Declaration of Authorship

I, Cebisile MTHABELA, declare that this dissertation titled, "Path planning approach for ground robots in 3D environments using sampling-based algorithms " and the work presented here within is my own. I affirm that:

- This work was done while registered at the University of the Witwatersrand for research degree.
- This work has not been submitted at any other University before for any degree or examination.
- The source is always given for the quoted work of others in this document. Without such quotations, this work is completely my own.
- All sources of help are acknowledged.

Signature: 

Date: 13/12/2021

Acknowledgements

My special gratitude goes to my supervisors, Daniel Withey and Chioniso Kuchwa-Dube for their patient guidance and continuous support during this study. This research project would not have been a success without their exceptional support. Their enthusiasm, deep knowledge in the field, and exacting attention to detail have kept my work on track. My gratitude extends to the Council for Scientific and Industrial Research (CSIR) for the funding opportunity to undertake my studies at the University of Witwatersrand. Additionally, I am grateful to all my colleagues at the CSIR who have provided both professional and personal guidance. They taught me about both scientific research and life in general. It is their kind help and support that have made my study and life at the CSIR a wonderful time. They have taught me much more than I could ever credit them for. I am also grateful to everyone, who has contributed to the completion of this project directly or indirectly. Finally, I would like to express my gratitude to my mother (Thulile E. Mlambo) and my siblings. Without their enormous understanding and support, it would have been impossible for me to complete this study.

Contents

List of Figures

List of Tables

List of Abbreviations

CSIR	**C**ouncil for **S**cientific and **I**ndustrial **R**esearch
RRT	**R**apidly-exploring **R**andom **t**ree
PRM	**P**robabilistic **r**oad**m**ap
DOF	**Degree** of **F**reedom
3D	**T**hree *Dimensional*
2D	**T**wo *Dimensional*
$\mathcal{C}$**-space**	**C**onfiguration space
ARA*	**A**nytime **R**epairing **A***
AD*	**A**nytime **D***
BFS	**B**readth **F**irst **S**earch
FIFO	**F**irst **I**n **F**irst **O**ut
CGAL	**C**omputational **G**eometry **A**lgorithms **L**ibrary

List of Symbols

q	Position	
$\mathcal{W}$	Workspace	
$\mathcal{R}$	Rigid body	
$\mathbb{R}$	Set of real numbers	
n	Dimension	
G	Graph	
T	Tree	
V	Set of vertices	
E	Set of edges	
$\mathcal{Q}$	Set of positions	
γ	Continuous function	
$\dot{q}$	Transition state	
t	time span	
ρ	Distance metric	
$\mathcal{U}$	Set of control input	
$\measuredangle$	Angle	
N	Surface normal	
$\mathcal{M}$	Mesh	
$\mathcal{F}$	Set of geometric regions(faces/triangles)	
ϵ	Step size	m
r	Radius	m
$rand_r$	Random point's radius	m
πr^2	local region area	m^2

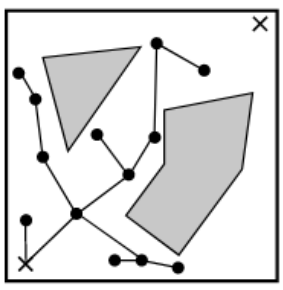
Chapter 1

Introduction

Path planning for mobile robots has attracted the attention of many researchers over the past decade, as robots have started to play a vital role in industry and our daily life. Path planning can be described as a navigation problem that a robot has to undertake correctly to move from one configuration to another. It is essential for a robot to perform this task effectively. There are three general navigation problems: localisation, path planning, and motion control (Sariff and Buniyamin, 2006). Given the map and a goal position, the path planning problem seeks to find an optimal collision-free path from the initial position to the goal position in configuration space.

The study of autonomous mobile robot path planning has attracted the attention of many researchers since the 1970s (Sariff and Buniyamin, 2006) due to its wide application in the industry. Several path planning algorithms have since been developed. These algorithms can be classified into two categories, graph-based and sampling-based depending on how the environment is represented and explored. Graph-based algorithms such as A* (Hart, Nilsson, and Raphael, 1968) and Dijkstra's (Dijkstra, 1959) algorithm have been widely used to find the shortest path in the path planning problem. Graph-based algorithms are simple to implement. However, the path produced by a graph-based algorithm usually has unnecessary turns. These algorithms use the idea of cell decomposition to decompose the workspace into a finite number of contiguous cells (Russell et al., 2003). A regularly spaced grid such as the Sukharev grid shown in Figure 1.1a (LaValle, 1998) which places a vertex at the centre of each cell to obtain good

(A) The Sukharev grid (adapted from LaValle, 1998).



(B) A tree produced by connecting the randomly sampled free space (adapted from Noreen, Khan, and Habib, 2016b).

FIGURE 1.1: Two different environment representations

results is a simple example of cell decomposition. The path planning problem is therefore reduced into a discrete graph-search problem (Russell et al., 2003; LaValle, 2006). These algorithms explore among the set of cells in the grid where all the information about the environment is contained. The advantage of cell decomposition is that it is simple to implement. However, some cells are neither free nor occupied and it is always an issue when a path must cross over these cells. A solution path that includes such cells might not be a real solution, because it might be difficult to cross the cell in the desired direction in a straight line (Russell et al., 2003). These algorithms are focused on the completeness of the path, such as resolution complete (LaValle, 2006), meaning that the algorithm will find a solution if and only if the solution exists in a finite amount of time. However, the performance of these algorithms degrades in high dimensional spaces. The running time for graph-based algorithms increases exponentially with the dimensionality of the state space (Yang et al., 2016).

As a result, sampling-based algorithms were developed to overcome the complexity of deterministic planning algorithms for robots with a high number of degrees of freedom (DOF). These algorithms have been shown to work well in complex environments (Karaman and Frazzoli, 2010; Karaman and Frazzoli, 2011; LaValle, 1998). Sampling-based algorithms are unique in the sense that planning occurs by randomly sampling the workspace to find robot configurations within the free space. A path planner then creates a continuous connection between the sampled points as shown in Figure 1.1b (Noreen, Khan, and Habib, 2016b). Although they are not complete, they guarantee probabilistic completeness in the sense that the probability of getting a solution tends to unity as the number of samples tends to infinity. The most common sampling-based algorithms are Probabilistic Roadmaps (PRMs) and Rapidly-exploring Random Trees (RRTs). PRMs are primarily multi-query methods, in which several planning problems are solved in the same environment. Even though multi-query methods are valuable in complex environments, most problems do not require multi-queries. RRT is the counterpart of PRM. It is a single query method that quickly returns a feasible solution due to its incremental nature. These algorithms have attracted the attention of many researchers in recent years. However, they do not focus on the quality of the solution, they only find a feasible path. To overcome this, Karaman and Frazzoli (2010) introduced the first asymptotically optimal single-query sampling-based method, RRT*. The development of optimal sampling-based planners have renewed the interest of many researchers not only in robotics but also in other disciplines such as computational biology, computer animation, and verification (Branicky et al., 2006; Bhatia and Frazzoli, 2004; Latombe, 1999).

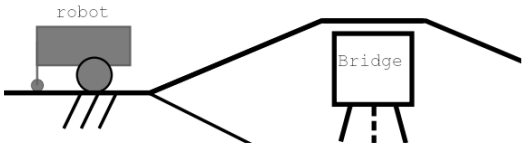


FIGURE 1.2: An example of a bridge in which a robot can choose to pass over or under the bridge (adapted from Škoda and Barták, 2017).

1.1 Motivation and Aims

With advances in technology and research, modern robots can operate without or with little guidance from humans. Moreover, robots appear in our everyday life, and they face unstructured and uncertain environments. Due to the increase of robotic applications in industry, robots occupy space in human environments. Therefore, the ability to navigate in three-dimensional (3D) cluttered environments is of supreme importance. Path planning is a strategic competency in which a robot must decide what to do to achieve its goal. Robots are usually given partial knowledge about the environment and a set of tasks to perform. They are equipped with sensors to frequently update changes in the environment as they move from one point to another.

Path planning in two-dimensional (2D) for mobile robots is the most studied problem. Though 2D planning algorithms are simple to implement, planning in 2D is not sufficient when the real world is considered. Mathematical methods such as interpolation and cubic splines have been used to overcome the typical angular defects of graph-based methods (Škoda and Barták, 2018). By adding an elevation into a 2D environment researchers have been able to solve the path planning problem in 2.5D (Pütz et al., 2016; Yang et al., 2016), but still, this is not sufficient for robots to fully operate in complex environments. An elevation grid cannot be used to represent non-spherical topologies, such as bridges or tunnels. Current research in 3D path planning has concentrated on aerial and underwater robots which are not constrained to the ground surface (De Filippis, Guglieri, and Quagliotti, 2012; Xu et al., 2015; Petres et al., 2007). Aerial and underwater robots do not need to consider traversability on a surface. Little work has been done in 3D planning for ground robots. Meanwhile, the complexity of the real world requires ground robots to be able to operate in 3D environments. Due to the layered structure of the surface with different running costs for different parts of the environment such as uphill, downhill, and flat surfaces. Simplification of the robot workspace into a simple 2D environment causes loss of some information which can prevent robots from operating correctly in more complex environments. For example in Figure 1.2 (Škoda and Barták, 2017), the robot has to pass through the bridge to reach its goal position. There are two ways that the robot can do this, it can either choose to go under the bridge or on top of it (Škoda and Barták, 2017). In a 2D environment, the bridge would likely be considered an obstacle. The inability of the existing path planning algorithms limits the usefulness of robots in real world applications. Therefore, planning

algorithms for ground mobile robots in uneven environments are urgently needed.

3D surface meshes can allow efficient computation for several navigation procedures (Kallmann, 2010). Meshes are popular in gaming. Meshes are data structures designed to aid pathfinding in complex and cluttered environments (Kallmann and Kapadia, 2014). Triangular meshes can represent both 2D and 3D environments. However, in a 2D mesh, only obstacles are polygonal. This type of a mesh is mostly used in computer games. A 3D mesh in which the environment is directly represented is employed in this study. Planning on 3D meshes allow for easy object identification and non-spherical object can be identified as they are reducing the challenge of planning in complex cluttered environments into a theoretical graph problem (Pütz et al., 2016).

The difficulty of planning in 3D environments increases exponentially, and the execution time also increases due to the complexity of the environment. However, since the introduction of sampling-based algorithms, planning in high dimensional spaces with constraints has become easy. Aiming at 3D local path planning for ground robots, in this project the RRT base method for planning in 3D surface meshes is presented. RRT has gained the interest of many researchers due to its ability to explore a wide area of the workspace, thus quickly converging towards the goal and it is easy to implement than its variants.

1.2 Definition of terms

In this section, some important concepts in path planning that are needed for further discussion are defined, which are the workspace and configuration space of robots. The path planning problem is formally defined and the difference between path planning and motion planning is discussed. In many cases, motion planning and path planning are often used interchangeably. Although these problems look similar, they are slightly different. The path planning problem seeks to find a collision-free path for the robot without considering the time the robot takes to reach the goal configuration and speed. Motion planning, at times referred to as trajectory planning, finds the path for the robot considering its geometry, motion, and time (Yang et al., 2016).

Robots are assumed to work in a two or three-dimensional space $\mathcal{W} = \mathbb{R}^n$ known as a workspace, where $n = 2, 3$. This workspace which is also known as the world is the actual physical space where the robot operates and is usually filled with obstacles. The main goal for path planning is to find a collision free path for a robot from the initial configuration to the goal configuration. To achieve this path planning for mobile robots is usually done in the configuration space $\mathcal{C}$ also known as C-space. It is the space of all possible configurations of the robot. The concept of C-space planning was introduced by Lozano-Perez (1983) to simplify difficult path planning scenarios in the workspace. In a configuration space $\mathcal{C}$, q specifies the robot's position and its orientation, $q \in \mathcal{C}$ (LaValle and Kuffner, 2001). The advantage of using the C-space is that the complex

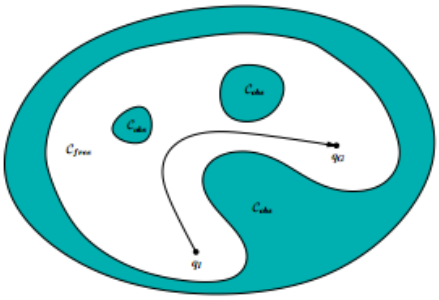


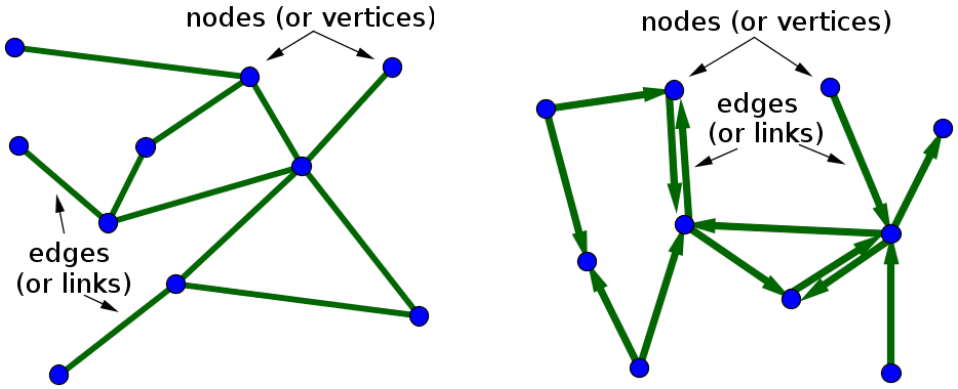
FIGURE 1.3: A simple path planning problem in a C-space containing obstacles (adapted from LaValle, 2006).

geometric shape of the robots can be mapped into a single point in $\mathcal{C}$ (Kavraki and LaValle, 2008). The minimum number of parameters required to specify the configuration or the dimensions of the C-space is determined by the degrees of freedom of a robot system. For example, a configuration of a robot in a plane can be specified by a point (x, y). Therefore, the dimensions of the C-space will be $\mathbb{R}^2$. A configuration space $\mathcal{C}$ is considered a high dimensional space if the parameters of q are greater than 3 where $q \in \mathcal{C}$.

Let the closed set $\mathcal{O}$ which is a proper subset of $\mathcal{W}$, $(\mathcal{O} \subset \mathcal{W})$ be an obstacle region, and the configuration space $\mathcal{C}$ be a set of all configurations q. The rigid body of the robot is given by $\mathcal{R}(q) \subset \mathcal{W}$, where $q \in \mathcal{C}$ is the current position of the robot. For example, in $\mathbb{R}^2$ workspace $q = (x_t, y_t)$ and $q = (x_t, y_t, z_t)$ in $\mathbb{R}^3$ workspace. In the configuration space $\mathcal{C}$, the obstacle region $\mathcal{C}_{obs} = \{q \in \mathcal{C} | \mathcal{R}(q) \cap \mathcal{O} \neq \emptyset\}$ is also a closed set since $\mathcal{O}$ and $\mathcal{R}(q)$ are closed sets in $\mathcal{W}$, containing all configurations q where the robot collides with the obstacle region. The obstacle-free region $\mathcal{C}_{free} = \mathcal{C} \backslash \mathcal{C}_{obs}$ is then the set of all configurations that are entirely contained within $\mathcal{W}_{free}$ (LaValle, 2006). The basic planning problem can be defined by a tuple $(q_i, q_g, \mathcal{C}_{free})$ (Yang et al., 2016). This problem is illustrated in Figure 1.3 (LaValle, 2006). Schwartz and Sharir (1983) have shown that solving this problem is NP-hard with unbounded $\mathcal{C}$ dimensions.

The ability to move from one point to another is a crucial skill for autonomous robots. Feasibility and optimality are used to describe the quality of the path based on the states and actions that were executed to reach the goal (LaValle, 2006). A feasible path is not always efficient. It only finds a path to the goal paying no attention to the quality of the path, whereas an optimal path optimizes the solution of the path subject to given constraints. Optimality of the path may include distance, energy, and the time to the goal position. Most of the recent research focuses on getting a feasible path. Below are formal definitions of these terms.

Definition 1: Graph
A graph is defined as a pair $G = (V, E)$, where V is a finite set of vertices or nodes q and E is a finite set of edges e. The size of G is the number n of vertices V and the order of G is the number m of edges E. A graph can be undirected



(A) Undirected graph containing 10 nodes and 11 edges.

(B) Directed graph containing 10 nodes and 13 edges.

FIGURE 1.4: Graph definition (adapted from Nykamp, 2020).

as in Figure. 1.4a (Nykamp, 2020) or directed and contain a cycle which is a closed walk with $q_i = q_n$, as an example see Figure. 1.4b (Nykamp, 2020). In Figure. 1.4b the arrows point to the direction in which a robot can move towards from a specific node.

Definition 2: Tree
A tree is an undirected acyclic connected graph. In other words, a tree does not contain even a single cycle. Each node has a set of child nodes in a tree. Sometimes this set can be empty, and the node will then be called a leave since it has no children. Each child node only has one parent node except for the root node which is the first node of the tree. The edges of the tree are known as branches. Figure. 1.5 (Black, 2017) shows an example of a tree.

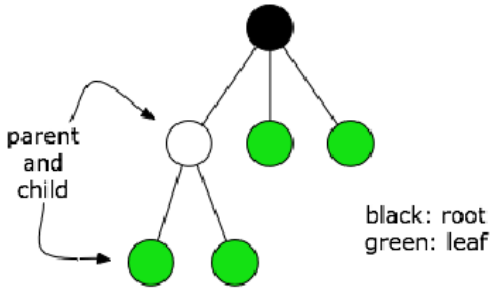


FIGURE 1.5: A simple tree with 6 nodes and 5 edges (adapted from Black, 2017).

Definition 3: Path
Given an initial configuration, $q_{init} \in \mathcal{C}_{free}$ and a set of goal configurations $\mathcal{Q}_{goal} \subset \mathcal{C}_{free}$. A path is a continuous function $\gamma : [0,1] \rightarrow \mathbb{R}^n$. It is a collision-free path if it is a path and $\gamma(t) \in \mathcal{C}_{free}$ for all $0 \leq t \leq 1$ (Yang et al., 2016). It is a feasible path if it is a collision-free path with $\gamma(0) = q_{init}$ and $\gamma(1) = (q_{goal})$.

Definition 4: Optimal Path
Given a cost function $C : \sum \rightarrow \mathbb{R} \geq 0$ for every path in a set of paths $\sum$. If Definition 3 is fulfilled for any given path planning problem $(q_{init}, q_{goal}, \mathcal{C}_{free})$, and if γ' minimizes the cost function of γ, $(C(\gamma') = min(C(\gamma)))$,where γ is the set of all feasible paths, γ' is then the optimal path (Yang et al., 2016).

Definition 5: Trajectory
A trajectory is a time-parameterized continuous path satisfying nonholonomic constraints (LaValle, 2006). Given a state transition equation $\dot{q} = f(q,u)$, where u is the control input in a set of all control inputs $\mathcal{U}$, $u \in \mathcal{U}$. This equation allows for configuration changes using control input. The motion planning problem finds an action trajectory $u : [0,T] \rightarrow \mathcal{U}$ that results into a collision-free trajectory from an initial configuration $q(0)$. The trajectory $q(t)$ where $t \in [0,T]$ is defined as:

$$q(t) = q(0) + \int_0^t f\big(q(\tau), u(\tau)\big) d\tau \tag{1.1}$$

Definition 6: Completeness
Let A be an Algorithm.

- *Complete*: A is complete if in a finite amount of time A always finds a solution if one exists. Otherwise, A states that the solution does not exist.
- *Resolution complete*: A is resolution complete if in a finite amount of time and for some small resolution step $\epsilon > 0$, A always finds a solution if one exists, otherwise, A states that the solution does not exist.
- *Probabilistically complete*: A is Probabilistically complete if the chances of finding the solution if the solution exists, approaches 1 as the time approaches infinity.

For both path planning and motion planning problems, the important task is finding a feasible path or trajectory. In this project, the study of the path planning problem which does not include the dynamics of the robot is conducted. Finding a feasible path for a robot in a 3D space is a challenging task on its own.

1.3 Research Questions and Objectives

Questions

Path planning in 3D environments is one of the difficult problems in robotics, with more work being done for aerial and underwater robots which are not constrained onto a surface. This project aims to answer the following questions for ground robots:

- How effective is the RRT in 3D path planning for ground robots on a 3D mesh surface?
- How can RRT on the mesh computation be improved?

Objectives

The long-term goal for this research project is to develop and implement a fast path planning algorithm for wheeled robots in 3D environments based on RRT. To achieve this goal, the following objectives are outlined;

- To develop a synthetic 3D mesh surface as the robot's workspace for simulation,
- To design and implement a 3D planning algorithm based on RRT,
- To assess this algorithm in different environments,
- To evaluate the effectiveness of the proposed algorithm by comparing it with variant planning methods on a synthetic mesh and real mesh surfaces.

1.4 Overview of Dissertation Contributions

Path planning methods presented in this research project have been inspired by the RRT's ability to quickly explore a vast area of a robot's workspace and by the increasing application of robotics in many areas. The anticipated contributions of this research project are as follows:

- Path planning on 3D surface meshes
  - An implementation of RRT on a mesh surface based on the computation of geodesics to ensure that all points and paths lie on the surface of the mesh.
- Fast path computation on a mesh (Cebisile, Daniel, and Chioniso, 2021b)
  - A new version of RRT that uses local regions formed around each node to optimize the speed of RRT on a mesh is presented.

1.4.1 Publications

Part of this research (section 4.3) was published in a peer-reviewed conference.

- Mthabela Cebisile, Withey Daniel, and Kuchwa-Dube Chioniso (2021b). "RRT based path planning for mobile robots on a 3D surface mesh". In: *2021 Southern African Universities Power Engineering Conference - Robotics and Mechatronics - Pattern Recognition Association of South Africa (SAUPEC/RobMech/PRASA)*, pp. 1–6
- Mthabela Cebisile, Withey Daniel, and Kuchwa-Dube Chioniso (2021a). "Ground Robot Path Planning on 3D Mesh Surfaces Using Bi-directional RRT based on Local Regions". In: *2021 Rapid Product Development Association of South Africa - Robotics and Mechatronics - Pattern Recognition Association of South Africa (RAPDASA-RobMech-PRASA)*, pp. 1–6.

1.5 Dissertation Outline

A review of path planning methods is provided in Chapter 2, focusing only on graph-based methods and sampling-based methods. Chapter 3 introduces the geodesic distances on mesh surfaces, basic RRT, RRT on a mesh surface (RRT-M), RRT on a mesh surface with local regions (RRT-ML), the basic RRT-Connect (Kuffner and LaValle, 2000) on a mesh (RRT-CM) and RRT-Connect with local regions (RRT-CML). The simulations of the RRT-M, RRT-ML, RRT-CM and RRT-CML algorithms are presented and compared to each other in Chapter 4. The discussion about simulation results is described in Chapter 5. Finally, the conclusion and future work are presented in Chapter 6.

Chapter 2

Related Work

The study of path planning in robotics has been ongoing for at least three decades. Since the early 1970s, a number of path planning algorithms have been developed. These algorithms can be classified into different categories depending on how the environment is represented and explored. In this chapter, a brief review and background of some existing path planning methods are presented with emphasis on graph-based and sampling-based methods. These methods are different from each other and have unique properties. Psuedocodes are used to describe these planning methods with reference to material from Chapter 2 and Chapter 5 of LaValle's book (LaValle, 2006). These psuedocodes shows the logic that is followed when implementing the algorithms.

2.1 Graph-Based Algorithms

Early approaches towards path planning for robots were graph-based. Graph-based algorithms such as A* and Dijkstra's are well known for finding the shortest paths in a grid representing the working environment and for robots with fewer degrees of freedom. These algorithms can successfully find an optimal path in a finite discrete space such as the grid space shown in Figure 1.1a and are considered to be a special form of dynamic programming (Bellman, 1957; LaValle, 2006). A* computes the shortest paths almost the same way as Dijkstra's algorithm except that it guides its search towards the least cost nodes, potentially reducing the computational time. The following section introduces the Breadth-first search algorithm which is seen as a basis of the two algorithms mentioned above. Sections 2.1.2 and 2.1.3 further discuss how these two algorithms work in more detail.

2.1.1 Breadth-first search

Breadth-First search, also known as BFS, is a simple strategic search algorithm in which the root node is explored first, then all its neighbouring nodes. BFS explores all unexplored neighbours for every node continuously until the goal node is found. In BFS, nodes are expanded in the order of proximity to the root node (Siegwart, Nourbakhsh, and Scaramuzza, 2011). Proximity here is defined as the shortest number of edge transitions. Generally, in a search tree, all nodes at a given tree depth are explored first before moving to the next level. The BFS

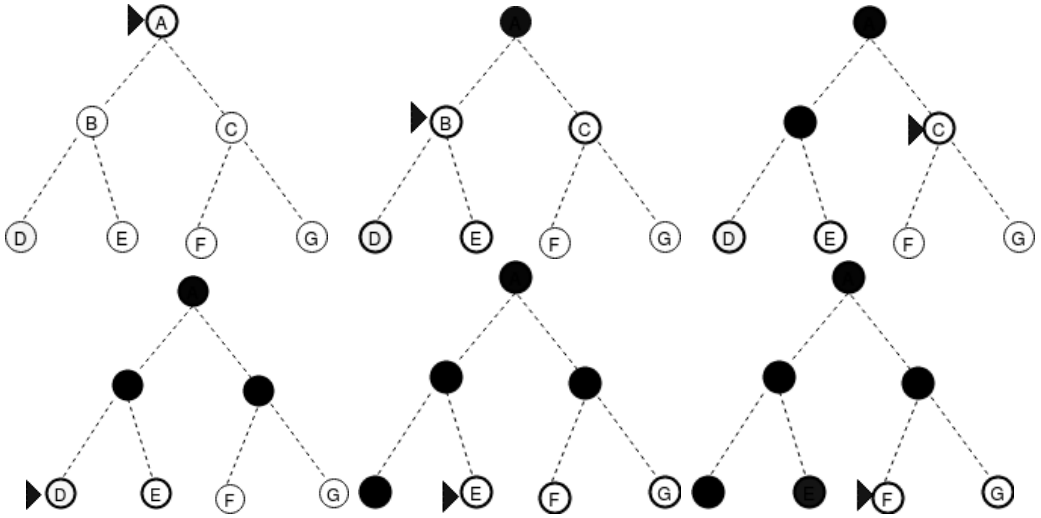


FIGURE 2.1: BFS on a binary tree, at each level the node to be expanded is pointed by an arrow and node that has been explored is indicated by black. Nodes to be explored next have thick lines.

method uses the First-In-First-Out (FIFO) queue principle to ensure that nodes that are visited first are expanded first (LaValle, 2006; Russell et al., 2003), Figure 2.1 shows the process of the BFS in a simple binary tree. The return value of the BSF is always the path with a fewer number of edges from the root node to the goal node. If the cost for all edges is constant say 1, and that is the returned minimum path cost then in that sense the BFS is optimal. However, the optimality of this method is not guaranteed if the graph has different costs associated with each edge (Siegwart, Nourbakhsh, and Scaramuzza, 2011). For complex search problems, BFS will typically require more time to explore all the nodes and in turn uses large memory space to keep track of visited nodes with $O(q + e)$ time complexity (Siegwart, Nourbakhsh, and Scaramuzza, 2011), where q and e are the numbers of vertices (nodes) and edges respectively.

2.1.2 Dijkstra's Algorithm

Dijkstra (1959) introduced Dijkstra's algorithm, named after its inventor, to allow optimal pathfinding in nonuniform cost maps (Siegwart, Nourbakhsh, and Scaramuzza, 2011). This algorithm works almost the same as the BSF described in section 2.1 except that edge cost $g(q)$ is assumed to be positive and this adds to the complexity of the algorithm. Here neighbours of the expanded node are stored in a list and the node with the least cost in the list is extracted and explored next. Due to this, the complexity of Dijkstra's algorithm rises to $O(q \log(q) + e)$ (Siegwart, Nourbakhsh, and Scaramuzza, 2011). The path from the initial node to the goal node is backtracked from the goal and is of a minimal total cost, where the total cost $f(q)$ is the sum of all edge costs $g(q)$ that form the path.

$$f(q) = g(q) + h(q) \tag{2.1}$$

2.1.3 A-star (A*)

A* (Hart, Nilsson, and Raphael, 1968) is the most commonly used graph-based algorithm due to the fact that it guarantees optimal and complete solution, provided that the heuristic function $h(q)$ is admissible and consistent (Russell et al., 2003), where q represents a node. It is an extension of Dijkstra's algorithm with some characteristics of BFS. The A* algorithm introduces a heuristic into a regular graph-searching algorithm which reduces the number of visited nodes in the quest of finding the shortest path from the initial node q_i to the goal node q_g. Characteristics of A* include building two lists, a closed list, and an open list, to keep track of nodes that have already been visited, and those that are to be visited next and adjacent to the explored nodes, respectively. Each node q is associated with an approximation $h(q)$ cost from the current node q to the goal position. A heuristic $h(q)$ is admissible only when $h(q)$ does not overestimate the cost to the goal. It is consistent if, for every node q and every adjacent node q' the estimated cost from q to the goal is not greater than the cost $c(q, q')$ from q to q' plus the estimated cost from q' to the goal (Russell et al., 2003; Costa and Silva, 2019): $h(q) \leq c(q, q') + h(q')$. A* evaluate nodes to be expanded next using the evaluation function in equation 2.1, sometimes called the $f-$values (Russell et al., 2003), which is a sum of the accumulated cost $g(q)$ from the start node to node q and the cost from node q to the goal node $h(q)$. A* is referred to as a heuristic search or informed search because its search is biased towards the goal based on the estimated total cost and initially

Algorithm 1 *A* Algorithm*

Input: A graph $\mathbf{G} = (V, E)$ with an initial node q_i and a goal node q_g.

Initialise:
for each $q \in G$ **do**
    $g(q_i) = \infty$
$Open\_List = q_i$
$Closed\_List = \emptyset$
$g(q_i) = 0$
$h(q_i) = heuristic(q_i, q_g)$
$f(q_i) = g(q_i) + h(q_i)$
while $Open\_List \neq \emptyset$ **do**
    $q = Open\_List.PopLowestCost()$
    $Closed\_List.Push\_Back(q)$
    if $q == q_g$ **then**
        Path = ExtractPath()
    for each $q' \in child(q)$ **do**
        if $q' \in Closed\_List$ **then**
            continue
        cost = $g(q) + distance(q, q')$
        if $cost < g(q')$ **then**
            $g(q') = cost$
            $parent(q') = q$
            $h(q') = heuristic(q', q_g)$
            $f(q') = g(q') + h(q')$
            $Open\_List.Push\_Back(q')$
    return Path

avoids nodes which despite having a low $g(q)$, have a high total cost estimate. Due to the heuristic nature of the A*, the complexity of this algorithm depends on the heuristic function used and the structure of the graph (Siegwart, Nourbakhsh, and Scaramuzza, 2011). The pseudocode for A* is shown in Algorithm 1.

Several A* variants have been developed and successfully used for mobile robot path planning. In particular D*, D*-Lite, and Theta* which are an extension of A* has been widely used for planning in 3D. D* also known as the dynamic A* was originally introduced by Stentz (1994) as a fast path planner for partially known environments where the cost parameters change during the search. Unlike other grid-based algorithms, D* can re-plan quickly whenever an obstacle is discovered along the path. Koenig and Likhachev (2002) introduced D*-Lite which is fundamentally like D*. D*-Lite has been widely used since it is algorithmically simple and shorter than D* and that makes it easy to be implemented in a few lines of code. Theta* differs from A* by the way it selects the parent nodes for each node in the graph. In Theta*, if a node can be seen from the other node the path to that node is determined. Parents of these nodes do not need to be neighbours. De Filippis, Guglieri, and Quagliotti (2012) implemented Theta* in a 3D environment with orographic obstacles and compared its performance with A*. Their simulation experiments showed that Theta* takes longer to compute than A* on a grid but proves that Theta* finds paths that are not as constrained to the grid cell transitions. Therefore Theta* is more effective than A* when an optimal path is required.

If changes occur in an environment, graph planners find it difficult to quickly re-plan to find an optimal path within the time allocated for the planner to finish planning. Likhachev et al. (2008) introduced an Anytime Repairing A* (ARA*) which quickly finds a suboptimal solution and improves this solution as time goes on. ARA* reuses its previous results to find new paths in known environments. However, if the environment changes during the execution, like other graph search methods ARA* cannot re-plan quickly. Replanning algorithms such as Anytime D* (AD*) (Likhachev et al., 2008), Dynamic A* (Stentz, 1994) and Lifelong Planning A* (Koenig, Likhachev, and Furcy, 2004) can incrementally repair a solution when changes occur in the environment without having to re-plan from scratch. Most of the graph-based algorithms do not consider the kinematics of the robot and this has become an issue for non-holonomic robots since the solution that is obtained may be unrealistic. Ferguson and Stentz (2006) developed the interpolation-based planning algorithm for aerial and ground robots in 3D grids named Field-D*. 3D grid-based adjacency graphs have the same limitations as their 2D counterparts. Paths produced by grid-based algorithms are usually suboptimal and have unnecessary turning. By using interpolation Ferguson and Stentz (2006) were able to produce less costly and straighter paths compared to regular 3D grid-based algorithms. However, in this algorithm, they did not consider the optimality of the path but focused on the time complexity of the algorithm. For more accurate results in 3D environments, the effect of discretization of the workspace needs to be overcome.

Current research, (De Filippis, Guglieri, and Quagliotti, 2012; Xu et al., 2015; Petres et al., 2007) in 3D path planning has been concentrated on aerial and underwater robots. These systems do not need to consider traversability on a surface. Colas et al. (2013), presented a method that allows robots to navigate in 3D environments. This method uses point cloud data and assumes that the environment is known prior to planning and thus this system does not try to reconstruct the environment if changes occur. The performance of graph-based methods degrade in high dimensional spaces. They involve discretization of the workspace which becomes computationally expensive in high dimensions. Efficient discretization can be achieved at the expense of completeness (Elbanhawi and Simic, 2014).

Sampling-based algorithms such as RRTs and PRMs were introduced to overcome the typical angular defects of the graph-based methods. More detail on these randomized algorithms is discussed in the following section.

2.2 Sampling-based Algorithms

Sampling-based algorithms were developed to overcome the complexity of deterministic planning algorithms for robots with higher degrees of freedom (DOF). Karaman and Frazzoli (2010) proved that these algorithms work well in high dimensional spaces. The most popular sampling-based methods are the RRTs and PRMs. These randomized planning algorithms have the advantage of providing a fast solution for high dimensional problems with lower computational costs. Sampling-based algorithms are unique in the way that they discretely sample the configuration space (Elbanhawi and Simic, 2014) to find robot configurations within the free space. Then continuous connections are created between the sampled points. In that sense, the discretization of the workspace is not required. PRMs are known as multi-query planners where multiple queries are performed in the same environment. In contrast, RRTs are generally known as single query planners. Sampling-based planners have been used in fields other than robotics, *i.e,* digital animation, and computational biology (Latombe, 1999). Section 2.2.1, briefly discusses how the RRT algorithm works and two of its variants namely RRT-Connect and RRT* are presented in sections 2.2.2 and 2.2.3 respectively.

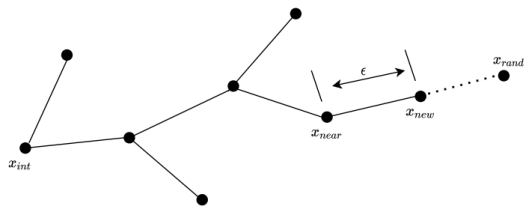


FIGURE 2.2: RRT tree expansion (adapted from LaValle and Kuffner 2001).

2.2.1 RRTs

Amongst other sampling-based algorithms, RRTs are the well-known single query algorithm due to their ability to rapidly explore a vast area of the robot's working space. LaValle (1998), introduced this algorithm as an attempt to solve path planning under holonomic, nonholonomic, and kinodynamic constraints and it can be used in high-dimensions. This algorithm incrementally grows a tree of feasible configurations, beginning from the start configuration to the goal configuration simply by randomly sampling the free space and attempting to grow the tree by connecting the closest node into the randomly selected node. The process of adding a new node (q_{new}) in a tree is shown in Figure 2.2 (Cebisile, Daniel, and Chioniso, 2021b). In each iteration, a random node (q_{rand}) is sampled and a step up to the specified step size (ϵ) is taken from the nearest node (q_{near}) toward q_{rand}, using the steering function. This forms q_{new}. If there are no obstacles between q_{near} and q_{new}, then q_{new} is added to the tree. The RRT construction algorithm is given in Algorithm.2, excerpted from (Noreen, Khan, and Habib, 2016b).

Algorithm 2 *RRT Algorithm*

1: $\mathbf{T} = (V, E)$
2: $\mathbf{T} \leftarrow$ **InitializeTree()**
3: $\mathbf{T} \leftarrow$ **AddNode(T,** q_{init}**)**
4: **for** $i = 0$ to $i = N$ **do**
5: $\quad q_{rand} \leftarrow$ **Sample**(i)
6: $\quad q_{near} \leftarrow$ **Nearest**$(\mathbf{T}, q_{rand})$
7: $\quad q_{new} \leftarrow$ **Steer**$(\mathbf{T}, q_{near})$
8: $\quad$ **if ObstacleFree**(q_{new}) **then**
9: $\quad\quad \mathbf{T} \leftarrow$ **AddNode(**q_{new}**,T)**
10: **return T**

2.2.2 RRT-Connect

Kuffner and LaValle (2000) invented RRT-Connect specifically for path planning problems with differential constraints. This algorithm uses a greedy heuristic function, Connect, that attempts to move over a long distance. RRT-Connect is based on a bi-directional search idea described by Pohl (1969) and grows two trees, with one starting from the initial node q_{init} and the other tree starting from the goal node q_{goal}. These two procedures reduce the importance of incremental motion (Kuffner and LaValle, 2000). The algorithm grows both trees towards each other in search of the common node, a node that belongs to both trees. Two nodes q and q', are considered to be the same if $\rho(q, q') < \epsilon$ for some metric ρ and small $\epsilon > 0$ (LaValle and Kuffner, 1999). Only one tree is grown in one iteration and the algorithm attempts to connect the nearest node to the other tree. If the attempt to connect the trees did not succeed, then the trees are swapped to reverse the roles. The algorithm stops when the first common node is found and thus the path from the initial node to the goal node is obtained. The greedy heuristic in the algorithm includes the rapid and uniform exploration

Algorithm 3

1: **function** EXTEND($\mathbf{T}, q$)
2: $q_{near} \leftarrow$ **Nearest**($\mathbf{T}, q$)
3: **if** **NewConfig**(q, q_{near}, q_{new}) **then**
4: **T** $\leftarrow$ **AddNode**(q_{new},**T**)
5: **if** $q_{new} = q$ **then**
6: **return** Reached
7: **else**
8: **return** Advanced
9: **return** Trapped

Algorithm 4

1: **function** CONNECT($\mathbf{T}, q$)
2: **repeat** **S** $\leftarrow$ **EXTEND**($\mathbf{T}, q$)
3: **until S** $\neq$ Advanced
4: **return S**

properties of the basic RRT.

The algorithm pseudocode for RRT-Connect is shown in Algorithm 5, it is adapted from the work of Kuffner and LaValle (2000) where t_1 and t_2 are proper subsets of the tree, $T = (V, E)$. This algorithm uses two functions described in Algorithm 3 and Algorithm 4. Firstly the trees are initialized with their respective root nodes, the first tree (t_1) is initialized with start (q_{init}) configuration as its root node, line 1 of algorithm 5 and the second tree (t_2) initially has the goal(q_{goal}) configuration has its root node, line 2 of algorithm 5. Random points are sampled from the entire workspace, $i = 0, ..., N$ is the number of iterations with maximum iteration-N. The function **EXTEND** check if either the common node is found or the tree is expanded. If one of the two is true then function **CONNECT** checks if q_{new} can be reached from the other tree, if this is valid then the path is found, otherwise, trees are interchanged in line 8, algorithm 5 and the whole process is repeated until the common node is found and trees are connected.

Algorithm 5 ***RRT_CONNECT*** *Algorithm*

1: $t_1 \leftarrow$ **InitializeTree**(q_{init})
2: $t_2 \leftarrow$ **InitializeTree**(q_{goal})
3: **for** $i = 0$ to $i = N$ **do**
4: $q_{rand} \leftarrow$ **Sample**(i)
5: **if** **EXTEND**(t_1, q_{rand})! $= Trapped$ **then**
6: **if** **CONNECT**(t_2, q_{new}) $= Reached$ **then**
7: **return** **PATH**(t_1, t_2)
8: **SWAP**(t_1, t_2)
9: **return** Failure

Algorithm 6 *RRT* Algorithm*

1: $\mathbf{T} = (V, E)$
2: V $\leftarrow \{v_{init}\}$
3: E $\leftarrow \{\emptyset\}$
4: **for** $i = 0$ to $i = N$ **do**
5: $\quad q_{rand} \leftarrow$ **Sample**(i)
6: $\quad q_{nearest} \leftarrow$ **Nearest**$(\mathbf{T}, q_{rand})$
7: $\quad q_{new} \leftarrow$ **Steer**$(\mathbf{T}, q_{near})$
8: $\quad$ **if ObstacleFree**(q_{new}) **then**
9: $\quad\quad q_{near} \leftarrow$ **Near**$(\mathbf{T}, q_{new})$
10: $\quad\quad q_{min} \leftarrow$ **ChooseParent**$(q_{near}, q_{nearest}, q_{new})$
11: $\quad\quad \mathbf{T} \leftarrow$ **AddNode**$(\mathbf{T}, q_{new})$
12: $\quad\quad \mathbf{T} \leftarrow$ **Rewire**$(\mathbf{T}, q_{near}, q_{min}, q_{new})$
13: **return T**

2.2.3 RRT-Star(RRT*)

The first asymptotically optimal sampling-based algorithms named PRM* and RRT* were introduced by Karaman and Frazzoli (2011). RRT* produces less jagged and shorter paths compared to RRT (Noreen, Khan, and Habib, 2016b). Unlike the RRT, RRT* guarantees to return the asymptotically optimal path, if one exists, as the number of sampled points approaches infinity. This algorithm has been mostly used in addressing optimal path planning in recent years (Noreen, Khan, and Habib, 2016b). This algorithm works like the RRT except that it has two unique processes: near-neighbour search and rewiring (Karaman and Frazzoli, 2010). Near-neighbour search and rewiring ensure that the algorithm converges to an optimal solution. The near-neighbour process returns the set of nearest nodes within the area of the ball radius, defined in equation 2.2.

$$k = \gamma \left(\frac{log(n)}{n} \right)^{\frac{1}{d}} \tag{2.2}$$

where γ is a constant based on the environment, n is the number of nodes in the tree and d is the dimension of the configuration space (Elbanhawi and Simic, 2014). To maintain the minimal cost path between the nodes, the rewiring process regenerates the tree within the radius, k. The path produced by the RRT* improves as the number of iterations increases due to its asymptotical behaviour as compared to RRT which does not improve its suboptimal path. The RRT* algorithm is shown in Algorithm 6, excerpted from (Noreen, Khan, and Habib, 2016a). Though this algorithm has better results than RRT due to the rewiring process its convergence rate is slower than that of the simple RRT (Noreen, Khan, and Habib, 2016a). RRT* spends time expanding the tree toward nodes that are not even promising to give an optimal solution (Elbanhawi and Simic, 2014).

RRT and its variants have been used to solve the path planning problem in a 3D environment. However, most studies in 3D environments were focused on Unmanned Aerial Vehicles (UAVs) (Yang and Sukkarieh, 2008) and Autonomous Underwater Vehicles (UAVs) (Hernández et al., 2015). These robots are not

constrained onto the surface. Hernández et al. (2015) used the Transition-based RRT (T-RRT) (Jaillet, Cortes, and Simeon, 2008) which is the variant of RRT that considers a cost function for online 3D path planning. In this paper (Hernández et al., 2015) an Octomap which is different from the mesh was used for workspace representation. The issue with occupancy maps like Octomap which is Octree based framework is that it does not define the trafficability of obstacles (Pütz et al., 2016). RRT is used for 3D path planning for a UAV utilizing cubic Bezier spiral curves for smoothing (Yang and Sukkarieh, 2008). To solve this problem in a 3D environment, Yang and Sukkarieh (2008) mapped the 3D space into a 2D then the problem became a 2D path planning problem, and the problem was re-mapped back to 3D. The work of (Aguilar and Morales, 2016) also addresses the problem of 3D path planning using a 3D binary occupancy grid to represent the robot's workspace using RRT based algorithm. Like planning in a 2D environment, path planning on an occupancy grid is not sufficient for full 3D path planning.

2.3 RRT Discussion

The introduction of the RRT* was a significant leap forward in optimal path planning for high dimensional complex problems (Noreen, Khan, and Habib, 2016a). However, RRT* takes longer to find the solution than the basic RRT method (Noreen, Khan, and Habib, 2016a). Hitherto, various RRT* based planning algorithms have been developed to speed up the convergence rate of the RRT*. RRT-Connect introduced in (Kuffner and LaValle, 2000), has shown fast convergence in most scenarios. RRT-Connect was evaluated against other RRT based planners based on the time it took to find the solution path (Kuffner and LaValle, 2000). However, RRT-Connect produces suboptimal results. Applying the bi-direction idea on RRT* has been of great interest to scholars recently. Akgun and Stilman (2011) proposed the bi-directional RRT* (B-RRT*) which showed empirical results with faster convergence toward the solution when compared to other bi-directional planners based on the path costs and the total number of iterations taken to find the solution. B-RRT* uses an admissible heuristic to perform path refinement and reject nodes that are not promising to give a solution which helps to avoid heavy exploration of the space. The Connect heuristic used in RRT-Connect can be computationally expensive when applied on RRT* since the planner has to consider $\log_n$ neighbours for rewiring at each iteration which could give suboptimal results (Jordan and Perez, 2013). Jordan and Perez (2013) introduced yet another bi-directional RRT* namely Optimal B-RRT* which uses several heuristic methods to improve the convergence rate. RRT-Connect stops immediately when the first solution is found, and it does not try to improve that path even if the time given for the planner allows. To overcome this, Klemm et al. (2015) presented an asymptotically optimal bi-directional RRT, called RRT*-Connect. After the first solution is found RRT*-Connect continues to search the workspace for a better solution. RRT*-Connect proved to find the first solution faster than RRT* which increases its performance in complex environments (Klemm et al., 2015). RRT*-Connect

in (Klemm et al., 2015) was evaluated against RRT* based on the number of iterations taken by each of the planners. Even though RRT*-Connect finds the solution much sooner, the convergence rate does not change from that of RRT*. Nevertheless, in complex environments, the basic RRT is still effective in solving path planning problems although it gives suboptimal solutions and this will be highlighted in Chapter 3. The weakness of the basic RRT is that it does not take path cost into consideration.

One way to improve the performance of the RRT is by adding goal bias. Given some probability p_{goal} when generating random nodes, goal-biased RRT uses the goal node instead of a random node, this results in a faster convergence rate (Urmson and Simmons, 2003; Zucker, Kuffner, and Branicky, 2007). This then introduces a new parameter known as the goal biasing factor. Introducing a small biasing factor such as 0.05 can make a significant improvement in the convergence rate of RRT (Urmson and Simmons, 2003; LaValle and Kuffner, 2001). However, if the goal biasing factor is too small the planner will behave like the basic RRT algorithm and take a long time to find the solution. Meanwhile, if the goal biasing factor is too big then RRT will behave like a randomized potential field planner which is trapped in a local minimum (LaValle and Kuffner, 2001). Without goal biasing the tree may grow towards the goal but sometimes fail to find it (Urmson and Simmons, 2003). Kuffner and LaValle (2000) suggested that a biasing factor between $5 - 10\%$ might be the right choice. However, the choice of a biasing factor depends on the nature of the robot's workspace.

Yershova et al. (2005) introduced the Dynamic-Domain RRT method that reduces sampling domain by growing Voronoi-based regions in a ball of a specified radius. The random sampling for new random nodes is done within the radius of q_{near}, otherwise, tree expansion will not occur. In this research project, the basic RRT which is easier to implement is investigated on the mesh surface. Finding the shortest paths on mesh surfaces can be computationally expensive. A local region-based RRT is proposed to improve the performance of RRT on a mesh. This method uses local regions formed around each node as a sampling space for RRT restricting RRT to only explore within this region. A bi-directional RRT which is known to find a path faster than RRT is also investigated (Kuffner and LaValle, 2000).

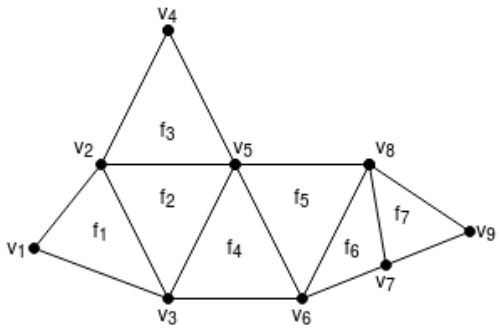


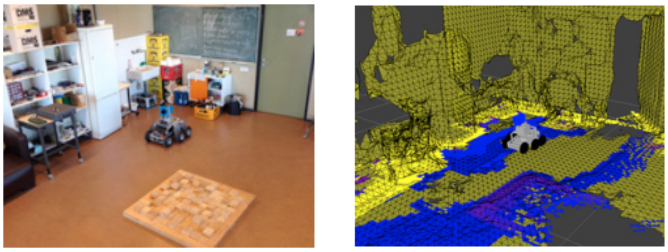
FIGURE 2.3: A planar representation of a triangle mesh.

2.4 3D surface meshes for path planning

Triangle meshes enable efficient computation for several navigation procedures Kallmann (2010). Meshes are popular in gaming, they are data structures designed to aid pathfinding in complex and cluttered environments. A mesh can represent 2D or 3D environments, presented as convex polygons usually triangles. A 2D mesh with polygonal obstacles is the one that is mostly used in computer games. Breitenmoser and Siegwart (2012) presented a mesh construction approach for a climbing robot path planning. The work of (Breitenmoser and Siegwart, 2012) uses only the triangle strips to implement a graph-based planner. A mesh can be defined as a tuple $\mathcal{M} = (\mathcal{F}, G)$ (Toll et al., 2016) which contains an embedded undirected graph $G = (V, E)$ which is ideal for path planning and a collection of all geometric regions $\mathcal{F} = \{f_0, f_1, ...\}$ in $\mathbb{R}^3$, where each region does not intersect itself. Elements of a triangle mesh are faces, vertices, edges, and half-edges. Every vertex has its own position, and an edge is a connection between two vertices. A face is a closed set of three edges. A surface mesh is a set of faces as shown in Figure 2.3 which is a planar unfolded mesh with a list of adjacent faces $\mathcal{F} = \{f_1, ..., f_7\}$ such that f_i is edge adjacent to f_{i+1} and vertex list $V = \{v_,..., v_9\}$ (Mitchell, Mount, and Papadimitriou, 1987).

2.4.1 Mesh Construction

3D surface meshes have become useful in the representation of large and cluttered environments that are not easy to present using simple grids (Kallmann, 2010). A 3D mesh surface enables navigating through a non-spherical topology such as a tunnel. Moreover, 3D spaces can be directly represented on a mesh and that makes it easy to identify obstacles and the traversable space. Figure 2.4b (Pütz et al., 2016) shows an example of a triangle mesh.



(A) Physical robot workspace (B) Mesh representation of the workspace

FIGURE 2.4: Original workspace of a robot and its mesh representation (adapted from Pütz et al. 2016).

3D surface meshes can be constructed from 3D point clouds. The process used in mesh construction from the 3D point cloud is depicted in Figure 2.5. These points can be structured or unstructured. Unstructured data contains only the coordinates of the point while structured data contains additional information such as face normals (Khatamian and Arabnia, 2016). There are many surface

construction methods. Most surface construction methods were developed for computer graphics related problems with different goals in mind. This makes it a bit difficult to know which one will work properly for a given path planning problem based on the input data. The input data can produce different results depending on the method used for surface construction. Point cloud processing may be necessary to reduce noise and filter unwanted data before surface reconstruction, estimate normals, and segment objects in a scene. Further processing of the mesh is done after reconstruction to prepare the mesh for path planning. The following sections discuss some well-known surface construction methods. These methods can be categorized into two types, explicit and implicit. In the former an explicit surface representation depends on the point cloud data given as input data and in the latter the implicit surface representation is defined by a function that best fits the input data (Khatamian and Arabnia, 2016).

2.4.2 Explicit Mesh Construction Methods

Explicit methods precisely present the location of the surface as either parametric or triangulated surfaces depending on the representation of the constructed surface (Zhao, Osher, and Fedkiw, 2001). The earliest techniques used for surface construction were parametric methods such as B-Spline and non-uniform Rational B-Spline, known as NURBS. Data must be parameterized in such a way that the generated surface is a graph in the parameter space for these methods to work (Zhao, Osher, and Fedkiw, 2001). For NURBS the input data can be non-uniform, but the reconstructed surface will be smooth. The disadvantage of using parametric methods for surface construction is that they cannot deal with noisy data. Delaunay triangulation and Voronoi diagrams are topical issues in computation geometry and are often used to construct triangulated explicit methods (Fortune, 1992). Each Voronoi diagram has a Delaunay triangle that is unique and has no other faces in its interior. The assumption for these methods is that the input data is noise-free and dense since the resulting surface mesh depends on the quality of the input (Zhao, Osher, and Fedkiw, 2001; Lim and Haron, 2014).

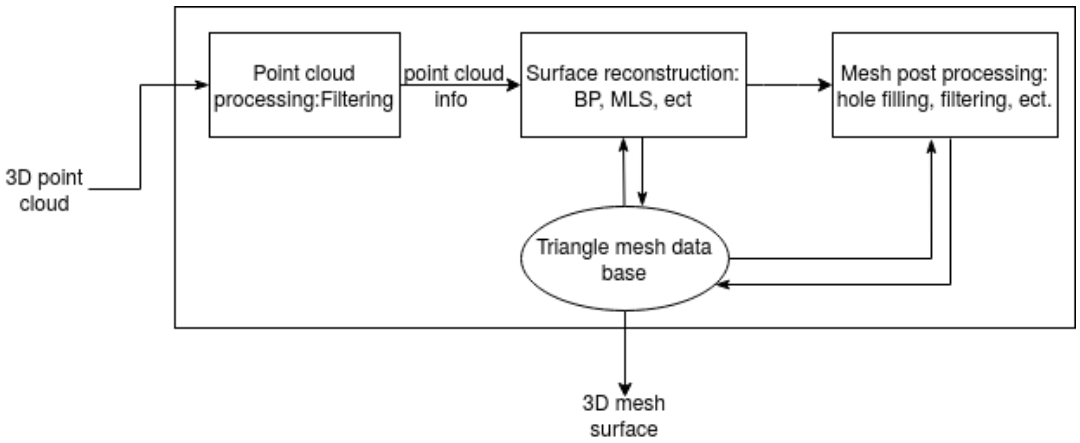


FIGURE 2.5: An overview of mesh construction from 3D point clouds.

There are many surface reconstruction methods based on Delaunay triangulation and Voronoi diagrams such as Ball Pivoting (BP) (Bernardini et al., 1999), Alpha shapes (Edelsbrunner, Kirkpatrick, and Seidel, 1983), and the Crust algorithm (Amenta, Bern, and Kamvysselis, 1998). The majority of these algorithms create a mesh by connecting the points that are closest to each other. Many surface reconstruction methods are based on Delaunay triangulation and Voronoi diagrams because this approach can maximize minimum angles and avoid tiny triangles (do Rego, Araujo, and de Lima Neto, 2007). Although this method can be adapted easily and can deal with general data, it only produces piecewise linear surfaces and struggles to handle noisy and non-uniform data (Zhao, Osher, and Fedkiw, 2001). Moreover, it is quite difficult to use explicit methods when dealing with large deformations and topological changes.

2.4.3 Implicit Mesh Construction Methods

Implicit methods sometimes referred to as volumetric or function fitting methods (Lim and Haron, 2014; Zhao, Osher, and Fedkiw, 2001) need to be preceded by a triangulation method because they produce manifold surfaces defined by complex functions (Khatamian and Arabnia, 2016). The best-fitting isocontour of a scalar function can be obtained using numerical methods (Khatamian and Arabnia, 2016). Signed-distance function, marching cubes, radial basis function, moving least square, and indicator function are examples of implicit surface reconstruction techniques, which can be classified as global or local fitting approaches. The least-squares method, which approximates a solution near to the real solution of the problem, is one of the most commonly used fitting methods (Lim and Haron, 2014). Alexa et al. (2003), explore an approximation method based on moving least squares for implicit surface representation on local differential geometry. For large data sets, this method showed to be computationally expensive. Input data is usually partitioned to reduce the computational cost (Khatamian and Arabnia, 2016). On the other hand, Poisson surface reconstruction is a combination of both local and global implicit functions and considers all data points at once, giving a global solution (Kazhdan, Bolitho, and Hoppe, 2006). However, a small change in a data set may produce a totally different surface structure making it difficult to deal with incremental changes (Zhao, Osher, and Fedkiw, 2001). In this method, the surface representation is estimated as a Poisson problem making it resilient to noise. Like the radial basis function, Poisson reconstruction creates smooth surfaces and it does data filling as well as hole filling. The advantage of using implicit methods is that they are flexible with any topology. They provide better and smooth surfaces and are memory efficient.

In this research project, both explicit and implicit mesh contraction methods are used. The Ball Pivoting method which is exact is used for the creation of the synthetic mesh because the data is noise-free. The Poisson surface construction was used for the creation of the real mesh to consider all data points at once. Since the data was noisy, the Poisson method was able to produce a

smoother mesh surface.

2.4.4 Discrete geodesic paths on mesh surfaces

Path planning on mesh surfaces is not as straightforward as path planning in 2D, it requires the computation of geodesic paths which can be computationally expensive. Discrete geodesic paths determine the locally shortest path between two points, p and q on a triangular mesh M. Finding the shortest paths in a 3D space is a well-known problem in computational geometry and arises naturally in other applications such as robotics and Geographic Information Systems (Sethian, 1999). Computing shortest paths in a 3D mesh surface with obstacles is generally NP-hard (Canny and Reif, 1987). Path planning for a mobile robot on a 3D surface mesh is a typical application of discrete geodesics. A geodesic path can be seen as a straight line connecting the set of points on the unfolded mesh. The problem of getting the geodesic paths or distance can be done using different approaches, depending on the number of source and target points. The most commonly known geodesic problems are finding the shortest path between one source point and one target point, finding the shortest path between one target point and a set of source points, and finding the shortest path between all points in a surface mesh.

Sharir and Schorr (1984) were the first to tackle the problem of finding the exact shortest path for a single convex polyhedron in $O(n^3 \log n)$ time, where n is the number of edges on the surface. This algorithm subdivides the surface mesh into connected regions called peels (Bose et al., 2011). The shortest path problem can then be viewed as a point locating problem, and the path between any two points in a peel can be quickly computed. Mount (1985) later improved the time and space complexity of the Sharir and Schorr (1984) algorithm to $O(n^2 \log n)$ and $O(n^2)$, respectively. Mount (1985) proved that the surface mesh peels can be thought of as Voronoi regions of the set of points containing the planar unfolding of the source point (Bose et al., 2011). These methods inherit behaviour like Dijkstra's algorithm and the computational time to find the shortest path between a point in a mesh to a source point becomes $O(k \log n)$ (Bose et al., 2011; Chen and Han, 1990), where k is the number of edges along that path. Mitchell, Mount, and Papadimitriou (1987) used the continuous Dijkstra's method to solve the single source point shortest path (SSSP) problem in $O(n^2 \log n)$. The algorithm of Mitchell, Mount, and Papadimitriou (1987) proved to be technical and practical as well. Surazhsky et al. (2005) implemented this algorithm and showed that the running time is much lower for objects having tens of thousands or hundreds of thousands of triangles. The average running time of $O(n \log n)$ was obtained after modifying the algorithm of Mitchell, Mount, and Papadimitriou (1987) for solving the SSSP problem (Surazhsky et al., 2005).

Chen and Han (1990) introduced an $O(n^2)$ algorithm that does not require continuous Dijkstra's approach. This algorithm uses a tree data structure called a sequence tree resulting from the computation of Voronoi diagram for all faces at once rather than one for each face as it is done by Sharir and Schorr (1984) method. Kaneva and O'Rourke (2000) used synthetic data to implement Chen and Han (1990) method, which confirmed the quadratic time complexity and linear space complexity in practice. However, Kaneva and O'Rourke (2000) discovered that the space complexity of this approach was a bottleneck because the algorithm of Chen and Han (1990) assumes that the unfolding of faces passed by a path has constant complexity. An improved Chen and Han (1990) algorithm is presented in (Xin and Wang, 2009). Xin and Wang (2009) proved that their method outperforms the original algorithm (Chen and Han, 1990) in both time and space and can be used to backtrack the path (see (Kiazyk, Loriot, and Verdière, 2020) for examples on the implementation of this method).

2.5 Summary

In this chapter a review and a brief history of existing path planning algorithms and mesh construction methods were discussed. Two categories of algorithms were discussed, graph-based algorithms and sampling-based algorithms. It is noted that the performance of graph-based algorithms deteriorates in cluttered environments. On the other hand, sampling-based algorithms such as PRM and RRT which were developed to handle path planning problems with constraints have made planning in complex 3D environments easier. These algorithms can produce a solution faster in high dimensional spaces with lower computational costs. Since RRT produces suboptimal results, an asymptotically optimal algorithm, namely RRT-Star(RRT) was introduced. However, due to this method's additional processes, its convergence rate is slower than that of RRT. This makes RRT the best algorithm when the optimality of the path is not considered. Surface meshes that are popular in computer gaming enable efficient path planning in cluttered 3D environments. On a mesh surface, 3D space can be directly represented which makes it easy to distinguish between obstacles and traversable space. These meshes are explicitly or implicitly generated from 3D point clouds. Many surface reconstruction methods were mainly developed for computer graphics problems with different goals in mind. This makes it difficult to know which method works for a particular path planning problem. Since ground robots are constrained into the surface, path planning on a mesh requires the computation of geodesic paths which is NP-hard.

Chapter 3

Planning on a Mesh Surface

Path planning on a mesh surface is more challenging than its 2D counterpart. All vertices and edges of the search tree must lie on the surface of the mesh. Finding paths between points on a surface requires the computation of geodesics. It is quite difficult for the basic RRT to find a solution on a mesh surface in a reasonable time. This chapter details the implementation of RRT on a mesh which is based on a powerful Computational Geometry Algorithms Library (CGAL) (Fabri and Pion, 2009). From CGAL only the Triangulated Surface Mesh Shortest Paths package (Kiazyk, Loriot, and Verdière, 2020) is used for the computation of exact geodesic paths on a surface mesh. To improve the speed of RRT on a mesh, section 3.3 discusses the proposed method which uses local regions to guide the tree growth towards the goal in less time than the basic RRT on a mesh.

3.1 Goal Biasing

RRT quickly explores the vast area of the workspace by uniformly sampling points based on Voronoi biasing which guides the tree to grow towards unexplored regions of the workspace. Thus, the area of a node's Voronoi region is proportional to the likelihood that it will be chosen for expansion (Lindemann and LaValle, 2004). This Voronoi biasing is performed at every sampling and nearest neighbour search step and this can be time consuming for large cluttered environments (Lindemann and LaValle, 2004; Yershova et al., 2005). To speed up the performance of RRT, the growth of the search tree can be pulled towards the goal configuration using the goal biasing procedure (LaValle and Kuffner, 2001; Zucker, Kuffner, and Branicky, 2007), which replaces q_{rand} by q_{goal} with a probability p_{goal} also known as biasing factor, to draw samples towards the goal. The choice of the biasing factor depends on the nature of the environment and the maneuverability of the robot. The introduction of goal biasing can increase the performance of RRT; however, a large biasing factor could lead to the local minima pitfall (LaValle, 1998) and that would negatively impact the success rate of RRT.

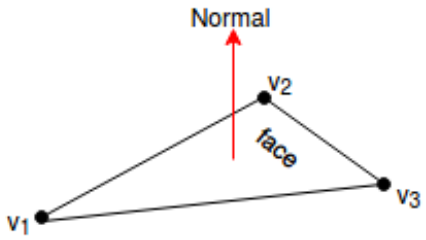


FIGURE 3.1: A triangle showing the normal vector direction.

3.2 RRT on a mesh surface (RRT-M)

Implementing RRT on a mesh surface is more challenging than in 2D since all tree vertices and edges must be on the surface, which requires the computation of geodesics. For this reason the Triangulated Surface Mesh Shortest Paths package from CGAL which is an implementation of the Chen and Han (1990) algorithm is used. This package computes exact geodesic paths ensuring that all tree edges and points are on the surface of the mesh. As mentioned before that RRT quickly samples the entire robot's workspace. In the case of RRT-M, which is an implementation of the basic RRT, shown in Algorithm 2, on a mesh, the random point q_{rand} is selected anywhere on a mesh. Unlike in 2D environments, the search for the nearest neighbour node q_{near} for RRT-M is done through the computation of shortest geodesic paths between q_{rand} and all tree nodes. When q_{rand} is found to be collision-free and satisfies the step size ϵ condition, it is added to the tree. Otherwise, a new node q_{new} is derived using the Steer function which determines the path between q_{near} and q_{new} to be the segment of the path between q_{near} and q_{rand} that is one step size ϵ or less from q_{near} and obstacle-free.

Obstacles are determined by their normal vector orientation. Any triangle face with a normal vector orientation with angle, $\measuredangle$, from the vertical that is above a threshold is considered as an obstacle, and, therefore, the robot cannot pass through that face. The surface normal for each triangle is calculated as a cross-product vector of two adjacent edges of the triangle. As an example, consider the triangle in Figure 3.1, with the red arrow pointing in the direction of the normal. For the three vertices v_1, v_2, v_3 the surface normal N is then given by $N = (v_2 - v_1) \times (v_3 - v_1)$ and can be calculated as in equation 3.2, with the angle to the normal being the cosine of the angle given by equation 3.3. The computation of geodesic paths on a mesh is computationally expensive and for the RRT-M method, geodesic paths are computed for each tree edge which results to high computational time.

Let:

$$\begin{aligned} V &= (v_2 - v_1) \\ W &= (v_3 - v_1) \end{aligned} \tag{3.1}$$

The coordinates of the surface normal N are then:

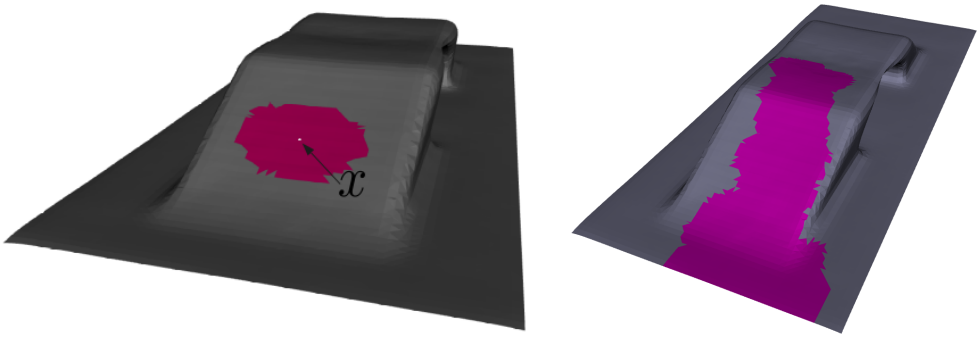
$$\begin{aligned} n_x &= (V_y * W_z) - (V_z * W_y) \\ n_y &= (V_z * W_x) - (V_x * W_z) \\ n_z &= (V_x * W_y) - (V_y * W_x) \end{aligned} \tag{3.2}$$

where the angle θ between the surface normal and the vertical norm v being the dot product of these vectors divided by their vector magnitude as follows:

$$\cos\theta = \frac{N \cdot v}{|N| \cdot |v|} \tag{3.3}$$

3.3 RRT on a mesh surface with local regions (RRT-ML)

From the previous section, a way of reducing the computational time of RRT on a mesh is required. In this section, a proposed local region-based RRT method is introduced. In this method, every mesh vertex has a local region of a specific radius r formed around it with the vertex being a centre point of that region as depicted in Figure 3.2a, where x is a vertex (white point) of πr^2 area local region in pink. The idea behind the local regions is to select random points that are within a specified radius r from nodes in the tree. This will reduce the length of geodesic paths to be computed, which then improves the time taken by the RRT-ML algorithm to complete planning on a mesh.



(A) 1m radius local region of a single node x. (B) A union of local regions.

FIGURE 3.2: A 1m radius single local region and the combined set of local regions.

A local region of a point consists of triangles/faces that are within the area less than πr^2, where r is the radius of the local region. As the tree grows, these local regions are combined to form a small subset of the mesh called sub-mesh, which is a union of all local regions. An example of a sub-mesh embedded in

Algorithm 7 *RRT-ML Algorithm*

1: $\mathbf{M} = (\mathcal{R}, \mathbf{T})$
2: $\mathbf{T} \leftarrow$ **InitializeTree**(q_{init})
3: SubMesh $\leftarrow$ **LocRegion(**q_{init}, r**)**
4: **for** $i = 0$ to $i = N$ **do**
5: $q_{rand} \leftarrow$ **Sample**(SubMesh)
6: XrandReg $\leftarrow$ **LocRegion**$(q_{rand}, rand\_r)$
7: $q_{near} \leftarrow$ **Nearest**(XrandReg, q_{rand})
8: $q_{new} \leftarrow$ **Steer**$(\mathbf{T}, q_{near})$
9: **if ObstacleFree**(q_{new}) **then**
10: $\mathbf{T} \leftarrow$ **AddNode(**q_{new}**,T)**
11: SubMesh $\leftarrow$ **AddFaces**(LocRegion(q_new, r))
12: **return T**

the original mesh is depicted in Figure 3.2b, where the original mesh is the grey area and the sub-mesh is the pink area. The root and the goal configurations are also within the sub-mesh. Algorithm 7 shows the steps that were taken to implement RRT-ML. The random point q_{rand} is selected from the union of all local regions which is called SubMesh in line 3 of algorithm 7, so for the first random point selection, sampling only takes place within the local region of the root node. The sub-mesh is continually growing as the tree grows. When searching for the nearest neighbour node, only tree nodes that are found within the random point's local region are considered, which reduces the number of geodesic paths to be computed. Where the random point's local region is the area of radius $rand\_r$ formed around the random point, this is given by line 5 in algorithm 7. The steering function used in RRT-ML works in the same manner as in RRT-M. It determines an obstacle-free path segment between q_{near} and q_{rand} which is one ϵ or less from q_{near}.

3.4 RRT-Connect on a mesh surface (RRT-CM)

In an n-node search tree, finding the shortest path can take at most $n(n-1)/2$ comparisons (Dantzig, 2016). However, growing two trees as two separate problems, one from the initial point and the other from the goal, can reduce the number of comparisons (Dantzig, 2016). The idea of bi-directional search is well developed in classical AI. RRT-Connect (Kuffner and LaValle, 2000) described in chapter 2 have shown huge success in solving path planning problems for autonomous robots. In this algorithm, each tree is grown separately at each iteration until the connection point, which is a common point where these trees connect is found. Points q_1 and q_2 are considered equal if the distance between them is less than a small threshold.

The implementation of RRT-Connect on a mesh also requires the computation of geodesics to ensure that all paths lie on the mesh surface. There are many variants of RRT-Connect as mentioned in (Kuffner and LaValle, 2000). One way that showed improvements over the method proposed by LaValle and Kuffner (1999), is by replacing the CONNECT function in algorithm 5 with EXTEND

Algorithm 8 REACHED

```
1: function REACHED((M, t1, t2))
2:     q_rand ← GetRand(M)
3:     {res1, q_new1} ← EXTEND(t1, q_rand)
4:     if res1 != Trapped then
5:         {res2, q_new2} ← EXTEND(t1, q_rand)
6:         if EXTEND(t2, q_new1) = Reached then
7:             return True
8:     return False
```

function, given in algorithm 3 from section 2.2.2, which results into two simple RRTs. To implement this variant, algorithm 5 had to be modified in order to work well on a mesh surface. The modified algorithm is given by algorithm 9 which relies on the function REACHED given in algorithm 8. Function REACHED takes in a mesh $\mathcal{M}$ as an input and determines if the connection point is reached by extending or growing one tree at a time while the other tree is not trapped. From the EXTEND function given by algorithm 3 in section 2.2.2, the planner is Trapped only if the connection point is not found or there is no q_{new} to be added in either one of the trees. This could happen if the maximum number of iteration is reached without finding a solution or an obstacle is encountered. When an obstacle is encountered, the process of finding the connection point is repeated until the connection point is found or until one of the trees is extended by adding a new point into the tree, this is presented by Advanced in the function EXTEND, as defined in algorithm 3. The algorithm stops when the first connection point is found and the path from this point to the initial point is traced back and the path from the connection point to the goal point is also traced back. These two path segments are later joined to obtain a single path from the initial point to the goal point. Since this approach grows two trees alternately, and in each iteration each tree attempt to grow towards the other tree in search of the common node, the need for goal biasing is removed.

Algorithm 9 ***RRT-CM***

```
1: t1 ← InitializeTree(q_init)
2: t2 ← InitializeTree(q_goal)
3: M = (R, T)
4: for i = 0 to i = N do
5:     Connected ← REACHED(M, t1, t2)
6:     if Connected = True then
7:         return Path
8:     else
9:         Connected ← REACHED(M, t2, t1)
10: return Failure
```

Algorithm 10 *Function NewConfig for the RRT-CML algorithm*

```
1: function NewConfig(T, q_near)
2:     {q_new, ShortPath} ← Steer(T, q_near)
3:     Add-Node ← False
4:     if ShortPath ! = 0 then
5:         NewConfig ← q_new
6:         T ← AddNode(T, NewConfig)
7:         Add-Node = True
8:     return {Add-Node, NewConfig}
```

3.5 RRT-Connect on a mesh surface with local regions (RRT-CML)

Although RRT-Connect on a mesh could speed up the process of finding the solution. The underlying classical RRT in RRT-Connect still has the tendency of exploring a vast area in a robot's workspace. Thus, resulting in many geodesics paths to be computed which could possibly be long geodesics paths. The performance of RRT-Connect on a mesh can be improved through the introduction of local regions to restrict the sampling space into a small subset of the mesh, which reduces the exploration space for the planner into a growing sub-mesh instead of the entire mesh surface. In this section RRT-Connect with local regions (RRT-CML) is discussed. Since this approach grows two trees alternately, one rooted at the initial configuration and the other at the goal configuration. Both initial and goal configurations are initialized with local regions of radius, r. As each tree grows towards the other, two subsets of local regions are created and later intersect. As soon as these local regions intersect, the connection point, which belongs to both trees will be found and the path from the initial configuration to the goal configuration will be obtained. Depending on which tree is being expanded in each iteration, these subsets are interchanged when sampling for a random point. The nearest neighbour search is done through the computation of shortest geodesic paths between tree nodes and the random point. Again, this is done in each tree expansion, everything that was done for a single tree in RRT-ML in section 3.3 is done twice in RRT-CML.

Algorithm 11 *Function EXTEND for the RRT-CML algorithm*

```
1: function EXTEND(T, q, SubMesh)
2:     {Neighbour, q_near} ← Nearest(T, q)
3:     if Neighbour = True then
4:         {NewNode, q_new} ←  NewConfig(T, q_near)
5:         if NewNode = True then
6:             return {Reached, q_new}
7:         else
8:             return Advanced
9:     return Trapped
```

Algorithm 12 *Function REACHED for the RRT-CML algorithm*

```
1: function REACHED(t_1, t_2, list_1, list_2)
2:     q_rand ← GetRand(list_1)
3:     {res1, q_new1} ← EXTEND(t_1, q_rand, list_1)
4:     if res_1! = Trapped then
5:         {res2, q_new2} ← EXTEND(t_2, q_new1, list_2)
6:         if res2 = Reached then
7:             return {True, q_new2}
8:     return False
```

The EXTEND and REACHED procedures are used for the RRT-CML are given by the pseudocodes in algorithm 11 and algorithm 12, respectively. The function NewConfig (10) used in this method is slightly different from the one used in the basic RRT-Connect. RRT-CML method works almost the same as the RRT-CM method except that it uses the local region's approach to minimize the number and the length of geodesics to be computed. Pseudocode for this method is given in algorithm 13. This is the main algorithm taking in a mesh $\mathcal{M} = (\mathcal{F}, T)$ as input, with t_1 and t_2 being proper subsets of T. Since growing two trees alternately, two local regions are also grown alternately. Two lists are created for this purpose. $List_1$ in line 3 of algorithm 13 initially contains nodes that are within the local region of the starting configuration q_{init} and it grows simultaneously with the first tree (t_1). $List_2$ in line 4 of algorithm 13 initially contains nodes that are within the local region of the goal configuration q_{goal}. $List_2$ also grows simultaneously with the second tree (t_2). By combining $list_1$ and $list_2$ a SubMesh is obtained, since $list_1$ is a set difference of SubMesh and $list_2$, and $list_2$ is a set difference of SubMesh and $list_1$. In line 7, REACHED function checks if the connection point is found or not found while searching in $list_1$'s local region area. If the connection point is found (line 8), the local region of the point q_{mid} is added into $list1$ (line9) then the path is extracted in line 10 and returned in line 11. The planner will stop at this point. Otherwise,

Algorithm 13 *RRT-CML Algorithm*

```
1: t_1 ← InitializeTree(q_init)
2: t_2 ← InitializeTree(q_goal)
3: list_1 ← LocRegion(q_init, r)
4: list_2 ← LocRegion(q_goal, r)
5: Path ← {}
6: for i = 1 to i = N do
7:     {Connected, q_mid} ← REACHED(t_1, t_2, list_1)
8:     if Connected = True then
9:         list_1 ← AddFaces(LocRegion(q_mid, r))
10:        Path = ExtractPath(list_1, list_2)
11:        return Path
12:    else
13:        {Connected, q_mid} ← REACHED(t_2, t_1, list_2)
14:        if Connected = True then
15:            list_2 ← AddFaces(LocRegion(q_mid, r))
16:            Path = ExtractPath(list_2, list_1)
17:            return Path
18: return Path
```

if line 9, returned false the process is repeated from line 12 with t_1 replaced by t_2 and vice-versa and $list_1$ replaced with $list_2$. REACHED function is called again in line 13 to check if the connection point is found now searching in $list_2$'s local region area. If the connection point is found (line 14), the local region of the point, q_{mid} is added into $list_2$ (line 15) then the path is extracted in line 15 and returned (line 16) and the planner stops. The process is repeated and if the connection point is not found after N iterations the empty path is returned in line 18.

3.6 Summary

The goal biasing factor and its impact on improving RRT performance was introduced in this chapter. RRT on a mesh together with RRT-Connect on the mesh are described. The usage of normal vectors for collision avoidance was also discussed. RRT on a mesh with local regions was introduced as a new planning method that limits the exploration space of the RRT into a small subset of the mesh. A bi-directional version of this method which grows two trees and two local region areas alternately was introduced in this chapter. The implementation of these algorithms is presented in the next chapter.

Chapter 4

Simulation Results

In this Chapter, the effectiveness and the behaviour of the four methods discussed in the previous chapter are compared. As discussed in section 2.3, the convergence rate is the most widely used measure when evaluating path planners. In this study, planning algorithms are compared by the run time required to find a solution and by the number of iterations an algorithm takes to find a solution. Comparison based on the quality of the solution is not as suitable for non-optimal planner such as RRT and RRT-Connect.

To evaluate the performance of RRT-M, RRT-ML, RRT-CM, and RRT-CML the set of starting and goal positions is used. This is motivated by the application of path planning in mobile robots where multiple paths must be computed as part of navigation. The path for each different starting and goal position pair could have the same distance but the time taken for the planner to find the path may differ since the planner might encounter obstacles in one route that are not necessarily on the other route.

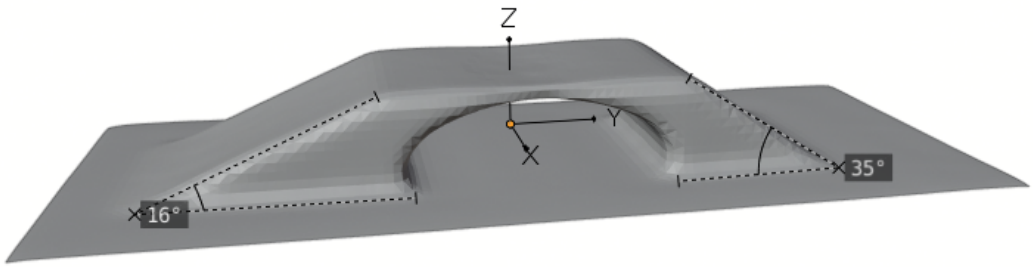


FIGURE 4.1: A synthetic mesh showing the direction of primary axes from the origin and the angles of the slopes.

4.1 Synthetic Mesh Setup

To demonstrate how RRT-M, RRT-ML, RRT-CM, and RRT-CML work on a mesh surface, a synthetic mesh (Figure 4.1) which consists of a tunnel and a bridge, representing an environment with non-spherical topology, was created. The mesh has two slopes of different angles, one side is 16° and the other side is 35°. In Figure 4.1 the XYZ-axis are shown with the orange point being the origin. This synthetic mesh has 18362 triangles and 9330 vertices. The mesh was initially created in Blender v2.82 (Community, 2018) and it was processed using MeshLab (Cignoni et al., 2008). The processing of the mesh includes uniform re-sampling of the points on the mesh, re-meshing, and hole filling resulting in a smoother surface mesh.

On this surface mesh, four different sets of starting and goal positions, see Table 4.1, were selected to assess the run time, averaged per iteration for each method. The step size ϵ is $1m$ and the maximum number of iterations is 1500 for all simulations. For collision avoidance, the normal vector orientation of the face is used to determine if the triangle face is traversable or not. Any face having a normal vector orientation with an angle $\measuredangle$ of 30° or above is considered an obstacle, therefore the planner cannot pass through that face. For the single tree planners, RRT, and RRT-ML the goal biasing factors 0.2 and 0.5 are used, as the probability of using the goal in place of x_{rand}. As mentioned in section 3.1 there is no specific way of determining the best biasing factor. Although 5-10% are suggested as biasing values by LaValle and Kuffner (2001), 0.2 and 0.5 were selected to prove that choosing a big biasing factor such as 0.5 does not necessarily mean improvement in RRT performance. Whereas, choosing a relatively small value such as 0.2 can improve the planner's performance. For new point selection in the bi-directional methods, RRT-ML and RRT-CML, which do not use goal biasing, the radius (r) of $1m$ is used for local regions of all tree nodes and for each random point's local region, the radius ($rand\_r$) of $2m$ is used.

TABLE 4.1: Sets of starting and goal points used for simulations

Case	Starting point	Goal point
1	$-3.76, -5.17, -0.64$	$-0.43, -4.29, 0.13$
2	$-3.76, -5.17, -0.64$	$0.59, 1.15, -0.62$
3	$1.39, -6.99, -0.62$	$-0.73 - 0.75, 1.02$
4	$-0.22, 1, -0.62$	$-0.73 - 0.75, 1.02$

All simulations were implemented in C++ on a Dell Latitude E6530 computer with Intel^R Core^{TM} i7-3720QM CPU @ 2.60GHz processor and 8 GB memory.

TABLE 4.2: Effect of goal biasing RRT on a mesh surface.

Bias	Avg. Iterations	Tree size	Runtime/iter.(s)	Success rate (%)
0	1500	1235	0.18	0
0.2	145	55	0.35	100
0.5	467	104	0.3	$\sim$ 100

4.2 Goal bias in RRT on a mesh surface

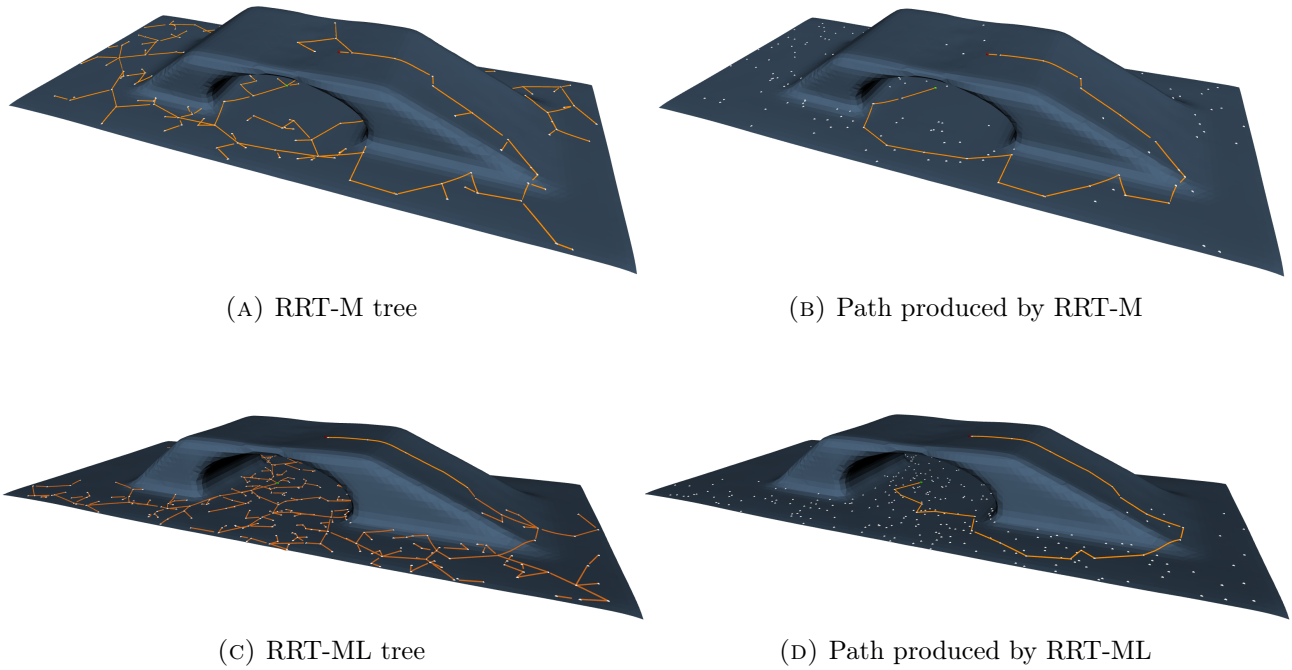
The basic RRT can take too long to find a solution on a mesh surface with obstacles. Introducing the goal bias improves the performance of RRT, Table 4.2 shows the performance of RRT on a mesh with and without goal bias. This is done for 1500 iterations on a synthetic mesh with 18362 triangles. RRT-M was first tested on this mesh with zero biasing factor then with 0.2 biasing factor and with 0.5 biasing factor. Although LaValle and Kuffner (2001) suggested that biasing values should be between 5-10%, after some trial and error test on different biasing factors 0.2 and 0.5 biasing values were selected to prove the concept of goal biasing and produces fair results. In Table 4.2 tree size is the total number of nodes in the tree after the path has been found. This number is averaged over five trials. The runtime per iteration is the time taken by the planner to complete each iteration also averaged over five trials and the success rate is the percentage of trials where the planner was able to find the goal in $\leq$ 1500 iterations.

TABLE 4.3: RRT simulation results for different starting and goal points

	Bias	**Case #**	**Avg. Iterations**	**Avg. time(s)**	**Avg. time/ iter.(s)**	**Avg. Path length(m)**
RRT-M	0.2	1	226	84.03	0.37	7.33
		2	29	16.23	0.56	9.36
		3	56	34.39	0.62	9.3
		4	529	186.46	0.35	19.99
	0.5	1	346	135.98	0.39	6.07
		2	27	17.72	0.66	8.72
		3	9	6.93	0.77	7.77
		4	776	267.22	0.34	18.79
RRT-ML	0.2	1	40	5.87	0.15	5.71
		2	61	8.61	0.14	9.44
		3	27	5.1	0.19	8.16
		4	566	66.53	0.12	18.47
	0.5	1	32	10.89	0.34	5.65
		2	23	6.56	0.29	8.48
		3	12	5.75	0.48	7.39
		4	996	197.31	0.2	19.21

4.3 Simulation Verification of RRT-M and RRT-ML

To evaluate the performance of RRT-M and RRT-ML for each point set in Table 4.1, each planner was tested five times per set. These methods are compared here based on the average time each method took to find the solution. Table 4.3 shows the results of single tree planners, RRT-M, and RRT-ML. In Table 4.3, Bias is the probability of choosing q_{goal} instead of q_{rand} during the RRT tree expansion process. The case number corresponds to the case number in Table 4.1 which indicates the corresponding pair of starting point and goal point. Average iterations are the total number of iterations that each of these planners took to find the path averaged over five trials. To evaluate the performance of these methods on a mesh, the total number of iterations taken by each method to find the path is also considered. In Table 4.3 the average time per iteration is the time taken by RRT-M and RRT-ML to complete one iteration and the average time is the total runtime of the planner. The average path length indicates the average distance of the path produced by each of the planners for each case over five trials. Both RRT-M and RRT-ML were tested on the same mesh described in section 4.1 with the same parameter set up and for a maximum of 1500 iterations.



(A) RRT-M tree

(B) Path produced by RRT-M

(C) RRT-ML tree

(D) Path produced by RRT-ML

FIGURE 4.2: Results for RRT-M and RRT-ML on a mesh for the first case in Table 4.1

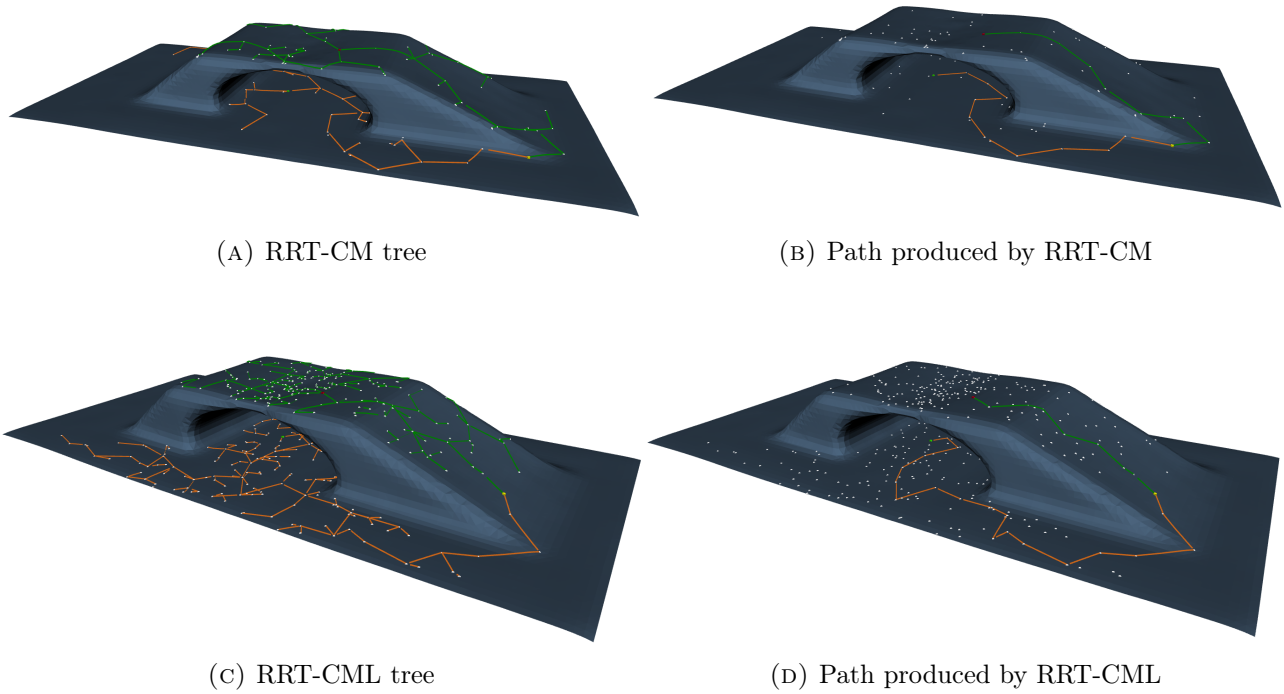
The trees produced by RRT-M and RRT-ML are shown in Figure 4.2a and Figure 4.2c, respectively, in orange. The results shown in Figure 1.4 are for the first case from Table 4.1. For this result, the biasing factor was set to 0.2 for both methods. Figure 4.2b and Figure 4.2d shows the path produced by RRT-M and RRT-ML for the same parameter set up, in orange starting from the green dot which is the starting position to the red dot representing the goal position. The white dots are tree nodes and the orange lines in Figure 4.2a and Figure 4.2c connecting each tree node to another tree node are tree edges.

TABLE 4.4: RRT-CM and RRT-CML simulation results for different starting and goal points

	Case #	**Avg. Iterations**	**Avg. time(s)**	**Avg. time/ iter.(s)**	**Avg. Path length(m)**
RRT-CM	1	48	84.14	1.75	7.78
	2	12	23.28	1.94	10.83
	3	9	21.24	2.36	8.95
	4	170	223.52	1.32	20.05
RRT-CML	1	14	4.67	0.93	7.27
	2	38	2.53	0.07	11.13
	3	45	4.81	0.11	10.58
	4	211	32.85	0.16	19.35

4.4 Simulation Verification of RRT-CM and RRT-CML

In Table 4.4 RRT-CM is compared to RRT-CML since both methods follow the RRT-Connect procedure to grow two trees alternately. These methods are also compared based on the average time taken to find a solution on a mesh surface. The performance of RRT-CM and RRT-CML is evaluated by comparing the average number of iterations each method took to find the path and by the time taken to compute one iteration. The number of iterations is averaged over five trials for all cases as listed in Table 4.1.



(A) RRT-CM tree

(B) Path produced by RRT-CM

(C) RRT-CML tree

(D) Path produced by RRT-CML

FIGURE 4.3: Results for RRT-MC and RRT-CML on a mesh for the first case in Table 4.1

Figure 4.3a and Figure 4.3c show the trees produced by RRT-CM and RRT-CML, respectively. In both, the tree rooted at the start configuration is the orange one and the one rooted at the goal configuration is green. The two trees are connected by the yellow dot which is a common vertex in both trees. The results shown in Figure 4.3 are for the first case from Table 4.1. Figure 4.3b and Figure 4.3d show the path produced by RRT-CM and RRT-CML which is produced by two line segments, one part of this path is obtained from the first tree forming the orange part from the starting point to the connection point and the other part is from the second tree which is the green line from the goal point to the connection point. The white dots in both Figure 4.3b and Figure 4.3d are tree nodes.

4.5 Simulation Verification of RRT-CML on a real mesh

In this section, the effectiveness and the behaviour of the RRT-CML are further evaluated on a real mesh. RRT-CML is not compared to other methods on this mesh since only verifying if the size or the layout of the mesh affects this planner. The mesh was created from real point cloud data obtained from the Commonwealth Scientific and Industrial Research Organization's data access portal (Cox, Borges, and Lowe, 2018). The point cloud data used here is part of the colourized 3D LiDAR SLAM point cloud selection collected by Cox, Borges,

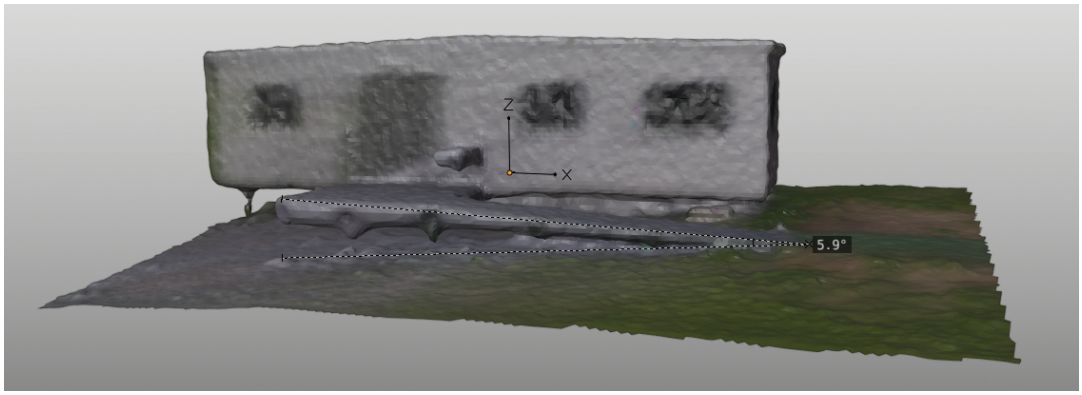


FIGURE 4.4: A mesh from colorized point cloud data.

and Lowe (2018) using the collection method described in (Vechersky et al., 2018). The original dataset has 3521563 vertices but for this simulation, a small section of this data containing a ramp with a slope of approximately 5.9° was used, as in Figure 4.4. The point of origin is represented by the orange point attached to the XYZ axis. The mesh in Figure 4.4 has 67263 vertices and 133822 faces. Meshlab (Cignoni et al., 2008) was used to select the area of interest from the original data. The data points were further down-sampled using MeshLab's Poisson-disk sampling filter and the mesh was created using MeshLab's Screened Poisson Surface Reconstruction filter with reconstruction depth of 11 and the adaptive octree depth being 8.

TABLE 4.5: Sets of starting and goal points used for simulations on a real mesh

Case	**Starting point**	**Goal point**
1	$-6.15, -16.24, -0.81$	$-15.47, -21.05, -0.27$
2	$-7.19, -20.04, -1.4$	$-15.47, -21.05, -0.27$

TABLE 4.6: Simulation results of RRT-CML on a real mesh surface.

Case	**Avg. Iterations**	**Avg. time(s)**	**Avg. time/ iter.(s)**	**Avg. Path length(m)**
1	71	35.44	0.49	15.32
2	201	123.47	0.61	25.48

For these simulations, only two sets of starting and goal points, Table 4.5, were used to evaluate the overall runtime, time per iteration, and path length of RRT-CML on the real mesh averaged over five trials. Table 4.6 shows the results of RRT-CML on the real surface mesh. The tree produced by RRT-CML on this mesh for case 2 of Table 4.6 is shown in Figure 4.5 and the path formed by two segments from each tree is shown in Figure 4.6 for the same case. In this scenario, a robot can pass under the ramp and the container house since these objects are not fixed on the surface. In both these pictures, the starting position is presented by the green dot and the goal position is presented as the red dot in the ramp. The first tree in orange and the second tree in green can be seen connecting at the yellow dot (connection point) which is a common point for both trees.

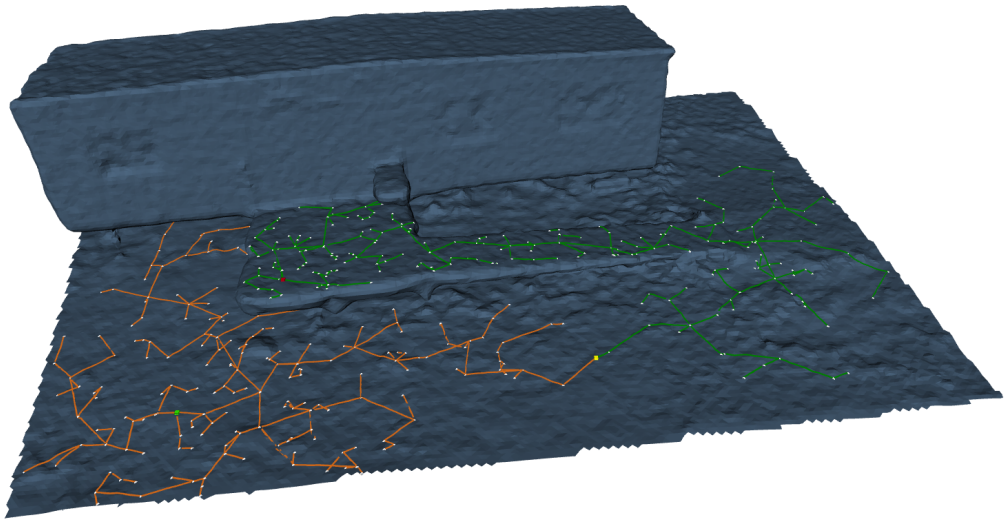


FIGURE 4.5: RRT-CML trees on a real mesh surface.

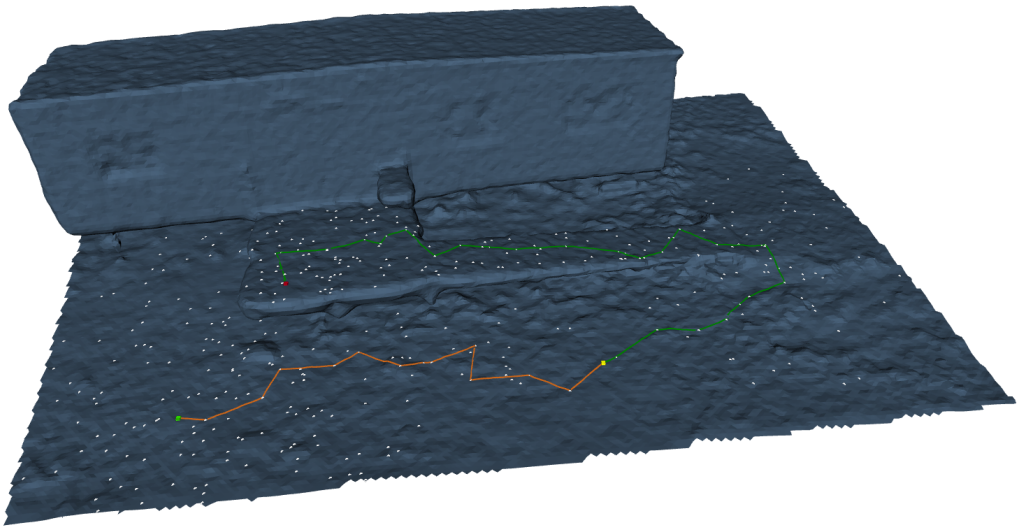


FIGURE 4.6: RRT-CML path on a real mesh surface for case 2 of Table 4.6. The path is from the starting point (green) to the goal point (red). The point where the trees are connecting is shown in yellow.

4.6 Summary

This chapter detailed the simulation results of the methods that were introduced in the previous chapter. A synthetic mesh containing non-spherical typology was created for simulations using Blender and processed in MeshLab for smooth path planning. Single tree methods were tested and evaluated on this mesh and later compared to each other based on the time each method took to find the final path. Bi-directional methods were also evaluated and compared on a synthetic mesh based on the time taken to find the final solution. RRT-CML was also evaluated on a mesh generated from the real point cloud which is much bigger than the synthetic mesh to test if the density of the mesh affects the performance of this method or not. The results of these simulations are discussed in the next chapter.

Chapter 5

Discussion

The performance of RRT on a mesh surface has been evaluated through simulation and it was compared to the bi-directional version of RRT on a mesh in chapter 4. The comparison is based on the time these methods took to find a feasible path from the start configuration to the goal configuration for four sets of start and goal positions. Like other path planners, RRT has its advantages and disadvantages. One of the advantages of using RRT in high dimensional spaces is that it quickly explores the entire space. However, when planning on a mesh surface this can be computational expensive since geodesic paths need to be computed on every iteration. A local region-based RRT planner and its bi-directional version introduced in section 3.3 to improve RRT performance on a mesh, are also compared to the basic RRT on a mesh surface through simulations. The performance of RRT-CML was also evaluated on a real mesh surface in section 4.5. In this chapter, the results presented in chapter 4 are discussed. These four methods, RRT-M, RRT-ML, RRT-CM and RRT-CML, are compared based on the time and the number of iterations each method took to find a solution within a specified maximum number of iterations, and on the real mesh, the performance of RRT-CML is compared to its performance on the synthetic mesh which is smaller than the real mesh in size based on the time it took to find the path. Each of these approaches declares failure to find a path if the maximum number of iterations is reached and the solution is not found.

5.1 Planning With Local Regions

Introducing local regions of radius r, formed around each node ensures that random points are not too far from the tree nodes. The local region area keeps growing as the tree grows, forming what is called a sub-mesh. The exploration space for the planner is then the sub-mesh, thus reducing the time the planner takes to complete planning since it is no longer sampling from the entire mesh. Having a smaller exploration space reduces the length of the geodesic paths to be computed and as a result, the planner gets to the goal position much faster than planning from the entire space. When searching for the nearest node only tree nodes that are within the random point's local region are considered. This results in fewer geodesic paths to be computed.

To avoid the use of goal biasing, a bi-directional RRT method based on RRT-Connect algorithm which grows two trees alternately was combined with the local region procedure. The results from this method were better than that of the basic RRT-Connect on the mesh and are discussed below.

5.2 Simulation Results Comparison

The simulation results in Table 4.2 shows that goal biasing can improve the success rate of RRT. Without goal biasing, RRT builds the tree with 1235 nodes on average and in most cases, the goal point was not reached by the tree. Without goal biasing, RRT can explore the entire mesh space before it can find a goal point which requires more iterations. With goal bias greater than zero, the chances of reaching the goal position increased from 0 to 100 percent, for the goal bias values tested. The average tree size is reduced to lower than 105 nodes when goal biasing is introduced. This proves that introducing the goal bias in RRT improves the chances of getting to the goal in fewer iterations; however, looking at the time to complete one iteration goal biasing does not improve the total runtime of the planner. As the biasing factor was increased, the number of iterations taken to find the goal also increase. Therefore, using a large biasing factor might not be a good choice in some cases.

In almost all cases in table 4.3 the average number of iterations taken by the RRT-M was more than that taken by the RRT-ML. Because RRT-M samples random points anywhere on the mesh, it required more iterations to find a path. For example, in case 1 of table 4.3 when the biasing factor was 0.2, RRT-M found the path in 226 iterations while it only took 40 iterations for the RRT-ML to find the path. The results in table 4.3 show that RRT-ML outperforms RRT-M on the mesh taking at least 0.2 seconds on average to complete one iteration. However, there are a few cases where RRT-ML required slightly more iterations to find the solution, for example in case 4 of table 4.3 when the bias was set to 0.5. In this case, RRT-ML took 996 iterations to find the path while RRT-M took 776 iterations. Regardless of taking many iterations to find the path in case 4, RRT-ML was able to find the path in 197.31 seconds on average, whereas, RRT-M took 267.22 seconds.

Table 4.3 also shows that the use of biasing factor does affect the performance of the planner. However, choosing a perfect biasing factor is not trivial. A specific biasing factor can work perfectly with a specific set of points and produce not-so-good results when the start and goal position is changed. As an example, in table 4.3 for case 4, the bias of 0.5 seems not to be a good choice for this set of points, while it was working well for case 2 and case 3 in reducing the number of iterations taken to find the path. Increasing the biasing factor from 0.2 to 0.5 in the first case from table 4.3, did not reduce the time for the RRT-M neither did it reduce the number of iterations for this method. However, this is not the case for the RRT-ML. The biasing factor of 0.5 worked well in reducing the total number of iterations for the RRT-M resulting in the reduced computational

time. When the biasing factor is high the method uses the goal biasing more often, this can be time consuming, resulting in an increased overall time to find a path. With RRT-ML the tree growth is limited to within the sub-mesh, which is a small subset of the original mesh, thus reducing the chances of considering points on the mesh that are far from the solution path. As a result of this, the total time taken by RRT-ML to find the path is reduced to less than half of what RRT-M takes to find the path for most cases from the five trials that were evaluated.

Growing two trees alternately has had a tremendous result in improving the performance of RRT (Kuffner and LaValle, 2000). In section 4.4, the results of RRT-Connect on a mesh are presented in table 4.4. This table contains the results of RRT-CM and RRT-CML. Since these methods grow two trees alternately at each iteration, the need for goal biasing was removed. For each case in table 4.2, both methods were tested for five trials and the averages for iterations, time, time per iteration, and path length of RRT-CM and RRT-CML over the five trials are presented in table 4.4.

Comparing RRT-M results in table 4.3 with that of RRT-CM in table 4.4. It is quite clear that the bi-directional RRT quickly completes its search. However, it does so at almost the same time as RRT-M, thus the total execution time is not improved. For example, RRT-M took an average of 84.03s and 226 iterations to find the path when the bias was 0.2 and RRT-CM took an average of 84.14s and 48 iterations to find the path. In this case, RRT-M and RRT-CM took almost the same time to find the solution even though the number of iterations was reduced when growing two trees instead of one tree. From Figure 4.3a RRT-CM works the same way as RRT-M when growing the trees except that RRT-CM alternately grows two trees. Nodes are selected from almost everywhere on the mesh. As mentioned before, this is quite computationally expensive since the planner must compute geodesic paths for each node to be added into the tree. RRT-CM is doing twice the computation since it is growing two trees alternately, hence the increased time per iteration from that of RRT-M.

By introducing local regions in RRT-CM to get the RRT-CML method, the average time taken to find the path was reduced for all cases in table 4.4, thus reducing the time per iteration. Comparing all other methods to RRT-CML shows that RRT-CML outperformed all of them in terms of the total time it takes to find the path and the number of iterations it needs to do so. This does not necessarily mean that the distance from the starting position to the goal position is also reduced. However, the sizes of both trees are reduced as in Figure 4.3c which means that the number of geodesics to be computed has been reduced. In section 4.5, RRT-CML was tested on a real mesh which is different from the synthetic one in terms of the total number of faces and vertices it has and the density of points. The results in table 4.5 show that RRT-CML can find a feasible path in a short amount of time on a two-manifold mesh surface. However, it is more likely to have trees crossing each other in both RRT-CM and RRT-CML since each tree is grown as an independent RRT tree.

Depending on how far the starting position is from the goal position all four methods (RRT-M, RRT-ML, RRT-CM, and RRT-CML) take different times for each trial to find the path. Looking at the results presented in Table 4.3 and Table 4.4 it looks like both RRT-ML and RRT-CML are able to find the path in about 5 seconds when the path between the starting and goal position does not have many obstacles. However, this is not the case for the basic RRT which takes a minimum of approximately 7 seconds to find a path.

5.3 Summary

In this chapter, the simulation results presented in the previous chapter were discussed. Single tree planners were compared to each other based on the time taken to find the solution and the bi-directional methods were also compared to each other. Using local regions to limit the search tree into a sub-mesh has shown some effective results with both RRT-ML and RRT-CML doing well on the simulation when compared to their respective basic methods, RRT-M and RRT-CM.

Chapter 6

Conclusions and Future work

6.1 Conclusion

This research project addresses the path planning problem for ground robots in 3D environments. Path planning arises naturally in robotics and its application also plays an important role in other fields such as computational biology and computer animation. A wide range of path planning techniques can be used to solve path planning problems for mobile robots in 2D environments. However, in real world situations ground robots, which are known to be constrained to the surface, face uneven and cluttered environments with different running costs. The problem with 2D representations is that they cannot capture every detail about the environment, especially when the environment contains non-spherical topology, removing some planning possibilities.

Path planning in 3D environments for ground robots is a challenging task since the path must lie on the surface. The overall goal for this project was to develop a planning algorithm using a sampling-based approach on a 3D mesh surface. Mesh surfaces are useful in the representation of large cluttered environments with non-spherical topology, including, for example, bridges and tunnels. When using a mesh as a workspace, the physical workspace of a robot can be directly presented making it easy to differentiate between traversable and non-traversable spaces within the mesh surface. The proposed solution, namely RRT-ML, is based on the RRT algorithm and uses local regions to reduce the computational time of RRT on a mesh surface. RRT-ML has been implemented and tested on a synthetic mesh and it was compared to RRT-M and RRT-CM which are the basic RRT and RRT-Connect on a mesh. The local region approach was also combined with RRT-CM to get a new bi-directional local region-based method, called RRT-CML.

A synthetic mesh was created to evaluate these methods through simulations. A mesh from a real point cloud was also constructed to further assess the performance of RRT-CML on a different mesh environment. Planning on a mesh requires the computation of geodesic paths to ensure that all paths lie on the surface of the mesh. Using CGAL, the computation of geodesic paths on a mesh was done. However, the computation of geodesic paths is computationally expensive. In RRT-M where the nearest neighbour search is done through the computation of geodesic paths. This is done many times since RRT-M grows the tree anywhere on the mesh, leading to high computational time. By introducing

local regions, the exploration space for RRT is reduced to a sub-mesh, thus reducing the length and number of geodesics to be computed. As a result, the runtime of the algorithm is reduced. To further reduce the runtime of the planner on a mesh surface RRT-CML, which grows two trees alternately removing the need for goal biasing, was designed. RRT-CML was implemented on a synthetic and the real mesh and it was compared to other methods (RRT-M, RRT-ML and RRT-CM).

All four methods, RRT-M, RRT-ML, RRT-CM, and RRT-CML, were experimentally verified and compared to each other on the same mesh surface using the same parameter set up. The use of local regions helps to reduce the total runtime in RRT-ML and RRT-CML, requiring fewer iterations to reach the goal. RRT-CML outperformed all other methods by finding a path in fewer iterations and in less time compared to RRT-M, RRT-ML, and RRT-CM on a synthetic mesh. RRT-CML operation was also shown in the context of real 3D surface meshes.

RRT-M took long to find a path, with an average time of 16.23 seconds for the second case in Table 4.3 with a 0.2% biasing factor. The time taken by RRT to find a path on a mesh surface might not be practical when planning in real-time. RRT-M was very slow even when the path between the starting and goal position did not have lots of obstacles. To improve the effectiveness of RRT on a mesh, local regions were introduced. This method was able to improve the speed of RRT on a mesh. Taking half the time that RRT took to find the path, RRT-ML took 8.61 seconds to find the path for the second case in Table 4.3 when the biasing factor was set to 0.2. To further improve the speed of RRT on a mesh, RRT-Connect with local regions was implemented. By growing two trees alternately RRT-CML was able to find a path in 2.53 seconds for the second case of Table 4.4, showing capabilities to work well in real-time planning.

6.2 Future Work

The proposed local region strategy seems to be useful in speeding up the performance of RRT on a mesh surface. However, the results produced by the RRT planner on a mesh might be further improved by combining the local regions procedure with RRT* which is an optimal variant of RRT to obtain asymptotically optimal paths. Perhaps, incorporating local regions in RRT*-Connect would give excellent results on a cluttered mesh surface and reduce the chances of having trees crossing each other.

6.2.1 Local Regions in an Optimal Planner

It is well known that as the number of nodes increases, the quality of the path found using RRT and RRT-Connect algorithms does not improve (LaValle, 2006), and this is the same for RRT-ML and RRT-CML. Meanwhile, RRT*

which is discussed in section 2.2.3 is a well-known asymptotically optimal variant of RRT which has two unique functions, tree rewiring and near neighbour search. The near neighbour in RRT* is somehow a generalized nearest neighbour search process of the RRT which returns nodes that are closest to the tree node based on some cost function, whereas in RRT the cost function is not used. In the near neighbour search of the RRT* a set of nodes that are close to q is returned, whereas in the nearest neighbour search of the RRT only one vertex is returned. The set of nodes close to q is defined as all points that are within a closed ball of radius k, as defined in equation 2.2. The rewiring procedure then builds the tree within this radius to maintain minimum cost between tree nodes.

Introducing local regions in RRT* could aid the near neighbour search in RRT* since all nodes are initialized with local regions of radius r. Finding nodes that are in a ball of radius k that are considered as closest to q can be done using the local region concept. This will allow nodes to be selected for the rewiring process to be within this local region and the cost between the nodes could be further minimized.

Although RRT* produces asymptotically optimal solutions, it takes more time to find its path than the basic RRT (Noreen, Khan, and Habib, 2016b). Planning on a mesh is computationally expensive on its own, therefore, using RRT* on a mesh could take even longer to find a solution even with local regions in place. From the simulations growing two trees helps to speed up the runtime of the planners and removes the need for goal biasing. It can be expected that the RRT*-Connect with local regions will be faster than RRT* alone, based on the results from comparing RRT and RRT-Connect (Kuffner and LaValle, 2000) and it may produce outstanding results while guaranteeing asymptotically optimal paths.

Appendices

Appendix A

RRT-M on a synthetic mesh results

TABLE A.1: Five trials for RRT-M in a synthetic mesh for all cases in Table 4.1

Bias	Case #	Iterations	Time(s)	Time/ iter.(s)	Path length(m)
0.2	1	251	85.197	0.34	7.69
		392	134.681	0.34	7.95
		107	44.61	0.42	5.51
		153	71.61	0.47	8.15
		222	78.62	0.35	7.62
	2	24	12.81	0.53	8.57
		39	22.05	0.57	9.67
		24	12.68	0.53	9.67
		36	20.12	0.56	10.62
		22	13.51	0.61	8.29
	3	36	26.64	0.74	8.67
		14	10.14	0.72	8.36
		10	8.05	0.81	9.03
		131	54.71	0.42	10.38
		87	72.43	0.83	9.21
	4	485	159.4	0.33	20.89
		650	222.63	0.34	18.54
		618	222.15	0.36	17.98
		468	169.22	0.36	21.26
		424	158.876	0.37	21.27
0.5	1	109	49.73	0.46	7.79
		520	205.996	0.39	6.29
		257	110.81	0.43	5.43
		216	97.39	0.45	8.08
		627	215.99	0.35	75.5
	2	15	9.72	0.65	8.99
		53	35.61	0.67	9.93
		14	8.8	0.63	8.18
		24	16.24	0.68	8.02
		29	18.25	0.63	8.48
	3	12	8.85	0.72	8.07
		12	8.65	0.72	8.17
		7	6.06	0.87	7.59
		6	5.32	0.89	7.01
		7	6.02	0.86	8.02
	4	1333	425.21	0.32	20.67
		369	149.05	0.4	17.39
		760	250.696	0.33	16.32
		717	260.551	0.36	22.23
		699	250.61	0.36	17.35

Appendix B

RRT-ML on a synthetic mesh

TABLE B.1: Five trials for RRT-ML in a synthetic mesh for all cases in Table 4.1

Bias	Case #	Iterations	Time(s)	Time/ iter.(s)	Path length(m)
0.2	1	13	2.56	0.19	5.02
		35	7.09	0.2	5.43
		46	6.29	0.14	6.03
		66	9.26	0.14	5.94
		42	4.17	0.99	6.15
	2	127	17.27	0.14	8.94
		56	9.04	0.16	10.66
		68	8.85	0.13	9.67
		34	3.59	0.11	10.62
		20	4.32	0.22	8.29
	3	32	5.15	0.16	8.96
		42	7.53	0.18	9.06
		32	3.59	0.11	7.89
		27	4.39	0.16	7.32
		38	5.06	0.13	7.56
	4	373	47.89	0.13	16.76
		487	52.55	0.11	17.24
		518	58.798	0.11	17.45
		860	103.61	0.12	20.49
		594	69.82	0.12	20.39
0.5	1	20	8.69	0.44	5.58
		66	23.27	0.35	6.46
		6	2.46	0.41	4.44
		55	13.76	0.25	6.57
		13	6.27	0.48	5.18
	2	21	5.97	0.28	7.78
		21	34.26	0.2	8.83
		14	5.63	0.4	7.92
		39	12.48	0.32	9.73
		18	4.45	0.25	8.12
	3	9	4.7	0.52	7.39
		10	5.38	0.54	7.65
		20	7.77	0.39	8.13
		12	5.46	0.46	6.88
		10	5.42	0.54	6.88
	4	955	189.99	0.19	21.81
		1353	256.412	0.19	18.92
		954	194.481	0.2	19.696
		987	197.28	0.199	17.03
		729	148.37	0.2	18.57

Appendix C

RRT-CM on a synthetic mesh results

TABLE C.1: Five trials for RRT-CM in a synthetic mesh for all cases in Table 4.1

Case #	**Iterations**	**Time(s)**	**Time/ iter.(s)**	**Path length(m)**
1	58	90.99	1.57	7.65
	43	86.22	2.01	7.89
	20	48.232	2.41	6.85
	53	89.41	1.69	7.31
	64	105.84	1.65	9.21
2	10	19.58	1.96	9.02
	6	12.34	2.06	9.65
	4	8.8	2.2	12.55
	25	49.85	1.99	12.95
	13	25.82	1.99	9.88
3	28	63.73	2.28	9.67
	3	7.33	2.44	10.43
	3	8.82	2.94	7.96
	7	16.71	2.39	8.99
	4	9.59	2.39	7.69
4	224	271.34	1.21	17.64
	148	213.49	1.44	18.38
	161	199.34	1.24	17.99
	120	163.29	1.36	22.76
	197	270.12	1.37	23.48

Appendix D

RRT-CML on a synthetic mesh results

TABLE D.1: Five trials for RRT-CML in a synthetic mesh for all cases in Table 4.1

Case #	**Iterations**	**Time(s)**	**Time/ iter.(s)**	**Path length(m)**
1	10	0.69	0.07	6.71
	10	0.62	0.06	5.81
	6	0.37	0.06	6.43
	22	1.38	0.06	9.19
	22	1.61	0.07	8.19
2	37	2.29	0.06	11.91
	20	1.11	0.06	9.32
	41	2.74	0.07	11.17
	65	4.81	0.07	12.93
	26	1.72	0.07	10.3
3	53	5.39	0.1	11.99
	24	2.34	0.09	10.45
	62	6.44	0.1	12.59
	24	2.16	0.09	9.52
	64	7.7	0.12	8.36
4	272	47.72	0.18	18.24
	158	19.09	0.12	20.21
	198	29.2	0.15	19.45
	216	34.29	0.16	19.49
	210	33.95	0.16	19.38

Appendix E

Planning on a real mesh

E.1 RRT-CML on a real mesh

TABLE E.1: Five trials for RRT-CML in a real mesh for all cases in Table 4.5

Case #	Iterations	Time(s)	Time/ iter.(s)	Path length(m)
1	52	23.44	0.45	16.71
	85	50.31	0.59	13.71
	66	33.79	0.51	13.38
	72	31.75	0.44	14.48
	82	37.89	0.46	18.34
2	243	132.85	0.55	26.52
	233	132.91	0.57	26.03
	172	116.80	0.68	24.18
	173	110.77	0.64	23.86
	186	124.04	0.67	26.82

E.2 Point cloud data

The data used in this dissertation was obtained from the CSIRO data portal Cox, Borges, and Lowe, 2018. The original data looks like the figure E.1 below with 3521563 points.

FIGURE E.1: UAV Tower colourised point cloud data, adapted from Cox, Borges, and Lowe, 2018

References

Aguilar, Wilbert G and Stephanie G Morales (2016). "3D environment mapping using the Kinect V2 and path planning based on RRT algorithms". In: *Electronics* 5.4, p. 70.

Akgun, B. and M. Stilman (2011). "Sampling heuristics for optimal motion planning in high dimensions". In: *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2640–2645. DOI: 10.1109/IROS.2011.6095077.

Alexa, M. et al. (2003). "Computing and rendering point set surfaces". In: *IEEE Transactions on Visualization and Computer Graphics* 9.1, pp. 3–15. DOI: 10.1109/TVCG.2003.1175093.

Amenta, Nina, Marshall Bern, and Manolis Kamvysselis (1998). "A new Voronoi-based surface reconstruction algorithm". In: *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 415–421.

Bellman, Richard (1957). *Dynamic Programming*. 1st ed. Princeton, NJ, USA: Princeton University Press.

Berger, Matthew et al. (2014). "State of the art in surface reconstruction from point clouds". In: *EUROGRAPHICS star reports*. Vol. 1. 1, pp. 161–185.

Bernardini, Fausto et al. (1999). "The ball-pivoting algorithm for surface reconstruction". In: *IEEE transactions on visualization and computer graphics* 5.4, pp. 349–359.

Bhatia, Amit and Emilio Frazzoli (2004). "Incremental search methods for reachability analysis of continuous and hybrid systems". In: *International Workshop on Hybrid Systems: Computation and Control*. Springer, pp. 142–156.

Black, Paul (2017). "Algorithms and Theory of Computation Handbook". In: *Dictionary of Algorithms and Data Structures*. URL: https://www.nist.gov/dads/HTML/tree.html.

Bose, Prosenjit et al. (2011). "A survey of geodesic paths on 3D surfaces". In: *Computational Geometry* 44.9, pp. 486 –498.

Branicky, Michael S et al. (2006). "Sampling-based planning, control and verification of hybrid systems". In: *IEE Proceedings-Control Theory and Applications* 153.5, pp. 575–590.

Breitenmoser, Andreas and Roland Siegwart (2012). "Surface reconstruction and path planning for industrial inspection with a climbing robot". In: *2012 2nd International Conference on Applied Robotics for the Power Industry (CARPI)*, pp. 22–27. DOI: 10.1109/CARPI.2012.6473354.

Canny, John and John Reif (1987). "New lower bound techniques for robot motion planning problems". In: *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*. IEEE, pp. 49–60.

Carsten, Joseph, Dave Ferguson, and Anthony Stentz (2006). "3d field d: Improved path planning and replanning in three dimensions". In: *2006 IEEE/RSJ international conference on intelligent robots and systems*. IEEE, pp. 3381–3386.

Cebisile, Mthabela, Withey Daniel, and Kuchwa-Dube Chioniso (2021a). "Ground Robot Path Planning on 3D Mesh Surfaces Using Bi-directional RRT based on Local Regions". In: *2021 Rapid Product Development Association of South Africa - Robotics and Mechatronics - Pattern Recognition Association of South Africa (RAPDASA-RobMech-PRASA)*, pp. 1–6.

— (2021b). "RRT based path planning for mobile robots on a 3D surface mesh". In: *2021 Southern African Universities Power Engineering Conference - Robotics and Mechatronics - Pattern Recognition Association of South Africa (SAUPEC/RobMech/PRASA)*, pp. 1–6.

Chen, Jindong and Yijie Han (1990). "Shortest Paths on a Polyhedron". In: New York, NY, USA: Association for Computing Machinery, 360–369. ISBN: 0897913620. DOI: 10.1145/98524.98601. URL: https://doi.org/10.1145/98524.98601.

Cignoni, Paolo et al. (2008). "MeshLab: an Open-Source Mesh Processing Tool". In: *Eurographics Italian Chapter Conference*. Ed. by Vittorio Scarano, Rosario De Chiara, and Ugo Erra. The Eurographics Association. ISBN: 978-3-905673-68-5.

Colas, Francis et al. (2013). "3D path planning and execution for search and rescue ground robots". In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, pp. 722–727.

Community, Blender Online (2018). *Blender - a 3D modelling and rendering package*. Blender Foundation. Stichting Blender Foundation, Amsterdam. URL: http://www.blender.org.

Costa, M. M. and M. F. Silva (2019). "A Survey on Path Planning Algorithms for Mobile Robots". In: *2019 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 1–7. DOI: 10.1109/ICARSC.2019.8733623.

Cox, Mark, Paulo Borges, and Thomas Lowe (2018). "PaintCloud". In: 3. URL: https://doi.org/10.4225/08/5b31c59fca196.

Dantzig, George (2016). *Linear programming and extensions*. Princeton university press.

De Filippis, Luca, Giorgio Guglieri, and Fulvia Quagliotti (2012). "Path planning strategies for UAVS in 3D environments". In: *Journal of Intelligent & Robotic Systems* 65.1-4, pp. 247–264.

Dijkstra, E. W. (1959). "A Note on Two Problems in Connexion with Graphs". In: *NUMERISCHE MATHEMATIK* 1.1, pp. 269–271.

do Rego, R. L. M. E., A. F. R. Araujo, and F. B. de Lima Neto (2007). "Growing Self-Organizing Maps for Surface Reconstruction from Unstructured Point Clouds". In: *2007 International Joint Conference on Neural Networks*, pp. 1900–1905. DOI: 10.1109/IJCNN.2007.4371248.

Dong, Yiqun, Changhong Fu, and Erdal Kayacan (2016). "RRT-based 3D path planning for formation landing of quadrotor UAVs". In: *2016 14th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 1–6. DOI: 10.1109/ICARCV.2016.7838567.

Edelsbrunner, H., D. Kirkpatrick, and R. Seidel (1983). "On the shape of a set of points in the plane". In: *IEEE Transactions on Information Theory* 29.4, pp. 551–559. DOI: 10.1109/TIT.1983.1056714.

Elbanhawi, Mohamed and Milan Simic (2014). "Sampling-based robot motion planning: A review". In: *Ieee access* 2, pp. 56–77.

Fabri, Andreas and Sylvain Pion (2009). "CGAL: The computational geometry algorithms library". In: *Proceedings of the 17th ACM SIGSPATIAL international conference on advances in geographic information systems*, pp. 538–539.

Ferguson, Dave and Anthony Stentz (2006). "Using interpolation to improve path planning: The Field D* algorithm". In: *Journal of Field Robotics* 23.2, pp. 79–101.

Fortune (1992). "Voronoi Diagrams and Delaunay Triangulations". In: *Computing in Euclidean Geometry, Edited by Ding-Zhu Du and Frank Hwang, World Scientific, Lecture Notes Series on Computing – Vol. 1*. URL: citeseer.nj.nec.com/article/fortune92voronoi.html.

Gelperin, David (1977). "On the optimality of A*". In: *Artificial Intelligence* 8.1, pp. 69–76. DOI: https://doi.org/10.1016/0004-3702(77)90005-4.

Gopi, M. and Shankar Krishnan (Feb. 2002). "A Fast and Efficient Projection-Based Approach for Surface Reconstruction". In: vol. 1, pp. 179–186. ISBN: 0-7695-1846-X. DOI: 10.1109/SIBGRA.2002.1167141.

Hart, P. E., N. J. Nilsson, and B. Raphael (1968). "A Formal Basis for the Heuristic Determination of Minimum Cost Paths". In: *IEEE Transactions on Systems Science and Cybernetics* 4.2, pp. 100–107. DOI: 10.1109/TSSC.1968.300136.

Hart, Peter E, Nils J Nilsson, and Bertram Raphael (1968). "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2, pp. 100–107.

Hernández, Juan D. et al. (2015). "On-line 3D Path Planning for Close-proximity Surveying with AUVs". In: *IFAC-PapersOnLine* 48.2, pp. 50–55. ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2015.06.009.

Hoppe, Hugues et al. (1992). "Surface Reconstruction from Unorganized Points". In: *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*. Association for Computing Machinery, 71–78. ISBN: 0897914791. DOI: 10.1145/133994.134011. URL: https://doi.org/10.1145/133994.134011.

Hrabar, Stefan (2008). "3D path planning and stereo-based obstacle avoidance for rotorcraft UAVs". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 807–814. DOI: 10.1109/IROS.2008.4650775.

Jaillet, Leonard, Juan Cortes, and Thierry Simeon (2008). "Transition-based RRT for path planning in continuous cost spaces". In: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2145–2150. DOI: 10.1109/IROS.2008.4650993.

Jaishankar, S and RN Pralhad (2011). "3D off-line path planning for aerial vehicle using distance transform technique". In: *Procedia Computer Science* 4, pp. 1306–1315.

Jordan, M. and A. Perez (2013). "Optimal Bidirectional Rapidly-Exploring Random Trees". In: MIT Libraries.

Kallmann, Marcelo (2010). "Navigation queries from triangular meshes". In: *International Conference on Motion in Games*. Springer, pp. 230–241.

Kallmann, Marcelo and Mubbasir Kapadia (2014). "Navigation meshes and real-time dynamic planning for virtual worlds". In: *ACM SIGGRAPH 2014 Courses*, pp. 1–81.

Kaneva, Biliana and J. O'Rourke (2000). "An Implementation of Chen Han's Shortest Paths Algorithm". In: *CCCG*.

Karaman, Sertac and Emilio Frazzoli (2010). "Incremental sampling-based algorithms for optimal motion planning". In: *Robotics Science and Systems VI* 104, p. 2.

— (2011). "Sampling-based algorithms for optimal motion planning". In: *The international journal of robotics research* 30.7, pp. 846–894.

Kavraki, Lydia E. and Steven M. LaValle (2008). "Motion Planning". In: *Siciliano B., Khatib O. (eds) Springer Handbook of Robotics*. Springer, Berlin, Heidelberg. DOI: 10.1007/978-3-540-30301-5_6.

Kazhdan, Michael, Matthew Bolitho, and Hugues Hoppe (2006). "Poisson Surface Reconstruction". In: *Proceedings of the Fourth Eurographics Symposium on Geometry Processing*. Goslar, DEU: Eurographics Association, 61–70. ISBN: 3905673363.

Khatamian, A. and H.R. Arabnia (2016). "Survey on 3D Surface Reconstruction". In: *Journal of Information Processing Systems* 12.3, pp. 338–357.

Kiazyk, Stephen, Sébastien Loriot, and Éric Colin de Verdière (2020). "Triangulated Surface Mesh Shortest Paths". In: *CGAL User and Reference Manual*. 5.1. CGAL Editorial Board.

Klemm, S. et al. (2015). "RRT-Connect: Faster, asymptotically optimal motion planning". In: *2015 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1670–1677. DOI: 10.1109/ROBIO.2015.7419012.

Koenig, Sven and Maxim Likhachev (2002). "D* Lite". In: *AAAI/IAAI* 28.

Koenig, Sven, Maxim Likhachev, and David Furcy (2004). "Lifelong Planning A". In: *Artificial Intelligence* 155.1, pp. 93 –146. DOI: https://doi.org/10.1016/j.artint.2003.12.001.

Koukuntla, Snehal Reddy et al. (2019). "Deep Learning rooted Potential piloted RRT* for expeditious Path Planning". In: *Proceedings of the 2019 4th International Conference on Automation, Control and Robotics Engineering*, pp. 1–8.

Kuffner, James J and Steven M LaValle (2000). "RRT-connect: An efficient approach to single-query path planning". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*. Vol. 2. IEEE, pp. 995–1001.

Latombe, Jean-Claude (1999). "Motion planning: A journey of robots, molecules, digital actors, and other artifacts". In: *The International Journal of Robotics Research* 18.11, pp. 1119–1128.

LaValle, S. M. (2006). *Planning Algorithms*. Cambridge University Press.

LaValle, S. M. and J. J. Kuffner (1999). "Randomized kinodynamic planning". In: *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*. Vol. 1, 473–479 vol.1. DOI: 10.1109/ROBOT.1999.770022.

LaValle, Steven M (1998). "Rapidly-exploring random trees: A new tool for path planning". In:

LaValle, Steven M and James J Kuffner (2001). "Rapidly-exploring random trees: Progress and prospects". In: *Algorithmic and computational robotics: new directions* 5, pp. 293–308.

Li, Jiadong et al. (2014). "RRT-A* Motion planning algorithm for non-holonomic mobile robot". In: *SICE Annual Conference (SICE), 2014 Proceedings of the*. IEEE, pp. 1833–1838.

Likhachev, Maxim et al. (2008). "Anytime search in dynamic graphs". In: *Artificial Intelligence* 172.14, pp. 1613 –1643. ISSN: 0004-3702. DOI: https://doi.org/10.1016/j.artint.2007.11.009.

Lim, Seng Poh and Habibollah Haron (2014). "Surface reconstruction techniques: a review". In: *Artificial Intelligence Review* 42.1, pp. 59–78.

Lindemann, S. R. and S. M. LaValle (2004). "Steps toward derandomizing RRTs". In: *Proceedings of the Fourth International Workshop on Robot Motion and Control (IEEE Cat. No.04EX891)*, pp. 271–277. DOI: 10.1109/ROMOCO.2004.240739.

Lozano-Perez, Tomas (1983). "Spatial planning: A configuration space approach". In: *IEEE transactions on computers* 2, pp. 108–120.

Masehian, Ellips and MR Amin-Naseri (2004). "A voronoi diagram-visibility graph-potential field compound algorithm for robot path planning". In: *Journal of Field Robotics* 21.6, pp. 275–300.

Mitchell, J S.B., D M Mount, and C H Papadimitriou (1987). "The discrete geodesic problem". In: *SIAM J. Comput.* 16. DOI: 10.1137/0216045.

Mount, Dave (1985). "On Finding Shortest Paths on Convex Polyhedra." In:

Noreen, Iram, Amna Khan, and Z. Habib (2016a). "Optimal Path Planning using RRT* based Approaches: A Survey and Future Directions". In: *International Journal of Advanced Computer Science and Applications* 7.

Noreen, Iram, Amna Khan, and Zulfiqar Habib (2016b). "A comparison of RRT, RRT* and RRT*-smart path planning algorithms". In: *International Journal of Computer Science and Network Security (IJCSNS)* 16.10, p. 20.

Nykamp, Duane (2020). "Graph definition". In: *Math Insight*. URL: https://mathinsight.org/definition/graph.

Otte, Michael and Emilio Frazzoli (2016). "RRTX: Asymptotically optimal single-query sampling-based motion planning with quick replanning". In: *The International Journal of Robotics Research* 35.7, pp. 797–822.

Petres, Clment et al. (2007). "Path planning for autonomous underwater vehicles". In: *IEEE Transactions on Robotics* 23.2, pp. 331–341.

Pohl, Ira Sheldon (1969). "Bi-Directional and Heuristic Search in Path Problems". PhD thesis.

Pütz, Sebastian et al. (2016). "3D Navigation Mesh Generation for Path Planning in Uneven Terrain". In: *IFAC-PapersOnLine* 49.15, pp. 212–217.

Qureshi, Ahmed Hussain and Yasar Ayaz (2016). "Potential functions based sampling heuristic for optimal path planning". In: *Autonomous Robots* 40.6, pp. 1079–1093.

Russell, Stuart Jonathan et al. (2003). *Artificial intelligence: a modern approach*. Vol. 2. 9. Prentice hall Upper Saddle River.

Sariff, N and Norlida Buniyamin (2006). "An overview of autonomous mobile robot path planning algorithms". In: *Research and Development, 2006. SCOReD 2006. 4th Student Conference on*. IEEE, pp. 183–188.

Schøler, Flemming, Anders la Cour-Harbo, and Morten Bisgaard (2012). "Generating approximative minimum length paths in 3D for UAVs". In: *Intelligent Vehicles Symposium (IV), 2012 IEEE*. IEEE, pp. 229–233.

Schwartz, Jacob T and Micha Sharir (1983). "On the ?piano movers? problem. II. General techniques for computing topological properties of real algebraic manifolds". In: *Advances in applied Mathematics* 4.3, pp. 298–351.

Sethian, James Albert (1999). *Level set methods and fast marching methods: evolving interfaces in computational geometry, fluid mechanics, computer vision, and materials science*. Vol. 3. Cambridge university press.

Sharir, Micha and Amir Schorr (1984). "On Shortest Paths in Polyhedral Spaces". In: *Proceedings of the Sixteenth Annual ACM Symposium on Theory of Computing*. Association for Computing Machinery, 144–153. ISBN: 0897911334. DOI: 10.1145/800057.808676.

Siegwart, Roland, Illah Reza Nourbakhsh, and Davide Scaramuzza (2011). *Introduction to autonomous mobile robots*. MIT press.

Škoda, Jan and Roman Barták (2017). "3D Navigation for a Mobile Robot". In: *Iberian Robotics conference*. Springer, pp. 345–356.

— (2018). "3D Navigation for a Mobile Robot". In: *ROBOT 2017: Third Iberian Robotics Conference*. Ed. by Anibal Ollero et al. Cham: Springer International Publishing, pp. 345–356. ISBN: 978-3-319-70836-2.

Stentz, Anthony (1994). "Optimal and efficient path planning for partially-known environments". In: *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*. IEEE, pp. 3310–3317.

Stoyanov, Todor et al. (2010). "Path planning in 3D environments using the normal distributions transform". In: *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, pp. 3263–3268.

Stuart, Russell, Norvig Peter, et al. (2003). *Artificial intelligence: a modern approach*.

Surazhsky, Vitaly et al. (2005). "Fast Exact and Approximate Geodesics on Meshes". In: *ACM Trans. Graph.* 24.3, 553–560. DOI: 10.1145/1073204.1073228. URL: https://doi.org/10.1145/1073204.1073228.

Toll, Wouter van et al. (2016). "A Comparative Study of Navigation Meshes". In: *Proceedings of the 9th International Conference on Motion in Games*. Association for Computing Machinery, 91–100. DOI: 10.1145/2994258.2994262.

Urmson, Chris and Reid Simmons (2003). "Approaches for heuristically biasing RRT growth". In: *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*. Vol. 2. IEEE, pp. 1178–1183.

Vechersky, Pavel et al. (2018). "Colourising Point Clouds Using Independent Cameras". In: *IEEE Robotics and Automation Letters* 3.4, pp. 3575–3582. DOI: 10.1109/LRA.2018.2854290.

Xin, Shi-Qing and Guo-Jin Wang (2009). "Improving Chen and Han's algorithm on the discrete geodesic problem". In: *ACM Transactions on Graphics (TOG)* 28.4, pp. 1–8.

Xu, Shengdong et al. (2015). "Real-time 3D navigation for autonomous vision-guided MAVs". In: *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*. IEEE, pp. 53–59.

Yan, Fei, Yi-Sha Liu, and Ji-Zhong Xiao (2013). "Path planning in complex 3D environments using a probabilistic roadmap method". In: *International Journal of Automation and computing* 10.6, pp. 525–533.

Yang, Kwangjin and Salah Sukkarieh (2008). "3D smooth path planning for a UAV in cluttered natural environments". In: *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE, pp. 794–800.

Yang, Liang et al. (2016). "Survey of robot 3D path planning algorithms". In: *Journal of Control Science and Engineering* 2016, p. 5.

Yershova, A. et al. (2005). "Dynamic-Domain RRTs: Efficient Exploration by Controlling the Sampling Domain". In: *Proceedings of the 2005 IEEE International Conference on Robotics and Automation*, pp. 3856–3861. DOI: 10.1109/ROBOT.2005.1570709.

Yuncheng, Li and Shao Jie (2017). "A revised Gaussian distribution sampling scheme based on RRT* algorithms in robot motion planning". In: *Control, Automation and Robotics (ICCAR), 2017 3rd International Conference on*. IEEE, pp. 22–26.

Zhao, Hong-Kai, Stanley Osher, and Ronald Fedkiw (2001). "Fast surface reconstruction using the level set method". In: *Proceedings IEEE Workshop on Variational and Level Set Methods in Computer Vision*. IEEE, pp. 194–201.

Zucker, M., J. Kuffner, and M. Branicky (2007). "Multipartite RRTs for Rapid Replanning in Dynamic Environments". In: *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 1603–1609. DOI: 10.1109/ROBOT.2007.363553.