Norwegian University of Science and Technology

TTK4550 - Specialization Project

Project Thesis

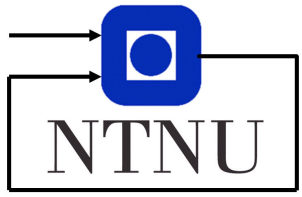
Dynamic Path Planning for Underwater Exploration

Supervisor
Annette Stahl

Author
Kjetil S. Gjerden

Co-Supervisor
Marco Leonardi

June 17, 2019



Department of Engineering Cybernetics

Abstract

Robot path planning is the act of finding a feasible, collision-free path between the start and goal positions of a mobile robot. Efficient and robust path planning systems are important in many robotic applications, including autonomous vehicles, planetary exploration and even agricultural robotics. Many methods for path planning have been developed, each with different strengths and weaknesses, and some catering to specific robotic applications. This report provides a short review of some existing path planning and exploration methods for 2D and 3D planning situations, before concentrating on state-of-the-art techniques, focusing on their potential applications for online planning for autonomous underwater vehicles (AUVs) in an exploratory setting. The different methods have been evaluated based on simple test cases and results obtained in the literature based on simulations or real-world data. Advantages and disadvantages of the various approaches are discussed before an efficient, safe and dynamic method combining a three-dimensional Voronoi diagram with an optimal heuristic search algorithm is presented. Different waypoint generation strategies that facilitate autonomous exploration are reviewed and methods to guarantee the generation of continuous paths briefly examined.

Keywords

3D path planning, Autonomy, Collision avoidance, Path planning, Path smoothing, Robotic exploration, Survey, Voronoi diagram

Contents

List of Figures

List of Symbols

$\mathcal{A}$	space occupied by the robot or vehicle (see section 2.1)
C	configuration space (see section 2.1)
C_{free}	free configuration space (see section 2.1)
C_{obst}	obstacle-occupied configuration space (see section 2.1)
$\mathcal{O}$	space occupied by obstacles (see section 2.1)
$P_i,\ P_g$	initial and goal poses
q	robot configuration
$\mathbf{q}$	quaternion
$\mathbf{r}(\varpi)$	path connecting P_i and P_g (see section 2.1.1)
$\mathbf{R}$	rotation matrix
$\mathbf{T}$	homogeneous transformation matrix (see section 2.1)
$\mathcal{T}$	a set of homogeneous transformations
u	action or control input
$\mathcal{U}$	the set of all possible actions (see chapter 2)
$\mathcal{W}$	the environment or workspace (see section 2.1)
X	state space
x	state
X_g	set of all goal states
ϖ	path parameter (see section 2.1.1)
ϕ, θ, ψ	roll, pitch and yaw angles

1 | Introduction

Path planning for mobile robots is the process of finding a feasible path connecting two points in space. Autonomous robots have seen considerable advancement in recent years due to the escalating dependency of increasingly complex autonomous systems, and the field of path planning has evolved with it. An autonomous system is a system capable of performing tasks with little to no human interaction by doing intelligent reasoning based on sensory information. Such systems are then able to fulfil tasks unaided, even with loosely defined goals and when subjected to dynamic environments. This results in a wide range of commercial, military and scientific applications, including underwater inspections [1], [2], oceanographic mapping and pipeline inspection [3]; planetary explorers [4], [5]; and aerial drone inspection [6].

To facilitate these kinds of autonomous operations, the robot itself needs to be able to perceive its environment and from that information plan actions to fulfil a goal. Good decision making is directly reliant on sufficient knowledge of the environment, or *workspace*, the robot operates in. With state-of-the-art sensor technology, more robust perception systems are ever available, allowing progressively more potent motion estimation, tracking and localisation, even in underwater environments — albeit still challenging, particularly with underwater visual sensors [7]–[9]. Many components in an autonomous robot depend on its sensory capabilities, one of them is its path planning system. Path planning in itself has been a field subject to extensive research in the past few decades, with increased attention in recent years. This is largely due to a vast number of applications, including robotics, manufacturing, even molecular- and computational biology. To date, the majority of work has focused on motion or path planning for mobile robots in two-dimensional environments.

The path planning problem can in its essence be boiled down to calculating a feasible, collision-free path given the initial and a goal configuration or pose. A feasible path is one that leads the robot from its initial pose to the goal while keeping the robot from interacting with the obstacle space, as defined by sensor data. The robot's initial pose is estimated using a location, pose or simultaneous location and mapping (SLAM) technique, while the goal pose can, for example, be calculated using an exploration technique [10], [11]. Calculation of the path is performed on a discrete representation of the real-world environment. Popular representations include graphs, occupancy grids and tree structures. In recent years, a considerable amount of

research has been conducted on robotic path planning, resulting in a multitude of proposed solutions. The earliest methods were quite computationally expensive, like the one introduced by [12], solving the piano mover's problem in doubly-exponential time, making them unsuitable for online applications. These early solutions were examples of so-called *complete* algorithms. A complete algorithm either finds a solution or, correctly, states that no solution exists.

The algorithm proposed by Schwartz and Sharir in 1983 [12], using Collins decomposition, is an example of what is often referred to as a *combinatorial* method [13]. These methods solve the planning problem by calculating paths through the continuous configuration space. By doing certain approximations, i.e. relaxing the notion of completeness, more efficient algorithms were developed, namely the *sampling-based* methods. These methods circumvent the problem of computational cost by not explicitly constructing the obstacles in the configuration space. Instead, these methods rely on a collision detection-based sampling scheme, which is often based on random sampling. Although this simplification means the loss of algorithmic completeness, they manage to attain *probabilistic completeness*, as the probability that the algorithm finds a solution converges to one as the number of samples increases.

Due to their efficiency, these sampling-based methods have become the go-to solution for many modern robotic applications, as computational power often is limited. This has, in turn, lead to a myriad of different variants being developed, each with their advantages and disadvantages. Some methods obtain computational efficiency even in cluttered environments or in situations where the robot as a high number of DoF, while others are developed with the intention of obtaining the most optimal solution. More and more modern applications, however, push for the need for three-dimensional planning systems. In three dimensions, many established and computationally feasible methods become impractical without modifications due to the expanded space. The specific vehicle dynamics, environment complexity and additional operation-specific constraints then weigh heavily in on which methods are viable.

Looking at autonomous underwater applications, the robot is ideally capable of performing long-range missions without human interaction. Planning a new path in the case of unforeseen events then must happen quickly while still providing a guaranteed safe path. The planning system must, therefore, be *online*, while also, preferably, taking energy consumption into consideration. The underlying scenario for this work is a partial or fully autonomous underwater robot with a SLAM system capable of creating a continuously expanding map. The main objective of this paper is then to create a short survey of the most promising state-of-the-art methods in recent literature and, based on the constraints of the stated scenario, try to answer the question of which path planning system is best suited for an underwater exploratory operation. Advantages and disadvantages of classic and recent state-of-the-art path planning methods will be reviewed and different strategies for exhaustive exploration of the environment

based on waypoint generation, as well as techniques for generating smooth paths, will be discussed.

The report is organised as follows. Chapter 2 gives an introduction to the path planning problem and how to mathematically represent the robot and the operational environment. In chapter 3, different well-established methods for 2D path planning and related research are discussed. Chapter 4 provides some insight into different environment representations and their use-cases and reviews the methods introduced in chapter 3 their applicability in 3D path planning. Furthermore, in chapter 5, strategies for performing exhaustive environment exploration are reviewed and the concepts of safe and energy-sensible planning discussed. A short summary of the most important points is provided in chapter 6, together with a proposition for a complete, safe and efficient path planning system.

1.1 Literature Search

To research the topic at hand, different journals and databases were consulted. The main databases used were NTNU's own library database Oria combined with Elsevier's SCOPUS database for broader searches. Queries in Oria give results from NTNU's own digital and physical sources as well as from subscribed journals. Combining this with results from SCOPUS provide a wide search-base. From these two databases, more refined searches were conducted through the specific journals' databases that showed relevance to the field in question. The most prominent among the journals used were IEEE, with supplementary searches being conducted through Google Scholar, among others. Specific literature searches included — depending on the database — queries along the lines of:

- "Exploration + Path Planning : articles & books"
- "Robotic exploration AND autonomous, articles"
- "Path planning"
- "Autonomous AND navigation"
- "Dynamic path planning"

Articles in journals not familiar to the author were tried validated by consulting the list of the *Norwegian accredited journals list for publications 2018*, provided by the *Norwegian Register for Scientific Journals, Series and Publishers* [14]. Articles found in journals on this list were deemed of satisfactory quality, given that they demonstrated adequate quality in writing and

presentation of results. More recently dated papers were favoured when researching state-of-the-art methodology, but previous work has also been investigated to examine different view points and approaches.

2 | Background Theory

Solving the path planning problem requires the definition of an underlying mathematical representation. This section is heavily based on the works of Latombe, LaValle and Tsourdos et. al, see [13], [15]–[17] for a more in-depth coverage of the topic at hand.

Similarly to many dynamical systems, the planning problem can be represented using a *state space* representation. Each state $x \in X$ contains information about the robot's pose and is part of a finite set of states, X. The state space then consists of all necessary information needed to sufficiently solve the planning problem at hand, disregarding all non-relevant information. The robot, or *agent*, transitions from one state to another specified by a *state transition function*, $f(\cdot)$. This function describes the transition from state x to state x' when the agent executes the action u, calculated by the planner [13], [18]:

$$x' = f(x, u) \ . \tag{2.1}$$

Furthermore, let $U(x)$ denote all allowed actions for each state x. The set of all possible actions over all states can then be defined as in [13]:

$$\mathcal{U} = \bigcup_{x \in X} U(x) \ .$$

The path planning problem can be generalised to calculating a path from an initial pose P_i to one or more goal poses, $P_g \in X_g \subset X$, where X_g represents the set of goal states.

The ultimate goal of a path planning algorithm can then be described as calculating a sequence of actions, u, that transforms the initial state x_i to a goal state $x_g \in X_g$. In the simplified case where the environment is assumed discrete and two-dimensional, the state space X can be represented as a graph, with the vertices representing the states. In this case, there exists a directed edge between x and x' iff. there exists a $u \in U(x)$ such that $x' = f(x, u)$. In the case of a continuous state space, the assumption of a finite state space is no longer valid as X is no longer countably infinite [13] [1], and more care is needed when modelling the state space.

[1] Countably infinite set: Any infinite set which can be put in a one-to-one correspondence with $\mathbb{N}$ (countable). Cardinality: $\aleph_0$.

2.1 Modelling the Path Planning Problem

To be able to relate the robot's actions and state to its environment, the *workspace* $\mathcal{W} = \mathbb{R}^n$ it resides in, must be defined[2]. Let $\mathcal{O} \subseteq \mathcal{W}$ be the *obstacle region* and $\mathcal{A} \subseteq \mathcal{W}$ be the space in $\mathcal{W}$ occupied by the robot. The obstacle region is the part of the world that is occupied by static bodies, such as an exterior wall, a fence or an underwater rock formation, and will in most cases be fixed in $\mathcal{W}$. The robot, $\mathcal{A}$, will be modelled as a moving rigid-body.

In rigid body kinematics, the act of coordinate transformation is essential. Two concepts used to execute such transformations are coordinate *rotation* and *translation*. In the 3D case, a robot $\mathcal{A}$ can be rotated around any of the three defined orthogonal axes. Such a rotation about a given fixed axis is called a *simple rotation*. Furthermore, it can be shown that more complex rotations can be described as a set of *composite rotations* around the *roll* (x), *pitch* (y) and *yaw* (z) axes [19]. The rotation matrices for each cardinal axis can be described as a three-dimensional matrix performing a counterclockwise rotation around each axis.

By combining the rotation and translation of a rigid body, the *homogeneous transformation matrix* can be defined as follows [19]:

Definition 2.1.1 *A rigid-body transformation is a function, $T : \mathcal{A} \to \mathcal{W}$, that maps every point of the rigid-body $\mathcal{A}$ into $\mathcal{W}$ while preserving the distance between any pair of points p, $q \in \mathcal{A}$ as well as the cross-product of any two vectors.*
The homogeneous transformation matrix for a 3D rigid body is then given by:

$$\mathbf{T}(\mathbf{t}, \phi, \theta, \psi) = \begin{bmatrix} \mathbf{R}(\phi, \theta, \psi) & \mathbf{t} \\ \mathbf{0}^\top & 1 \end{bmatrix} \in SE(3) \ . \tag{2.2}$$

Definition 2.1.1 assumes $R(\phi, \theta, \psi)$ to be any composite 3D rotation matrix, $\mathbf{t}$ is the translation vector and $SE(3)$ denotes the *Special Euclidean Group* of dimension three, defined by [19]:

$$SE(3) \coloneqq \left\{ \mathbf{T} \mid \mathbf{T} = \begin{bmatrix} \mathbf{R}(\phi, \theta, \psi) & \mathbf{t} \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix}, \ \mathbf{R} \in SO(3), \ \mathbf{t} \in \mathbb{R}^3 \right\} \ . \tag{2.3}$$

Equation (2.3) then defines the group of all 4×4 dimensional homogeneous transformation matrices. Using definition 2.1.1, the subspace in the world, $\mathcal{W}$, occupied by the transformed robot can be defined by

$$\mathbf{T}(\mathcal{A}) = \{T(a) \in \mathcal{W} | a \in \mathcal{A}\} \ . \tag{2.4}$$

[2]For most cases: $n = 2$ (2D) or $n = 3$ (3D).

Based on these definitions, the set of all allowable transformations can be defined. This set is called the *configuration space*, C, also known as the C-space [15]. For a robot moving in a three-dimensional environment, its configurations, q, can be represented as $q = (x, y, z, \eta, \epsilon_1, \epsilon_2, \epsilon_3)$, where $(\eta, \epsilon_1, \epsilon_2, \epsilon_3)$ describes a quaternion $\mathbf{q} = \eta + \epsilon_1\mathbf{i} + \epsilon_2\mathbf{j} + \epsilon_3\mathbf{k}$ [20]. Due to this double covering of $SO(3)$, two quaternions will correspond to the exact same rotation [21]. More explicitly, this means that $\mathbf{q} \sim -\mathbf{q}$ with respect to spatial rotation.

To define the encompassed space, consider a homeomorphism $f : S^3 \mapsto SO(3)$ [22] with a kernel $f = \{\pm 1\}$ equal to the centre of S^3. Thus, the co-sets of $\ker f$ in S^3 defines antipodal pairs, each pair further defining a line between them in $\mathbb{R}^4$-space. From definition 2.1.2 it can be seen that this is similar to the definition of the *real projective space*, $\mathbb{RP}^3$, in terms of the antipodal map $\pi : S^3 \mapsto \mathbb{RP}^3$, indicating $SO(3) = \mathbb{RP}^3$ [23].

Definition 2.1.2 Real Projective Space
Let $\mathbb{S}^n \subseteq \mathbb{R}^{n+1}$ *be an n-dimensional unit sphere defined by* $\mathbb{S}^n \coloneqq \left\{ x \in \mathbb{R}^{n+1} \mid |x - y| = 1 \right\}$. *The real projective space of dimension n is then defined as the quotient space [24], [25]:*

$$\mathbb{RP}^n \coloneqq \mathbb{S}^n / \{\pm\} \ .$$

$\{\pm\}$ *here identifies the* antipodal points $\in \mathbb{S}^n$.

Therefore, the resulting C-space for all 3D transformations can be defined as

$$C = \mathbb{R}^3 \times \mathbb{RP}^3 \ . \tag{2.5}$$

From the definition of C in eq. (2.5) it is possible to define two other sets, namely the *free configuration space*, C_{free}, and the *obstacle region of the configuration space*, C_{obst}, simply being the complementing set of C_{free} [15]:

$$C_{\text{free}} \coloneqq \{ q \in C \mid \mathcal{A}(q) \cap \mathcal{O} = \varnothing \} \tag{2.6a}$$

$$C_{\text{obst}} \coloneqq \{ q \in C \mid q \notin C_{\text{free}} \} = C \setminus C_{\text{free}} \ , \tag{2.6b}$$

where $\mathcal{A}(q) \subset \mathcal{W}$ is the closed set of points occupied by the robot when transformed to configuration q.

2.1.1 Continuous-Space Path Planning

After defining these sets of transformations and representations, the path planning problem can be defined as the search for a set of transformations, $\mathcal{T}$, that transforms the robot from an

initial pose, P_i, to a goal pose, P_g:

$$\mathbf{T}(\mathcal{A}, P_i) \overset{\mathcal{T}}{\longmapsto} \mathbf{T}(\mathcal{A}, P_g) \ .$$

This formulation is more closely related to the act of motion planning. For the case of basic path planning, this can be simplified to the problem of producing one or more *flyable* paths, $r(\varpi)$, connecting P_i and P_g such that [26]:

$$\begin{gathered} P_i \overset{\mathbf{r}(\varpi)}{\longmapsto} P_g \\ P_i(x_i, y_i, z_i, \phi_i, \theta_i, \psi_i) \overset{\mathbf{r}(\varpi)}{\longmapsto} P_g(x_g, y_g, z_g, \phi_g, \theta_g, \psi_g) \ , \end{gathered} \tag{2.7}$$

where ϖ represents the path parameter, e.g. a length variable. A flyable path here refers to a path that satisfies the given vehicles kinodynamic constraints [17]. The path planning problem can then be boiled down to generating a collision-free, continuous path, r, from P_i to P_g, defined as [15]:

$$\mathbf{r} : [0, 1] \mapsto C_{\text{free}} \ , \quad \mathbf{r}(0) = P_i, \ \mathbf{r}(1) = P_g \ . \tag{2.8}$$

A much used strategy for solving continuous-space problems is to transform the model into a discrete-space model. The two main strategies of performing this transformation is known as *combinatorial-* and *sampling-based* planning.

Combinatorial planning here refers to the branch of mathematics called *combinatorics*, which focuses on the study of countable discrete structures, including the field of graph theory. Combinatorial planning tries to capture the information in the state space by partitioning the free space into an exact representation using discrete data structures. By using exact representations — e.g. visibility graphs — they attain the property of completeness, i.e. if the problem has a solution it will find it or correctly conclude that no solution exist. The general approach of a combinatorial planner is to first compute a representation of C_{free} without approximating, then applying an optimal search algorithm to find an optimal path.

Sampling-based planners, on the other hand, doesn't explicitly characterise the free space or the occupied space. These methods lets a collision detection algorithm decide whether or not a given configuration lies in C_{free}. These planners then incrementally search the free space for a path, gradually revealing more using an obstacle detector. These planners don't rely on building a complete map, meaning they don't compute more than they have to. This makes these planners more suitable for high-dimensional problems. Two popular methods based on this scheme is the rapidly exploring random tree (RRT) and probabilistic road map (PRM).

3 | Path Planning in Two Dimensions

The path planning problem formulation for practical robot implementation has traditionally been two-dimensional. One of the main contributing factors to an early spark of interest in the field of path planning was Lozano-Perez and Wesley's work in 1979 [27]. A majority of the work in this period resulted in so-called complete methods, such as the work of [12], [28], [29]. The common factor for such methods is that they compute a retraction of C. This endeavour has been proved PSPACE-hard, with complexity increasing doubly exponentially with the robot's DoF [30]–[32]. This is problematic in applications of a higher order, such as planning for a multi-jointed robot, as the complexity can become too great to solve efficiently. For certain applications where online planning is required, these methods will not be satisfactory. This high complexity sparked the development of more efficient, but relaxed methods. The most used methods of this kind are the sampling-based methods. These methods sample C, usually in a probabilistic manner. This results in the loss of completeness, but they attain the property of *probabilistic completeness*. This means that the solution converges to the optimal, complete solution as the number of samples approaches infinity.

In the next sections, a few different methods that have obtained good results in various real-world applications will be presented. Examples include both sample-based and combinatorial methods. These methods will then be demonstrated on a 2D test case, simulating a cave environment.

3.1 Artificial Potential Field

There have been numerous different methods developed solving the path planning problem. One of the earliest methods not based on graph representations or grid searches was the artificial potential field method [33]. This method represents the workspace as an artificial force field, with obstacles exerting repulsive forces while the goal is bestowed an attractive force, referred to as an attractive pole. There are several ways of representing the attractive

potential, but a commonly used potential function is:

$$U_a(\mathbf{x}) = \frac{1}{2}\rho(\mathbf{x} - \mathbf{x}_{\text{goal}})^2 \ , \tag{3.1}$$

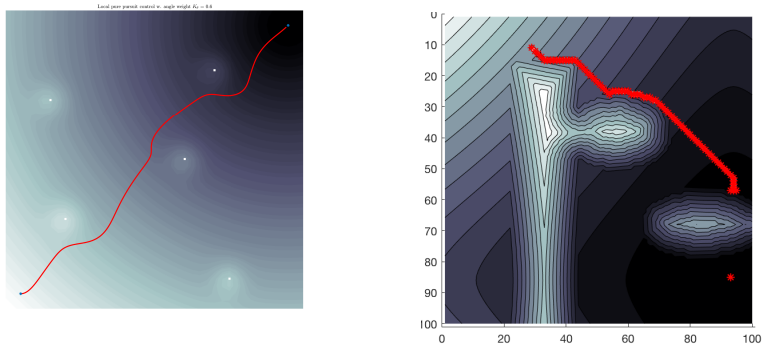
where $\mathbf{x}_{\text{goal}}$ is the goal position and $\rho > 0$ is a scaling factor, or position gain and has a zero-minimum at $\mathbf{x} = \mathbf{x}_{\text{goal}}$. Similarly, the repulsive potential can be expressed as

$$U_r(\mathbf{x}) = \begin{cases} \frac{1}{2}\eta\left(\frac{1}{\kappa} - \frac{1}{\kappa_0}\right)^2 & , \text{if } \kappa \leq \kappa_0 \\ 0 & , \text{if } \kappa > \kappa_0 \end{cases} \ , \tag{3.2}$$

where κ and κ_0 represent the shortest distance to the obstacle and the distance limit of the potential field, respectively. From this the attractive and repulsive forces can be found to be given as

$$\begin{bmatrix} F_a \\ F_r \end{bmatrix} = -\nabla \begin{bmatrix} U_a(\mathbf{x}) \\ U_r(\mathbf{x}) \end{bmatrix} \ . \tag{3.3}$$

Potential field-based methods tend to be relatively computationally efficient, allowing for



(a) Potential field planner navigating a stationary environment consisting of point obstacles.

(b) Example of a potential field planner failing to find a path due to getting stuck in a local minimum.

Figure 3.1: Comparison of the artificial potential field algorithm run on environments of different complexity.

real-time implementations. However, these methods are not without their disadvantages. Most notably is perhaps the fact that they are prone to get stuck in local minima, as exemplified in fig. 3.1a and fig. 3.1b. This susceptibility to local minima becomes greater as the dimensionality of the problem increases. From this, it follows that artificial potential field methods in modern literature are primarily used as local planners [34] together with a global planner, e.g. a visibility

graph-based method as presented in [35].

3.2 Combinatorial Methods

Some of the first proposed algorithms to solve the path planning problem can be characterised as combinatorial methods. These methods usually revolve around building a *roadmap*. A roadmap, $\mathcal{M}$, is a graph spanning the current working environment. The roadmap can be defined as follows in definition 3.2.1:

Definition 3.2.1 Roadmap:
Let $\mathcal{M}$ be a graph mapping into C_{free} and let $\mathcal{S} \subset C_{free}$ represent the set of all reachable points in the graph. $\mathcal{M}$ is said to be a roadmap if the following conditions are satisfied:

- Accessibility - *There exists a path $r : [0,1] \mapsto C_{free}$ from $q \in C_{free}$ to some $s \in \mathcal{S}$ $\forall\, q, s$*
- Connectivity - *If there exists a path $r : [0,1] \mapsto C_{free}$ s.t. $r(0) = q_i$ and $r(1) = q_g$, then there also exists a path $r' : [0,1] \mapsto \mathcal{S}$ s.t. $r'(0) = s_1$ and $r'(1) = s_2$.*

From definition 3.2.1, the accessibility-condition implies its always possible to connect some initial state and a goal state to $s_1, s_2 \in \mathcal{S}$, respectively, whereas the connectivity-condition ensures algorithmic completeness by assuring no missed connections. This roadmap can be built in many different ways, with some of the most prominent methods including cell decomposition, reduced visibility graphs, and Voronoi diagrams. These methods are all techniques for generating a complete representation of the environment, but with different goals. Cell decomposition methods first execute a cell decomposition of the environment before extracting the roadmap, while the other two generate the roadmap directly. The reduced visibility graph provides a shortest-path roadmap [15], whereas the Voronoi diagram produces a maximum clearance roadmap [13]. Even though visibility graphs and cell decomposition have successfully been applied to solve the path planning problem in multiple studies — see for example [35]— the main focus in this report will be on Voronoi diagrams. This is mostly due to their clearance property, which is favourable especially in marine applications, considering the possibility of uncertainty in locating both oneself and obstacles. The primary downside to these methods is the large computational cost, especially for more complex environments. To reduce this cost, methods have been developed that probabilistically build a roadmap. This increase in efficiency and simplicity, however, leads to loss of algorithmic completeness. An example of such a method is the probabilistic roadmap method.

3.2.1 Maximum Clearance Planning

As mentioned, roadmap methods work by reducing the free C-space into a subset of connected lines. The following path search is then done on this resulting subspace. There are many ways of constructing such a roadmap, but one well-established method is the use of Voronoi diagrams. A Voronoi diagram partitions the environment into convex regions, or *Voronoi cells*. Each polygonal region has one *generating point* p_i, and every point inside a given cell is always closer to that cell's corresponding generating point than to any other generating point. This generated diagram supplies a maximum clearance roadmap to be used for safe path planning.

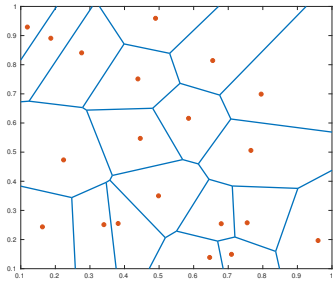


Figure 3.2: Simple example of a 2D Voronoi diagram, where the generator points are indicated in red.

Definition 3.2.2 Voronoi regions *[36]*
Let

$$d(x, \mathbf{A}) = \inf \{d(x, p) \mid p \in \mathbf{A}\}$$

denote the distance between the point x and the subset $\mathbf{A}$, where $p_i \in \mathcal{P}$ is a generator point. The Voronoi region is then defined as

$$\mathbf{R}_k = \left\{x \in X \mid d(x, \mathbf{P}_k) \leq d(x, \mathbf{P}_j) \, \forall \, j \neq k\right\} .$$

From definition 3.2.2, the final Voronoi diagram $\mathbf{V}(p_i)$ is then represented by the finite ordered list of cells $\mathbf{R}_k$. A simple example of a Voronoi diagram is shown in fig. 3.2. The standard Voronoi diagram can then be generated in $O(n \log n)$ time and updated in $O(n)$ [37]. One thing to note is that different representations can be achieved by changing the metric d. An example of such a diagram is the Möbius diagram, using a Möbius-based metric [38]. The shape of the resulting piecewise paths can then be constructed to best fit the given application. The L_2 metric will be used in this work, as the resulting paths will be piecewise linear, which have advantageous properties with respect to the robot's guidance system. An example of a Voronoi diagram of a more complex environment is shown in fig. 3.3. The resulting diagram where edges wholly or partly inside obstacles have been removed, and the total number of nodes have been reduced [1], are shown in fig. 3.4.

[1] These reduced Voronoi diagrams are often referred to as Generalised Voronoi Diagrams (GVD).

There exist several ways of constructing a Voronoi diagram, but an efficient algorithm is Fortune's algorithm [39] introduced in 1987, also known as the sweep-line algorithm. Once the diagram has been generated, it is very easy to calculate a path to a target node. The goal is simply added to the graph, by means of connecting nearest neighbour, before a heuristic search algorithm is applied. Using an algorithm such as A* results in very quick planning, as the Voronoi diagram is sparse, resulting in an efficient search.

One attractive property of the Voronoi diagram, and one of the main reasons for using it in path planning, comes from defining the obstacles as generator points. Doing that, the edges of each region is composed of the points that are the furthest from any obstacle. Calculating a path along the edges then results in a guaranteed safe path. For certain applications, it is worth noting that, albeit giving a safe path, it is almost always sub-optimal wrt. time and energy constraints. In addition, the path is piecewise linear, resulting in the need of path smoothing before being fed to the guidance controller [40].

3.3 Sampling-based Methods

The main motivation behind sampling-based methods was the computational burden of the combinatorial methods. Path planning systems needed to be able to run online, and to deal with hardware limitations, certain relaxations regarding the discrete C-space formulation had to be done. Representations of the C-space was then proposed done by random sampling of the environment, hence the name sampling-based methods.

3.3.1 The Probabilistic Roadmap Method

One of the most popular methods to be based on the sampling paradigm is the *probabilistic roadmap* method (PRM), introduced by Kavraki et al. [41], [42]. The PRM is divided into two phases: a preprocessing — or construction — phase, and a query phase. The preprocessing phase builds a graph by taking random samples and approximating feasible motions in C in which all nodes corresponds to certain collision-free configurations $q \in C_{\text{free}}$. A feasible, collision-free path between a set of nodes is then given by the graph edges. The edges themselves are calculated by using a local planner, e.g. a straight line planner as used in the original paper. The given initial and goal configuration of the robot is added to the constructed graph in the query phase. From there, the graph is searched to find a path connecting q_{init} and q_{goal}. This method randomly samples C and checks if the samples are in C_{free} before using a local planner to add edges between vertices and new samples. Vertices n in the created graph can be assigned a weight $w(n)$ that assigns a heuristic "risk" to the surrounding region in space. A larger $w(n)$

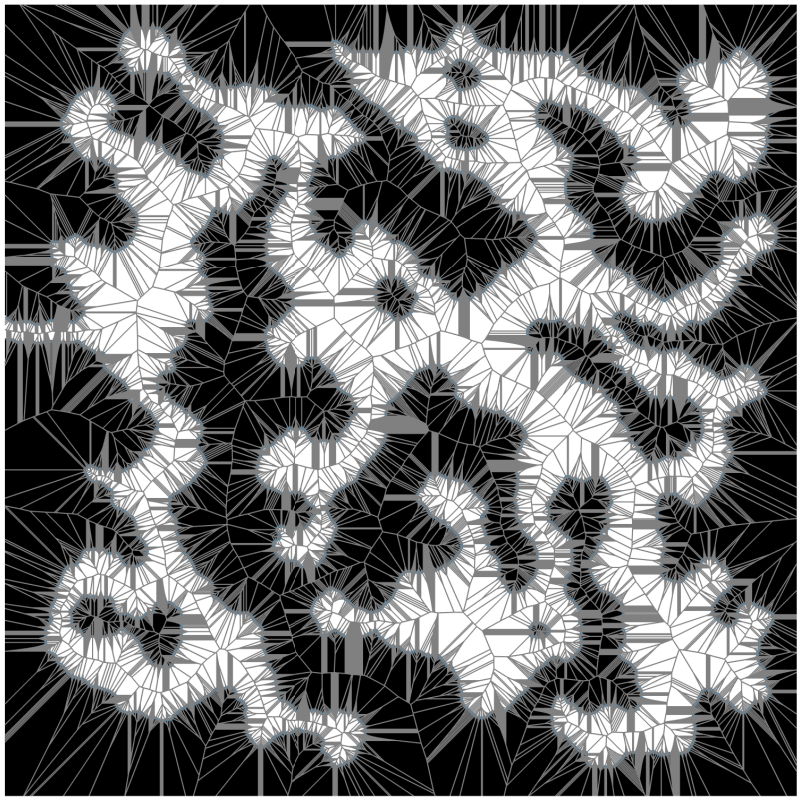


Figure 3.3: Voronoi diagram of a semi-closed environment where obstacle edges are used as generator points. Further removing of edges and nodes part of C_{obst} is needed.

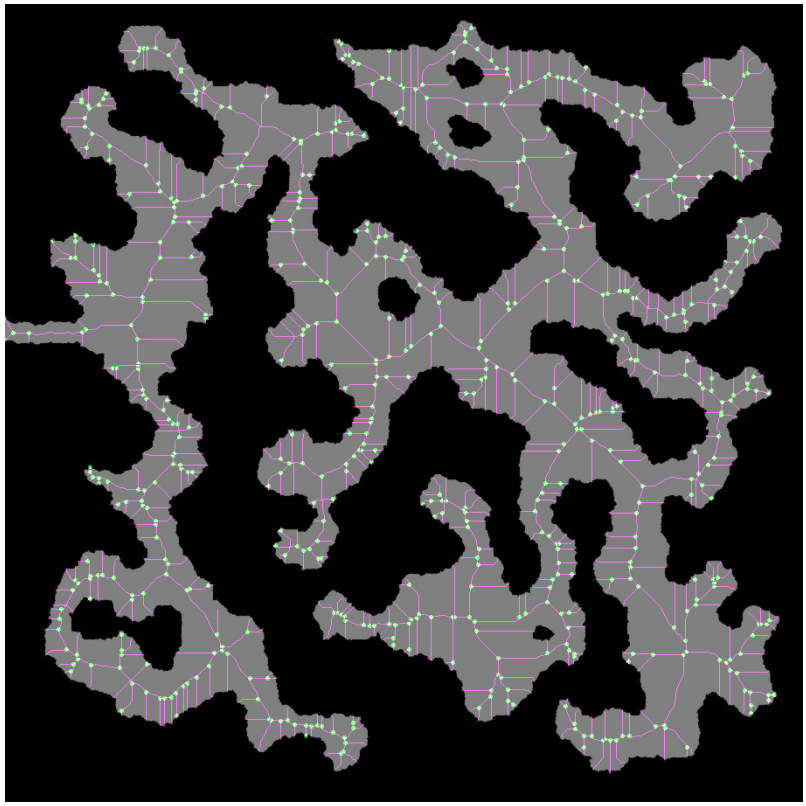


Figure 3.4: Example of the resulting graph from fig. 3.3 where edges and nodes part of C_{obst} have been removed. Green points indicate intersection points and the pink represent the leftover edges.

translates to a tougher region, which in turn increases the probability of the given vertex being chosen for expansion. This heuristic can, for example, be chosen as the distance from n to the closest sub-tree not containing n, or be based on connection failure from the local planner, i.e. increasing the weight if the local planner fails to expand from n multiple times. This *failure ratio* can be calculated as

$$\eta_{\mathrm{f}}(n) = \frac{N_{\mathrm{f}}(n)}{N_{\mathrm{c}}(n) + 1} \ , \tag{3.4}$$

where N_{f} is the number of failed expansion attempts from n and N_{c} is the number of times the planner tried to connect n to another vertex. The heuristic weights can then be calculated as

$$w(n) = \frac{\eta_{\mathrm{f}}(n)}{\sum_{v \in V} \eta_{\mathrm{f}}(v)} \ . \tag{3.5}$$

An example of the PRM in action is shown in fig. 3.5. PRM results in paths that are quite similar to that of the Voronoi diagram in fig. 3.3 and thus have the same need for path smoothing. The biggest difference here is the increased risk of the PRM-calculated path.

One disadvantage of the PRM method itself is that it works well for holonomic robots, but is not directly suitable for nonholonomic robots without modifications. One such addition was made in [43], where the roadmap essentially is built up from a set of local feedback controllers acting as the local planners. Another disadvantage with PRM is their intrinsic problem with narrow passages between obstacles. Again, modifications have been done to address this. A variant, called *obstacle-based PRM* (OBPRM) [44], chooses points from C_{obst} to expand the roadmap. After choosing a point $p \in C_{\mathrm{obst}}$ the algorithm moves in a random direction until it finds C_{free}. This strategy results in both the ability to handle more narrow spaces, but also a graph with much fewer nodes than the base PRM. Probably the biggest drawback of the PRM, however, is the fact that it is not guaranteed to be completely connected. This means that parts of the map might end up being inaccessible since the roadmap is incapable of covering the whole C_{free}. This is apparent in fig. 3.5, where the upper left corner, around (100,800), and the whole bottom left part of the map, around (200,400) and downwards, is left inaccessible due to the incompleteness of the generated map. This can, of course, be fixed by lowering the connection distance, but at the cost of an increased number of computations. Furthermore, there have been published proofs that, under certain conditions, PRM-based planners are guaranteed to fail [45]. This is important to note if one wants to guarantee a robust path planning system.

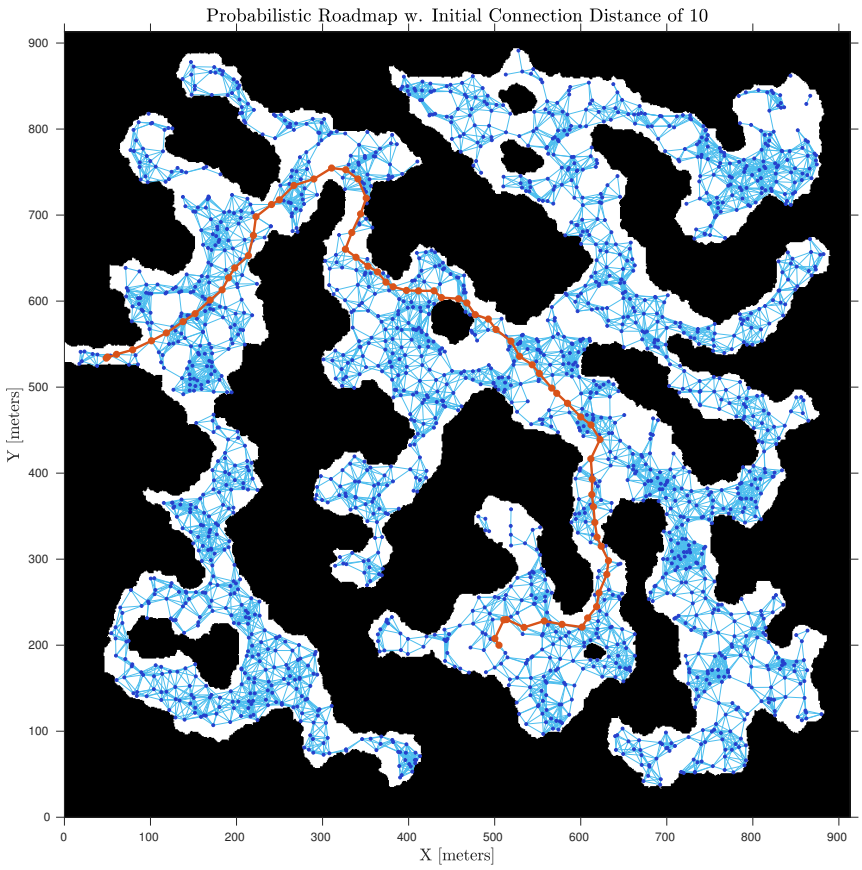


Figure 3.5: Probabilistic roadmap of a semi-closed 2D environment at the resulting path to the exit. The initial connection distance between nodes was set to 10, with the initial number of nodes at 500.

3.3.2 Rapidly Exploring Random Trees

Another sampling-based method that has seen much use in recent literature is the *rapidly exploring random tree* (RRT) method. RRTs is essentially a data structure that is built by randomly adding leaf nodes to grow the tree into C_{free}. In a way, the RRT expansion is biased towards the largest Voronoi regions in C [46]. The underlying concepts of the RRT are relatively simple, leading to a huge amount of offspring variants being developed.

The RRT algorithm works by initialising the initial robot pose $x_i \in X_{\text{free}}$ as the root node[2]. After initialisation, the tree is continuously expanded by randomly sampling the state space. New nodes $x_{\text{new}} \in X$ are added to the tree iff. $x_{\text{new}} \in X_{\text{free}}$ and the edge connecting x_{new} and the existing tree is collision-free. Sampling the state space can be done either by randomly choosing points and connecting them with straight lines, or by *steering* the search. This steering function, shown on line 5 in algorithm 1, chooses the input u that results in the state $x' \in X_{\text{free}}$, using eq. (2.1), that is closest to x_{new}. x' is then added to the tree, given a collision-free path. Using this steering function, the vehicle dynamics can be included in the tree expansion, resulting in nodes the vehicle actually can reach by applying the input u. By nature of the performed expansion, the RRT algorithm is biased towards unexplored parts of C [46]. An addition that can be made to this method to force the expansion of the tree towards the goal is to use the goal position as the sample at certain intervals.

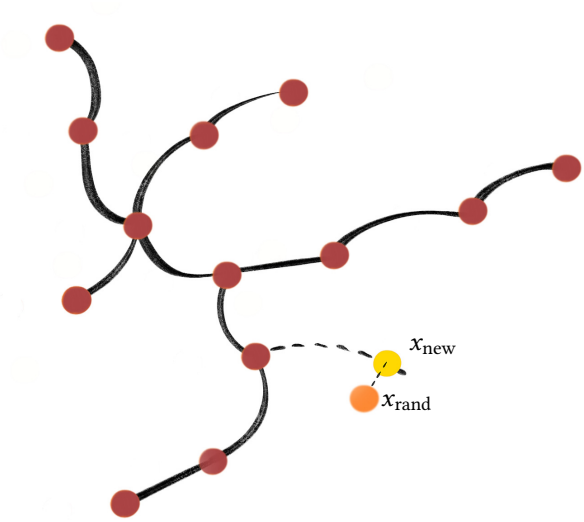


Figure 3.6: Illustration of the RRT expansion process. The random sample x_{rand} (orange) is found by following the steering function. The new node x_{new} (yellow) is found to be the point along the steered line closest to the random sample. This node is then added to the existing tree.

A nice feature of the resulting graph, is that it will always be connected, which is not always the case for the PRM method. Another advantage of this method, using this state space formulation, is the possibility to easily incorporate the dynamics of the specific robot in question, resulting in trees that conform to the given kinodynamic constraints. One drawback however, which holds for all randomised planning algorithms, is that the ideal metric ρ is not

[2] $X_{\text{free}} = C_{\text{free}}$ for basic path planning.

Algorithm 1 GenerateRRT (x_i, N)

Input: Initial state x_i, num. vertices in RRT N
Output: RRT graph G(V, E)

G.Initialize(x_i) ▷ x_i as root, edges are empty
for n = 1 to N **do**
    x_{rand} = RandomState( )
    x_{nearest} = NearestNode(x_{rand}, G)
    x_{new}, u = Steer(x_{near}, x_{rand})
    if $x_{\text{new}} \in \mathcal{X}_{\text{free}}$ **then**
        G.AddVertex(x_{new})
        G.AddEdge(x_{near}, x_{new})
return G

easily estimated [47]. Different metrics and their requirements are discussed more in-depth in [13].

Comparing fig. 3.5 and fig. 3.7, it can be seen that the resulting data structure representing the environment is much larger for the PRM method as opposed to the RRT method. This is largely due to the fact that the PRM algorithm builds the whole graph before finding the path, whereas the RRT algorithm continuously builds towards the goal, stopping when the goal is reached. This means that the ideal method-of-choice will be dependent on their application. In general though, a tree structure is easier to maintain than a graph with respect to memory requirements. This argument also holds for the graph-version of the RRT, the *rapidly-exploring random graph* (RRG) [48].

The introduction of the RRT algorithm further sparked numerous offspring algorithms. The main goal of these was to increase efficiency, increase optimality, and in general solve some of the mentioned problems - e.g. the problem of narrow passages. As this list is growing rather large, only a few of them will be discussed here, namely the *RRT**, the *bi-directional RRT*, and the *RRT*FND*. The bi-directional RRT is a simple extension that alternates between expanding the tree rooted at the start node and the tree rooted at the goal node. This has been shown through experiments to often be more efficient in practice [16]. A simple extension such as this helped solve almost-closed environment problems, such as the bug-trap problem, by expanding one tree from the inside and one from the outside.

RRT* works very similarly to the basic RRT and builds on the briefly mentioned RRG, but converges to the optimal solution by adding two main features: a near-neighbour search and tree-rewiring. The near-neighbour search works by defining a spherical area around the new node and finding the best parent node within this area. The radius ρ of this area is determined

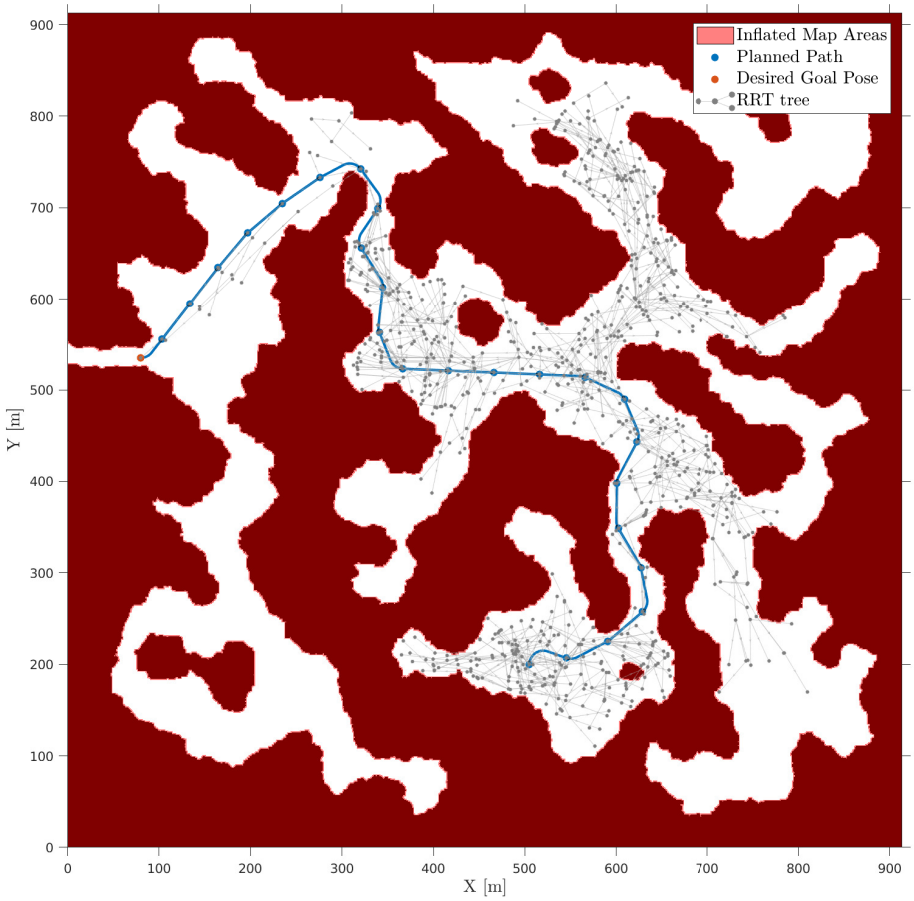


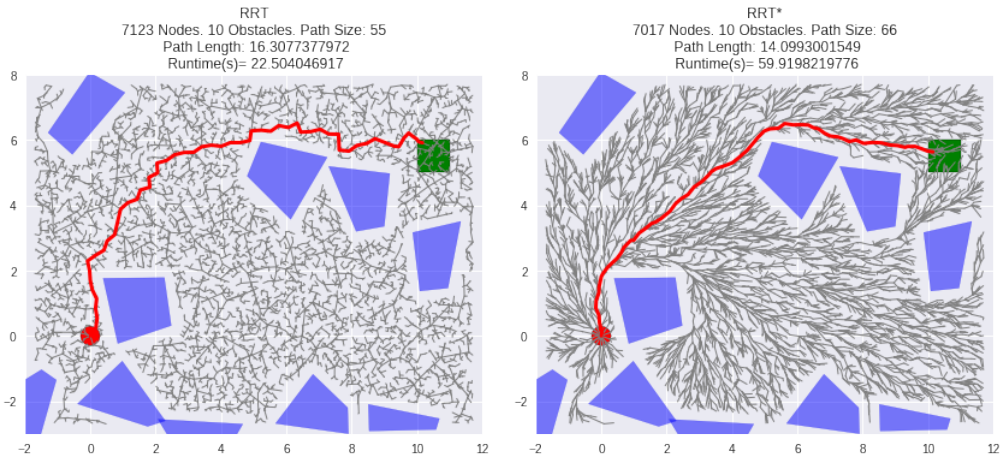
Figure 3.7: Example path found by the basic RRT algorithm together with the resulting tree. Solved using MATLAB and its costmap implementation using a minimum turning radius of 15 m, Dubins path interpolation, and a bias towards the goal pose.

by:

$$\rho = \gamma \left(\frac{\log n}{n} \right)^{\frac{1}{d}} , \tag{3.6}$$

where n is the number of nodes in the tree, d is the spatial dimension and γ is a parameter based on the specific characteristics of the environment [48]. Searching through this circular neighbourhood is similar to how the RRG algorithm and the RRT*FND includes new nodes,

and is illustrated in fig. 3.9. The rewiring happens within the same radius, updating the edges between the encompassed nodes to minimize edge weights between nodes, similar to what is shown in fig. 3.9. These additions results in a path that in general are much less rough due to the local optimisations done underway. A further extension of this include the removal of redundant nodes, leading to a straighter path. These improvements in path quality included in the RRT* leads to a substantial increase in runtime; from tests the runtime was roughly tripled, leading to a weighting between optimality and computational efficiency. The



(a) Example path from RRT in a relatively simple environment.

(b) RRT* obtaining a shorter and smoother path than RRT due to the local optimisations. RRT* used approximately three times as long to calculate the final path.

Figure 3.8: Comparison of RRT and RRT* implemented in Python, credit to GitHub-user yrouben. Experiments were run using Google Colaboratory.

main disadvantages of the RRT* is precisely the mentioned increase computational efficiency. Another thing to keep in mind is that neither RRT nor RRT* revisit the already built tree when traversing to check for any new obstacles, which can occur in a dynamic environment. Furthermore, RRT and RRT* also tend to generate a rather large number of nodes. An attempt was made to address this, reducing the linearly increasing memory usage of the RRT* to the fixed memory usage of the introduced *RRT*Fixed-Nodes* (RRT*FN) algorithm [49]. A very recent algorithm that addresses the memory usage, the path optimality and the issue of dynamic environments, is the dynamic RRT*FN (RRT*FND) [50], building on the RRT*FN algorithm.

In the growth phase, the RRT*FND algorithm builds the tree structure identically as RRT until the number of nodes reaches a set maximum allowed number. At this point the *forced removal* activates, removing leaf nodes as long as the leaf node in question is not the last node

on the solution path. When the initial solution is found the algorithm continues to build the tree until the robot initiates movement. Following this, the algorithm updates information on any obstacles at every node the robot reaches. Robot motion is then stopped if a collision is detected with any segments between current node and the goal. The planning stage is then rerun. The already visited nodes of the tree, as well as the nodes colliding with the obstacle(s) are removed. RRT*FND will then try to connect the rest of the tree with the goal state, possibly regrowing the current tree basing these steps on greedy heuristics. The simulations performed in [50] show that the RRT*FN algorithm outperforms RRT* in both runtime and the success rate of finding a path in a dynamic environment. While RRT*FND gives significant improvements in runtimes for complex environments, it does, however, only obtain marginally faster runtimes in simpler environments — even at times having a slower runtime due to initial overhead.

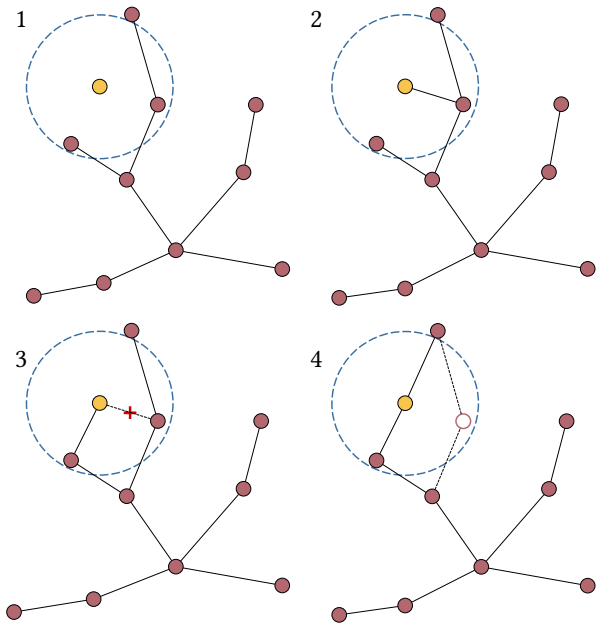


Figure 3.9: Illustration of the initial tree expansion of the RRT*FND algorithm. **1)** A CIRCULAR AREA encompassing the NEW NODE is searched. **2)** The NEW NODE is added to the EXISTING TREE. **3) & 4)** The rewiring is done by optimising the edges connecting the nodes, before any child nodes connecting a path with a higher cumulative cost is removed.

3.4 Graph Search Algorithms

In 2D path planning, heuristic graph search algorithms have long been the go-to method for a vast number of applications. This is partly due to the ease of which the path planning problem can be represented using a graph. One of the most popular environment representations for path planning is the occupancy grid. With this representation as a basis, it becomes simple to implement graph search algorithms for the sake of path planning, since each cell in the grid can be represented as a node One of the most used search algorithms is the A* algorithm. A* is an extension of Dijkstra's famous algorithm. It calculates an optimal path wrt. a cost function between a start and a goal node. The cost can represent different optimality conditions, e.g. path length or energy consumption. A* differs from Dijkstra in that it includes a heuristic that, for a given node, estimates the cost from that node to the goal. For completeness, the pseudocode for A* is shown in algorithm B.1.

Due to its simplicity and performance, A* has seen much use in path planning. From its popularity, it has received a number of extensions such as *lifelong planning* A* (LPA*), seeking to reduce the cost of replanning, and the weighted A*, sacrificing optimality for a faster runtime. The biggest drawback of A* is the above-mentioned computationally expensive replanning in the case of a dynamic environment. There are many proposed ways of handling this, but the two main successors that will be discussed here is the *dynamic A** and *D*-Lite* algorithms.

3.4.1 Dynamic A*

Dynamic A*, from now on referred to as D*, seeks to reduce the A* cost of replanning when planning in dynamic environments, while still keeping the optimality and completeness properties of A*. D* was introduced by A. Stentz in 1993 [51]. Differing from A*, the D* algorithm performs its graph search backwards from the goal. Each found node then contains a back-pointer to the next node as well as the cost from that node to the goal. The algorithm completes when the initial node is reached. If an obstacle is encountered along the way, all nodes that are affected by the new information are placed in the OpenSet, which have the same functionality as in A* (algorithm B.1). The directly affected nodes' weights are then modified to account for the obstacle information, before a new path is calculated.

D* became the go-to search algorithm for robots in dynamic environments after its introduction. It has since been more or less totally replaced by another algorithm, namely *D*-Lite*, with even the creators of D* transitioning to using D*-Lite. This is mainly due to the fact that the D* implementation was somewhat complicated and that the D*-Lite algorithm usually is more efficient.

3.4.2 D*-Lite

The D* algorithm has shown good results in dynamic environments, as mentioned. One of the main problems is that it is somewhat complex to implement. To account for this, another algorithm was developed based on LPA*, namely the D*-Lite algorithm [52]. This algorithm is, in essence, simpler and can be implemented with fewer lines of code. D*-Lite works in much the same way as the D* variant *Focussed D** [53], in that a heuristic is used to focus the search. The non-optimised version of D*-Lite based on heaps is - for completeness - shown in algorithm B.2. Alternatively, see [52] for the optimised version.

The algorithm uses $g(v)$ to represent the estimate of the starting distances for a node, similar to the $g-$value in algorithm B.1. Furthermore, it defines rhs(v) as alook-ahead value based on $g(v)$, which is defined as

$$\text{rhs}(v) = \begin{cases} 0 & \text{if } v = v_{\text{start}} \\ \min\limits_{v' \in v.\pi} \{g(v') + c(v', v)\} & \text{otherwise}, \end{cases} \tag{3.7}$$

where $v.\pi$ represents the predecessors of node v. An important note here, is that, since D*-Lite begins at the goal node, the defined predecessors corresponds to successor nodes in an A* search.

rhs$(\cdot)$ then checks the value of a node one step ahead. If rhs$(v) = g(v)$, then v is said to be *consistent*. D*-Lite keeps all inconsistent nodes in what corresponds to the OpenSet, where they are further refined.

Initially, only the goal is inconsistent. The algorithm then goes through each predecessor, checking for inconsistencies and updating rhs and g values until the start point is found. If at any point during the execution of the found path and obstacle is encountered, the algorithm engages replanning. The replanning stage checks all edges connecting the newly affected cells until a new path is found. D*-Lite then performs the replanning much faster than compared to A*, as it only considers the locally affected areas. Similarly to A*, D*-Lite can easily include a heuristic h to guide the search based on some critera.

3.5 Summary

Probabilistic planning have long been the go-to methods for online path planning, largely due to their simplicity and efficiency. RRT and its variants seem to be the most promising to test in three-dimensional planning, much due to their straightforward implementation and the ease of which vehicle dynamics can be included.

For planning based on graph searches using grid maps, it is worth noting that they suffer from what is known as *resolution completeness*. This refers to the fact that the calculated path's optimality is constrained by the grid's ability to capture the environment, i.e. the grid resolution. Thus the path will only be optimal if the grid itself is optimal, and one would need to be conscious of the grid choice.

Search algorithms could instead be applied to an already generated graph, e.g. using the Voronoi method. The big advantage then is that the Voronoi diagram is a *sparse* graph, resulting in the possibility of a very efficient search. An efficient global planner could then be implemented by combining the Voronoi diagram with an A* search. Experiments show that searching through a sparse graph representation of a complex environment using A* can be several hundred times more efficient than searching through a grid representation of the same environment or by doing random sampling through RRT* [54]. This then seems as a very promising way of designing the path planning system, as long as the sparse graph construction is performed efficiently enough.

When it comes to choice of search algorithm, the two main alternatives are A* and D*-Lite. Experiments performed to compare search algorithms indicate that A* is the preferred choice in simple environments or when the environment is sparse — as is the case with a Voronoi graph —. In most other cases, the D*-Lite triumphs over A* when looking at run-times, with D*-Lite also appearing to give better results when more than one optimal path exists [55]. For these reasons, A* could then be used as part of a global planning system on a global graph representation, whereas D*-Lite could be implemented as part of a denser, grid-based local planner, possibly taking ocean currents and other disturbances into account. It is important to note, however, that both the A* and the D*-Lite converges to the optimal solution.

The next chapter introduces some of the problems encountered when extending the path planning to three dimensions. A few more state-of-the-art sampling-based planning methods are discussed before methods using 3D graphs for planning are more closely studied.

4 | Path Planning in 3D

An increasing number of modern robotic applications are developed with the intention of operating in more complex and higher dimensional environments. Because of this, the need for higher-dimensional planning systems only grows. Planning in a three-dimensional environment quickly becomes much more difficult and computationally expensive compared to its two-dimensional counterpart. Section 3 introduced a set of methods used for the two-dimensional case. Most of these methods can be extended to three dimensions, but there are a few aspects that need to be considered more thoroughly, namely the environment representation. The environment representation has a huge impact on the efficiency of the planning system, and must, therefore, be chosen carefully.

Path planning can loosely be described as a two-part problem: (*i*) modelling the environment (*ii*) calculating the path through the constructed representation under the given constraints. The next section introduces some of the most used approaches for representing the 3D environment for use in a path planning system.

4.1 Representing the Environment

Accurately describing the world the robot operates in is key to the path planning process. The way the world is represented will have a big impact on the performance as well as on the quality of the calculated path. In general, the path planning will be more efficient if there are fewer nodes in the map and more accurate if the given nodes match more closely to the world itself. There are many different ways of representing the environment, but broadly speaking they can be categorised as *topological* or *metric*[1]. A topological representation describes the environment without explicit references to numeric data. I.e. it models the environment as a set of nodes, possibly describing features, with edges between them containing relational information. Metric mapping, on the other hand, directly utilises a certain data structure in which waypoints can be explicitly stated based on global data. Metric methods tend to favour optimal path-methods and have become the most popular.

Numerous representations exist, so this paper will focus on a few of the most used metric techniques. The next section will introduce the most popular representations, namely the

[1] Commonly also referred to as qualitative and quantitative, respectively.

Voronoi diagram, the *occupancy grid*, and the *octree*. The rest of the chapter will discuss three-dimensional path planning methods.

4.1.1 Voronoi diagrams

Voronoi diagrams, as introduced in section 3.2.1, create a graph in which paths maximally distant from obstacles can be calculated. This results in very safe paths, which is preferable for autonomous underwater operations. Algorithms have been developed that generate $d-$dimensional Voronoi diagrams based on convex hulls in $O(n \log r)$ time for $d \leq 3$ [56], where r is the number of already processed points in the diagram. This allows efficient generation as long as n does not grow too large. An example of a three-dimensional Voronoi diagram is shown in fig. 4.1. For mapping and exploration in three dimensions, however, the number of points can be expected to grow quite large. Voronoi diagrams representing $d-$dimensional space require $O(n^{\lceil \frac{1}{2} d \rceil})$ storage space. This requirement is linear in the 2D case, but exhibit polynomial growth for $d \geq 3$. Reducing the diagram is, therefore, of interest.

The reduction — or simplification — of Voronoi diagrams, is widely used in path planning. These diagrams are known as *generalised Voronoi diagrams* (GVD) [2]. An example of a GVD is shown in fig. 3.3. Simplifying the diagrams in this manner is advantageous in path planning, as it results in fewer nodes to be searched through while retaining the maximum clearance property for safe planning. Even further reduction is possible by a process known as *skeletonisation*, reducing the graph to a very sparse representation. Planning over the skeleton graph can result in very efficient path planning [54], but requires efficiently processing sensor data and the possibility of incremental graph generation.

4.1.2 Occupancy grids

Occupancy grids are perhaps the most used representation used in robot path planning to date. lergely due to their simplicity. An occupancy grid structures the environment into a discrete grid with a specified resolution. Each cell in the grid is then given some value depending on whether it is *free, occupied* or *unknown*. This occupancy information can be preset by some a priori knowledge — e.g. from a pre-built map — or be determined from real-time sensor information, like in a SLAM-setting. The occupancy is usually represented in one of two ways: (*i*) binary (*ii*) probabilistic. Binary grids simply assign each cell a *true* or *false* value depending on whether or not it is occupied by some obstacle[3] The probabilistic map is slightly more complex and is the most common occupancy representation. This representation assigns

[2] GVDs are also known as the discretised variant of the *medial axis*, also referred to as a topological skeleton.
[3] A cell with a value of *true* is usually determined to be occupied.

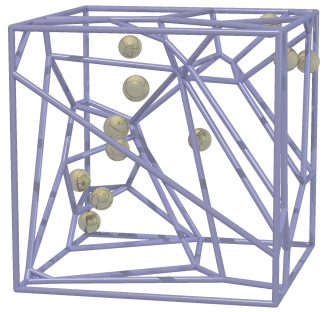


Figure 4.1: A three-dimensional Voronoi diagram created from points inside a cubic region. Generated using the VORO++ library and rendered using POV-RAY.

occupancy values from calculated probabilities, with probabilities close to 1 indicating a high certainty for a cell being occupied. Cells with probabilities close to 0.5 are often referred to as *unknown* cells. In a mapping setting, these occupancy probabilities are calculated as

$$p(m_i \mid z_{1:t}, x_{1:t}) = \prod_i p(m_i \mid z_{1:t}, x_{1:t}) \ ,$$

where $p(m_i)$ denotes the probability of a specific cell i being occupied, $z_{1:t}$ being the set of measurements up until time t and $x_{1:t}$ is the set of robot configurations or states up until time t. Figure 5.1 shows an example of a probabilistic occupancy grid, where light gray cells represent free space, dark gray cells indicate unknown space and black cells illustrates obstacles.

Grid representations are not without disadvantages, however, with the main drawbacks being the denseness of the representation and their problems with *resolution completeness*. The denseness simply results in a larger number of nodes having to be evaluated during a search compared to e.g. sparse graphs. Due to this, grids are better for smaller environments or for applications where a denser set of information is needed. Resolution completeness, or digitisation bias, refers to the errors of quantising the real world as grids. In other words, cells including just a sliver of an obstacle are marked as occupied, even though the majority of the cell is free space. This means the optimality of the calculated path, for example with respect to path length, is dependent on the grid resolution. Occupancy grids have traditionally been represented as regular grids, but the problem of resolution completeness can be overcome to a certain degree by instead representing using octrees.

4.1.3 Octrees

Octrees is a data structure that is commonly used in search and optimisation algorithms as well as in image processing. Octrees are most easily explained through their two-dimensional equivalent: quadtrees. Quadtrees structure the data by dividing a square portion into four smaller squares, hence quadtrees, and recursively subdividing into four quadrants. In this representation, more "interesting" parts of the environment have a deeper subdivision. E.g. an image-based map with clustered obstacles can result in some parts of the quadtree being very deep with each sub-cell representing one pixel, whereas parts of the map with no obstacles might be represented by larger sub-cells of four pixels, as illustrated in fig. 4.2. Quadtrees are mainly used to partition two-dimensional space, e.g. to use as 2D navigation maps for marine surface vehicles (MSVs). To accommodate the third dimension, the octree encoding was developed [57]. This data structure is analogous to the quadtree but instead partitions the three-dimensional space into octants. The main advantage of these structures compared to the regular grid is that the problem of too crude a quantisation is reduced due to the increased resolution of areas in or close to C_{obst}. As a result, the quad-/octrees generally consists of fewer cells than their grid counterparts. A comparison between a 2D occupancy grid and a quadtree representation is shown in fig. 4.2. The occupancy grid is the same one as shown in fig. 5.1.

4.2 Sampling-based 3D Planning

As tried exemplified in section 4.1, the way the environment is represented is highly dependent on the specific application. This also holds for the planning algorithm, even more so for 3D than for the 2D case. Due to this, the methods presented in section 3.1 will be evaluated with their applicability towards the case of an exploratory AUV in a dynamic and unknown environment.

Much of modern research on 3D path planning is based on numerical optimisation methods [58] which can give very good results, but tend to be very computationally expensive. In turn, this leads to offline planning methods. Work has been done to increase the computational efficiency of numerical methods such as to facilitate online planning. This does, however, result in loss of global optimality Due to this, state-of-the-art methods today are usually sampling-based. Most of the discussed methods in this section will, therefore, fall into this category.

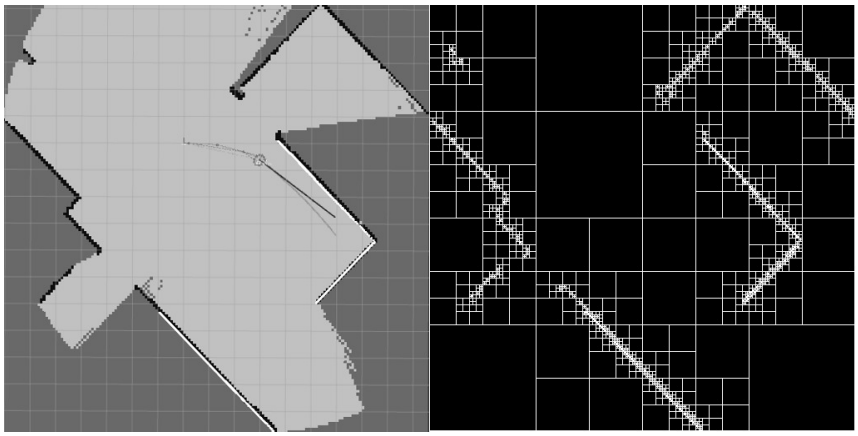


Figure 4.2: Comparison between an occupancy grid and the correseponding quadtree representation. Empty areas results in larger, empty cells, giving a more compact representation wrt. storage. *Left*: Screenshot from a 2D SLAM simulation based on a 2D occupancy grid. *Right*: The same screenshot, but here represented using a quadtree. (The grid overlay in the left image is not indicative of the occupancy grid resolution.)

4.2.1 The RRT Family

Path planning for more constrained vehicles such as an AUV or fixed-wing UAV imposes certain constraints on the quality and smoothness of the calculated path. This becomes one of the main problems of sampling-based planners such as those part of the RRT-family. Methods like the RRT* and its variants have tried to partly overcome this by producing more natural looking paths. The calculated path can then be further smoothed by using techniques such as Dubin's paths, Bezier curves or Fermat's spirals.

The Spline-RRT* (SRRT*) method [59] tries to include the smoothing in the planning stage such as to avoid costly post-processing. This method includes a Bézier curve-based spline method to calculate a smooth path for a UAV in a 3D environment. Being based on the RRT*, this method results in an asymptotically cost-optimal path that satisfies the constraints imposed by the environment and the vehicle dynamics. The main drawback of this method is it's slow runtime, resulting in it only being useful as an offline planner. This high computational cost is due to its RRT* basis, which doesn't cap the number of nodes, leading to a very large tree, and its extra computations when rewiring the tree. For methods based on RRT*, this drawback must be handled to facilitate online planning, as this leads to slow convergence as well as heavy memory usage.

In other relatively recent methods, such as the RABIT*, RRT*i, DT-RRT, and CARRT*, these drawbacks have been tried addressed by using different sampling strategies such as *two-stage sampling*[59], *uniform sampling* and *goal-biased sampling* [60]. This improved both the convergence and the memory requirements, but they are still in need of improvement to guarantee robust online planners. One interesting concept to include is the modifications added to the RRT*FN/RRT*FND algorithm discussed in section 3.1, where the maximum number of nodes are kept constant allowing for a smaller memory requirement. It would, therefore, be of interest to extend and test the RRT*FND in three-dimensional environments.

Another very recently proposed method is the *RRT*-Normal* (RRT*N) [61]. This method seeks to reduce the computational time by directing the state space sampling by using a normal distribution, hence RRT*-Normal. This is done by only generating points within a neighbourhood of the straight line between q_i and q_g, L as specified by a specified standard deviation. This standard deviation is suggested to be based on the distance between q_i and q_g or on the size of the largest intruding obstacle projected on the normal of L. Using this constraint on the tree expansion leads to a much lower number of total nodes, while also focusing the search towards the goal. Conducted simulations in [61] indicate that this method is roughly three times faster than RRT* and gives shorter paths while also having a much higher success rate. The original authors are currently working on extending this method to

three dimensions, and it would be of great interest to see in practice.

The quality of the path itself is still of concern. Due to this, a post-processing step is likely needed. This step usually consists of pruning and smoothing. For an underactuated system such as an AUV, and in general, it is preferred to reduce the number of unnecessary course changes caused by redundant waypoints or nodes. Because of this, pruning is usually performed. Similarly, to make sure the path supports the dynamics of the robot, smoothing is applied. This can be done in many ways, e.g. using a C^2-continuous clamped B-spline method [62] or 3D Dubins paths [63] (see section 5.2 for a more detailed discussion of smoothing techniques). This post-processing step needs to be efficient as to support frequent updates for an online planner.

In section 3.3.2, the importance of the correct choice of metric was briefly mentioned. For nonholonomic robots in 3D, for example, the Euclidean metric fails to completely capture the ideal cost of a specific node. Taking the SRRT* as an example. The utilised spline-based metric provides a fairly good path, but results in unsatisfactory runtimes with simulations exceeding several minutes. Choosing the right metric is thus essential as they can limit the optimality of the path as well as the total planning time [64].

4.3 Combinatorial 3D Planning

Much of modern planning research concern rapid mobile robots in complex settings, such as a UAV in a cluttered environment. In such cases, combinatorial methods are usually impractical due to the computational cost of generating the environment representations. For an underwater robot in a much more sparse environment, however, such methods might just be applicable, if not preferred. As mentioned in section 3.2, there are several different ways of representing C to facilitate a complete planning algorithm. These include visibility graphs, cell decompositions, shortest-path roadmaps and Voronoi graphs. Due to some of the disadvantages related to cell decomposition methods, e.g. combinatorial explosion, limited granularity and infeasible solutions [65], they will not be discussed further in this paper.

Underwater applications have, as mentioned, traditionally had path safety as the main priority, meaning that the path should always keep good clearance to all detected obstacles. This clearance is automatically guaranteed by the Voronoi representation, leading to a very promising planning method for underwater operation. Due to this, together with the relative computational efficiency of the Voronoi diagram , the Voronoi diagram is deemed the best method for underwater path planning. Furthermore, recent research shows that Voronoi diagrams perform well for marine settings [36], [66].

Recent articles showcase the practicality of Voronoi diagrams for marine surface operations

[36], [66] as well as its feasibility for use in three-dimensional underwater autonomous applications [63]. This proposed method uses a Voronoi diagram representation of C calculated using the quickhull algorithm [56], where the generator points are points along the bounding boxes placed around obstacles. To limit the generated Voronoi diagram to the region of interest, a set of random vertices are added along the edges of C_{free}. They then apply the Yen-modification of Dijkstra [67] to search for the K optimal paths. Thus, if the most optimal path is found to be inaccessible, the next optimal path can be used. Path segments are further smoothed by the use of 3D Dubin paths. The local planner incorporated is also based on a Voronoi diagram, but this local diagram is calculated inside a local subspace in which the detected moving obstacle resides. This subspace is calculated based on the observed and estimated obstacle positions and is based on the *closest time of approach* (t_C) and *closest point of approach* (d_C), defined as [63]

$$t_C = \frac{(\mathbf{p}_R - \mathbf{p}_O) \cdot (\mathbf{v}_R - \mathbf{v}_O)}{\|\mathbf{v}_R - \mathbf{v}_O\|}$$
$$d_C = \|(\mathbf{p}_R + \mathbf{v}_R t_C) - (\mathbf{p}_O + \mathbf{v}_O t_C)\| \quad .$$

Here $\mathbf{p}_R$, $\mathbf{p}_O$ refers to the position vectors of the robot and the obstacle respectively and $\mathbf{v}_R$, $\mathbf{v}_O$ refers to the velocity vectors of the robot and the obstacle respectively. After defining the obstacle subspace, the new collision-free path is calculated with the first and last vertex being the intersections between the original path and the replanning subspace.

The method outlined in [63] shows the applicability of a Voronoi-based planner in a 3D underwater environment. An important note regarding this implementation, however, is that they assume all obstacles known and static (with the exception of the one moving obstacle). In an exploratory setting, this is not known. Placing bounding boxes around detected while exploring is also likely not feasible, as this would require efficient object detection techniques applicable to point clouds. It is, therefore, fair to say that this assumption does not hold for the intended use-case. When obstacle positions are unknown, it is necessary to estimate their placement online based on the sensor or SLAM data. One way of doing this could be by using a clustering algorithm, such as DBSCAN [68], to estimate point cloud clusters. From these clusters a minimum bounding polygon could be created using a convex hull algorithm [69]. The polygon's defining points could then be used as Voronoi generator points, such as in [63], before nodes and edges inside the polygon can be removed from the resulting Voronoi diagram. An example of running DBSCAN clustering on a point cloud from the KITTI dataset (fig. 4.3) is shown in figure fig. 4.4 and a corresponding bounding box around a single cluster is shown in fig. 4.5. The results are not perfect and the clustering parameters needs tuning based on the intended application and environment, but it illustrates the applicability of DBSCAN combined

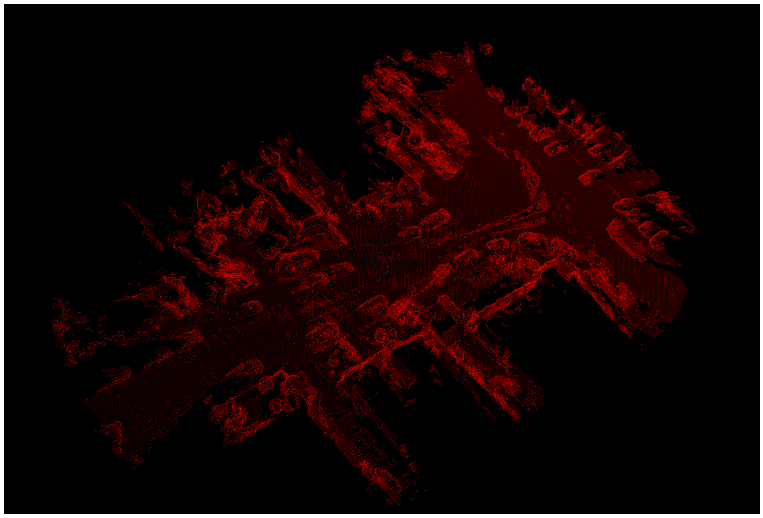


Figure 4.3: Real-world point cloud from the KITTI dataset [70].

with a convex hull algorithm for roadmap generation. Another way of generating the Voronoi diagram, that is based on the concept of skeletonisation introduced in section 4.1.1, is by way of signed distance fields [54].

In a complete system, the input to the path planner is the SLAM-map. SLAM algorithms can be characterised as *dense* or *sparse* based on the characteristics of this generated map [7]. Under the assumption of a sparse SLAM-map, the generated map is essentially represented as a sparse point cloud. Since no such thing as perfect sensors exists, especially not in underwater settings, this map is created from noisy data, giving rise to uncertainties. Directly generating a Voronoi diagram from this 3D point cloud could then result in a large number of nodes having edges colliding with C_{obst} between them. A method that might be applicable to handle this is based around the generation of a GVD-based skeleton graph by incrementally calculating the *Euclidean signed distance field* (ESDF) [54], [71]. The initial step in doing this, is to first calculate the *truncated signed distance field* (TSDF) and from this estimate the ESDF.

Distance fields is essentially a description of the inter-surface space. Each point in a distance field contains the distance from that point to the closest surface. TSDFs are based on the projective distance between a point and an obstacle, i.e. the relative distance between the sensor and the obstacle along the sensor's line-of-sight (LOS) [72]. Compared to a full distance field, TSDFs are only based on partial observations. Thus, instead of building the complete field, it is focused inside small positive and negative distances around the surface. These distances are called the *truncation radius*, and describes the maximum and minimum values

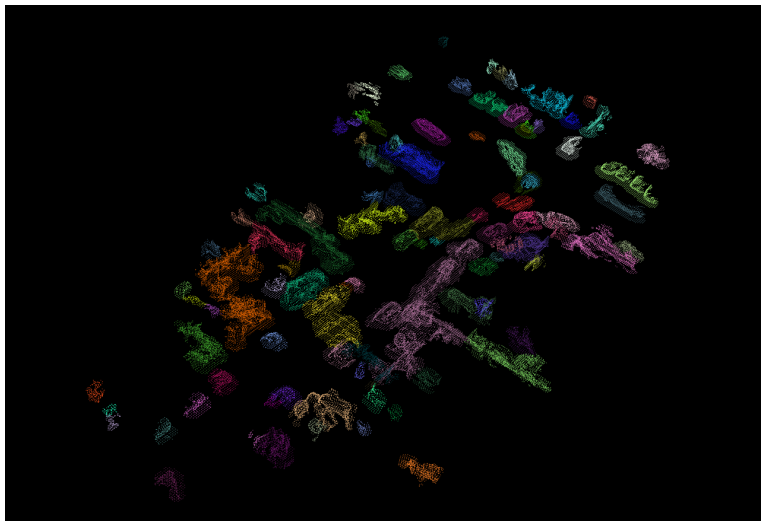


Figure 4.4: Result of a DBSCAN run on the point cloud shown in fig. 4.3. Output is generated by using Python bindings for the PCL C++ library and the scikit-learn Python library.

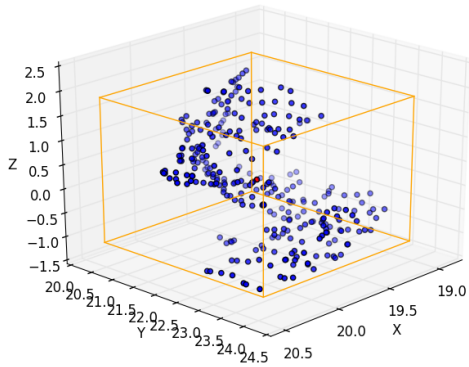


Figure 4.5: Example of a bounding box around a single DBSCAN cluster from fig. 4.4. More sophisticated results can be obtained by using e.g. Chan's algorithm [69] or the Quickhull algorithm [56]. The estimated cluster centre is marked in red.

of the distance field. Since the TSDF only operates within the truncated area it is relatively efficient to calculate. A simple example illustrating the TSDF is shown in section 4.3. A more complete example on a point cloud is shown in fig. C.2.

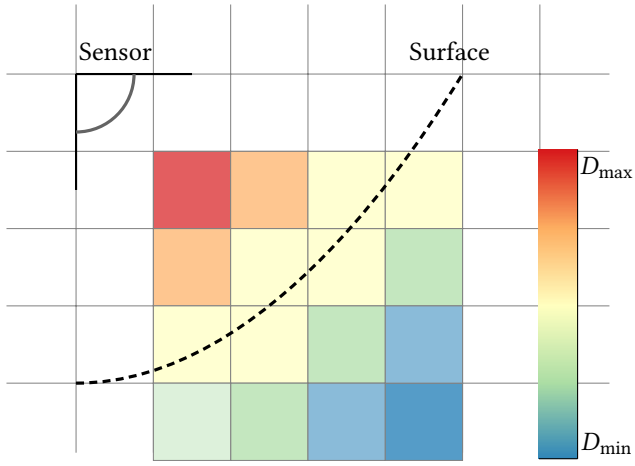


Figure 4.6: Illustration of how the TSDF allocates distance values. Red cells indicate positive values whereas cells on the blue side of the spectrum indicate negative samples.

From the created TSDF, the ESDF if incrementally generated, which is then used to calculate the GVD. The resulting GVD is then further thinned, before the skeleton is constructed by preserving the best nodes, connecting them with edges and fixing any unconnected subgraphs. The best nodes, in this case, are chosen using a k-D tree, as the nodes in each neighbourhood that is maximally distant from obstacles.

The combination of a sparse graph and a heuristic search, as mentioned in section 3.5 and as exemplified in [54], makes for very efficient planning. Experiments show the efficiency of the method, and results show that the use of a sparse A* graph search produces a path 800 times faster than with a standard RRT* algorithm run on the ESDF [54]. Figure 4.7 shows an example of the sparse graph generated via the use of distance fields applied on a dataset from ETH ASL. In fig. 4.8, the path from a set start and goal pose is shown. Results from an additional test case is shown in figs. C.3 to C.4.

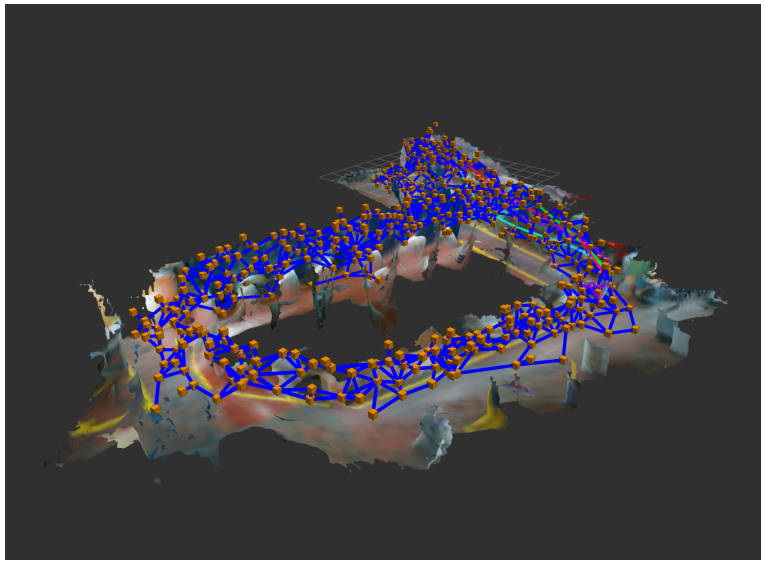


Figure 4.7: Resulting sparse graph generated via the distance field method described in [54], visualised in RViz.

4.4 Summary

In addition to algorithmic efficiency in the planner, the environment needs to be represented efficiently to obtain an online path planning system for three dimensions. This chapter discussed three of the most common and promising environment representations presently used in modern path planning systems. Occupancy grids is probably the most used one, and provides a basic framework for defining free and occupied space together in an intuitive grid model. This representation still suffers from resolution completeness and the intrinsic grid density discussed in section 3.5. The density being the biggest problem if it is to be used as a global 3D map. Octrees are a natural substitute to the regular 3D grid, as they provide a sparser representation with increased resolution in neighbourhoods surrounding obstacles, as exemplified in fig. 4.2. By basing the 3D occupancy grid on octrees, a much more memory efficient representation can be obtained, where random queries can be completed in $O(\log n)$ time. Furthermore, in practice, $\log n$ is bounded by the tree depth, resulting in a constant upper bound [73].

Voronoi diagrams, being the third discussed alternative, poses a promising alternative, basing the environment representation on the medial axis defined by the surrounding obstacles. The sparseness obtainable through the related GVD can result in extremely efficient planning queries. Coupling the sparseness with the maximum clearance property results in the possibility of a very robust and safe path planning system. Practical experiments is needed to verify that

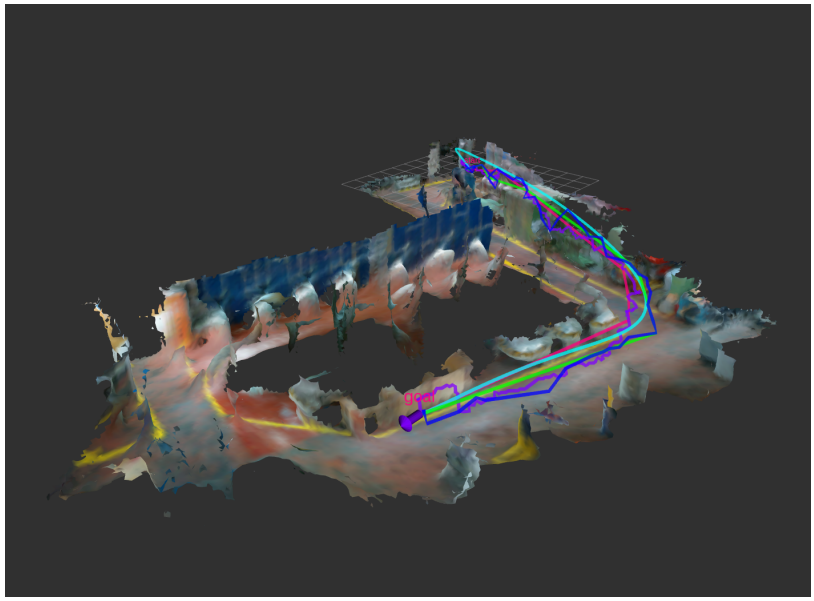


Figure 4.8: Path calculated based on the sparse graph in fig. 4.7. The blue path is the route through the sparse graph, the cyan path is the smoothed variant, whereas the pink path is a simplified version of the blue path, cutting a subset of nodes. The purple path is a pure A* search through the ESDF. Calculated using the open source Voxblox library (github.com/eth-asl/voxblox).

the sparse GVD can be created real-time.

Similarly to the two-dimensional case, sampling-based methods quickly became the industry standard for solving 3D path planning problems. The RRT-family is heavily represented in recent literature; again, largely due to their general algorithmic simplicity and the ease of which they can be extended to 3D space. Another key point, is that the environment representation doesn't affect the sampling-based methods as heavily as they do the combinatorial methods. Partly due to their randomness, however, a method based on the GVD is deemed preferable, given a suitably efficient implementation.

Having now discussed different methods for obtaining a path from $P_{\text{init}}(x, y, z)$ to $P_{\text{goal}}(x, y, z)$, the question still remains of how to autonomously set P_{goal}. The next chapter discusses the more complete autonomous exploration system, including environment exploration as well as safe and energy-conscious path planning.

5 | Path Planning for Dynamic Exploration

Key tasks in many modern robotic applications include autonomous *exploration* and *mapping*. This means that the robot should estimate its pose while simultaneously seeking to exhaustively explore and keep track of the environment. This process is widely known as *Simultaneous Location and Mapping* (SLAM) [7]. Traditionally, this entails manually or remotely steering the robot while building a map. Having this process run autonomously establishes a need for the system to, independently, decide where to go next based on the estimated map and sensory information. This problem of deciding the best exploration goals, and in turn planning paths to reach them, is referred to as *Active SLAM* [74]. There are several ways of deciding these exploration goals, one such being focusing on *loop closures*. Loop closing connects previously explored parts with newly discovered ones and updates the map with the combined information. This procedure reduces the uncertainty of the mapping, as it allows for correcting any drift that has inevitably occurred [7]. Thus, the exploration strategy could be to exhaustively explore the environment while aiming to minimise the uncertainty. The challenge then becomes how to calculate a satisfactory exploration goal.

5.1 Environment Exploration Strategies

Planning in a dynamic exploration setting entails not knowing the whole map. Therefore, the system needs to be able to characterise unknown areas of the map and from that calculate a new exploration goal. Early exploration methods were based on wall-following [75], which continuously follows obstacle edges to complete the map. Difficulties then arise if one were to implement such methods in more open areas. Other methods have since been developed, based on different techniques. Some of these methods include *nearest frontier, cost-utility, behaviour-based* and *hybrid* approaches.

In 1997, Yamauchi presented a straightforward nearest frontier solution [10]. For this method, regions on the boundary between explored and unexplored parts of the map are characterised as *frontiers*. The main idea is then to move the robot to the closest frontier to

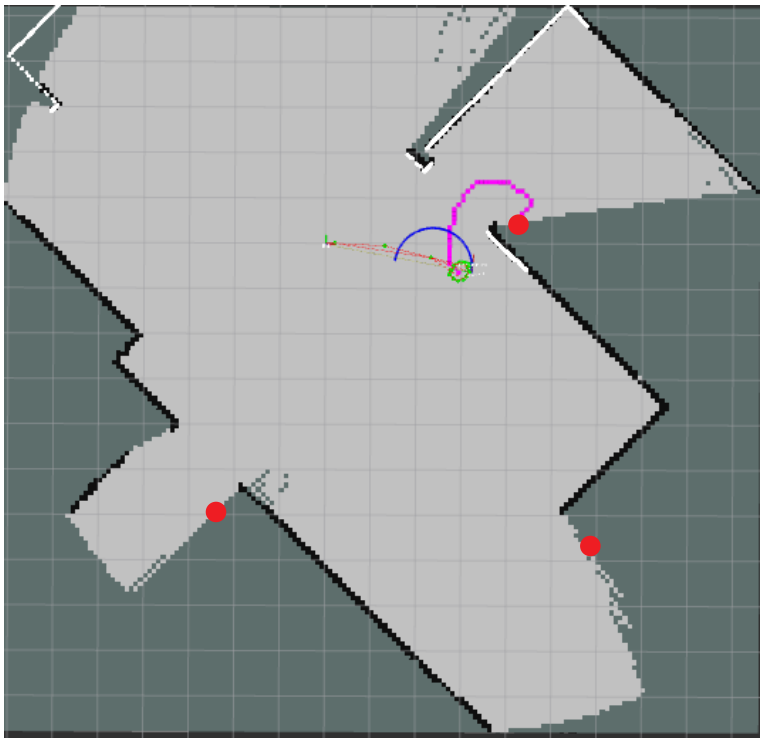


Figure 5.1: RVIZ screenshot showing a 2D SLAM simulation using an occupancy grid representation with frontier examples marked in red and the RRT-calculated path from the current robot position (green) to the closest frontier shown in pink.

extract new information about the world. A simple illustration of the concept is shown in fig. 5.1, where the nearest frontier exploration approach is applied to a 2D SLAM simulation. This method uses a probabilistic occupancy grid representation, where cells are divided into three categories depending on their occupancy probability: Using these boundaries, clusters of

Free	Occupied	Unknown
$P_{\text{occ}} < P_{\text{prior}}$	$P_{\text{occ}} > P_{\text{prior}}$	$P_{\text{occ}} = P_{\text{prior}}$

open cells above a certain size that lies adjacent to unknown cells are labelled as frontiers. The centroid of each frontier region is then calculated. The exploration goal is then set to be the centroid closest to the robot's current position. Multiple other studies have also based their goal choosing method on a nearest frontier strategy, but with different search strategies. Examples include topological vision-based approaches [76] and feature-based methods [77]. The big advantage of this method is its simplicity. As stated in [10], it also allows for exploration in

narrow spaces. Seeing as it always chooses the closest frontier, it has no notion of the *safeness* of a path and can then be more susceptible to environment and vehicle modelling errors.

In an attempt to circumvent this, an exploration method was developed which combines the closest frontier approach [10] and a path cost evaluation based on the *path transform* of [78]. This method, presented in [11], bases the exploration on minimising both the path length and the path risk. This is formulated as

$$\Psi(x) = \min_{x_g \in F} \left\{ \min_{r \in \Xi_x^{x_g}} \left\{ l(r) + \alpha \sum_{x_i \in r} C_{\text{danger}}(x_i) \right\} \right\} , \tag{5.1}$$

where F is the set of all frontier cells, $\Xi_x^{x_g}$ is the set of all paths from x to x_g, $l(r)$ is the length of the path r, $c_{\text{danger}}(x_i)$ is the cost function for the risk of entering cell x_i and $\alpha \geq 0$ is a weighting factor. C_{danger} was originally stated in [78], but this formulation essentially enforces a repulsive force on the robot from the obstacles no matter the distance between them. To bypass this, the cost function was reformulated as

$$C_{\text{danger}} = \begin{cases} \infty, & \text{if } d < d_{\min} \\ (d_{\text{opt}} - d)^2, & \text{else} \end{cases} , \tag{5.2}$$

where $d_{\min}$ represents the closest allowable distance between the robot and obstacles and d_{opt} is an estimated optimal, or encouraged, clearing [11]. The main advantages of this approach are that the path safety is taken into account and that it finds the most appropriate path without the risk of running into local minima. In practical implementations, this has shown good results. This method can be considered a version of a cost-utility method, as it combines the frontier search with cell and path costs. Another cost-utility based way of choosing the next waypoint would be to estimate the utility, or information gain, of a waypoint candidate.

Information gain, in this sense, represents the amount of information about the environment gained by reaching a certain point in space. The expected information gain can be defined in terms of *entropy* [79]. The entropy can be approximated as the joint entropy of the path $\mathbf{r}(\varpi) = \mathbf{x}_k = x_{1:k}$ and the map $\mathcal{W}_m$, given a series of control inputs $\mathbf{U}_k = u_{0:k-1} \in \mathcal{U}$ and a set of observations $\mathcal{Z}_k = z_{1:k}$. The total entropy then becomes [80]

$$H(\mathbf{x}_k, \mathcal{W}_m \mid \mathbf{U}_k, \mathcal{Z}_k) \approx H(\mathbf{x}_k \mid \mathbf{U}_k, \mathcal{Z}_k) + H(\mathcal{W}_m \mid \mathbf{U}_k, \mathcal{Z}_k) , \tag{5.3}$$

where the individual entropies are defined as in [80]

$$H(\mathbf{x}_k \mid \mathbf{U}_k, \mathcal{Z}_k) \approx \frac{1}{k} \sum_{i=1}^{k} \ln\left(2\pi \exp\left[\frac{n'}{2}\right]\right) |\Sigma_{ii}| \tag{5.4}$$

$$H(\mathcal{W}_m \mid \mathbf{U}_k, \mathcal{Z}_k) = -w^2 \sum_{c \in \mathcal{W}_m} \{p(c) \ln p(c) + [1 - p(c)] \ln[1 - p(c)]\} \ , \tag{5.5}$$

where n is the size of the state vector and Σ represents the covariance matrix. This formulation is based on pose-SLAM, i.e. $\mathcal{W}_\mathrm{m}$ is represented as a pose graph. To find the input, or action, that maximises the expected information gain one can instead find the input that minimises the joint posterior entropy $H(\mathbf{x}', \mathcal{W}_\mathrm{m} \mid \mathbf{U}_k + \mathbf{U}', \mathcal{Z}_k + \mathcal{Z}')$ [80]:

$$\mathbf{U}'* = \operatorname{argmin} \{H(\mathbf{x}', \mathcal{W}_\mathrm{m} \mid \mathbf{U}_k + \mathbf{U}', \mathcal{Z}_k + \mathcal{Z}')\} \ . \tag{5.6}$$

Applicable actions $\mathbf{U}$ include exploration to specific waypoints, but also include actions where the robot tries to perform loop closure. The advantage of such an exploration method is that the resulting algorithm gives good results regarding loop closure and frontier exploration, since actions can be chosen to either reduce pose uncertainties by exploring the environment or to reduce path uncertainties by looping back to known locations. This expected information gain can further be combined with an evaluation of the path length to then be able to weigh the information gain against travel cost.

Related to this, focusing on minimising landmark uncertainty, is the method presented in [81]. This method aims to plan towards minimising the uncertainty of landmarks and robot location in the map by weighting the two uncertainties. I.e., if the robot location has a high uncertainty, the next waypoint should be chosen based on nearby landmarks with low uncertainty, and vice versa. Both of these methods, however, build upon a grid-based environment representation by design. If generating the global graph based on distance fields, the ESDF representation could instead be used. Nevertheless, studies have explored the use of topological representations for exploration.

Topological exploration techniques are favourable for larger environments, as these approaches have a better scalability. Different approaches exist also here. A data structure put forth to facilitate visibility-based robotic exploration, was the *Gap Navigation Tree* (GNT) [82]. This structure aims to track discontinuities in the depth information available to the robot at its current position. Thus, the GNT essentially tracks gaps in the environment. These gaps are classified as *primitive* if the gap is discovered and later disappears, or *non-primitive* if it is yet to be explored. Simple exploration strategies can then be implemented to follow non-primitive gaps until none remains. This method, while promising, has not shown reliable results in

practice [83]. This is somewhat due to the dependency of the gap detector which, as was shown in [83], can lead to a failure to correctly detect gaps. In addition, this method was mainly developed for planar exploration, meaning that heavy modifications would have to be made to expand it to three-dimensional exploration.

Recently, another topological method, combining Graph-SLAM [84] with contour segmentation and an online depth-first search, was introduced [85]. While this method is online and shows promising results in simulations, it is also restricted to two dimensions. The majority of exploration strategies in modern literature have, similarly to path planning techniques, been focused on two-dimensional problems. Only very recently have research targeted the extension of frontier exploration to three dimensions. These approaches convert 3D point cloud data to an octree, from which frontiers are extracted [86], [87]. While being based on octrees, this shows the promise of real-time frontier extraction in 3D exploration.

After a new waypoint has been set, for example using the nearest frontier approach, there is the possibility of the robot having sufficiently explored the region associated with the chosen frontier. To account for this, the method of *repetitive re-checking* has been proposed [88]. If the chosen frontier at any point during execution ceases to be a frontier cell, it is dropped and a new waypoint is chosen. Results show that this addition does not increase the computational burden given the constant-time look-up to determine if it is still a valid frontier. This simple addition can help decrease the over-all path length and exploration duration by decreasing the time spent in already mapped terrain. Another problem that can occur, if the explorations goals aren't chosen optimally, is that semi-closed parts of the map can be visited multiple times. In 2D, the analogy to this is that a room the robot explores is left before being fully explored, due to a goal being set in the adjacent room. The suggested solution is then to segment the environment based on the Voronoi diagram. Frontier cells can then be associated with the nearest Voronoi region. Frontier cells lying in another Voronoi region is then only chosen if the set of frontiers in the current Voronoi region is empty. Thus, the robot will not leave its current enclosed region until it is fully explores. In the case of exploring sparse underwater environments, this is unlikely to be a problem. If the robot is to be utilised in underwater caverns or similar, this might need to be taken into consideration. Nevertheless, this extension is very simple and can decrease the overall path length, and since it is based on Voronoi diagrams, should be relatively easy to extend to 3D.

All the aforementioned approaches have done their experiments above water, where sensor information is much more reliable. By virtue of noisy acoustic sensors, environmental disturbances and a resulting high localisation uncertainty and low map quality, underwater exploration and mapping have been shown to be much more challenging than its above-ground counterpart. Methods have been developed, however, that have shown relatively good results

for underwater mapping and exploration using sonar and cameras with exploration based on view planning and frontier extraction [89]. This work was carried out in a two-dimensional slice of 3D space, so the challenge of developing a complete exploration strategy for exploration and mapping in three-dimensional submarine space still remains.

5.2 Secure Path Planning

Another core concept in autonomous exploration and path planning that needs to be discussed, is the safety of the calculated path. It is crucial for autonomous systems to be able to guarantee the safeness of a calculated path, both with respect to the vehicles own safety and its surroundings. This is especially true for submarine vehicles, as repairs quickly become cumbersome and expensive. The concept of safe planning was briefly mentioned in section 4.3, regarding the maximum clearance property of Voronoi diagrams and in section 5.1 with the inclusion of path risk in the exploration strategy, known as the *exploration transform* [11]. As discussed in section 3.1, sampling-based methods, such as the RRT-family, bases their exploration into C on a local planner. This ensures that the path is always contained in C_{free}. The strategy of basing the path planning on a Voronoi diagram gives a guarantee on the safety of the calculated path given the detected obstacles as generator points. The underlying assumptions here is that all obstacles are detected and that their position is certain, to some degree. This is often not the case for submarine robots, as the perception systems often have problems with uncertainties or noise as a result of difficulties imposed by the challenging underwater environment [9].

A Voronoi-based planner is, in essence, a global planning system. While it ensures safety from static obstacles by producing maximum clearance paths, it would be very computationally inefficient to dynamically recalculate the whole diagram if a dynamic obstacle enters the environment. Global planning systems therefore often need a local planner to take care of unforeseen events or to do local path optimisations. This local planner could be e.g. based on local Voronoi diagrams [63] or use potential fields [34], [35]. Using a local planner can help counteract uncertainties in the perception. For example, in the case an obstacle is detected at a later stage than preferred, the local planner would be able to calculate a new collision-free path faster than the global planner. Another way of handling uncertainties is to model them into the planning system. Another way of handling uncertainties, however, is to model them into the planning system.

A strategy to accomplish this, put forth in [90], introduces the concept of *towers of uncertainties*. If all nodes $n \in C$ is to be modelled with a corresponding uncertainty, the dimension of C would increase by three. Uncertainties towers were introduced to reduce the dimensionality of the resulting *uncertain C-space*, $\tilde{C}$. Each tower has a dimension of one and a position, with

every *level* of the tower containing a node with its associated uncertainties (L, l, ϕ, Θ). A modified version of A* is used to search through $\tilde{C}$, where each node is defined as $(x, y, \theta, L, l, \phi, \Theta)$. Experimental results show that this way of handling uncertainties leads to the robot choosing safer, but longer paths if its initial uncertainties in pose are large, whereas a more risky path is chosen if the uncertainties in the goal pose are large. This was implemented and tested in two-dimensional environments, but should be possible to extend to 3D. It could, however, be of more interest to base the planning more directly on the uncertainties of the states in the SLAM system.

5.2.1 Path Smoothing

Going beyond uncertainties in the map, the safety and flyability of the path are still of concern. Using a graph representation, especially, the raw path obtained from the graph search is comprised of a set of waypoints with straight lines connecting them. The *continuity* is one of the fundamental definitions of a path. In chapter 2, the notion of continuity was implicitly stated in the path definition (eq. (2.8)). The path calculated directly from the graph representation of C will contain discontinuities and curvatures likely to not coincide with the given vehicle dynamics. To overcome this, different *path smoothing* techniques have been developed. One of the first methods to handle this was the Dubins path [91] which finds the shortest path to the goal by representing the path as a set of straight lines and circular arcs, given a maximum curvature constraint. This method helps to avoid overshoot when switching waypoints, but introduces two discontinuities at the start and end of the circular arc. It can be shown that the resulting Dubins path satisfies C^1 continuity [1], i.e. once continuously differentiable, but fails to obtain C^2 continuity. The curvature of a 2D path $r(\varpi)$ can be calculated as

$$\kappa(\varpi) = \frac{\dot{x}(\varpi)\ddot{y}(\varpi) - \dot{y}(\varpi)\ddot{x}(\varpi)}{\left(\sqrt{\dot{x}(\varpi)^2 + \dot{y}(\varpi)^2}\right)^3} \ . \tag{5.7}$$

From eq. (5.7), it can be seen that the path is required to be twice continuously differentiable. The control system responsible for controlling the attitude of the robot is then subject to a discontinuous reference signal, which can result in unwanted behaviour (e.g. due to coupled dynamics in underactuated vehicles) and excessive strain on the actuators. From a control system point of view, C^2 continuity is, therefore, favoured [17], [26].

Attempts have been made to combine Dubins paths with clothoids, which have linearly changing curvatures. One big disadvantage of using clothoids, however, is that they have no analytic solution [26]. This leads to longer computation times, which in turn could make them

[1] See appendix A.1 for a short definition of different levels of smoothness.

very impractical for online 3D planning purposes. In addition, this property can prove clothoids difficult to use in the presence of obstacles and can reduce the algorithmic completeness of the planning system [92]. A more promising interpolation method can be obtained by using splines. Splines facilitate combining path segments of lower polynomial degree so that the combined path *appears* smooth [93]. For the three-dimensional case, this requires the path to be C^3 continuous. This can be obtained by constructing a *shape-preserving*[2] C^1-G^2 cubic spline and adding a correction term, which can be of variable degree [94].

While spline methods can give good results, yet another less computationally heavy method exists. Fermat's spiral, introduced in 1636 by Pierre de Fermat as a variant of the Archimedean spiral, is another good candidate for solving the path smoothing problem. Fermat's spiral is defined as

$$\underbrace{r = a\sqrt{\theta}}_{\text{polar}} \quad \underset{\mathbf{T}(x,y)}{\overset{\mathbf{T}(r,\theta)}{\rightleftharpoons}} \quad \underbrace{\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} x_0 + au\cos(\rho u^2 + \chi_0) \\ y_0 + au\sin(\rho u^2 + \chi_0) \end{bmatrix}}_{\text{cartesian}}, \tag{5.8}$$

where $u = \sqrt{\theta}, u \in [0, \theta_{\max}]$[3], a is a parameter that defines the spiral turning (scaling factor in Cartesian), $\rho = \{\pm 1\}$ is the turning direction and χ_0 is the initial tangent angle [36]. The transformation $\mathbf{T}(\cdot)$ here indicate a transformation between polar and Cartesian coordinates which allows different initial positions $(x_0, y_0)^\top$ and initial tangent angles χ_0 [40]. This alternative parametrisation — instead of the direct Polar-to-Cartesian transform — guarantees the wanted C^2 continuity. The tangential angle can be written as

$$\chi(\theta) = \operatorname{atan2}\left(\sin\theta + 2\theta\cos\theta, \cos\theta - 2\theta\sin\theta\right) , \tag{5.9}$$

but suffers from discontinuities at $\theta = \frac{\pi}{2}$. Rewriting eq. (5.9), an equation continuous on $\theta \in [0, \pi]$ can be found [40]:

$$\chi(\theta) = \theta + \arctan(2\theta) , \tag{5.10}$$

the mirrored curve can then be used to reach 2π coverage. The curvature of eq. (5.8) is given as

$$\kappa(\theta) = \frac{\|\dot{r} \times \ddot{r}\|}{\|\dot{r}\|^3} = \frac{2\sqrt{\theta}(4\theta^2 + 3)}{a(4\theta^2 + 1)^{\frac{3}{2}}} . \tag{5.11}$$

[2] Shape-preserving here refers to a path that satisfies the collinearity, convexity, coplanarity and torsion criteria defined in appendix A.2.

[3] $\frac{d\kappa}{d\theta}(\theta) = 0 \;\Rightarrow\; \theta_{\max} = \sqrt{\frac{\sqrt{7}}{2} - \frac{5}{4}}$

From eq. (5.11) it is clear that the initial curvature $\kappa(0)$ is zero. This makes for a smooth transition between a straight line segment and the spiral path. The whole arc segment consists of an initial spiral curve segment and its mirrored version, connecting back into the straight line segment. A benefit of its simple parametric definition is that the Fermat's spiral is computationally efficient. Compared to splines, Fermat's spiral interpolation is more robust against disturbances [26]. The biggest drawback of the spiral parametrisation comes to light when calculating the path length. For a Fermat spiral, calculating the arc length results in evaluating a hypergeometric Gaussian function. This function has no analytic solution, but is guaranteed to converge. It is also worth noting that a spiral-interpolated path will always be longer than the corresponding Dubins path. An example of how the piece-wise linear path can be interpolated using Fermat spirals is illustrated in fig. 5.2.

When comparing path interpolation techniques in environments with obstacles, it is useful to compare the path *allowance*. The allowance, $\mathfrak{a}$, indicates how much a smoothed segment differs from the connected straight line path [26]. In other words, the magnitude of the cross-track error between the smoothed path and the straight lines. If the smoothed path deviates too much from the calculated path, the robot might run the risk of hitting obstacles. A small allowance is therefore ideal. The allowance of Fermat's spirals is in general smaller than for clothoids, but larger than that of Dubins paths, hence

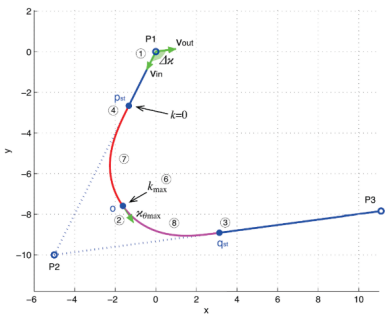


Figure 5.2: An illustration of how the Fermat spiral interpolation can be executed [36]. $\kappa_{\max}$ indicate the point of maximum curvature, also being the point where the initial and mirrored curves meet.

$$\mathfrak{a}_{\text{Dubins}} \leq \mathfrak{a}_{\text{Fermat}} < \mathfrak{a}_{\text{Clothoid}} \ .$$

Discarding clothoids based on their computational inefficiency, this leaves Fermat's spirals as possibly the most promising interpolation technique compared to spline-based methods or Dubins path methods. This is further backed by the fact that spline-based methods, by design, give paths that rarely are completely straight. A continuously curving path means that at least one robot actuator will be active at any given time, which is not ideal. Fermat's spirals, however, have not yet been generalised to three dimensions. It would, therefore, be of interest to extend this definition to 3D, while preserving the continuity benefits of the 2D parametrisation, and test it in a three-dimensional planning system. There is also the possibility of combining Fermat's spirals with Dubins paths to overcome the discontinuities between

the straight line segment and the circular arc [95]. This could further decrease the allowance obtained when using a spiral arc by introducing the circular arc. Three-dimensional Dubins paths have already been developed and tested [63], [96], thus the challenge would just be to connect the spiral segment to this 3D arc.

5.3 Energy-sensible Planning

As discussed in the previous section, a safe path is paramount for autonomous operations. A growing number of operations are intended to last a prolonged amount of time, with AUV operations exceeding 500km travel distances [97]. When mission duration increases, planning systems that take energy consumption into mind becomes more important. The most intuitive way of decreasing the energy spent on traversing a path is naturally to find a shorter path. This is not always the case, as one of the main processes that influence a submarine robot's motion, is the ocean currents. These currents can impede the robot's motion, having it consume more energy to fight against it or even have it alter course. For mobile underwater robots, it is, therefore, necessary to be able to counteract disturbing currents and exploit beneficial currents. This methodology has been explored in previous work, often based on optimisation methods such as evolutionary or *Ant Colony Optimization* techniques [98], [99], or with a probabilistic basis [100]. Optimisation-based methods tend to give good results and are able to utilise any available positive contribution from the currents themselves. If based on global current information, however, the problem of their high computational costs quickly becomes relevant. Probabilistic methods, e.g. based on an RRT variant, while solving the problem, are prone to give results not fully utilising the currents [100].

A proposed method to solve the current problem is to instead base it on a locally optimal heuristic grid search using D*-Lite. A local three-dimensional grid can then be projected around a discretised model of the robot. Cells in the grid then contain information about local currents which is used when calculating the heuristic. The initiallly proposed heuristic can then be stated combining the distance to a target node and the local current velocities

$$w = w_{\mathrm{c}} + w_{\mathrm{d}} \ , \tag{5.12}$$

where w_{c} is calculated based on the measured current velocities and direction in the cell and w_d is calculated based on the distance to the target node. In addition to this, the travel time could be considered. This would yield a somewhat naive local planner that, while not taking direct energy consumption into mind, does so implicitly by finding the path locally optimising current effects on the robot by weighting both the current effects and the path distance. Weighting

the search based on the direct energy cost is possible [101], [102], but requires more in-depth information about the actuator systems and more complex calculations.

There are two problems with this method, however. Namely that the grid resolution and the robot discretisation will affect the optimality of the solution. It could be interesting to instead base this on an RRT variant, such as in [101], and base the tree expansion on the vehicle dynamics together with the current velocities. This proposed method only evaluates local currents, as more global currents are assumed changing, likely resulting in replanning if the global state was considered. As a side note, since only local currents are considered, more complex optimisation methods could be promising, as the increased computational complexity will matter less in the smaller state space.

5.4 Summary

In this chapter, the importance of exploration strategy has been introduced, and different techniques for selecting them has been discussed. The ease of which already developed methods can be applied is somewhat dependent on the way the map is represented. In general, however, the frontier-based methods seems to give a very good balance between simplicity and path quality. Experiments show that the nearest frontier strategy combined with the aforementioned repetitive rechecking and map segmentation result in much shorter travel distances than simple information gain-based approaches [88]. While being a good starting point for implementation and testing, it would be of interest to have the goal placement integrate more with the SLAM system. This can result in better map quality, but will likely result in longer paths, especially when trying to obtain loop closures. As exemplified in [80], however, the final map quality can be significantly better when applying the entropy-approach over the frontier exploration. This is evident in fig. 5.4, where significant localisation errors are evident. While it comes to evaluating path risk, methods like the exploration transform [11] give promising results. If basing the planning system on GVDs, however, this is likely not needed, as the risk of the path is already minimised with respect to obstacle distance. In situations where the robot is required to enter very narrow spaces, a different strategy might be necessary, since the GVD planner might fail to find a feasible path through is. A possible workaround could be to have a secondary planner, e.g.

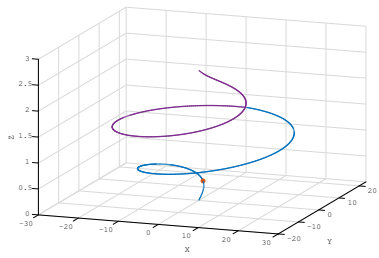


Figure 5.3: Illustration of the simple z-modification to Fermat's spiral that conserve. The blue line is the Fermat spiral and the purple is its mirror. Red indicates the first point of maximum curvature.

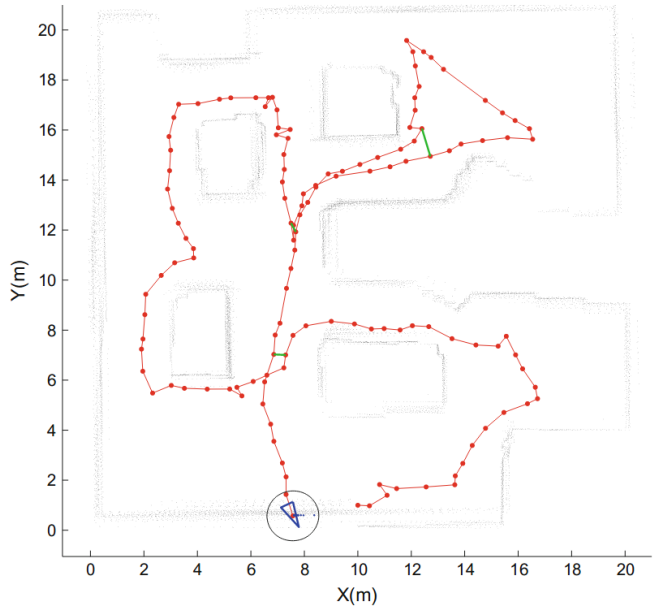


Figure 5.4: Example map from running frontier-based SLAM exploration (image credit: [80]).

based on an RRT-variant, which is deployed in cases where the GVD planner is unable to calculate the complete path.

One aspect of the path that needs to be discussed no matter the applied planning system, is its smoothness. Different tried and tested methods have been discussed. The importance of the degree of continuity of the generated path as well as the computational time has been the main factors in evaluating the different techniques. Based on literature, the Fermat's spiral is a promising interpolation method. Due to this, experiments in generalising the Fermat spiral to three dimensions while preserves the properties of the 2D parametrisation will likely be the aim of future work. A possible extension would be to augment the parametrisation in eq. (5.8) with the z-coordinate simply as $z = z_0 + n\varepsilon u$ (similar to what was done in [103]), where n is the number of turns and ε is the number of units along z-axis per turn. By choosing these parameters based on the vehicle's controllability and the path curvature, this could result in a satisfyingly smooth path. A simple example of this extension is shown in fig. 5.3. Another alternative z-parametrisation can be found in [96], based on their *Dubins plane* model with helix paths.

The problem of energy-sensible planning has been briefly discussed. Literature shows different simulated methods that include currents in the planned path. Most methods are based on numerical optimization, where all currents in the global map are considered. In order to

account for, or take advantage of, drift induced by ocean currents, a simple local planner based on a dynamic heuristic search algorithm was instead suggested. This local planner will operate on a local 3D grid in which cells are directionally weighted based on the given current angle and magnitude and the distance from it to the target. The main motivation behind this is that the current is assumed fluctuating, thus the state of the ocean currents might have changed at a certain point in time, resulting in the need for replanning. However, further simulations and experiments are needed to fully test this hypothesis.

6 | Conclusion

The goal of this work was to, through a survey of approaches in modern literature, propose a suitable planning system that takes robot safety as the number one priority, while also facilitating energy considerations and autonomous exploration for the mapping of unknown environments.

Different planning methods in both two and three dimensions have been presented and discussed with respect to their applicability in an underwater robotic exploration system. The simplicity of sampling-based methods have been weighed up against the completeness and inherent path safety obtained from combinatorial methods. While definitely having their use-cases, sampling-based methods has been deemed inadequate regarding the resulting path safety when considering the intended application. It must be noted, however, that the higher risk paths obtainable using sampling-based methods could be used as a secondary planner in conjunction with a GVD-based global planner; to be deployed in situations where the more risk-aware planner fails to find a guaranteed safe path.

The suggested path planning method is based on a global, maximum obstacle clearance graph planner together with an optimal heuristic search algorithm and a local energy-considerate planner. The exploration strategy is based on frontiers to visit previously unexplored sections of the map. It is proposed to include pose uncertainties and include this in the goal calculations to increase map quality and aid SLAM loop closures.

The main challenges for implementing the global path planning system comes when requiring online generation of the 3D Voronoi diagram. Building the diagram base on distance fields [54] seems like a promising way to go. However, the fact that the the proposed implementation in [54] generates the ESDF live, but prunes the graph offline, indicate that the distance field-method for generating GVDs still require some refining. An alternative could be to instead incrementally build a 3D occupancy grid based on octrees [73], then estimate clusters of the incoming point cloud data from the SLAM system using an algorithm such as DBSCAN. These clusters can then be encapsulated in a minimum bounding polygon, using a convex hull algorithm [69], from which the defining points can be used as generator points for the Voronoi diagram. The resulting diagram can then be efficiently pruned by removing the nodes and connected edges that lie inside the bounding polygons. If the DBSCAN algorithm can be implemented to incrementally generate clusters, the total planning system should be

computationally satisfactory.

Simulations performed in this work and its cited sources show the potential of a global Voronoi-based planner in underwater environments. Going forward, complete simulations should be done to more closely evaluate the global and local planning system. There are, however, some important notes to take into considerations when simulating systems that are to be implemented on real robotic systems. Simulators, of course, allows for testing without risking equipment, but does so while doing certain assumptions. In simulations we can work with idealised sensors or simplified dynamics [104]. Some of the sources in this paper have based their proposed methods solely on simulated results where specific assumptions have been made regarding obstacles. One such example is the results in [63], where all obstacles was assumed known and static, with the exception of the single moving obstacle. For the intended application presented in this paper, these assumptions will not hold. This, together with the fact that the results from the perception system cannot be assumed perfect, leads to the need of extensive practical tests in closed environments to fully evaluate the proposed methods.

6.1 Future Work

Depending on the results of the practical evaluation of the planning system, the additional future work include:

- Online implementation of the proposed planning system on an actual ROV/AUV and test it on an underwater dataset as well as practical tests in ocean-like environments. A version of the DBSCAN algorithm will have to be implemented to allow for incremental clustering based on incoming SLAM-data.
- Generalise the Fermat's spiral parametrisation to three dimensions and/or combine them with 3D Dubins path for C^2 continuous path interpolation, potentially comparing them and evaluating advantages and disadvantages of the respective methods.
- Incorporation of the complete vehicle dynamics in the planning system to allow for safer planning.
- Development of a more robust exploration strategy for 3D underwater exploration. Extend the basic frontier cell formulation to include information from the SLAM system.
- Since little research has been don comparing different 3D path planning methods, it would be of interest to perform benchmarking comparisons of different planning methods for underwater autonomous robots. This could further help give a better framework for determining the most appropriate methods for underwater 3D path planning.

- Research the use of, and implement, machine learning algorithms to estimate obstacle regions in unknown territories of the map to better (and safer) set new exploration goals.

Bibliography

[1] J. A. Dowdeswell, J. Evans, R. Mugford, G. Griffiths, S. Mcphail, N. Millard, P. Stevenson, M. A. Brandon, C. Banks, K. J. Heywood, M. R. Price, P. A. Dodd, A. Jenkins, K. W. Nicholls, D. Hayes, E. P. Abrahamsen, P. Tyler, B. Bett, D. Jones, P. Wadhams, J. P. Wilkinson, K. Stansfield, and S. Ackley, "Instruments and Methods Autonomous underwater vehicles (AUVs) and investigations of the ice-ocean interface in Antarctic and Arctic waters," Tech. Rep. [Online]. Available: www.bodc.ac.uk/.

[2] G. Marani, S. K. Choi, and J. Yuh, "Underwater autonomous manipulation for intervention missions AUVs," *Ocean Engineering*, vol. 36, no. 1, pp. 15–23, 2009.

[3] R. B. Wynn, V. A. Huvenne, T. P. Le Bas, B. J. Murton, D. P. Connelly, B. J. Bett, H. A. Ruhl, K. J. Morris, J. Peakall, D. R. Parsons, E. J. Sumner, S. E. Darby, R. M. Dorrell, and J. E. Hunt, "Autonomous Underwater Vehicles (AUVs): Their past, present and future contributions to the advancement of marine geoscience," *Marine Geology*, vol. 352, pp. 451–468, 2014. [Online]. Available: http://dx.doi.org/10.1016/j.margeo.2014.03.012.

[4] A. Lavin, "Optimized Mission Planning for Planetary Exploration Rovers," *CoRR*, vol. abs/1511.0, 2015. [Online]. Available: http://arxiv.org/abs/1511.00195.

[5] P. Tompkins, "Mission-directed path planning for planetary rover exploration," PhD thesis, The Robotics Institute, Carnegie Mellon University, 2005, pp. 1–192. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.69.9399&rep=rep1&type=pdf.

[6] M. Faria, I. Maza, and A. Viguria, "Applying Frontier Cells Based Exploration and Lazy Theta* Path Planning over Single Grid-Based World Representation for Autonomous Inspection of Large 3D Structures with an UAS," *Journal of Intelligent and Robotic Systems: Theory and Applications*, pp. 1–21, 2018.

[7] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age," *IEEE Trans. Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016. [Online]. Available: http://arxiv.org/abs/1606.05830%0Ahttp://dx.doi.org/10.1109/TRO.2016.2624754.

[8] L.-Y. Weng, M. Li, Z. Gong, and S. Ma, "Underwater object detection and localization based on multi-beam sonar image processing," *2012 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, no. December, pp. 514–519, 2012. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=6491018.

[9] M. Leonardi, A. Stahl, M. Gazzea, M. Ludvigsen, I. Rist-Christensen, and S. M. Nornes, "Vision based obstacle avoidance and motion tracking for autonomous behaviors in underwater vehicles," *OCEANS 2017 - Aberdeen*, vol. 2017-Octob, no. June, pp. 1–10, 2017.

[10] B. Yamauchi, "A frontier-based exploration for autonomous exploration," *IEEE International Symposium on Computational Intelligence in Robotics and Automation, Monterey, CA*, pp. 146–151, 1997.

[11] S. Wirth and J. Pellenz, "Exploration transform: A stable exploring algorithm for robots in rescue environments," *SSRR2007 - IEEE International Workshop on Safety, Security and Rescue Robotics Proceedings*, 2007.

[12] J. T. Schwartz and M. Sharir, "On the "piano movers" problem. II. General techniques for computing topological properties of real algebraic manifolds," *Advances in Applied Mathematics*, vol. 4, no. 3, pp. 298–351, 1983.

[13] S. M. LaValle, "Planning algorithms," *Planning Algorithms*, vol. 9780521862, pp. 1–826, 2006.

[14] *Norwegian Register for Scientific Journals, Series and Publishers*, 2018. [Online]. Available: `https://dbh.nsd.uib.no/publiseringskanaler/AlltidFerskListe`.

[15] J.-c. Latombe, *Robot motion plannig*, 9. Springer Science + Business Media, LLC, 1991, vol. 53, pp. 58–105.

[16] S. M. Lavalle, "Motion planning: Part I: The essentials," *IEEE Robotics and Automation Magazine*, vol. 18, no. 1, pp. 79–89, 2011.

[17] A. Tsourdos, B. White, and M. Shanmugavel, *Cooperative Path Planning of Unmanned Aerial Vehicles Cooperative Path Planning of Unmanned Aerial Vehicles Aerospace Series List Design and Analysis of Composite Structures: With Applications to Aerospace Structures*. 2011.

[18] H. K. Khalil, *Nonlinear Systems*, 3rd. Prentice Hall, 2002.

[19] O. Egeland and T. Gravdahl, *Modeling and Simulation for Automatic Control*. 2002, vol. 53, pp. 1689–1699.

[20] T. I. Fossen, *Handbook of Marine Craft Hydrodynamics and Motion Control*. 2011, pp. 15–41.

[21] F. A. Leve, B. J. Hamilton, and M. A. Peck, *Spacecraft momentum control systems*. 2015, pp. 1–247.

[22] T. W. Gamelin and R. E. Greene, *Introduction to Topology*, 2nd. Mineola, N.Y.: Drover Publications, 1999, p. 234.

[23] Shonk, "Special Groups and Projective Planes," Tech. Rep. [Online]. Available: `http://www.sellingwaves.com/projplane.pdf`.

[24] M. do Carmo, M. Ritoré, and A. Ros, *Compact minimal hypersurfaces with index one in the real projective space*. Springer, 2012, pp. 407–414.

[25] ProofWiki.com, *Projective Space*, 2017. [Online]. Available: `https://proofwiki.org/wiki/Definition:Projective_Space`.

[26] L. Anastasios and T. I. Fossen, *Introduction to Path Planning, Properties of Curves, Dubins Paths and Clothoids - TTK8190 Lecture Notes*, 2016.

[27] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979. [Online]. Available: `http://portal.acm.org/citation.cfm?doid=359156.359164`.

[28] T. Lozano-Perez, “Spatial Planning: A Configuration Space Approach,” *Computers, IEEE Transactions on*, vol. c, no. 2, pp. 108–120, 1983. [Online]. Available: http://lis.csail.mit.edu/pubs/tlp/spatial-planning.pdf.

[29] B. Chazelle, *Approximation and Decomposition of Shapes*, 1987. [Online]. Available: https://www.cs.princeton.edu/~chazelle/pubs/ApproxDecompShapes.pdf.

[30] J. H. Reif, “Complexity of the mover’s problem and generalizations,” *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pp. 421–427, 1979. [Online]. Available: http://ieeexplore.ieee.org/document/4568037/.

[31] J. E. Hopcroft, J. T. Schwartz, and M. Sharir, “On the Complexity of Motion Planning for Multiple Independent Objects; PSPACE-Hardness of the "Warehouseman’s Problem",” Tech. Rep. [Online]. Available: http://journals.sagepub.com/doi/pdf/10.1177/027836498400300405.

[32] J. F. Canny, *The Complexity of Robot Motion Planning*. 1988, vol. Doctoral D, p. 195. [Online]. Available: https://ia801604.us.archive.org/21/items/TheComplexityOfRobotMotionPlanning/The%20Complexity%20of%20Robot%20Motion%20Planning.pdf%20http://books.google.com/books?hl=en&lr=&id=_VRM_sczrKgC&pgis=1.

[33] O. Khatib, “Real-Time Obstacle Avoidance for Manipulators and Mobile Robots,” *The International Journal of Robotics Research*, vol. 5, no. 1, 1986.

[34] S. Campbell, W. Naeem, and G. W. Irwin, “A review on improving the autonomy of unmanned surface vehicles through intelligent collision avoidance manoeuvres,” *Annual Reviews in Control*, vol. 36, no. 2, pp. 267–283, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.arcontrol.2012.09.008.

[35] M. Imran and F. Kunwar, “A hybrid path planning technique developed by integrating global and local path planner,” *2016 International Conference on Intelligent Systems Engineering, ICISE 2016*, pp. 118–122, 2016.

[36] M. Candeloro, A. M. Lekkas, A. J. Sørensen, and T. I. Fossen, *Continuous curvature path planning using voronoi diagrams and Fermat’s spirals*, PART 1. IFAC, 2013, vol. 9, pp. 132–137. [Online]. Available: http://dx.doi.org/10.3182/20130918-4-JP-3022.00064.

[37] I. Gowda, D. Kirkpatrick, D. Lee, and A. Naamad, “Dynamic Voronoi diagrams,” *IEEE Transactions on Information Theory*, vol. 29, no. 5, pp. 724–731, Sep. 1983. [Online]. Available: http://ieeexplore.ieee.org/document/1056738/.

[38] J.-d. Boissonnat and M. Teillaud, *Mathematics and Visualization Series Editors*, G. Farin, H.-C. Hege, D. Hoffman, C. R. Johnson, K. Polthier, and M. Rumpf, Eds. Springer, 2006, p. 82. [Online]. Available: https://link.springer.com/content/pdf/10.1007%2F978-3-540-33259-6.pdf.

[39] S. Fortune, “A Sweepline Algorithm for Voronoi Diagrams,” Tech. Rep., 1987, pp. 153–174. [Online]. Available: http://www.wias-berlin.de/people/si/course/files/Fortune87-SweepLine-Voronoi.pdf.

[40] A. M. Lekkas, A. R. Dahl, M. Breivik, and T. I. Fossen, "Continuous-Curvature Path Generation Using Fermat's Spiral," vol. 34, no. 4, pp. 183–198, 2013. [Online]. Available: http://www.mic-journal.no/PDF/2013/MIC-2013-4-3.pdf%20http://www.mic-journal.no/ABS/MIC-2013-4-3.asp.

[41] L. E. Kavraki, "RANDOM NETWORKS IN CONFIGURATION SPACE FOR FAST PATH PLANNING," Tech. Rep., 1994. [Online]. Available: http://i.stanford.edu/pub/cstr/reports/cs/tr/95/1535/CS-TR-95-1535.pdf.

[42] L. E. Kavraki, P. Švestka, J. C. Latombe, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[43] A.-a. Agha-mohammadi, S. Chakravorty, and N. M. Amato, "Sampling-based nonholonomic motion planning in belief space via Dynamic Feedback Linearization-based FIRM," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, Oct. 2012, pp. 4433–4440. [Online]. Available: http://ieeexplore.ieee.org/document/6385970/.

[44] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo, "OBPRM: An Obstacle-Based PRM for 3D Workspaces," *Proceedings of the Third Workshop on the Algorithmic Foundations of Robotics on Robotics : The Algorithmic Perspective: The Algorithmic Perspective*, pp. 155–168, 1998. [Online]. Available: files/1338/Amato%20et%20al.%20-%201998%20-%20OBPRM%20An%20Obstacle-Based%20PRM%20for%203D%20Workspaces.pdf%5Cnfiles/1339/summary.html%5Cnhttp://dl.acm.org/citation.cfm?id=298960.299002.

[45] K. Solovey and M. Kleinbort, "The Critical Radius in Sampling-based Motion Planning *," Tech. Rep., 2018. [Online]. Available: https://arxiv.org/pdf/1709.06290.pdf.

[46] S. M. LaValle, "Rapidly-Exploring Random Trees: A New Tool for Path Planning," *In*, vol. 129, pp. 98–11, 1998. [Online]. Available: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Rapidly-exploring+random+trees:+A+new+tool+for+path+planning#0.

[47] S. M. LaValle and J. J. Kuffner, "Rapidly-exploring random trees: Progress and prospects," *4th Workshop on Algorithmic and Computational Robotics: New Directions*, pp. 293–308, 2000. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.38.1387.

[48] S. Karaman and E. Frazzoli, "Sampling-based Algorithms for Optimal Motion Planning," pp. 1–76, 2011. [Online]. Available: http://arxiv.org/abs/1105.1186.

[49] O. Adiyatov and H. A. Varol, "Rapidly-exploring random tree based memory efficient motion planning," in *2013 IEEE International Conference on Mechatronics and Automation, IEEE ICMA 2013*, 2013, pp. 354–359.

[50] ——, "A novel RRTstar-based algorithm for motion planning in Dynamic environments," in *2017 IEEE International Conference on Mechatronics and Automation, ICMA 2017*, IEEE, 2017, pp. 1416–1421.

[51] A. Stentz, "Optimal and Efficient Path Planning for Unknown and Dynamic Environments," *International Journal of Robotics and Automation*, vol. 10, no. August, pp. 89–100, 1993.

[52] S. Koenig and M. Likhachev, "D* Lite," *Proceedings of the Eighteenth National Conference on Artificial Intelligence*, pp. 476–483, 2002.

[53] A. Stentz, "The Focussed D* Algorithm for Real-Time Replanning," Tech. Rep., 1995. [Online]. Available: http://www.frc.ri.cmu.edu/~axs/doc/ijcai95.pdf.

[54] H. Oleynikova, Z. Taylor, R. Siegwart, and J. Nieto, "Sparse 3D Topological Graphs for Micro-Aerial Vehicle Planning," no. Section III, 2018. [Online]. Available: http://arxiv.org/abs/1803.04345.

[55] D. H. Kim, N. T. Hai, and W. Y. Joe, "A Guide to Selecting Path Planning Algorithm for Automated Guided Vehicle (AGV)," 2018, pp. 587–596. [Online]. Available: http://link.springer.com/10.1007/978-3-319-69814-4_56.

[56] C. B. Barber, D. P. Dobkin, and H. Huhdanpaa, "The quickhull algorithm for convex hulls," *ACM Transactions on Mathematical Software*, vol. 22, no. 4, pp. 469–483, Dec. 1996. [Online]. Available: http://dpd.cs.princeton.edu/Papers/BarberDobkinHuhdanpaa.pdf.

[57] D. Meagher, "Octree Encoding: A New Technique for the Representation, Manipulation and Display of Arbitrary 3-D Objects by Computer," *Rensselaer Polytechnic Institute*, no. Technical Report IPL-TR-80-111, 1980.

[58] M. P. Aghababa, "3D path planning for underwater vehicles using five evolutionary optimization algorithms avoiding static and energetic obstacles," *Applied Ocean Research*, vol. 38, pp. 48–62, 2012. [Online]. Available: http://dx.doi.org/10.1016/j.apor.2012.06.002.

[59] D. Lee, H. Song, and D. H. Shim, "Optimal path planning based on spline-RRT* for fixed-wing UAVs operating in three-dimensional environments," in *2014 14th International Conference on Control, Automation and Systems (ICCAS 2014)*, IEEE, Oct. 2014, pp. 835–839. [Online]. Available: http://ieeexplore.ieee.org/document/6987895/.

[60] B. Akgun and M. Stilman, "Sampling heuristics for optimal motion planning in high dimensions," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, IEEE, Sep. 2011, pp. 2640–2645. [Online]. Available: http://ieeexplore.ieee.org/document/6095077/.

[61] H. Mohammed, M. Jaradat, and L. Romdhane, "RRT N: An improved rapidly-exploring random tree approach for reduced processing times," in *2018 11th International Symposium on Mechatronics and its Applications (ISMA)*, IEEE, Mar. 2018, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/8330126/.

[62] M. Elbanhawi, M. Simic, and R. N. Jazar, "Continuous Path Smoothing for Car-Like Robots Using B-Spline Curves," *Journal of Intelligent & Robotic Systems*, vol. 80, no. S1, pp. 23–56, Dec. 2015. [Online]. Available: http://link.springer.com/10.1007/s10846-014-0172-0.

[63] M. Candeloro, A. M. Lekkas, J. Hegde, and A. J. Sorensen, "A 3D dynamic Voronoi diagram-based path-planning system for UUVs," *OCEANS 2016 MTS/IEEE Monterey, OCE 2016*, 2016.

[64] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *49th IEEE Conference on Decision and Control (CDC)*, IEEE, Dec. 2010, pp. 7681–7687. [Online]. Available: http://ieeexplore.ieee.org/document/5717430/.

[65] M. Seda, "Roadmap Methods vs. Cell Decomposition in Robot Motion Planning," *Proceedings of the 6th WSEAS International Conference on Signal Processing, Robotics and Automation*, pp. 127–132, 2007. [Online]. Available: https://pdfs.semanticscholar.org/d4b6/2b89acbf9eb50782f04c4b38c47cac515dbe.pdf.

[66] M. Candeloro, A. M. Lekkas, and A. J. Sørensen, "A Voronoi-diagram-based dynamic path-planning system for underactuated marine vessels," *Control Engineering Practice*, vol. 61, no. February, pp. 41–54, 2017. [Online]. Available: http://dx.doi.org/10.1016/j.conengprac.2017.01.007.

[67] J. Y. Yen, "An algorithm for finding shortest routes from all source nodes to a given destination in general networks," *Quarterly of Applied Mathematics*, vol. 27, no. 4, pp. 526–530, 1970. [Online]. Available: http://www.ams.org/qam/1970-27-04/S0033-569X-1970-0253822-7/.

[68] M. Ester, K. Hans-Peter, S. Jorg, and X. Xiaowei, "Density-Based Clustering Algorithms for Discovering Clusters," *Comprehensive Chemometrics*, vol. 2, pp. 635–654, 2010.

[69] T. M. Chan, "A Minimalist's Implementation of the 3-d Divide-and-Conquer Convex Hull Algorithm," Tech. Rep., 2003. [Online]. Available: http://www.cs.uwaterloo.ca/~tmchan/..

[70] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[71] H. Oleynikova, Z. Taylor, M. Fehr, R. Siegwart, and J. Nieto, "Voxblox: Incremental 3d euclidean signed distance fields for on-board mav planning," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017.

[72] D. R. Canelhas, "Truncated Signed Distance Fields Applied To Robotics," PhD thesis, Örebro University, 2017. [Online]. Available: www.publications.oru.se.

[73] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees," *Autonomous Robots*, 2013. [Online]. Available: http://octomap.github.com..

[74] L. Carlone, J. Du, M. Kaouk Ng, B. Bona, and M. Indri, "Active SLAM and exploration with particle filters using Kullback-Leibler divergence," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 75, no. 2, pp. 291–311, Aug. 2014. [Online]. Available: http://link.springer.com/10.1007/s10846-013-9981-9.

[75] M. Mataric, "Integration of representation into goal-driven behavior-based robots," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 3, pp. 304–312, Jun. 1992. [Online]. Available: http://ieeexplore.ieee.org/document/143349/.

[76] L. Murphy and P. Newman, "Using incomplete online metric maps for topological exploration with the Gap Navigation Tree," in *2008 IEEE International Conference on Robotics and Automation*, IEEE, May 2008, pp. 2792–2797. [Online]. Available: http://ieeexplore.ieee.org/document/4543633/.

[77] A. Makarenko, S. Williams, F. Bourgault, and H. Durrant-Whyte, "An experiment in integrated exploration," in *IEEE/RSJ International Conference on Intelligent Robots and System*, vol. 1, IEEE, pp. 534–539. [Online]. Available: http://ieeexplore.ieee.org/document/1041445/.

[78] A. Zelinsky, "Using Path Transforms to Guide the Search for Findpath in 2D," *The International Journal of Robotics Research*, vol. 13, no. 4, pp. 315–325, Aug. 1994. [Online]. Available: http://journals.sagepub.com/doi/10.1177/027836499401300403.

[79] P. Norvig and S. J. Russel, *Artificial intelligence A modern approach*, 3rd. Harlow, UK: Pearson Education Ltd., Mar. 2010. arXiv: 9809069v1 [gr-qc]. [Online]. Available: http://www.journals.cambridge.org/abstract_S0269888900007724.

[80] R. Valencia and J. Andrade-cetto, *Springer Tracts in Advanced Robotics 119 Mapping , Planning and Exploration with Pose SLAM*, B. Siciliano and O. Khatib, Eds. Springer International Publishing, 2018, pp. 53–98.

[81] T. Tao, Y. Huang, F. Sun, and T. Wang, "Motion planning for SLAM based on frontier exploration," *Proceedings of the 2007 IEEE International Conference on Mechatronics and Automation, ICMA 2007*, no. 60605021, pp. 2120–2125, 2007.

[82] B. Tovar, L. Guilamo, and S. LaValle, "Gap navigation trees: Minimal representation for visibility-based tasks," *Algorithmic Foundations of Robotics VI*, 425 440, 2005. [Online]. Available: http://ai2-s2-pdfs.s3.amazonaws.com/317b/c5e157d946826dd894e370fb70ccde6aee1d.pdf%0Ahttp://www.springerlink.com/index/AD8BHRE0X1QKBT23.pdf.

[83] E. Murphy, "Planning and Exploring Under Uncertainty," PhD thesis, Somerville College, 2010, p. 240.

[84] S. Thrun and M. Montemerlo, "The graph SLAM algorithm with applications to large-scale mapping of urban structures," *International Journal of Robotics Research*, vol. 25, no. 5-6, pp. 403–429, 2006.

[85] L. Fermin-Leon, J. Neira, and J. A. Castellanos, "TIGRE: Topological graph based robotic exploration," in *2017 European Conference on Mobile Robots (ECMR)*, IEEE, Sep. 2017, pp. 1–6. [Online]. Available: http://ieeexplore.ieee.org/document/8098718/.

[86] C. Zhu, R. Ding, M. Lin, and Y. Wu, "A 3D Frontier-Based Exploration Tool for MAVs," in *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*, IEEE, Nov. 2015, pp. 348–352. [Online]. Available: http://ieeexplore.ieee.org/document/7372156/.

[87] D. Priyasad, Y. Jayasanka, H. Udayanath, D. Jayawardhana, S. Sooriyaarachchi, C. Gamage, and N. Kottege, "Point Cloud Based Autonomous Area Exploration Algorithm," in *2018 Moratuwa Engineering Research Conference (MERCon)*, IEEE, May 2018, pp. 318–323. [Online]. Available: https://ieeexplore.ieee.org/document/8421954/.

[88] D. Holz, S. Behnke, N. Basilico, and F. Amigoni, "Evaluating the Efficiency of Frontier-based Exploration Strategies," *ISR/Robotik*, no. 2, p. 8, 2010. [Online]. Available: https://www.vde-verlag.de/proceedings-en/453273006.html.

[89] E. Vidal, J. D. Hernandez, K. Istenic, and M. Carreras, "Optimized Environment Exploration for Autonomous Underwater Vehicles," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, May 2018, pp. 6409–6416. [Online]. Available: https://ieeexplore.ieee.org/document/8460919/.

[90] A. Lambert and D. Gruyer, "Safe path planning in an uncertain-configuration space," in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 3, IEEE, pp. 4185–4190. [Online]. Available: http://ieeexplore.ieee.org/document/1242246/.

[91] 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*, vol. 79, no. 3, p. 497, Jul. 1957. [Online]. Available: https://www.jstor.org/stable/2372560?origin=crossref.

[92] F. Lamiraux and J.-P. Lammond, "Smooth motion planning for car-like vehicles," *IEEE Transactions on Robotics and Automation*, vol. 17, no. 4, pp. 498–501, 2001. [Online]. Available: http://ieeexplore.ieee.org/document/954762/.

[93] A. M. Lekkas and T. I. Fossen, "Integral LOS Path Following for Curved Paths Based on a Monotone Cubic Hermite Spline Parametrization," *IEEE Transactions on Control Systems Technology*, vol. 22, no. 6, pp. 2287–2301, Nov. 2014. [Online]. Available: http://ieeexplore.ieee.org/document/6767080/.

[94] P. Costantini, T. N. T. Goodman, and C. Manni, "Constructing C 3 shape preserving interpolating space curves," Tech. Rep., 2001, pp. 103–127. [Online]. Available: https://link.springer.com/content/pdf/10.1023%2FA%3A1016664630563.pdf.

[95] A. M. Lekkas, "Guidance and Path-Planning Systems for Autonomous Vehicles," PhD thesis, Norwegian University of Science and Technology, 2014.

[96] R. W. Beard and T. W. Mclain, "Implementing Dubins Airplane Paths on Fixed-wing UAVs," in *Handbook for Unmanned Aerial Vehicles*, Springer, 2013, pp. 5–15.

[97] J. Binney, "Risk-aware Path Planning for Autonomous Underwater Vehicles using Predictive Ocean Models," vol. 30, no. 5, pp. 741–762, 2013.

[98] S. M. Zadeh, D. M. W. Powers, A. Yazdani, K. Sammut, and A. Atyabi, "Differential Evolution for Efficient AUV Path Planning in Time Variant Uncertain Underwater Environment," *arXiv preprint arXiv:1604.02523*, 2016. [Online]. Available: http://arxiv.org/abs/1604.02523.

[99] S. Mahmoud Zadeh, D. M. Powers, and K. Sammut, "An autonomous reactive architecture for efficient AUV mission time management in realistic dynamic ocean environment," *Robotics and Autonomous Systems*, vol. 87, pp. 81–103, 2017.

[100] Z. Zeng, K. Sammut, L. Lian, F. He, A. Lammas, and Y. Tang, "A comparison of optimization techniques for AUV path planning in environments with ocean currents," *Robotics and Autonomous Systems*, vol. 82, pp. 61–72, 2016. [Online]. Available: http://dx.doi.org/10.1016/j.robot.2016.03.011.

[101] D. S. Rao and S. B. Williams, *Large-scale path planning for Underwater Gliders in ocean currents*, 2009. [Online]. Available: https://www.semanticscholar.org/paper/Large-scale-path-planning-for-Underwater-Gliders-in-Rao-Williams/2d5c2dae59cee6a2664148026c5d1b768fa7075c.

[102] Z. Hong-han, G. Liming, C. Tao, W. Lu, and Z. Xun, "Global path planning methods of UUV in coastal environment," in *2016 IEEE International Conference on Mechatronics and Automation*, IEEE, Aug. 2016, pp. 1018–1023. [Online]. Available: http://ieeexplore.ieee.org/document/7558702/.

[103] I. Schjølberg, "3D Path Following and Tracking for an Inspection Class ROV," in *ASME 2017 International Conference on Ocean, Offshore and Arctic Engineering*, Trondheim, Norway, 2017, pp. 1–10.

[104] F. R. Lera, F. C. Garcia, G. Esteban, and V. Matellan, "Mobile Robot Performance in Robotics Challenges: Analyzing a Simulated Indoor Scenario and Its Translation to Real-World," *Proceedings - 2nd International Conference on Artificial Intelligence, Modelling, and Simulation, AIMS 2014*, pp. 149–154, 2014. [Online]. Available: https://ieeexplore.ieee.org/document/7102451/.

Appendices

A Path Smoothing Definitions

A.1 Path Smoothness

An often used criteria for evaluating paths is their smoothness. The notion of path smoothness can be divided into *geometric* and *parametric* continuity. Geometric continuity is in essence constraints on the speed along the curve and is defined based on the curve's geometrical properties [26]:

- G^0: All curves are joined
- G^1: The path-tangential angle is continuous
- G^2: The center of curvature is continuous

Parametric continuity is defined based on the the analytical properties of the curve [26]:

- C^0: All subpaths are connected (i.e. curves are joined)
- C^1: The first derivatives (velocities) of the curve are continuous
- C^2: The second derivatives (accelerations & curvatures) of the curve are continuous
- C^n: The nth derivatives of the curve are continuous

A.2 Supplementary Definitions 3D Spline Interpolation

The C^3 spline-based interpolation technique introduced in [94] builds on the concept of *shape preserving* curves. For the sake of completeness, their curve criteria are included here, adopting the notation from [94].

Let Q be a curve with curvature vector $K(t)$ and torsion vector $\tau(t)$. Furthermore, let $I_i \in \mathbb{R}^3 \;\; i \in [0, N]$ be interpolation points, where no two points I_i, I_j are identical, and $L_i =$

$I_{i+1} - I_i, \quad i \in [0, N-1]$. Vectors N_i can then be defined as

$$N_i = \begin{cases} \dfrac{L_{i-1} \times L_i}{\|L_{i-1} \times L_i\|} & \text{if} \|L_{i-1} \times L_i > 0\,, \\ 0 & \text{otherwise}\ . \end{cases}$$

Finally, let $\Delta_i = \det\left(\begin{bmatrix} L_{i-1} & L_i & L_{i+1} \end{bmatrix}\right)$.

Definition A.1 Shape preserving curve*:*
The curve, $Q(t)$, is said to be shape preserving *if the following criteria are satisfied:*

- Colinearity*:*
  If $\|N_i\| = 0$ and $L_{i-1} \cdot L_i > 0$, then $\dfrac{\|\dot{Q} \times L_j\|}{\|\dot{Q}\|}$, $j = i-1, i$ can be arbitratily reduced in each fixed compact subinterval of $(t_{i-1}, t_{i+1}$ where $\|\dot{Q}\| \neq 0$.

- Convexity*:*
  If $N_i \cdot N_{i+1} > 0$, then $K(t) \cdot N_j > 0$, $j = i-1, im\, t \in [t_i, t_{i+1}]$.
  If $N_i \cdot N_{i+1} < 0$, then $[K(t) \cdot N_j][N_l \cdot N_j] > 0$, $l, j = i, i+1$ and $K(t) \cdot N_j$ has precisely one sign change in $t \in [t_i, t_{i+1}]$, $j = i, i+1$.

- Coplanarity*:*
  If $\Delta_i = 0$, then $\left\| \dfrac{\dot{Q} \times \ddot{Q}}{\|\dot{q} \times \ddot{Q}\|} \times N_i \right\|$, $t \in [t_i, t_{i+1}]$, can be arbitrarily reduced if $\|K(t)\| \neq 0$.

- Torsion*:*
  If $\Delta_i \neq 0$, then $\tau(t)\Delta_i > 0, \quad t \in (t_i, t_{i+1})$.
  If $\Delta_{i-1}\Delta_i > 0$, then $\tau(t_i)\Delta_i > 0, \quad j = i-1, i$.

B Heuristic Search Algorithms

B.1 A* Pseudocode

Algorithm B.1 A* (G, P_{start},P_{goal})

```
1:  OpenSet = ∅                                  ▷ Set of all discovered nodes not visited
2:  ClosedSet = ∅                                ▷ Set of all visited nodes
3:  OpenSet = OpenSet ⋃ P_start
4:  P_start.g = 0
5:  P_start.h = Heuristic(P_start,P_goal)        ▷ Heuristic(.) can f.ex. be the Euclidean distance
6:  P_start.f = P_start.h
7:  while OpenSet ≠ ∅ do
8:      u = ExtractMinF(OpenSet)                 ▷ Get node with lowest f-score
9:      if u = P_goal then
10:         return OptimalPath
11:     OpenSet = OpenSet \ u
12:     ClosedSet = ClosedSet ⋃ u
13:     for each neighbour v of u do
14:         if v ∈ ClosedSet then
15:             Do Nothing                       ▷ Neighbour already visited, ignore
16:         v̄.g = u.g + Heuristic(u,v)           ▷ Calculate distance from start to neighbour
17:         if v ∉ OpenSet then
18:             OpenSet = OpenSet ⋃ v
19:         else if v̄.g ≥ v.g then               ▷ Not a better path
20:             Continue
21:         v.π = u                              ▷ Best path so far, update predecessor and f- & g-value
22:         v.g = v̄.g
23:         v.f = v.g + Heuristic(v,P_goal)
```

B.2 D*-Lite Pseudocode

Algorithm B.2 D*-Lite (G, v_{start}, v_{goal})

```
1:  function CalculateKey(v)
2:      return min{g(v), rhs(v)} + h(v_start, v) + k_m; min{g(v), rhs(v)}
3:  function UpdateVertex(u)
4:      if v ≠ v_goal then
5:          rhs(v) = min_{v' ∈ Succ(v)} {c(v, v') + g(v')}
6:      if v ∈ H then
7:          H.Remove(v)
8:      if g(v) ≠ rhs(v) then
9:          H.Insert(u, CalculateKey(u)
10: function ComputeShortestPath
11:     while H.TopKey( ) < CalculateKey(v_start) or rhs(v_start ≠ g(v_start) do
12:         k_old = H.TopKey()
13:         u = H.Pop( )
14:         if k_old < CalculateKey(u) then
15:             H.Insert(u, CalculateKey(u)
16:         else if g(u) > rhs(u) then g(u) = rhs(u)
17:             for v ∈ u.π do
18:         else
19:             g(u) = ∞
20:             for v ∈ u.π ⋃{u} do UpdateVertex(v)
21: function Main
22:     v_last = v_start
23:     H = ∅
24:     k_m = 0
25:     for v ∈ V do
26:         rhs(v) = g(v) = ∞                                   } Initialisation
27:     rhs(v_goal) = 0
28:     H.Insert(v_goal, CalculateKey(v_goal))
29:     ComputeShortestPath()
30:     while v_start ≠ v_goal do
31:         v_start = argmin_{v' ∈ Succ(v_start)}{ c(v_start, v') + g(v') }   ▷ If g(v_start = ∞), then no known
    path exists
32:         MoveTo(v_start)
33:         UpdatedEdges = ScanForChangedEdges()   ▷ Scan G for any change in edge costs
34:         if UpdatedEdges ≠ ∅ then
35:             k_m = k_m + h(v_last, v_start)
36:             v_last = v_start
37:             for Edge (p, q) ∈ UpdatedEdges do
38:                 Update edge cost c(p, q)
```

C Supplimentary Figures

This appendix contains a set of supplementary figures used as additional illustrations or examples of themes discussed in the main text.

C.1 Point Cloud Processing

This section simply contains an example of an authentic point cloud dataset (shown in fig. C.1), acquired from ETH ASL, Zurich [71], and its truncated signed distance field (fig. C.2).

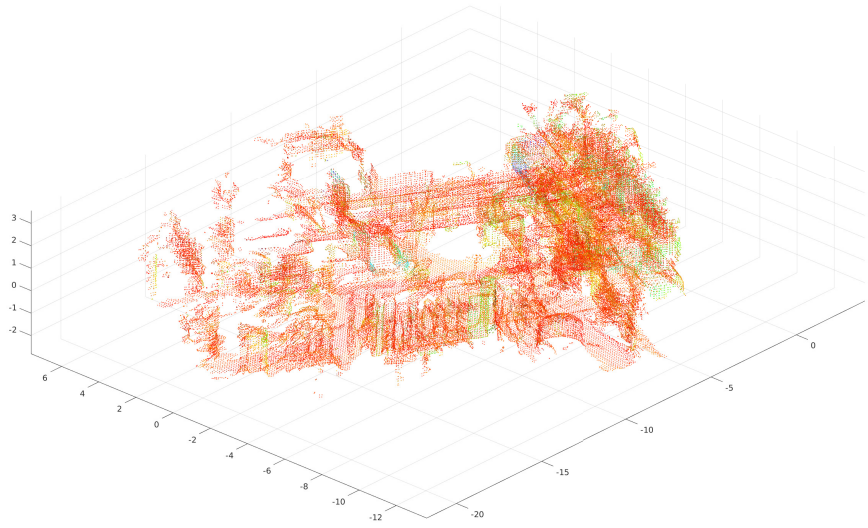


Figure C.1: Point cloud from ETH machine basement, courtesy of ETH ASL, Zurich [71].

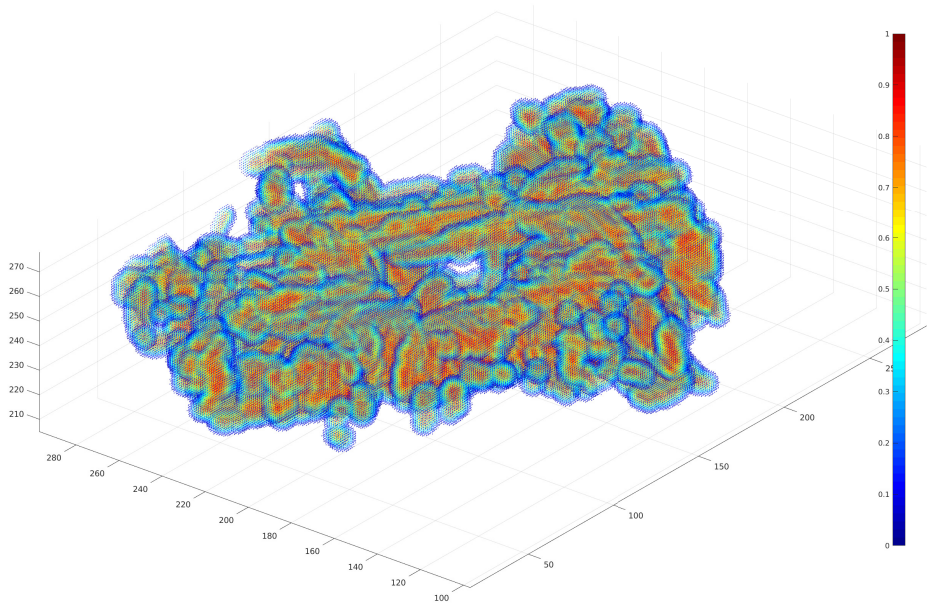


Figure C.2: The truncated signed distance field of the point cloud shown in fig. C.1. Blue parts indicate closeness to obstacles while the redder areas are more distant.

C.2 Distance Field GVD

This section contains additional test results of the distance field graph generation method presented in [54] shown in fig. C.3 with the generated path in fig. C.4.

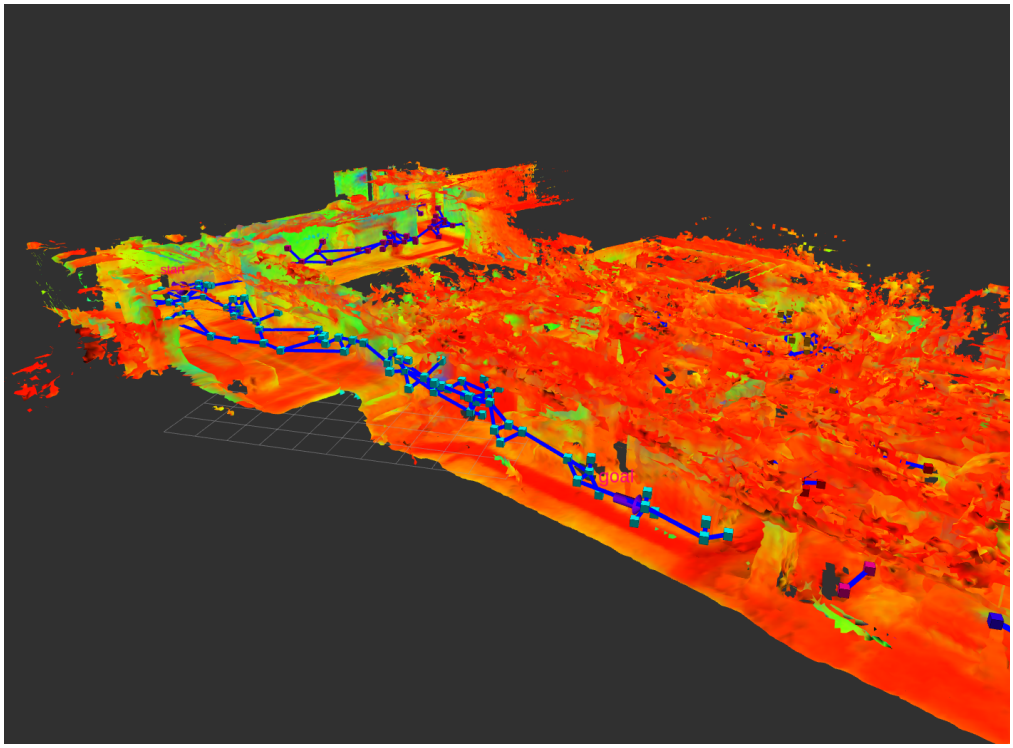


Figure C.3: Part of the generated sparse graph using the distance field method on a point cloud dataset calcuated using the voxblox library (github.com/ethz-asl/voxblox).

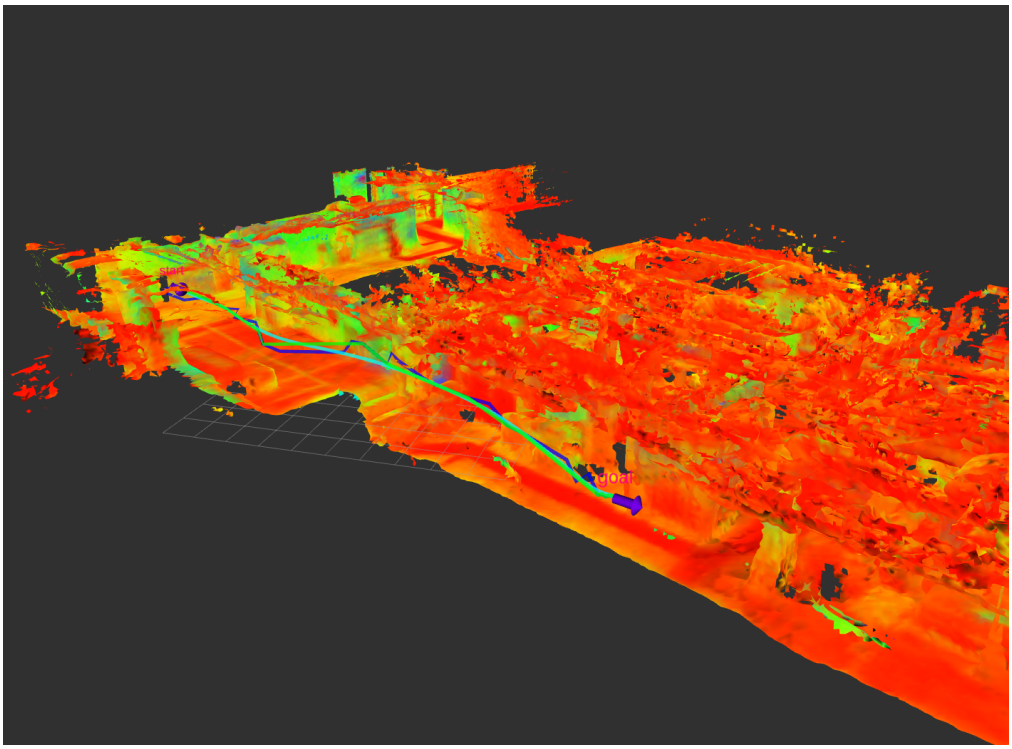


Figure C.4: The resulting path using the sparse graph shown in fig. C.3. Cyan indicate the smoothed path, blue is the path through the noes in the sparse graph and green is a simplified version of the blue path (redundant nodes removed, replaced by more straight lines).