Path Planning Framework for Unmanned Ground Vehicles on Uneven Terrain

by

Olzhas Adiyatov

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Applied Science
in
Electrical and Computer Engineering

Waterloo, Ontario, Canada, 2022

Author's Declaration

Abstract

In this thesis, I address the problem of long-range path planning on uneven terrain for non-holonomic wheeled mobile robots (WMR). Uneven terrain path planning is essential for search-and-rescue, surveillance, military, humanitarian, agricultural, constructing missions, etc. These missions necessitate the generation of a feasible sequence of waypoints, or reference states, to navigate a WMR from the initial location to the final target location through the uneven terrain. The feasibility of navigating through a given path over uneven terrain can be undermined by various terrain features. Examples of such features are loose soil, vegetation, boulders, steeply sloped terrain, or a combination of all of these elements. I propose a three-stage framework to solve the problem of rapid long-range path planning. In the first stage, RRT-Connect provides a rapid discovery of the feasible solution. Afterward, Informed RRT* improves the feasible solution. Finally, Shortcut heuristics improves the solution locally. To improve the computational speed of path planning algorithms, we developed an accelerated version of the traversability estimation on point clouds based on Principal Component Analysis. The benchmarks demonstrate the efficacy of the path planning approach.

Acknowledgments

I would like to thank my supervisors Professors Stephen Smith and Baris Fidan for the guidance through my time at University of Waterloo. I would also like to thank my colleagues in the Autonomous Systems Laboratory for the insightful discussions. I would like to thank Frank, Barry, Armin, Rinat, Assylbek, and Iliyas for being good friends. Finally, I would like to thank my family for their love and support.

Dedication

This is dedicated to my family.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

Over the past 60 years, robotics and automation have demonstrated successful results in various aspects of human life. Robotic arms began replacing people in dirty, dull, and dreary jobs in the 1970s. However, until the beginning of the 21st century, robots were mainly deployed only in a controlled industrial setting or research laboratories at universities. Starting from 2005, owing to developments in computer hardware, sensors, and actuators, we have observed great progress in robotics. Interestingly, the advancement occurs during the same period as the DARPA Grand Challenge. The challenge has demonstrated the efficacy of self-driving vehicles. Simultaneously, quadrotors became ubiquitous, and they have found applications in digital photography, videography, and entertainment.

The Fukushima accident in 2011 and the COVID-19 outbreak have shown the importance of autonomous robots for disaster management. Robotic systems can provide significant assistance during anthropogenic and natural disasters, through search and rescue. Unmanned Aerial Vehicles (UAVs) are commonly used for surveillance of the disaster zone [66], serving as "eyes" for Unmanned Ground Vehicles (UGVs), which will undertake the rescue part of the operation. A UAV/UGV autonomous cooperation is an interesting robotic system configuration for search-and-rescue robotics (SAR) [62]. SAR, like any other field of robotics, includes several distinct aspects: mapping that enables us to represent the surroundings, localization that enables us to find our robot on the previously generated map, and planning that produces a sequence of actions we need to undertake to reach the goal, and obtain controls that ensure the execution of the generated plan. Autonomous UGVs are also required for Sample Return Robot scenarios [15].

Path planning is a major problem in autonomous robotics. Solving a path planning problem results in a sequence of safe robot states from a start state to a goal. The basic safety index for path planning is an absence of obstacle collisions.

In the case of an autonomous ground vehicle that must navigate a terrain, we can take into consideration the stability of the autonomous ground vehicle, the operator's safety/comfort, and/or clearance from obstacles. Uneven terrain delivers a significant hindrance to ground robot mobility. For a successful traversal of uneven terrain, we should consider geometry, the material of the terrain. For example, a wheeled ground vehicle, due to its physical capabilities, cannot traverse a very steep hill, or the wheels can get stuck in the mud. A careful reader may notice that the mechanical design of a ground robot greatly affects the traversability of uneven terrain, i.e., legged robots are better at crossing uneven terrain. Hence, path planning on uneven terrain is an interesting problem to tackle from various aspects including mapping, planning, and control of the robot in the environment.

1.1 Motivation

Recently, self-driving car navigation in a structured environment has made significant progress due to advancements in computation and usage of deep learning [47]. Navigation in structured environments allows the exploitation of simplifications, such as travelling on a flat plane. Self-driving cars treat obstacles as objects that should be avoided. However, unlike navigation in a structured environment, rough terrain navigation requires assessment of the environment for navigable regions. The notion of an obstacle in rough terrain depends on the mobile robot itself. Navigable regions are different in a way that flat terrain is easier to traverse compared to rugged terrain or sloped terrain or a combination of both. The function that determines the navigability of terrain is called the traversability assessment function. Autonomous navigation on uneven terrain has a wide variety of applications. Search and rescue, space exploration. Autonomous navigation for an unmanned ground vehicle consists of well-known and defined subsystems:

- Perception, Mapping, and Localization
- Path or Motion Planning and Control

The perception subsystem consists of sensors mounted on the robot. The perception subsystem is utilized as an input to the mapping and localization subsystem. These two components update the internal model of the robot and the surrounding world. The robot and world model are used further for planning and control. Planning generates a motion plan that is executed by a control subsystem. Fig. 1.1 outlines the block diagram for the autonomous navigation system. Currently, sampling-based planners [37] are a state-of-

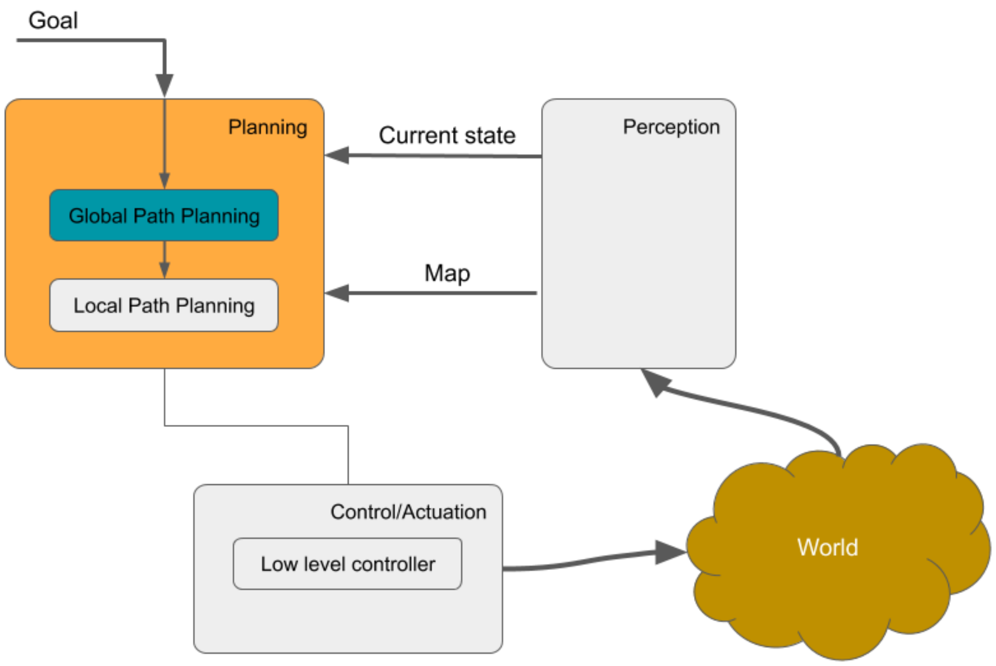


Figure 1.1: General architecture of the Autonomous Navigation system for an Unmanned Ground Vehicle.

the-art approach for the solution of motion planning problems. In short, sampling-based planners iteratively explore the map in a random fashion. To determine a solution this kind of planner is executed for a considerable number of iterations. Hence, it requires each of the subroutines to be computationally inexpensive, so that one can have reasonable execution times to obtain solution paths. The following section outlines the contribution of this thesis.

1.2 Contribution

The main contribution of the thesis is the design of a path planning framework for uneven terrain navigation. The key aspects of the proposed path planning framework include traversability estimation and the layout of the approach. The path planning framework combines two existing sampling-based planners into a multi-stage planner that obtains the feasible path first and refines the path afterwards. For traversability assessment, I propose an accelerated traversability estimation function of the existing approach by utilizing the auxiliary information generated by Principal Component Analysis. Finally, working with our industry partner, the path planning framework was deployed on a hardware platform operating in a large off-road environment.

1.3 Organization

Chapter 2 reviews the traversability estimation and path planning on uneven terrain. Chapter 3 introduces the problem of traversability assessment and points out the consequential effect of the computationally expensive traversability estimation function. Chapter 4 introduces the problem of path planning on uneven terrain. It clarifies the proposed three-stage hierarchical approach to the path planning problem. Finally, the last chapter concludes the thesis and provides directions for future works.

Chapter 2

Background and Literature Survey

Historically, most of the research on path planning for mobile robots has focused on indoor applications. In the indoor setting, a robot faces a structured environment, mainly consisting of straight lines and flat planes. The indoor setting greatly simplifies the path planning problem, as the mobile robot operates on the flat, leveled ground that can be easily represented with a 2-D grid. In essence, this kind of path planning problem disregards the interaction with the terrain except for obstacle collisions. In addition, flat or leveled terrain simplifies the modeling of the system dynamics of the robot.

However, mobile robots cannot exclusively facilitate human operations in warehouses, laboratories, or services for the elderly. Outdoors are also of great interest, for example, search and rescue, agriculture, military, development, and humanitarian. The problem of navigation in uneven terrain has two main aspects: mapping (representation) and planning. What is an uneven terrain? According to Iagnemma and Dubowsky in [22] rough or *uneven terrain* can be loosely described as an unstructured terrain with features that can cause a robot to become entrapped or lose balance. Examples of these features are loose soil, vegetation, boulders, steeply sloped terrain, or a combination of all of these elements.

Before we continue with our literature review, let us provide the notation that will be utilized throughout the thesis. Afterward, we will talk about the representation of uneven terrain first, followed by the discussion on the planning aspect of the navigation problem.

In this work, the robot is operating in an outdoor environment on uneven terrain. All possible variations of the environment can be expressed with a physical state space $\mathcal{P}$. We also have to model the internal state of the robot in the surrounding environment, this will be denoted as $\mathcal{I}$. Robots have the means to interact with the environment, and it can be accomplished by exciting the system, this will be denoted as the Action/Control space $\mathcal{U}$.

Now, we have the necessary definitions to articulate the following sections over the representation of the terrain and the methods to generate motion plans on it.

2.1 Terrain representation and traversability measures

The terrain representation governs the solution approach to the path planning problem. Depending on the application, we have to account for the geometry/topology of terrain and for texture, for example, vegetation and dust, trees, overhanging structures, etc. The seminal work on environment representation was performed by Moravec and Effes in [45]. They introduced occupancy grid maps, a widely used probabilistic approach to represent the surroundings of a mobile robot. The original occupancy grid was generated with a rangefinder. The probabilistic foundation of the approach allows roboticists to represent obstacles, free space, and unknown regions. Occupancy grid maps come with the drawback that the approach only addresses mobile robot navigation on flat surfaces in the presence of obstacles that are detected by a rangefinder or LIDAR.

The following is a review of the uneven terrain representations.

2.1.1 Elevation grid maps

One of the most common representations of the terrain is a digital elevation map (DEM) [18, 34]. In this respect, we have a function $h(x, y)$ that outputs an elevation value for a certain x and y. Generally, DEM is a 2-D matrix with elevation information in each cell. In this case we have $h(x_i, y_i)$, where x_i and y_i are uniformly separated from each other on the respective axes. DEMs provide efficient storage and retrieval of terrain elevation information, yet they inherently suffer from a lack of ability to represent overhanging structures (bridges, trees, poles). Furthermore, since DEMs are uniformly discretized, they tend to lose terrain shape details. To alleviate the limitations of DEMs, the extension of the method was proposed by Pfaff in [51]. Multi-Level Surface maps (MLS maps) utilize scene segmentation. Further, locally connected components are used as an elevation map. The elevation grid uniformity is also a serious thing to consider because of the mismatch of the sensor resolution and map resolution. Vegetation representation is also a serious problem with elevation grids.

2.1.2 Meshes

As stated above, elevation grids are an efficient and easily implementable method, yet they are restricted to certain types of terrain. We can go further and use polyhedrons to represent the terrain. A polygon mesh is a collection of vertices, edges, and faces that represent a polyhedral object. Unlike DEMs, meshes are capable of representing terrain shapes at various detail levels (e.g. overhanging structures). In addition to that, meshes are a memory-efficient representation of surfaces. The significant disadvantage of the generation of meshes of a complete environment is that it is a computationally intensive procedure [32, 40]. The correct extraction of surfaces is the main challenge in working with mesh maps. Ruetz et al. in [57] introduce *On Visible Point Clouds Mesh* (OVPC Mesh). This mesh is generated on-demand using the generalized hidden point removal (GHPR) operator to determine the visible points. To estimate the traversability of the mesh, the method considers the normal plane to a polygon of the mesh and the difference in height of the vertices of the polygon. Their benchmarks have demonstrated that OVPC mesh generation is faster than Octomap (3-D version of the occupancy grids) generation [20], and has more stable timing compared to elevation mapping generation.

2.1.3 Point clouds and voxel maps

A point cloud is a set of data points relative to a coordinate frame. There are different ways to obtain a point cloud: LIDAR, depth cameras (e.g., Microsoft Kinect), structure-from-motion [60]. One of the by-products of the SLAM is point clouds (point sets). Unlike

DEM, point clouds do not have limitations on geometry representation. The key issue of point sets for terrain representation is memory consumption. To address the issue of high memory consumption of point clouds, Hornung et al. introduced OctoMaps in [20]. OctoMap is the octree-based [44] 3-D occupancy grid that provides probabilistic occupancy estimation. Having discussed the uneven terrain representations, let us provide a review of the methods of assessing the suitability of sections of the map for mobile robot locomotion.

2.1.4 Traversability and cost maps

In different types of terrain, mobile robots have different levels of maneuverability based on their mechanical design. Thus, an elevation map must be transformed into a cost map according to an appropriate cost function. We will refer to this cost function as *traversability*. Traversability measures the ability of a mobile robot to successfully occupy and traverse a patch of terrain (or the environment, in general). In this work, the definition of traversability takes into consideration the kinematic and dynamic constraints of a mobile robot and terrain characteristics. Papadakis in [48] pointed out two to estimate traversability without physical interaction with the terrain:

1. Geometry-based (Point cloud, LIDAR, elevation mapping);
2. Appearance-based (RGB images, multi-spectrum imaging).

Here are a few examples of safety factors for wheeled mobile robots using geometry-based methods slope of the terrain [13], ground clearance [46], terrain roughness [48]. There is a slight variability in the way one can determine terrain roughness or slope. Let's consider some of the methods to determine these quantities.

Principal component analysis (PCA) is the go-to method for normal vector estimation on point clouds. Intuitively, one can think of PCA as a process of fitting an ellipsoid to data points. Lalonde et al. in [35] used the PCA outputs as features for shape detection. In their method, they have studied 3 cases of point distribution of the points on a point cloud generated by LIDAR. The method demonstrated the efficacy of PCA to distinguish three classes: sphere, plane (a thin disk), and cylinder. A variant of the point cloud traversability feature was explored in this paper [55]. The traversability feature generation algorithm can be applied both in indoor and outdoor applications. However, the researchers reported that the outdoor application suffers from not being robust. The related work on environment representation for uneven terrain addressed the problem deterministically. Krusi et. al. in [32] also utilized PCA to estimate the position of the mobile robot and the roughness of the terrain underneath the mobile robot.

Full or partial mobile robot and terrain interaction might be simulated and used for traversability estimation. High-fidelity wheeled mobile robot kinematic and dynamic models were introduced in [61]. The paper demonstrated 1000 times faster than real-time computation, which enables us to use the model for path planning. Ma and Shiller [43] proposed to estimate the pose by setting up a simple interaction of robot-terrain dynamics, which requires a terrain to be modeled as a B-patch (which is a 3-D version of a B-spline). In their method, researchers simulate a robot that is brought down on the terrain from a small height. A drawback of the method, we should provide the initial pose height of the vehicle.

Roan in [56] surveys tip-over measures. The study considers three algorithms, the zero moment point (ZMP), the force angle stability measure (FA), and the moment height stability measure (MHS). The clearance determination algorithm for the implementation of the planet rover was discussed in [46].

Thrun et al. introduced probabilistic terrain analysis in [64], which enabled the Stanford team to win DARPA Grand Challenge. As research in the field of determining a nice traversability cost function evolved, it turned out that Machine Learning techniques might find a better use than a hand-picked traversability cost function. In [58] the authors employ a Gaussian Process machine learning approach to classify 2 types of terrain patches, traversable or not, followed by the HRA* planner and costmap manipulation.

2.2 Motion planning

Motion planning is a field of robotics that solves the problem of finding a safe sequence of states of the robot from the initial state to the goal state (region). The basic motion planning problem, also known as the Piano mover's problem, or geometric motion planning, focuses on generating the sequence of states for a robot without hitting any obstacles. Reif has shown geometric motion planning to be a PSPACE-hard problem in [54]. Related work on planning on uneven terrain falls into five main categories:

1. Graph search algorithms using motion primitives.
2. Artificial potential fields planners.
3. Sampling-based motion planners.
4. Variational methods.
5. Learning-based motion planners.

2.2.1 Graph search algorithms

As was stated before, modeling the behavior of the system is an essential part of planning. Planners have a representation of the configuration space of the robot. A graph is a common data structure that represents the $\mathcal{C}$-space or environment of the robot. Mathematically, we record them as $G = (V, E)$, where V is a set of vertices of the graph, and it encodes discrete states of the space, while E are the edges that represent the transition from one state to another. E is a set of pairs of vertices of V. A grid is a type of graph. Each cell of a grid is a vertex and depending on the connectivity of the grid, each cell can be connected to 4 or 8 of its immediate neighbors.

The main idea behind the graph search algorithm method is to represent the terrain as an interconnected graph with weighted edges depending on an objective, most frequently the path length, followed by the execution of a shortest-path algorithm with a modified path length objective to account for the terrain geometry and the robot capabilities. In [23], the researchers developed a framework for a planetary rover, which considers the terrain information. The framework includes a model-based assessment of the terrain with the analysis of rover stability followed by the A* [16] is used as a path planning algorithm. The researchers in [7] proposed a two-stage A*-based algorithm for the generation of drivable trajectories. The first stage is called Hybrid-State A* search, and it utilizes two heuristics. The first heuristic is based on the shortest nonholonomic path to the goal with no obstacles present. The second is the shortest path to the goal pose, considering obstacles but ignoring the nonholonomic constraints of the vehicle. The greater of the two values is then used as a heuristic function in the regular A* algorithm for a discretized lattice. The second stage of the approach is the local optimization of the trajectory based on the conjugate gradient descent algorithm. Unfortunately, this work only addresses the problem of trajectory generation on flat terrain with binary obstacles.

The authors of [41] present a ROS [52] package for layered costmaps. The main advantage of their method is that by storing separate layers independently, a user can modify the layers and add them all and use the resultant map for path planning, consequently used with the A* search algorithm.

Optimal control-based trajectory generation was shown to be able to include terrain information and rover dynamics [21]. Howard and Kelly employ numerical optimization to solve this. This approach accounts for risk, obstacles, duration, and energy consumption. The proposed framework can operate in real time. They also used neural networks to provide a good initial estimate for the trajectory generation routine.

2.2.2 Artificial potential fields

The use of the artificial potential field (APF) [42, p. 386] for path planning was inspired by the potential energy fields, such as magnetic or gravitational fields. In these natural potential fields, a particle tends to move from higher potential regions to lower potential regions. The concept of a particle is applicable to a robot that navigates the environment. Obstacles and untraversable regions have higher potential, thus repelling the robot from being near them. The goal is a lower potential region (minimum). Following the negative gradient direction, we will eventually arrive at a minimum of an artificial potential field. The gradient descent algorithm has a drawback due to its myopic nature of the algorithm. The algorithm can get stuck in a local minimum; thus, the robot will not reach the goal region [42]. Therefore, the usual application of the APF is narrowed down to obstacle avoidance [42]. Artificial potential fields were applied to rough terrain path planning in [63].

2.2.3 Sampling-based planners

Sampling-based algorithms can alleviate the problem of getting stuck at a local minimum. Sampling-based planners provide a generic framework that can solve global planning problems as well as local planning problems. In general, a sampling-based motion planner samples randomly the configuration space followed by the construction of a graph until the algorithm discovers the solution. Notable algorithms include Probabilistic Roadmaps [29, 28] and its variants and Rapidly-exploring Random Tree (RRT) [36] and its variants. In 2011, Karaman and Frazzoli presented RRT* [27] —a sampling-based planner that is asymptotically optimal, i.e., an asymptotically optimal planner guarantees the discovery of an optimal solution if the planner is allowed to run for an infinite time if the optimal solution exists. Creating a handcrafted cost function is a challenge with the above-mentioned methods.

In [11] the notion of obstacleness was introduced. Obstacleness is a cost function to determine the traversability for the mobile robot. After establishing the cost function, researchers modified the RRT algorithm to explore low-cost regions first, followed by gradual approval for sampling in higher-cost regions. The article does not provide any guarantees on the optimality of the path.

Brunner et al. in [2, 3] demonstrate a solution to the motion planning problem for a reconfigurable tank tread UGV. The authors introduced several cost functions for the reconfigurable UGV system. After the cost map was generated, RRT* was utilized and, as a performance index, the cumulative cost of each of the states in the path.

Krusi et al. [32] proposed a framework for path planning on point-cloud maps. In this framework, each point in a point cloud receives a value of traversability, and we treat it as a motion planning constraint. The traversability measure consists of two components: the roughness of the terrain patch and the orientation of the robot on this patch. Roughness is estimated for each of the points in the point cloud by querying a smaller sphere and fitting the plane, after computing the distance to each of the points and ordering it from the biggest to the lowest and taking the subset of the points, and finding the variance of the patch. The framework has a hierarchical structure consisting of three main stages. The first step is global path planning solved by the bidirectional RRT-Connect algorithm [38] that considers the static traversability of the terrain. Subsequently, RRT* (the asymptotically optimal RRT) refines the initial feasible solution, taking into account only the path length. Lastly, a gradient-free perturbation-based trajectory optimization generates the solution for the feedback controller. Importantly, the mobile robot cannot turn on the spot or move sideways; hence, researchers represent the segments between valid states as cubic splines with continuous curvature. A trajectory is a sequence of robot states on planar patches, i.e., the motion between states in the path is assumed to be performed on small planes.

Customarily, in sampling-based planners, the candidate points are sampled randomly from a configuration space. Krusi et al. do not sample random states, instead, they randomly sample a point from the point cloud map and convert it to a robot state. Interestingly, the proposed framework cannot handle moving obstacles and stops to recompute the new motion plan every time the environment has changed. Another disadvantage is that during RRT* stage, the planning framework takes into account only the path length as an optimization objective. We can conclude that the behavior at the second stage is unpredictable due to disregard of the traversability criterion. They reported that during the outdoor experiments, wheel slip was reported as an issue.

FMT* [24] as well as other PRM-based planners are notorious for generating paths that have unnecessary turning actions [67]. Zhu et al. in their work [67] addressed the issue by formulating the trajectory smoothing technique as a convex optimization problem. In their work, the path is a sequence of states that are used as key points to smooth the jittery input path. The researchers formulate the mathematical optimization problem using the key points as constraints to the problem in addition to the constraint of the system dynamics of the vehicle. The drawback of their approach is the disregard of the traversability measure, via generating only obstacle-free trajectories.

2.2.4 Learning-based planners

Recently, Machine Learning (ML) has found a variety of applications in robotics and automation. In [5] researchers established a traversability estimation technique using the convolutional neural network that classifies traversable and untraversable patches in the heightmap. Currently, learning-based methods raise concerns regarding their reliability. To the best of the author's knowledge, there are no analytical guarantees for any learning-based planning or control algorithms.

Chapter 3

Ground Vehicle Traversability Assessment for Uneven Terrain

Wheeled mobility is the most popular and energy-efficient approach to ground environment navigation. Though most wheeled mobile robots (WMRs) operate in planar environments, and it simplifies the analysis and design of navigation systems; yet, applications like search and rescue (SAR) require navigation on uneven terrain, and WMR cannot easily traverse all terrains. For instance, a mobile robot navigating irregular terrains such as loose sand or rubbled slopes might experience low wheel traction resulting in poor maneuverability, unlike the case of moving on a flat regular plane. To plan a safe path, we have to establish a measure that will show us if a robot can successfully traverse the region. The robot's ability to traverse the terrain depends both on the robot's kinematic and dynamic constraints and terrain geometry and structural composition.

To characterize all these limitations of the robot's capabilities to traverse the navigable terrain, we will define a notion of *traversability* of the terrain captured by a cost function that indicates the probability of successful traversal of the patch of terrain.

Traversability estimation is a well-studied topic that was initially aimed at planetary exploration, later it finds applications in SAR and Agriculture Robotics [48]. Let us start from a plain language definition of the traversability measure and move on to the more formal definition later in this chapter. A traversable region is a patch of terrain where we

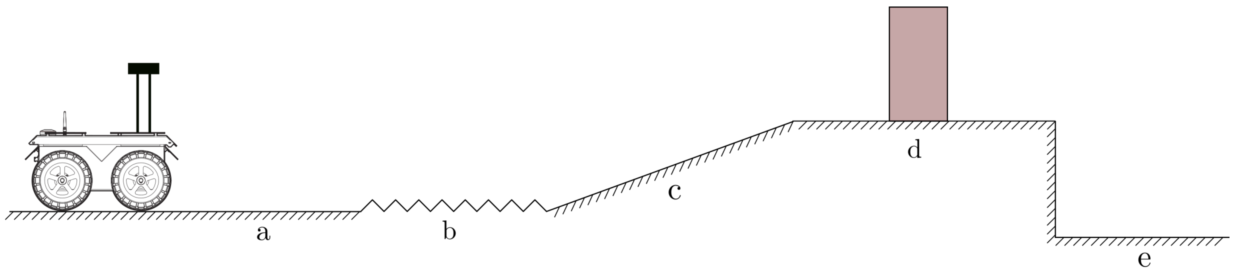


Figure 3.1: Geometrical aspects of traversability. From left to right, (a) robot moving on a plane, (b) robot moving on gravel (c) robot moving up a hill (also risk of losing stability (tip over)), (d), (e) are examples of completely untraversable regions.

can safely place a robot. In other words, a terrain patch will have no steps higher than the wheels can go, and the robot can remain stable on that terrain patch. In the most rudimentary case, we can compose the traversability from two characteristics of the terrain patch:

- *roughness* is defined as the biggest step on the terrain patch, it is related to the irregularity of the underlying terrain;
- *slope* is a component that determines the level of slip and tip-over hazard via constraining the allowable roll and pitch angle of WMR.

The footprint of the robot governs the size of the terrain patch.

Let us assume that a point cloud represents a terrain that is made of homogeneous material, Figure 3.1 demonstrates popular scenarios during the robot navigation on uneven terrain, the top subfigure demonstrates the best-case scenario when a robot traverses a flat plane. The latter subfigures demonstrate cases when there is a risk of losing stability or structural integrity of the vehicle during the traversal of the rubbled terrain or going up a hill.

Navigation on point clouds has been researched in the past. Particularly, Liu and Siegwart [39] proposed a Tensor Voting Framework-based metric computation on a raw point cloud that enables graph search algorithms to be utilized on the graphs generated. Krusi et al. [32] devised a traversability estimation by determining a normal vector to a point cloud patch using PCA, followed by computing the maximum signed distance variance of the point from the fitted plane. We can move on and specify the problem definition of traversability inspection. The body of work on uneven terrain motion planning models the terrain interaction as a deterministic quantity, rather than a probabilistic quantity [49]. Depending on the texture of the uneven terrain, we can observe wheel slip, i.e., a wheel turns faster than it is possible on the type of terrain, subsequently, wheel slip can cause the loss of controllability of the robot [65]. The aspect of terramechanics should be modeled and foreseen in the planning phase for better results; however, we leave it for future work.

3.1 Problem definition and preliminaries

Let us first discuss the robot pose description, followed by a representation of the robot environment, and finally conclude with the problem definition of assessing WMR terrain traversability.

WMR is modeled as a rigid body in 3-D space, and thus the robot configuration space has 6 dimensions. Three dimensions come from the translational part (x, y, z), and the latter three from the rotational component about each of the axes. The 3-D rigid body transformation without deformation is captured by *special Euclidean group* SE(3).

Definition 3.1.1. The **Special Orthogonal Group** SO(3) is the set of 3×3 real matrices, also known as rotation matrices, such that $\mathbf{R}^{\mathsf{T}}\mathbf{R} = \mathbf{I}$ and $\det(\mathbf{R}) = 1$.

Definition 3.1.2. The **Special Euclidean group** SE(3) is the set of transformation matrices that describes the reference frame B relative to the coordinate frame A,

$$\mathbf{T}_{AB} := \begin{bmatrix} \mathbf{R}_{AB} & \mathbf{t}_{AB} \\ 0 & 1 \end{bmatrix} \in \mathrm{SE}(3), \tag{3.1}$$

where $\mathbf{t}_{AB} \in \mathbb{R}^3$ is a translation column vector, and $\mathbf{R}_{AB} := [\mathbf{x}_{AB}, \mathbf{y}_{AB}, \mathbf{z}_{AB}] \in SO(3)$ is a rotation matrix. The column vector $\mathbf{x}_{AB}$ is the orientation of x-axis of frame B in terms of frame A. Similarly, with column vectors $\mathbf{y}_{AB}$ and $\mathbf{z}_{AB}$ and y and z are the axes of frame B.

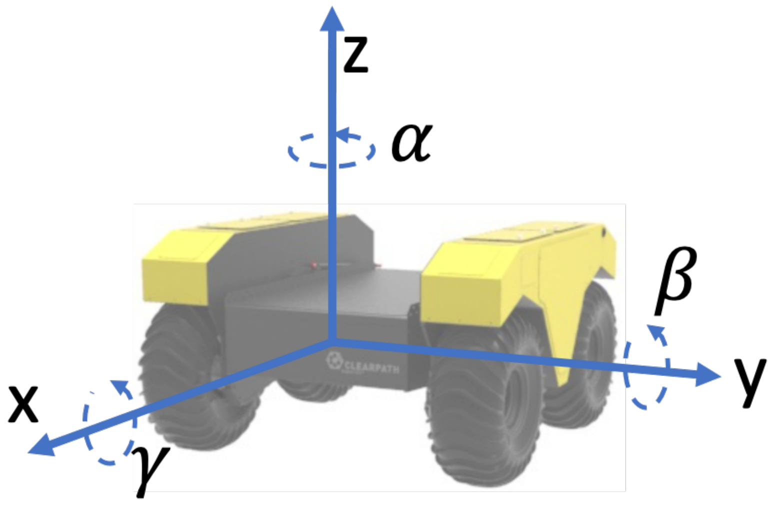


Figure 3.2: Configuration space of the robot.

The orientation of the rigid body might be accomplished with consecutive rotations of the object about its axes. Although the orientation of the object is unique, the sequence of rotations to obtain it is not unique. In this paper, we will stick to the roll-pitch-yaw convention. Rotation angles about z-axis, y-axis, and x-axis are yaw α, pitch β, and roll γ angles, respectively. The upper and lower limits of these angles are $\alpha, \beta, \gamma \in [-\pi, \pi)$. See figure 3.2. Appendix A provides a refresher on 3-D rigid body transformation.

A **robot pose** will be denoted as $\mathbf{T}_{MR}$, where R is the frame of reference fixed to the robot and M is the coordinate frame of a map. The robot footprint is the region under the robot, i.e., the approximate projected bounding box.

In Chapter 2, we reviewed different types of robot world representation. Let us consider the case for a point cloud map representation.

Definition 3.1.3 (Planning map)**.** A *point cloud* is a set of points in Cartesian 3-D space. Let us denote a point cloud as $P = \{p_i \in \mathbb{R}^3 \mid i = 1, 2, \ldots, n\}, n \in \mathbb{Z}^+$. However, a point p_i is often augmented with color, class (e.g., material), or confidence level. Let us say that all possible point clouds belong to a set of point clouds $\mathcal{P}$. Finally, a *planning map* is a point cloud with a sufficient resolution where a robot-sized terrain patch is not represented by a single point.

To assess the terrain regions for an autonomous mobile robot, let us adopt the notation and problem definition presented in Krüsi et al. [32], where *terrain inspection* is a function

$$f_{ti} : (P, \mathbf{T}_{MR}) \mapsto (\mathbf{T}_{M\tilde{R}}, \tau), \tag{3.2}$$

where $P \in \mathcal{P}$ is a point cloud planning map, $\mathbf{T}_{MR} \in \mathrm{SE}(3)$ is an assessment pose. Two outputs of the function are $\mathbf{T}_{M\tilde{R}} \in \mathrm{SE}(3)$ is the anticipated robot pose on the point cloud patch surface, and τ is the likelihood of traversability, with $\tau = 0$ meaning the terrain is untraversable (an obstacle), and $\tau \in (0, 1]$ is a traversable region for the robot. Closely following Krüsi [32], the traversability estimation function can be decomposed into two distinct functions, which compute the anticipated robot pose $\mathbf{T}_{M\tilde{R}}$ and the traversability τ of the point cloud patch.

$$f_{ti}^{p} : (P, \mathbf{T}_{MR}) \mapsto \mathbf{T}_{M\tilde{R}} \tag{3.3}$$

$$f_{ti}^{\tau} : (P, \mathbf{T}_{M\tilde{R}}) \mapsto \tau \tag{3.4}$$

Pose anticipation function f_{ti}^{p} requires robot dimensions (height, width, length) to determine the attainable robot pose. To generate a traversability index τ, we must consider the constraints of the mechanical design of the robot. For a successful region traversal, we need to account for maximum and minimum pitch and roll angles to prevent roll or tip-over.

Within the proposed framework that will be discussed later in the following chapters, one will need to evaluate the traversability of points in the environment. Hence, the traversability cost function will be invoked a lot. In [32], the traversability cost is evaluated on raw point cloud data in each stage of their path planning framework. The main disadvantage of their method is that they compute the normal vector to the point cloud patch using PCA and discard valuable information about the spread of the data. Lalonde et al. [35] use eigenvalues to infer the shape of the point cloud patch, the relation of eigenvalues indicates if we have a plane, sphere, or tube. Employing this, we can easily filter out obstacles and untraversable terrain by examining the eigenvalues of the terrain patch.

3.2 Methodology

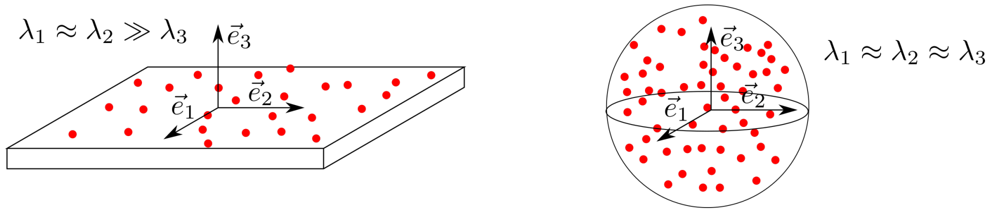
The following section closely follows the work of Krüsi [32]. The main contribution is the introduction of a hierarchy of traversability tests that range from simple to thorough and enable a faster generation of motion plans.

Let us reiterate the assumptions I take into account to devise a terrain inspection function. Firstly, I only consider the local geometry of the point cloud. In the second assumption, I assume that the point cloud chunk we are assessing is made of a material that allows us to have sufficient traction on the terrain. The third assumption is the robot has no articulated joints between the wheel axes and little to no suspension system. Then the problem of placing a robot on a point cloud boils down to fitting a plane on top of the point cloud chunk. The last assumption is that the input planning map is sufficiently fine. In other words, a region where all points are within the radius of the robot bounding circle of the point cloud can support the robot.

To determine a chunk of a point cloud that can support the robot, we will need a notion of neighborhoods of point $p \in P$. Let us consider a *ball* point cloud neighborhood.

Definition 3.2.1 (Neighborhood ball)**.** Let $p \in P$ and $r > 0$, then a point cloud **neighborhood ball** is $\mathcal{B}_{r,P}(p) = \{v : v \in P, \|p - v\| \leq r\}, r > 0$.

In this work, the radius r of the neighborhood ball is the radius of a bounding sphere of the robot. A bounding sphere is a sphere of minimum radius r that encloses the robot. The analysis of the patch will indicate if the point cloud region can accommodate the WMR, i.e., the attainable robot pose will be estimated and the presence of obstacles will be detected. The first step is to estimate the possible robot pose on the point cloud patch. Thus, similarly to [32], we need to fit a plane on robot-sized point cloud data $\mathcal{B}_{r,P}$. Hoppe in [19] treats a point cloud as a set of data points and utilizes Principal Component Analysis (PCA) [50] to inspect the data spread and estimate the normal vector to a plane. To perform PCA, we need to compute two statistical measures of the point cloud patch: a centroid of N points $p_i \in \mathcal{B}_{r,P}(\mathbf{t}_{MR})$, and their covariance matrix via the outer product,



$\lambda_1 \approx \lambda_2 \gg \lambda_3$
$\vec{e}_3$
$\vec{e}_2$
$\vec{e}_1$
$\vec{e}_3$
$\vec{e}_2$
$\vec{e}_1$
$\lambda_1 \approx \lambda_2 \approx \lambda_3$

Figure 3.3: Eigenvalues hint at the spread of the data point. In (a), we can observe a plane, while $\lambda_1 \approx \lambda_2 \gg \lambda_3$. In case (b) all eigenvalues are approximately the same number; hence we are dealing with a spherical surface.

formulated as

$$\bar{p} = \frac{1}{N} \sum_{p_i \in \mathcal{B}_{r,P}(\mathbf{t}_{MR})} p_i, \tag{3.5}$$

$$\mathbf{S} = \frac{1}{N} \sum_{p_i \in \mathcal{B}_{r,P}(\mathbf{t}_{MR})} (p_i - \bar{p})(p_i - \bar{p})^T. \tag{3.6}$$

Next, we perform eigenvalue decomposition of the covariance matrix [26]. The result of the decomposition is unit-length eigenvectors $\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3$, and their corresponding eigenvalues sorted as $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$. Often the smallest eigenvalue λ_3 and the corresponding eigenvector $\mathbf{e}_3$ are used to estimate the normal vector to the fitted plane. Yet, the smallest eigenvalue will not necessarily indicate the normal to the point cloud region, for example, there is no unique normal to a spherical object, and this object should be treated as an obstacle, i.e., a non-traversable region.

The eigenvalues of $\mathbf{S}$ are suitable for filtering out extreme cases of non-traversable regions using the notion of *planarity* [1]

$$P_\lambda = \frac{\lambda_2 - \lambda_3}{\lambda_1} \tag{3.7}$$

which ranges from 0 to 1, where 0 means the point cloud patch is not planar and 1 means a point cloud patch is a plane with no thickness. In the case of $P_\lambda < P_{\lambda,\min}$ the underlying point cloud patch is not planar; hence it is possible to label it as an untraversable region. For $P_\lambda < P_{\lambda,\min}$, it is acceptable to proceed with the orientation estimation of

Algorithm 1 $f_{ti}^{\rho}(P, \mathbf{T}_{M\tilde{R}}) \mapsto \rho$

```
 1: Compute $\bar{p}$ as in Eq. 3.5
 2: Compute $\mathbf{S}$ as in Eq. 3.6
 3: Perform eigenvalue decomposition on $\mathbf{S}$ with $e_1, e_2, e_3$ and $\lambda_1 \geq \lambda_2 \geq \lambda_3 \geq 0$
 4: Compute planarity feature $P_\lambda$ as in Eq. 3.7
 5: if $P_\lambda < P_{\lambda,min}$ or $\lambda_3 > \rho_{\max}$ then
 6:   return 0
 7: else
 8:   for all $p_i \in \mathcal{B}_{r,P}$ do
 9:     $d_i = (\bar{p} - p_i) \cdot \mathbf{e_3}$
10:     $[\min_d, \max_d] = \mathbf{minmax}(\min_d, \max_d, d_i)$
11:   if $|\max_d - \min_d| > \rho_{\max}$ then
12:     return 0
```

the robot. Since the planarity is a relative measure, there is a need to use the λ_3 (the smallest eigenvalue, also variance, i.e., the spread of points) to figure out the maximum step present in the point cloud patch. If $\lambda_3 > \rho_{\max}$ then we can conclude that the region is not traversable, where $\rho_{\max}$ is the maximum step that the WMR can traverse. If the preliminary traversability tests fail, we have to analyze the point cloud patch in more detail.

Having detected that the terrain patch contains a planar portion, we can resume analysis of the terrain patch for untraversable obstacle detection. Similarly to Krusi et al. [32] we find the largest distance (d_i) from a fitted plane to a point in the point cloud patch. We will keep track of the minimum and maximum signed distances using **minmax**, that is, the function outputs the smallest, biggest values out of the three input values. The absolute distance between the minimum and maximum values is a maximum step in the point cloud patch that will be referred to as the roughness of the terrain patch. The pictorial representation of the process is on Fig 3.4. The algorithm of the proposed approach, please refer to Algorithm 1 for details.

Closely following [32], by estimating the orientation of the robot on the point cloud patch. Here is the procedure for robot pose estimation f_{ti}^{p} for a given assessment pose $\mathbf{T}_{MR}$. The normal vector $\mathbf{e}_3$ is collinear with the z-axis of the estimated robot pose, yet these vectors can point in opposite directions. The estimated configuration will inherit the

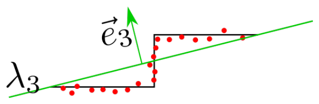


Figure 3.4: Roughness computation. The red points are the point cloud we obtain with the mapping, and the black lines represent the geometry of the environment. The green line is the computed plane from the point set. $\vec{e}_3$ is the normal to the computed plane. λ_3 is the smallest eigenvalue of PCA.

direction of the normal from the query pose,

$$\mathbf{n} = \mathbf{z}_{M\tilde{R}} = \operatorname{sgn}(\mathbf{x}_{MR} \cdot \mathbf{e}_3)\mathbf{e}_3. \tag{3.8}$$

To project x and y axes onto the new plane, we use the following equation,

$$\mathbf{x}_{M\tilde{R}} = \frac{\mathbf{n} \times \mathbf{y}_{MR}}{||\mathbf{n} \times \mathbf{y}_{MR}||} \tag{3.9}$$

$$\mathbf{y}_{M\tilde{R}} = \mathbf{n} \times \mathbf{x}_{M\tilde{R}}. \tag{3.10}$$

As a result, the rotation matrix $\mathbf{R}_{M\tilde{R}} = [\mathbf{x}_{M\tilde{R}}, \mathbf{y}_{M\tilde{R}}, \mathbf{z}_{M\tilde{R}}]$. The final step of pose estimation is projection of the translational component of the query configuration onto the fitted plane,

$$\mathbf{t}_{M\tilde{R}} = \mathbf{t}_{MR} + \frac{(\bar{p} - \mathbf{t}_{MR}) \cdot \mathbf{n}}{\mathbf{z}_{MR} \cdot \mathbf{n}} \mathbf{z}_{MR}. \tag{3.11}$$

Rotation matrix is used to compute the roll α and pitch β angles for a query yaw γ,

$$\beta = \operatorname{asin}(-r_{31}) \tag{3.12}$$

$$\alpha = \operatorname{acos}\left(\frac{r_{32}}{\cos\beta}\right), \tag{3.13}$$

where r_{31}, r_{32} are first and second elements of the third row of the matrix $R_{M\tilde{R}}$. These two values are constrained due to the robot static stability on the inclined plane, or formally $\alpha \leq \alpha_{max}$ and $\beta_{min} \leq \beta \leq \beta_{max}$.

Finally, here is the traversability estimation function which is a linear combination of terms,

$$f_{ti}^{\tau}(P, \mathbf{T}_{M\tilde{R}}) = \tau := 1 - w_{rough}\frac{f_{\rho}(P, \mathbf{T}_{M\tilde{R}})}{\rho_{\max}} + w_{roll}\frac{|\alpha|}{\alpha_{\max}} + w_{pitch}\max(\frac{\beta}{\beta_{\min}}, \frac{\beta}{\beta_{\max}}) \in [0, 1],$$

where $w_{rough}, w_{roll}, w_{pitch}$ are the weights of each of the components of the traversability function. The set of weights is established empirically, yet future work will include the procedure for determining the optimal values. In this section, the pose estimation and the terrain inspection method were outlined. The expectation for the new terrain traversability cost function is a faster function that will result in a faster path planning framework for the ground vehicle on the uneven terrain.

3.3 Experiments

In this section, we cover a set of computational benchmark studies for the studied path planning problems. In the experiment, we compare two versions of traversability cost; the traversability cost function from [32] and the proposed method. The experiment was per-

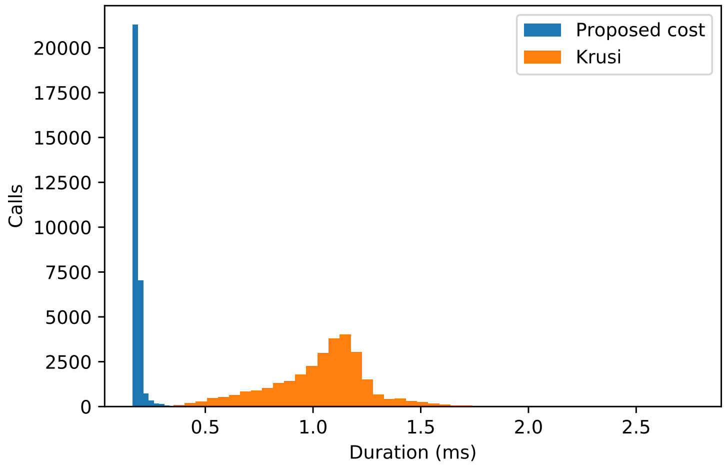


Figure 3.5: Computation time for traversability computation

formed on Clearpath Husky, a small-scale research platform with no suspension and skid-steering drive. This robot can be bounded by a sphere with radius of 0.5 m. To compare these two terrain traversability functions, computation time benchmarks were performed. In this comparison of the terrain traversability computation, these two algorithms were executed for 30000 iterations. The results of the benchmark simulation experiments are shown in Fig. 3.5. In these experiments, the proposed cost has a faster response, which will directly influence the path planning framework execution time.

The mean execution time of the proposed cost function and the original cost function are 0.20 ms and 1.04 ms with standard deviation of 0.033 ms and 0.23 ms, respectively. Thus, the improvement is at least 5 times with a lesser spread of the time execution for the proposed terrain inspection function.

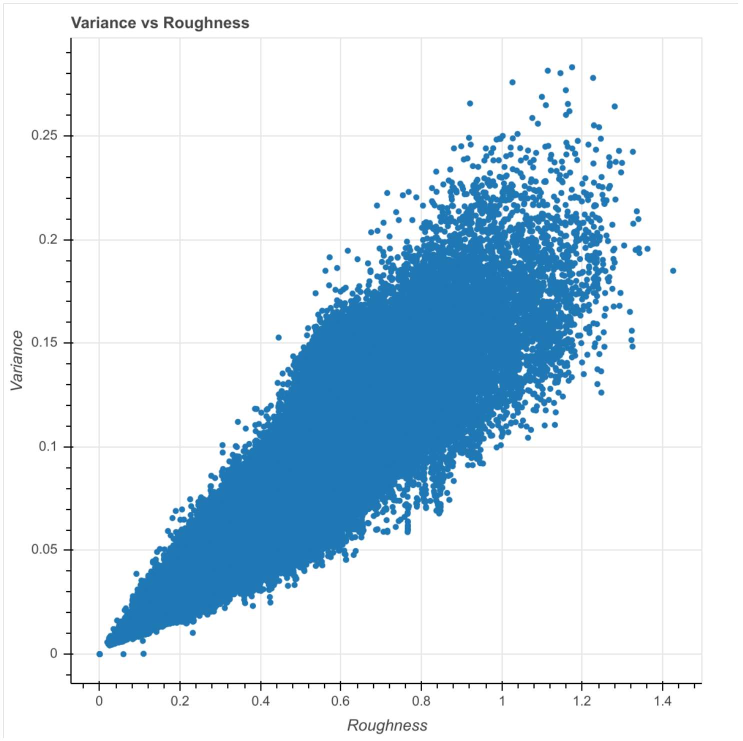


Figure 3.6: The linear relation of variance and the maximum step (roughness) in the terrain patch.

For further inspection of the new traversability estimation technique, we can refer to

Fig 3.6. The figure demonstrates the scatter plot of point set variance and the maximum step determined by the traversability cost function. We can claim that there is a linear relationship between these two variables with a Pearson correlation coefficient $r = 0.9979$. Variance in this case is the square root of the smallest eigenvalue, and it indicates the spread of data relative to the collinear normal vector. The main observation is that there is a linear relation between variance and maximum step determined by the elaborate procedures proposed in [32]. This justifies the utilization of the threshold based only on variance. The figure demonstrates the result of running the original traversability inspection for 500000 different points on the map. One can take a look at the map used in the experiment in Fig. 3.7. The map contains a gravel road and obstacles such as sandpits, hills, bushes, fences, piles of bricks, and other types of uneven terrain.

To demonstrate the efficacy of the method, the traversability estimation function was utilized in the path planning framework. The traversability estimation function was used as a binary obstacle state validity checker for the RRT* algorithm. We recorded the execution time of the path planning algorithm for these different traversability functions. The algorithm was executed for 20000 iterations on the same map. As a result, the proposed method has an average execution time of 17.8 s and the previous traversability estimation function on average works for 18.1 s. The results of these two methods are depicted in Fig. 3.8.

3.4 Conclusion

A novel simplified point cloud terrain inspection function was proposed. The method is based on the traversability inspection function presented by Krusi in [32]. The traversability method utilized by Krusi has to perform additional calculations of the distance for each of the points on the inspected area. However, since we have to perform PCA decomposition we obtain the direction-wise variance of data of the point cloud patch; hence we can eliminate cases of untraversable terrain regions. The benchmark experiments demonstrated that the proposed method was at least 5 times faster than the method introduced by Krusi in [32]. The future work includes providing analytical proof that our traversability function is

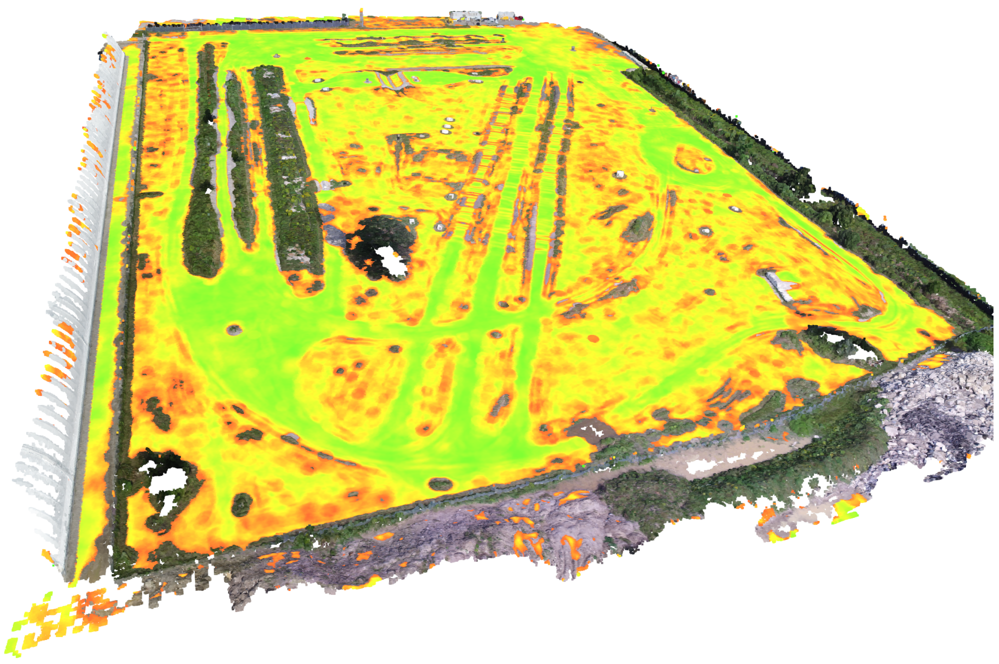
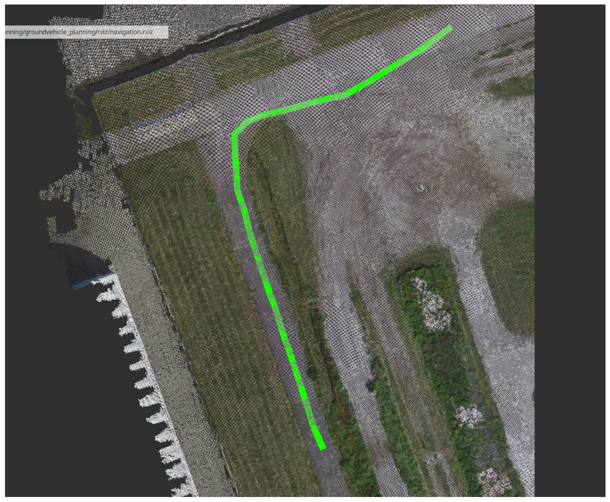
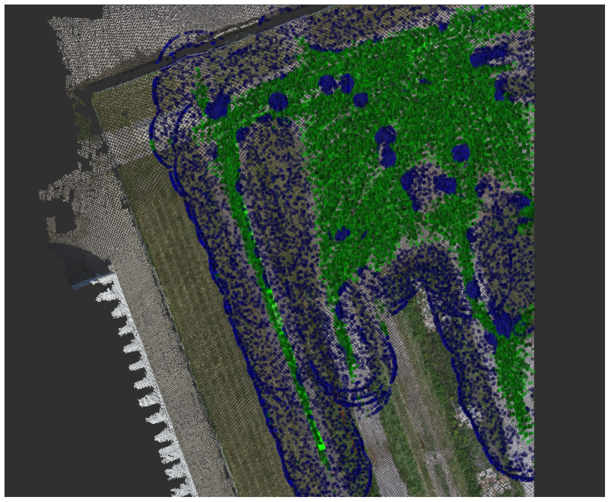


Figure 3.7: Green to yellow denotes candidates for traversable regions, using only the variance metric to decide the traversability of the point cloud patch.

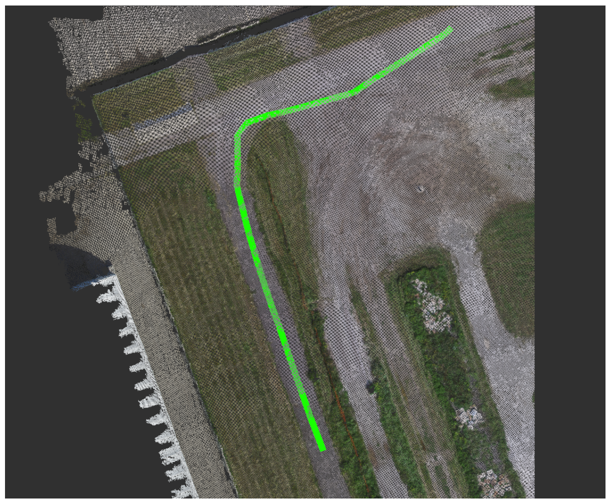
a suitable metric for terrain inspection. The hand-crafted traversability function is a difficult function to come up with. This issue might be addressed with machine learning-based approaches for the traversability estimation.



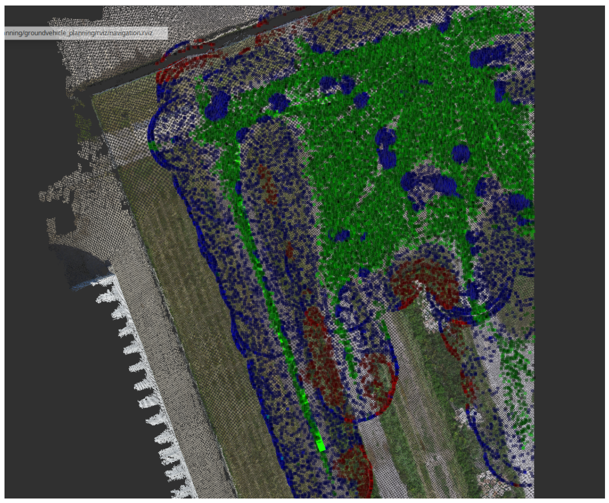
(a)



(b)



(c)



(d)

Figure 3.8: Path planning solution for the ground vehicle based on the original traversability estimation and proposed modification. (b) and (d) are point clouds of the examined points, where green points are the traversable regions, blue ones are the detected untraversable region via the original method and the red points are the early untraversable region detections.

Chapter 4

Multistage Path Planning on Uneven Terrain

Uneven terrain path planning is an essential problem for search and rescue, surveillance, military, humanitarian, agricultural, constructing missions, etc [25, 59, 6, 10]. These missions necessitate the generation of a feasible sequence of waypoints, or reference states, to navigate a wheeled mobile robot from the initial location to the final target location through the uneven terrain. The feasibility of navigating through a given path over uneven terrain can be undermined by various terrain features. Recall from Chapter 2 that rough/uneven terrain is an unstructured terrain that can immobilize or entrap the robot. The robot can lose control due to a geometrical arrangement of terrain or due to a texture of a terrain. Historically, research in path planning for mobile robots considered flat surfaces in structured environments with static obstacles. However, unlike path planning on flat terrain, the objective of rough terrain path planning includes avoiding obstacles and still being able to complete the mission considering constraints on motion caused by the geometry or texture of the terrain. Real-world mobile robots usually cannot turn on the spot; they move along continuous curves. For flat terrain, we can generate an obstacle-free continuous curve that represents the feasible path. However, if we try to execute it on an actual, possibly rough, terrain the autonomous vehicle can roll over and damage the user and itself. Hence, the path planning methods that assume flat terrain are not applicable

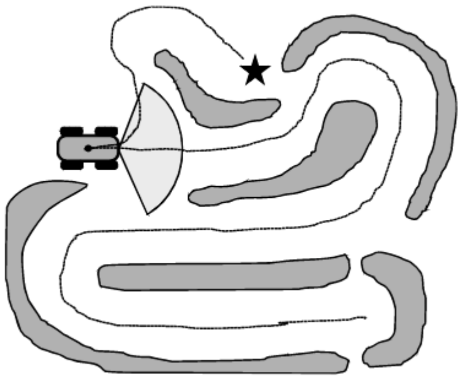


Figure 4.1: Short-horizon look ahead strategy is suboptimal [30, p. 642]. The graph depicts the case where short-horizon planning can result in a disastrous result of getting stuck. The left turn would be a better option, compared to a strategy that minimizes the distance between the robot and the goal position.

as is. To quantify the features of rough terrain, in Chapter 3, we were accustomed to the traversability estimation function that differentiates safe and unsafe terrain regions. Uneven terrain navigation requires us to forecast the behavior of an underlying terrain, unlike navigating a smooth asphalt in the case of autonomous parking.

The motivation for this research is a lack of a planning framework for robust motion planning on uneven terrain. A framework for motion planning on uneven terrain, introduced by Iagnemma and Dubowsky in [22], utilizes a two-step planning framework. Their framework utilizes a two-step algorithm that gradually increases the analysis of the terrain traversability. Their framework was designed to generate plans for a short horizon restricted by a sensing device (a range finder). Greedy motion planning in receding horizon fashion is likely to result in unreasonable behavior. Refer to Fig. 4.1. Howard and Kelly in [21] introduced a generic approach for the trajectory generation that can accomplish motion planning on rough terrain. According to Howard and Kelly[21] in most of the work, research groups rely on feedback controllers to account for the wheel slip. Their approach utilizes the optimal control formulation for the trajectory generation with the performance index that accounts for the traversability aspects of the path. In their framework, Howard and Kelly rely on an elevation map and exact obstacle locations. These require an additional step that generates the map from the sensory inputs. The conversion from sensory inputs to elevation map is not a computationally cheap operation [4]. The navigation framework presented by Krusi et al. [32] addresses the issue of deficiency of elevation map representation via point clouds. In their framework, the point cloud is a by-product of the SLAM process. As was mentioned in Chapter 2, point clouds can represent a wider variety of topologies compared to elevation maps, and polygonal meshes. In their framework, Krusi et al. used a three-stage approach for motion planning. The first two stages utilize sampling-based planners for the fast discovery of a feasible solution, followed by the optimization of this feasible path. The motion segments at these two stages were approximated with cubic spline trajectory, which was described in [31]. Due to the stochastic nature of sampling-based planners and the limited computational power of their onboard computer, their path contained sporadic unnecessary turning actions. The third stage resolves this problem using the gradient-free smoothing approach, yet the actual system dynamics of the vehicle are not considered.

To move forward, we will need to talk about two types of motion planning problems: geometric and kinodynamic planning. Geometric path planning, also known as the piano mover's problem, considers the generation of an obstacle-free path disregarding the kinematics/dynamics of the robot of the resultant motion plan. Kinodynamic planning can be considered as an extension of the geometric path planning problem that accounts for the system dynamics of the robot.

Structured environments provide abundant information that enables simplifications of the autonomous driving problem. A survey paper by Paden et al. in [47] establishes that the hierarchical approach is prevalent for the planning component of a navigation system in structured environments. This structure consists of four layers: Route Planning, Behavior planning, Motion planning, and Feedback control. Having this in mind, we can draw inspiration from the methods used in autonomous driving for enabling uneven terrain motion planning.

It was determined in Chapter 2 that sampling-based planners provide the generic formulation and an acceptable solution to the path planning problem. The power of sampling-based planners comes from the generic problem formulation, which does not require an explicit representation of obstacles and constraints from the perspective of a robot, enabling their applicability virtually to any path planning problem in robotics. This led to a natural choice of the sampling-based planner as a backbone of the uneven terrain motion planner. However, the sampling-based approach has its disadvantages. One disadvantage is the long computation times for certain cases. For example, an attempt to solve the problem in one shot can result in a subpar solution, induced by the complexity of the path planning problem. The majority of the previous work on uneven terrain path planning simplifies the environment to be able to generate motion plans using given computational resources (running time, memory).

Solving one comprehensive problem with differential constraints and terrain traversability aspects is a computationally complex problem to tackle. To address the differential constraints of the motion planning problem, we must solve two-point boundary value problems (BVPs)[17]. Conventional approaches for solving BVP cannot incorporate obstacles. Another aspect is the availability of an efficient traversability estimation procedure. One approach is to break down the problem into several manageable problems with a lesser detail and gradually improve the solution. A hierarchical approach is a kind of approach that iteratively improves the path solution quality at each stage of a hierarchy, and uneven terrain motion planning can leverage this kind of approach.

The first stage ensures that a robot feasibly is placed in the configuration space of the robot (i.e., the attempt to traverse the terrain patch will not result in a collision with the environment), and the second stage will ensure that the physical system can execute the chain of states.

The chapter proposes a practical approach for motion planning on uneven terrain. Prior work on uneven terrain path planning [32, 22, 21] has addressed the problem on distances no more than 100 m. Our work differs from the above-mentioned methods in the following way. We propose a method to solve the path planning problems on the scale of 200-500 m. To address the shortcomings of the previous approaches, we propose an algorithm that consists of two stages. The first stage computes an admissible path, considering terrain traversability and simplified system dynamics of our autonomous robot. The second stage optimizes the path length of the admissible solution. The second stage also examines the traversability in more detail. The terrain representation is an important aspect of path planning. A point cloud is a suitable representation of uneven terrain. We leverage the point cloud representation using our traversability function proposed in Chapter 3.

4.1 Problem formulation

The chapter will concentrate on the generation of an admissible motion plan (path) for a wheeled mobile robot on uneven terrain. An admissible path has to be generated from an initial 3D pose to a goal region, respecting the constraints imposed by the map and capabilities of the robot. In our case, the environment map available to the robot is in the form of a point cloud (set of points). The robot's capabilities are in the form of kinematic, dynamic, and holonomic constraints of the WMR.

Definition 4.1.1 (*Admissible/Feasible* Motion planning)**.** The motion planning problem would be to find a sequence of states such that our robot $\mathcal{A}$ does not collide with any obstacles $\mathcal{O} \subset \mathcal{W}$. The state of the robot $\mathcal{A}$ is referred to as a configuration q. The set of all robot configurations makes up the *configuration space* Q, also known as $\mathcal{C}$-space. It is worth noting that workspace $\mathcal{W}$ is not the same as configuration space Q. In general, mapping $\mathcal{C}$-space to a workspace is a tedious task. Let $Q_{invalid} \subset Q$ be the set of all invalid configurations of the robot in the environment. An example of an invalid configuration state is the case when a robot penetrates the ground or an obstacle. Consequently, we denote the set of all valid configurations by $Q_{valid} = Q \setminus Q_{invalid}$. A basic motion planning problem generates a path $\pi : [0, 1] \rightarrow Q_{valid}$, such that $\pi(0) = q_{start}$ and $\pi(1) \in Q_{goal}$.

Definition 4.1.2 (Optimal motion planning)**.** Optimal motion planning extends the motion planning problem with a cost function $c : \Pi \rightarrow \mathbb{R}_+$, where Π is a set of all possible paths. Therefore, the optimal motion planning problem is to find such a path π^*:

$$\begin{aligned} \pi^* = \quad & \underset{\pi}{\arg\min} && c(\pi) \\ & \text{s.t.} && \pi(0) = q_{start}, \\ &&& \pi(1) \in Q_{goal} \end{aligned} \tag{4.1}$$

The cost function is well-behaved, meaning that the cost of the subsections cannot be higher than the whole path cost.

Let's move on and revise the configuration space for the rigid body in the 3-D space.

Definition 4.1.3 (Pose representation)**.** WMR is a rigid body in 3-D, its $SE(3)$ pose is represented by $\mathbf{T}_{AB} = (\mathbf{R}_{AB}, \mathbf{t}_{AB})$, where $\mathbf{T}_{AB} = \begin{bmatrix} \mathbf{R}_{AB} & \mathbf{t}_{AB} \\ \mathbf{0} & 1 \end{bmatrix} \in \mathbb{R}^{4\times4}, \mathbf{R}_{AB} \in \mathbb{R}^{3\times3}, \mathbf{t}_{AB} \in \mathbb{R}^{3\times1}$. For details, please refer to Appendix A.

Definition 4.1.4. Trajectory $\sigma : [0, T] \to Q_{valid}$, $\sigma(0) = q_0, \sigma(T) \in Q_{goal}$, T is the final time. A trajectory $\sigma \in \Sigma$, i.e., an element of a set of all possible trajectories, termed Σ. The study will address the trajectory generation since path π, is just a trajectory that was limited to a 0,1 period.

Definition 4.1.5 (Differential constraints)**.** Differential constraints are constraints on derivative terms of the system dynamics.

Definition 4.1.6 (Motion planning on Uneven Terrain)**.** Let us give the formal definition of the motion planning problem on uneven terrain. The WMR operates in the 3-D environment represented by a point cloud $P_M \in \mathcal{P}$, where $P_M := \{p_i \in \mathbb{R}^3 \mid i = 1, 2, \ldots, n\}, n \in \mathbb{Z}_+$. The study only considers the point cloud maps that are sufficiently dense relative to the robot size.

The robot resides at a known state $\mathbf{T}_{MS} \in SE(3)$ and the objective is to compute a trajectory $\sigma : [0, T] \to SE(3)$ should be generated from the start robot pose to the goal position $\mathbf{T}_{MG} \in SE(3)$ respecting following criteria:

1. No obstacle collision of the robot with the environment.
2. The robot should maintain constant contact with the terrain.
3. Non-holonomic constraints are respected by the planning framework.
4. The output solution should consider constraints on kinematics and dynamics of the WMR.
5. The planning framework optimizes a provided adjustable cost function $c : \Sigma \to \mathbb{R}_+$ to a local minimum.

4.1.1 Framework

Inspired by [32], our framework for uneven terrain path planning comprises three stages. The first stage discovers the initial feasible solution. The second stage improves the solution given this feasible path. Lastly, we smooth the resultant path and reduce the number of waypoints that represent the final path (see Fig. 4.2). The second and third stages of planning share the same objective function. Uneven terrain path planning has to consider the following aspects: the constraint on the orientation of the WMR, the presence of obstacles, and path length. These aspects will be described in the following subsections. All stages of the path planning framework use *Dubins car* [9] model to connect two points.

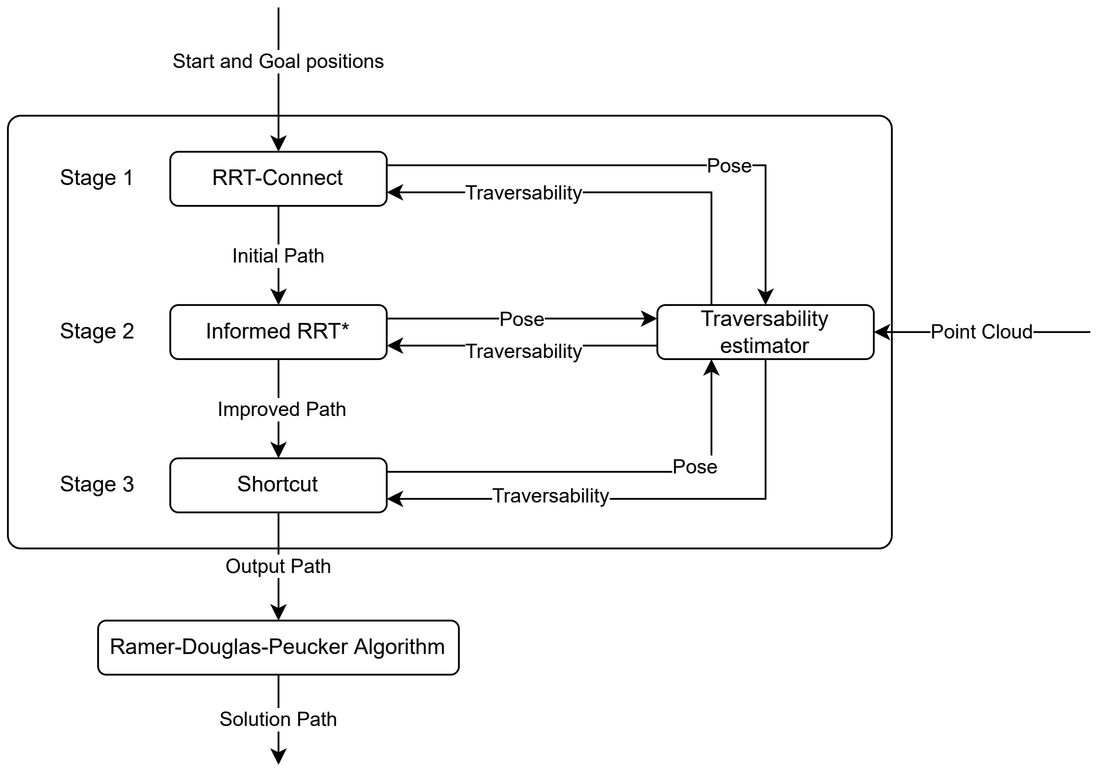


Figure 4.2: Block diagram of the proposed path planning framework.

The path planning framework consists of three stages. The first step uses the RRT-

Connect algorithm [33]. It is an efficient heuristic algorithm that rapidly discovers a feasible solution. Here is the short walk-through of the algorithm. RRT-Connect extends the basic Rapidly-Exploring Random Tree [36] planning algorithm by the utilization of two separate trees that stem from start and goal states. At each iteration of the algorithm, we switch between the start and goal trees and refer to the active and inactive ones as major and minor trees. Then a new sampled state is checked if it can be added to the active tree. If so, we proceed with the connect stage of the algorithm, in which we greedily grow the inactive tree to the valid node of the active tree. Eventually, both of the trees overlap and we have a solution. This solution is passed down to the next stage of the path planner. The detailed description of the RRT-Connect algorithm is in the Appendix B with pseudocodes in Algorithms 2, 3, 4.

The second stage of the path planning framework utilizes Informed RRT* [12] a variant of RRT* [27] to generate an optimized version of the initial feasible path. RRT* is an asymptotically optimal motion planning algorithm. It means that, if we allow the algorithm to work for an infinite amount of time, it will surely find the global optimal solution if one exists. Therefore, RRT* was chosen to improve the quality of the initial feasible path. Briefly, regular RRT* works in the following way. The tree is initialized with a start pose as a root node. Then the following steps are executed until a termination condition. For example, the number of allowed iterations is the termination condition. As with regular RRT, a random state is sampled. The next step is to find the nearest tree node to the random state. Then the steering function determines the new node that is sufficiently close to the tree to be added. Afterward, RRT* determines the set of nearest neighbors of this new node within a certain radius. In the next step of the algorithm, we should iterate through the nearest-neighbor set and determine the best parent for the new node. After the best parent was determined, RRT* iterates the set of nearest neighbors and reconnects these nodes in case the total cost of the path from the root node to the nearest node is better through the new node. Please refer to detailed explanation of RRT* in the Appendix B, with pseudocode listed in Algorithms 5,6,7.

In the second stage of the path planning framework, we are restricting the sampling to an ellipse around the initial path generated in the first step. The method is also known as Informed RRT*. In this case, we will spend the computational resources only on the improvement of the feasible path. RRT* optimizes the path with a provided objective function. In this thesis, the cost function includes path length, traversability measure, and change in robot heading. The total cost of the path is computed for each of the segments of the path. The cost of the segment of the path is,

$$\begin{aligned} c\left(\pi_{T_{MR_1},T_{MR_2}}\right) = \eta_1 L(\pi_{T_{MR_1},T_{MR_2}}) + \eta_2 \sum_{k=1}^{K} \big[f_{ti}^{\tau}\left(P, \pi_{T_{MR_1},T_{MR_2}}(k\Delta k)\right) \\ + \eta_3 \Delta\alpha\left(\pi_{T_{MR_1},T_{MR_2}}((k-1)\Delta k), \pi_{T_{MR_1},T_{MR_2}}(k\Delta k)\right)\big], \end{aligned} \tag{4.2}$$

where $\eta_i \in [0,1]$ and $\eta_1 + \eta_2 + \eta_3 = 1$ is an importance coefficient, L is path length between two states T_{MR_1}, T_{MR_2}, $\pi_{T_{MR_1},T_{MR_2}} : [0,1] \rightarrow SE(3)$ is a function that returns all intermediate states between two points, $\Delta k = \frac{1}{K}$ is a path resolution, and absolute value of the change in the heading of the robot $\Delta\alpha : SE(3) \times SE(3) \rightarrow \mathbb{R}^+$. The cost function is intended to penalize change in the heading angle, and navigate through most traversable regions of the path while keeping the length of the path at a minimum.

Recall that, RRT* is an asymptotically optimal planner, yet for practical considerations, we cannot afford to run the planner for an infinite amount of time. We terminate the planner after a certain number of iterations. As a result, the output path needs additional refinement. Please refer to Fig. 4.2.

Path smoothing & simplification: To smooth the path, we use the shortcut path simplification algorithm from [14]. The shortcut heuristic algorithm works in the following way. A path, consisting of a sequence of states, is randomly subdivided into two segments. If there is a valid path segment between states that omit this segment, then the segment is removed and the path is optimized. After the path was smoothed, we need to eliminate unnecessary path points on the resultant path. The resultant path is a concatenation of different fragments of the Dubins path, i.e., it consists of arcs and straight lines. These arcs and lines are represented with straight line segments of the same length. The path can be a straight line, but it will still be represented with numerous linear segments of uniform size. We would like to eliminate this redundancy. This was accomplished with Ramer–Douglas–Peucker algorithm [53, 8]. We recursively choose two endpoints on the path and look for the farthest away point from the line, if it is more than some ϵ, we say it is one of the endpoints and proceed, and we examine these two new lines.

4.2 Experiments and results

We evaluate the proposed method in three different scenarios on a map that has both structured and unstructured terrain. The computer used had the following specifications: Intel®Core™i7-8700K CPU @ 3.70GHz with 16 GB RAM at 2400MHz. The framework was implemented using ROS Noetic. PCL and Eigen libraries were used for point cloud processing and handling. Path planning algorithms were implemented on top of the Open Motion Planning Library (OMPL). Our testbed system is Clearpath Warthog. This wheeled robot is a large all-terrain vehicle that can also be bounded in a sphere of radius 1.5 m. The resulting paths were tested on a real test track. The proposed solution paths were tracked successfully by the robot vehicle. Though the robot vehicle was moving too close to the edge of the road.

A series of benchmarks were conducted to demonstrate the efficacy of the path planning framework. The mission consisted of three sections (cases). Each case was from about 100 to 500 meters. Each of the cases had different traversability features. The first case was a scenario where a robot has to drive a long stretch of road that has a few obstacles. The second case contains rough terrain on an inclined plane. The third case consists of another stretch of road with obstacles. For each of the cases, the first stage was executed until it discovers a solution, followed by 5000 iterations of Informed RRT*. The benchmark was conducted 10 times for each of the cases. Here are the metrics used in the study: success rate (%), path length (m), and computation time (s).

The success rate is the rate of successful discovery of exactly the goal state. Please refer to Table 4.1 for details. The path length and computation time are listed as mean values with standard deviation. Notice that in all the cases when the solution was discovered the path length denoted as L in the table varies less than 3%. However, the computation time varied significantly, due to the stochastic nature of sampling-based planners such as RRT-Connect and Informed RRT*.

Please also refer to Fig. 4.3. It depicts case A of benchmarks of the path generated

	case A		
run #	L (m)	t (s)	success
1	456.91	60	1
2	450.44	43	1
3	453.05	110	1
4	442.58	90	1
5	449.58	55	1
6	452.22	35	1
7	453.11	105	1
8	449.86	56	1
9	453.24	77	1
10	458.13	132	1
avg	451.91	76.3	100%
stdev	4.31	31.97	

	case B		
run #	L (m)	t (s)	success
1	189.77	63	1
2	184.89	41	1
3	186.76	83	1
4	187.76	73	1
5	187.18	58	1
6	117.00	26	0
7	186.44	75	1
8	190.34	58	1
9	191.41	120	1
10	188.51	64	1
avg	181.01	66.1	90%
stdev	22.58	25.17	

	case C		
run #	L (m)	t (s)	success
1	147.13	53	1
2	137.92	46	1
3	144.39	136	1
4	135,17	48	1
5	135.27	48	1
6	134.73	46	1
7	134.75	30	1
8	195.14	44	0
9	135.81	56	1
10	135.88	51	1
avg	143.62	55.8	90%
stdev	18.62	29.02	

Table 4.1: Results of the benchmark of the proposed path planning framework.

by the path planner framework. The yellow path is an initial path generated by RRT-Connect. The cyan path is the optimized path generated by RRT*. The red path is the smoothened version of the optimized path.

In order to demonstrate the advantage of this multistage approach, an additional experiment was conducted. The same path planning problem was solved with the proposed approach, with RRT-Connect and Informed RRT*. The proposed approach provided a relatively good compromise between execution time and solution quality. The proposed method on average took 51.7 s with a total cost of 485.8. RRT-Connect was the fastest with an average duration of 13.7 s and a total cost of 518.7. The last was Informed RRT*, with an average execution time of 221.5 s and a total cost of 490.1.

4.3 Conclusion

A multi-stage path planning framework for rough terrain navigation for wheeled robots was developed and tested. The path planning framework can generate paths on uneven terrain, taking into account geometric features of the terrain. The efficacy of the framework was demonstrated through simulation experiments and execution of the solution path on a real-world test track.

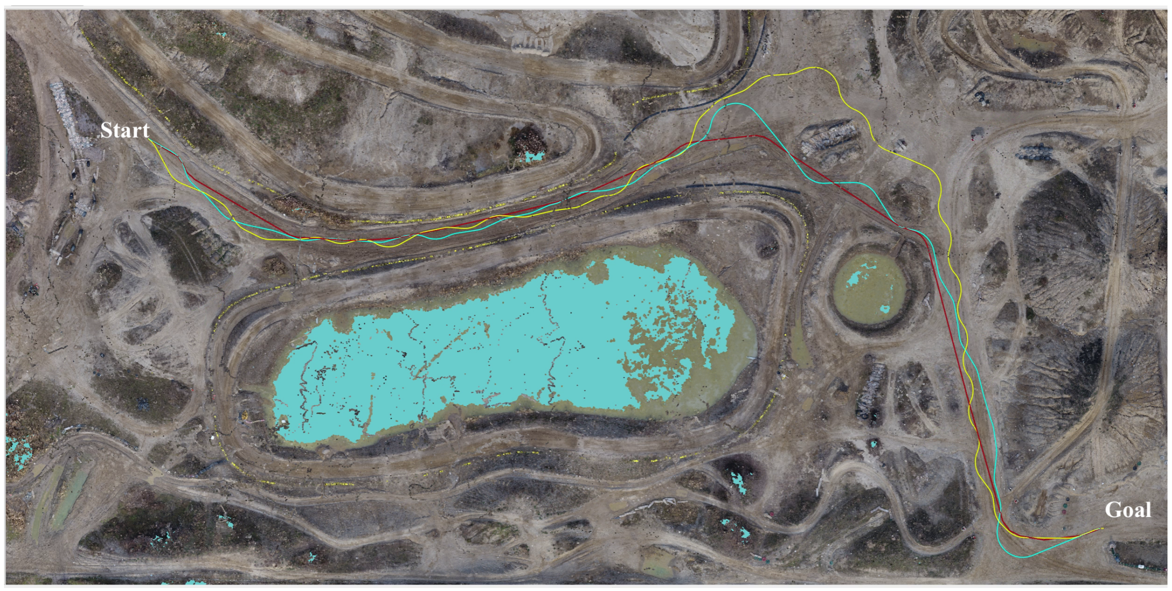


Figure 4.3: Yellow path is an initial path generated by RRT-Connect. The cyan path is the optimized path generated by RRT*. The red path is a final path generated by the Shortcut heuristic algorithm.

Chapter 5

Conclusion

This thesis has considered two important aspects of path planning on uneven terrain. Initially, I demonstrated the advantage of the accelerated traversability estimation function. Since the validity checker in our case traversability function is a frequently used function in the sampling-based path planners, it provided a speed-up in obtaining the solution path. Furthermore, in this thesis, I proposed a multi-stage path planning framework for uneven terrain. The path planning framework was tested on a test track with the industry partner. It was able to traverse the path on a real test track. The benchmark experiment comparing the framework against each of the algorithms used demonstrated the advantage of the three-stage approach. The resulting solution path length was consistent, however, the solution time was less consistent.

5.1 Future work

In future work, one could optimize the computation time for rough terrain motion planning to enable online path planning on uneven terrain. Another important consideration for uneven terrain is information from color, which should be incorporated in the following works. Objects with the same geometry and different color can pose different risks to the mobile robot. Another direction of research is the incorporation of the confidence level of the sensor readings. Depth cameras and structure-from-motion approaches are able to return the confidence level for each of the readings. These data can be employed for a probabilistic approach to the path planning problem to improve the robustness of the output paths.

References

[1] R. Blomley, M. Weinmann, J. Leitloff, and B. Jutzi. Shape distribution features for point cloud analysis – a geometric histogram approach on multiple scales. *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci.*, II-3:9–16, August 2014.

[2] M. Brunner, B. Brüggemann, and D. Schulz. Autonomously traversing obstacles - metrics for path planning of reconfigurable robots on rough terrain. In *Proceedings of the 9th International Conference on Informatics in Control, Automation and Robotics.* SciTePress - Science and and Technology Publications, 2012.

[3] M. Brunner, B. Brüggemann, and D. Schulz. Hierarchical rough terrain motion planning using an optimal sampling-based method. In *IEEE International Conference on Robotics and Automation*, pages 5539–5544. IEEE, May 2013.

[4] W. Burgard, M. Hebert, and M. Bennewitz. World modeling. In *Springer Handbook of Robotics*, pages 1135–1152. Springer International Publishing, 2016.

[5] R. O. Chavez-Garcia, J. Guzzi, L. M. Gambardella, and A. Giusti. Learning ground traversability from simulations. *IEEE Robotics and Automation Letters*, 3(3):1695–1702, July 2018.

[6] F. Colas, S. Mahesh, F. Pomerleau, M. Liu, and R. Siezwart. 3D Path Planning and Execution for Search and Rescue Ground Robots. *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 722–727, 2013.

[7] D. Dolgov, S. Thrun, M. Montemerlo, and J. Diebel. Practical search techniques in path planning for autonomous driving. In *Proceedings of the First International*

Symposium on Search Techniques in Artificial Intelligence and Robotics (STAIR-08), Chicago, USA, June 2008. AAAI.

[8] D. H. Douglas and T. K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Cartographica: The International Journal for Geographic Information and Geovisualization*, 10(2):112–122, December 1973.

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

[10] M. Eich, F. Grimminger, and F. Kirchner. A Versatile Stair-Climbing Robot for Search and Rescue Applications. *2008 IEEE International Workshop on Safety, Security and Rescue Robotics*, pages 35–40, 10 2008.

[11] A. Ettlin and H. Bleuler. Rough-Terrain robot motion planning based on obstacleness. In *2006 9th International Conference on Control, Automation, Robotics and Vision*, pages 1–6. IEEE, December 2006.

[12] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot. Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2997–3004, 2014.

[13] D. B. Gennery. Traversability analysis and path planning for a planetary rover. *Auton. Robots*, page 16, 1999.

[14] R. Geraerts and M. H. Overmars. Creating high-quality paths for motion planning. *Int. J. Rob. Res.*, 26(8):845–863, August 2007.

[15] Y. Gu, J. Strader, N. Ohi, and et al. Robot foraging: Autonomous sample return in a large outdoor environment. *IEEE Robot. Autom. Mag.*, 25(3):93–101, September 2018.

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

[17] M. T. Heath. *Scientific Computing: An Introductory Survey.* McGraw-Hill, New York, 2 edition, 2002.

[18] M. Herbert, C. Caillas, E. Krotkov, I. S. Kweon, and T. Kanade. Terrain mapping for a roving planetary explorer. In *International Conference on Robotics and Automation*, pages 997–1002 vol.2, May 1989.

[19] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, and W. Stuetzle. Surface reconstruction from unorganized points. In *Proceedings of the 19th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '92, pages 71–78, New York, NY, USA, July 1992. Association for Computing Machinery.

[20] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. *Auton. Robots*, 34(3):189–206, April 2013.

[21] T. M. Howard and A. Kelly. Optimal rough terrain trajectory generation for wheeled mobile robots. *Int. J. Rob. Res.*, 26(2):141–166, February 2007.

[22] K. Iagnemma and S. Dubowsky. *Mobile Robots in Rough Terrain: Estimation, Motion Planning, and Control with Application to Planetary Rovers.* Springer Science & Business Media, July 2004.

[23] K. Iagnemma, F. Genot, and S. Dubowsky. Rapid physics-based rough-terrain rover planning with sensor and control uncertainty. In *IEEE International Conference on Robotics and Automation*, volume 3, pages 2286–2291 vol.3, May 1999.

[24] L. Janson, E. Schmerling, A. Clark, and M. Pavone. Fast marching tree: a fast marching sampling-based method for optimal motion planning in many dimensions. *Int. J. Rob. Res.*, 34(7):883–921, June 2015.

[25] E. Jelavic, F. Farshidian, and M. Hutter. Combined Sampling and Optimization Based Planning for Legged-Wheeled Robots. *IEEE International Conference on Robotics and Automation (ICRA)*, pages 8366–8372, 2021.

[26] I. T. Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer, New York, NY, New York, NY, 1986.

[27] S. Karaman and E. Frazzoli. Sampling-based algorithms for optimal motion planning. *Int. J. Rob. Res.*, 30(7):846–894, June 2011.

[28] L. E. Kavraki, M. N. Kolountzakis, and J. Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Trans. Rob. Autom.*, 14(1):166–171, February 1998.

[29] L. E. Kavraki, P. Svestka, J. Latombe, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Trans. Rob. Autom.*, 12(4):566–580, August 1996.

[30] A. Kelly. *Mobile Robotics: Mathematics, Models, and Methods*. Cambridge University Press, November 2013.

[31] A. Kelly and B. Nagy. Reactive nonholonomic trajectory generation via parametric optimal control. *Int. J. Rob. Res.*, 22(7-8):583–601, July 2003.

[32] P. Krüsi, P. Furgale, M. Bosse, and R. Siegwart. Driving on point clouds: Motion planning, trajectory optimization, and terrain assessment in generic nonplanar environments: Driving on point clouds. *J. field robot.*, 34(5):940–984, August 2017.

[33] J. Kuffner and S. M. LaValle. RRT-connect: An efficient approach to single-query path planning. *IEEE International Conference on Robotics and Automation*, 2:995–1001, 2000.

[34] I. S. Kweon and T. Kanade. High-resolution terrain map from multiple sensor data. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2):278–292, 1992.

[35] J.-F. Lalonde, N. Vandapel, D. F. Huber, and M. Hebert. Natural terrain classification using three-dimensional ladar data for ground robot mobility. *J. field robot.*, 23(10):839–861, October 2006.

[36] S. M. Lavalle. Rapidly-Exploring random trees: A new tool for path planning. Technical report, Computer Science Dept., Iowa State Univ., 1998.

[37] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, May 2006.

[38] S. M. LaValle and J. Kuffner. Randomized kinodynamic planning. *Int. J. Rob. Res.*, 20(5):378–400, May 2001.

[39] M. Liu and R. Siegwart. Navigation on point-cloud — a riemannian metric approach. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 4088–4093. IEEE, May 2014.

[40] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, August 1987.

[41] D. V. Lu, D. Hershberger, and W. D. Smart. Layered costmaps for context-sensitive navigation. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 709–715. IEEE, September 2014.

[42] K. M. Lynch and F. C. Park. *Modern robotics: mechanics, planning, and control*. Cambridge University Press, Cambridge, UK, 2017.

[43] Y. Ma and Z. Shiller. Pose estimation of vehicles over uneven terrain. March 2019.

[44] D. Meagher. Geometric modeling using octree encoding. *Computer Graphics and Image Processing*, 19(2):129–147, 1982.

[45] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *IEEE International Conference on Robotics and Automation*, volume 2, pages 116–121, March 1985.

[46] K. Otsu, G. Matheron, S. Ghosh, O. Toupet, and M. Ono. Fast approximate clearance evaluation for rovers with articulated suspension systems. *J. field robot.*, 37(5):768–785, August 2020.

[47] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli. A survey of motion planning and control techniques for Self-Driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1(1):33–55, March 2016.

[48] P. Papadakis. Terrain traversability analysis methods for unmanned ground vehicles: A survey. *Eng. Appl. Artif. Intell.*, 26(4):1373–1385, April 2013.

[49] P. Papadakis and F. Pirri. 3D Mobility Learning and Regression of Articulated, Tracked Robotic Vehicles by Physics-based Optimization. In *Virtual Reality Interaction and Physical Simulation, Eurographics*, Darmstadt, Germany, December 2012.

[50] K. Pearson. LIII. *On* lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11):559–572, November 1901.

[51] P. Pfaff, R. Triebel, and W. Burgard. An efficient extension to elevation maps for outdoor terrain mapping and loop closing. *Int. J. Rob. Res.*, 26(2):217–230, February 2007.

[52] M. Quigley, K. Conley, B. Gerkey, and et al. ROS: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[53] U. Ramer. An iterative procedure for the polygonal approximation of plane curves. *Computer Graphics and Image Processing*, 1(3):244–256, November 1972.

[54] J. H. Reif. Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science*, pages 421–427, October 1979.

[55] G. Reina, M. Bellone, L. Spedicato, and N. I. Giannoccaro. 3D traversability awareness for rough terrain mobile robots. *Sensor Review*, 34(2):220–232, 2014.

[56] P. R. Roan, A. Burmeister, A. Rahimi, K. Holz, and D. Hooper. Real-world validation of three tipover algorithms for mobile robots. In *IEEE International Conference on Robotics and Automation*, pages 4431–4436, May 2010.

[57] F. Ruetz, E. Hernández, M. Pfeiffer, and et al. OVPC mesh: 3D free-space representation for local ground vehicle navigation. In *International Conference on Robotics and Automation (ICRA)*, pages 8648–8654, May 2019.

[58] À. Santamaria-Navarro, E. H. Teniente, M. Morta, and J. Andrade-Cetto. Terrain classification in complex three-dimensional outdoor environments: Terrain classification in complex 3D outdoor environments. *J. field robot.*, 32(1):42–60, January 2015.

[59] F. N. Santos, H. Sobreira, D. Campos, and et al. Towards a Reliable Robot for Steep Slope Vineyards Monitoring. *Journal of Intelligent & Robotic Systems*, 83(3-4):429–444, 2016.

[60] J. L. Schönberger and J. Frahm. Structure-from-Motion revisited. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4104–4113. IEEE, June 2016.

[61] N. Seegmiller and A. Kelly. High-Fidelity yet fast dynamic models of wheeled mobile robots. *IEEE Trans. Rob.*, 32(3):614–625, June 2016.

[62] C. Shen, Y. Zhang, Z. Li, and et al. Collaborative air-ground target searching in complex environments. In *IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, pages 230–237, October 2017.

[63] S. Shimoda, Y. Kuroda, and K. Iagnemma. High-speed navigation of unmanned ground vehicles on uneven terrain using potential fields. *Robotica*, 25(4):409–424, July 2007.

[64] S. Thrun, M. Montemerlo, and A. Aron. Probabilistic terrain analysis for high-speed desert driving. In *Robotics: Science and Systems II*. Robotics: Science and Systems Foundation, August 2006.

[65] T. Thueer and R. Siegwart. Mobility evaluation of wheeled all-terrain robots. *Rob. Auton. Syst.*, 58(5):508–519, May 2010.

[66] A. Wallar, E. Plaku, and D. A. Sofge. Reactive motion planning for unmanned aerial surveillance of Risk-Sensitive areas. *IEEE Trans. Autom. Sci. Eng.*, 12(3):969–980, July 2015.

[67] Z. Zhu, E. Schmerling, and M. Pavone. A convex optimization approach to smooth trajectories for motion planning with car-like robots. In *54th IEEE Conference on Decision and Control (CDC)*, pages 835–842, December 2015.

APPENDICES

Appendix A

3D rigid body rotation

Aviation terms roll, pitch, yaw were adopted in the robotics community to represent the rotation of the rigid body around its frame of reference.

We will follow the right-hand rule, i.e., all rotations will be counterclockwise to the corresponding axis.

Yaw is the counterclockwise rotation around z-axis,

$$R_z(\alpha) = \begin{pmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{pmatrix}. \tag{A.1}$$

Pitch is the counterclockwise rotation around the y-axis,

$$R_y(\beta) = \begin{pmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{pmatrix}. \tag{A.2}$$

Roll is the counterclockwise rotation around x-axis,

$$R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{pmatrix}. \tag{A.3}$$

Performing roll, pitch, and yaw rotation yields a rotation matrix that is the result of the multiplication of yaw, pitch, and roll matrices:

$$R(\alpha,\beta,\gamma) = R_z(\alpha)R_y(\beta)R_x(\gamma) = \tag{A.4}$$
$$\begin{pmatrix} \cos\alpha\cos\beta & \cos\alpha\sin\beta\sin\gamma - \sin\alpha\cos\gamma & \cos\alpha\sin\beta\cos\gamma + \sin\alpha\sin\gamma \\ \sin\alpha\cos\beta & \sin\alpha\sin\beta\sin\gamma + \cos\alpha\cos\gamma & \sin\alpha\sin\beta\cos\gamma - \cos\alpha\sin\gamma \\ -\sin\beta & \cos\beta\sin\gamma & \cos\beta\cos\gamma \end{pmatrix}$$

Appendix B

Path planning algorithms

The following is explanation of the planners that are used in the proposed path planning framework. Let us start with RRT-Connect [33]. Followed by the explanation of RRT* [27] and clarification of the difference between original RRT* and Informed RRT* [12]. At every iteration of the algorithm, RRT-Connect operates only on one of the two trees. Please follow the Algorithms 2. We refer to the operand as a main or major tree $T_{major} = (\mathcal{N}, \mathcal{E})$, while the other one is referred as a minor tree T_{minor}. Each iteration of the algorithm we alternate the major tree T_{major} between trees rooted in start and goal states. Similarly, with the minor tree T_{minor}. As in the original RRT, at each iteration, the main tree grows towards the randomly sampled state q_{rand} via `extend` routine (see Algorithm 3). The subroutine outputs the result, $S \in \{\texttt{ADVANCED}, \texttt{REACHED}, \texttt{TRAPPED}\}$, of the attempt to connect to the provided states. The output is later used for flow control of the expansion of the trees. Subsequently, we determine the nearest neighbour $q_{nearest}$ to the randomly sampled state q_{rand}. The algorithm then allows to extend the major tree T_{major} from $q_{nearest}$ to q_{rand} only to some extent ϵ and we refer to this node as q_{new}. It means that every node in the tree is within ϵ distance from the nearest node. If an attempt to connect a new node fails $S = \texttt{TRAPPED}$, we switch the major and minor trees and proceed with the algorithm. Otherwise, the algorithm greedily attempts to connect (see Algorithm 4 the minor tree T_{minor} to the new state of T_{major}, until it is connected to the minor tree or until we hit an obstacle. The moment trees overlap, we have a feasible solution. We pass this feasible solution down to the next stage.

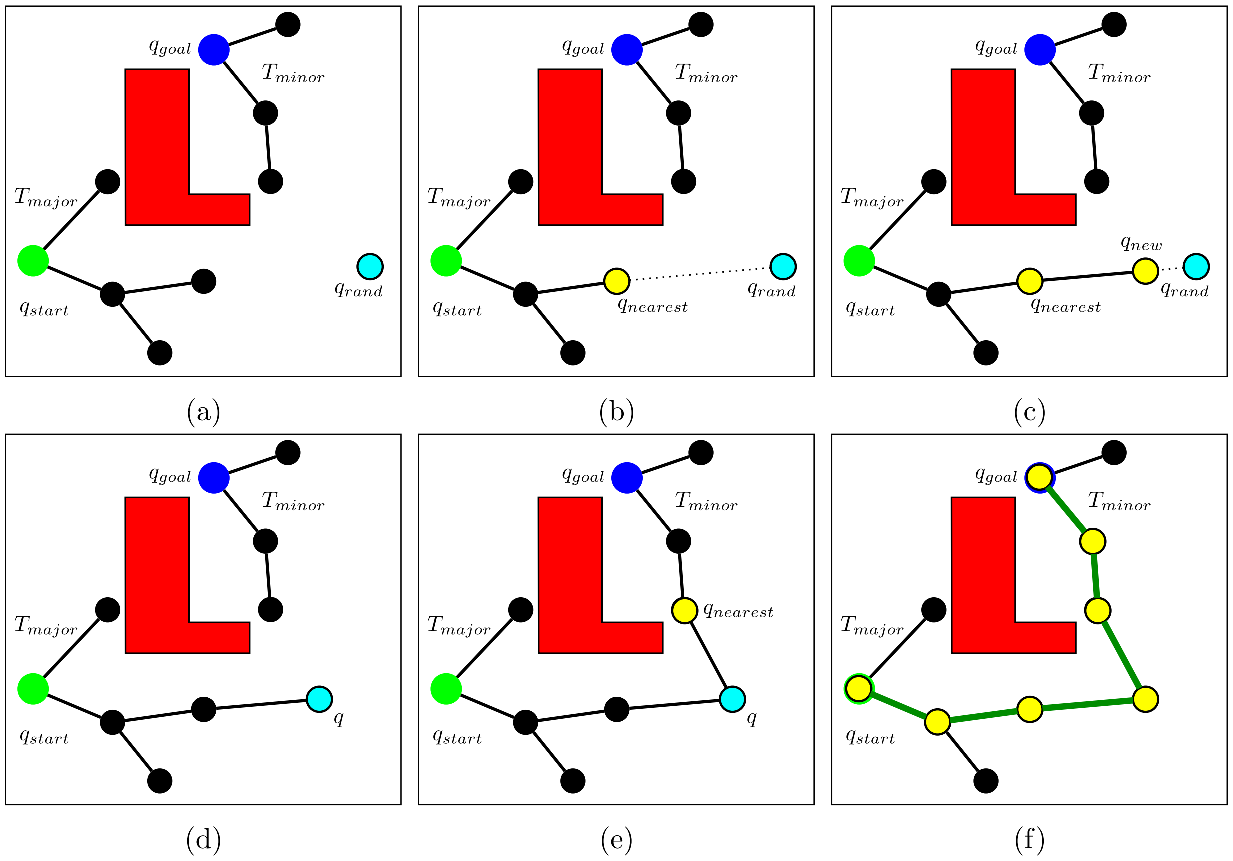


Figure B.1: (B.1a) Sample a random node q_{rand}. (B.1b) Find a nearest neighbour node $q_{nearest}$ in T_{major}. (B.1c) `extend()` computes q_{new} that is ϵ distance from $q_{nearest}$ along the motion to q_{rand}. (B.1d) Try to connect T_{minor} to q. (B.1e) Successful connection we terminate the algorithm. (B.1f) Solution path is path with yellow nodes and green edges.

Now, we will familiarize the reader with the core RRT*, since our framework is built upon this algorithm. To facilitate the understanding of the algorithm, one can refer to the pseudocode of RRT* in Algorithms 5, 6 and 7. First, the empty tree graph $\tau = (\mathcal{N}, \mathcal{E})$ is initialized. The initial state q_{start} of the robot is assigned as the root node of the tree τ. Afterwards, at each iteration, we randomly sample a node q_{rand} from a configuration space until the maximum number of iterations N is reached. In our framework, random nodes

Algorithm 2 $\pi \leftarrow$ `RRT-Connect`(q_{start}, q_{goal})

```
1: $T_{major} \leftarrow$ init_tree($q_{start}$)
2: $T_{minor} \leftarrow$ init_tree($q_{goal}$)
3: for $N$ iterations do
4:   $q_{rand} \leftarrow$ sample()
5:   if extend($T_{major}, q_{rand}$) $\neq$ TRAPPED then
6:     if connect($T_{minor}, q_{new}$) $=$ REACHED then
7:       return path($T_{major}, T_{minor}$)
8:   swap($T_{major}, T_{minor}$)
9: return empty_path()
```

Algorithm 3 $S \leftarrow$ `extend`(T, q)

```
1: $q_{nearest} \leftarrow$ nearest_neighbor($T, q$)
2: $q_{new} \leftarrow$ steer($q_{nearest}, q$)
3: if is_valid($q_{nearest}, q_{new}$) then
4:   $T \leftarrow$ place_node($q_{nearest}, q_{new}$)
5:   if $q_{new} = q$ then
6:     return REACHED
7:   else
8:     return ADVANCED
9: return TRAPPED
```

Algorithm 4 $S \leftarrow$ `connect`(T, q)

```
1: repeat
2:   $S \leftarrow$ extend($T, q$)
3: until not ($S =$ ADVANCED)
4: return empty_path()
```

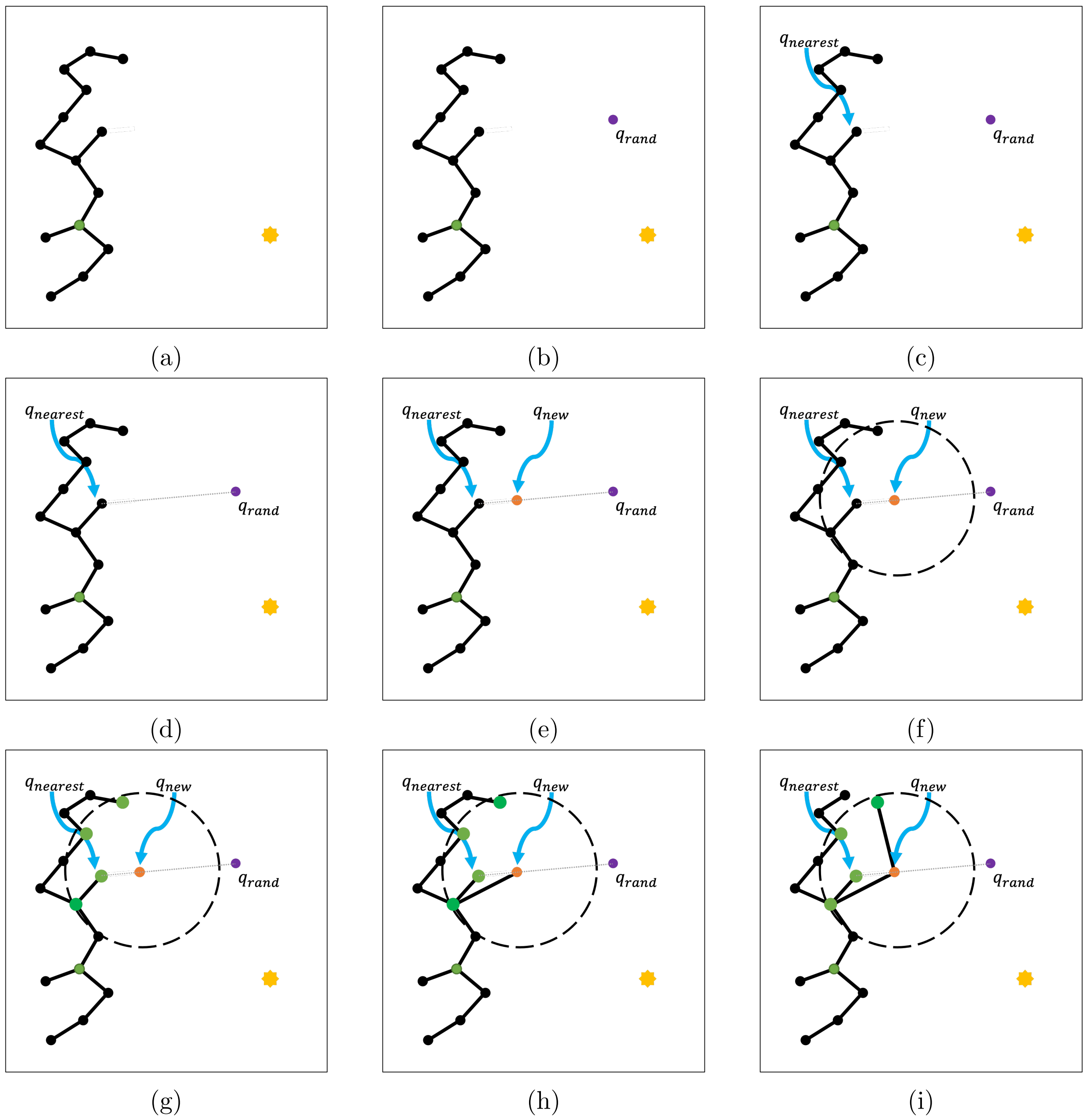


Figure B.2: First sample and find a nearest neighbour to the randomly sampled node (B.2g). Determine the valid state you can add to the tree. Find the best parent for the candidate node. Reroute all the neighbouring nodes through the new node if it results in a better cost from root to the neighbouring node.

Algorithm 5 $\tau = (\mathcal{N}, \mathcal{E}) \leftarrow$ `RRT*`(q_{start})

```
1: $\tau \leftarrow$ init_tree($q_{start}$)
2: for $N$ iterations do
3:   $q_{rand} \leftarrow$ sample()
4:   $q_{nearest} \leftarrow$ nearest_neighbor($\tau, q_{rand}$)
5:   $q_{new} \leftarrow$steer($q_{nearest}, q_{rand}$)
6:   if is_valid($q_{nearest}$, $q_{new}$) then
7:     $Q_{near} \leftarrow$ near_neighbors($\tau, q_{new}$)
8:     $q_{min} \leftarrow$ choose_parent($Q_{near}, q_{nearest}, q_{new}$)
9:     $\tau \leftarrow$ place_node($q_{min}, q_{new}, \tau$)
10:    $\tau \leftarrow$ rewire($\tau, Q_{near}, q_{min}, q_{new}$)
11: return $\tau$
```

will be sampled from the planning map (point cloud map P_M) this way we will decrease the search space of the planner. Afterwards, the nearest neighbour $q_{nearest} \in \tau$ to the new sample q_{rand} is determined and `steer` function finds the closest reachable point q_{new}. The steering function requires solving a two-point boundary value problem, at this stage of our framework, we will use a Dubins path to connect points. Then, the validity of the motion from $q_{nearest}$ to q_{new} is checked. The validity in our particular case is the absence of obstacles and admissible orientation of the robot. If a valid motion exists, RRT* initiates two local optimizations, `choose_parent` and `rewire`. To perform the optimization, the algorithm obtains Q_{near}, which is a set of nodes considered near to the q_{new} according to a distance metric (a radius or the number of near neighbours). RRT* uses the `choose_parent` routine to check all the nodes in Q_{near} and sets the parent of q_{new} as the node which minimizes the cost from q_{init}. A function `cost(`q`)` returns an accumulated cost from q_{init} to a particular node q, while $c(q_i, q_j)$ computes a cost-to-go from q_i to q_j. Next, `rewire` inspects the nodes in Q_{near} and connects them to q_{new} in case when q_{new} as the parent decreases the cost from q_{init}.

Algorithm 6 $q_{min} \leftarrow$ `choose_parent`$(Q_{near}, q_{nearest}, q_{new})$

```
1: $q_{min} \leftarrow q_{nearest}$
2: $c_{min} \leftarrow$ cost($q_{nearest}$) $+ c(q_{new}, q_{nearest})$
3: for $q_{near} \in Q_{near}$ do
4:   if is_valid($q_{near}$, $q_{new}$) then
5:     $c' =$ cost($q_{near}$) $+ c(q_{new}, q_{near})$
6:     if $c' < c_{min}$ then
7:       $q_{min} \leftarrow q_{near}$
8:       $c_{min} \leftarrow c'$
9: return $q_{min}$
```

Algorithm 7 $\tau \leftarrow$ `rewire`$(\tau, Q_{near}, q_{min}, q_{new})$

```
1: for $q_{near} \in Q_{near} \backslash q_{min}$ do
2:   if is_valid($q_{new}, q_{near}$) and
       cost($q_{new}$) $+ c(q_{new}, q_{near}) <$ cost($q_{near}$) then
3:     $\tau \leftarrow$ reconnect$(q_{new}, q_{near}, \tau)$
4: return $\tau$
```