goes beyond the scope of this thesis, the error introduced in the model has been estimated from the covariance matrix of the navigation filter. Since the discretized model adds noise in each iteration, $\mathrm{P_{n_t}}$ has been adjusted for a $\Delta \mathrm{T} = 0.1$ seconds, leading to

$$\mathrm{P_{n_t}} = \begin{bmatrix} 0.00618 & 0 \\ 0 & 0.00618 \end{bmatrix}. \tag{6.1}$$

A more intuitive explanation of the modelled $\mathrm{P_{n_t}}$ is that after a 120 seconds of navigation, the real robot's location will be with a 99% confidence within a radius of 8.25 metres around the robot's belief. Note that this example considers the initial robot's belief to be completely certain.

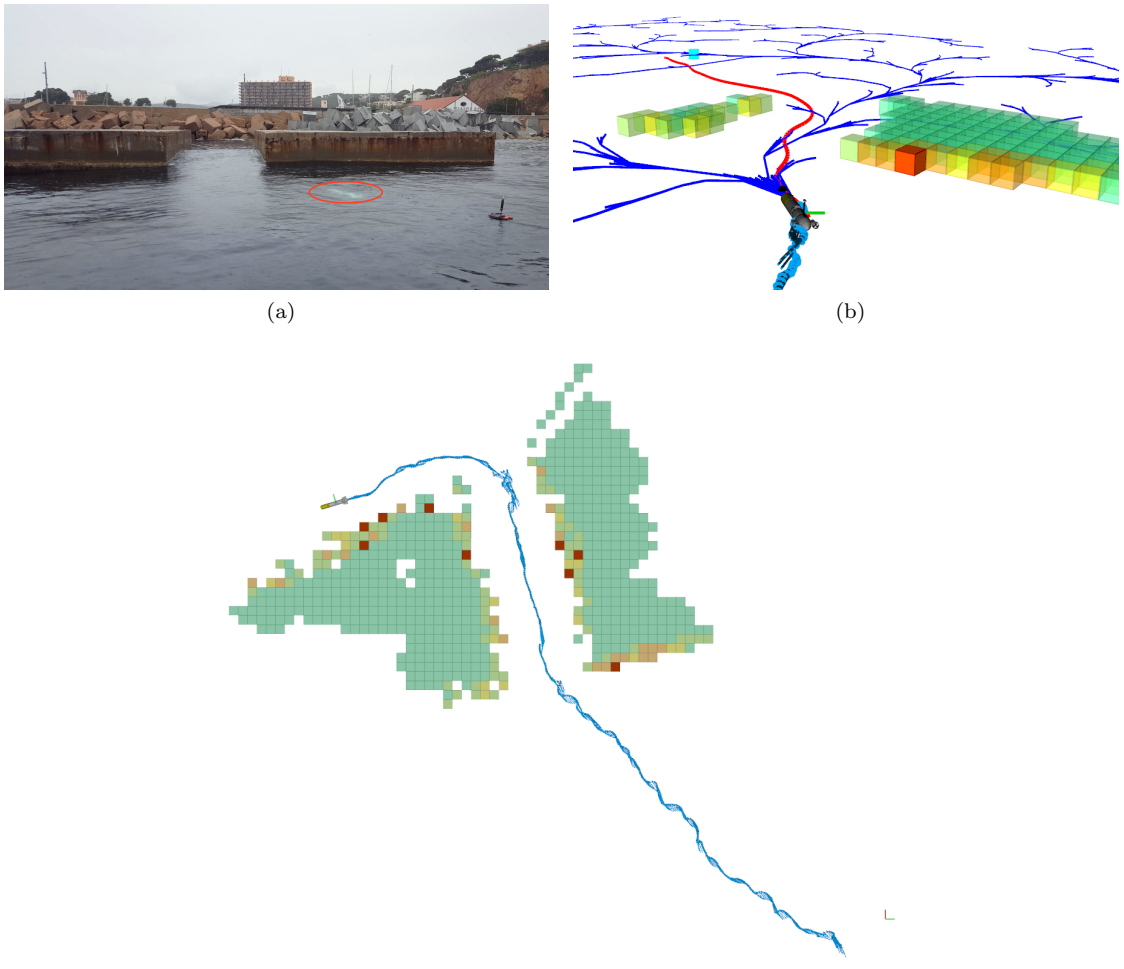
6.3.2 Undiscovered environment

The experiment simulated in the *blocks* scenario has been conducted in real conditions, but setting $\mathrm{p_{safe}} = 0.9$. The objective of this trial is to show the suitability of the proposed framework in real-world applications, where many non-modelled perturbations are present, such as waves and currents. Moreover, there are particles in suspension which difficult a clear perception of the real

obstacles in the surroundings.

The same mission setup, but with different initial positions, has been attempted 6 times. In one of them, the mission has been aborted to avoid the vehicle colliding with one of the blocks. In this case, the planner has found a path through an unmapped area which, because of the perception limitations stated at the end of Section 6.2.2, has not been perceived as occupied in time.

In the other 5 trials, SPARUS II has safely achieved the goal by following the computed path. One of those experiments is reported in Figure 6.11. Figure 6.11a and Figure 6.11b depicts Sparus II simultaneously mapping and planning to reach the goal, while Figure 6.11c shows the path executed for achieving the goal through the breakwater structure.



(a) (b)

(c)

Figure 6.11: Experiment in the real *blocks* scenario. (a)-(b) Sparus II AUV during the survey. (c) Environment awareness and path found through the breakwater structure.

In this experiment, Sparus II is deployed at approximately 25 metres from the breakwater structure. As soon as the mission starts, the solutions proposed by the SST planner are almost straight lines to the goal, what proves its asymptotically near-optimal property. After navigating approximately 15 metres, the robot starts perceiving the block on the right but it does not invalidate

the current path. However, when the block on the left is detected and added to the map, the current path is probabilistically in collision with the environment. From that point, the framework does a continuous mapping-validating-replanning routine to address the incremental nature of the environment awareness. After a total of 34 replannings from the starting point, the goal is achieved. The executed trajectory has a length of 57.9 metres and has been accomplished in $3'07''$.

The overall framework's behaviour when dealing with real conditions has been similar to the one observed in the simulations. However, after analysing each module of the framework separately, it has been noticed that (i) the perception sensor is noisier and the robot's localisation is less accurate than in the simulations, thus compromising the quality of the generated map, and (ii) the trajectory tracking algorithm oscillates around the path because of the waves and currents. Despite these challenging conditions, the framework succeeds in finding and driving the system towards the desired goal region (Figure 6.11c).

Chapter 7

Conclusions and future work

The present dissertation comes to an end with a summary of all the work that has been done. Moreover, some future work is suggested as a continuation of this dissertation.

7.1 Conclusions

This dissertation has presented a framework conceived to efficiently and jointly address all the challenges arisen in the perception, mapping, planning and control fields when conducting online mapping and motion planning under probabilistic constraints and in unknown environments. In contrast to other online mapping and planning methods in the literature, this novel approach dodges the need of using ad-hoc heuristics for determining the validity of a trajectory by kinodynamically and probabilistically constraining the path. Thus, the main contribution of this thesis is the uncertainty-based procedure which ensures the safety of a feasible path going through an undiscovered environment.

The presented framework is threefold. A mapping algorithm based on local maps incrementally builds a representation of the environment while keeping the relative uncertainty of the system with the different parts of the map. Then, the planning problem with partial knowledge of the environment is addressed with an anytime policy, which lets the SST algorithm to find an asymptotically nearly-optimal path during a specific period of time. To find a solution path, the SST planner builds a tree in the B-Space, where nodes represent uncertain states or beliefs and edges are local controllers that meet the kinodynamic constraints. Knowing the relative uncertainty of each portion of the map with respect to the system's belief allows establishing some minimum safety guarantees over the path even in undiscovered environments. Finally, a trajectory tracking algorithm based on the planner propagation model drives the system over the calculated path to minimise unexpected behaviours and thus, the risk of collision.

The whole framework has been implemented along the ROS and uses the OctoMap library and the OMPL to address the mapping and planning challenges, respectively. Then, the framework has been integrated with the COLA2 architecture of the Sparus II AUV to conduct several experiments in simulated and real-world scenarios. The results demonstrate the potential and suitability of the proposed framework to deal with real robotics constraints at the same time of exploring

undiscovered environments. The kernel-based routine to compute the probability of collision of a certain belief in an uncertain environment turns to be slightly costly but it explicitly computes such probability without conservationism and deals with nonconvex representations of the environment. Even though the obtained results are promising, the algorithm can be further improved by tackling the aspects suggested in the future work section.

7.2 Future work

The actual framework proved to be a novel alternative to plan under kinodynamic and probabilistic constraints over undiscovered environments while being suitable for real-world robotic applications. Thus, the imminent efforts will be focused on conducting more experiments in real-world scenarios in order to publish this work in a journal or conference of robotics. Apart from that, some long-term future work is also proposed to either upgrade the actual framework's performance or to expand it to new horizons.

- *Enhance the mapping capabilities.* Planning online requires high-quality maps. Thus, it is essential to reduce the presence of noise on the environment awareness and to work towards a globally consistent map.
- *Consider mapping constraints.* Even if the mapping capabilities are enhanced, they should be considered when planning a path. Specifically, the system should be able to explore the areas that the path goes through before getting into them.
- *Explore new sampling strategies.* Sampling-based algorithms are known to lose efficiency in narrow passages when a proper sampling strategy is not chosen. Thus, it would be interesting to study an intelligent method for switching between different sampling strategies according to the environment characteristics.
- *Explore the concept of distance in* $\mathbb{R}^2 \times \mathbb{S}^1$ *systems.* In high-dimensional spaces, all states turn to be close to each other. For that reason, it is essential to study a proper way to represent distances in $\mathbb{R}^2 \times \mathbb{S}^1$ systems.
- *Explore the concept of distance in the belief space.* If the uncertainty is included in the distance measurement, the planner would not only aim to reach the goal with a short path but with a few uncertainty.
- *Enhance the probabilistic collision checking method.* Checking the validity of a state is the most expensive task in a sampling-based algorithm. Even though this work has reported the proposed approach to have an acceptable performance, it would be great to enhance it to obtain a denser space exploration.
- *Automatically adapt the replanning period.* Replanning should be done as soon as the SST planner has approximately converged to a nearly optimal solution or whenever new obstacles are discovered in the environment.

- *Work towards framework correctness.* The framework should deal with those scenarios where the robot can get trapped. Once the system is stuck, a possible solution could be to execute a motion primitive to make the robot spin and then, drive the robot back along the performed path until the planner finds a new path to the goal.
- *Work towards an asymptotically nearly-optimal framework.* The SST planner is one of the key elements of the proposed framework. Thus, the framework could keep its properties regarding asymptotically nearly-optimality.
- *Extend the work to 3D workspaces.* Many real-world systems deal with 3D environments and their corresponding constraints. Thus, expanding this work is of great interest to completely exploit those systems' capabilities.

Bibliography

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

[2] David Ribas, Narcis Palomeras, Pere Ridao, Marc Carreras, and Angelos Mallios. Girona 500 AUV: From survey to intervention. *IEEE/ASME Transactions on Mechatronics*, 17(1):46–53, 2012.

[3] Marc Carreras, Carles Candela, David Ribas, Angelos Mallios, Lluís Magí, Eduard Vidal, Narcís Palomeras, and Pere Ridao. Sparus II, design of a lightweight hovering AUV. In *Proceedings of the 5th International Workshop on Marine Technology (MARTECH), Girona, Spain*, volume 911, page 163164, 2013.

[4] Nils J Nilsson. A mobile automaton: An application of artificial intelligence techniques. Technical report, DTIC Document, 1969.

[5] Tomás Lozano-Pérez and Michael A Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, 1979.

[6] Tomás Lozano-Pérez. Spatial planning: A configuration space approach. In *Autonomous robot vehicles*, pages 259–271. Springer, 1990.

[7] Vladimir J. Lumelsky and Alexander A. Stepanov. Path-planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. *Algorithmica*, 2:403–430, 1987.

[8] Ishay Kamon, Ehud Rivlin, and Elon Rimon. New range-sensor based globally convergent navigation algorithm for mobile robots. In *Proceedings - IEEE International Conference on Robotics and Automation.*, volume 1, pages 429–435, April 1996.

[9] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

[10] Yoram Koren and Johann Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1398–1404. IEEE, 1991.

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

[12] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310–3317. IEEE, 1994.

[13] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[14] Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[15] David Hsu, J-C Latombe, and Rajeev Motwani. Path planning in expansive configuration spaces. In *Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on*, volume 3, pages 2719–2726. IEEE, 1997.

[16] Steven M LaValle. Rapidly-exploring random trees: A new tool for path planning. 1998.

[17] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[18] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[19] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *Journal of the ACM (JACM)*, 40(5):1048–1066, 1993.

[20] Liang Yang, Juntong Qi, Dalei Song, Jizhong Xiao, Jianda Han, and Yong Xia. Survey of robot 3D path planning algorithms. *Journal of Control Science and Engineering*, 2016:5, 2016.

[21] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, 2016.

[22] Joao Sequeira and Maria Isabel Ribeiro. A two level approach for underwater path planning. In *OCEANS'94.'Oceans Engineering for Today's Technology and Tomorrow's Preservation.'Proceedings*, volume 2, pages II–87. IEEE, 1994.

[23] Shinji Arinaga, Shoji Nakajima, Hideo Okabe, Akira Ono, and Yutaka Kanayama. A motion planning method for an AUV. In *Autonomous Underwater Vehicle Technology, 1996. AUV'96., Proceedings of the 1996 Symposium on*, pages 477–484. IEEE, 1996.

[24] Maxim Likhachev and Dave Ferguson. Planning long dynamically feasible maneuvers for autonomous vehicles. *The International Journal of Robotics Research*, 28(8):933–945, 2009.

[25] Dmitri Dolgov, Sebastian Thrun, Michael Montemerlo, and James Diebel. Path planning for autonomous vehicles in unknown semi-structured environments. *The International Journal of Robotics Research*, 29(5):485–501, 2010.

[26] Yoshiaki Kuwata, Justin Teo, Gaston Fiore, Sertac Karaman, Emilio Frazzoli, and Jonathan P How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on Control Systems Technology*, 17(5):1105–1118, 2009.

[27] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[28] Dustin J. Webb and Jur van den Berg. Kinodynamic RRT*: Optimal motion planning for systems with linear differential constraints. *CoRR*, abs/1205.5088, 2012.

[29] Alessandro De Luca, Giuseppe Oriolo, and Marilena Vendittelli. Stabilization of the unicycle via dynamic feedback linearization. In *6th IFAC Symp. on Robot Control*, pages 397–402, 2000.

[30] Alessandro De Luca and Giuseppe Oriolo. Motion planning and trajectory control of an underactuated three-link robot via dynamic feedback linearization. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 3, pages 2789–2795. IEEE, 2000.

[31] Gustavo Goretkin, Alejandro Perez, Robert Platt, and George Konidaris. Optimal sampling-based planning for linear-quadratic kinodynamic systems. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 2429–2436. IEEE, 2013.

[32] L. E. Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of Mathematics*, 79(3):497–516, 1957.

[33] James Reeds and Lawrence Shepp. Optimal paths for a car that goes both forwards and backwards. *Pacific journal of mathematics*, 145(2):367–393, 1990.

[34] Jeong Hwan Jeon, Sertac Karaman, and Emilio Frazzoli. Anytime computation of time-optimal off-road vehicle maneuvers using the RRT. In *CDC-ECE*, pages 3276–3282. IEEE, 2011.

[35] Yanbo Li, Zakary Littlefield, and Kostas E Bekris. Asymptotically optimal sampling-based kinodynamic planning. *The International Journal of Robotics Research*, 35(5):528–564, 2016.

[36] Konstantinos I Tsianos, Ioan A Sucan, and Lydia E Kavraki. Sampling-based robot motion planning: Towards realistic applications. *Computer Science Review*, 1(1):2–11, 2007.

[37] Juan David Hernández, Mark Moll, Eduard Vidal, Marc Carreras, and Lydia E Kavraki. Planning feasible and safe paths online for autonomous underwater vehicles in unknown environments. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 1313–1320. IEEE, 2016.

[38] Leslie Pack Kaelbling, Michael L Littman, and Anthony R Cassandra. Planning and acting in partially observable stochastic domains. *Artificial intelligence*, 101(1):99–134, 1998.

[39] Ron Alterovitz, Thierry Siméon, and Kenneth Y Goldberg. The stochastic motion roadmap: A sampling framework for planning with markov motion uncertainty. In *Robotics: Science and systems*, volume 3, pages 233–241, 2007.

[40] Andrea Censi, Daniele Calisi, Alessandro De Luca, and Giuseppe Oriolo. A bayesian framework for optimal motion planning with uncertainty. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1798–1805. IEEE, 2008.

[41] Samuel Prentice and Nicholas Roy. The belief roadmap: Efficient planning in belief space by factoring the covariance. *The International Journal of Robotics Research*, 28(11-12):1448–1465, 2009.

[42] Ali-Akbar Agha-Mohammadi, Suman Chakravorty, and Nancy M Amato. FIRM: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements. *The International Journal of Robotics Research*, 33(2):268–304, 2014.

[43] Lars Blackmore, Hui Li, and Brian Williams. A probabilistic approach to optimal robust path planning with obstacles. In *American Control Conference, 2006*, pages 7–pp. IEEE, 2006.

[44] Brandon Luders, Mangal Kothari, and Jonathan How. Chance constrained RRT for probabilistic robustness to environmental uncertainty. In *AIAA guidance, navigation, and control conference*, page 8160, 2010.

[45] Robert Platt Jr, Russ Tedrake, Leslie Kaelbling, and Tomás Lozano-Pérez. Belief space planning assuming maximum likelihood observations. 2010.

[46] T. Fraichard and R. Mermond. Path planning with uncertainty for car-like robots. In *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, volume 1, pages 27–32 vol.1, May 1998.

[47] A. Sankaranarayanan and M. Vidyasagar. *Path planning for moving a point object amidst unknown obstacles in a plane: The universal lower bound on the worst path lengths and a classification of algorithms*, volume 2, pages 1734–1741. Publ by IEEE, 1991.

[48] Thomas L Dean and Mark S Boddy. An analysis of time-dependent planning. In *AAAI*, volume 88, pages 49–54, 1988.

[49] Shlomo Zilberstein and Stuart J. Russell. Approximate reasoning using anytime algorithms. In S. Natarajan, editor, *Imprecise and Approximate Computation*. Kluwer Academic Publishers, 1995.

[50] Suman Chakravorty and R Scott Erwin. Information space receding horizon control. In *Adaptive Dynamic Programming And Reinforcement Learning (ADPRL), 2011 IEEE Symposium on*, pages 302–309. IEEE, 2011.

[51] Noel E Du Toit and Joel W Burdick. Robotic motion planning in dynamic, cluttered, uncertain environments. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, pages 966–973. IEEE, 2010.

[52] Robert Bohlin and Lydia E Kavraki. Path planning using lazy PRM. In *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, volume 1, pages 521–528. IEEE, 2000.

[53] Kostas E. Bekris, Konstantinos I. Tsianos, and Lydia E. Kavraki. Safe and distributed kinodynamic replanning for vehicular networks. *Mob. Netw. Appl.*, 14(3):292–308, June 2009.

[54] Nikolaus Vahrenkamp, Tamim Asfour, and Rudiger Dillmann. Efficient motion planning for humanoid robots using lazy collision checking and enlarged robot models. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 3062–3067. IEEE, 2007.

[55] R. W. Brockett. Asymptotic stability and feedback stabilization. In *Differential Geometric Control Theory*, pages 181–191. Birkhauser, 1983.

[56] OJ Sordalen and O Egeland. Exponential stabilization of nonholonomic chained systems. *IEEE transactions on automatic control*, 40(1):35–49, 1995.

[57] Juan D Tardós, José Neira, Paul M Newman, and John J Leonard. Robust mapping and localization in indoor environments using sonar data. *The International Journal of Robotics Research*, 21(4):311–330, 2002.

[58] José A Castellanos, José Neira, and Juan D Tardós. Limits to the consistency of EKF-based SLAM. In *5th IFAC Symp. on Intelligent Autonomous Vehicles, IAV'04*, 2004.

[59] Pedro Piniés and Juan D Tardós. Scalable SLAM building conditionally independent local maps. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 3466–3471. IEEE, 2007.

[60] Morgan Quigley, Josh Faust, Tully Foote, and Jeremy Leibs. ROS: an open-source robot operating system. 2009.

[61] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at `http://octomap.github.com`.

[62] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. `http://ompl.kavrakilab.org`.

[63] Thor I Fossen, Morten Breivik, and Roger Skjetne. Line-of-sight path following of underactuated marine craft. *Proceedings of the 6th IFAC MCMC, Girona, Spain*, pages 244–249, 2003.

[64] Narcís Palomeras Rovira, Andrés El-Fakdi Sencianes, Marc Carreras Pérez, and Pere Ridao Rodríguez. COLA2: A control architecture for AUVs. © *IEEE Journal of Oceanic Engineering, 2012, vol. 37, p. 695-716*, 2012.