MINIMALISTIC AND LEARNING-ENABLED NAVIGATION ALGORITHMS FOR UNMANNED GROUND VEHICLES

by

Tixiao Shan

A DISSERTATION

Submitted to the Faculty of the Stevens Institute of Technology
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Tixiao Shan, Candidate

ADVISORY COMMITTEE

Brendan Englot, Chairman	Date
Sven Esche	Date
Steven Hoffenson	Date
Long Wang	Date

STEVENS INSTITUTE OF TECHNOLOGY
Castle Point on Hudson
Hoboken, NJ 07030
2019

MINIMALISTIC AND LEARNING-ENABLED NAVIGATION ALGORITHMS FOR UNMANNED GROUND VEHICLES

ABSTRACT

Limited by the on-board computational resources of most unmanned mobile robots, autonomous navigation becomes a challenging task as it requires real-time planning, robust localization, and accurate mapping simultaneously. In this dissertation, we present several minimalistic and learning-enabled navigation algorithms that can achieve real-time performance on the hardware of a lidar-equipped unmanned ground vehicle (UGV). First, we introduce three sampling-based multi-objective path planning algorithms that are designed for relevant tasks, such as planning under risk and planning under uncertainty. These planning algorithms are suitable for both single-query and multi-query planning problems. Second, we present a lightweight and ground-optimized lidar odometry and mapping system (LeGO-LOAM) that provides real-time ego-estimation and is applicable in a wide range of indoor and outdoor environments. The system is lightweight in that it is able to run in real-time on an embedded system. It is also ground-optimized as it leverages the presence of the ground plane for ego-estimation. Third, we propose two learning-aided algorithms, Bayesian generalized kernel (BGK) terrain mapping, and lidar super-resolution, to address the sparse data problem that is encountered during mapping. BGK terrain mapping is a back-end approach that infers the traversability of gaps that exist in a robot's terrain map. Lidar super-resolution is a front-end approach, which uses deep learning to enhance range sensor resolution directly.

The motivating application of this work has been real-time autonomous navigation of GPS-denied ground robots in complex indoor-outdoor environments. Along with making optimal decisions for path planning, knowing the robot's position during operation and reflecting the surrounding world accurately in the map are also essential. The presence of slopes, vegetation, curbs and moving obstacles pose a challenging navigation problem. Deploying the proposed algorithms on a ground robot, we give results for autonomous navigation in a variety of unstructured environments where our UGV achieves high-performance path planning, low-drift localization, and accurate mapping.

Author: Tixiao Shan

Advisor: Brendan Englot

Date: August 16, 2019

Department: Mechanical Engineering

Degree: DOCTOR OF PHILOSOPHY

To my family.

Acknowledgments

I would like to thank my advisor Dr. Brendan Englot, for all the support and guidance during my time in Robust Field Autonomy Lab (RFAL). Dr. Englot is a wonderful mentor, teacher, leader, and friend to me. I have had a fantastic time working under his supervision. Dr. Englot and I would hold a one-on-one meeting every week to discuss research-related problems. He always shows a great positive attitude, which is not only encouraging but also inspiring. Dr. Englot also created such a relaxing environment for us. I always felt so comfortable while working in the lab. Besides that, Dr. Englot always tries his best to satisfy all our needs, such as lab supplies and research equipment.

I want to thank Dr. Sven Esche, Dr. Steven Hoffenson, and Dr. Long Wang for serving as my committee members. I also want to thank you for your brilliant comments and suggestions, and for letting my defense be a memorable and enjoyable moment.

I also would like to thank all my labmates of RFAL: Shi Bai, Fanfei Chen, Kevin Doherty, John Martin, Jake McConnell, Sumukh Patil, Erik Pearson, Paul Szenher and Jinkun Wang. Without their help and advice, many of my work would be impossible. It's a great pleasure working with them. I want to thank the visiting scholars of our lab, Dong Cui and Dengwei Gao, for being good friends during their stay. I want to thank Chenhui Zhao for being the most entertaining lunch buddy. I want to thank Dr. Souran Manoochehri for the support of the application for Fernando L. Fernandez Robotics and Automation Fellowship. I want to thank Dr. Mishah Salman for all the advice for my research career and it has been a great pleasure to be his teaching assistant. I want to thank Ton Duong and Jennifer Field for their help with my teaching assistantship duties. Last but not least, I want to thank all the friends, teachers and staff I met at Stevens.

Finally, I want to thank my parents for their constant support pursuing my doctoral degree. Especially, I want to thank Mr. Jeffrey Shore, for being the most supportive companion. None of the above may happen without you.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

The challenging task of performing fully embedded autonomous navigation in complex outdoor 3D environments has motivated this dissertation. This challenge has inspired our development of new algorithms, which feature minimalistic design, real-time performance, and machine learning techniques, for autonomous navigation on ground robots with limited computational resources. A detailed motivation and problem statement are introduced in Section 1.1. The contributions of this dissertation are summarized in Section 1.2.

1.1 Motivation and Problem Statement

Mobile robots have been attracting more and more research interest during the last few decades. They are preferable to manually executing dangerous missions such as mine sweeping, environment exploration and bomb diffusion. However, many tasks still need human intervention as full autonomy is still a challenge to achieve in many contexts. In this dissertation, we focus on the autonomous navigation algorithms of unmanned ground vehicles as they are more widely used in real-world applications.

Planning is a fundamental component of a autonomous navigation system. Planning converts the high-level requirements of a task to low-level commands of how to execute. Planning algorithms provide solutions to a mission by considering two basic criteria: feasibility and optimality. The feasibility of a solution guarantees the robot's safe arrival at a goal state. The optimality of a solution means that it is the most efficient way with respect to some criterion, such as time or distance, to reach at this goal state. Numerous efforts has been proposed in the last few decades to address planning problems with various cost constraints. A common approach for motion planning is to divide the workspace into 2D or 3D grid cells. Then a graph search algorithm (e.g., Dijkstra, A* and D*) can be utilized for finding an optimal path if it exists. However, such approaches are only suitable for robots with low state dimension. Over the past two decades, sampling-based planning algorithms, such as rapidly-exploring random trees and probabilistic roadmaps have drawn great attentions as they are highly successful in solving problems with high dimension.

Simultaneous Localization and Mapping (SLAM) is also a key component in designing a truly autonomous system. Both internal and external measurements from sensors can be used to localize a robot. Given this position information, the robot is able to build a map that contains its knowledge of the surrounding environment. Many successful 2D SLAM techniques are available for UGVs. However, assuming the surrounding world is planar limits the capabilities of UGVs in many circumstances. Stairs, slopes or obstacles that lie outside of a sensor's field of view may cause a navigation failure and thus cannot be ignored. SLAM in 3D environments remains challenging due to various constraints, including rough terrain and the presence of complex ambient structures. Incorporating these constraints into SLAM is essential, as flat environments exist only in a few highly structured settings.

Though many SLAM methods have been proposed in the last decade to perform localization and mapping in 3D environments, one common problem among them is that they are usually computationally expensive. Utilization of rich features in 3D environments is often required to achieve accurate localization. However, the quality of such features can make this problem non-trivial. For example, the most popular scan-matching algorithm, Iterative Closest Point (ICP), has difficulty with the large amount of points that is typically generated by a 3D lidar. Due to the small size of many UGVs, few of them have the necessary computational resources to perform real-time SLAM. Besides this, achieving accurate mapping with limited sensor data in 3D environments is also a challenging task. Point cloud maps and digital elevation maps are two of the most widely used representations. However, acquiring a dense map of the environment typically requires a significant amount of data from various sensors. One of the most popular sensors for such task is a lidar, which stands for Light Detection and Ranging. A typical 3D lidar has multiple channels and uses light in the form of a pulsed laser to measure ranges. A lidar with more channels is able to produce a denser point cloud, which may benefit the mapping task greatly. However, increasing the number of channels can be very costly.

Many robot platforms have achieved full autonomy in complex outdoor environments. However, the complex sensor setup and the custom-designed system for each robot make the extension of these work to other platforms difficult. With the goal of a minimalistic and lightweight design in mind, we want to develop autonomous navigation algorithms that have such properties:

- They should require as few sensors as possible. A common sensor that can be used in various environments should be supported.

- They should be lightweight and don't demand much computational power to achieve real-time performance. With a lightweight navigation system, the robot will be able to perform other on-board computations.

- They should be able to support UGV operation in both indoor and complex outdoor environments. Navigation algorithms that support the operation of a robot in outdoor environments will open many opportunities for its application.

- They should be applicable to general platforms. We will achieve this by ensuring compatibility with Robot Operating System (ROS). Currently, more than 80

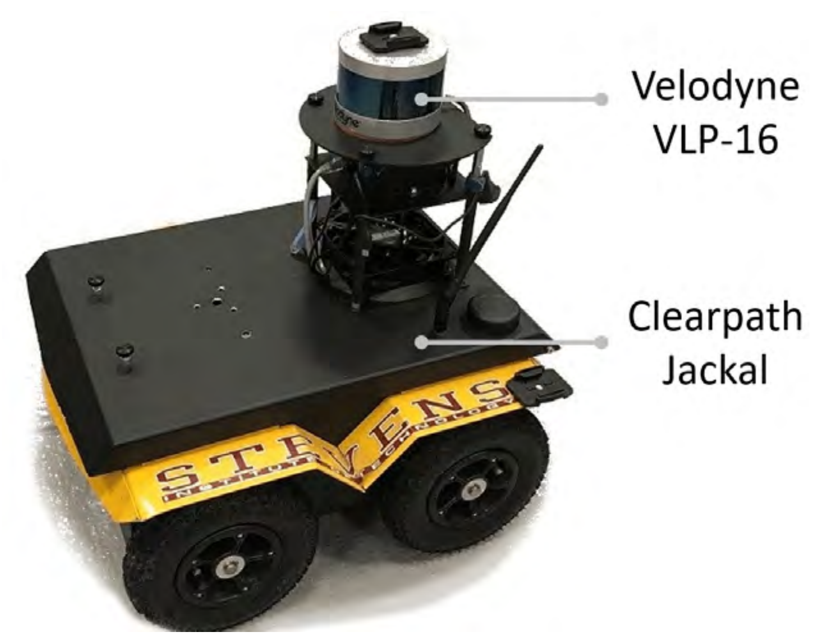


Figure 1.1: Clearpath Jackal, an unmanned ground vehicle.

different ground robots are fully supported by ROS. The compatibility of a autonomous navigation algorithm with ROS will greatly extend its usage.

The goal of this dissertation is to design autonomous navigation algorithms that can achieve the properties mentioned above. The proposed algorithms in this dissertation will be validated on a real robot - Clearpath Jackal, which is shown in Figure 1.1. Powered by a 270 Watt hour Lithium battery, it has a maximum speed of 2.0m/s and maximum payload of 20kg. The Jackal's computer is equipped with a core i7-4770s processor and 8GB ram. The Jackal is equipped with a Velodyne VLP-16 "puck" 3D Lidar. The puck's measurement range is up to 100m with an accuracy of $\pm$ 3cm. It has a vertical field of view of $30°(\pm 15°)$ and a horizontal field of view of 360°. The 16-channel sensor provides a vertical angular resolution of 2°. The horizontal angular resolution varies from 0.1° to 0.4° based on the rotation rate. Throughout the dissertation, we choose a scan rate of 10Hz, which provides a horizontal angular resolution of 0.2°.

1.2 Overview and Contributions

This dissertation is organized as follows. In Chapter 2, we review the methodologies that are proposed for achieving autonomous navigation. These methodologies cover multi-objective motion planning, lidar-based localization, traversability mapping and lidar super-resolution.

In Chapter 3, we present three novel multi-objective planning algorithms that leverage lexicographic optimization method for various planning settings. In Chapter 3.1, we propose a new sampling-based path planning algorithm, the Optimal Minimum Risk Rapidly Exploring Random Tree (MR-RRT*), that plans minimum risk paths in accordance with primary and secondary cost criteria. The primary cost criterion is a user-defined measure of accumulated risk, which may represent proximity to obstacles, exposure to threats, or similar. Risk is only penalized in areas of the configuration space where it exceeds a user-defined threshold, causing many graph nodes to achieve identical primary cost. The algorithm uses a secondary cost criterion to break ties in primary cost. The proposed method affords the user the flexibility to tune the relative importance of the alternate cost criteria, while adhering to the requirements for asymptotically optimal planning with respect to the primary cost. The algorithm's performance is compared with T-RRT*, another optimal tunable-risk planning algorithm, in a series of computational examples with different representations of risk. In Chapter 3.2, we describe a new sampling-based path planning algorithm, the Min-Max Rapidly Exploring Random Tree (MM-RRT*), for robot path planning under localization uncertainty. The projected growth of error in a robot's state estimate is curbed by minimizing the maximum state estimate uncertainty encountered on a path. The algorithm builds and maintains a tree that is shared in state space and belief space, with a single belief per robot state. Due to the fact that many states will share the same maximum uncertainty, resulting from a shared parent node, the algorithm uses secondary objective functions to break ties among neighboring nodes with identical maximum uncertainty. The algorithm offers a compelling alternative to sampling-based algorithms with additive cost representations of uncertainty, which will penalize high-precision navigation routes that are longer in duration. In Chapter 3.3, we characterize and propose advances in the technique of Belief Roadmap Search (BRMS), the process of searching a roadmap in belief space for robot motion planning under localization uncertainty. We discuss the conditions required for optimal substructure in the single-source search of a roadmap in belief space, demonstrating that there are several desirable cost functions for which this property cannot be achieved. Practical performance issues of BRMS are discussed, including the implications of a commonly-used anti-cycling rule, and the computational complexity realized in practical applications of the technique. We propose a best-first implementation of BRMS, in contrast to the standard breadth-first implementation, which we show to improve the computational cost of search by up to 49% by eliminating unnecessary node expansions - the mechanics of both approaches are compared in detail. A variety of motion planning examples are explored.

In Chapter 4, we propose a lightweight and ground-optimized lidar odometry and mapping method, LeGO-LOAM, for real-time 6 degree-of-freedom (DOF) pose estimation with ground vehicles. LeGO-LOAM is lightweight, as it can achieve real-time pose estimation on a low-power embedded system. LeGO-LOAM is ground-optimized, as it leverages the presence of the ground in its segmentation and optimization steps. We apply point cloud segmentation to filter out noisy points. A two-step Levenberg-Marquardt optimization method is proposed to provide reliable and fast optimization across two consecutive scans. We compare the performance of LeGO-LOAM with a state-of-the-art method, LOAM, using datasets gathered from variable-terrain environments with ground vehicles, and show that LeGO-LOAM achieves similar or better accuracy with reduced computational expense. We also integrate LeGO-LOAM into a SLAM framework to eliminate the pose estimation error caused by drift, which is tested using the KITTI dataset.

In Chapter 5, we introduce two novel learning-enhanced perception methods. First, we present a new approach for traversability mapping with sparse lidar scans collected by ground vehicles, which leverages probabilistic inference to build descriptive terrain maps. Enabled by recent developments in sparse kernels, Bayesian generalized kernel inference is applied sequentially to the related problems of terrain elevation and traversability inference. The first inference step allows sparse data to support descriptive terrain modeling, and the second inference step relieves the burden typically associated with traversability computation. We explore the capabilities of the approach over a variety of data and terrain, demonstrating its suitability for online use in real-world applications. Second, we propose a methodology for lidar super-resolution with ground vehicles driving on roadways, which relies completely on a driving simulator to enhance, via deep learning, the apparent resolution of a physical lidar. To increase the resolution of the point cloud captured by a sparse 3D lidar, we convert this problem from 3D Euclidean space into an image super-resolution problem in 2D image space, which is solved using a deep convolutional neural network. By novelly applying Monte-Carlo dropout in the network and removing the predictions with high uncertainty, our method produces high accuracy point clouds comparable with the observations of a real high resolution lidar. We present experimental results applying our method to several simulated and real-world datasets. We argue for the method's potential benefits in real-world robotics applications such as occupancy mapping and terrain modeling.

The main contributions of this dissertation are as follows:

- Three novel and efficient multi-objective planning algorithms that leverage lexicographic optimization methods for planning under risk and uncertainty [1, 2, 3];
- A novel lidar odometry framework that achieves state-of-the-art low drift and runs on a low-powered embedded system [4];
- A novel traversability mapping method that utilizes Bayesian generalized kernel inference [5];
- A novel architecture for deep learning-enabled lidar super-resolution.

Chapter 2
Background

This chapter contains a survey of prior work on multi-objective motion planning, lidar-based localization, traversability mapping and lidar super-resolution topics. We start with an introduction of multi-objective motion planning, reviewing algorithms and methods that are generally used in this field. Recent achievements in lidar-based localization are presented next. Then we give a review of traversability mapping methods as well as their applications. We close the chapter with a survey of lidar super-resolution-related techniques.

2.1 Multi-Objective Motion Planning

Multi-objective motion planning has been an area of interest in robotics for many years. Continuous multi-objective motion planning in two and three dimensions has been achieved by gradient descent, paired with sampling the Pareto front to identify feasible solutions under added constraints [6]. [6] looks for paths with feasible costs combination by sampling on the Pareto optimal surface. However, it may not find the optimal feasible solution if a nonconvex Pareto front exists. The number of state space dimension and cost metrics also limit the application of this method. Genetic algorithms [7, 8] and dynamic programming [9, 10, 11] have also been applied to solve multi-objective motion planning problems. Early work on multi-objective planning over configuration space roadmaps [12] has been succeeded by methods that recover Pareto fronts from probabilistic roadmaps [13, 14].

In pursuit of solutions that can be produced quickly, preferably in real-time, and applied to problems of high dimension, sampling-based motion planning algorithms such as the PRM [15], the rapidly-exploring random tree (RRT) [16], and their optimal variants PRM*, RRT*, and rapidly-exploring random graphs (RRG) [17] have been adapted to solve a variety of multi-objective motion planning problems. Such approaches have typically considered the tradeoff between a resource such as time, energy, or distance traveled and information gathered [18], localization uncertainty [19, 20], collision probability [21], clearance from obstacles [22], adherence to rules [23], and exposure to threats [13] and other generalized representations of risk [24, 25].

2.1.1 Weighted Sum Method

A weighted sum method is probably the most popular optimization approach to multi-objective planning. Optimization is achieved by minimizing a weighted sum of the competing costs (Equation 2.1). This method will be abbreviated as WS in the discussion and results to follow.

$$\begin{aligned} f_{ws}(\sigma) &= \sum_{i=1}^{k} w_i f_i(\sigma) \\ where \quad & \sum_{i=1}^{k} w_i = 1 \end{aligned} \tag{2.1}$$

Since a simple weight can be assigned to a cost metric of interest, WS method is particularly suitable for problems with more than two different types of cost criteria. A particle filter based path planning approach is proposed in [26]. This paper uses a cost function that combines distance cost and collision avoidance requirement. The collision cost of a path segment in the single composite cost function is calculated by inverting the shortest distance from this segment to the nearest obstacle. Various clearance paths can be obtained by adjusting the weight of each cost. [27] and

[28] also propose similar solutions for solving path safety problems while considering path duration. The difference is that [28] uses an entropy method to determine the weights for several criteria. [29] and [30] take map uncertainty into account along with distance, which can offer trade-offs between the speed of building map versus the accuracy of resulting map. A probabilistically conservative heuristic for planning under uncertainty is proposed in [31] by combining distance cost and uncertainty cost using the weighted sum method.

Note that the WS method can be sensitive to the choices of weights, where very small differences can have a large impact on the quality of the solutions obtained. This is potentially caused by the sum of different units of costs. For example, combining two costs, fuel consumed in gallons and distance traveled in centimeters, needs extra attention. In order to recover a Pareto front, a systematic procedure for solving an optimization problem using WS would involve choosing many combinations of weights, building a tree or searching a roadmap with each, and choosing, among various paths, the solution that satisfies the user. As a result, a tree or roadmap needs to be built or searched repeatedly for many resulting weight combinations, which can be computationally expensive. If the weight is discretized finely enough, WS can recover a satisfying Pareto front. However, if the weight is not discretized finely enough, the user is left to guess where finer discretization is needed, and the user must reason in unintuitive units.

2.1.2 Constraint-based Methods

$$\sigma* = \underset{\sigma \in \Sigma}{\operatorname{argmin}} f_i(\sigma) \quad s.t. \quad f_j(\sigma) \leq B \tag{2.2}$$

Another popular approach for multi-objective planning is constraint-based methods. As is described in Equation 2.2, constraint-based methods find the minimum cost solution within the limit of a user-defined constraint (or budget, i.e., fuel, time, collision probability). For problems that need to maximize the cost function (i.e., gathering information, transmitting data), without losing any generality, constraint-based methods can be represented as the formation in Equation 2.3.

$$\sigma* = \underset{\sigma \in \Sigma}{\text{argmax}} f_i(\sigma) \quad s.t. \quad f_j(\sigma) \leq B \tag{2.3}$$

Chance Constraint RRT, CC-RRT, uses collision probability as a constraint to extend an RRT [32]. Asymptotic optimality is achieved in [33] by adapting RRT* with collision probability. CC-RRT and CC-RRT* can generate probabilistically safe paths for linear Gaussian systems that are subject to sensor noise and localization uncertainty. In [21], chance constraints are used again in an rapidly-exploring random graph for similar systems as mentioned previously. A heuristic search method is presented in [34] for solving motion planning problems under uncertainty. The heuristic cost function in this work is a combination of distance cost and collision probability, which is similar to the WS method. The search of the roadmap, which is an adapted A* algorithm, is also constrained by collision probability. An expanded graph search that is constrained by budget is proposed in [35]. This work uses Dijkstra search on multiple layers that are assigned with a fixed budget of some resource. As a result, a Pareto front can be recovered by increasing the budget for each layer. [36] and [19] use a new metric, the maximum eigenvalue of state estimation error covariance, to calculate collision probability and as a constraint for multi-objective planning. Information gathering algorithms, which maximize an information quality metric under a budget constraint, are proposed in [18].

Constraint-based methods can usually return a Pareto front of costs by adjusting the value of the constraint. However, in order to recover a Pareto front, this method has the same problem, discretization of the constraint, as the WS method does. Again, if the constraint is discretized finely enough, this method can recover optimal paths that land on the Pareto front. If the constraint is not discretized finely enough, the user is left to guess where finer discretization is needed. Another problem of using this method is caused by the selection of the constraint before the planning. If the value of constraint is set too small, paths may never be found. On the other hand, if this value is set too large, a satisfactory solution may also be hard to find. For example, we want to find the shortest distance path that satisfies a specified collision probability α. The selection of α needs to take the environment into consideration. For a cluttered environment, a large α is preferred as a lower collision probability path may not exist.

2.1.3 Lexicographic Method

The lexicographic method [37] is the technique of solving a multi-objective optimization problem by arranging each of several cost functions in a hierarchy reflecting their relative importance. The objectives are minimized in sequence, and the evolving solution may improve with respect to every subsequent cost function if it does not worsen in value with respect to any of the former cost functions. In effect, regions of the feasible set where there are ties in high-priority cost functions will have those ties broken using lower-priority cost functions. Use of this methodology has been prevalent in the civil engineering domain, in which numerous regulatory and economic criteria often compete with the other objectives of an engineering design problem. Variants of the lexicographic method have been used in land use planning [38], for vehicle detection in transportation problems [39], and in the solution of complex multi-objective problems, two criteria at a time [40].

The problem of lexicographic optimization can be formulated as follows [41]:

$$\begin{aligned}
&\underset{\sigma_i \in \Sigma}{min} f_i(\sigma_i) \\
&subject\ to:\ \ f_j(\sigma_i) \leq f_j(\sigma_j^*) \\
&j = 1, 2, ...i-1;\ \ i = 1, 2, ..., k.
\end{aligned} \tag{2.4}$$

In the current phase of the procedure depicted in Equation 2.4, a new solution σ_i^* will be returned if it does not increase in cost with respect to any of the prior cost functions $j < i$ previously examined. Necessary conditions for optimal solutions of Equation 2.4 were first established by [42]. Relaxed versions of this formulation have also been proposed, in which $f_j(\sigma_i) \leq f_j(\sigma_j^*)$ is permitted, provided that $f_i(\sigma_i)$ is no more than a small percentage larger in value than $f_j(\sigma_j^*)$. This approach, termed the *hierarchical method* [43], has also been applied to multi-criteria problems in optimal control [44].

2.1.4 Applications

Since sampling-based planning algorithms have proven to be highly successful in solving problems of high dimension [45], we limit the scope of this dissertation to such planning algorithms. Sampling-based planning algorithms are capable of planning under a variety of challenging costs and constraints. Algorithms such as the probabilistic roadmap (PRM), rapidly exploring random tree (RRT), and their optimal variants PRM* and RRT* [17], have been adapted to curb robot state uncertainty in the objective [46], [33] and constraints [21], [47], maximize information-gathering under energy constraints [18], minimize distance traveled under task-based [48] and risk-based [25] constraints, efficiently explore narrow passages [49], and formulate multi-objective Pareto fronts [14]. Due to the limit of the scope of this disserta-

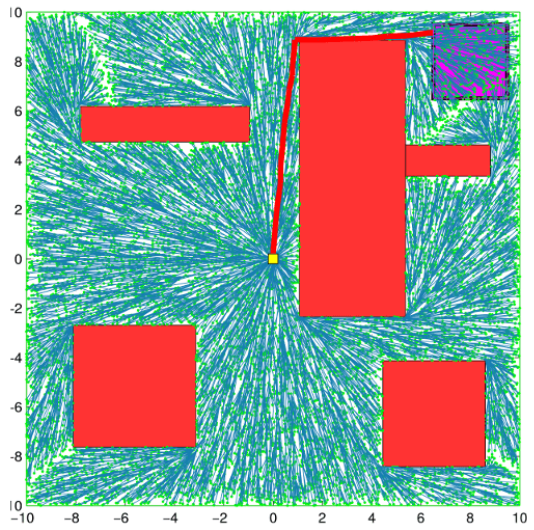


Figure 2.1: RRT* for minimum-distance planning. Red rectangles represent the obstacles in the 2D worksprace. Image credit: S. Karaman, and E. Frazzoli, 2011 [17].

tion, we focus on utilizing sampling-based planning algorithms to solve two problems: planning under risk and planning under uncertainty.

2.1.4.1 Planning Under Risk

One challenging motion planning problem is planning under risk. The minimization of risk may entail maintaining a safe distance from obstacles, avoiding exposure to threats, or other related objectives. To reduce risk by achieving clearance from obstacles, many successful feasible planning methods have been proposed. Medial axis probabilistic roadmaps (MAPRM) [50] and medial axis biased rapidly-exploring random trees (MARRT) [51] improve the performance of sampling in narrow passages and maintain clearance by retracting randomly generated configurations onto the medial axis of the free space. A PRM driven by a sum of weighted costs has been used to address path length, path clearance, and kinematic and dynamic constraints [22]. Obstacle-based rapidly-exploring random trees (OBRRT) use “hints” from obstacles

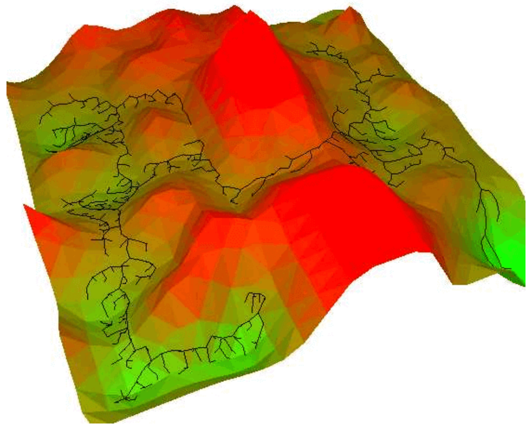


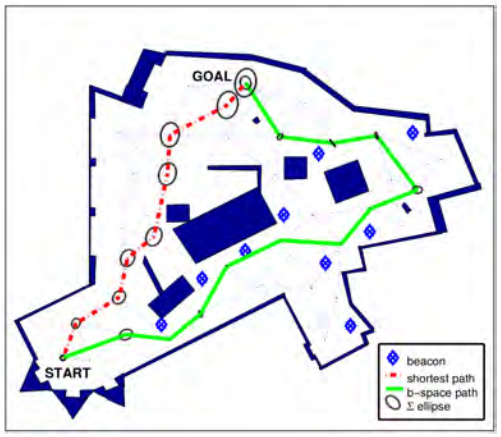
Figure 2.2: Search tree built by T-RRT. Traveling at high elevation introduces risk cost. Image credit: D. Devaurs, T. Simeon, and J. Cortes, 2011 [53].

to assist in clearing narrow passages [52].

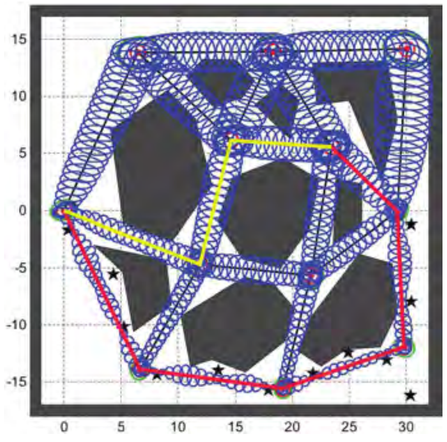
To minimize exposure to threats, a multi-objective PRM (MO-PRM) was employed to find the shortest path meeting constraints on exposure [14]. This algorithm is also capable of handling a variety of generalized primary and secondary costs. Similarly, transition-based rapidly exploring random trees (T-RRTs) produce generalized low-risk solutions by probabilistically rejecting samples that are drawn in high-risk regions of the configuration space [53]. This can be used to maximize clearance from obstacles, or to avoid exposure to threats.

T-RRT*, an extension of T-RRT [53], inherits the property of asymptotic optimality from RRT* while maintaining T-RRT's ability to achieve high clearance from obstacles [24]. This method has successfully solved complex optimal planning problems while maintaining clearance from obstacles. However, to maintain a high safety margin, this algorithm must avoid high-risk regions throughout the entire configuration space. It is challenging to plan to or through a high-risk location while also maintaining safety elsewhere in the graph.

A key property shared among the above formulations of risk is the dependency



(a) BRM



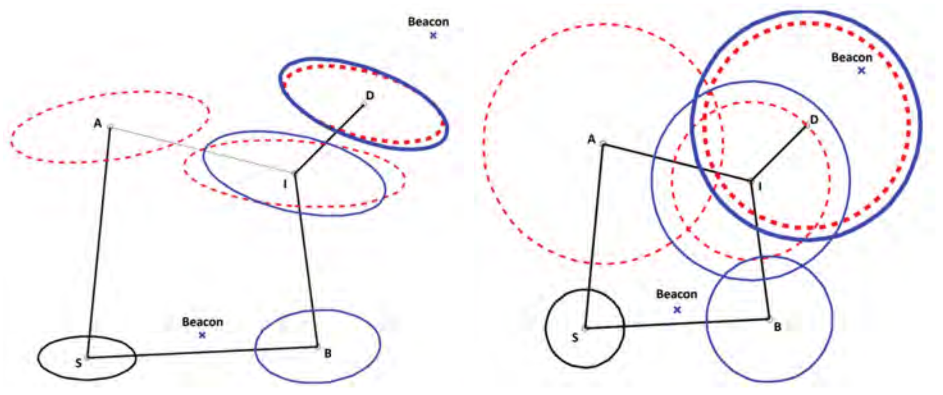
(b) FIRM

Figure 2.3: Belief roadmap (BRM) and feedback controller-based information-state roadmap (FIRM). Image credit: S. Prentice, and N. Roy, 2009 [46], A. Agha, S. Chakravorty, and N. M. Amato, 2014, [54]

of the risk function on a single robot configuration. Although quantities such as state estimate uncertainty and collision probability may also serve as measures of risk, such measures depend on the robot's initial error covariance, and measurement and action histories, lying outside the scope of the risks discussed in this dissertation, which depend on a single robot configuration.

2.1.4.2 Planning Under Uncertainty

Another challenging motion planning problem is planning under uncertainty, which may seek to find feasible or asymptotically optimal plans in the presence of probabilistic actions, measurements, and/or environment maps. Sampling-based approaches to planning under uncertainty have typically assumed that belief spaces are Gaussian. The objective function itself may address the uncertainty associated with a localization process, as in the case of the belief roadmap (BRM) [46] and robust belief roadmap (RBRM) [47]. Standard minimum-distance or minimum-energy objective functions may also be used in combination with constraints related to localiza-



(a) Planning using trace as cost metric (b) Planning using maximum eigenvalue as cost metric

Figure 2.4: An example in which the optimal substructure property does not hold when using the error covariance trace as a planning metric. Image credit: S. Bopardikar, B. Englot, A. Speranzon, and J. van den Berg, 2016 [47]

tion uncertainty, as in the case of the rapidly-exploring random belief tree (RRBT) [21], bounded-uncertainty RRT* (BU-RRT*) [20], Box-RRT [55], and an expanded adaptation of the PRM [19]. Chance-constrained RRT* (CC-RRT*) has combined these concepts into a sampling-based algorithm that enforces constraints on a robot's collision probability while also penalizing collision risk in the objective [33]. Such problems can also be formulated as stochastic optimal control problems, solved by feedback motion planning, as with linear quadratic Gaussian motion planning (LQG-MP) [56] and the Feedback Controller-Based Information-State Roadmap (FIRM) [54], or by approximating optimal policies for Markov decision processes (MDPs) and partially observable Markov decision processes (POMDPs), as with the Stochastic Motion Roadmap (SMR) [57], incremental-MDP (iMDP) [58], and Adaptive Belief Tree (ABT) [59].

The feedback controller-based information-state roadmap (FIRM) [54] generates a graph in belief space that has independent edge costs due to the implementation of belief-stabilizing controllers, which preserve the optimal substructure property. A latter work enforces constraints on localization uncertainty over a PRM by searching an expanded graph that represents the PRM at different levels of uncertainty [19]. The robust belief roadmap (RBRM) [47] employs a novel uncertainty metric, an upper bound on the maximum eigenvalue of the EKF covariance matrix **P**, which preserves the optimal substructure property when searching to minimize goal-state uncertainty. However, when the trace of **P** or the true maximum eigenvalue of **P** are used as planning metrics instead, optimal substructure is not guaranteed. A roadmap search example in which the true maximum eigenvalue of **P** fails to produce optimal substructure is given in Figure 2.4(a). The optimal path from S to I is not a subset of the optimal path from S to Goal, unlike the case in Figure 2.4(b), which uses the eigenvalue upper bound metric instead.

2.2 Lidar-based Localization

Among the capabilities of an intelligent robot, map-building and state estimation are among the most fundamental prerequisites. Great efforts have been devoted to achieving real-time 6 degree-of-freedom simultaneous localization and mapping (SLAM) with vision-based and lidar-based methods. Although vision-based methods have advantages in loop-closure detection, their sensitivity to illumination and viewpoint change may make such capabilities unreliable if used as the sole navigation sensor. On the other hand, lidar-based methods will function even at night, and the high resolution of many 3D lidars permits the capture of the fine details of an environment at long ranges, over a wide aperture. Therefore, we focus on the techniques using 3D lidar to support real-time state estimation.

The typical approach for finding the transformation between two lidar scans is iterative closest point (ICP) [60]. By finding correspondences at a point-wise level,

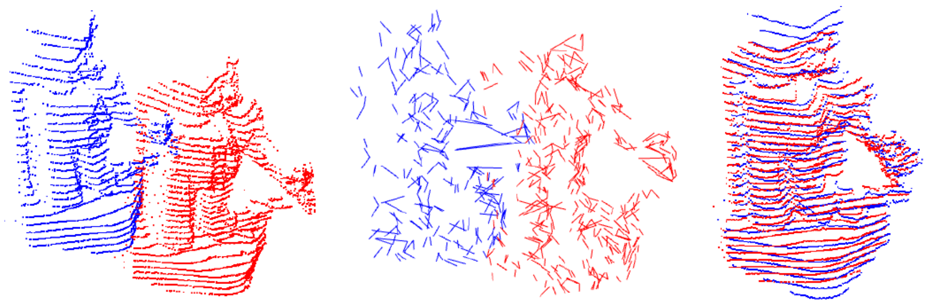


Figure 2.5: An example of a collar line registration process. Two sets of extracted collar lines from two unaligned scans are aligned for yielding a final transformation. Image credit: M. Velas, M. Spanel, and A. Herout, 2016 [72].

ICP aligns two sets of points iteratively until stopping criteria are satisfied. When the scans include large quantities of points, ICP may suffer from prohibitive computational cost. Many variants of ICP have been proposed to improve its efficiency and accuracy [61]. [62] introduces a point-to-plane ICP variant that matches points to local planar patches. Generalized-ICP [63] proposes a method that matches local planar patches from both scans. In addition, several ICP variants have leveraged parallel computing for improved efficiency [64, 65, 66, 67].

Feature-based matching methods are attracting more attention, as they require less computational resources by extracting representative features from the environment. These features should be suitable for effective matching and invariant of point-of-view. Many detectors, such as Point Feature Histograms (PFH) [68] and Viewpoint Feature Histograms (VFH) [69], have been proposed for extracting such features from point clouds using simple and efficient techniques. A method for extracting general-purpose features from point clouds using a Kanade-Tomasi corner detector is introduced in [70]. A framework for extracting line and plane features from dense point clouds is discussed in [71].

Many algorithms that use features for point cloud registration have also been proposed. [74] and [75] present a keypoint selection algorithm that performs point

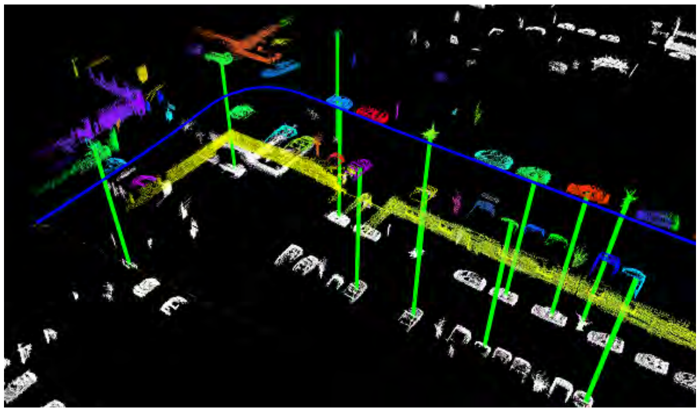


Figure 2.6: Segmented point clouds are used for loop closure detection. Image credit: R. Dube, D. Dugas, E. Stumm, J. Nieto, R. Siegwart, and C. Cadena, 2016 [73].

curvature calculations in a local cluster. The selected keypoints are then used to perform matching and place recognition. By projecting a point cloud onto a range image and analyzing the second derivative of the depth values, [76] selects features from points that have high curvature for matching and place recognition. Assuming the environment is composed of planes, a plane-based registration algorithm is proposed in [77]. An outdoor environment, e.g., a forest, may limit the application of such a method. A collar line segments (CLS) method, which is especially designed for Velodyne lidar, is presented in [72]. CLS randomly generates lines using points from two consecutive "rings" of a scan. Thus two line clouds are generated and used for registration. However, this method suffers from challenges arising from the random generation of lines. A segmentation-based registration algorithm, SegMatch, is proposed in [73]. SegMatch first applies segmentation to a point cloud. Then a feature vector is calculated for each segment based on its eigenvalues and shape histograms. A random forest algorithm is used to match the segments from two scans.

A low-drift and real-time lidar odometry and mapping (LOAM) method is proposed in [78] and [79]. LOAM performs point feature to edge/plane scan-matching to find correspondences between scans. Features are extracted by calculating the

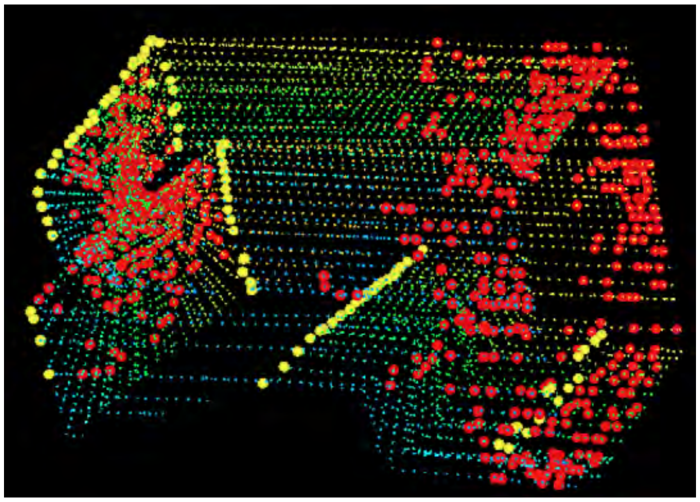


Figure 2.7: An example of extracted edge points (yellow) and planar points (red) from a 3D lidar scan. Image credit: J. Zhang, and S. Singh, 2014 [78].

roughness of a point in its local region. The points with high roughness values are selected as edge features. Similarly, the points with low roughness values are designated planar features. Real-time performance is achieved by novelly dividing the estimation problem across two individual algorithms. One algorithm runs at high frequency and estimates sensor velocity at low accuracy. The other algorithm runs at low frequency but returns high-accuracy motion estimation. The two estimates are fused together to produce a single motion estimate at both high frequency and high accuracy. LOAM's resulting accuracy is the best achieved by a lidar-only estimation method on the KITTI odometry benchmark site [80].

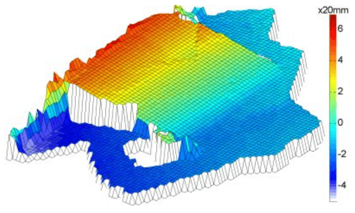
2.3 Traversability Mapping

Applying autonomous navigation in real-world scenarios require accurate representation of the environment. Methodologies for 2D planar navigation [81] require fewer constraints to be considered while mapping and manuevering, however, assuming the surrounding world is planar limits the capabilities of UGVs in many circumstances.

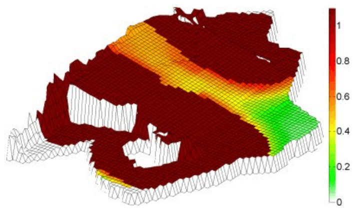
Stairs, slopes or obstacles that lie outside of a sensor's field of view may cause a navigation failure when they are encountered during task execution. Autonomous navigation in 3D environments remains challenging due to various constraints, including rough terrain and the presence of complex ambient structures. Incorporating these constraints into UGV navigation is essential, as flat environments exist only in a few highly structured settings. To perceive and evaluate the environment, a wide field of view is typically required, enabled by cameras or a volumetric scanning lidar. Using these sensors, many methods have achieved successful terrain traversability analysis for various UGV platforms.

When a 2D lidar is used to perceive 3D environments, it is typically mounted on a rotating mechanism to generate a 3D point cloud. Three such lidars were used to generate point clouds for autonomous navigation in [82]. The point cloud is converted into a 2D occupancy grid map by classifying planes in the environment as clear, obstacles, unknown or expensive (traversable but should be avoided). Assuming the world is composed of flat planes, [83] used a segmentation method to partition a point cloud into traversable planes. A fuzzy inference method was proposed in [84] for terrain analysis. Two traversability associated values, roughness and slope, are input to the inference method for analysis, generating a vector field histogram for motion planning.

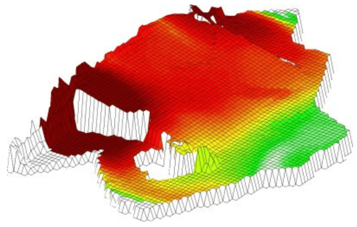
In contrast with 2D lidar, cameras and 3D lidar have seen wider usage in traversability analysis for UGVs. A navigation system was developed in [85] based on stereo cameras, in which the perceived stereo images are used for pose estimation and terrain modeling. The traversability of the terrain is calculated using three criteria: slope, roughness and step height. [86] presented a method that constructs a mesh representation of the 3D navigable terrain for motion planning tasks. Using a Kinect sensor, [87] analyzed terrain traversability by three factors: maximum step height

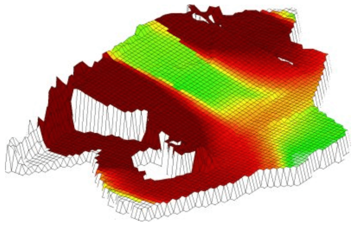


(a) Elevation map



(b) Traversability value


(c) Slope


(d) Roughness

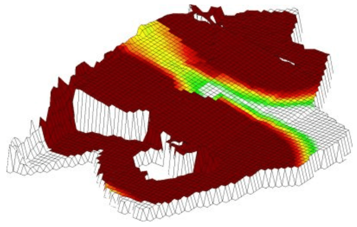

(e) Step height

Figure 2.8: Traversability values calculated using an elevation map: slope, roughness and step height. Image credit: A. Chilian, and H. Hirschmüller, 2014 [85].

limit, maximum slope and robot height. A graph-based approach that maps a point cloud onto a 2D grid of cells with elevation information was mentioned in [88]. The traversability of a cell is then labeled based on the slope value between this cell and the neighboring cells. Equipped with a 2D lidar and stereo camera, BigDog [89] is able to navigate in unstructured outdoor environments. By projecting obstacles into a 2D world, a grid-based model of the terrain is used for planning along with a proposed A* path planner. An RRT*[17] based planner was implemented in [90] for planning with a rescue robot. Terrain roughness is evaluated by comparing height differences with a threshold that is determined by the robot's maximal traversable height. [91] proposed a traversability estimation method for UGVs in rough terrain. Two sensors, a 3D lidar and camera, are used independently to build separate traversability maps. The 3D lidar measures the slope of the ground, while the camera collects training data for a terrain classifier. The separate maps are then fused using Bayes' rule to improve terrain characterization.

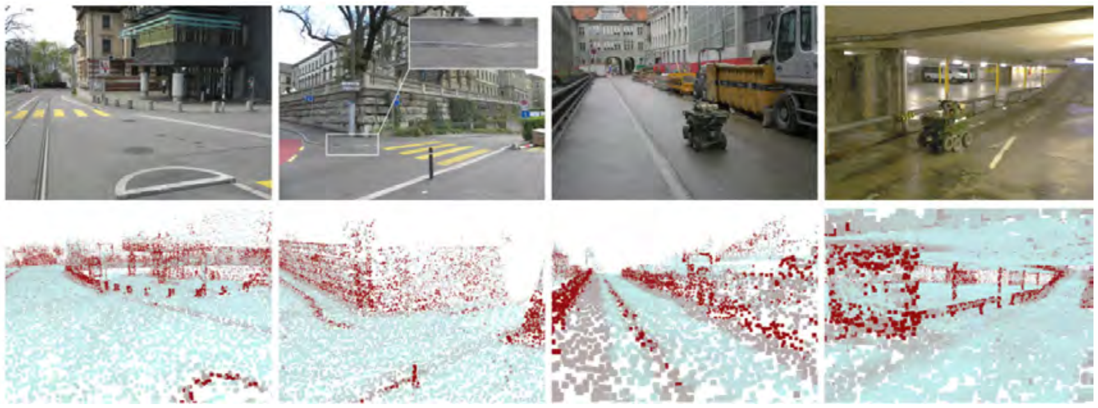


Figure 2.9: Terrain assessment results in an urban environment. Image credit: P. Krsi, P. Furgale, M. Bosse, and R. Siegwart, 2016 [97].

Many learning-based evaluation methods have been proposed for traversability analysis, using cameras and ranging. [92] presented an approach for terrain surface detection in forests using self-supervised visual learning. In this work, the ground planes, which are assigned using a supervised support vector machine (SVM) classifier, are used to generate training samples for a visual classifier. [93] introduced a reinforcement learning based traversability analysis method, which is trained on obstacles composed of pallets, and tested in natural forest environments. An offline method that uses Gaussian processes to classify the traversability of terrain was proposed in [94], in which the samples for training are acquired while driving the robot through traversable terrain. A semi-supervised learning method was used in [95] for traversability analysis, in which the underlying model is inferred from positive labeled training data that is acquired from traversable terrain. A normal distributions transform traversability mapping (NDT-TM) approach was introduced in [96]. A SVM was used to distinguish traversable and non-traversable areas in densely forested terrain.

Raw lidar-derived point cloud maps can also be used for traversability classification and navigation. Successful path planning was achieved in [98] by assessing the traversable surface of a point cloud with a tensor voting framework. A tensor voting process was also applied in [99] to evaluate the traversability of point cloud data for a rescue robot. The robot, which has 10 degrees-of-freedom, is able to navigate in complex 3D environments that require stair-climbing using the D* algorithm for path planning. The "driving on point clouds" framework was introduced in [97], in which the traversability of the point cloud map is evaluated by a local terrain assessment method, computing the robot-size terrain roughness for a map point, without reconstructing the terrain like many previous works. The proposed planning approach in [97] is composed of three main steps: (1) an initial feasible trajectory is computed using RRTs, (2) the initial path is optimized by an RRT*-based method, and (3) the local trajectory is optimized according to a user-defined cost function. Though [99] and [97] can achieve autonomous navigation for a ground robot in 3D environments, and [97] can classify map points as static/dynamic and manage them accordingly, these frameworks need a relatively dense point cloud map to be provided prior to planning.

Despite the success of these methods, applying traversability mapping to real-world online navigation is still nontrivial, as a prior map is not always available and a UGV may lack the required perceptual and/or computational capability to navigate robustly. Lidar-based traversability mapping methods often suffer from sparse data, which limits their ability to provide sufficient coverage to support autonomous navigation. To solve this problem, a terrain modeling process can be introduced to predict terrain height in unknown locations before traversability analysis. Gaussian process (GP) regression has been applied to estimate terrain height in locations not directly observed by a robot's range sensor [100]. However, the complexity of GPs has limited their use in real-time computation requiring incremental updates. A variational Hilbert regression (VHR) framework is proposed in [101] to generate accurate terrain models. VHR outperforms GPs in terms of both accuracy and efficiency. However, the parameter tuning required, along with a potentially costly iterative optimization, may limit its application to real-world online mapping tasks.

2.4 Lidar Super-resolution

The work of lidar super-resolution is most related to the *image super-resolution* problem, which aims to enhance the resolution of a low-res image. Many techniques have been proposed over the past few decades and have achieved remarkable results [102]. Traditional approaches such as linear or bicubic interpolation [103], or Lanczos resampling [104], can be very fast but oftentimes yield overly smooth results. Recently, with developments in the machine learning field, deep learning has shown superiority in solving many prediction tasks, including the image super-res problem. Methods based on deep learning aim to establish a complex mapping between low-res and high-res images. Such a mapping is usually learned from massive training data where high-res images are available. For example, a super-resolution convolutional neural network, SR-CNN, trains a three-layer deep CNN end-to-end to upscale an image [105]. Over time, deeper neural networks with more complex architectures have been proposed to further improve the accuracy [106, 107, 108, 109]. Among them, SR-GAN [109] achieves state-of-the-art performance by utilizing a generative adversarial network [110]. The generator of SR-GAN, which is called SR-ResNet, is composed of two main parts, 16 residual blocks [111] and an image upscaling block. A low-res image is first processed by the 16 residual blocks that are connected via skip-connections and then upscaled to the desired high resolution. The discriminator network of SR-GAN is a deep convolutional network that performs classification. It discriminates real high-res images from generated high-res images. It outperforms many other image super-res methods, including nearest neighbor, bicubic, SR-CNN and those of

[106, 107, 108], by a large margin.

Another problem that is related to lidar super-resolution is *depth completion*. The goal of this task is to reconstruct a dense depth map with limited information. Such information usually includes a sparse initial depth image from a lidar or from an RGB-D camera [112, 113]. Typically, an RGB image input is also provided to support depth completion, since estimation solely from a single sparse depth image is oftentimes ambiguous and unreliable. For instance, a fast depth completion algorithm that runs on a CPU is proposed in [114]. A series of basic image processing operations, such as dilation and Gaussian blur, are implemented for acquiring a dense depth map from a sparse lidar scan. Though this method is fast and doesn't require training on vast data, its performance is inferior when compared with many other approaches. A self-supervised depth completion framework is proposed in [115]. In this work, a deep regression network is developed to predict dense depth from sparse depth. The proposed network resembles an encoder-decoder architecture and uses sparse depth images generated by a lidar, with RGB images as optional inputs. Another problem that is closely related to depth completion is *depth prediction*, which commonly utilizes images from a monocular or stereo camera [116, 117, 118, 119]. Due to our focus here on a lidar-only super-resolution method, an in-depth discussion of this problem lies beyond the scope of this dissertation.

Instead of solving the super-resolution problem in image space, PU-Net [120] operates directly on point clouds for upsampling, and adopts a hierarchical feature learning mechanism from [121]. This approach performs super-resolution on point cloud models of individual small objects.

Chapter 3
Efficient Multi-Objective Planning

In this chapter, we utilize the lexicographic optimization method that is mentioned in Section 2.1.3, and introduce three novel and efficient sampling-based planning algorithms for two problems: planning under risk and planning under uncertainty. The rest of the chapter is organized as follows. A sampling-based planning algorithm that optimizes two costs in a hierarchy is proposed for planning under risk in Section 3.1. A Min-Max rapidly-exploring random tree algorithm that optimizes three costs is presented for planning under uncertainty in Section 3.2. A probabilistic roadmap-based planning algorithm is described in Section 3.3 for multi-query planning under uncertainty.

3.1 Minimum-Risk Planning

To address the problem of planning under risk, we introduce the optimal Minimum Risk Rapidly Exploring Random Tree (MR-RRT*), which can enter high risk regions when they are a critical component of feasibility, while penalizing them in general. Like T-RRT*, the level of risk avoidance is tunable, except risk is addressed in the objective rather than through a sample rejection constraint. A risk penalty is only imposed on regions of the configuration space that lie above a user-designated threshold. As a result, many nodes in the tree achieve identical values of primary cost, and a secondary cost function, such as accumulated distance along the path, may be used to break ties in the primary, risk-based cost. In areas of the configuration space where risk is not a priority, paths may be constructed with respect to the secondary cost criterion instead, allowing a user to tune the relative importance of the two cost criteria.

3.1.1 Problem Definition

Let $\mathcal{C} \subset \mathbb{R}^d$ be a robot's configuration space. $x \in \mathcal{C}$ represents the robot's position and volume occupied in $\mathcal{C}$. $\mathcal{C}_{obst} \subset \mathcal{C}$ denotes the set of obstacles in $\mathcal{C}$ that will cause collision with the robot. $\mathcal{C}_{free} = cl(\mathcal{C} \backslash \mathcal{C}_{obst})$, in which $cl()$ represents the closure of an open set, denotes the space that is free of collision in $\mathcal{C}$. We will assume that given an initial configuration $x_{init} \in \mathcal{C}_{free}$, the robot must reach a goal region $X_{goal} \subset \mathcal{C}_{free}$. Let a *path* be a continuous function $\sigma : [0, 1] \rightarrow \mathbb{R}^d$ of finite length. Let Σ be the set of all paths σ in a given configuration space. A path is collision-free if $\sigma \in \mathcal{C}_{free}$ for all arguments, and a collision-free path is *feasible* if $\sigma(0) = x_{init}$ and $\sigma(1) \in X_{goal}$. The problem of finding a feasible path may be specified using the tuple $(C_{free}, x_{init}, X_{goal})$. A *cost function* $c : \Sigma \rightarrow \mathbb{R}_0^+$ returns a non-negative cost for all collision-free paths. We may now define an optimal path planning problem.

Definition 1 (Optimal Path Planning)**.** *Given a path planning problem and a cost function* $c : \Sigma \rightarrow \mathbb{R}_0^+$*, find a feasible path* $\sigma*$ *such that* $c(\sigma*) = min\{c(\sigma) \mid \sigma \in \mathcal{C}_{free},\ \sigma(0) = x_{init},\ \sigma(1) \in X_{goal}\}$ *if such a path exists, and report failure otherwise.*

We express optimal path planning problems using the tuple $(C_{free}, x_{init}, X_{goal}, c)$. We are principally concerned with optimal path planning problems that admit a *robustly optimal solution*, which is required for the optimality properties established in [17] to apply. A robustly optimal solution must have neighboring paths in the same homotopy class with spatial clearance from surrounding obstacles in all dimensions of $\mathcal{C}$. Limiting our analysis to the broad class of problems that admit this type of solution will allow MR-RRT* to inherit the *asymptotic optimality* properties of RRT*. We now define asymptotic optimality [17].

Definition 2 (Asymptotic Optimality). *A sampling-based algorithm $\mathcal{A}$ for solving the optimal path planning problem is asymptotically optimal if, for any optimal path planning problem $(\mathcal{C}_{free}, x_{init}, X_{goal}, c)$ admitting a robustly optimal solution of finite cost $c* \in \mathbb{R}_0^+$, the cost $c(\sigma)$ of the current solution obtained by $\mathcal{A}$ approaches $c*$ as the number of samples drawn by the algorithm approaches infinity.*

Due to our consideration of sampling-based algorithms only for the solution of optimal path planning problems, we will assume the robot moves through $\mathcal{C}_{free}$ along paths obtained from a directed graph $G(V, E)$, with vertices V and edges E. Our proposed algorithm uses the following primary cost function, which penalizes the risk accumulated along a path that is derived from G:

$$c_{Risk}(\sigma) := \int_{\sigma(0)}^{\sigma(1)} Risk(\sigma(s))ds \tag{3.1}$$

$$Risk(x) := \begin{cases} \mathcal{R}(x), & \text{if } \mathcal{R}(x) > RiskThreshold \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

where the function $Risk : \mathcal{C}_{free} \rightarrow \mathbb{R}_0^+$ evaluates the risk at an individual robot configuration. We penalize a robot's risk using the tunable risk threshold $RiskTheshold \in \mathbb{R}^+$. $\mathcal{R} : \mathcal{C}_{free} \rightarrow \mathbb{R}^+$ represents the strictly positive underlying risk at a robot configuration, which is evaluated against $RiskThreshold$ and returned if the threshold is exceeded. We will alternately express $c_{Risk}(\sigma)$ using the notation $RI(\sigma(0), \sigma(1))$ to indicate the specific configurations at the limits of the risk integral.

In addition, we define a secondary cost c_{second} as follows:

$$c_{second}(\sigma) := \int_{\sigma(0)}^{\sigma(1)} Second(\sigma(s))ds \tag{3.3}$$

where the function $Second : \mathcal{C}_{free} \rightarrow \mathbb{R}^+$ represents a strictly positive cost, ensuring that ties in secondary cost do not occur as they do for primary cost. We will alternately express $c_{second}(\sigma)$ using the notation $SI(\sigma(0), \sigma(1))$ to indicate the limits of the secondary cost integral.

The contribution of each configuration x_i to either cost function is determined by the *location* of x_i of exclusively; there are no other dependencies. Two possible formulations of $\mathcal{R}(x)$ will be considered in the examples to follow, along with $Second(x) = 1$, giving us a path integral for the secondary objective that returns the distance accumulated over a path.

3.1.2 Algorithm Description

Our proposed thresholded minimum-risk planning algorithm, MR-RRT*, is given in Algorithm 3.1. Provided with input $(\mathcal{C}_{free}, x_{init}, X_{goal})$, the algorithm begins each iteration by drawing a random sample and steering toward the sample from the nearest node in G, resulting in the generation of a new node x_{new}. The best parent for x_{new} is identified by examining all neighbors, X_{near}, within a ball radius that shrinks logarithmically according to $\gamma(log(card(V))/card(V))$, where $\gamma > 2(1 + 1/d)^{1/d}(\mu(\mathcal{C}_{free})/\mu^d_{ball})^{1/d}$. The volume of the d-dimensional unit ball in Euclidean space is indicated by μ^d_{ball}. Since the algorithm performs geometric planning in $\mathbb{R}^d$, the $Steer(x_i, x_j)$ function simply attempts to connect a line segment from x_i to x_j, limiting the segment to the length of the ball radius if x_j is farther than this from x_i. These steps are also employed by the standard RRT* algorithm.

The function $RI(x_{near}, x_{new})$ is used to evaluate, per Equation 3.1, the risk accumulated while traveling to x_{new} from x_{near}. $C(x_{near})$ denotes the total risk accumulated in traveling from the root of the tree, x_{init}, to x_{near}. X_{min} comprises the node(s) in X_{near} offering the path(s) of minimum accumulated risk to x_{new}. Due to the fact that we employ a risk threshold, and some regions of $\mathcal{C}_{free}$ do not penalize

Algorithm 3.1: MR-RRT*

```
Input: $V \leftarrow \{x_{init}\}; E \leftarrow \emptyset$;
1  for $i = 1, ..., n$ do
2      $x_{rand} \leftarrow Rand()$ ;
3      $x_{nearest} \leftarrow Nearest(V, x_{rand})$;
4      $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$;
5      if $ObsFree(x_{nearest}, x_{new})$ then
6          $X_{near} \leftarrow Near(V, E, x_{new}); V \leftarrow V \cup \{x_{new}\}$;
7          $X_{min} \leftarrow \{x_{nearest}\}$;
8          $c_{min} \leftarrow RI(x_{nearest}, x_{new}) + C(x_{nearest})$;
9      for $x_{near} \in X_{near}$ do
10         if $ObsFree(x_{near}, x_{new})$ then
11             if $RI(x_{near}, x_{new}) + C(x_{near}) < c_{min}$ then
12                 $X_{min} \leftarrow \{x_{near}\}$;
13                 $c_{min} \leftarrow RI(x_{near}, x_{new}) + C(x_{near})$;
14             else if $RI(x_{near}, x_{new}) + C(x_{near}) = c_{min}$ then
15                 $X_{min} \leftarrow X_{min} \cup \{x_{near}\}$;
16     $x_{min} \leftarrow Tiebreak(X_{min}, x_{new})$;
17     $E \leftarrow E \cup \{(x_{min}, x_{new})\}$;
18     for $x_{near} \in X_{near} \setminus \{x_{min}\}$ do
19         $replace \leftarrow false$;
20         $x_{parent} \leftarrow Parent(x_{near})$;
21         if $ObsFree(x_{new}, x_{near})$ then
22             $c_{near} \leftarrow C(x_{near})$;
23             $c_{new} \leftarrow RI(x_{new}, x_{near}) + C(x_{new})$;
24             if $c_{new} < c_{near}$ then $replace \leftarrow true$; ;
25             else if $c_{new} = c_{near}$ then
26                 $X_{pair} \leftarrow \{x_{parent}, x_{new}\}$;
27                 if $x_{new} = Tiebreak(X_{pair}, x_{near})$ then
28                     $replace \leftarrow true$;
29         if $replace = true$ then
30             $E \leftarrow (E \setminus \{(x_{parent}, x_{near})\}) \cup \{(x_{new}, x_{near})\}$;
31             $X_{children} \leftarrow Children(x_{near})$;
32             $RecursivePropagate(x_{near}, X_{children})$;
33 return $G = (V, E)$
```

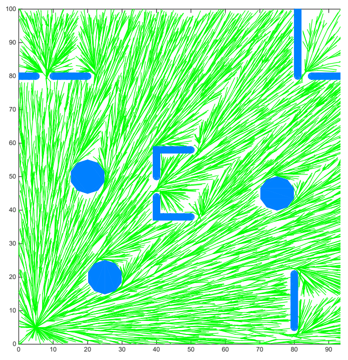
Algorithm 3.2: $Tiebreak(X_{tied}, x_{child})$

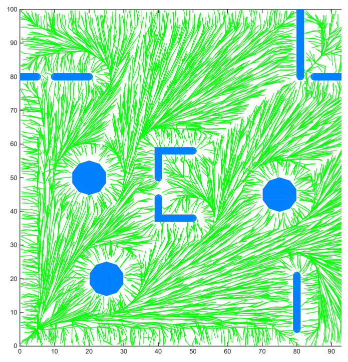
```
1  $c_{best} \leftarrow +\infty$ ;
2  for $x_i \leftarrow X_{tied}$ do
3      if $SI(x_i, x_{child}) + SC(x_i) < c_{best}$ then
4          $c_{best} \leftarrow SI(x_i, x_{child}) + SC(x_i)$;
5          $x_{best} \leftarrow x_i$;
6  return $x_{best}$
```

the travel of the robot, several nodes in X_{near} may offer identical primary cost. Consequently, $X_{min} \subseteq X_{near}$ may contain multiple nodes, requiring the tie to be broken using a secondary cost function. $Tiebreak()$, detailed in Algorithm 3.2, breaks ties in primary cost using $SC(x_i)$, the accumulated secondary cost from x_{init} to x_i, and the secondary cost integral $SI(x_i, x_j)$, from the candidate node x_i to the child node x_j .

After x_{new} is connected to a parent in X_{near}, MR-RRT* attempts to "rewire" all of the nodes in X_{near} by replacing each parent by x_{new} if it can reduce the primary cost, or in the instance of a tie, reduce the secondary cost. This process compares the risk cost of traveling to x_{near} via x_{new}, denoted as c_{new}, with the current risk cost at x_{near}, denoted c_{near}. If x_{new} offers a lower risk cost, it will replace the parent of x_{near}. If x_{new} and the parent of x_{near} offer identical risk cost, then secondary cost is used to break the tie between the two candidate parent nodes of x_{near}.

The value of $RiskThreshold$ effects the influence of the primary and secondary cost functions on the resulting tree. Let us assume that the inverse distance transform is adopted as $\mathcal{R}(x)$, and our secondary cost function is represented by the accumulated distance along a path. When $RiskTheshold$ is infinite, risk is penalized nowhere in the configuration space. In this case, MR-RRT* reduces to RRT* and produces minimum-distance paths. When $RiskThreshold$ is positive and finite, the regions within a designated distance of the obstacles are considered dangerous, and risk is penalized in those regions. When $RiskThreshold$ is zero, risk is penalized everywhere and the secondary cost function does not play a role. The paths generated by MR-RRT* follow the medial axis of the free space before heading toward their respective goal regions. Figure 3.1 shows an example of trees in these three situations respectively, with 10,000 samples used in the construction of each tree.


(a) $RiskThreshold = \infty$


(b) $RiskThreshold = 0.2$

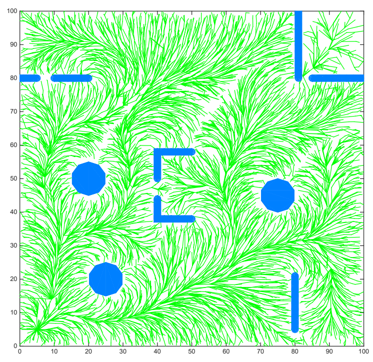

(c) $RiskThreshold = 0$

Figure 3.1: Trees generated by MR-RRT* under different values of *RiskThreshold*, when $\mathcal{R}(x)$ is the inverse distance transform.

3.1.3 Algorithm Analysis

3.1.3.1 Asymptotic Optimality

Here we state several key assumptions and lemmas governing the MR-RRT* algorithm. These assert that the algorithm possesses the required properties of the cost function $c(\sigma)$, established in the analysis of the original RRT* algorithm [17], to produce asymptotically optimal solutions, and that the problem space will admit asymptotically optimal solutions. We conclude with a theorem supported by these points that states the asymptotic optimality of MR-RRT* with respect to its primary cost function.

Assumption 1. *All problems of interest to be solved by MR-RRT* admit robustly optimal solutions.*

For the property of asymptotic optimality to apply (i.e., for MR-RRT* to converge almost surely toward an optimal solution in the limit), the problems solved by MR-RRT* must admit *robustly optimal solutions*, as defined in Section 3.1.1. Consequently, we will assume that all problems of interest have clearance between obstacles, and the risk penalty is defined such that no regions of measure zero exist in which risk penalties are absent.

Assumption 2. *The primary cost $c_{risk}(\sigma)$ of any path produced by MR-RRT* is bounded.*

We assume that the risk-based cost functions adopted in MR-RRT* cannot take on infinite values. When the inverse distance transform is adopted as the underlying risk function $\mathcal{R}(x)$ used in $c_{risk}(\sigma)$, a choice that will enforce clearance from obstacles, a limit must be imposed on the maximum value that $\mathcal{R}(x)$ is allowed to take on in the close vicinity of obstacles. If a finite maximum value is imposed on $\mathcal{R}(x)$, then $c_{risk}(\sigma)$ will be bounded.

Lemma 1. *The primary cost $c_{risk}(\sigma)$ of any path produced by MR-RRT* is monotonic.*

Proof. Monotonicity, as defined in the original analysis of RRT* [17], implies that $c(\sigma_1) \leq c(\sigma_1|\sigma_2)\ \forall\ \sigma_1, \sigma_2 \in \Sigma$, where $\sigma_1|\sigma_2$ is the concatenation of two paths such that they are joined together at the terminal configuration of σ_1 and the initial configuration of σ_2. The risk cost $Risk(x)$ at every individual configuration x that contributes to the cost of a path is a function of the geometric location of x only, by the function definitions in Equations 3.1 and 3.2. Hence, the concatenation of two paths σ_1 and σ_2 will not cause the individual costs of either path to change, and due to the use of an integral cost function, the cost of $\sigma_1|\sigma_2$ will simply be the sum of $c(\sigma_1)$ and $c(\sigma_2)$. Because the risk cost of every node and every path is non-negative, $c(\sigma_1)$ cannot exceed $c(\sigma_1|\sigma_2)$ in value. $\square$

Together these results and assumptions allow us to state the following:

Theorem 1 (Asymptotic Optimality of MM-RRT*)**.** *The MR-RRT* algorithm is asymptotically optimal.*

This is due to the fact that the requirements for MR-RRT* to inherit the asymptotic optimality properties of RRT* are satisfied with respect to the cost function, which must be bounded and monotonic, and satisfied with respect to the class of path planning problems considered, which must admit robustly optimal solutions.

3.1.3.2 Computational Complexity

We now comment on the computational complexity of MR-RRT*. We begin by reviewing the computational complexity of RRT*. Calls to the functions $Nearest()$ and $ObsFree()$, used in Algorithm 3.1 with the same frequency that they are used in RRT*, are typically the most expensive operations for large graphs that also require many geometric primitives to represent the surrounding obstacles. In a single iteration of both Algorithm 3.1 and standard RRT*, $\mathcal{O}(log(n))$ is the worst-case complexity of finding nearest neighbors, and $\mathcal{O}(log(n))$ total calls are made to $ObsFree()$. The time complexity of building a standard RRT* tree over n iterations is hence considered to be $\mathcal{O}(n\ log(n))$.

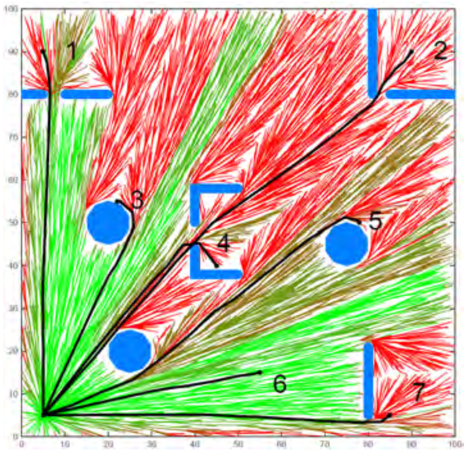
This complexity also holds for MR-RRT* with respect to these two operations. However, in the case of MR-RRT* we are also concerned with any additional computational expense due to evaluating a primary and secondary cost function. In a single iteration of MR-RRT*, the primary cost function will be evaluated over $\mathcal{O}(log(n))$ edges, equivalent to the number of calls made to $ObsFree()$. In the worst case, MR-RRT* also requires a second cost evaluation of every graph edge, by the secondary cost function, that would not occur in the case of standard RRT*. However, this is only a constant-factor increase in complexity, and does not change the original worst-case result. Hence, the time complexity of building the MR-RRT* tree over n iterations remains at $\mathcal{O}(n\ log(n))$, even if evaluating the cost of edges is considered to be the dominant time-consuming operation. This is equivalent to the complexity of the original RRT* algorithm.

3.1.4 Experiments

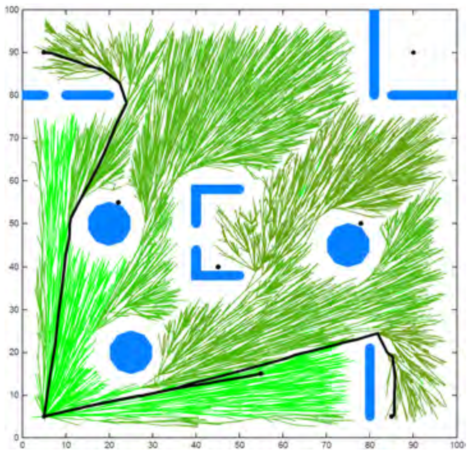
We employ a comparison of different algorithms to verify the effectiveness of path planning with MR-RRT*. We assume that a robot is moving in a 100-by-100 meter two-dimensional workspace, shown in Figure 3.2. This domain contains several obstacles depicted in blue. Seven goal regions, numbered from one to seven in Figure 3.2(a), are chosen to compare the results of the competing algorithms. Each goal region is a circle with radius 2 meters, and the robot is placed at $x_{init} = (5, 5)$ in all problem instances compared.

In these experiments, the inverse distance transform is adopted as the risk-based primary cost function, and we set $RiskThreshold = 0.2$. In other words, when the robot is greater than or equal to 5 meters away from the obstacles or boundaries of the domain, we consider such configurations to be safe for the robot, and they are not penalized. When the distance from the robot to the nearest obstacle or environment boundary is smaller than 5 meters, the risk threshold is surpassed and a penalty is imposed. All subsequent children and the branches descending from such configurations inherit a non-zero risk cost because they share the same a common ancestor exposed to risk. In Figure 3.2, nodes are plotted in colors varying from green (zero risk) to red (high risk) depicting the value of the primary cost function throughout the tree.

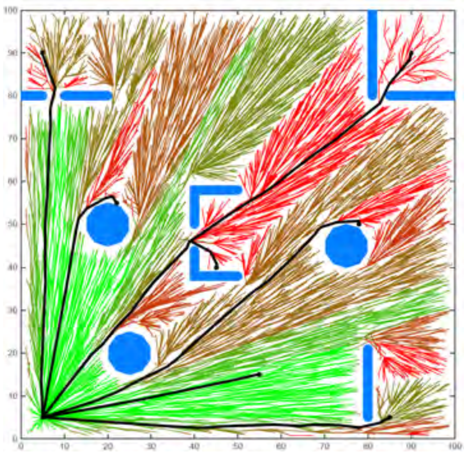
The first algorithm used in the comparison is RRT*. The results of path planning offered by RRT* are given in Figure 3.2(a). RRT* can produce shorter paths



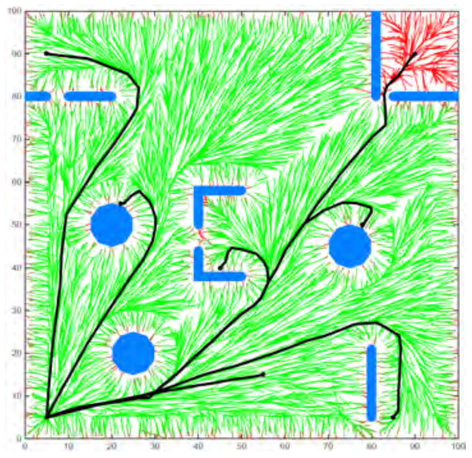
(a)



(b)



(c)



(d)

Figure 3.2: A comparison of the RRT*, T-RRT* and MR-RRT* algorithms in an obstacle-filled environment, with a risk cost equivalent to the inverse distance transform. The trees formed by RRT*, T-RRT* ($T_{rate} = 0.1$), T-RRT* ($T_{rate} = 0.5$) and MR-RRT* are shown in (a), (b), (c) and (d) respectively. Obstacles are depicted in blue. The best paths to seven representative goal regions from the root node are highlighted in black. Graph edges and nodes are colored from green to red to indicate the risk cost of each path.

compared with T-RRT* and MR-RRT*. However, the risk costs of paths produced by RRT* are the highest of all the methods explored. When the tree bypasses the obstacles in the domain, some nodes are very close to obstacles due to the fact that RRT* strictly minimizes the accumulated distance cost. All children of these nodes will share a parent with a high risk cost.

The second algorithm employed is T-RRT*. The tunable parameter T_{rate} : $(0, 1]$ regulates a rejection sampling criterion that governs the clearance of paths from surrounding obstacles, while the cost function penalizes accumulated distance. When $T_{rate} = 0.1$ in Figure 3.2(b), the paths generated by T-RRT* can always maintain safe distances from obstacles. But in this case, T-RRT* fails to offer feasible paths for some goal regions as it has a hard time landing samples in high-risk locations near the obstacles or boundaries of the environment. When $T_{rate} = 0.5$ in Figure 3.2(c), T-RRT* succeeds in reaching all goal regions after generating 10,000 nodes, but some paths have a higher risk cost than the paths with $T_{rate} = 0.1$.

The last algorithm compared is MR-RRT*. MR-RRT* can always find paths, which are shown in Figure 3.2(d), to reach goal regions that are feasible for RRT*, while maximizing the safety of these paths. The differences between MR-RRT* and T-RRT* are clear when we examine the effect of T-RRT*'s T_{rate} parameter. T-RRT* can behave like MR-RRT* when the value of T_{rate} is small enough, but in these instances T-RRT* fails to reach many goal regions. With a higher value of T_{rate}, T-RRT* can reach all goal regions. However, the paths obtained from T-RRT* with $T_{rate} = 0.5$ become dangerous as a result. For example, the path which connects the start position and goal region 5, which lies behind a circular obstacle, is dangerous since it includes a node that is close to the first obstacle met. MR-RRT* can identify feasible paths to all goal regions while keeping a safe distance from obstacles. When approaching goal regions 3 and 5, MR-RRT* tries to avoid getting too close to the obstacles before reaching destinations. When approaching goal regions 1 ,4 and 7, MR-RRT* selects a detour to avoid narrow corridors.

Further comparisons are made in a terrain map shown in Figure 3.4. The robot travels in a 180 by 360 by 50 meter domain shown in Figure 3.3. The risk cost is

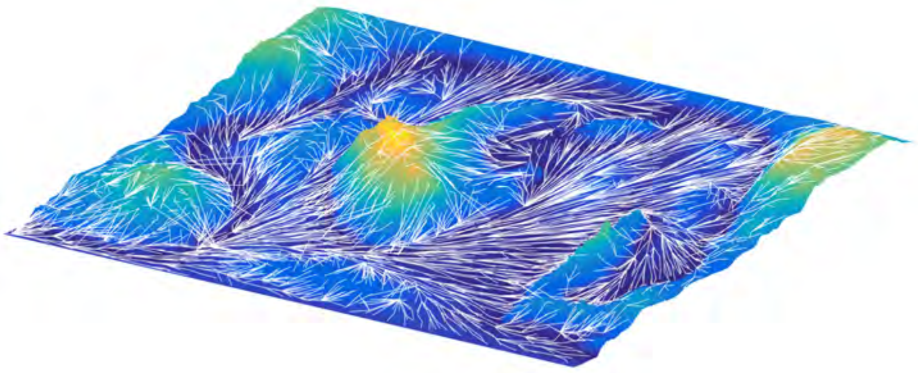
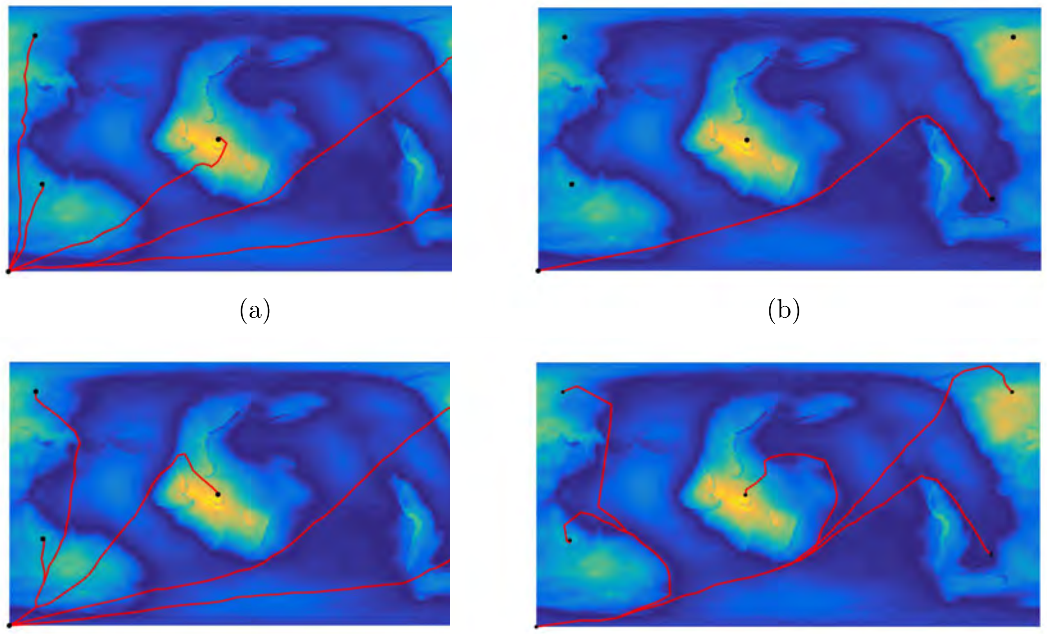


Figure 3.3: An MR-RRT* tree in a terrain map that penalizes altitude at all altitudes higher than $RiskThreshold$. Terrain that is not penalized is depicted in purple.



(a)

(b)

(c)

(d)

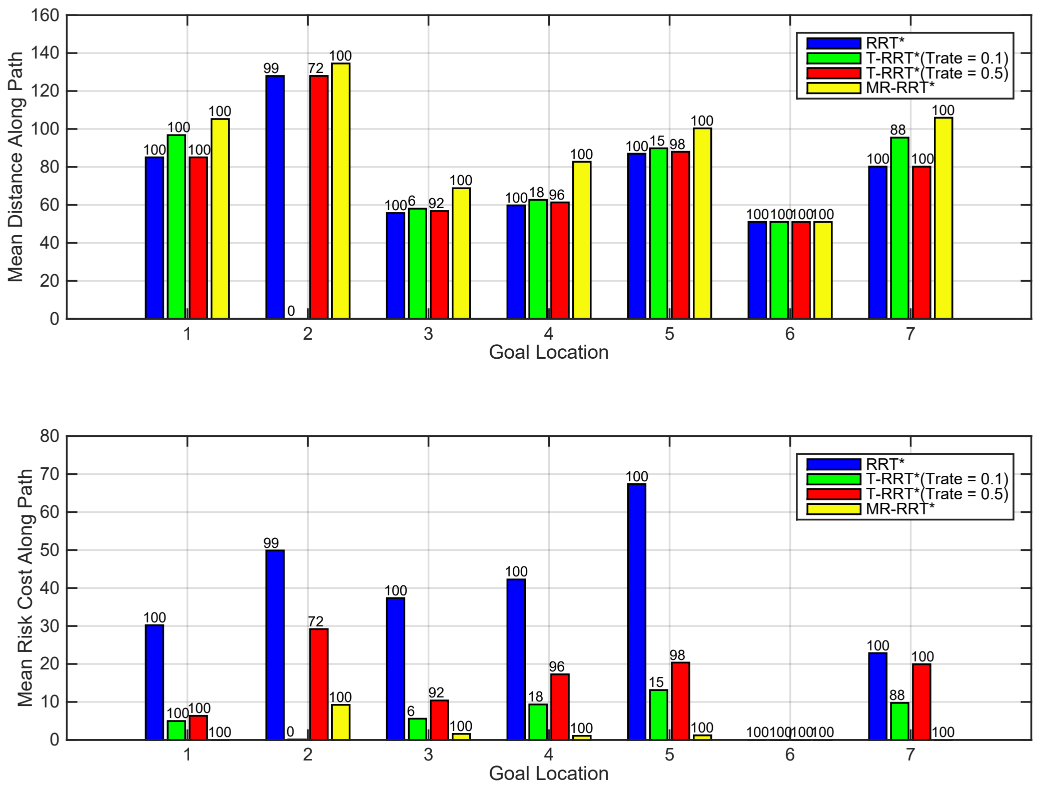
Figure 3.4: Paths produced by RRT*, T-RRT* ($T_{rate} = 0.1$), T-RRT* ($T_{rate} = 0.9$) and MR-RRT* are shown in (a), (b), (c) and (d) respectively over a terrain map, for five example goal regions. Each tree used to obtain the paths was constructed using 10,000 samples. The colors of the terrain map depict altitude, which serves as the risk representation used in this example, with yellow denoting high altitude.

dictated by the altitude of robot, as we assume that high altitudes expose the robot to threats. In these experiments, we set $Risk_{threshold} = 5$. Regions with elevation below 5 meters (depicted in purple) are considered safe. Traveling in the regions with elevation above 5 meters increased the possibility being detected. The color of the domain changes from purple to yellow as the altitude increases.

The paths given by the three algorithms in the terrain map are similar to the previous results given in 2-D environments. RRT* travels over high elevation regions to obtain minimum distance paths. T-RRT* can travel while avoiding being detected in low elevation regions with a smaller T_{rate}, but it has difficulties reaching the goal regions of high elevation. This is because T-RRT* with a low T_{rate} will reject the new samples landing in the higher elevation regions, and this results in these regions being unexplored. The probability of arriving at a high elevation region increases with a larger T_{rate}, but the safety of the path becomes worse. Since we are testing T-RRT* in different environments, T_{rate} must be adjusted to obtain less conservative results, in contrast to the parameters depicted in Figure 3.2. When T_{rate} is increased to 0.9, satisfactory results are obtained, although they are exposed to high risk. The tree generated by MR-RRT*, however favors travel through low-altitude valleys where possible. MR-RRT* also has no trouble arriving at any of the goal regions in $\mathcal{C}_{free}$ that are also reached by RRT*.

In order to perform a fair and comparable evaluation for these three algorithms, we perform 100 trials for each algorithm in the same environment with same start and goal regions. Each trial includes 10,000 samples in the tree. When calculating the risk cost for a path, all three algorithms use same $RI()$ and $SI()$ functions to achieve an impartial comparison.

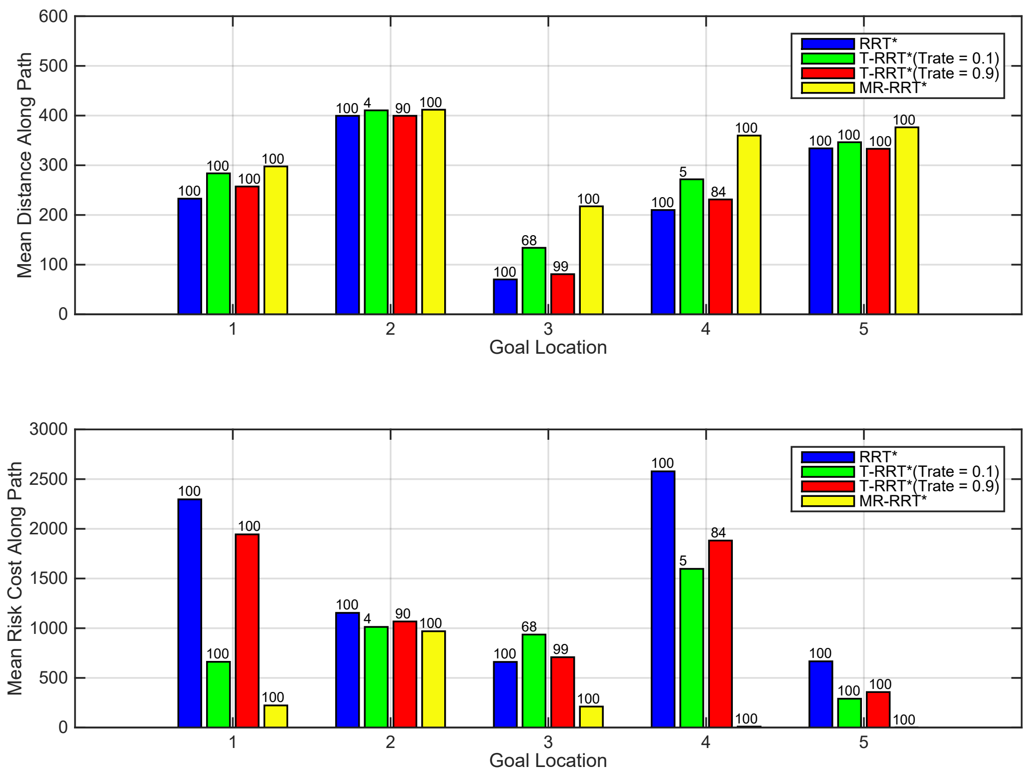
The results depicted in Figure 3.5 come from the solutions in 2-D obstacle-filled environments shown in Figure 3.2. The mean distance and mean risk cost of 7 representative goal regions selected in Figure 3.2 are used for this test. The number above each column shows the times of a goal region is reached after 100 trials. In



(b)

Figure 3.5: The mean accumulated distance cost and mean risk cost to the seven marked goal regions in the workspace of Figure 3.2 is depicted, averaged over 100 trials.

Figure 3.5(a), the cumulative distance costs using each algorithm are quantified for each of the seven goal regions. It is obvious that RRT* outperforms the other two algorithms by offering shorter paths. When $T_{rate} = 0.1$, T-RRT* is very conservative and fails to reach goal region 2 after 100 trials. Goal regions 3, 4 and 5 are only reached 6, 18 and 15 times respectively. When $T_{rate} = 0.5$, T-RRT* becomes less conservative and the numbers of reaching different goal regions increase. All three algorithms have zero risk cost when arriving at goal region 6 since it is located in safe regions. The risk cost along paths are presented in Figure 3.5(b). MR-RRT* has obvious advantages over standard RRT* and T-RRT* by offering lower risk cost. The



(b)

Figure 3.6: The mean accumulated distance cost and mean risk cost to the five marked goal regions in the terrain map of Figure 3.4 is depicted, averaged over 100 trials.

mean risk cost of seven paths is decreased by 94.75% compared with RRT*, 69.7% compared with T-RRT* with $T_{rate} = 0.1$ and 87.3% compared with T-RRT* with $T_{rate} = 0.9$ by using MR-RRT*.

Similar results are obtained and shown in Figure 3.6 after performing each algorithm in a terrain map over 100 trials for five representative goal regions. RRT* can offer the shortest paths for each goal region. MR-RRT* can always offer the lowest risk cost paths. The mean risk cost of four paths is decreased by 78.86% compared with RRT*, 66.39% compared with T-RRT* with $T_{rate} = 0.1$ and 74.75% compared with T-RRT* with $T_{rate} = 0.9$ by using MR-RRT*.

Conservative T-RRT* with a low T_{rate} can offer solutions with high clearance, but it encounters difficulty reaching goal regions in risky areas. T-RRT* with a high T_{rate} is less conservative, but the risk cost of path is increased dramatically. The ability to obtain feasible, low-risk paths with MR-RRT* is a compelling alternative. Our computational trials were performed with the use and modification of the RRT* code provided by the authors of [17].

3.1.5 Conclusions

In this section, we have introduced a new path planning algorithm, MR-RRT*, that allows planning to and through high-risk areas of the configuration space using a tunable hierarchy of primary and secondary cost functions, while retaining the property of asymptotic optimality in the primary cost function. By placing risk in the primary objective, rather than addressing it through a sample rejection constraint, MR-RRT* will land samples in areas that typically cannot be reached by T-RRT* when the safest route possible is desired. A compromise between safety and efficiency can be achieved through adjustment of the tunable risk threshold, without sacrificing thorough coverage of the configuration space.

3.2 Min-Max Uncertainty Planning

In this section, we introduce a min-max variant of the RRT* algorithm we term the Min-Max Rapidly Exploring Random Tree, or MM-RRT*. In addition to utilizing a min-max uncertainty metric, which will be detailed in the sections to follow, the algorithm builds and maintains a tree that is shared in state space and belief space, with a singe belief per robot state. The algorithm also uses secondary objective functions to break ties among neighboring nodes with identical maximum uncertainty.

3.2.1 Problem Definition

3.2.1.1 Path Planning:

The problem definition is similar to the definition mentioned in Section 3.1.1. Let $\mathcal{C}$ be a robot's configuration space. We will assume that a configuration $x \in \mathcal{C}$ describes the pose of the robot. $\mathcal{C}_{obst} \subset \mathcal{C}$ denotes the subset of configurations in $\mathcal{C}$ that are in collision with an obstacle, and $\mathcal{C}_{free} \subseteq (\mathcal{C} \setminus \mathcal{C}_{obst})$ denotes the configurations that are free of collision. We will assume that given an initial configuration $x_{init} \in \mathcal{C}_{free}$, the robot must reach a goal region $X_{goal} \subset \mathcal{C}_{free}$. Let a *path* be a continuous function $\sigma : [0, T] \to \mathcal{C}$ of finite length, traversed in finite time T, for which we assume a series of control inputs must be applied to the robot to achieve this path in $\mathcal{C}$. Let Σ be the set of all paths σ in a given configuration space. A path is collision-free if $\sigma \in \mathcal{C}_{free}$ for all arguments, and a collision-free path is *feasible* if $\sigma(0) = x_{init}$ and $\sigma(T) \in X_{goal}$. A *cost function* $c : \Sigma \to \mathbb{R}_+$ returns a strictly positive cost for all non-trivial collision-free paths.

3.2.1.2 State Estimation and Uncertainty

We consider a robot whose state evolves as a nonlinear, discrete-time dynamical system:

$$\mathrm{x}_{k+1} = \mathbf{f}(\mathrm{x}_k, \mathrm{w}_k) \tag{3.4}$$

$$\mathrm{y}_k = \mathbf{h}_k(\mathrm{x}_k, \mathrm{v}_k) \tag{3.5}$$

where $\mathrm{x}_k = \begin{bmatrix} \dot{x}_k & x_k \end{bmatrix}$ is the state of the system at time k, and describes the position and velocity of the robot in each degree of freedom. The robot's measurement at time k is y_k, and is supplied by a suite of sensors whose availability will vary as a function of the robot's location in the environment. At time k, the robot is influenced by independent zero-mean Gaussian process noise w_k, with covariance $\mathbf{Q}_k$, and sensor noise v_k, with covariance $\mathbf{R}_k$.

The robot's state may be estimated using an extended Kalman filter (EKF) [122], which is propagated as follows:

$$\mathbf{P}_{k+1}^{-1} = (\mathbf{F}_k\mathbf{P}_k\mathbf{F}_k' + \mathbf{Q}_k)^{-1} + \mathbf{H}_{k+1}'\mathbf{R}_{k+1}^{-1}\mathbf{H}_{k+1} \tag{3.6}$$

$$\hat{\mathrm{x}}_{k+1} = \mathbf{P}_{k+1}\big((\mathbf{F}_k\mathbf{P}_k\mathbf{F}_k' + \mathbf{Q}_k)^{-1}\mathbf{F}_k\hat{\mathrm{x}}_k + \mathbf{H}_{k+1}'\mathbf{R}_{k+1}^{-1}(y_{k+1} - \mathbf{H}_{k+1}\hat{\mathrm{x}}_k)\big) \tag{3.7}$$

where $\hat{\mathrm{x}}_k$ is the state estimate, $\mathbf{P}_k$ is the robot's estimation error covariance at time k, and $\mathbf{F}_k$ and $\mathbf{H}_k$ represent the Jacobians of $\mathbf{f}$ and $\mathbf{h_k}$ about $(\hat{\mathrm{x}}_k, 0)$ and $(f(\hat{\mathrm{x}}_k), 0)$, respectively. To penalize growth of the error covariance in the course of path planning, estimation error is ideally represented as a scalar cost metric. This has been achieved previously in the context of sampling-based planning algorithms using $tr(\mathbf{P})$, the error covariance trace [46], and $\overline{\lambda}(\mathbf{P})$, the maximum eigenvalue of $\mathbf{P}$ [47].

However, this latter work also proposes an upper bound on $\overline{\lambda}(\mathbf{P})$, $\ell \geq \overline{\lambda}(\mathbf{P})$, which, unlike the above metrics, admits optimal substructure when applied to the breadth-first search of a graph in belief space. A recursive update for ℓ results from the following inequality, which is derived from Equation 3.7:

$$\overline{\lambda}(\mathbf{P}_{k+1}) \leq \frac{\overline{\lambda}(\mathbf{F}_k\mathbf{F}'_k)\overline{\lambda}(\mathbf{P}_k) + \overline{\lambda}(\mathbf{Q}_k)}{(\underline{\lambda}(\mathbf{H}'_k\mathbf{R}_k^{-1}\mathbf{H}_k))(\overline{\lambda}(\mathbf{F}_k\mathbf{F}'_k)\overline{\lambda}(\mathbf{P}_k) + \overline{\lambda}(\mathbf{Q}_k)) + 1} \tag{3.8}$$

where the operator $\underline{\lambda}(\ )$ represents the minimum eigenvalue of a positive definite matrix. Due to this inequality, the upper bound $\ell \geq \overline{\lambda}(\mathbf{P})$ may be propagated according to the following update rule, initialized using $\ell_0 = \overline{\lambda}(\mathbf{P}_0)$:

$$\ell_{k+1} = \frac{\overline{\lambda}(\mathbf{F}_k\mathbf{F}'_k)\ell_k + \overline{\lambda}(\mathbf{Q}_k)}{(\underline{\lambda}(\mathbf{H}'_k\mathbf{R}_k^{-1}\mathbf{H}_k))(\overline{\lambda}(\mathbf{F}_k\mathbf{F}'_k)\ell_k + \overline{\lambda}(\mathbf{Q}_k)) + 1}. \tag{3.9}$$

Using ℓ as an uncertainty metric offers a small improvement in computational efficiency over other metrics, since the propagation of uncertainty in Equation 3.9 uses extreme eigenvalues [123], rather than the repeated computation of $\mathbf{P^{-1}}$ required in Equation 3.6. We thus adopt ℓ as our scalar representation of uncertainty, although the proposed MM-RRT* algorithm may be used with any scalar uncertainty metric.

3.2.1.3 Robot Motion Assumptions

We assume a robot moves through $\mathcal{C}_{free}$ along paths obtained from a directed graph $G(V, E)$, with vertices V and edges E. Specifically, all graphs considered are *trees*. We make several assumptions regarding the consistency, stability, and synchrony of the estimation, motion, and measurement processes, adapted from [19]:

1. The filter produces a consistent estimate, and the corresponding error covariance is a measure of the precision of the filter.

2. There exists a low-level controller that ensures the robot follows the planned, nominal trajectory between any two vertices of the graph.

3. The measurements from all sensors are synchronized.

The first assumption implies that our motion and measurement models accurately represent the behavior of the robot. The second assumption implies that the robot is capable of recovering from process noise disturbances. The third assumption implies that measurements from all sensors will arrive in the order that their observations occur, such that a planning algorithm may predict their arrival. Our goal is to leverage our knowledge of the environment, motion and sensor models to prevent the growth of uncertainty to an extent that the above assumptions may be violated.

In general, a robot's state and uncertainty comprise a *belief*, (x, ℓ). A vertex of an arbitrary graph may represent many beliefs achieved over different motion and measurement histories. In a tree, however, there is only one path from the root x_{init} to each node, and if all paths are initialized from the root with the same uncertainty $\ell_{init} = \overline{\lambda}(\mathbf{P}_{init})$, there exists only one belief per node. Hereafter, when referring to a configuration x_k, we imply that x_k has an associated error covariance $\mathbf{P}_k$, an eigenvalue bound ℓ_k, a dynamical state x_k, and state estimate $\hat{\mathrm{x}}_k$ that are uniquely identified by x_k in concert with the initial uncertainty at the root of the tree.

3.2.1.4 Cost Function

To aid in defining our cost function, we first define $G(\ell_0)|_{\sigma(0)}^{\sigma(T)}$, a function that represents the composition of evaluations of Equation 3.9 over a path from $\sigma(0)$ to $\sigma(T)$, where ℓ_0 represents the uncertainty at the start of the path. Every discrete-time instant along the path represents a measurement update of the EKF. Our proposed cost function, $c_{max}(\sigma, \ell_0)$, then scores a path by evaluating G from the start of the path, where the intial uncertainty is ℓ_0, to every measurement update along the path, returning the maximum.

$$G(\ell_0)|_{\sigma(0)}^{\sigma(T)} := (\ell_T \circ \ell_{T-1} \circ ... \circ \ell_1)(\ell_0) \tag{3.10}$$

$$c_{max}(\sigma, \ell_0) := max\{G(\ell_0)|_{\sigma(0)}^{\sigma(T)}, G(\ell_0)|_{\sigma(0)}^{\sigma(T-1)}, \tag{3.11}$$
$$..., G(\ell_0)|_{\sigma(0)}^{\sigma(2)}, G(\ell_0)|_{\sigma(0)}^{\sigma(1)}\}$$

We will hereafter abbreviate $c_{max}(\sigma, \ell_0)$ using the notation $\overline{\ell}\,|_{\sigma(0)}^{\sigma(T)}$ to indicate the start and end states of the specific path evaluated, from which the maximum uncertainty $\overline{\ell}$ is returned. The algorithm we propose below builds a tree that minimizes $\overline{\ell}$ from root node x_{init}, initialized with uncertainty $\ell_{init} = \overline{\lambda}(\mathbf{P}_{init})$, to all destinations in $\mathcal{C}_{free}$.

In addition, we define a secondary cost and tertiary cost, which are integral cost functions, as follows:

$$c_{second}(\sigma) := \int_{\sigma(0)}^{\sigma(T)} Second(\sigma(s))ds \tag{3.12}$$

$$c_{third}(\sigma) := \int_{\sigma(0)}^{\sigma(T)} Third(\sigma(s))ds \tag{3.13}$$

where $Second(x)$ and $Third(x)$ are generalized functions of a single robot state. Each subsequent cost function in our hierarchy will be used for breaking ties that occur in the former. In the examples explored in our computational results below, all workspaces are comprised of two types of spatial regions: regions in which the robot's pose is not directly observable, in which the pose uncertainty grows under dead reckoning, and regions in which the robot's pose is directly observable (via GPS, the measurement of environmental features, or similar), where error growth is curbed. In these examples, $c_{second}(\sigma)$ returns the distance traveled by a path σ in all regions that lack observability of the robot pose, and $c_{third}(\sigma)$ returns the distance traveled by a

Algorithm 3.3: MM-RRT*

```
   Input: $V \leftarrow \{x_{init}\}; E \leftarrow \emptyset$;
1  for $i = 1, ..., n$ do
2      $x_{rand} \leftarrow Rand()$ ;
3      $x_{nearest} \leftarrow Nearest(V, x_{rand})$;
4      $x_{new} \leftarrow Steer(x_{nearest}, x_{rand})$;
5      if $ObsFree(x_{nearest}, x_{new})$ then
6          $X_{near} \leftarrow Near(V, E, x_{new}); V \leftarrow V \cup \{x_{new}\}$;
7          $X_{min} \leftarrow \{x_{nearest}$;
8          $c_{min} \leftarrow MaxBnd(x_{nearest}, x_{new})$;
9      for $x_{near} \in X_{near}$ do
10         if $ObsFree(x_{near}, x_{new})$ then
11             if $MaxBnd(x_{near}, x_{new}) < c_{min}$ then
12                 $X_{min} \leftarrow \{x_{near}\}$;
13                 $c_{min} \leftarrow MaxBnd(x_{near}, x_{new})$;
14             else if $MaxBnd(x_{near}, x_{new}) = c_{min}$ then
15                 $X_{min} \leftarrow X_{min} \cup \{x_{near}\}$;
16     $x_{min} \leftarrow Tiebreak(X_{min}, x_{new})$;
17     $E \leftarrow E \cup \{(x_{min}, x_{new})\}$;
18     for $x_{near} \in X_{near} \setminus \{x_{min}\}$ do
19         $replace \leftarrow false$;
20         if $ObsFree(x_{new}, x_{near})$ then
21             $c_{near} \leftarrow Cost(x_{near})$;
22             $c_{new} \leftarrow MaxBnd(x_{new}, x_{near})$;
23             if $c_{new} < c_{near}$ then $replace \leftarrow true$; ;
24             else if $c_{new} = c_{near}$ then
25                 $X_{pair} \leftarrow \{x_{parent}, x_{new}\}$;
26                 if $x_{new} = Tiebreak(X_{pair}, x_{near})$ then
27                     $replace \leftarrow true$;
28         if $replace = true$ then
29             $E \leftarrow (E \setminus \{(x_{parent}, x_{near})\}) \cup \{(x_{new}, x_{near})\}$;
30             $X_{children} \leftarrow Children(x_{near})$;
31             $RecursivePropagate(x_{near}, X_{children})$;
32 return $G = (V, E)$
```

path in all regions that offer such observability. We will alternately express $c_{second}(\sigma)$ using the notation $SI(\sigma(0), \sigma(T))$ and $c_{third}(\sigma)$ using the notation $TI(\sigma(0), \sigma(T))$ to indicate the limits of each integral.

Algorithm 3.4: $Tiebreak(X_{tied}, x_{child})$

```
1  $c_{second} \leftarrow +\infty$; $c_{third} \leftarrow +\infty$;
2  for $x_i \leftarrow X_{tied}$ do
3      if $SI(x_{init}, x_i) + SI(x_i, x_{child}) < c_{second}$ then
4          $c_{second} \leftarrow SI(x_{init}, x_i) + SI(x_i, x_{child})$;
5          $x_{min} \leftarrow x_i$;
6      else if $SI(x_{init}, x_i) + SI(x_i, x_{child}) = c_{second}$ then
7          if $TI(x_{init}, x_i) + TI(x_i, x_{child}) < c_{third}$ then
8              $c_{third} \leftarrow TI(x_{init}, x_i) + TI(x_i, x_{child})$;
9              $x_{min} \leftarrow x_i$;
10 return $x_{min}$
```

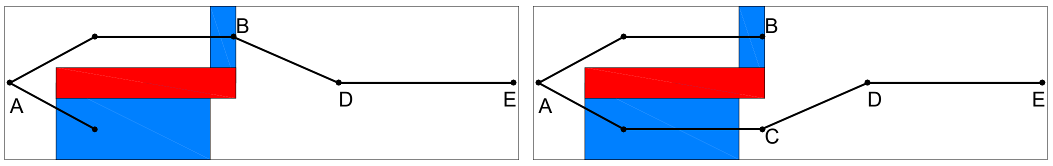
3.2.2 Algorithm Description

MM-RRT* is outlined in Algorithm 3.3. The algorithm proceeds similarly to RRT*, beginning each iteration by drawing a random sample, steering toward the random sample from the nearest neighboring node in the existing tree, and subsequently searching for the best parent for the newly-generated candidate node by examining all neighbors within a ball radius that shrinks logarithmically according to $\gamma(log(card(V))/card(V))$, where $\gamma > 2(1 + 1/d)^{1/d}(\mu(\mathcal{C}_{free})/\mu^d_{ball})^{1/d}$. The volume of the d-dimensional unit ball in Euclidean space is indicated by μ^d_{ball}. The function $MaxBnd(x_{near}, x_{new})$ is used to evaluate the maximum uncertainty encountered while traveling to x_{new} from x_{init} on a path that includes x_{near}. To evaluate $MaxBnd(x_{near}, x_{new})$, we need only propagate Equation 3.9 from x_{near} to x_{new}, and the maximum error covariance eigenvalue bound encountered, $\overline{\ell}\,|^{x_{new}}_{x_{near}}$, is compared to $\overline{\ell}\,|^{x_{near}}_{x_{init}}$, the stored maximum value associated with parent node x_{near}. The larger of the two values is returned by $MaxBnd(x_{near}, x_{new})$, and the resulting min-max after a survey of X_{near} is adopted as $\overline{\ell}\,|^{x_{new}}_{x_{init}}$, and stored as the cost associated with x_{new}. Because belief propagation is occurring on a tree, we will only have a single belief per node and there is no need to distinguish between geometric states x_i and their associated beliefs.

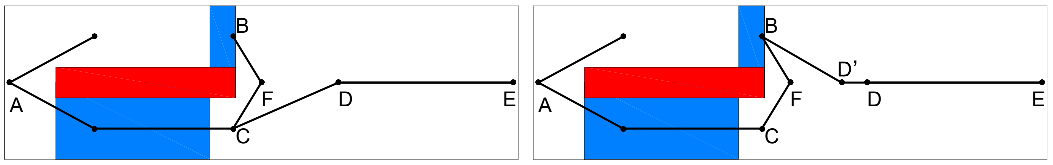
The min-max objective function will frequently result in ties among candidate nodes in X_{near}. This occurs when a maximum uncertainty value is derived from a common ancestor node and shared among multiple descendants. MM-RRT* implements tie-breaking using secondary and tertiary objective functions to impose an ordering scheme among nodes of identical maximum uncertainty. This is indicated by the use of the $Tiebreak()$ function in Algorithm 3.4. In the implementation of MM-RRT* discussed here, c_{second}, the cumulative distance traveled in the regions where no GPS measurements are available, and c_{third}, the cumulative distance traveled in the regions where GPS measurements are available, are used as the secondary and tertiary objectives. Consequently, if two nodes, x_1 and x_2, are equally suitable parents of x_{new} by the min-max objective, the node offering the minimum c_{second} to x_{new} will be selected as the parent. If x_1 and x_2 offer the same primary *and* secondary cost to x_{new}, then the node offering the minimum c_{third} to x_{new} will be selected as the parent.

After a suitable parent is identified for x_{new}, both RRT* and MM-RRT* next attempt to improve the cost of the other nodes in X_{near} by considering the replacement of their parents by x_{new}, known as rewiring. This is achieved by evaluating $MaxBnd(x_{new}, x_{near})$ and comparing the result with the existing value of $\overline{\ell}\,|_{x_{init}}^{x_{near}}$. If x_{new} offers a lower max uncertainty cost, it will replace the parent of x_{near}. If x_{new} ties with the existing parent, then the $Tiebrk()$ function is once again used to break the tie and choose a parent for x_{near}.

If the rewiring procedure succeeds in replacing the parent of x_{near}, the costs of all descendants of x_{near} must be updated. Unlike the standard RRT* algorithm, the min-max costs are not additive from node to node and the bound $\overline{\ell}$ must be propagated anew from x_{near} to all of its descendants. This is performed by the



(a) Initial tree (b) Rewiring after C is sampled



(c) Rewiring after F is sampled (d) Rewiring after D' is sampled

Figure 3.7: This example shows an instance in which a child node's (E) max uncertainty will first increase, then subsequently decrease, after a series of rewiring operations that are favorable for E's parent node, D. At top, path A-B-D-E will be rewired to A-C-D-E upon the sampling of node C, and at bottom, path A-C-D-E will be rewired to A-C-F-B-D'-D-E upon the sampling of nodes F, then D'. Blue regions contain GPS availability, with an obstacle depicted in red.

$RecursivePropagate()$ function, which recursively propagates the bound from the parent to all children until all branches of the tree that descend from x_{near} have been updated. The impact of this procedure on the algorithm's asymptotic optimality and computational complexity is discussed in the following section.

3.2.3 Algorithm Analysis

3.2.3.1 Monotonicity of the Min-Max Cost Metric

Unlike the standard RRT* algorithm, in MM-RRT*, it is possible for a rewiring operation, despite lowering the cost at a parent node, to cause the cost at one or more child nodes downstream from the parent to increase. A situation in which this can arise is illustrated in Figure 3.7, where an obstacle is depicted in red, regions of GPS availability are depicted in blue, and in all other regions, the robot, tasked with planning a path from A to E, must navigate using odometry. Upon the sampling of node C, a rewiring operation is performed, in which B, the original parent of node D, is replaced with C, a new parent offering lower $\bar{\ell}$, eliminating edge B-D. Despite the fact that a lower max uncertainty exists at node C, it also has a higher current uncertainty than node B. As a result, further downstream along the path from A to E, the current uncertainty at node C will eventually give rise to a new value of max uncertainty, which causes both the current uncertainty and max uncertainty at node E to increase. However, this increase in max uncertainty at E is eventually undone. When node F is added to the tree, it becomes the parent node of B through rewiring. Subsequently, node D' is added to the tree, and this triggers the rewiring of D, eliminates edge C-D, and reduces both the current uncertainty and the max uncertainty at node E to the lowest levels yet.

We first address why this occurs, given that the min-max cost function appears to satisfy the two most commonly articulated properties for an admissible RRT* cost metric: *monotonicity*, as the cost along a path is monotonic non-decreasing, and *boundedness*, as the cost cannot instantaneously take on arbitrarily large values. However, a property that the min-max metric lacks is monotonicity with respect to the *initial value* of the cost metric at the beginning of a path. Consider the path from D to E. Although, at top of Figure 3.7, parent node C offers a lower min-max cost going into node D than its previous parent B, it results in higher min-max cost coming out at node E. This second type of monotonicity is trivial in the case of most cost metrics, and as such, it has not been articulated in prior analysis of RRT*. However, this property does not hold for MM-RRT* in examples where a path's uncertainty undergoes a reset, such as by entering a GPS zone like those of Figure 3.7. Rewiring a node's parent to improve the child node's uncertainty does not necessarily improve the uncertainty of all of that child's descendants.

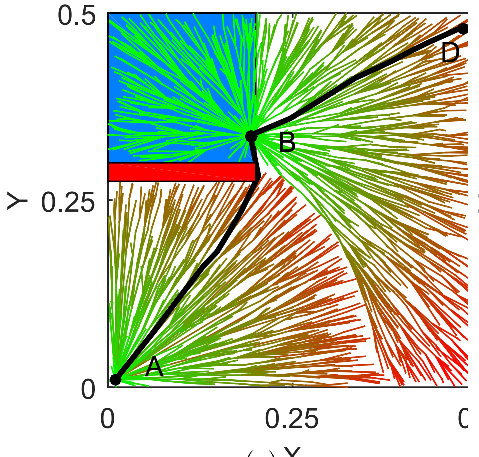
Despite this, as $\mathcal{C}$ is populated with more nodes over the course of the MM-

RRT* algorithm, these occasional instances of “bad rewiring” will be mitigated through subsequent rewiring of child nodes like E that are adversely affected by earlier events, as illustrated at bottom of Figure 3.7 after the new nodes F and D’ are added to the graph. In our computational results below, we provide empirical support that the limiting behavior of the algorithm is for path costs to converge asymptotically, toward optimal values. However, we cannot claim that these asymptotes represent globally optimal solutions. Unlike the standard RRT* algorithm, a single instance of rewiring will not cause all affected nodes to improve in cost simultaneously; there will be fluctuations in cost with a net decrease that tends toward an asymptote. In future work, we hope to analyze the global optimality of tree-based algorithms that can undergo occasional setbacks en route to optimal solutions.

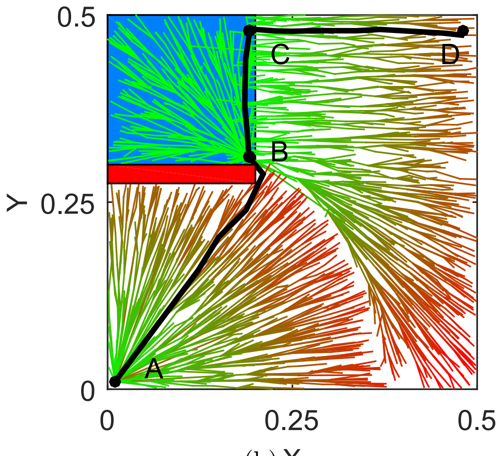
3.2.3.2 Computational Complexity

We now comment on the computational complexity of MM-RRT*. For a typical sampling-based path planning algorithm that does not propagate beliefs over the graph, the complexity of the $Near()$ operation and the number of calls made to $ObsFree()$ are typically of greatest concern, as these are expensive operations for large graphs that require many geometric primitives to represent the surrounding obstacles. In the case of RRT*, in a single iteration $\mathcal{O}(log(n))$ is the worst-case complexity of finding nearest neighbors, and $\mathcal{O}(log(n))$ total calls are made to $ObsFree()$. The time complexity of building a tree over n iterations is hence considered to be $\mathcal{O}(n\ log(n))$.

This complexity also holds for MM-RRT* with respect to these two operations. However, in the case of MM-RRT* we are also concerned about the number times Equation 3.9 will be propagated across an edge of the graph. This will happen $\mathcal{O}(n\ log(n))$ times if we count only the belief propagations that are used to evaluate candidate edges. These are the “expensive” belief propagation operations, occurring



(a)



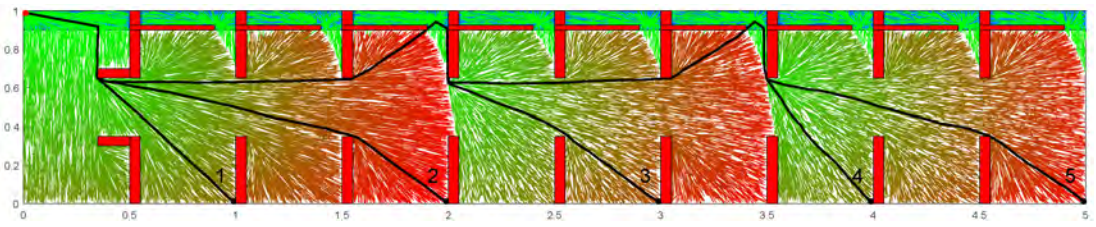
(b)

Figure 3.8: When only one secondary cost function $Dist()$ is used to penalize distance traveled, the resulting tree and a path connecting start and goal nodes are shown in (a). When a secondary cost function $SI()$ and a tertiary cost function $TI()$ are used together, the resulting tree and respective path are shown in (b). This approach curbs the growth of uncertainty at D. The blue region indicates GPS availability, with an obstacle in red. Edges of the trees are plotted in color varying from green to red to indicate the current value of the eigenvalue bound ℓ (with higher ℓ in red) along each path.

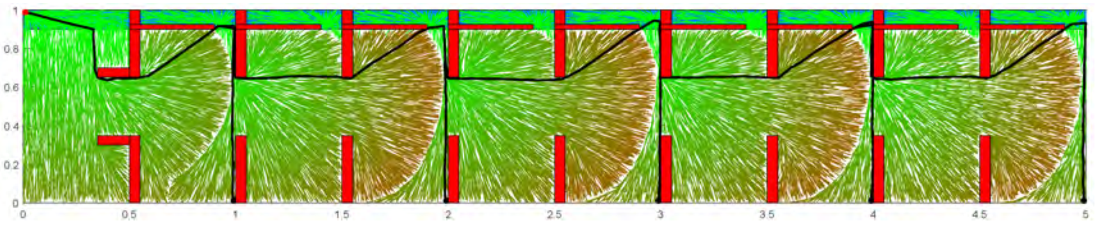
the first time an edge cost is computed. In addition, there are another worst-case $\mathcal{O}(n^2)$ propagations that will occur due to the calls made to $RecursivePropagate()$, in which new beliefs are propagated over pre-existing graph edges. These propagations may occur at substantially lower cost, as the min and max eigenvalue terms in Equation 3.9 will have already been evaluated over an edge. Consequently, MM-RRT* requires $\mathcal{O}(n\ log(n))$ non-trivial belief propagation operations over new and candidate graph edges, which is the same worst-case number of candidate edge evaluations required by RRT*.

3.2.3.3 On the Use of Hierarchical Cost Functions

Secondary and tertiary cost functions are used to curb additional growth of uncertainty when paths appear otherwise identical in primary cost. We can express the



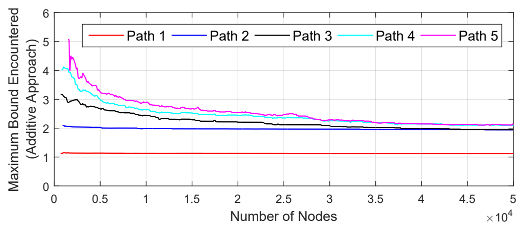
(a) Additive Approach



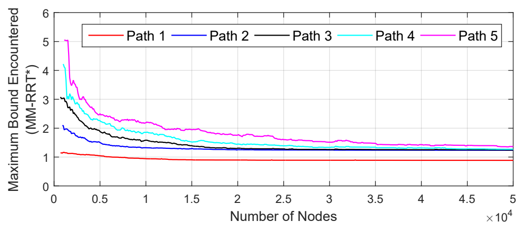
(b) MM-RRT*

Figure 3.9: Trees generated by two competing approaches for planning under uncertainty using the same sequence of random samples. Obstacles are indicated in red and GPS zones (at the upper boundary of the domain) are rendered in blue. Edges of the trees are plotted in color varying from green to red to indicate the current value of the eigenvalue bound ℓ (with higher ℓ in red) along each path.

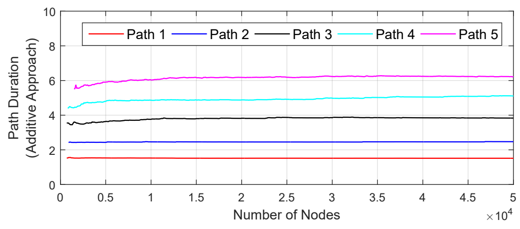
total accumulated distance along a path as $Dist(\sigma(0), \sigma(T)) = SI(\sigma(0), \sigma(T)) + TI(\sigma(0), \sigma(T))$ using these cost functions. The benefit of this approach is illustrated in Figure 3.8, in which there are four key tree nodes highlighted: A, the start node, B, the first node arriving at the GPS region, C, a node in the GPS zone which has the same y-axis value as node D, and D, the goal node. At node B, the path of interest enters the GPS region, and node B and its descendants will each have substantially lower localization uncertainty than the max uncertainty encountered a few steps earlier. As a result, ties exist among the children of node B, as they share the same primary cost. If only a secondary cost function $Dist()$ is applied here, the path to node D will not travel through node C, as a direct route from B to D will offer the shortest distance (shown in Figure 3.8(a)). However, we care more about the value of ℓ over the path than the path's length. When a tertiary cost function $TI()$



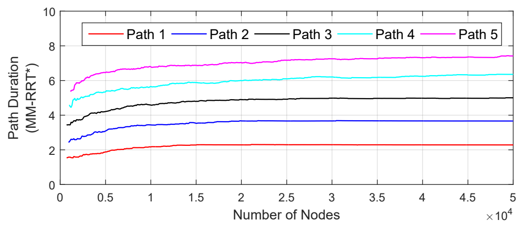
(a) Max ℓ encountered, Additive Approach



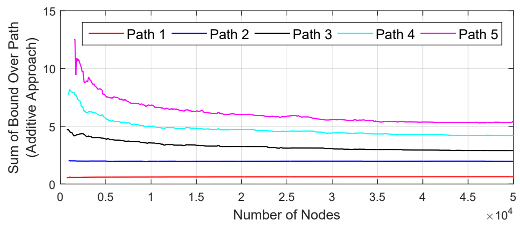
(b) Max ℓ encountered, MM-RRT*



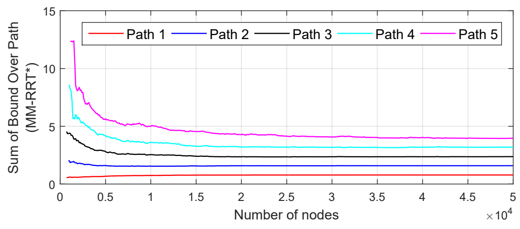
(c) Path duration, Additive Approach



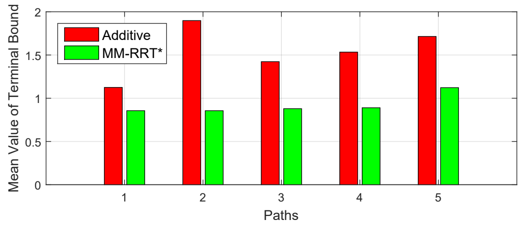
(d) Path duration, MM-RRT*



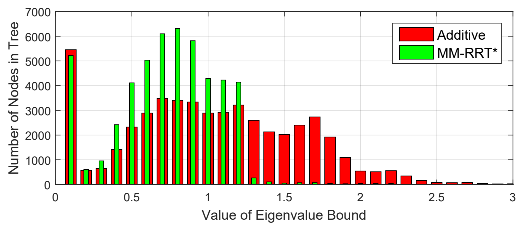
(e) Sum of ℓ over path, Additive Approach



(f) Sum of ℓ over path, MM-RRT*



(g) Mean value of terminal ℓ



(h) Eigenvalue ℓ at all nodes in the tree

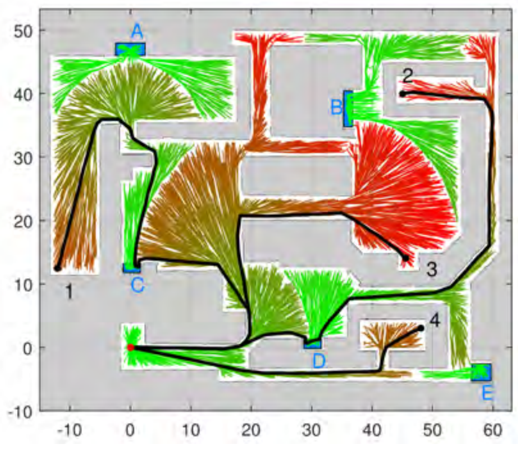
Figure 3.10: (a) - (f) show the evolution of the maximum eigenvalue bound encountered, path duration and the sum of the bound as a function of the number of nodes in the tree, over each of the 5 example paths from Figure 3.9, over 50 trials of our two competing approaches. (g) shows the mean terminal bound of each example path for the two approaches, and (h) shows a histogram of the bound's value at all nodes in the tree, averaged over 50 trials.

is introduced, the path, from B to C and finally to D, offers the shortest trajectory exposed to the regions that have no GPS access (shown in Figure 3.8(b)). In other words, we have $SI(B,C) + SI(C,D) < SI(B,D)$ as $SI(B,C)$ is equal to zero. All nodes in the GPS zone of Figure 3.8 share the same primary cost, as well as the same secondary cost from node B. Therefore, ties exist between these nodes. $TI()$ can be used to break these ties, resulting in the selection of node C as a parent node.

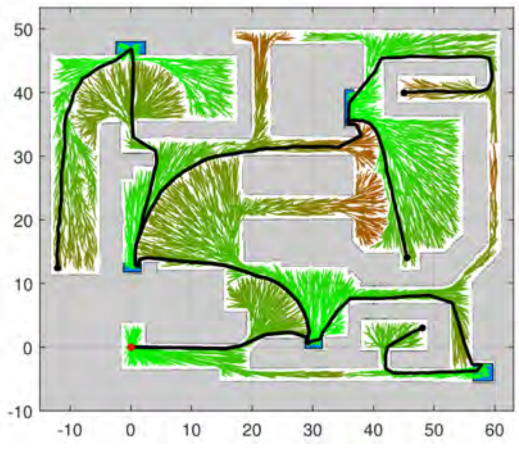
3.2.4 Experiments

We now describe a computational study performed to explore the effectiveness of planning under uncertainty using MM-RRT*. Two-dimensional robot workspaces are utilized to aid the visualization of the algorithm's performance, but it is also extensible to higher-dimensional systems. In our first example, we assume that a robot is capable of translation in two degrees of freedom, moves at constant speed, and is restricted to motion within the domain depicted in Figure 3.9. Throughout the domain the robot receives odometry measurements in both translational degrees of freedom. There are ten zones, illustrated in blue at the upper boundary of the domain, where the robot can receive GPS measurements. The robot's position estimate will drift and the error covariance terms associated with position will grow unless it obtains GPS measurements to curb error growth. All path planning problems explored in this domain are rooted at $x_{init} = (0.01, 0.99)$. Trees are constructed using two different strategies and the results are compared for paths to five different goal regions. The goal regions are indicated by the terminal locations of the paths depicted in Figure 3.9(a); they are numbered from one to five. Each goal is a circular region of radius 0.02 units.

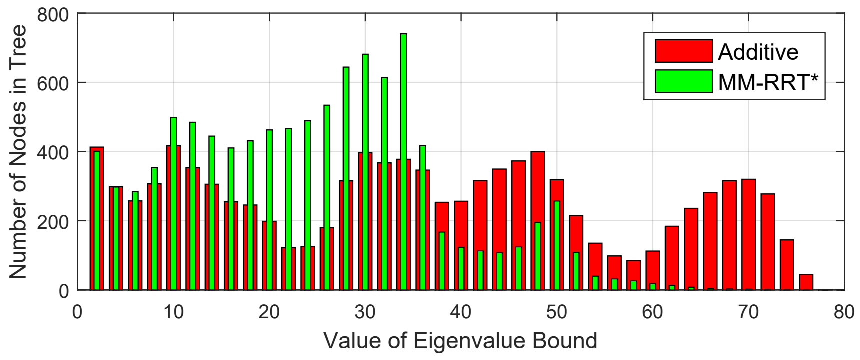
As a baseline for comparison, the first strategy explored is using an *additive* cost metric: the sum of the values of ℓ, summed at every filter update along a path. This metric, which penalizes the accumulation of uncertainty along a path, is intended to offer a basis for comparison for the MM-RRT* algorithm. A representative example



(a) Additive Approach



(b) MM-RRT*



(c) Eigenvalue ℓ at all nodes in the tree

Figure 3.11: A path planning example derived from a real-world building floorplan. Plots (a) and (b) show trees of 10,000 nodes each, generated by our two competing approaches using the same sequence of random samples. A histogram showing the value of the eigenvalue bound across all nodes in the tree, averaged over 150 trials, is shown in (c) for each approach.

of this approach is given in Figure 3.9(a). The second strategy explored is MM-RRT*, using $\bar{\ell}$ as the uncertainty cost metric, depicted in Figure 3.9(b). Fifty trials are performed of both the additive approach and MM-RRT* using ℓ as the basis for expressing uncertainty. In each trial, a tree of 50,000 nodes is constructed. Five quantities are compared between the two competing methods: (1) the maximum bound encountered over a path (Figure 3.10(a) and 4(b)), (2) path duration (Figure

Table 3.1: MM-RRT* vs. additive approach (aa) averaged over 150 trials (bound is given in units of distance, squared)

	Path	To 1	To 2	To 3	To 4
Max Bound Encountered	AA	52.837	69.947	67.907	50.386
	MM-RRT*	37.177	50.916	51.064	34.739
Sum of Bound	AA	2519	2926	2334	1288
	MM-RRT*	2513	3275	2515	1354
Terminal Bound	AA	52.837	69.947	67.907	50.386
	MM-RRT*	37.124	43.805	29.905	23.777
Runtime	AA	0.947s			
	MM-RRT*	1.042s			

3.10(c) and 4(d)), (3) the sum of the bound over each path (Figure 3.10(e) and 4(f)), (4) the mean value of each example path's terminal bound (Figure 3.10(g)), and (5) the value of the bound at all nodes in the tree (Figure 3.10(h)). All quantities shown in Figure 3.10 depict mean values averaged over 50 trials of each method. All computational experiments are performed using a single core of a laptop computer's Intel i7-4710MQ 2.5 GHz quad-core processor, equipped with 16GB RAM and the 64-bit Ubuntu 14.04 operating system. The RRT(*) C library provided by the authors of [17] is adapted to implement the algorithms considered in this section.

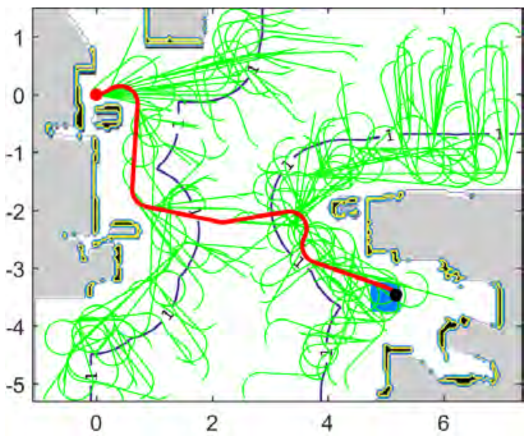
In Figure 3.10(a) and 4(b), the maximum value of ℓ encountered over each of the five highlighted paths is shown as a function of the number of nodes comprising the tree, averaged over 50 trials. The plot begins where a feasible path has been returned from all 50 trials. The mean maximum bound obtained over these five example paths is reduced by 21.21%, 36.26%, 35.98%, 39.51% and 35.6% respectively by MM-RRT* in comparison to the additive approach. Figure 3.10(b) demonstrates that MM-RRT*'s paths exhibit asymptotic behavior, however, even when averaged over 50 trials, it is clear that paths undergo small fluctuations in cost as they approach their respective asymptotes. Path duration is also shown as a function of the number of nodes in Figures 4(c) and 4(d). Due to its priority on curbing uncertainty, the lengths of the paths offered by MM-RRT* are greater than those offered by the additive approach. The mean terminal bound of each path is depicted in Figure 3.10(g). MM-RRT* reduces the value of each path's terminal bound by 23.9%, 54.91%, 38.21%, 41.96% and 34.54% respectively in comparison to the additive approach. Figure 3.10(h), using a histogram, visualizes the values of the eigenvalue bound $\bar{\ell}$ for all nodes in the final tree, averaged over 50 trials. The min-max approach offers a narrower spread of uncertainty among the nodes of its tree; 98.99% of the nodes of MM-RRT* have an eigenvalue bound smaller than 1.2 units of distance, squared. For the competing additive approach, only 70.31% of the tree's nodes fall within this threshold. The "fat tail" in the histogram for the additive approach, which contains 29.69% of the tree's nodes, demonstrates that penalizing the sum of uncertainties will often favor short paths that do not curb uncertainty to the same extent as a min-max approach.

Further comparisons are made in a more complex scenario inspired by the layout of a real-world office environment. A robot translates in a 78.1m by 63.3m domain from $x_{init} = (0.0, 0.0)$ shown in Figure 3.11. The same color spectrum used in Figure 3.9 is applied here. There are five regions that have GPS availability, labeled A through E. Four goal regions used in the planning problem are numbered from 1 to 4. One hundred fifty trials are performed in this scenario, and each trial grows a tree comprised of 10,000 nodes. The same quantities compared in Figure 3.10 are also compared in this case, and the computational results of the two competing planning methods, over 150 trials, are shown in Table 3.1. The paths of the additive approach only enter at most one GPS zone, while the paths produced by MM-RRT* enter at least two GPS zones each. 85.31% of the tree nodes of MM-RRT* have an eigenvalue bound smaller than $35m^2$. In the case of the additive approach, this number is only 51.33%.

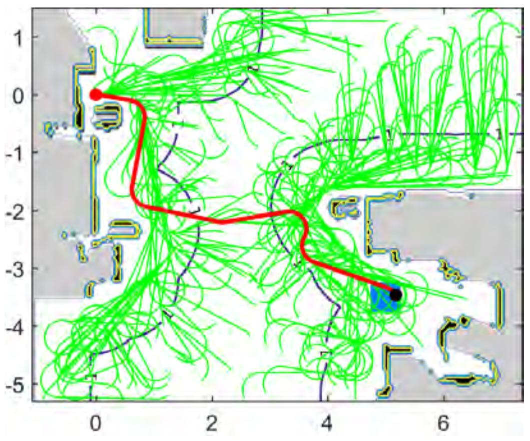
A final comparison is presented in Figure 3.12, showing a three degree-of-freedom Dubins vehicle planning over an experimentally derived map of our lab at Stevens Institute, where it is capable of localizing only within a 1m visibility range of the surrounding obstacles. Elsewhere, it must rely on odometry for navigation. In this case, trees are grown that obey Dubins constraints for a 0.3m turning radius, and a solution gradually evolves in real-time that limits the duration in which the vehicle relies exclusively on odometry.

3.2.5 Conclusions

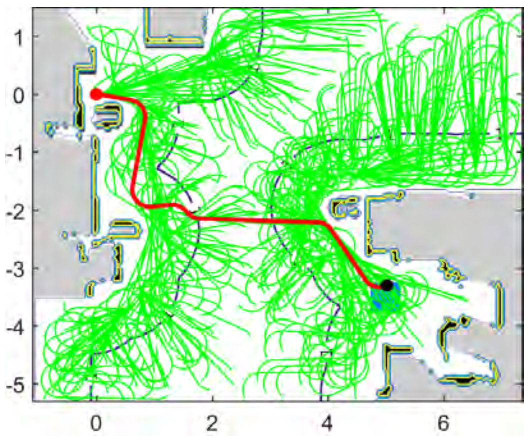
In this section, we have proposed a new path planning algorithm, MM-RRT*, that curbs localization uncertainty by minimizing the maximum value of its uncertainty metric encountered during the traversal of a path. This approach offers an alternative to other sampling-based belief space planning algorithms that employ additive cost representations of uncertainty. It also makes efficient use of a tree structure, with only a single belief allowed per graph node. As demonstrated in our computational results, an additive approach may often provide a suitable solution that curbs the growth of localization error, but the inherent preference for shorter paths will occasionally eliminate families of paths that are capable of further uncertainty reduction and safer operation. This initial exploration of a min-max uncertainty cost function also leaves room for future work: the approach may be combined with uncertainty-based rejection sampling to reduce collision risk, and the prospect of establishing formal optimality guarantees will be further explored.



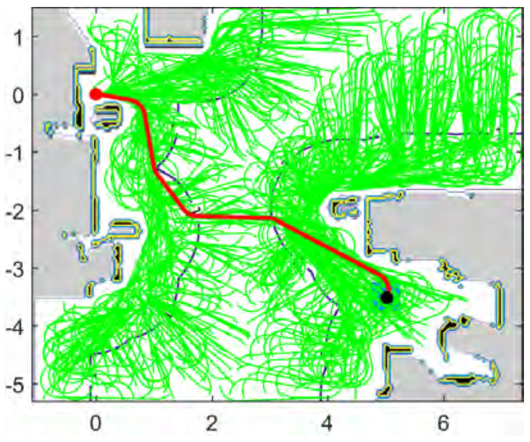
(a) 1 sec.



(b) 2 sec.



(c) 5 sec.



(d) 10 sec.

Figure 3.12: A real-time path planning example using an experimentally derived map from our lab, and a simulated ground robot with a three degree-of-freedom Dubins model. The robot can localize within a meter of the obstacles, but must rely on odometry elsewhere. The evolving solution and its tree is shown after 1, 2, 5, and 10 seconds of computation time.

3.3 Belief Roadmap Search

In this section, we give detailed characterization of what we term Belief Roadmap Search (BRMS), the search of a roadmap in belief space for minimum-uncertainty solutions that possess the optimal substructure property. However, we focus specifically on methods that achieve this without the use of belief-stabilizing controllers, which may not always be available in practice, especially under partial observability. Thus, we build on the work of BRM and RBRM to further refine the requirements for optimal substructure, discuss practical performance issues of BRMS, and we propose a new best-first implementation of BRMS, in contrast to the breadth-first implementations of BRM and RBRM, which yields significant improvement in the computational cost of belief-space search by eliminating unnecessary node expansions.

3.3.1 Problem Definition

Since we are still solving the problem of planning under uncertainty in this section, the problem definition, state estimation and uncertainty, and robot motion assumtions are the same as the definitions in Section 3.2.1. For the purpose of clarity, please refer to the previous section for the detailed descriptions.

3.3.1.1 Roadmap Search

We assume that a directed graph $G = (V, E)$ is provided as input, where V is the node set and E is the edge set. The robot's start belief (x_0, ℓ_0) and goal state x_{goal} define a search query. A node $n \in V$ is defined by $n = (x, \ell, dist, parent, children, \pi)$, which contains its state, eigenvalue bound, distance cost, parent, children and the path to n from x_0, which will be updated over the course of a search. These components can also be presented as $n._x$, $n._\ell$, $n._{dist}$, $n._{parent}$, $n._{children}$ and $n._\pi$ respectively. A node n_j

is accessible from node n_i if edge $e_{ij} \in E$ exists.

Now we define the cost function for the minimum goal-state uncertainty path planning problem. We first define $G_\pi(\ell_0)$, which is a composition of evaluations of Equation (3.9) over a path π. Every evaluation represents a measurement update of the EKF at each discrete-time instant:

$$G_\pi(\ell_0) := (\ell_T \circ \ell_{T-1} \circ \ldots \circ \ell_1)(\ell_0), \tag{3.14}$$

where ℓ_0 is the initial uncertainty at x_0. The optimal solution to a search query is a path π^* that has the lowest uncertainty at x_{goal} of all feasible paths that start from $(x_0,\, \ell_0)$:

$$\pi^* = \underset{\pi \in \mathcal{C}_{free}}{min}\, G_\pi(\ell_0), \tag{3.15}$$

where $\mathcal{C}_{free}$ denotes the states that are free of collision. To aid in expressing the cost of various paths, we let $c(n_i) = G_{\pi_{0,i}}(\ell_0)$ represent the error covariance eigenvalue upper bound $n_{i \cdot \ell}$ of node n_i for a path $\pi_{0,i}$ that connects $n_{0 \cdot x}$ and $n_{i \cdot x}$. Then the uncertainty at n_j for a path from $n_{0 \cdot x}$ to $n_{j \cdot x}$ while passing through $n_{i \cdot x}$ is defined as $c(n_i, n_j)$.

Our cost function presents the problem that equally suitable paths may exist when the robot is in an information-rich region where uncertainty is low everywhere. Similarly to [21], we define that node n_a dominates node n_b if:

$$n_a \lesssim n_b \Leftrightarrow (n_{a \cdot \ell} < (n_{b \cdot \ell} + \varepsilon)) \wedge (n_{a \cdot dist} < n_{b \cdot dist}), \tag{3.16}$$

where ε is a user-defined tolerance factor. In practice, we can set ε very small and prune redundant paths efficiently. Given these assumptions, we will use ℓ, the bound on the maximum eigenvalue of $\mathbf{P}$ given in Equation (3.9), to plan minimum goal-state uncertainty paths over roadmaps.

3.3.2 The Belief Roadmap Search

Here we give the details of the proposed search method, which we term the Belief Roadmap Search (BRMS); its presentation includes implementation details omitted from prior descriptions of minimum-uncertainty search methods (used with BRM and RBRM), and differences in implementation. We use a *queue* to keep track of nodes that have the potential to reduce a robot's localization uncertainty. Unlike the first-in, first-out approach employed by BRM and RBRM, we pop the node offering the lowest cost among all nodes in the queue. By popping nodes in a best-first manner, BRMS prioritizes the regions of the roadmap that expose a robot to high-quality measurements. We have observed this node selection strategy to have two key impacts: (1) it reduces the number of times that a node is placed back into the queue after being popped, and (2) it curbs the growth of *maximum* uncertainty that occurs over a path. The first aspect reduces search time, since unnecessary repetitive node expansion and cost updates are avoided. The second aspect will improve the quality of the path, since the robot spends less time in regions offering poor-quality measurements. Evidence of these observations will be provided in the following sections.

3.3.2.1 Algorithm Description

The proposed algorithm, BRMS, starts with a user-defined start belief (x_0, ℓ_0) and goal state x_{goal}. The input graph $G = (V, E)$ is a probabilistic roadmap, and the start node n_0 and goal node n_{goal} are connected to the nearest nodes in the roadmap if the connections are feasible (further sampling may be required if not).

Algorithm 3.5: The Belief Roadmap Search

```
   Input: Graph $G = (V, E)$, start node $n_0$ with $n_0{\cdot}x$ and $n_0{\cdot}\ell$, goal node $n_{goal}$
          with $n_{goal}{\cdot}x$
   // Append $G$ with nodes $\{n_0, n_{goal}\}$ and
   // put the start node $n_0$ into Queue
1  $Queue \leftarrow n_0$;
2  while $|Queue| > 0$ do
       // find the node that has minimum cost
3      $n_i \leftarrow FindMinBound(Queue)$;
       // delete node $n_i$ from Queue
4      $Queue \leftarrow Pop(Queue, n_i)$;
       // propagate if: #1. an edge $e_{ij}$ exists
       // between $n_i{\cdot}x$ and $n_j{\cdot}x$, #2. $n_j{\cdot}x$ is
       // not in the path to $n_i{\cdot}x$
5      for $\{n_j \mid e_{ij} \in E \; and \; n_j{\cdot}x \notin n_i{\cdot}\pi\}$ do
           // if $n_j{\cdot}\ell$ can be reduced by passing $n_i$,
           // then perform a swap
6          if $c(n_j) > c(n_i, n_j)$ then
7              $UpdateCost(n_i, n_j)$;
8              $RecursivePropagate(n_j)$;
9              $Queue \leftarrow Push(Queue, n_j)$;
10 return $n_{goal}{\cdot}\pi$
```

BRMS uses a queue to store all the nodes that have the potential to reduce uncertainty, as in the BRM and RBRM algorithms. Since an optimal solution cannot be guaranteed unless the graph is fully explored, the algorithm terminates only when the queue is empty. Instead of popping nodes in a first-in-first-out manner, BRMS pops the node that has the lowest cost in the queue (Algorithm 3.5, line 4). Then the popped node n_i expands the neighboring node n_j, if the edge e_{ij} exists in the graph, and n_j is not already in the path to n_i (line 5). This second condition disallows cycles in the path. The algorithm only proceeds if the cost at n_j can be improved in line 6. Lines 7 and 8 update the costs at node n_j and all of its children. When node n_j is pushed into the queue, it replaces any instances of n_j already in the queue. In

Algorithm 3.6: Cost Update Procedure

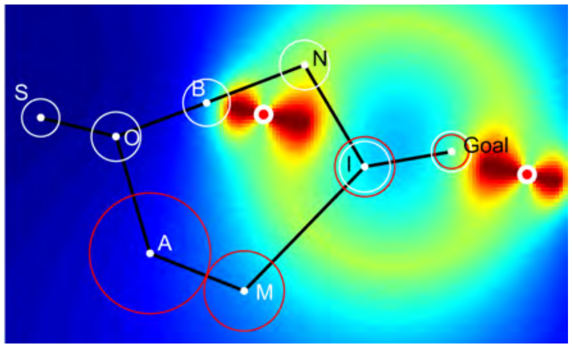
```
1  UpdateCost($n_i, n_j$)
       // propagate the new bound and distance cost
2      $n_{j \cdot \ell} \leftarrow c(n_i, n_j)$;
3      $n_{j \cdot dist} \leftarrow n_{i \cdot dist} + Distance(n_{i \cdot x}, n_{j \cdot x})$;
       // remove $n_j$ from its parent's children set
4      $Remove(n_{j \cdot parent}, n_j)$;
5      $n_{j \cdot parent} \leftarrow n_i$;
6      $n_{i \cdot children} \leftarrow n_{i \cdot children} \cup n_j$;
7      $n_{j \cdot \pi} \leftarrow n_{i \cdot \pi} \cup n_{j \cdot x}$;
8  RecursivePropagate($n_p$)
9      for $n_c \in n_{p \cdot children}$ do
10         $n_{c \cdot \ell} \leftarrow c(n_p, n_c)$;
11         $n_{c \cdot dist} \leftarrow n_{c \cdot dist} + Distance(n_{p \cdot x}, n_{c \cdot x})$;
12         $RecursivePropagate(n_c)$;
```

other words, there are no duplicated nodes in the queue at any time. When the queue is empty, the algorithm terminates and returns a path to x_{goal} that starts from x_0. Details of specific functions invoked in the pseudocode of Algorithms 3.5 and 3.6 are given below.

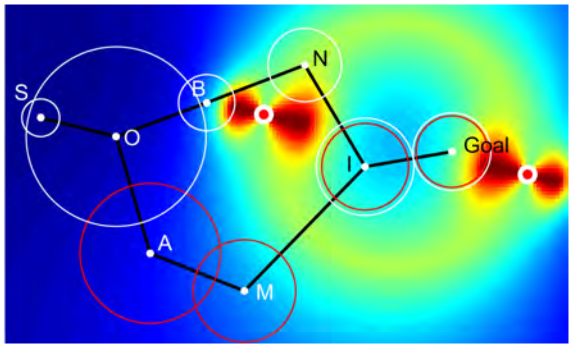
Find minimum bound: $FindMinBound()$ returns the node with the minimum bound among all nodes in *Queue*.

Pop a node: Unlike the *Pop*() function that deletes the first node of *Queue* in [46] and [47], the function *Pop*() here deletes the input node n_i from *Queue*.

Update node cost: The function *UpdateCost*(n_i, n_j) (Algorithm 3.6, line 1) is called if node n_j's uncertainty is reduced by passing through n_i. Accordingly, the costs at node n_j need to be updated. *Distance*() returns the length of edge e_{ij} (line 3). Node n_j will be removed from its existing parent's child list by calling function *Remove*() (line 4). Then n_j will be added to n_i's child list since n_i becomes n_j's new parent (lines 5 and 6). The path π to node n_j, $n_{j \cdot \pi}$, is also updated by passing through node n_i (line 7).



(a) Using the maximum eigenvalue of **P** as a cost function.



(b) Using an upper bound on the maximum eigenvalue of **P** instead.

Figure 3.13: A search example in which minimum goal-state uncertainty for a point robot is not achieved when we use the true maximum eigenvalue of the covariance matrix **P** as the cost function. Two red beacons provide noisy range and bearing measurements. The color of the workspace that varies from blue to red indicates the quality of positioning measurements, which vary from low (blue) to high (red). The radii at graph nodes represent the values of each metric.

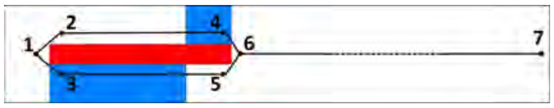
Update child costs: *RecursivePropagate*() updates the costs of all the input node's children. Since its children may have other children, this is a recursive function that will call itself and end when the child list of a node is fully explored.

3.3.3 Algorithm Analysis

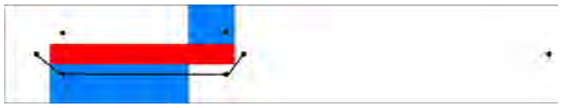
3.3.3.1 Optimal Substructure

We present BRMS as a search that minimizes the *goal-state uncertainty* of a path due to (1) the original application of BRM for this purpose [46], and (2) the proof of optimal substructure established when the eigenvalue bound ℓ is used to represent robot localization uncertainty [47]. Examples in this earlier work, in conjunction with our example from Figure 3.13, demonstrate the failure of other representations of localization uncertainty (such as the trace or true max. eigenvalue of **P**) to preserve optimal substructure.

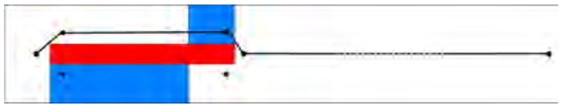
However, the allure remains of exploring other functions of ℓ, beyond its value



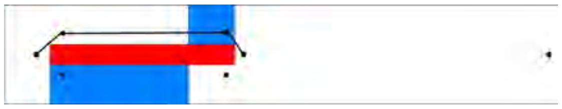
(a) The Roadmap Used for This Example



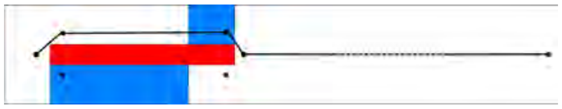
(b) Optimal Path to Node 6 Using Sum of ℓ or Min-Max ℓ Metric



(c) Optimal Path to Node 7 Using Sum of ℓ or Min-Max ℓ Metric



(d) Optimal Path to Node 6 Using Minimum Goal-State ℓ Metric



(e) Optimal Path to Node 7 Using Minimum Goal-State ℓ Metric

Figure 3.14: A planning example that fails to achieve optimal substructure when the sum of ℓ or max. value of ℓ across a path are used as cost metrics. A single-integrator point robot traverses a bounded region containing an obstacle (red). The robot's position can be measured in blue "GPS" regions, but it relies on odometry elsewhere. Robot localization uncertainty drops to the lowest level when the robot enters the blue regions. All paths shown originate from node 1.

at the robot's terminal state in the path, as potential cost metrics. Two of the most obvious candidates are (1) the sum of ℓ along the states in a path (from which other metrics, such as a robot's mean uncertainty, can be derived), and (2) the maximum value of ℓ along a path (we note that minimizing max. uncertainty was also propopsed in the original presentation of BRM [46]). A key property required of a cost metric in the proof of optimal substructure in [47] is monotonicity with respect to the *initial value* of the cost metric at the beginning of a path.

We provide counterexamples in Figure 3.14 which show that unfortunately, this type of monotonicity, and in turn, optimal substructure, are not possessed by cost functions that sum ℓ nor that evalute the max. of ℓ over all the filter updates along a path. In both cases, the optimal path from node 1 to node 6 is *1-3-5-6*. However, during the traversal of a very long edge from node 6 to node 7 without access to positioning measurements, the robot's uncertainty grows to such a large extent that the most recent high-value measurement (attainable at node 4) matters more than a previous history of high-quality measurements, and *1-2-4-6-7* provides the optimal path from node 1 to node 7. Although path *1-3-5-6-7* offers a lower sum of uncertainty and max. uncertainty *going in* to node 6, path *1-2-4-6-7* offers a lower sum of uncertainty and max. uncertainty *coming out* at node 7. Monotonicity with respect to the metrics' initial values at node 6 is thus not preserved along the path from 6 to 7, and the best paths to node 6 are not a subset of the best paths to node 7. However, this is not the case when using minimum goal-state uncertainty as a cost metric, for which monotonicity and optimal substructure are preserved.

3.3.3.2 Algorithm Complexity

The complexity of the search procedure used by the BRM and RBRM algorithms is given previously as $\mathcal{O}(b^d)$ [46], where d is the worst-case search depth of a roadmap, and b is the worst-case branching factor of the roadmap. In the worst case, the for-loop in line 5 of Algorithm 3.5 may require b iterations, and its outer while-loop (line 2) may continue further branching until nodes have been examined along the greatest depths that can be searched in the roadmap. This worst-case exponential complexity underscores the fact that nodes may be placed in and removed from the queue multiple times, and edges potentially re-examined, until no further reductions in uncertainty are possible. The step-by-step example given in Figure 3.15 illustrates both of these events in the course of performing an uncertainty-optimal search.

However, the previously stated search complexity of $\mathcal{O}(b^d)$ does not account for the use of the *RecursivePropagate*() function (line 8 of Algorithm 3.5), which is needed to propagate uncertainty to the *descendants* of a node when the search reduces uncertainty at a *parent* node. Although this function is not explicity stated in

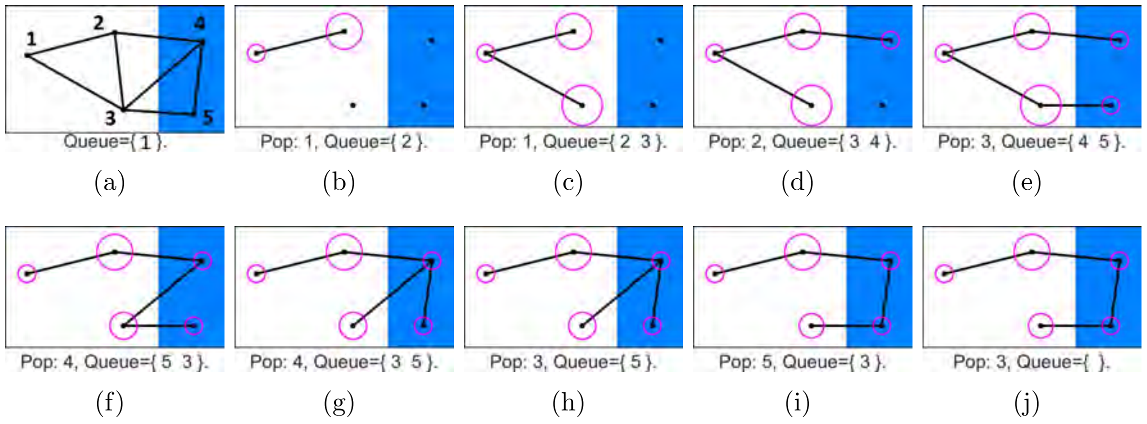


Figure 3.15: An example showing every step of a breadth-first search over a simple roadmap, producing a single-source mininium goal-state uncertainty solution with node 1 as the source node.

earlier presentations of the BRM and RBRM algorithms, this subroutine is required. Without it, sub-optimal solutions may result - its importance is illustrated in the example of Figure 3.15. In panel (f) of this example, node 4 is examined, and node 3 is identified as a candidate whose uncertainty will be reduced by re-routing through node 4. This also has implications for node 5, which is currently the child of node 3 along the best-so-far path from the source node 1. The new uncertainty at node 3 must be propagated to node 5, otherwise node 5 will not be evaluated accurately in future iterations of the search.

The worst-case complexity of the *RecursivePropagate*() function is $\mathcal{O}(n)$, where n is the number of nodes in the roadmap. A node may have all other nodes in the graph as descendants in a working mininum-uncertainty paths solution, and we assume, as in [46], that the cost of uncertainty propagation across edges using Equation 3.9 can be amortized. This $\mathcal{O}(n)$ operation may be required in every iteration of the for-loop of Algorithm 3.5, which brings the complexity of BRMS to $\mathcal{O}((bn)^d)$. Finally, there is a cost associated with our recommended best-first prioritization of the search queue. This is a worst-case $\mathcal{O}(n)$ operation that occurs in each iteration

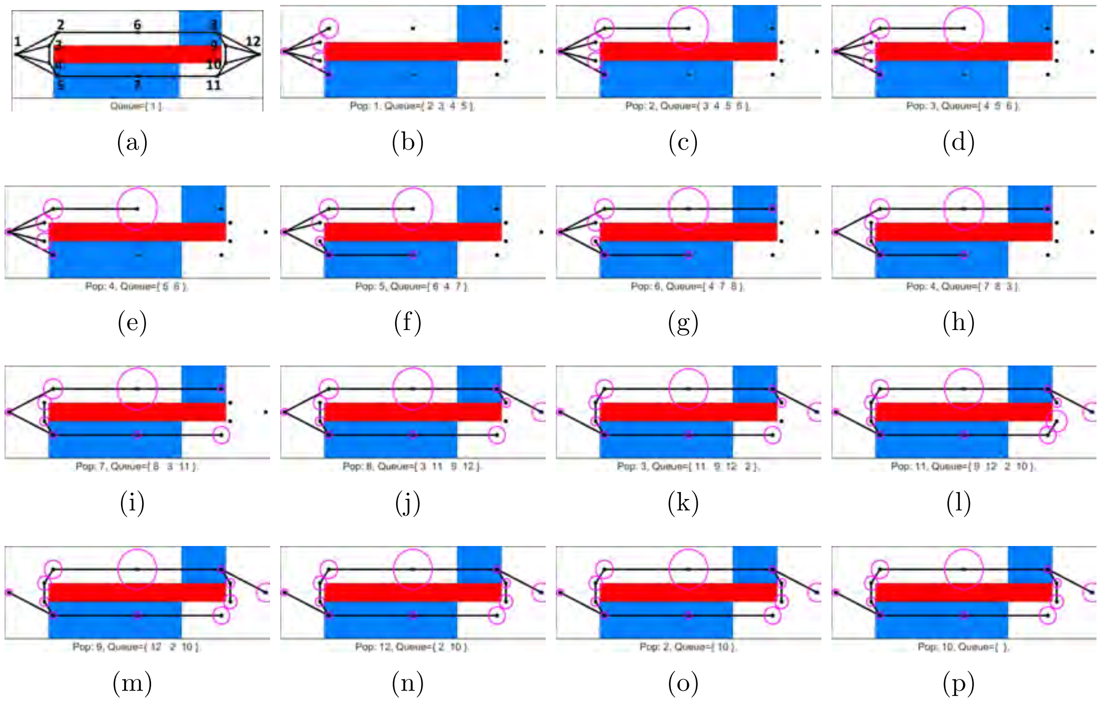


Figure 3.16: Search process of breadth-first search (cost metric ℓ is enlarged for visualization).

of the while-loop of Algorithm 3.5. This brings the cost of BRMS to $\mathcal{O}((n + bn)^d)$, but since the $\mathcal{O}(bn)$ contribution of the for-loop is greater, we may continue to state the algorithm's worst-case complexity as $\mathcal{O}((bn)^d)$. Hence, there is no difference in the worst-case cost of a best-first vs. breadth-first prioritization of the queue.

3.3.3.3 Anti-Cycling Rule

As a practical implementation measure (enforced in line 5 of Algorithm 3.5), preventing the occurrence of cycles in a minimum-uncertainty paths solution improves the BRMS algorithm's computational tractability, while also ensuring that the solutions produced are time-efficient. However, this anti-cycling rule, which is also enforced in [46] and [47], also prevents some of the paths in a single-source solution from representing true minimum goal-state uncertainty paths. This may be illustrated once

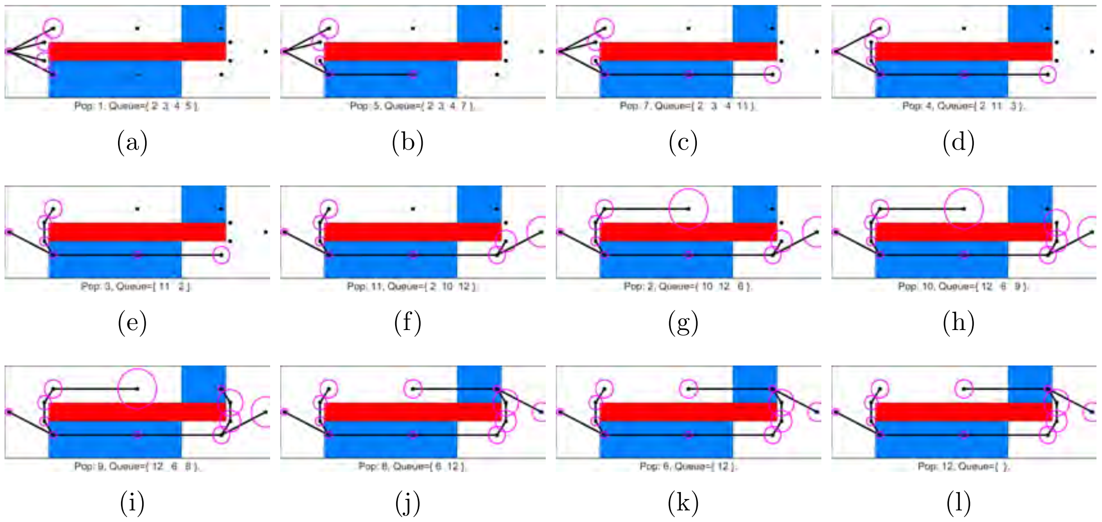

Pop: 1, Queue={ 2 3 4 5 }.
(a)
Pop: 5, Queue={ 2 3 4 7 }.
(b)
Pop: 7, Queue={ 2 3 4 11 }.
(c)
Pop: 4, Queue={ 2 11 3 }.
(d)
Pop: 3, Queue={ 11 2 }.
(e)
Pop: 11, Queue={ 2 10 12 }.
(f)
Pop: 2, Queue={ 10 12 6 }.
(g)
Pop: 10, Queue={ 12 6 9 }.
(h)
Pop: 9, Queue={ 12 6 8 }.
(i)
Pop: 8, Queue={ 6 12 }.
(j)
Pop: 6, Queue={ 12 }.
(k)
Pop: 12, Queue={ }.
(l)

Figure 3.17: Search process of the proposed best-first BRMS method (cost metric ℓ is enlarged for visualization).

again using the example in Figure 3.15.

It is clear that the resulting path from node 1 (the source node) to node 2 in Figure 3.15 is not the solution offering minimum goal-state uncertainty. Instead, path *1-3-5-4-2* would offer minimum goal-state uncertainty at node 2. However, en route to producing minimum goal-state uncertainty solutions at all other nodes in the roadmap, the current single-source solution originating at node 1, *1-2-4-5-3*, produces a sub-optimal solution at node 2. Due to the anti-cycling rule, in a solution with node 1 as the source node, either node 2 or node 3 must have node 1 as a parent, and the order in which they are popped from the queue will determine which node is forced to accept sub-optimal localization uncertainty.

3.3.3.4 Practical Implementation Notes

Despite the unimportance of queue prioritization to the worst-case computational complexity of BRMS, we have observed this aspect of the algorithm to have an important practical impact on its efficiency and the quality of planning outcomes. Figure

3.16 gives a planning example with a breadth-first, first-in first-out handling of the queue as in the original BRM and RBRM algorithms. In contrast, Figure 3.17 shows the same example with our proposed best-first handling of the queue. We employ the same assumptions as in Figure 3.14. A graph that is composed of 12 nodes is shown in Figure 3.16(a), and the uncertainty at each node, represented by ℓ, is colored magenta and scaled up for visualization. Starting from node 1, every step of the search is shown, for both queue prioritization schemes. As evident in panels 3.16 (f), (i) and (l), the breadth-first search propagates through the graph in parallel on both sides of the red obstacle. In contrast, the best-first search wraps around the lower half of the obstacle, proceeding first through an area of high-quality robot positioning measurements, before exploring other areas. Ultimately, the best-first BRMS terminates in fewer iterations than its breadth-first counterpart, as fewer nodes are re-inserted into the queue after their initial examination. Due to the anti-cycling rule, identical uncertainty is not achieved at all nodes. Node 6 in Figure 3.16 has substantially higher uncertainty than its counterpart in Figure 3.17, but nodes 9 and 10 in Figure 3.16 achieve lower uncertainty than their counterparts in Figure 3.17. These simple examples illustrate the potential impact of the anti-cycling rule, although these consequences lessen as denser roadmaps with a greater diversity of alternative routes are available. In the following section, we will illustrate the same practical observations about search efficiency over a larger class of examples.

3.3.3.5 Cost-to-go Heuristic

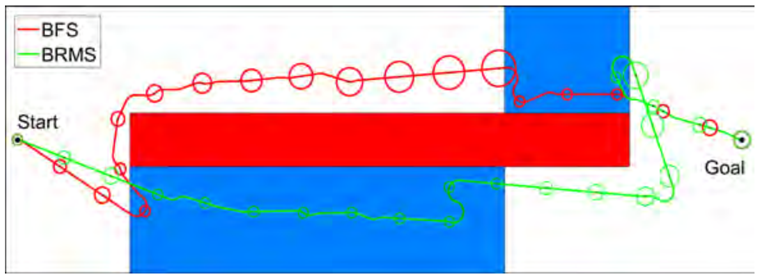
The $A*$ cost-to-go heuristic can be applied to some planning problems to find a solution faster. However, an admissible cost-to-go heuristic doesn't typically apply to the problem of planning under uncertainty - the evolution of ℓ over the duration of a path, for example, is non-monotonic. A probabilistically conservative heuristic is proposed in [31] by combining distance cost and uncertainty cost using a weighted sum method. Extra care is needed when selecting the optimal weights for these two costs that have different units. As a result, this method only finds optimal solutions probabilistically. A similar approach that combines distance cost with uncertainty cost is used in [29]. An $A*$ search algorithm for planning under uncertainty is proposed in [34], however, this approach imposes collision probability constraints, rather than evaluating uncertainty in the cost function. Nonetheless, developing an effective cost-to-go heuristic for use in BRMS is an interest for future work.

3.3.4 Experiments

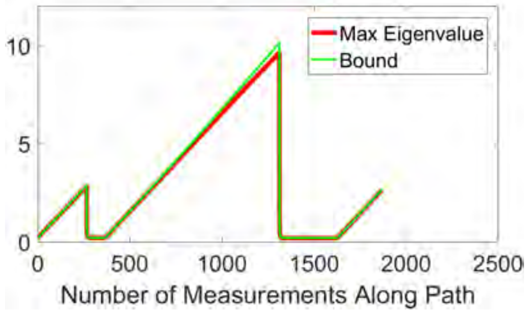
We now describe a computational study performed to explore the effectiveness of planning under uncertainty using our proposed method. The examples explored include a constant-velocity Dubins vehicle, a holonomic UAV with 10 degrees of freedom, and a Clearpath Jackal ground robot.

3.3.4.1 Dubins Vehicle

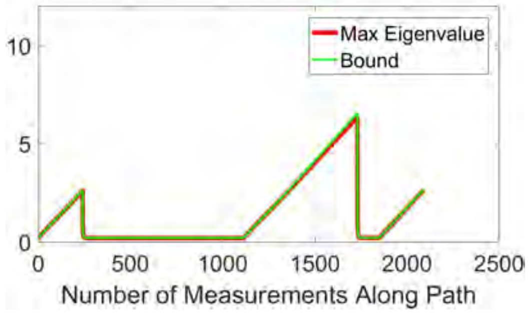
We use the scenario presented in Figure 3.16 for our first Dubins vehicle example. Again, an obstacle is colored red, a robot receives position measurements in blue regions, and odometry measurements are available throughout the domain. The uncertainty of robot position estimation drops to the lowest level after the robot enters blue regions, while the robot's position estimate will drift and the error covariance terms associated with position will grow elsewhere. Planned paths are plotted in Figure 3.18 (a) for both the proposed BRMS method, and a standard breadth-first method, hereafter abbreviated BFS. Though both methods return solutions offering minimum uncertainty at the goal state, our proposed BRMS method achieves a higher-quality path that curbs the growth of uncertainty throughout the entire route, rather than



(a)



(b) Upper (BFS) Path

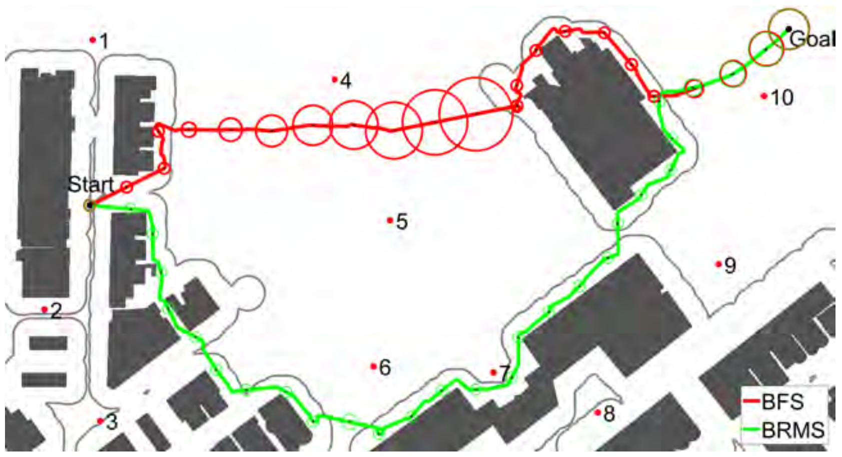


(c) Lower (BRMS) Path

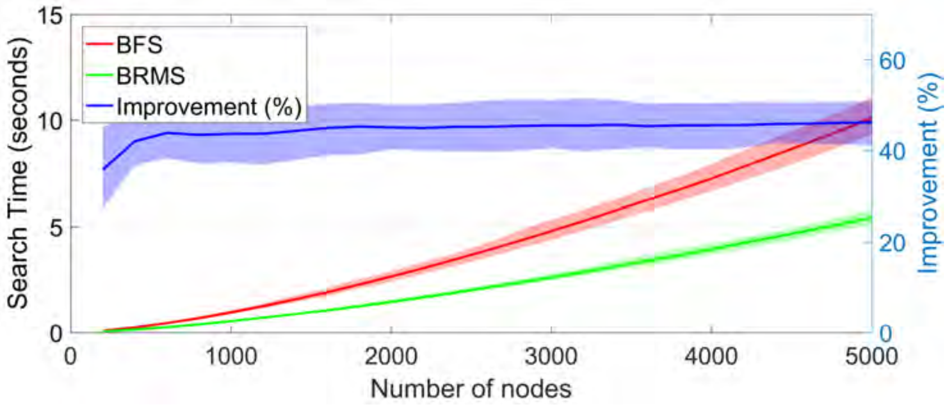
Figure 3.18: Minimum uncertainty paths returned from BFS and BRMS methods are shown in (a). The values of the max. eigenvalue of $\mathbf{P}$ and the bound ℓ are plotted as a function of number of measurements over the path for both paths in (b) and (c). The roadmap used to produce these paths contains 5000 nodes.

just at the goal state. Produced from a 5000-node probabilistic roadmap, the two competing solutions now share fewer nodes in common than the earlier examples of Section 3.3.3.

We next plan over a city map, which has a dimension of 1480 meters by 800 meters, for the same Dubins vehicle, shown in Figure 3.19. We assume the robot is now equipped with a range sensor, which has a 30 meter range, for localization. The continuous gray boundaries around the buildings indicate the regions where the robot can perform localization using the rangefinder. The value of ℓ is scaled up 10 times for visualization of the error ellipses plotted. Outside of its visibility range to a structure, the robot must rely on noisy odometry. As is shown in Figure 3.19(a), both BRMS and BFS return solutions that have identical goal-state uncertainty. However,



(a)

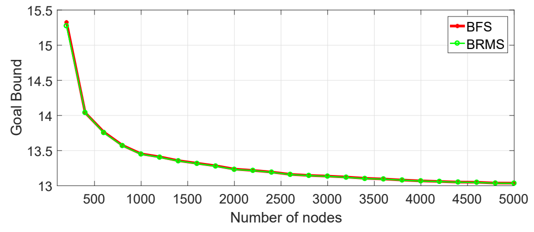


(b)

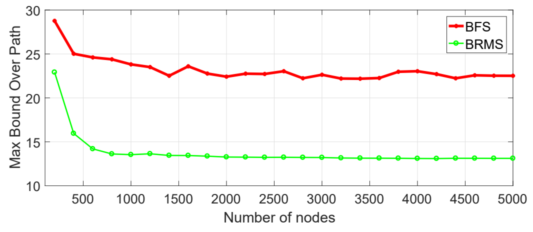
Figure 3.19: Dubins paths planned by BFS and BRMS are shown in (a). The search time and relative percentage improvement are plotted in (b) as a function of the number of nodes in the graph, averaged over 50 trials. The mean value is plotted as a solid line, and the shaded regions indicate the 10th-to-90th percentiles.

a large part of the path of the BFS method is exposed to regions where the robot can only rely on odometry. The robot will encounter a higher average and maximum uncertainty over the duration of the BFS path. In real-world applications, such a path may result in localization failure. On the other hand, due to the priority on exploring nodes offering high-quality sensing, the path of our proposed BRMS method stays in the visibility regions as long as possible before reaching the goal.

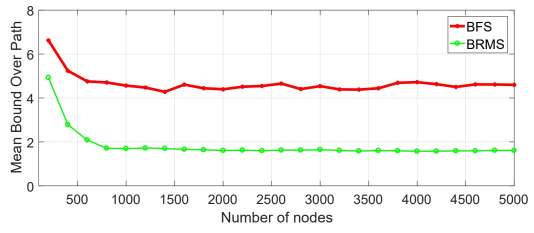
All computational experiments were performed using a single core of a laptop's



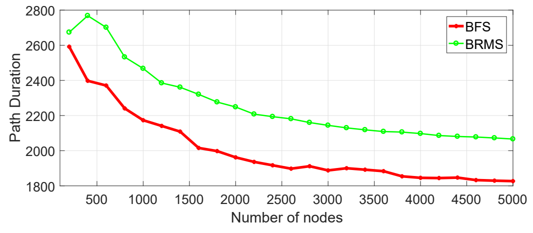
(a) Bound at Goal Location



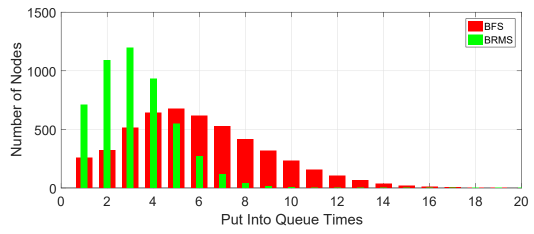
(b) Max Bound Over Path



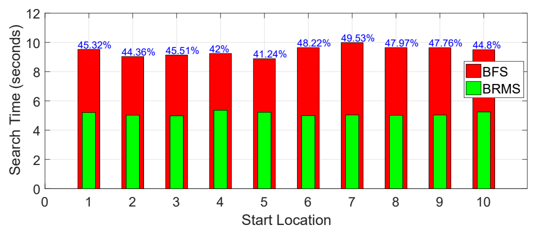
(c) Mean Bound Over Path



(d) Path Length



(e) Times a Node is Inserted into Queue

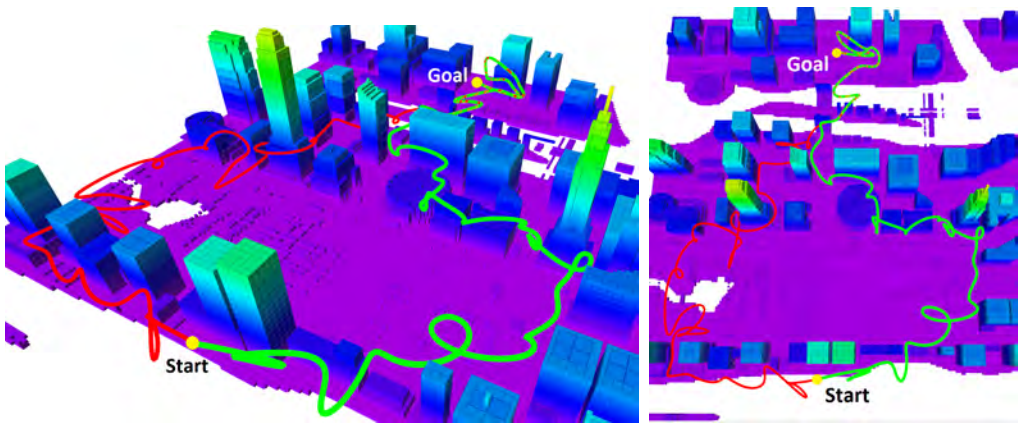


(f) Search Time for 10 Different Start Locations

Figure 3.20: Subfigures (a)-(d) show the evolution of the goal-state ℓ, the maximum ℓ encountered, the mean ℓ and the path length (in meters) as a function of the number of nodes in the graph, for the specific city map Dubins vehicle planning query illustrated in Figure 3.19. The histogram in (e) shows the number of times that nodes are inserted into the queue for both methods. Search time for 10 other randomly chosen start locations is shown in (f). All comparisons are averaged over 50 trials.

Intel i7 2.5 GHz quad-core processor, equipped with 16GB RAM and the 64-bit Ubuntu 14.04 operating system. A comparison of search time for the city map example is presented in Figure 3.19 (b). BRMS improves the computation time by about 47% when we have more than 5,000 nodes in the graph. Figure 3.20 (a)-(d) shows the evolution of the goal-state uncertainty, the max. uncertainty encountered, the mean uncertainty and the path length as a function of the number of nodes in the graph.

Though the BFS method achieves the same optimal value of ℓ at the goal location, our BRMS method outperforms it in computational efficiency and in max. and mean uncertainty achieved along a path. We note however that the length of the path offered by our method is greater than the one offered by the BFS approach. Figure 3.20(e) confirms that BRMS reduces the number of times that a node is inserted into the queue. In order to verify our method's effectiveness in other start locations, we randomly select 10 different locations in the map and plot their corresponding search times in Figure 3.20(f). Search time improvements are listed on top of each bar.



(a) (b)

Figure 3.21: UAV planning example, in which paths returned from BFS and BRMS are colored red and green respectively. The roadmap used to produce these paths contains 5000 nodes.

3.3.4.2 UAV Simulation

We next explore a kinodynamic UAV motion planning problem, in which a quadrotor model [124], which is 10-dimensional, is linearized about the aircraft's hover point. Its state can be expressed as $x = (p, v, r, w)^T$, where p and v are three-dimensional position and velocity, r and w are two-dimensional orientation and angular velocity, and yaw and its derivative are constrained to zero after linearization. We also assume the UAV is equipped with a planar rangefinder for localization purposes. The paths obtained from BFS and BRMS are plotted in Figure 3.21. We note that the paths from both methods share the same final portion. However, the BFS path (red) traverses a large open area, which results in greater max. uncertainty at the goal state. The BRMS path (green) is longer, but stays close to the buildings at all times to localize. A similar search time improvement to that of the Dubins example is also achieved here.

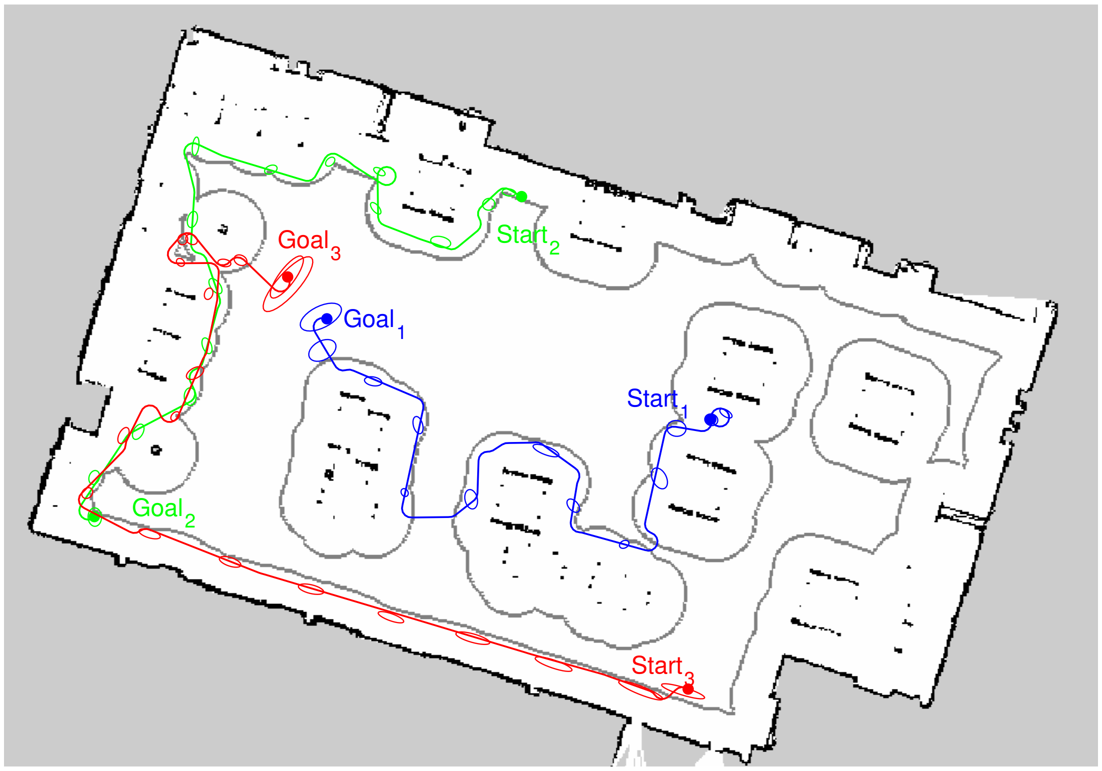


Figure 3.22: BRMS-derived trajectories executed using a Clearpath Jackal ground robot in an indoor office environment. The continuous gray line around obstacles indicates a visibility boundary, within which obstacles can be observed for localization. Three paths with different start and goal locations, which are derived from our proposed method, are colored blue, green and red. The roadmap used to produce these paths contains 3000 nodes.

3.3.4.3 Experimental Results with Robot Hardware

Finally, we implement the proposed algorithm on a mobile robot, the Clearpath Jackal, with motion planning using PRM subject to Dubins constraints. Our aim is to examine the effectiveness of the proposed method for a real-world mobile robot. A Hokuyo UTM-30LX laser scanner, which has a 30 meter range and 270° field of view, is mounted on the top of the robot. In order to visualize the benefits of our method during the mission, Adaptive Monte Carlo Localization (AMCL) [125], which uses a particle filter to track the pose of a robot, is employed. We discard laser range returns that are more than 1 m away from robot, allowing a small-scale indoor environment to produce varied localization outcomes. When the robot is more than 1 m away from features in the environment, AMCL will be forced to rely on noisy wheel odometry only. This "dead-reckoning region" is outlined by a gray line. The uncertainty of the robot, derived from the output of AMCL, is represented using 95% confidence ellipses along the plotted trajectories.

Representative robot execution traces of trajectories from different start and goal locations are illustrated in Figure 3.22. Note that the uncertainty grows dramatically when only odometry information, which is noisy and inaccurate, is available for AMCL. As mentioned in the previous section, the paths derived from our method stay in good measurement regions as long as possible before reaching the goal locations. We also note that sometimes, in the course of performing trials, AMCL fails to localize the robot after traversing without an obstacle detection for some amount of time. Though BRMS paths tend to be longer, they also tend to avoid localization-poor regions due to the prioritization of nodes offering high-quality measurements.

3.3.5 Conclusions

In this section, we have explored theoretical and practical implementation issues of optimal Belief Roadmap Search, proposing our own variant of the technique that plans minimum goal-state uncertainty paths using an upper bound on the EKF covariance max. eigenvalue (where we have shown mean and min-max functions of this metric to fail), implements best-first prioritization of its search queue, and, for the first time, formally incorporates the recursive propagation of uncertainty across the descendants of a node, an essential step for correct and consistent algorithm performance. We have established the exponential worst-case complexity of the search, while at the same time demonstrating the practical benefits achieved from a best-first formulation of the search - a speed-up due to reduced re-insertion of nodes into the queue, and high-quality paths that tend to achieve lower mean and max. uncertainty over the duration of a path.

Chapter 4
Lightweight Lidar Odometry

4.1 Introduction

A fundamental question encountered in autonomous navigation is “where am I”. Localization is a key prerequisite for executing most missions. The relative or absolute position of a robot needs to be obtained from measurements that can be internal or external. Given the measurements, the robot needs to know its position as accurately as possible. We pursue reliable, real-time six degree-of-freedom pose estimation for ground vehicles equipped with 3D lidar, in a manner that is amenable to efficient implementation on a small-scale embedded system. Such a task is non-trivial for several reasons. Many unmanned ground vehicles (UGVs) do not have suspensions or powerful computational units due to their limited size. Non-smooth motion is frequently encountered by small UGVs driving on variable terrain, and as a result, the acquired data is often distorted. Reliable feature correspondences are also hard to find between two consecutive scans due to large motions with limited overlap. Besides that, the large quantities of points received from a 3D lidar poses a challenge to real-time processing using limited on-board computational resources.

When we implement LOAM [78, 79] for our tasks, we can obtain low-drift motion estimation when a UGV is operated with smooth motion amidst stable features, and supported by sufficient computational resources. However, the performance of LOAM deteriorates when resources are limited. Due to the need to compute the roughness of every point in a dense 3D point cloud, the update frequency of feature extraction on a lightweight embedded system cannot always keep up with the sensor update frequency. Operation of UGVs in noisy environments also poses challenges for LOAM. Since the mounting position of a lidar is often close to the ground on a small UGV, sensor noise from the ground may be a constant presence. For example, range returns from grass may result in high roughness values. As a consequence, unreliable edge features may be extracted from these points. Similarly, edge or planar features may also be extracted from points returned from tree leaves. Such features are usually not reliable for scan-matching, as the same grass blade or leaf may not be seen in two consecutive scans. Using these features may lead to inaccurate registration and large drift.

In this chapter, we describe a lightweight and ground-optimized LOAM (LeGO-LOAM) for pose estimation of UGVs in complex environments with variable terrain. LeGO-LOAM is lightweight, as real-time pose estimation and mapping can be achieved on an embedded system. Point cloud segmentation is performed to discard points that may represent unreliable features after ground separation. LeGO-LOAM is also ground-optimized, as we introduce a two-step optimization for pose estimation. Planar features extracted from the ground are used to obtain $[t_z, \theta_{roll}, \theta_{pitch}]$ during the first step. In the second step, the rest of the transformation $[t_x, t_y, \theta_{yaw}]$ is obtained by matching edge features extracted from the segmented point cloud. We also integrate the ability to perform loop closures to correct motion estimation drift.

4.2 LeGO-LOAM

An overview of the proposed framework is shown in Figure 4.1. The system receives input from a 3D lidar and outputs 6 DOF pose estimation. The overall system is divided into five modules. The first, *segmentation*, takes a single scan's point cloud and projects it onto a range image for segmentation. The segmented point cloud

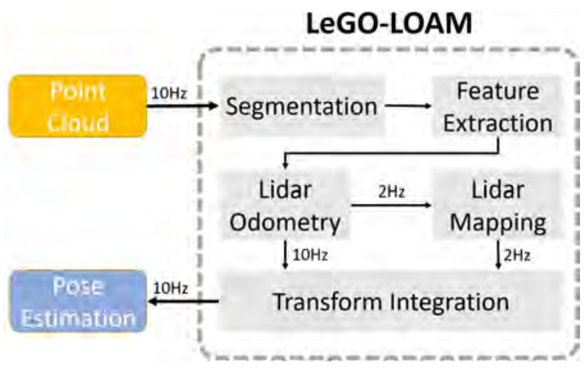
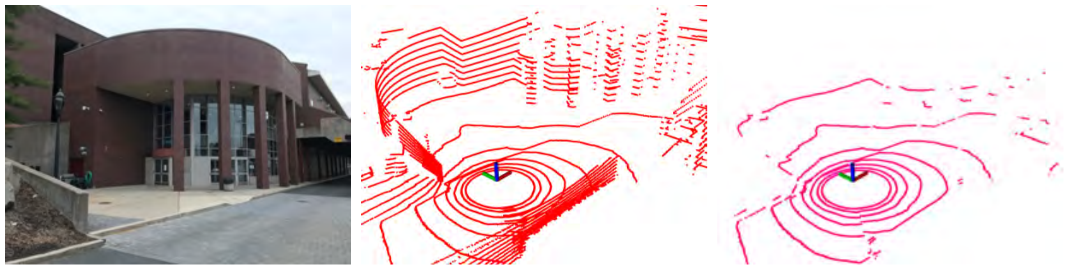


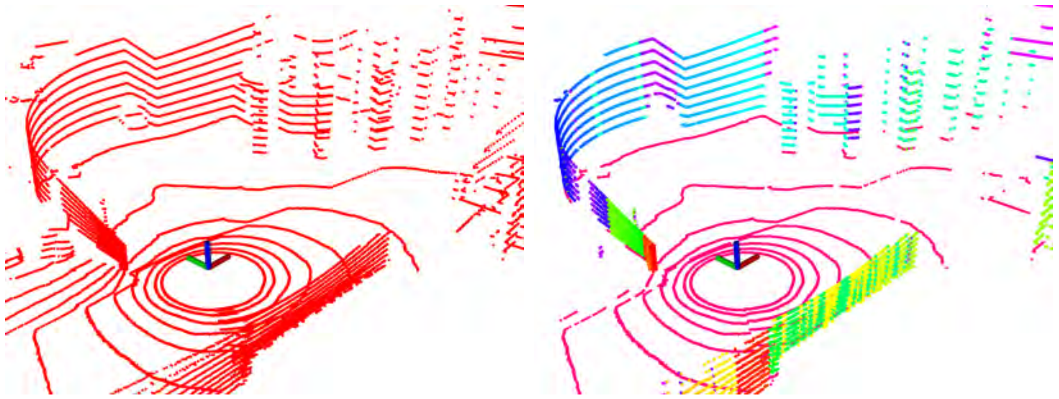
Figure 4.1: System overview of LeGO-LOAM.



(a) Stevens gym (b) Raw point cloud (c) Ground points

Figure 4.2: Ground separation for a scan in urban environment. The original point cloud from VLP-16 is shown in (b). The separated ground points are shown in (c).

is then sent to the *feature extraction* module. Then, *lidar odometry* uses features extracted from the previous module to find the transformation relating consecutive scans. The features are further processed in *lidar mapping*, which registers them to a global point cloud map. At last, the *transform integration* module fuses the pose estimation results from *lidar odometry* and *lidar mapping* and outputs the final pose estimate. The proposed system seeks improved efficiency and accuracy for ground vehicles, with respect to the original, generalized LOAM framework of [78] and [79]. The details of these modules are introduced below.



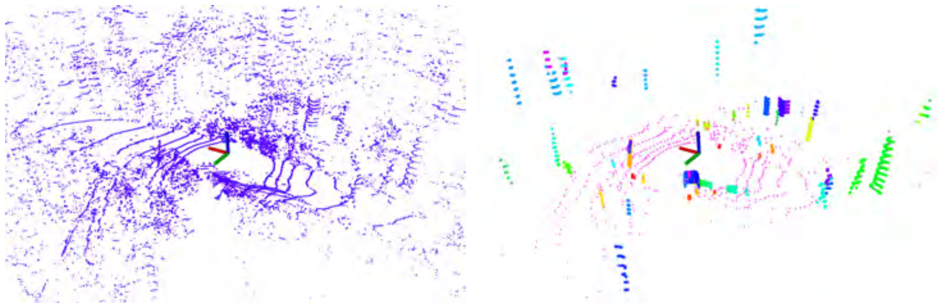
(a) Raw point cloud (b) Segmented point cloud

Figure 4.3: Point cloud segmentation. (b) shows the original point cloud from VLP-16. The segmented point cloud is shown in (c). Each color represents a segment.

4.2.1 Segmentation

Let $P_t = \{p_1, p_2, ..., p_n\}$ be the point cloud acquired at time t, where p_i is a point in P_t. P_t is first projected onto a range image. The resolution of the projected range image is 1800 by 16, since the VLP-16 has horizontal and vertical angular resolution of $0.2°$ and $2°$ respectively. Each valid point p_i in P_t is now represented by a unique pixel in the range image. The range value r_i that is associated with p_i represents the Euclidean distance from the corresponding point p_i to the sensor. Since sloped terrain is common in many environments, we do not assume the ground is flat. A column-wise evaluation of the range image, which can be viewed as ground plane estimation [126], is conducted for ground point extraction before segmentation. After this process, points that may represent the ground are labeled as ground points and not used for segmentation.

Then, an image-based segmentation method [127] is applied to the range image to group points into many clusters. Points from the same cluster are assigned a unique label. Note that the ground points are a special type of cluster. Applying segmentation to the point cloud can improve processing efficiency and feature extrac-



(a) Raw point cloud (b) Segmented point cloud

Figure 4.4: Point cloud segmentation results in a noisy environment. The original point cloud is shown in (a). In (b), the red points are labeled as ground points. The rest of the points are the points that remain after segmentation.

tion accuracy. Assuming a robot operates in a noisy environment, small objects, e.g., tree leaves, may form trivial and unreliable features, as the same leaf is unlikely to be seen in two consecutive scans. In order to perform fast and reliable feature extraction using the segmented point cloud, we omit the clusters that have fewer than 30 points. A visualization of a point cloud before and after segmentation is shown in Figure 4.3.

After this process, only the points that may represent large objects, e.g., tree trunks, and ground points are preserved for further processing. At the same time, only these points are saved in the range image. The benefits of performing point cloud segmentation for a scan that is captured in a noisy environment is shown in Figure 4.4. The original point cloud includes many points, which are obtained from surrounding vegetation that may yield unreliable features. At last, we also obtain three properties for each point: (1) its label as a ground point or segmented point, (2) its column and row index in the range image, and (3) its range value. These properties will be utilized in the following modules.

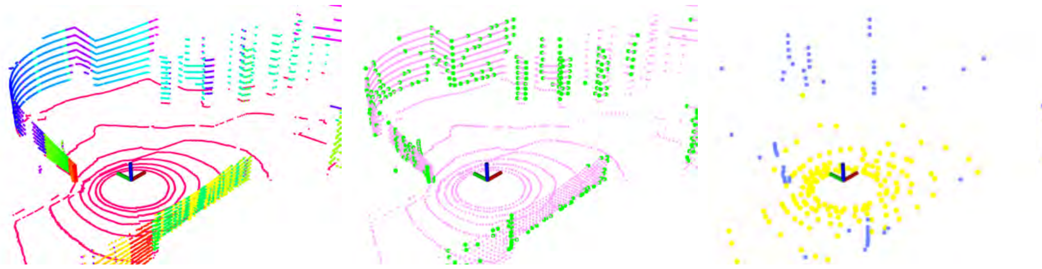

(a) Segmented point cloud (b) Feature sets $\mathbb{F}_e$ and $\mathbb{F}_p$ (c) Features F_e and F_p

Figure 4.5: Feature extraction on segmented point cloud. In (b), the green and pink points represent edge and planar features in $\mathbb{F}_e$ and $\mathbb{F}_p$ respectively. In (c), blue and yellow points indicate edge and planar features in F_e and F_p.

4.2.2 Feature Extraction

The feature extraction process is similar to the method used in [79]. However, instead of extracting features from raw point clouds, we extract features from ground points and segmented points. Let S be the set of continuous points of p_i from the same row of the range image. Half of the points in S are on either side of p_i. We set $|S|$ to 10 for all tests. Using the range values computed during segmentation, we can evaluate the roughness of point p_i in S,

$$c = \frac{1}{|S| \cdot \|r_i\|} \Big\| \sum_{j \in S, j \neq i} (r_j - r_i) \Big\|. \tag{4.1}$$

To evenly extract features from all directions, we divide the range image horizontally into several equal sub-images. Then we sort the points in each row of the sub-image based on their roughness values c. Similar to LOAM, we use a threshold c_{th} to distinguish different types of features. We call the points with c larger than c_{th} *edge* features, and the points with c smaller than c_{th} *planar* features. Then $n_{\mathbb{F}_e}$ edge feature points with the maximum c, which do not belong to the ground, are selected from each row in the sub-image. $n_{\mathbb{F}_p}$ planar feature points with the minimum c, which

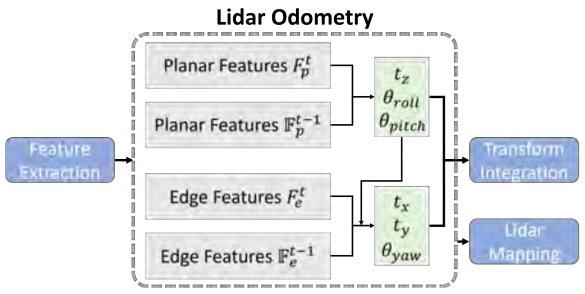

Lidar Odometry
Planar Features F_p^t
Planar Features $\mathbb{F}_p^{t-1}$
Edge Features F_e^t
Edge Features $\mathbb{F}_e^{t-1}$
t_z θ_{roll} θ_{pitch}
t_x t_y θ_{yaw}
Feature Extraction
Transform Integration
Lidar Mapping

Figure 4.6: Two-step optimization for the *lidar odometry* module. $[t_z, \theta_{roll}, \theta_{pitch}]$ is first obtained by matching the planar features extracted from ground points. $[t_x, t_y, \theta_{yaw}]$ are then estimated using the edge features extracted from segmented points while applying $[t_z, \theta_{roll}, \theta_{pitch}]$ as constraints.

may be labeled as either ground or segmented points, are selected in the same way. Let $\mathbb{F}_e$ and $\mathbb{F}_p$ be the set of all edge and planar features from all sub-images. These features are visualized in Figure 4.5(b). We then extract n_{F_e} edge features with the maximum c, which do not belong to the ground, from each row in the sub-image. Similarly, we extract n_{F_p} planar features with the minimum c, which must be ground points, from each row in the sub-image. Let F_e and F_p be the set of all edge and planar features from this process. Here, we have $F_e \subset \mathbb{F}_e$ and $F_p \subset \mathbb{F}_p$. Features in F_e and F_p are shown in Figure 4.5(c). We divide the 360° range image into 6 sub-images. Each sub-image has a resolution of 300 by 16. $n_{F_e}, n_{F_p}, n_{\mathbb{F}_e}$ and $n_{\mathbb{F}_p}$ are chosen to be 2, 4, 40 and 80 respectively.

4.2.3 Lidar Odometry

The *lidar odometry* module estimates the sensor motion between two consecutive scans. The transformation between two scans is found by performing point-to-edge and point-to-plane scan-matching. In other words, we need to find the corresponding features for points in F_e^t and F_p^t from feature sets $\mathbb{F}_e^{t-1}$ and $\mathbb{F}_p^{t-1}$ of the previous scan. For the sake of brevity, the detailed procedures of finding these correspondences can be found in [79]. However, we note that two changes can be made to improve feature matching accuracy and efficiency:

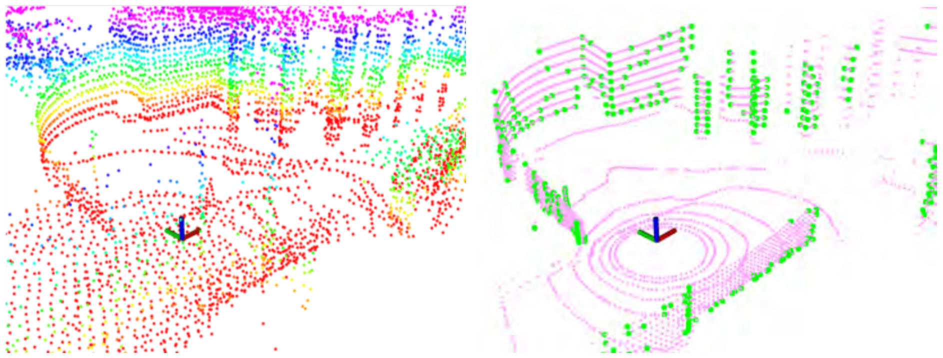
4.2.3.1 Label Matching

Since each feature in F_e^t and F_p^t is encoded with its label after segmentation, we only find correspondences that have the same label from $\mathbb{F}_e^{t-1}$ and $\mathbb{F}_p^{t-1}$. For planar features in F_p^t, only points that are labeled as ground points in $\mathbb{F}_p^{t-1}$ are used for finding a planar patch as the correspondence. For an edge feature in F_e^t, its corresponding edge line is found in the $\mathbb{F}_e^{t-1}$ from segmented clusters. Finding the correspondences in this way can help improve the matching accuracy. In other words, the matching correspondences for the same object are more likely to be found between two scans. This process also narrows down the potential candidates for correspondences.

4.2.3.2 Two-step L-M Optimization

In [79], a series of nonlinear expressions for the distances between the edge and planar feature points from the current scan and their correspondences from the previous scan are compiled into a single comprehensive distance vector. The Levenberg-Marquardt (L-M) method is applied to find the minimum-distance transformation between the two consecutive scans.

We introduce a two-step L-M optimization method here. The optimal transformation T is found in two steps: (1) $[t_z, \theta_{roll}, \theta_{pitch}]$ are estimated by matching the planar features in F_p^t and their correspondences in $\mathbb{F}_p^{t-1}$, (2) the remaining $[t_x, t_y, \theta_{yaw}]$ are then estimated using the edge features in F_e^t and their correspondences in $\mathbb{F}_e^{t-1}$ while using $[t_z, \theta_{roll}, \theta_{pitch}]$ as constraints. It should be noted that though $[t_x, t_y, \theta_{yaw}]$ can also be obtained from the first optimization step, they are less accurate and not used for the second step. Finally, the 6D transformation between two consecutive



(a) Map $\overline{Q}^{t-1}$ (b) Feature sets $\mathbb{F}_e^t$ and $\mathbb{F}_p^t$

Figure 4.7: The lidar mapping module matches features in $\{\mathbb{F}_e^t, \mathbb{F}_p^t\}$ to a surrounding point cloud map $\overline{Q}^{t-1}$ to obtain the pose transformation.

scans is found by fusing $[t_z, \theta_{roll}, \theta_{pitch}]$ and $[t_x, t_y, \theta_{yaw}]$. By using the proposed two-step optimization method, we observe that similar accuracy can be achieved while computation time is reduced by about 35% (Table 4.3).

4.2.4 Lidar Mapping

The *lidar mapping* module matches features in $\{\mathbb{F}_e^t, \mathbb{F}_p^t\}$ to a surrounding point cloud map $\overline{Q}^{t-1}$ to further refine the pose transformation, but runs at a lower frequency. Then the L-M method is used here again to obtain the final transformation. We refer the reader to the description from [79] for the detailed matching and optimization procedure.

The main difference in LeGO-LOAM is how the final point cloud map is stored. Instead of saving a single point cloud map, we save each individual feature set $\{\mathbb{F}_e^t, \mathbb{F}_p^t\}$. Let $M^{t-1} = \{\{\mathbb{F}_e^1, \mathbb{F}_p^1\}, ..., \{\mathbb{F}_e^{t-1}, \mathbb{F}_p^{t-1}\}\}$ be the set that saves all previous feature sets. Each feature set in M^{t-1} is also associated with the pose of the sensor when the scan is taken. Then $\overline{Q}^{t-1}$ can be obtained from M^{t-1} in two ways.

In the first approach, $\overline{Q}^{t-1}$ is obtained by choosing the feature sets that are in the field of view of the sensor. For simplicity, we can choose the feature sets whose sensor poses are within 100m of the current position of the sensor. The chosen feature sets are then transformed and fused into a single surrounding map $\overline{Q}^{t-1}$. This map selection technique is similar to the method used in [79].

We can also integrate pose-graph SLAM into LeGO-LOAM. The sensor pose of each feature set can be modeled as a node in a pose graph. Feature set $\{\mathbb{F}_e^t, \mathbb{F}_p^t\}$ can be viewed as a sensor measurement of this node. Since the pose estimation drift of the lidar mapping module is very low, we can *assume* that there is no drift over a short period of time. In this way, $\overline{Q}^{t-1}$ can be formed by choosing a recent group of feature sets, i.e., $\overline{Q}^{t-1} = \{\{\mathbb{F}_e^{t-k}, \mathbb{F}_p^{t-k}\}, ..., \{\mathbb{F}_e^{t-1}, \mathbb{F}_p^{t-1}\}\}$, where k defines the size of $\overline{Q}^{t-1}$. Then, spatial constraints between a new node and the chosen nodes in $\overline{Q}^{t-1}$ can be added using the transformations obtained after L-M optimization. We can further eliminate drift for this module by performing loop closure detection. In this case, new constraints are added if a match is found between the current feature set and a previous feature set using ICP. The estimated pose of the sensor is then updated by sending the pose graph to an optimization system such as [128]. Note that only the experiment in Section 4.3.4 uses this technique to create its surrounding map.

4.3 Experiments

We now describe a series of experiments to qualitatively and quantitatively analyze two competing methods, LOAM and LeGO-LOAM, on two hardware arrangements, a Jetson TX2 with a Cortex-A57, and a laptop with an i7-4710MQ. Both algorithms are implemented in C++ and executed using the robot operating system (ROS)[129] in Ubuntu Linux[1].

[1]The code for LeGO-LOAM is available at `https://github.com/RobustFieldAutonomyLab/LeGO-LOAM`

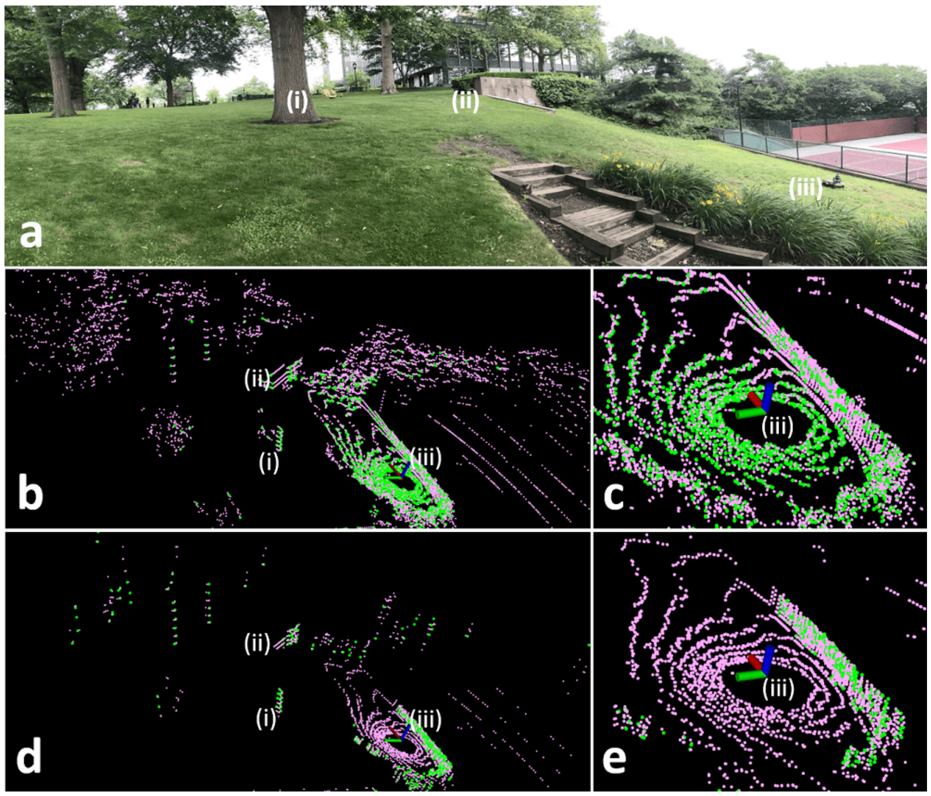


Figure 4.8: Edge and planar features obtained from two different lidar odometry and mapping frameworks in an outdoor environment covered by vegetation. Edge and planar features are colored green and pink, respectively. The features obtained from LOAM are shown in (b) and (c). The features obtained from LeGO-LOAM are shown in (d) and (e). Label (i) indicates a tree, (ii) indicates a stone wall, and (iii) indicates the robot.

4.3.1 Small-Scale UGV Test

We manually drive the robot in an outdoor environment that is covered with vegetation. We first show qualitative comparisons of feature extraction in this environment. Edge and planar features that are extracted from the same scan using both methods are shown in Figure 4.8. These features correspond to the $\{\mathbb{F}_e^t, \mathbb{F}_p^t\}$ that are sent to the lidar mapping module in Section 4.2.4. As is shown in Figure 4.8(d), the number of features from LeGO-LOAM is reduced greatly after point cloud segmentation. The majority of points that are returned from tree leaves are discarded, as they are not

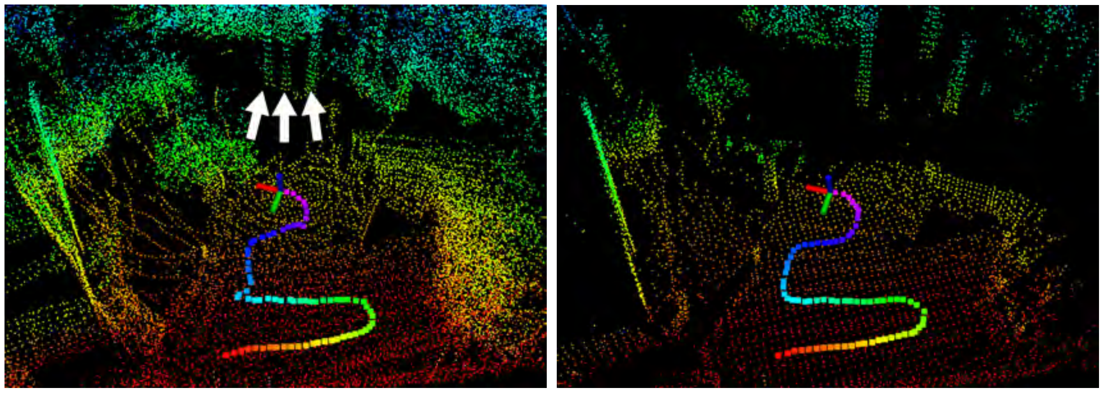
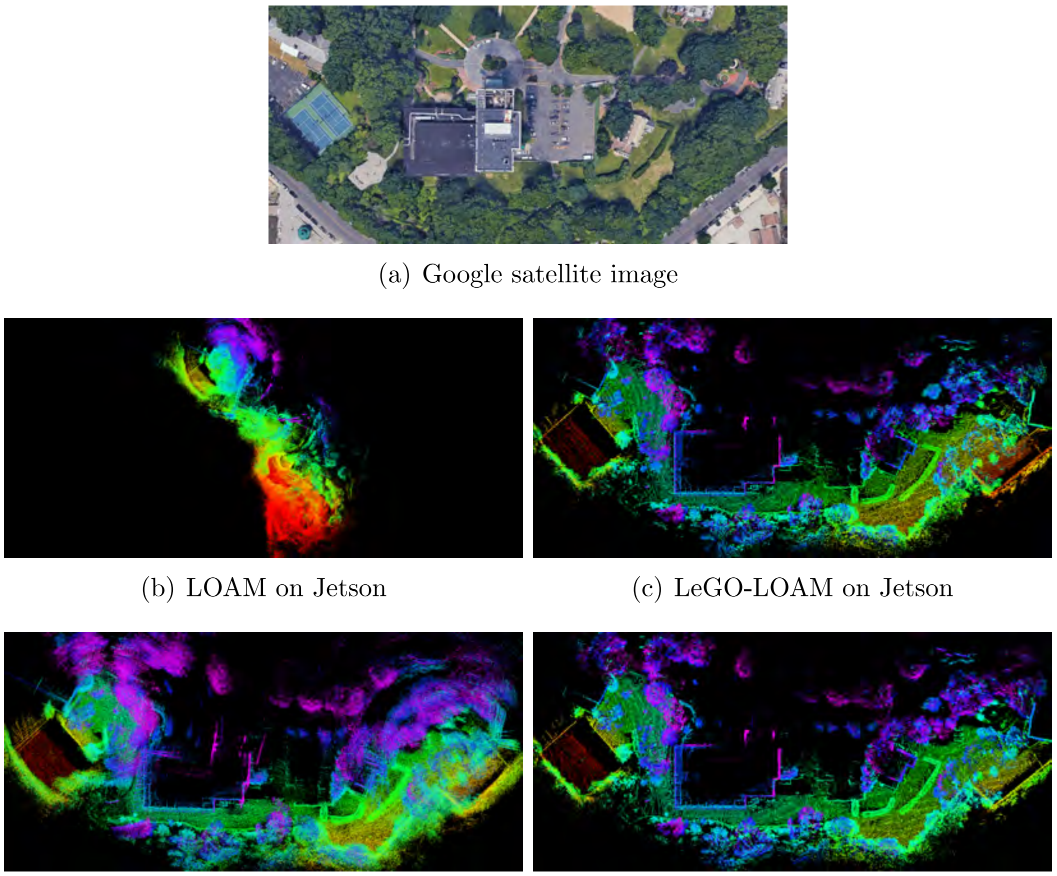

(a) LOAM (b) LeGO-LOAM

Figure 4.9: Maps from both LOAM and LeGO-LOAM over the terrain shown in Figure 4.8(a). The trees marked by white arrows in (a) represent the same tree.

stable features across multiple scans. On the other hand, since the points returned from grass are also very noisy, large roughness values will be derived after evaluation. As a result, edge features are unavoidably extracted from these points using the original LOAM. As is shown in Figure 4.8(c), edge features that are extracted from the ground are often unreliable.

Though we can change the roughness threshold c_{th} for extracting edge and planar features in LOAM to reduce the number of features and filter out unstable features from grass and leaves, we encounter worse results after applying such changes. For example, we can increase c_{th} to extract more stable edge features from an environment, but this change may result in an insufficient number of useful edge features if the robot enters a relatively clean environment. Similarly, decreasing c_{th} will also give rise to a lack of useful planar features when the robot moves from a clean environment to a noisy environment. Throughout all experiments here, we use the same c_{th} for both LOAM and LeGO-LOAM.

Now we compare the *mapping* results from both methods over the test environment. To mimic a challenging potential UGV operational scenario, we perform



(a) Google satellite image

(b) LOAM on Jetson

(c) LeGO-LOAM on Jetson

(d) LOAM on i7

(e) LeGO-LOAM on i7

Figure 4.10: Final point cloud maps of each method on rough terrain.

a series of aggressive yaw maneuvers. Note that both methods are fed an identical initial translational and rotational guess, which is obtained from an IMU, throughout all experiments. The resulting point cloud map after 60 seconds of operation is shown in Figure 4.9. Due to erroneous feature associations caused by unstable features, the map from LOAM diverges twice during operation. The three tree trunks that are highlighted by white arrows in Figure 4.9(a) represent the same tree in reality.

The mapping results of each method on different hardwares are shown in Figure 4.10. The mapped area is mostly covered in vegetation. When running LOAM on Jetson, it is not able to run in real-time due to its demand for processing large amounts of features. Thus the estimated pose diverged many times during the whole mapping process. When running LOAM on a laptop that is equipped with an i7 CPU, it performs better as more computational resources are available. However, it still suffers from wrong feature association in noisy environments. As is shown on Figure 4.10, a wall on the right side of the figure is registered as multiple objects by LOAM. On the other hand, LeGO-LOAM achieves consistent performance when running on both platforms. A visualization of the full mapping process for both odometry methods can be found in the video attachment[2].

4.3.2 Large-Scale UGV Tests

We next perform quantitative comparisons of LOAM and LeGO-LOAM over three large-scale datasets, which will be referred to as experiments 1, 2 and 3. The first two were collected on the Stevens Institute of Technology campus, with numerous buildings, trees, roads and sidewalks. These experiments and their environment are illustrated in Figure 4.11(a). Experiment 3 spans a forested hiking trail, which features trees, asphalt roads and trail paths covered by grass and soil. The environment in which experiment 3 was performed is shown in Figure 4.13. The details of each experiment are listed in Table 4.1. To perform a fair comparison, all of the performance and accuracy results shown for each experiment are averaged over 10 trials of real-time playback of each dataset.

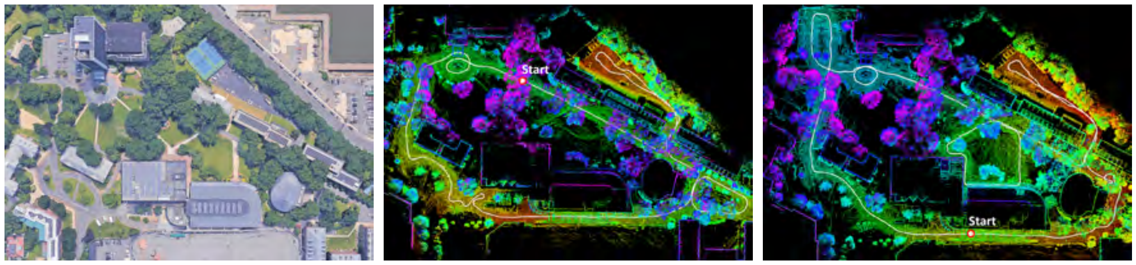
4.3.2.1 Experiment 1

The first experiment is designed to show that both LOAM and LeGO-LOAM can achieve low-drift pose estimation in an urban environment with smooth motion. We avoid aggressive yaw maneuvers, and we avoid driving the robot through sparse areas

[2]`https://youtu.be/O3tz_ftHV48`

Table 4.1: Large-Scale outdoor datasets

Experiment	**Scan Number**	**Elevation Change (m)**	**Trajectory Length (km)**
1	8077	11	1.09
2	8946	11	1.24
3	20834	19	2.71

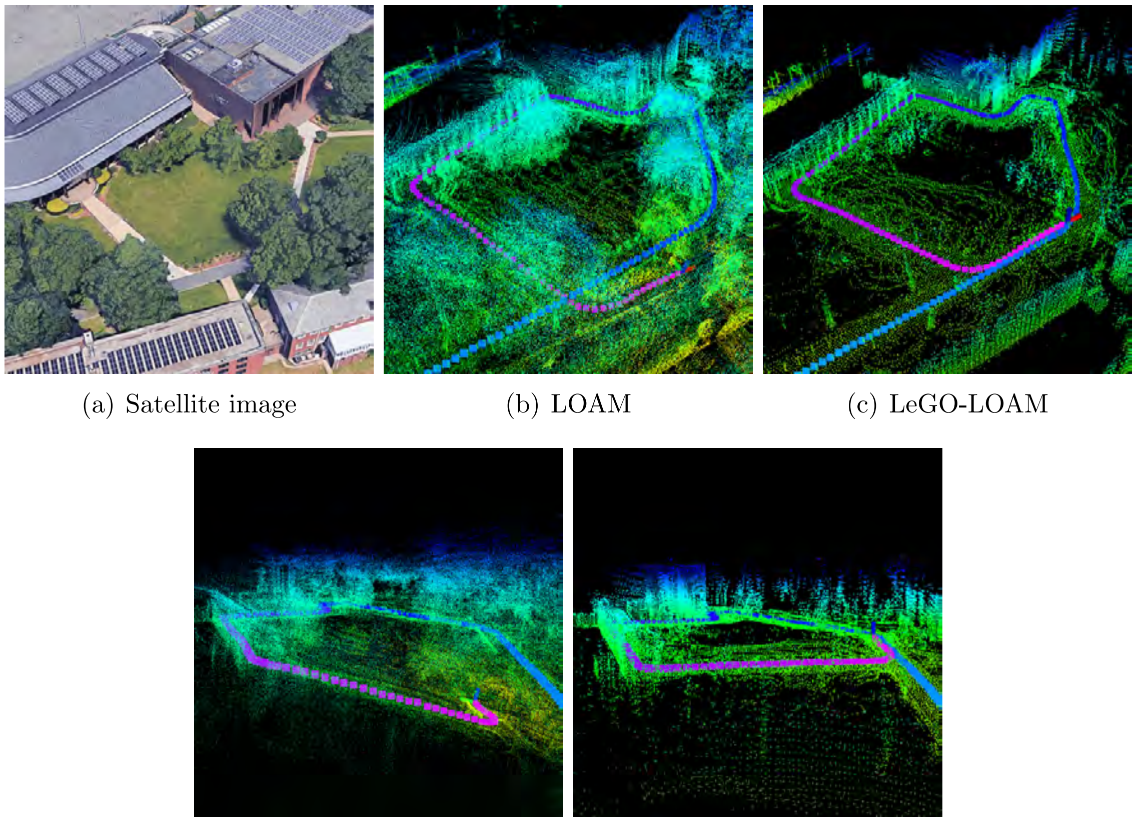


(a) Satellite image (b) Experiment 1 (c) Experiment 2

Figure 4.11: LeGO-LOAM maps from experiments 1 and 2. The color variation in (c) indicates true elevation change. Since the robot's initial position in experiment 1 is on a slope, the color variation in (b) does not represent true elevation change.

where only a few stable features can be acquired. The robot is operated on smooth roads during the whole data logging process. The initial position of the robot, which is marked in Figure 4.11(b), is on a slope. The robot returns to the same position after 807 seconds of travel with an average speed of 1.35m/s.

To evaluate the pose estimation accuracy of both methods, we compare the translational and rotational difference between the final pose and the initial pose. Here, the initial pose is defined as $[0, 0, 0, 0, 0, 0]$ through all experiments. As is shown in Table 4.5, both LOAM and LeGO-LOAM achieve similar low-drift pose estimation over two different hardware arrangements. The final map from LeGO-LOAM, when run on a Jetson, is shown in Figure 4.11(b).


(a) Satellite image (b) LOAM (c) LeGO-LOAM

(d) LOAM (e) LeGO-LOAM

Figure 4.12: A scenario where LOAM fails over a sidewalk crossing the Stevens campus in experiment 2 (the leftmost sidewalk in image (a) above). One end of the sidewalk is supported by features from a nearby building. The other end of the sidewalk is surrounded primarily by noisy objects, i.e., grass and trees. Without point cloud segmentation, unreliable edge and planar features will be extracted from such objects. Images (b) and (d) show that LOAM fails after passing over the sidewalk.

4.3.2.2 Experiment 2

Though experiment 2 is carried out in the same environment as experiment 1, its trajectory is slightly different, driving across a sidewalk that is shown in Figure 4.12(a). This sidewalk represents an environment where LOAM may often fail. A wall and pillars are on one end of the sidewalk - the edge and planar features that are extracted from these structures are stable. The other end of the sidewalk is an open area covered with noisy objects, i.e., grass and trees, which will result in unreliable

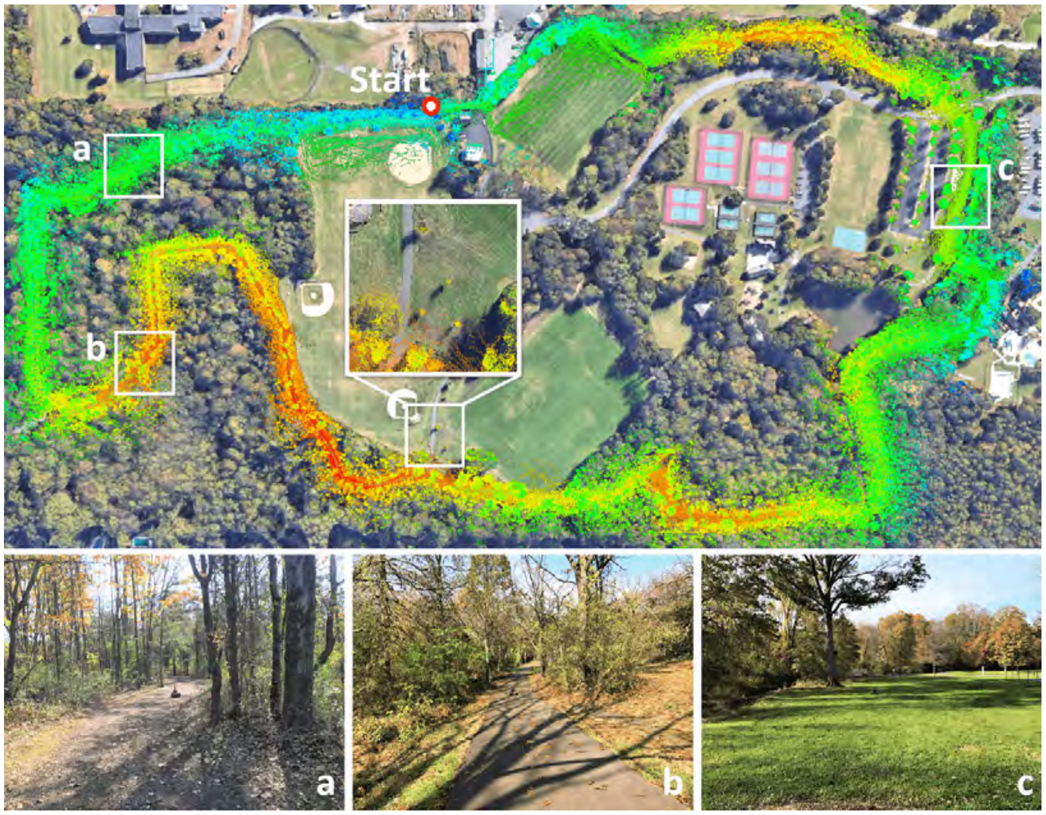


Figure 4.13: Experiment 3 LeGO-LOAM mapping result.

feature extraction. As a result, LOAM's pose estimation diverges after driving over this sidewalk (Figure 4.12(b) and (d)). LeGO-LOAM has no such problem as: 1) no edge features are extracted from ground that is covered by grass, and 2) noisy sensor readings from tree leaves are filtered out after segmentation. An accuracy comparison of both methods is shown in Table 4.5. In this experiment, LeGO-LOAM achieves higher accuracy than LOAM by an order of magnitude.

4.3.2.3 Experiment 3

The dataset for experiment 3 was logged from a forested hiking trail, where the UGV was driven at an average speed of 1.3m/s. The robot returns to the initial position after 35 minutes of driving. The elevation change in this environment is

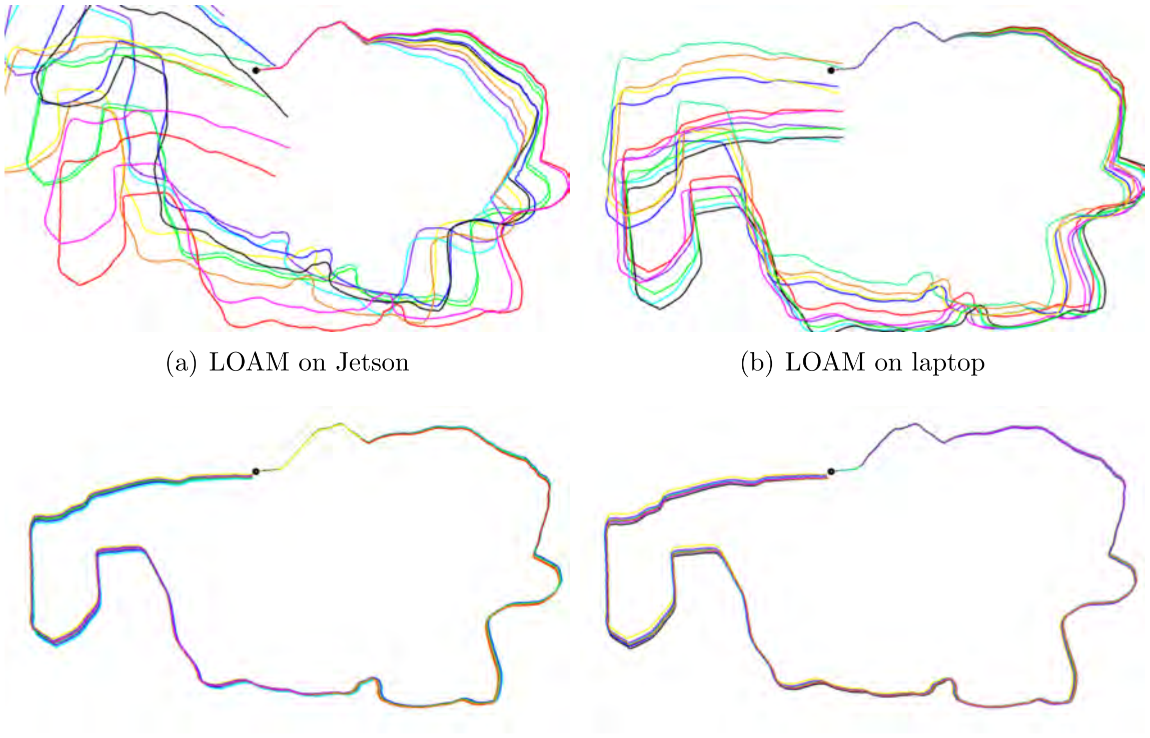
Table 4.2: Average feature content of a scan after feature extraction

Scenario	**Edge Features F_e**		**Planar Features F_p**		**Edge Features $\mathbb{F}_e$**		**Planar Features $\mathbb{F}_p$**	
	LOAM	LeGO-LOAM	LOAM	LeGO-LOAM	LOAM	LeGO-LOAM	LOAM	LeGO-LOAM
1	157	102	323	152	878	253	4849	1319
2	145	102	331	154	798	254	4677	1227
3	174	101	172	103	819	163	6056	1146

about 19 meters. The UGV is driven on three road surfaces: dirt-covered trails, asphalt, and ground covered by grass. Representative images of such surfacess are shown respectively at bottom of Figure 4.13. Trees or bushes are present on at least one side of the road at all times.

We first test LOAM's accuracy in this environment. The resulting maps diverge at various locations on both computers used. The final translational and rotational error with respect to the UGV's initial position are 69.40m and 27.38° on the Jetson, and 62.11m and 8.50° on the laptop. The resulting trajectories from 10 trials on both hardware arrangements are shown in Figure 4.14(a) and (b).

When LeGO-LOAM is applied to this dataset, the final relative translational and rotational errors are 13.93m and 7.73° on the Jetson, and 14.87m and 7.96° on the laptop. The final point cloud map from LeGO-LOAM on the Jetson is shown in Figure 4.13 overlaid atop a satellite image. A local map, which is enlarged at the center of Figure 4.13, shows that the point cloud map from LeGO-LOAM matches well with three trees visible in the open. High consistency is shown among all paths obtained from LeGO-LOAM on both computers. Figure 4.14(c) and (d) show ten trials run on each computer.



(a) LOAM on Jetson (b) LOAM on laptop

(c) LeGO-LOAM on Jetson (d) LeGO-LOAM on laptop

Figure 4.14: Paths produced by LOAM and LeGO-LOAM across 10 trials, and 2 computers, with the experiment 3 dataset.

4.3.3 Benchmarking Results

4.3.3.1 Feature number comparison

We show a comparison of feature extraction across both methods in Table 4.2. The feature content of each scan is averaged over 10 trials for each dataset. After point cloud segmentation, the number of features that need to be processed by LeGO-LOAM is reduced by at least 29%, 40%, 68% and 72% for sets F_e, F_p, $\mathbb{F}_e$ and $\mathbb{F}_p$ respectively.

Table 4.3: Iteration number comparison for LeGO-LOAM

	Scenario	**Original Opt.**		**Two-step Opt.**	
		Iter. Num.	Time	Step 1 Iter. Num	Step 2 Iter. Num
Jetson	1	16.6	34.5	1.9	17.5
	2	15.7	32.9	1.7	16.7
	3	20.0	27.7	4.7	18.9
i7	1	17.3	13.1	1.8	18.2
	2	16.5	12.3	1.6	17.5
	3	20.5	10.4	4.7	19.8

4.3.3.2 Iteration number comparison

The results of applying the proposed two-step L-M optimization method are shown in Table 4.3. We first apply the original L-M optimization with LeGO-LOAM, which means that we minimize the distance function obtained from edge and planar features together. Then we apply the two-step L-M optimization for LeGO-LOAM: 1) planar features in F_p are used to obtain $[t_z, \theta_{roll}, \theta_{pitch}]$ and 2) edge features in F_e are used to obtain $[t_x, t_y, \theta_{yaw}]$. The average iteration number when the L-M method terminates after processing one scan is logged for comparison. When two-step optimization is used, the step-1 optimization is finished in 2 iterations in experiments 1 and 2. Though the iteration count of the step-2 optimization is similar to the quantity of the original L-M method, fewer features are processed. As a result, the runtime for *lidar odometry* is reduced by 34% to 48% after using two-step L-M optimization. The runtime for two-step optimization is shown in Table 4.4.

4.3.3.3 Runtime comparison

The runtime for each module of LOAM and LeGO-LOAM over two computers is shown in Table 4.4. Using the proposed framework, the runtime of the *feature extrac-*

Table 4.4: Runtime of modules for processing one scan (ms)

Scenario		Segmentation		Extraction		Odometry		Mapping	
		LOAM	LeGO-LOAM	LOAM	LeGO-LOAM	LOAM	LeGO-LOAM	LOAM	LeGO-LOAM
Jetson	1	N/A	29.3	105.1	9.1	133.4	19.3	702.3	266.7
	2	N/A	29.9	106.7	9.9	124.5	18.6	793.6	278.2
	3	N/A	36.8	104.6	6.1	122.1	18.1	850.9	253.3
i7	1	N/A	16.7	50.4	4.0	69.8	6.8	289.4	108.2
	2	N/A	17.0	49.3	4.4	66.5	6.5	330.5	116.7
	3	N/A	20.0	48.5	2.3	63.0	6.1	344.9	101.7

Table 4.5: Relative pose estimation error when returning to start

Scenario		Method	Roll	Pitch	Yaw	Total Rot.(°)	X	Y	Z	Total Trans.(m)
Jetson	1	LOAM	1.16	2.63	2.5	3.81	1.33	2.91	0.43	3.23
		LeGO-LOAM	0.46	0.91	1.98	2.23	0.12	0.07	1.26	1.27
	2	LOAM	7.05	5.06	9.4	12.80	7.71	6.31	4.32	10.86
		LeGO-LOAM	0.61	0.70	0.32	0.99	0.04	0.10	0.34	0.36
	3	LOAM	7.55	3.20	26.12	27.38	34.61	56.19	21.46	69.40
		LeGO-LOAM	4.62	5.45	2.95	7.73	5.35	7.95	10.11	13.93
i7	1	LOAM	0.28	1.98	1.74	2.65	0.39	0.03	0.21	0.44
		LeGO-LOAM	0.33	0.17	2.06	2.09	0.03	0.02	0.22	0.22
	2	LOAM	21.49	4.86	4.34	22.46	1.39	2.59	11.63	11.99
		LeGO-LOAM	0.18	0.85	0.64	1.08	0.04	0.12	0.04	0.14
	3	LOAM	6.27	3.08	4.83	8.50	16.84	58.81	10.74	62.11
		LeGO-LOAM	4.57	5.39	3.68	7.96	6.69	7.79	10.76	14.87

tion and *lidar odometry* modules are reduced by one order of magnitude in LeGO-LOAM. Note that the runtime of these two modules in LOAM is more than 100ms on a Jetson. As a result, many scans are skipped because real-time performance is not achieved by LOAM on an embedded system. The runtime of *lidar mapping* is also reduced by at least 60% when LeGO-LOAM is used.

4.3.3.4 Pose error comparison

By setting the initial pose to $[0, 0, 0, 0, 0, 0]$ in all experiments, we compute the relative pose estimation error by comparing the final pose with the initial pose. Rotational

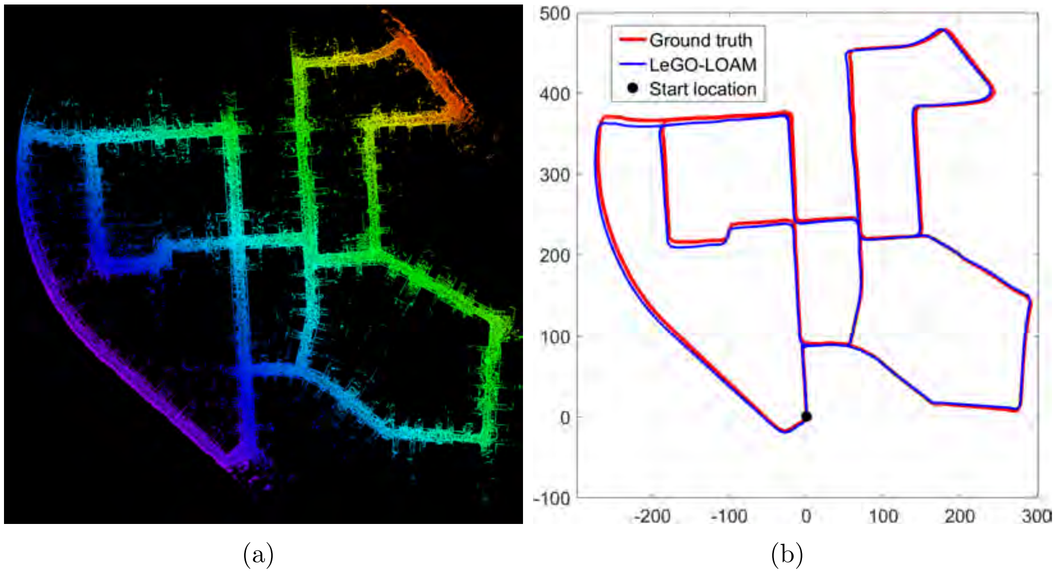


Figure 4.15: LeGO-LOAM, KITTI dataset loop closure test, using the Jetson. Color variation indicates elevation change.

error (in degrees) and translational error (in meters) are listed in Table 4.5 for both methods over both computers. By using the proposed framework, LeGO-LOAM can achieve comparable or better position estimation accuracy with less computation time.

4.3.4 Loop Closure Test using KITTI Dataset

Our final experiment applies LeGO-LOAM to the KITTI dataset [80]. Since the tests of LOAM over the KITTI datasets in [79] run at 10% of the real-time speed, we only explore LeGO-LOAM and its potential for real-time applications with embedded systems, where the length of travel is significant enough to require a full SLAM solution. The results from LeGO-LOAM on a Jetson using sequence 00 are shown in Figure 4.15. To achieve real-time performance on the Jetson, we downsample the scan from the HDL-64E to the same range image that is used in previous section for the VLP-16. In other words, 75% of the points of each scan are omitted before processing. ICP is used here for adding constraints between nodes in the pose graph. The graph is then optimized using iSAM2 [128]. At last, we use the optimized graph to correct the sensor pose and map. More loop closure tests can be found in the video attachment.

4.4 Conclusions

In this chapter, we have proposed LeGO-LOAM, a lightweight and ground-optimized lidar odometry and mapping method, for performing real-time pose estimation of UGVs in complex environments. LeGO-LOAM is lightweight, as it can be used on an embedded system and achieve real-time performance. LeGO-LOAM is also ground-optimized, leveraging ground separation, point cloud segmentation, and improved L-M optimization. Valueless points that may represent unreliable features are filtered out in this process. The two-step L-M optimization computes different components of a pose transformation separately. The proposed method is evaluated on a series of UGV datasets gathered in outdoor environments. The results show that LeGO-LOAM can achieve similar or better accuracy when compared with the state-of-the-art algorithm LOAM. The computation time of LeGO-LOAM is also greatly reduced. Future work involves exploring its application to other classes of vehicles.

Though LeGO-LOAM is especially optimized for pose estimation on ground vehicles, its application could potentially be extended to other vehicles, e.g., unmanned aerial vehicles (UAVs), with minor changes. When applying LeGO-LOAM to a UAV, we would not assume the ground is present in a scan. A scan's point cloud would be segmented without ground extraction. The feature extraction process would be the same for the selection of F_e, $\mathbb{F}_e$ and $\mathbb{F}_p$. Instead of extracting planar features for F_p from points that are labeled as ground points, the features in F_p would be selected from all segmented points. Then the original L-M method would be used to obtain the transformation between two scans instead of using the two-step optimization method. Though the computation time will increase after these changes, LeGO-LOAM is still efficient, as a large number of points are omitted in noisy outdoor environments after segmentation. The accuracy of the estimated feature correspondences may improve, as they benefit from segmentation. In addition, the ability to perform loop closures with LeGO-LOAM online makes it a useful tool for long-duration navigation tasks.

Chapter 5

Learning-Enhanced Perception

5.1 BGK Inference for Terrain Traversability Mapping

5.1.1 Introduction

In this chapter, we propose Bayesian generalized kernel (BGK) inference [130] for solving the traversability mapping problem, with the aid of sparse kernels [131]. We first apply BGK elevation inference to solve the sparse data problem encountered during terrain mapping. Then we relieve the typical computational burden by only performing traversability computations over the elevation data at selected locations. The traversability of locations elsewhere is estimated by BGK traversability inference. This framework enables us to perform online traversability mapping with sparse lidar data and hardware that is compatible with a small UGV. To the best of our knowledge, this is the first application of Bayesian generalized kernel inference to the problem of terrain mapping [5].

5.1.2 Technical Approach

We define the traversability mapping problem and give the details of our solution. Given a sampled 3D point cloud, we first represent the environment as an elevation map. Finely and uniformly discretized planar "grid cells" on the ground are each assigned a height value. We then classify each cell as traversable or non-traversable. To solve this problem efficiently and precisely, we employ the Bayesian kernel inference method of Vega-Brown et al. [130] in two forms: regression, to obtain a dense elevation map m_e; and classification, to determine traversability map m_v.

5.1.2.1 Bayesian Generalized Kernel Inference

Given observations $\mathcal{D} = \{(\mathbf{x}_i, y_i)_{i=1:N}\}$, we seek to infer a probability distribution of a target value parameterized on the latent space Θ :

$$p(y^*|\mathbf{x}^*, \mathcal{D}) \propto \int p(y^*|\boldsymbol{\theta}^*)p(\boldsymbol{\theta}^*|\mathbf{x}^*, \mathcal{D})d\boldsymbol{\theta}^*, \tag{5.1}$$

where

$$p(\boldsymbol{\theta}^*|\mathbf{x}^*, \mathcal{D}) \propto \int_{\boldsymbol{\theta}_{1:N}} \prod_{i=1}^{N} p(y_i|\boldsymbol{\theta}_i)p(\boldsymbol{\theta}_{1:N}, \boldsymbol{\theta}^*|\mathbf{x}_{1:N}, \mathbf{x}^*)d\boldsymbol{\theta}_{1:N} \tag{5.2}$$

is the posterior distribution of the latent parameters associated with the target input. Unlike Gaussian Processes, which assume all parameters Θ are correlated, in Bayesian generalized kernel inference, parameters over the observation input are conditionally independent given target input

$$p(\boldsymbol{\theta}_{1:N}, \boldsymbol{\theta}^*|\mathbf{x}_{1:N}, \mathbf{x}^*) = \prod_{i=1}^{N} p(\boldsymbol{\theta}_i|\mathbf{x}_i, \boldsymbol{\theta}^*, \mathbf{x}^*)p(\boldsymbol{\theta}^*|\mathbf{x}^*), \tag{5.3}$$

which enables us to marginalize latent parameters

$$p(\boldsymbol{\theta}^*|\mathbf{x}^*, \mathcal{D}) \propto \prod_{i=1}^{N} p(y_i|\boldsymbol{\theta}^*, \mathbf{x}^*, \mathbf{x}_i)p(\boldsymbol{\theta}^*|\mathbf{x}^*). \tag{5.4}$$

If we further construct a smooth extended likelihood model, we are able to represent the posterior parameters as follows [130],

$$p(\boldsymbol{\theta}^*|\mathbf{x}^*, \mathcal{D}) \propto \prod_{i=1}^{N} p(y_i|\boldsymbol{\theta}^*)^{k(\mathbf{x}_i, \mathbf{x}^*)}p(\boldsymbol{\theta}^*|\mathbf{x}^*), \tag{5.5}$$

where $k(\cdot, \cdot)$ is a kernel function. The posterior can be exactly determined if the likelihood model is from the exponential family and the corresponding conjugate prior is assumed. Two applications of this inference model are employed here.

5.1.2.2 Bayesian Kernel Elevation Regression

For elevation regression, we assume a Gaussian model $y \sim \mathcal{N}(\mu, \sigma^2)$ with fixed and known variances σ^2. The conjugate prior is also Gaussian $\mu \sim \mathcal{N}(\mu_0, \sigma^2/\lambda)$, where we define λ as a hyperparameter reflecting our confidence in the prior, with $\lambda = 0$ indicating no confidence, and $\lambda \to \infty$ indicating a state of perfect knowledge. Applying Equation 5.5 with the above assumptions,

$$p(\mu^*|\mathbf{x}^*, \mathcal{D}) \propto \prod_{i=1}^{N} \exp\left\{ -\frac{1}{2}\frac{(y_i - \mu)^2}{\sigma^2} k(\mathbf{x}_i, \mathbf{x}^*) \right\} \exp\left\{ -\frac{1}{2}\frac{(\mu - \mu_0)^2}{\sigma^2}\lambda \right\}. \tag{5.6}$$

The mean and variance of the posterior parameters can be shown as

$$\mathbb{E}[\mu^*|\lambda, \mathcal{D}, \mathbf{x}^*] = \frac{\lambda\mu_0 + \sum_{i=1}^{N} k(\mathbf{x}_i, \mathbf{x}^*) y_i}{\lambda + \sum_{i=1}^{N} k(\mathbf{x}_i, \mathbf{x}^*)}, \quad \mathrm{Var}[\mu^*|\lambda, \mathcal{D}, \mathbf{x}^*] = \frac{\sigma^2}{\lambda^*}, \tag{5.7}$$

where $\lambda^* = \lambda + \sum_{i=1}^{N} k(\mathbf{x}_i, \mathbf{x}^*)$. Therefore, we can derive the mean of the posterior predictive distribution, which is given by $\mathbb{E}[y^*|\lambda, \mathcal{D}, \mathbf{x}^*] = \mathbb{E}[\mu^*|\lambda, \mathcal{D}, \mathbf{x}^*]$. Consequently, applying this method to incremental elevation inference is straightforward. At each time instance, the elevation y^* at new location $\mathbf{x}^*$ can be estimated using Equation 5.7 with the new training data $\mathcal{D}$, where $\mathbf{x}$ indicates the discrete locations in m_e that are currently observed, y is the observed elevation, and $\mathbf{x}^*$ represents the map locations that are within distance l of $\mathbf{x}$ in m_e.

5.1.2.3 Bayesian Kernel Traversability Classification

To perform classification, we similarly treat traversability as a Bernoulli distributed binary random variable, i.e. $y \sim \text{Ber}(\theta)$, and we seek to estimate the value of the parameter θ^*. We again adopt a conjugate prior formulation where $\theta \sim \text{Beta}(\alpha_0, \beta_0)$, in which α_0 and β_0 are hyperparameters. The posterior is also a Beta distribution,

$$p(\theta^*|\mathbf{x}^*, \mathcal{D}) \propto \theta^{\alpha^*-1}(1-\theta)^{\beta^*-1}, \tag{5.8}$$

and we have

$$\alpha^* = \alpha_0 + \sum_{i=1}^{N} k(\mathbf{x}_i, \mathbf{x}^*) y_i, \quad \beta^* = \beta_0 + \sum_{i=1}^{N} k(\mathbf{x}_i, \mathbf{x}^*)(1 - y_i). \tag{5.9}$$

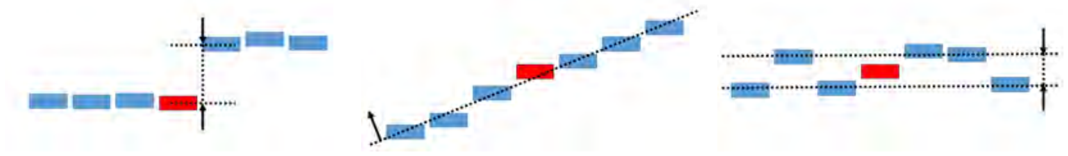
The mean and variance of the posterior predictive distribution are

$$\mathbb{E}[y^*|\alpha_0, \beta_0, \mathcal{D}, \mathbf{x}^*] = \frac{\alpha^*}{\alpha^* + \beta^*}, \quad \text{Var}[y^*|\alpha_0, \beta_0, \mathcal{D}, \mathbf{x}^*] = \frac{\alpha^* \beta^*}{(\alpha^* + \beta^*)^2}. \tag{5.10}$$

5.1.2.4 Traversability Training Data

Compared with the process of elevation regression that is described in Section 5.1.2.2, the training dataset $\mathcal{D}$ for traversability classification is not directly observed - instead our traversability training data are derived from the results of elevation inference. The $\mathbf{x}$ values in $\mathcal{D}$ are the same as those used in Section 5.1.2.2 for elevation inference. The y values in $\mathcal{D}$, which represent the traversability of these cells, are computed by adapting the traversability estimation framework of [85]. The traversability of a cell is determined by three criteria: the step height h, the slope s and the roughness r:

$$v = \alpha_1 \frac{h}{h_{\text{crit}}} + \alpha_2 \frac{s}{s_{\text{crit}}} + \alpha_3 \frac{r}{r_{\text{crit}}}, \tag{5.11}$$



(a) Step height h (b) Slope s (c) Roughness r

Figure 5.1: Three criteria that determines the traversability of a cell in the map: the maximum step height h, the slope s and the roughness r.

where α_1, α_2 and α_3 are weights which sum to 1. h_{crit}, s_{crit} and r_{crit}, which represent the maximum allowable step height, slope and roughness respectively, are critical values that may cause the robot to tip over or become stuck. The traversability v has a range of $[0, 1]$. A small value means the local terrain is flat and smooth, while a large value indicates rough terrain. When the traversability of a cell is estimated, all three criteria must be computed from m_e. If one of the criteria exceeds its critical value, the corresponding cell is labeled non-traversable. For the sake of brevity, the detailed procedures of obtaining step height h, slope s and roughness r can be found in [85].

We note that when the elevation of a grid cell is changed due to the arrival of new measurements, the traversability of all neighboring cells, within at least the radius of the robot, needs to be re-computed. However, the direct computation of traversability using Equation 5.11, which involves plane fitting and eigendecomposition, over all affected cells, is intractable for use in real-time. Thus we only perform this computation for the cells intersected directly by lidar points. We can also incorporate the estimated *elevation variance* into Equation 5.11 for a conservative traversability estimate in the regions where measurements are sparse. Although the elevation variance is not considered in Equation 5.11, we incorporate it into traversability inference in Section 5.1.2.5 below.

5.1.2.5 Traversability Inference

The estimated traversability of cells at new locations $\mathbf{x}^*$, which is the same set used in Section 5.1.2.2, can be obtained from Equation 5.10. The hyperparameters α^* and β^* at each new location are updated using Equation 5.9. Note that since $\mathbf{x}$ and $\mathbf{x}^*$ remain the same as in Section 5.1.2.2, some results of Equation 5.7 can be reused here to save computational resources.

We can also take advantage of variance predictions in a similar fashion to the occupancy mapping problem in [132]. The state of a grid cell in m_v is modeled as follows:

$$\text{state} = \begin{cases} \text{traversable}, & \text{if } v < v_{th},\ \sigma^2 < \sigma^2_{th} \\ \text{non-traversable}, & \text{otherwise} \end{cases} \tag{5.12}$$

in which v is the mean of the predicted traversability at this cell, and v_{th} is the traversability threshold. Additionally thresholded by σ^2_{th}, the cells with variance σ^2 larger than σ^2_{th} will also be labeled as non-traversable. Incorporating the variance into Equation 5.12 naturally gives us conservative traversability estimation in regions where observations are sparse.

5.1.2.6 Sparse Kernel

Exact inference is permitted by Equations 5.7 and 5.10 provided that the requisite kernel computation can be performed exactly. Data structures like k-d trees offer logarithmic time radius queries, which lend themselves to efficient inference if we can limit the search neighborhood. Kernels like the radial basis function kernel have infinite support, leading to approximation error in truncation. Instead, we opt for

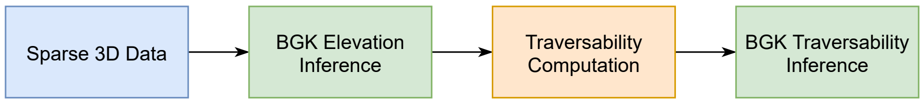


Figure 5.2: Traversability mapping process. Incoming lidar data, in the form of a point cloud, is incorporated as training data, and terrain elevation is estimated for all cells that lie within a designated distance threshold of the points. Traversability is then directly computed for the cells intersected by lidar points. This is used as the training data for traversability inference, applied to all grid cells that are within the same distance threshold used in the previous inference step.

the sparse kernel [131]:

$$k(\mathbf{x}, \mathbf{x}^*) = \begin{cases} \frac{2+\cos(2\pi\frac{d}{l})}{3}(1-\frac{d}{l}) + \frac{1}{2\pi}\sin(2\pi\frac{d}{l}), & \text{if } d \leq l \\ 0, & \text{otherwise} \end{cases} \tag{5.13}$$

where d is the L^2 norm $\|\mathbf{x}-\mathbf{x}^*\|_2$. The kernel has support on the interval $[0, l]$, which allows exact inference to be performed in log-linear time.

5.1.2.7 BGK Traversability Mapping Process

The proposed BGK traversability mapping framework is shown in Figure 5.2. Upon receiving lidar data in the form of a point cloud, we perform BGK elevation inference to obtain a dense height map m_e. Then we directly compute traversability for the cells that were explicitly intersected by the point cloud. BGK traversability inference then estimates the traversability of all cells whose elevation was inferred in the previous step, and the traversability map m_v is produced as output.

We note that an alternative approach for traversability mapping is to perform BGK traversability inference directly, without an elevation inference step. Compared with the proposed framework, however, we have encountered inferior results, due in part to the fact that the resulting traversability estimates are less accurate when their training data is supported by limited, sparse elevation data.

5.1.3 Experiments

We evaluate the proposed terrain traversability mapping framework quantitatively and qualitatively in simulated and real-world environments. The method is implemented in C++ and executed using the robot operating system (ROS) [129] in Ubuntu Linux. The computational hardware is a laptop with an i7 2.5GHz CPU and 16GB memory. Throughout all the experiments, no multi-threading or GPU parallel computation is used for speed improvements.

5.1.3.1 Simulated Data

Gazebo [133] is utilized for two simulated experiments, which feature structured and unstructured environments, since the ground truth of the environment can be known precisely. These two environments are referred to as *City*, an urban environment that features buildings, trees, roads and sidewalks, and *Aerial*, a mountainous environment that features rough terrain and hills. Two volumetric scanning lidar sensors, the Velodyne VLP-16 and HDL-32E, are simulated in Gazebo for data gathering (applied to the *City* and *Aerial* maps respectively). Both sensors operate at 10Hz in all experiments; the real-time experiments are shown in full in the video attachment[1].

Three approaches are compared here to evaluate the proposed method. The first approach implemented is the *baseline* approach. It only processes the raw point cloud data; no inference is used. The second approach, which performs BGK elevation inference and directly computes traversability for all cells where elevation is inferred, is referred to as $BGK + Trav$ for convenience. The proposed framework, which utilizes both BGK elevation and traversability inference in sequence, is referred to as

[1] `https://youtu.be/ewrCyDiWi-8`

BGK^+. For BGK elevation inference, we set $\lambda = 0$, which means that we have no prior knowledge of elevation at any position. For traversability inference, we apply the parameters $\alpha_0 = \beta_0 = 0.001$ to enforce a weak uninformative prior on all cells. The distance threshold l is selected to be 0.3m and 1.0m for our two simulated tests, respectively. The UGV is assumed to have a radius of 0.3m. Note that when we estimate the state of a cell using BGK$^+$, the variance in Equation 5.12 is not used, for the sake of fair comparison with the ground truth.

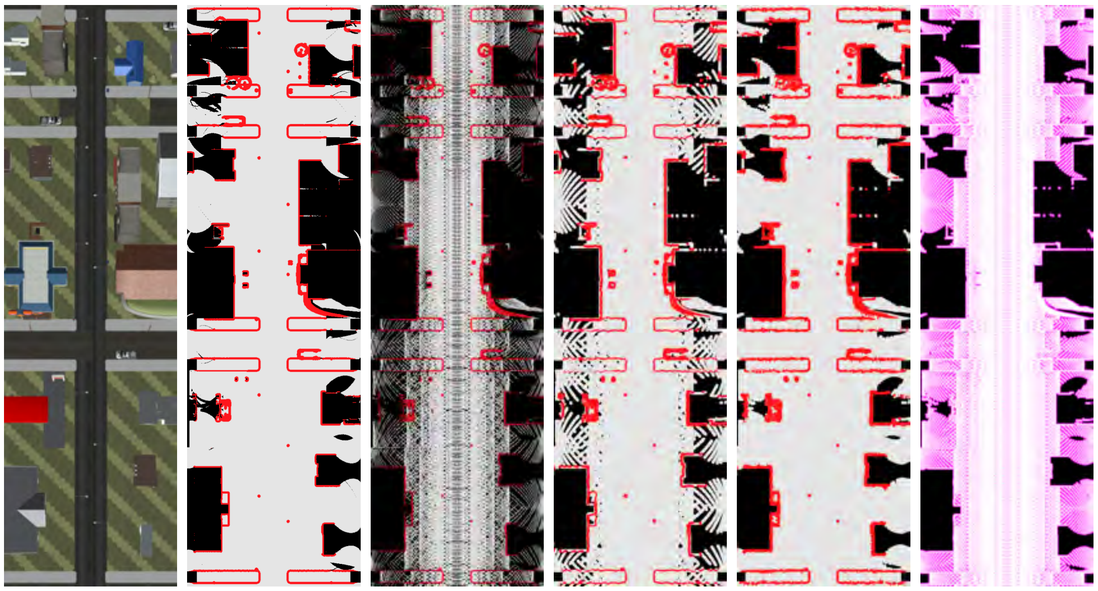
We implement the traversability mapping task as two independent processes. One process performs raw point cloud registration and BGK elevation inference, and the other performs the traversability computation of Equation 5.11 and traversability inference. Since we acquire lidar scans at a rate of 10Hz, scans may be dropped if they take any one of the two processes longer than 0.1 seconds to complete.

5.1.3.2 Structured Environment

The simulated structured environment[2], which is shown in Figure 5.3(a), spans 50 x 210 meters. The ground truth representation of the environment is obtained by taking lidar scans along the center of the road, from top to bottom, at a constant velocity of 1 m/s. The trajectory of the lidar is a straight line with a length of 210 meters, and there are 0.1 meters between scans. As a result, a total number of 2101 scans comprise our ground truth data. However, it is often undesirable to drive this slowly in real-world mapping scenarios. Thus, we will only use scans at 1m intervals for traversability mapping, equivalent to a vehicle that moves at a speed of 36 km/h with a scan rate of 10Hz. As a result, we obtain 211 scans for the traversability mapping comparison.

The ground truth is shown in Figure 5.3(b). Traversable, non-traversable and

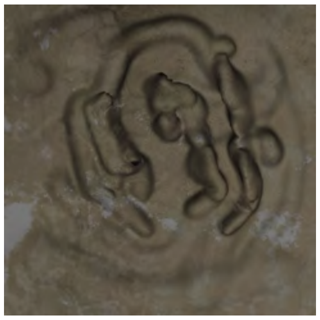
[2]Simulated structured environment: `https://bitbucket.org/osrf/citysim`

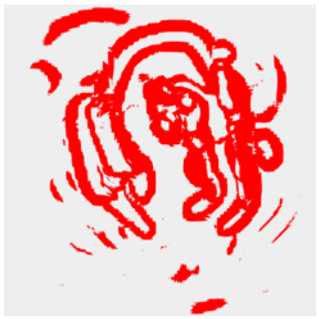


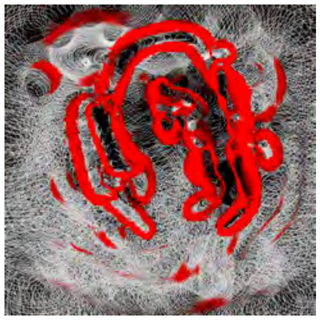
(a) Gazebo (b) GT (c) Baseline (d) *BGK+Trav* (e) BGK$^+$ (f) Variance

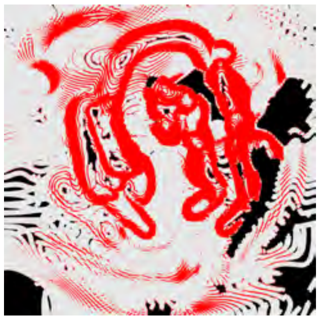
Figure 5.3: Structured environment simulation. The above plots illustrate (a) top view of the simulated urban environment in Gazebo, (b) ground truth of the traversability map, (c) traversability map produced by the baseline approach, (d) map produced by the BGK+Trav approach, (e) the results of the proposed method, BGK$^+$, and (f) the variance map of BGK$^+$ calculated by Equation 5.10, where white color indicates low variance, and magenta indicates high variance.

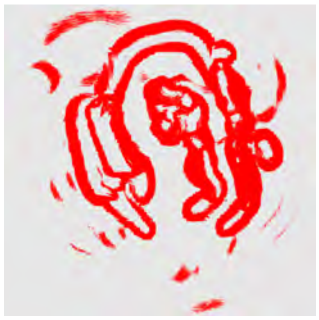
unknown regions are colored gray, red and black respectively. Figure 5.3(c) shows the traversability mapping result by applying the baseline approach. As no inference is performed, it only covers 57% of the area covered by the ground truth. When the BGK+Trav approach is applied, it can cover 93% of the area with the aid of BGK elevation inference. However, due to the intensive traversability computation, real-time performance is not achieved, and 114 out of 211 scans are skipped. At last, we test BGK$^+$, which performs both elevation and traversability inference, on the same data. It closes many of the gaps in Figures 5.3(c) and 5.3(d), achieving 100% map coverage. As expected, the variance map of BGK$^+$, which is obtained from Equation 5.7, shows that the regions covered by fewer observations have higher variance values


(a) Gazebo environment


(b) Ground truth


(c) Baseline


(d) BGK+Trav


(e) BGK$^+$

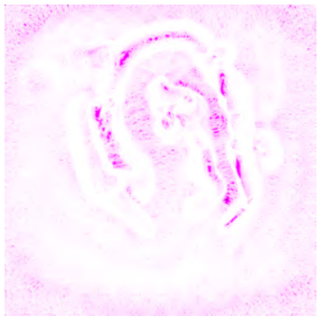

(f) Variance Map

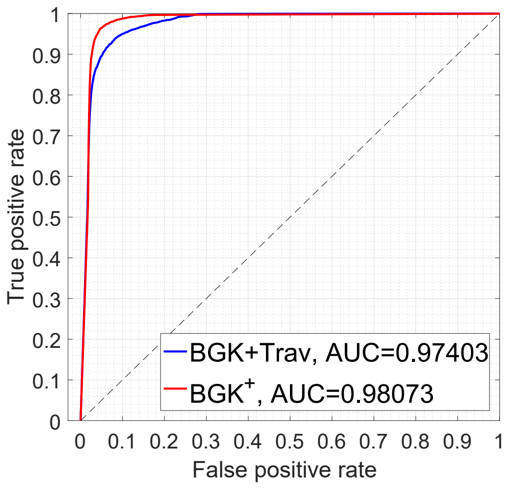
Figure 5.4: Unstructured environment simulation. The *Aerial* simulated terrain model is shown in (a). Traversability maps of the ground truth, baseline, BGK+Trav and BGK$^+$ methods are shown in (b), (c), (d) and (e) respectively. The variance map of BGK$^+$ is shown in (f).

(colored in magenta).

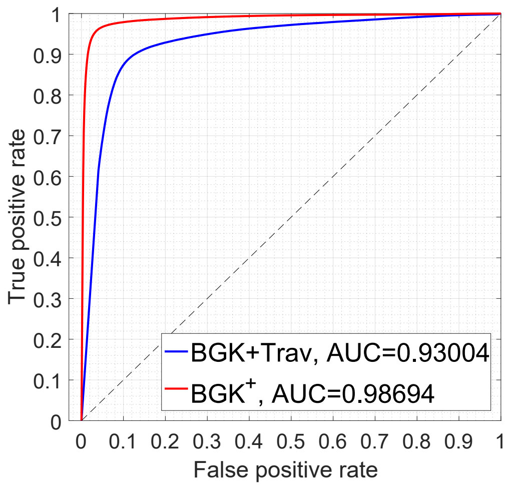
5.1.3.3 Unstructured Environment

The simulated unstructured environment[3], which is shown in Figure 5.4(a), spans 120 x 120 meters. In this test, we simulate an unmanned aerial vehicle's fixed-altitude flyover of the environment to produce a map for a UGV, in which it captures a scan every 10 meters along the latitude and longitude directions. Thus we obtain a total of 169 (13 x 13) scans for this mapping comparison. The resulting traversability maps of each method are shown in Figure 5.4 (d), (e) and (f) respectively. The same color

[3]Simulated unstructured environment: `http://wiki.ros.org/hector_quadrotor`



(a) *City* dataset



(b) *Aerial* dataset

Figure 5.5: Receiver operating characteristic curves. The ROC curves of BGK+Trav and BGK$^+$ methods for the *City* and *Aerial* datasets are shown in (a) and (b) respectively.

scheme of Figure 5.3 is also applied here. The baseline approach can only cover 50% of the area with the available scans. Because the BGK+Trav method suffers from prohibitive computational cost, it skips 84% of the scans. However, 89% of the map is covered due to elevation inference. BGK$^+$ is able to cover 100% of the map while maintaining real-time performance.

5.1.3.4 Benchmarking Results

The receiver operating characteristic (ROC) curves in Figure 5.5 show a predictive performance comparison between BGK+Trav and BGK$^+$. The ROC curves plot the true positive rate against the false positive rate. We compare the predicted traversability to a ground-truth traversability of 1 for a non-traversable cell and 0 for a traversable cell, so that the comparison of inference accuracy is independent of the choice of threshold in Equation 5.12. The plot can be viewed as a plot of predictive performance as a function of the threshold. The area under the curve (AUC) is also

Table 5.1: Quantitative results for different mapping approaches

Dataset	City			Aerial		
Method	Baseline	BGK+Trav	BGK$^+$	Baseline	BGK+Trav	BGK$^+$
Map coverage (%)	57	90	100	50	89	100
Skipped scans	32/211	114/211	1/211	82/169	142/169	3/169
Elevation inference time (s)	N/A	0.043	0.042	N/A	0.069	0.066
Traversability calculation time (s)	0.093	0.174	0.029	0.149	0.583	0.045
Traversability inference time (s)	N/A	N/A	0.017	N/A	N/A	0.020
Mean squared error (sq m)	N/A	0.0089	0.0052	N/A	0.039	0.011

provided in each case for comparison of prediction accuracy. BGK$^+$ outperforms BGK+Trav in both simulated environments with a higher AUC. We also perform tests for the one-step alternative approach that only performs BGK traversability inference without performing BGK elevation inference. The AUCs of this approach are 0.9617 and 0.9581 for the two tests respectively. Thus BGK$^+$, which performs two-step inference, achieves a higher AUC than either one-step inference approach.

Quantitative results for the different methods compared in simulation are summarized in Table 5.1. BGK$^+$ infers the contents of 100% of the visible terrain area despite sparse lidar coverage in both experiments. Since BGK$^+$ only explicitly computes the traversability of cells where new points arrive, it requires less computation time. Mean squared error (MSE) is provided for the elevation inference step of all methods. Since BGK$^+$ skips fewer scans, the MSE of BGK$^+$ is lower than the MSE of BGK+Trav. BGK$^+$ shows advantages in terms of both efficiency and accuracy. We also note that BGK+Trav yields the best results when unlimited computation time is available.

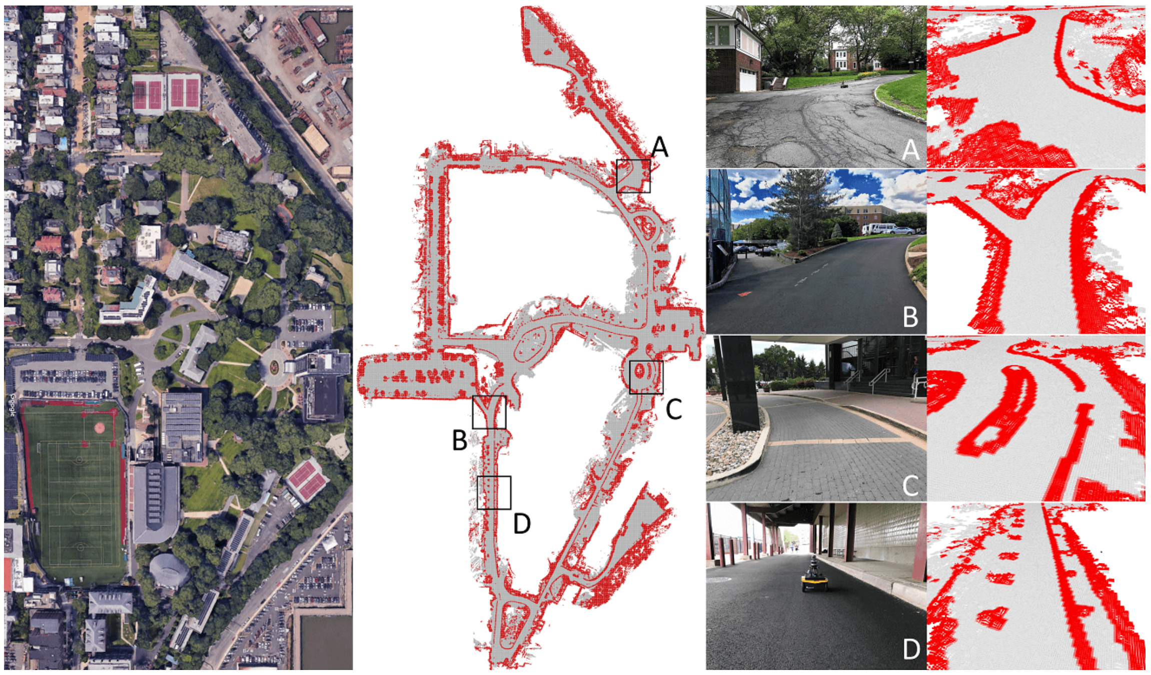


Figure 5.6: Traversability map of a large-scale urban area using BGK$^+$. A satellite image of the mapped area is shown at left. The traversability map from applying BGK$^+$ is shown at center. Representative scenes of the mapped environment are shown at right.

5.1.3.5 Large-scale Urban Environment

We also evaluate our framework in a large-scale urban area. We manually drove a Jackal UGV with a VLP-16 lidar across the area for about 39 min. at a speed of about 1.5 m/s. The mapped area has a maximum elevation change of 15 meters, and spans 285 x 550 meters. A satellite image of the mapped area is shown at the left of Figure 5.6. A total number of 22,636 scans, which are captured at a rate of 10Hz, are used for traversability mapping. The final traversability map from BGK$^+$ is shown in the center of the figure. Four representative images of the environment, along with the traversability map, are shown at right. The proposed system successfully distinguishes non-traversable areas and traversable areas - we note in particular that curbs, which are challenging for autonomous urban navigation, are precisely marked as non-traversable throughout the map.

5.1.4 Conclusions

In this section, we have proposed applying Bayesian generalized kernel inference to terrain traversability mapping. Our framework is unique in its composition, with two sequential inference steps. The first step performs elevation inference to address the sparsity of the available point clouds, and the second step performs traversability inference to relieve the burden of exhaustive traversability computation. The proposed framework is validated using both simulated and real-world data and provides efficiency and accuracy for real-time terrain mapping with lidar.

5.2 Lidar Super-resolution

5.2.1 Introduction

Lidar is an essential sensing capability for many robot navigation tasks, including localization, mapping, object detection and tracking. A typical 3D lidar has multiple channels that revolve at different heights, producing a 3D point cloud with ring-like structure. The number of channels in the sensor determines the vertical density of its point clouds. A denser point cloud from a lidar with more channels can capture the fine details of the environment; applications such as terrain modeling and object detection can benefit greatly from a higher resolution lidar. However, increasing the number of channels can be very costly. For example, the most popular 16-channel lidar, the Velodyne VLP-16, costs around \$4,000. The 32-channel HDL-32E and VLP-32C, and the 64-channel HDL-64E cost around \$30,000, \$35,000 and \$75,000 respectively.

In this chapter, we propose a dedicated deep learning framework for *lidar super-resolution*, which predicts the observations of a high-resolution (hi-res) lidar over a scene observed only by a low-resolution (low-res) lidar. We convert the resulting super-resolution (super-res) point cloud problem in 3D Euclidean space into a super-res problem in 2D image space, and solve this problem using a deep convolutional neural network. Unlike many existing super-res image methods that use high-res real-world data for training a neural network, we train our system using only computer-generated data from a simulation environment. This affords us the flexibility to train the system for operation in scenarios where real hi-res data is unavailable, and allows us to consider robot perception problems beyond those pertaining specifically to driving with passenger vehicles. We investigate the benefits of deep learning in a setting where much of the environment is characterized by sharp discontinuities

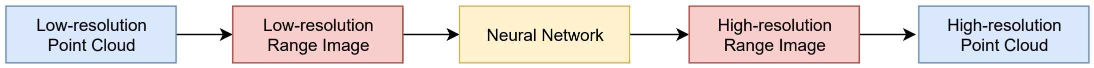

Low-resolution Point Cloud
Low-resolution Range Image
Neural Network
High-resolution Range Image
High-resolution Point Cloud

Figure 5.7: Workflow for lidar super-resolution.

that are not well-captured by simpler interpolation techniques. Furthermore, we use Monte-Carlo dropout [134, 135] to approximate the outputs of a Bayesian Neural Network (BNN) [136], which naturally provides uncertainty estimation to support our inference task.

5.2.2 Technical Approach

This section describes the proposed lidar super-resolution methodology in detail. Since the horizontal resolution of a modern 3D lidar is typically high enough, we only enhance vertical resolution throughout this work. However, the proposed approach, without loss of generality, is also applicable for enhancing the horizontal resolution of a lidar with only a few modifications to the neural network. The workflow of the proposed approach is shown in Figure 5.7. Given a sparse point cloud from a 3D lidar, we first project it and obtain a low-res range image. This range image is then provided as input to a neural network, which is trained purely on simulated data, for upscaling. A dense point cloud is received by transforming the inferred high-res range image pixels into 3D coordinates.

5.2.2.1 Data gathering

Similar to the method proposed in [137], we leverage a rich virtual world as a tool for generating high-res point clouds with simulated lidar. There are many open source software packages, e.g. CARLA, Gazebo, Unity, that are capable of simulating various kinds of lidar on ground vehicles. Specifically, we opt to use the CARLA simulator [138] due to its ease of use and thorough documentation.

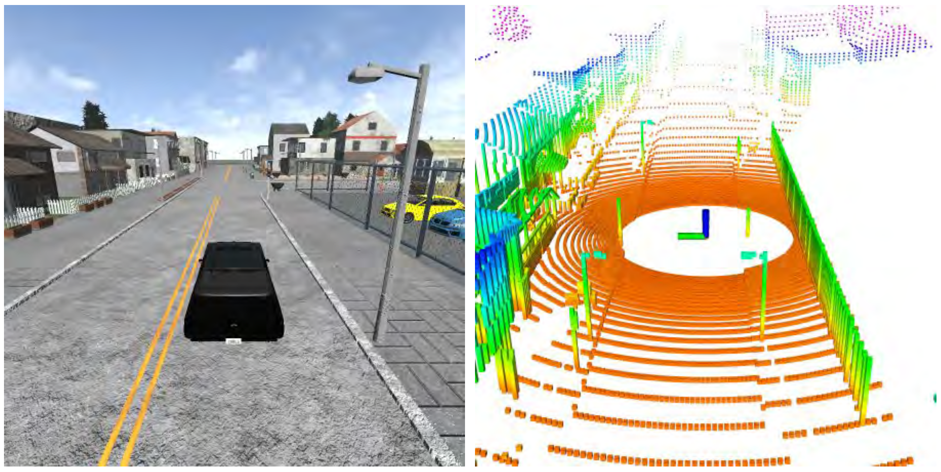

(a) CARLA Town 02 (b) “VLP-64” scan

Figure 5.8: A representative point cloud captured from CARLA using a simulated “VLP-64”. Color variation indicates elevation change.

The first step involves identifying the lidar we wish to simulate. Let us assume we have a VLP-16 and we wish to quadruple (4× upscaling (16 to 64)) its resolution. The VLP-16 has a vertical field of view (FOV) of 30° and a horizontal FOV of 360°. The 16-channel sensor provides a vertical angular resolution of 2°, which is very sparse for mapping. We want to simulate a 64-channel “VLP-64” in CARLA, which also has a vertical and horizontal FOV of 30° and 360° respectively. With the simulated lidar identified, we can either manually or autonomously drive a vehicle in the virtual environment and gather high-res point clouds captured by this simulated lidar. An example of the lidar data produced in CARLA is shown in Figure 5.8.

We note that the simulated high-res lidar should have the same vertical and horizontal FOV as the low-res lidar. For example, we cannot train a neural network that predicts the perception of HDL-64E using the data from VLP-16 because their vertical FOVs are different.

5.2.2.2 Data preparation and augmentation

We then project the simulated high-res point cloud onto a range image, which can be processed by the neural network. A scan from the simulated "VLP-64" 3D lidar will yield a high-res range image with a resolution of 64-by-1024. This high-res range image will serve as the ground truth comprising our training data. Then, we evenly extract 16 rows from this high-res range image and form a low-res range image, which has a resolution of 16-by-1024. This low-res range image is equivalent to the point cloud data captured by a VLP-16 after projection, and comprises the input to the neural network during training. We note that the resolution of the original range image from a VLP-16 sensor varies from 16-by-900 to 16-by-3600 depending on the sensor rotation rate. For the purpose of convenience and demonstration, we choose the horizontal resolution of all range images to be 1024 to accommodate different sensors throughout all experiments. We also "cut" every range scan at the location facing the rear of the vehicle, for the purpose of converting it to a flattened 2D image. This is typically the region of least importance for automated driving and robot perceptual tasks, and is in many cases obstructed by the body of the vehicle.

We then augment the data by performing top-down flipping, horizontal flipping and shifting, and range scaling to account for different environment structures and sensor mounting heights (such as driving on different sides of the road, and underneath structures). To increase prediction robustness, we also vary sensor mounting attitudes during data gathering before augmentation. Finally, the low-res and high-res range images are then normalized to $0 - 1$ and sent to train the neural network.

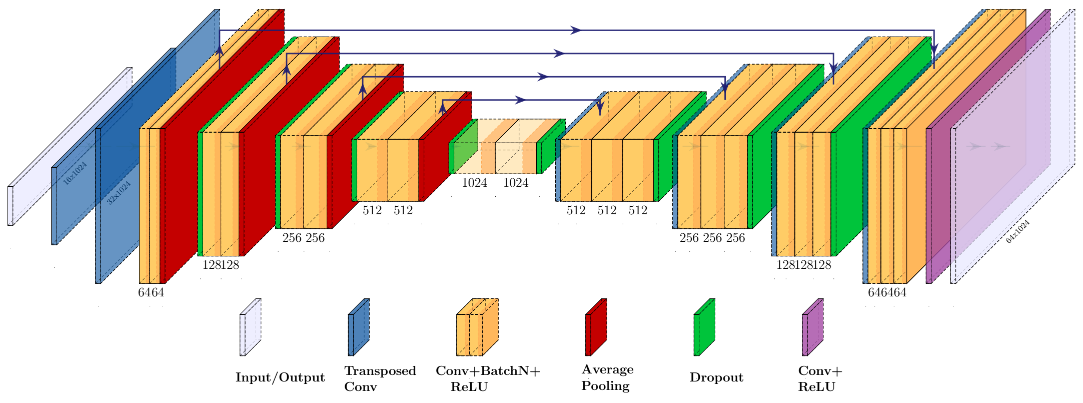


Figure 5.9: Our proposed neural network architecture for range image super-resolution. The network follows an encoder-decoder architecture. Skip-connections are denoted by solid lines with arrows.

5.2.2.3 Neural Network Architecture

The lidar super-res problem can now be formulated as an image super-res problem. Adapted from the encoder-decoder architecture of [139], we configure a neural network for range image super-resolution, shown in Figure 5.9. The input, low-res range image is first processed by two transposed convolutions for increasing the image resolution to the desired level. Then the encoder consists of a sequence of convolutional blocks and average pooling layers for downsampling the feature spatial resolutions while increasing filter banks. On the other hand, the decoder has a reversed structure with transposed convolutions for upsampling the feature spatial resolutions. All convolutions in the convolutional blocks are followed by batch normalization [140] and ReLU [141]. The output layer produces the final high-res range image using a single convolution filter without batch normalization.

5.2.2.4 Noise Removal

We note that we have placed numerous dropout layers before and after the convolutional blocks in Figure 5.9. This is because performing convolutional operations on

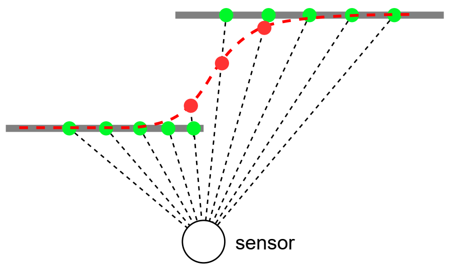


Figure 5.10: Smoothing effects after applying convolutions.

a range image will unavoidably cause smoothing effects on sharp and discontinuous object boundaries [142]. An illustrative example of this smoothing effect is shown in Figure 5.10. Ten range measurements from a lidar channel are shown in a top-down view. The gray lines represent two walls, and the green dots indicate the true range measurements. After convolution, the range measurements are smoothed (shown by the red curve) in places where environmental discontinuities occur. Incorporating smoothed range predictions, such as the three red dots shown, into a robot's point cloud will greatly deteriorate the accuracy of the resulting map.

To address this problem, we novelly apply Monte-Carlo dropout (MC-dropout) for estimating the uncertainty of our range predictions [135]. MC-dropout regularization approximates a BNN by performing multiple feed-forward passes with active dropout at inference time to produce a distribution over outputs [135]. Given observations $\mathcal{D} = \{(\mathbf{x_i}, y_i)_{i=1:N}\}$, we seek to infer a probability distribution of a target value parameterized on the latent space Θ:

$$p(y^*|\mathbf{x}^*, \mathcal{D}) \propto \int p(y^*|\boldsymbol{\theta}^*)p(\boldsymbol{\theta}^*|\mathbf{x}^*, \mathcal{D})d\boldsymbol{\theta}^*, \tag{5.14}$$

where $\boldsymbol{\theta}^*$ are the latent parameters associated with the target input. More specifically, given a test image $\mathbf{x}^*$, the network performs T inferences with the same dropout rate used during training. We then obtain:

$$p(y^*|\mathbf{x}^*) = \frac{1}{T}\sum_{t=1}^{T} p(y^*|\mathbf{x}^*, \boldsymbol{\theta}_t^*), \tag{5.15}$$

with $\boldsymbol{\theta}_t^*$ being the weights of the network for the t^{th} inference. We can evaluate the uncertainty of our range predictions by inspecting the variance of this probability distribution. The final prediction is obtained as follows:

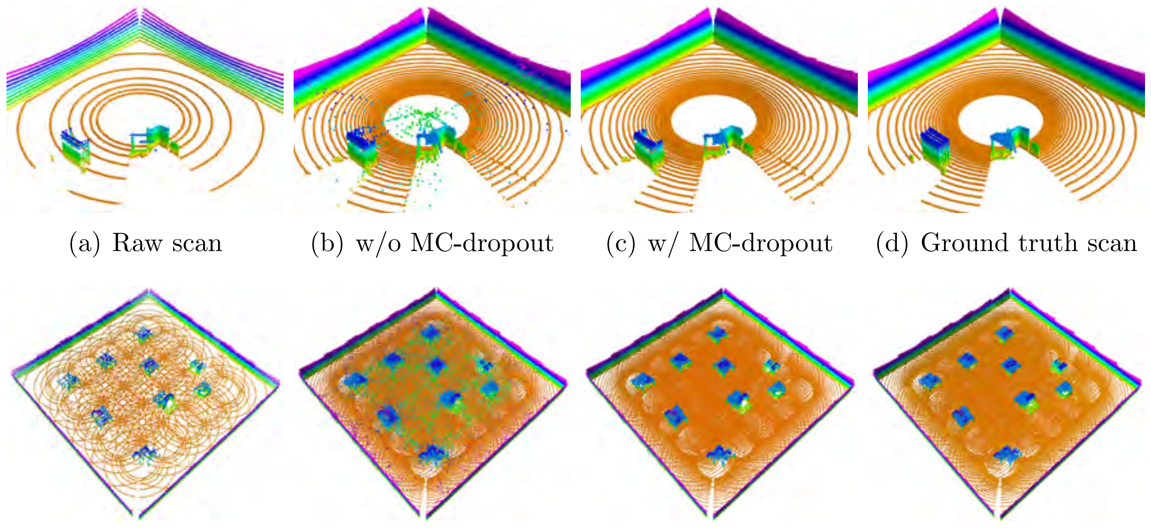
$$y_f^* = \begin{cases} y^*, & \text{if } \sigma < \lambda y^* \\ 0, & \text{otherwise} \end{cases} \tag{5.16}$$

in which y^* is the predicted mean from Equation 5.15, and σ is its standard deviation. The parameter λ causes the noise removal threshold to scale linearly with the predicted sensor range, capturing the fact that the noise level worsens with distance from the sensor. Throughout this work we choose a value of 0.03 for λ, as it is found to give the most accurate mapping results, and we choose an inference quantity T of 50 for all experiments. A larger T yields improved results, as the true probability distribution $p(y^*|\mathbf{x}^*)$ can be better approximated with more predictions.

5.2.3 Experiments

We now describe a series of experiments to quantitatively and qualitatively analyze the performance of our lidar super-resolution architecture. We perform 4× upscaling (16 to 64) for all experiments in this section.

Besides benchmarking various methods in 2D image space using $\mathcal{L}1$ loss, we also show that our method is able to produce dense Octomaps [143] with high accuracy in 3D Euclidean space. 3D occupancy maps can support a variety of robotics appli-



(a) Raw scan (b) w/o MC-dropout (c) w/ MC-dropout (d) Ground truth scan

(e) Baseline (f) w/o MC-dropout (g) w/ MC-dropout (h) Ground truth

Figure 5.11: Scans of an example input (a), predictions by our methods without and with MC-dropout (b and c) and ground truth (d). As is shown in (b), the inferred point cloud is noisy due to points that have high uncertainty, motivating our use of MC-dropout. Occupancy mapping results using a simulated dataset are shown in (e)-(h). Color variation indicates elevation change.

cations, e.g., planning [1, 2, 3] and exploration [144]. However, sparsity in the point cloud of a 3D lidar can leave gaps and inconsistencies in traditional occupancy grid maps, which can be misleading when applied in planning and exploration scenarios. Intuitively, 3D occupancy mapping can benefit greatly from a higher resolution lidar. We use receiver operating characteristic (ROC) curves to benchmark the predictive accuracy (with respect to the binary classification of occupancy) of each method. The ROC curves plot the true positive rate against the false positive rate. We compare all methods to the ground-truth occupancy (0 - free, 0.5 - unknown, 1 - occupied) for all cells in the map. The area under the curve (AUC) is provided for each method for comparison of prediction accuracy. We treat the underlying 64-channel range scan as ground truth, rather than a complete map with all cells filled, because our specific goal is to truthfully compare the range prediction accuracy of each method.

For the simulated experiments described in Section 5.2.3.1 and 5.2.3.2, we use the exact same neural network to demonstrate that the proposed method is capable of performing accurate prediction for sensors with different mounting positions in different environments. The training data for the neural network is gathered from CARLA Town 02, which features an urban environment, by simulating a 64-channel lidar "VLP-64" that has a vertical FOV of 30°. A low-res 16-channel lidar scan is obtained by evenly extracting 16-channel data from the high-res data. The low-res data here is equivalent to the scan obtained from the VLP-16. The training dataset contains 20,000 scans after data augmentation.

Since the real-world Ouster lidar used in Section 5.2.3.3 has a different FOV (33.2°), we gather a new training dataset for network training (see Section 5.2.2.1). Similarly, we simulate a 64-channel lidar, the OS-1-64, in CARLA Town 02 and gather high-res data. The 16-channel data is extracted in the same way as described before. The low-res data here is equivalent to the scan obtained from an OS-1-16 sensor. The training dataset also contains 20,000 scans after data augmentation.

For network training, Adam optimizer [145] is used with a learning rate of 10^{-4} and decay factor of 10^{-5} after each epoch. $\mathcal{L}1$ loss is utilized for penalizing the differences between the network output and ground truth, as it achieves high accuracy, fast convergence and improved stability during training. A computer equipped with a Titan RTX GPU was used for training and testing. The training framework was implemented in Keras [146] using Tensorflow [147] as a backend in Ubuntu Linux.

5.2.3.1 Simulated indoor dataset

We first demonstrate the benefits of applying MC-dropout. We simulate a 64-channel lidar "VLP-64" and gather 25 high-res scans in an office-like environment in Gazebo. The lidar is assumed to be installed on top of a small unmanned ground vehicle

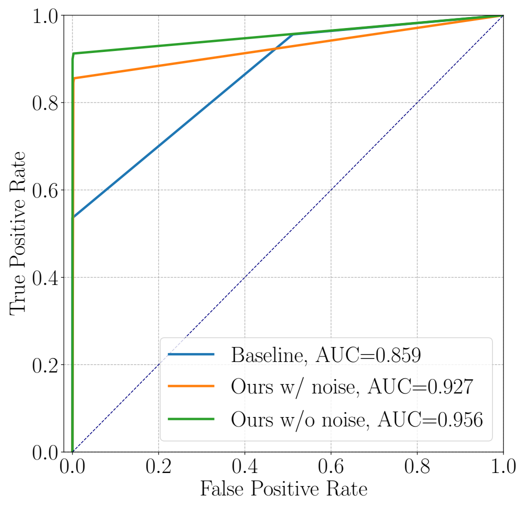


Figure 5.12: ROC curves for the simulated indoor dataset discussed in Section 5.2.3.1.

(UGV). The sensor is 0.5m above the ground. The environment is populated with desks and boxes. The low-res 16-channel testing scans are obtained by evenly extracting 16-channel data from the high-res data. Note that none of these scans are used for network training, nor is the height at which the sensor is mounted.

A representative low-res scan is shown in Figure 5.11(a). Using this scan as input, the predicted high-res scans using our method are shown in Figure 5.11(b) and (c). Without applying MC-dropout, the range prediction is noticeably noisy due to the smoothing effect caused by convolution, hence the scan shown in Figure 5.11(b). After noise removal by applying MC-dropout, the predicted scan shows significantly less noise and resembles the scan of ground truth. The resulting maps are shown in Figure 5.11(e)-(h). All the Octomaps have a resolution of 0.05m. We refer to the approach of using low-res lidar scans to produce an Octomap as the *baseline* approach. The ground truth Octomap is obtained by using the high-res scans. The map of baseline approach is sparse, as no inference is performed. As is shown in Figure 5.11(g), the proposed method is able to produce a dense Octomap that resembles the ground truth Octomap. The AUC and ROC curves of each method are shown in

Table 5.2: Quantitative results for the experiments discussed in Section 5.2.3.2 and 5.2.3.3.

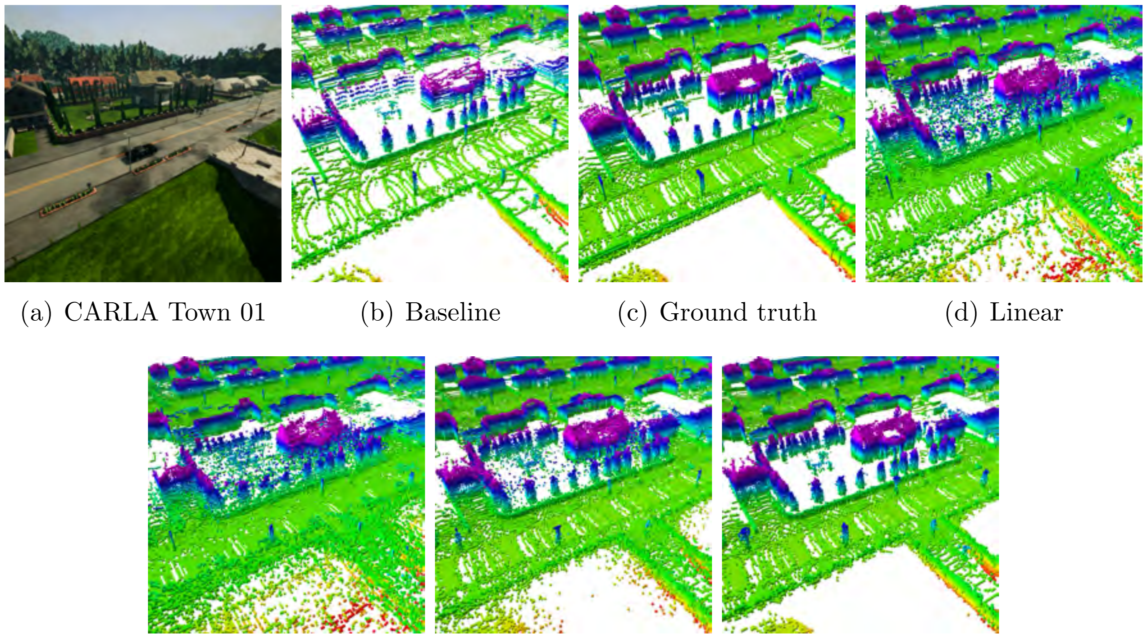
Dataset	Method	$\mathcal{L}1$ Loss	Removed points (%)
CARLA Town 01	Linear	0.0184	N/A
	Cubic	0.0303	N/A
	SR-ResNet	0.0089	12.37
	Ours	0.0087	4.13
Ouster	Linear	0.0324	N/A
	Cubic	0.0467	N/A
	SR-ResNet	0.0211	17.70
	Ours	0.0214	8.37

Figure 5.12. The AUC is improved when applying MC-dropout.

We also note that though the network is trained using data from an outdoor environment, our method is capable of producing meaningful and accurate predictions for indoor usage, with a different sensor mounting scheme. This demonstrates that the network is able to learn the complex mapping between low-res input and high-res output while properly maintaining the structure of surrounding objects.

5.2.3.2 Simulated outdoor dataset

In this experiment, we compare our method with various approaches, which include the standard linear and cubic interpolation techniques and also the state-of-the-art super-resolution approach - SR-ResNet [109], using a simulated large scale outdoor dataset that is gathered in CARLA Town 01. CARLA Town 01 features a suburban environment with roads, trees, houses, and a variety of terrain. The same sensor that is used in 5.2.3.1 is used here. The "VLP-64" sensor, which has a height of 1.8m from the ground, is mounted on top of a full-sized passenger vehicle. We drive the vehicle



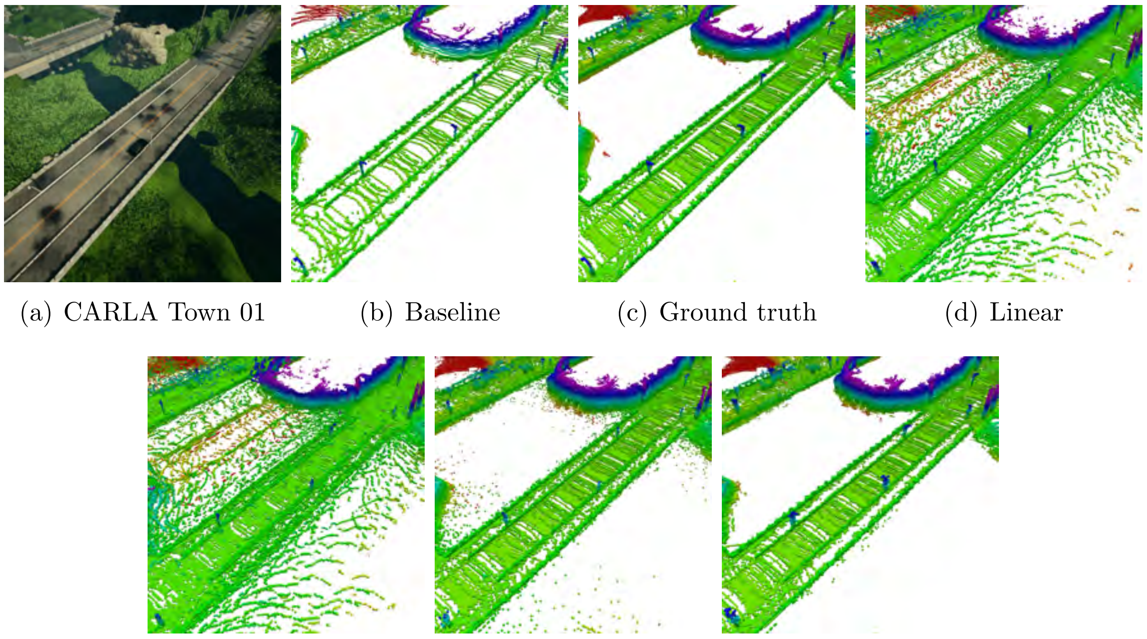
(a) CARLA Town 01 (b) Baseline (c) Ground truth (d) Linear

(e) Cubic (f) SR-ResNet (g) Ours

Figure 5.13: Occupancy mapping results using a simulated dataset from CARLA Town 01. Color variation indicates elevation change.

along a trajectory of 3300m and gather a lidar scan every 10m. Thus this dataset contains 330 scans.

The $\mathcal{L}1$ loss of each method is shown in Table 5.2. The deep learning approaches outperform the traditional interpolation approaches by a large margin. For fair comparison, we also apply MC-dropout on SR-ResNet by adding a dropout layer to the end of each residual block for noise removal. The losses of SR-ResNet and our method are very close. However, the amount of noise removed per scan from SR-ResNet is much larger than our method. Though we can adjust λ in Equation 5.16 to retain more points, the mapping accuracy deteriorates greatly as more noisy points are introduced. We can also decrease the value of λ for SR-ResNet to filter out more noise. The mapping accuracy then also deteriorates, as more areas in the map become unknown.



(a) CARLA Town 01 (b) Baseline (c) Ground truth (d) Linear

(e) Cubic (f) SR-ResNet (g) Ours

Figure 5.14: Occupancy mapping results using a simulated dataset from CARLA Town 01. Color variation indicates elevation change.

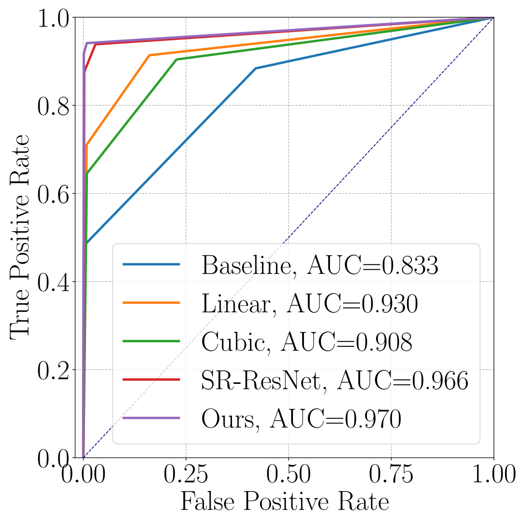


Figure 5.15: ROC curves for the simulated outdoor dataset discussed in Section 5.2.3.2, using 330 scans for all compared methods.

The Octomaps of the competing methods using a low-res scan as input are shown in Figure 5.13. The baseline approach naturally yields the most sparse map.

Though offering better coverage, the Octomaps of the linear and cubic methods are very noisy due to range interpolation between different objects. Though SR-ResNet outperforms linear and cubic interpolation methods in 2D image space by yielding smaller $\mathcal{L}1$ loss, its predictions, when shown in 3D Euclidean space, still contain a great deal of noise at object boundaries. Our method produces a map that is easier to interpret visually, and which also achieves the highest AUC among all methods. The AUC and ROC curves for each method using 330 scans are shown in Figure 5.15.

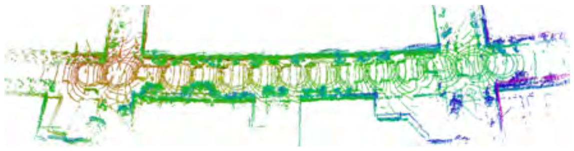
5.2.3.3 Real-world outdoor dataset

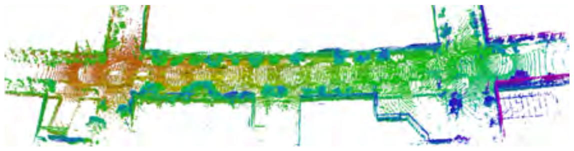
In this experiment, we evaluate the proposed method over one publicly available driving dataset, which we refer to as *Ouster*[4]. The Ouster dataset, which consists of 8825 scans over a span of 15 minutes of driving, is captured in San Francisco, CA using an Ouster OS-1-64 3D lidar. This 64-channel sensor naturally gives us the ground truth for validation, as we only need to extract a few channels of data for generating low-res range image inputs. As is shown in Table 5.2, both SR-ResNet and our method achieve similar $\mathcal{L}1$ loss, which is evaluated over 8825 scans. However, the percentage of removed points of our approach is much less when compared with the results of SR-ResNet. In other words, the predictions of our approach are of lower variance.

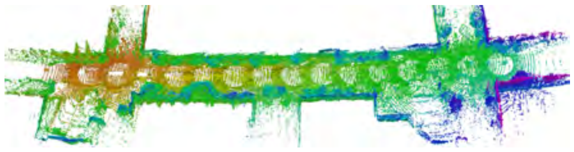
We use 15 scans from this dataset to obtain real-world low-res and high-res lidar scans, which are then used to obtain Octomaps, in the same way that is described in our previous experiments. The scans are registered using LeGO-LOAM [4]. The mapping results at two intersections are shown in Figure 5.17 and 5.18. All the Octomaps have a resolution of 0.3m. The AUC and ROC curves for each method using these 15 scans are shown in Figure 5.17(h). Again, our proposed approach

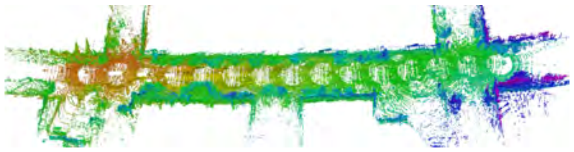
[4]`https://git.io/fhbBt`

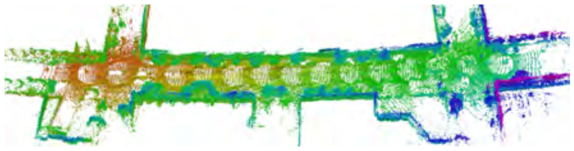
(a) Google earth image


(b) Baseline


(c) Ground truth


(d) Linear


(e) Cubic


(f) SR-ResNet

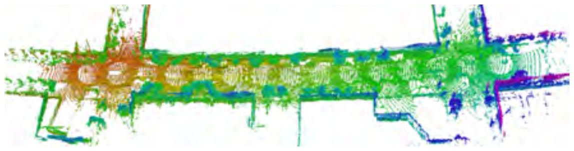
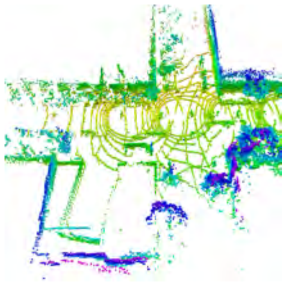

(g) Ours

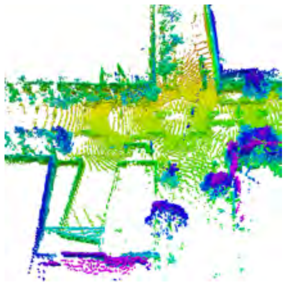
Figure 5.16: The resulting Octomaps from the Ouster dataset. (a) The mapped street in San Francisco. (b) Octomaps produced with 16-channel low-res point clouds. Performing 4× upscaling on the baseline data, the Octomaps generated from inferred 64-channel high-res point clouds using linear, cubic interpolation, SR-ResNet and our method are shown in (d), (e), (f) and (g) respectively. Maps are colored according to elevation.

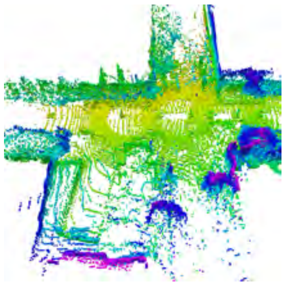
outperforms all methods by achieving the highest AUC.

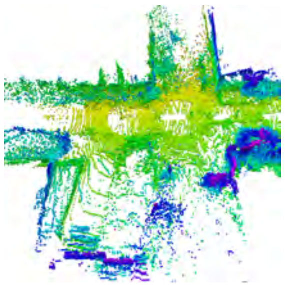
We also show three representative point clouds from Ouster dataset. These three point clouds are captured in a narrow street, an open intersection and a slope surrounded by vegetation. Sub-sampled 16-channel data is used to form a low-res point cloud. As is shown in Figure 5.19, the inferred point clouds resemble the ground truth point clouds well in various environments. The range images that are projected using the point clouds in Figure 5.19 are shown in 5.20.

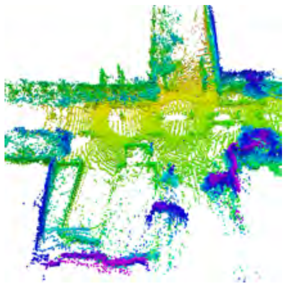
(a) Google earth image

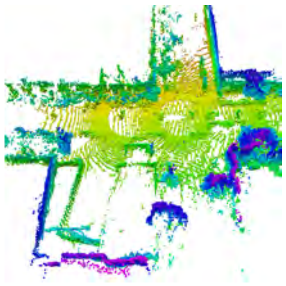

(b) Baseline

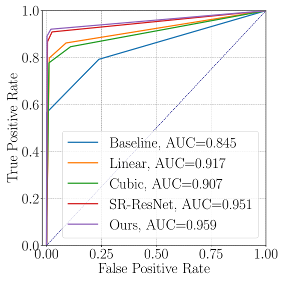

(c) Ground truth


(d) Linear


(e) Cubic


(f) SR-ResNet


(g) Ours

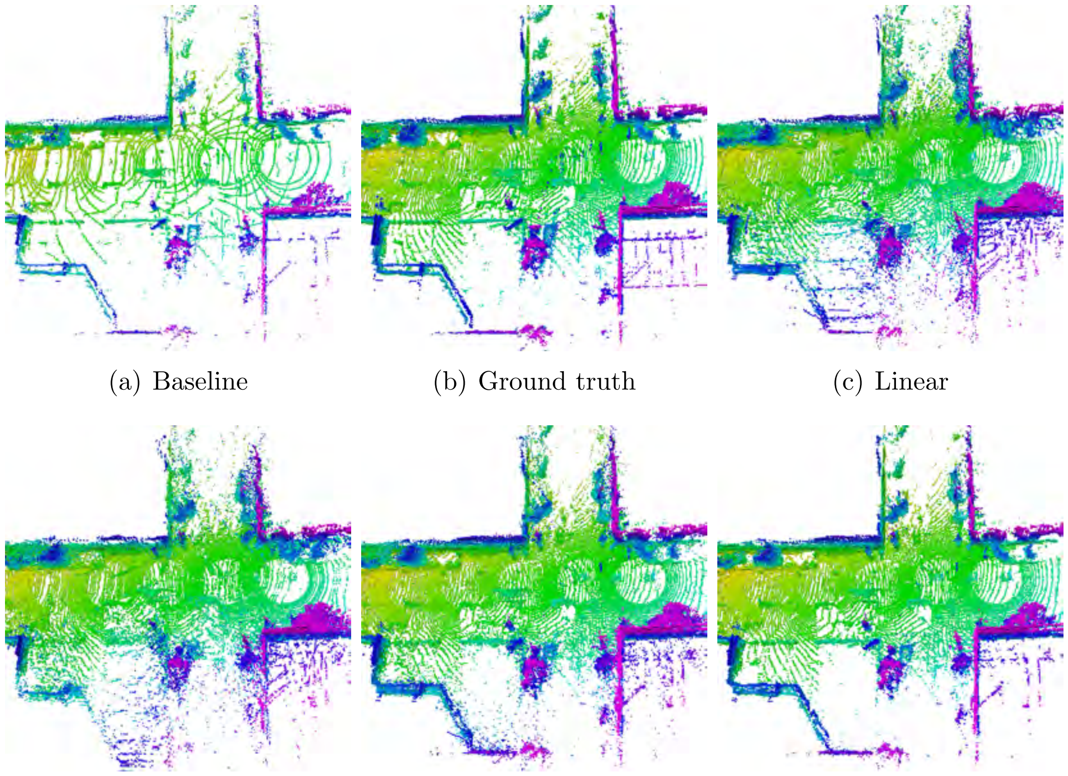


(h) ROC curves

Figure 5.17: Occupancy mapping results using the Ouster dataset. (h) shows the results using 15 scans of the compared methods. Color variation indicates elevation change.

5.2.4 Conclusions

We have proposed a lidar super-resolution method that produces high resolution point clouds with high accuracy. Our method transforms the problem from 3D Euclidean space to an image super-resolution problem in 2D image space, and deep learning is utilized to enhance the resolution of a range image, which is projected back into a point cloud. We train our neural network using computer-generated data, which affords the flexibility to consider a wide range of operational scenarios. We further improve the inference accuracy by applying MC-dropout. The proposed method is evaluated on a series of datasets, and the results show that our method can produce realistic high resolution maps with high accuracy. In particular, we evaluate the super-resolution framework through a study of its utility for *occupancy mapping*, since

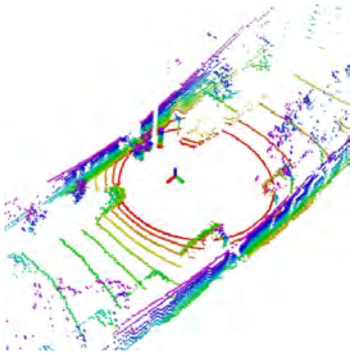


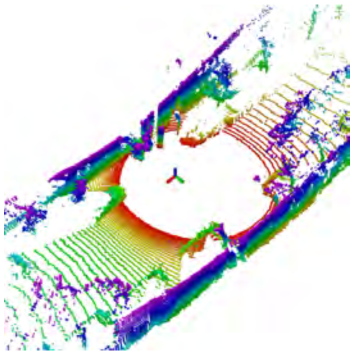
(a) Baseline (b) Ground truth (c) Linear

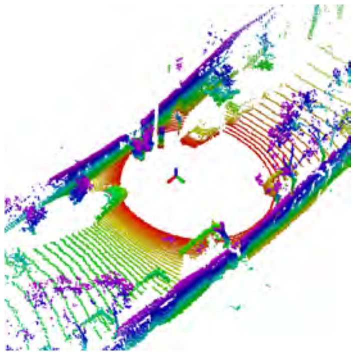
(d) Cubic (e) SR-ResNet (f) Ours

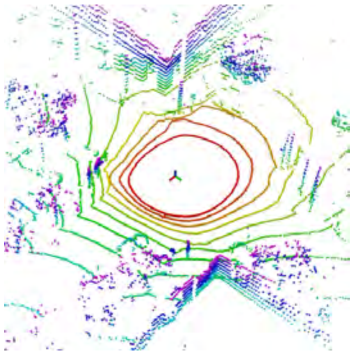
Figure 5.18: Local mapping results at an intersection (right side of Figure 5.16(a)) using Ouster dataset. Color variation indicates elevation change.

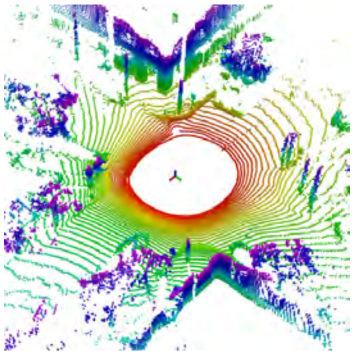
this is a widely useful perceptual end-product for which interpolation may occur in various locations along a robot's perceptual pipeline. In addition to the appealing generalizability of up-scaling at the front end, by predicting the measurements of a higher-resolution sensor, our approach also achieves superior accuracy in the end-stage maps produced, when compared with simpler methods that interpolate in Euclidean space.

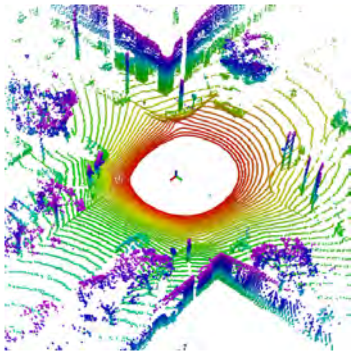

(a) Input - 1

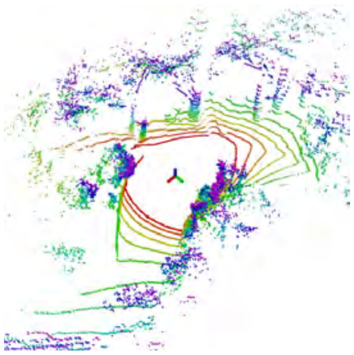

(b) Predicted - 1

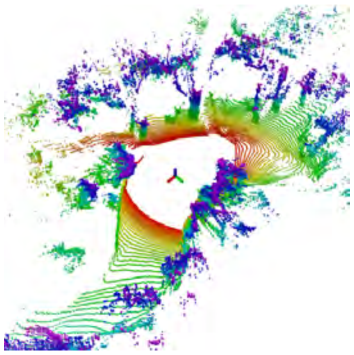

(c) Ground truth - 1


(d) Input - 2


(e) Predicted - 2


(f) Ground truth - 2


(g) Input - 3


(h) Predicted - 3

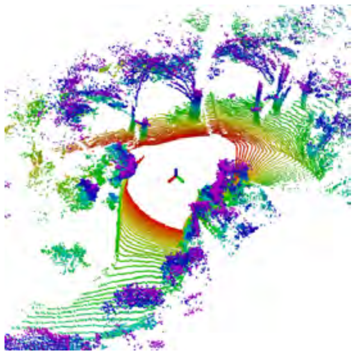
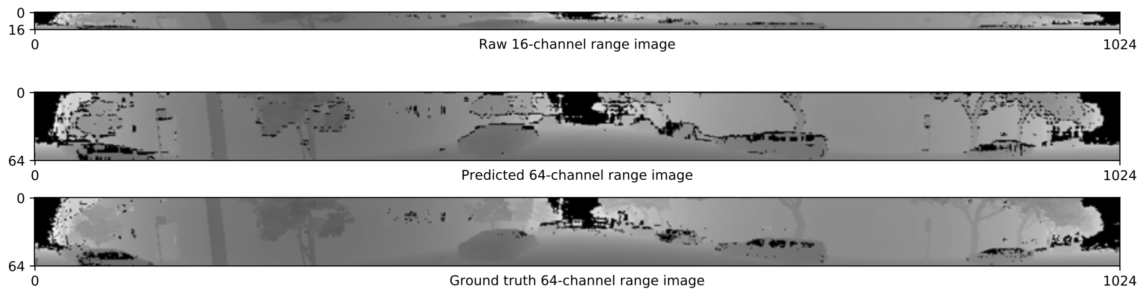
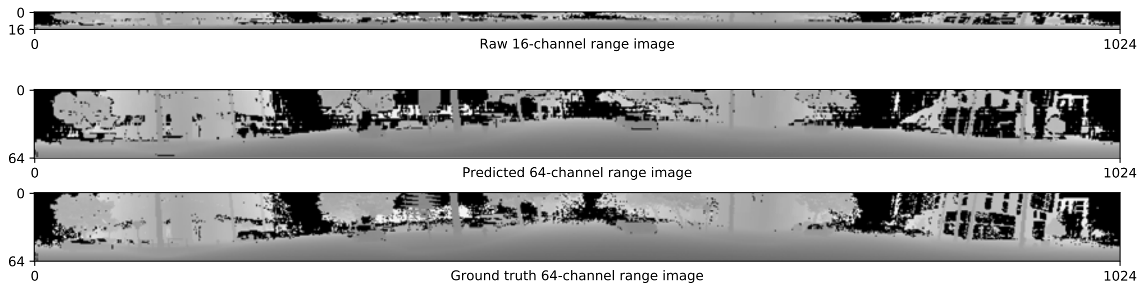

(i) Ground truth - 3

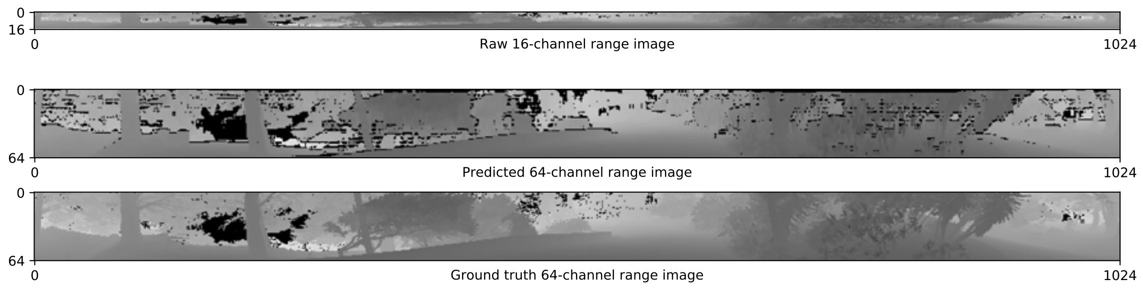
Figure 5.19: Visualization of several representative point clouds from the Ouster dataset. The low density point clouds before inference are shown in (a), (d) and (g). The inferred high-res point clouds (4× upscaling) are shown in (b), (e) and (h). The ground truth point cloud captured by the lidar is shown in (c), (f) and (i). Color variation indicates lidar channel index. The three representative point clouds are captured in a narrow street, a slope surrounded by vegetation and an open intersection. We can observe that objects such as buildings, roads and pillars are inferred well.


0
16
0
Raw 16-channel range image
1024
0
64
0
Predicted 64-channel range image
1024
0
64
0
Ground truth 64-channel range image
1024

(a) Range images for scene 1 shown in Fig. 5.19


0
16
0
Raw 16-channel range image
1024
0
64
0
Predicted 64-channel range image
1024
0
64
0
Ground truth 64-channel range image
1024

(b) Range images for scene 2 shown in Fig. 5.19


0
16
0
Raw 16-channel range image
1024
0
64
0
Predicted 64-channel range image
1024
0
64
0
Ground truth 64-channel range image
1024

(c) Range images for scene 3 shown in Fig. 5.19

Figure 5.20: Range images of the projected point clouds shown in Figure 5.19. Black color indicates a range value of zero, for which no points are added to the point cloud. Since performing convolutional operations on a range image will unavoidably cause smoothing effects on sharp and discontinuous object boundaries, we apply MC-dropout to identify these erroneous predictions. Accordingly, the range predictions with high variance are removed.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

In this thesis we worked towards developing several navigation algorithms that feature minimalistic and learning-enabled design for unmanned ground vehicles. These navigation algorithms, which tackle challenging problems in multi-objective path planning, real-time localization, and sparse sensing in perception, bring reliable autonomy to UGVs for their applications in various environments.

In Chapter 3, we presented three novel multi-objective path planning algorithms that leverage lexicographic optimization method for various planning scenarios. In Chapter 3.1, we proposed MR-RRT* that plans minimum risk paths in accordance with primary and secondary cost criteria. MR-RRT* affords the user the flexibility to tune the relative importance of the alternate cost criteria, while adhering to the requirements for asymptotically optimal planning with respect to the primary cost. In Chapter 3.2, we described MM-RRT* for robot path planning under localization uncertainty. The algorithm builds and maintains a tree that is shared in state space and belief space, with a single belief per robot state. The algorithm offers a compelling alternative to sampling-based algorithms with additive cost representations of uncertainty, which will penalize high-precision navigation routes that are longer in duration. In Chapter 3.3, we characterized and proposed advances in the technique of Belief Roadmap Search (BRMS), the process of searching a roadmap in belief space for robot motion planning under localization uncertainty. We proposed a best-first implementation of BRMS, in contrast to the standard breadth-first implementation, which we showed to improve the computational cost of search by up to 49% by eliminating unnecessary node expansions.

In Chapter 4, we proposed LeGO-LOAM for real-time pose estimation with ground vehicles. LeGO-LOAM is lightweight, as it can achieve real-time pose estimation on a low-power embedded system, Jetson TX2. LeGO-LOAM is ground-optimized, as it leverages the presence of the ground in its segmentation and optimization steps. We compared the performance of LeGO-LOAM with a state-of-the-art method, LOAM, using datasets gathered from variable-terrain environments with ground vehicles, and showed that LeGO-LOAM can achieve similar or better accuracy with reduced computational expense.

In Chapter 5, we introduced two novel learning-enhanced perception methods for sparse sensing problem. We first presented a new approach for traversability mapping with sparse lidar scans collected by ground vehicles, which leverages probabilistic inference to build descriptive terrain maps. We applied Bayesian generalized kernel inference sequentially to the problems of terrain elevation and traversability inference. We verified the capabilities of the approach over a variety of datasets, demonstrating its suitability for online use in real-world applications. Then, we proposed a framework for lidar super-resolution to increase the apparent resolution of a physical lidar. The framework is especially designed for ground vehicles driving on roadways. To increase the resolution of the point cloud captured by a sparse 3D lidar, we converted this problem from 3D Euclidean space into an image super-resolution problem in 2D image space, which is solved using a deep convolutional neural network. The deep network is trained completely using the data obtained from a driving simulator. We also improved the prediction accuracy by novelly applying Monte-Carlo dropout and removed the predictions with high uncertainty. The validity of the proposed framework was tested using several simulated and real-world datasets.

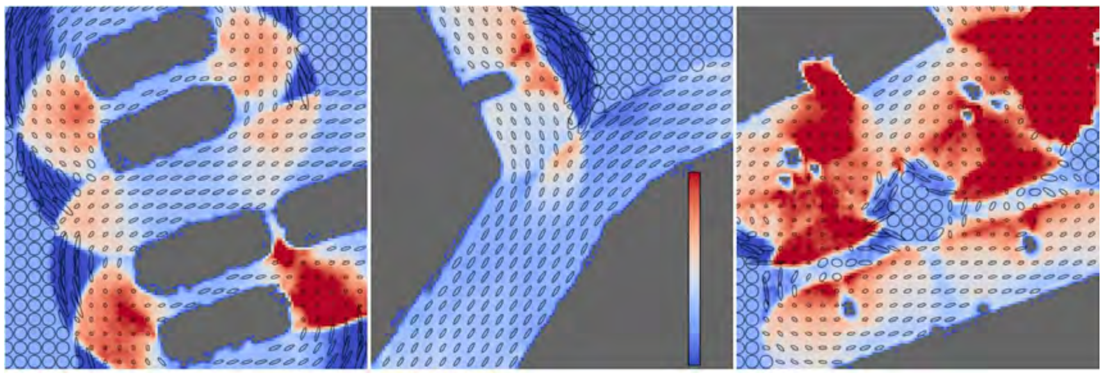


Figure 6.1: Localizability map. The ellipse represents the statistically derived error covariance associated with scan registration. In the areas where no lidar data is available, the ellipse degenerates to a circle. Grey areas indicate obstacles. Color variation indicates $\underline{\lambda}(\mathbf{H}_k'\mathbf{R}_k^{-1}\mathbf{H}_k)$ change.

6.2 Future Work

6.2.1 Deep Learning-Accelerated Planning Under Uncertainty

In Chapter 3, we used eigenvalue bound ℓ as an uncertainty metric for planning under uncertainty. Using this metric offers a small improvement in computational efficiency over other metrics. However, propagating ℓ using Equation 3.9 is still non-trivial as it involves calculating maximum or minimum eigenvalue for $\overline{\lambda}(\mathbf{F}_k\mathbf{F}_k')$, $\overline{\lambda}(\mathbf{Q}_k)$ and $\underline{\lambda}(\mathbf{H}_k'\mathbf{R}_k^{-1}\mathbf{H}_k)$. The first two terms can be obtained relatively easily as the state transition model and the process noise is known in most cases. However, the calculation of $\underline{\lambda}(\mathbf{H}_k'\mathbf{R}_k^{-1}\mathbf{H}_k)$ is non-trivial when a more complex sensor is used for observation, e.g., lidar. When a lidar is used for localization in our problem, we need to forward simulate the sensor reading for each filter update step. The computation of this process is intractable as obtaining the measurement noise matrix $\mathbf{R}$ involves computing the statistically derived localization error for each sensor reading. Thus, we propose to use a deep learning-aided technique to accelerate the evaluation of $\underline{\lambda}(\mathbf{H}_k'\mathbf{R}_k^{-1}\mathbf{H}_k)$. More specifically, we will use a deep neural network to predict the

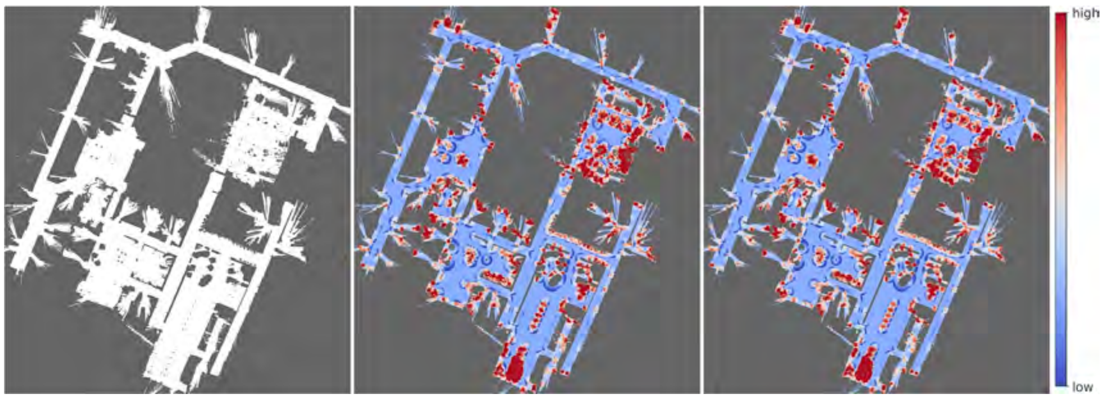


Figure 6.2: Localizability map for Willow Garage. Grey areas indicate obstacles. Color variation indicates $\underline{\lambda}(\mathbf{H}_k'\mathbf{R}_k^{-1}\mathbf{H}_k)$ change.

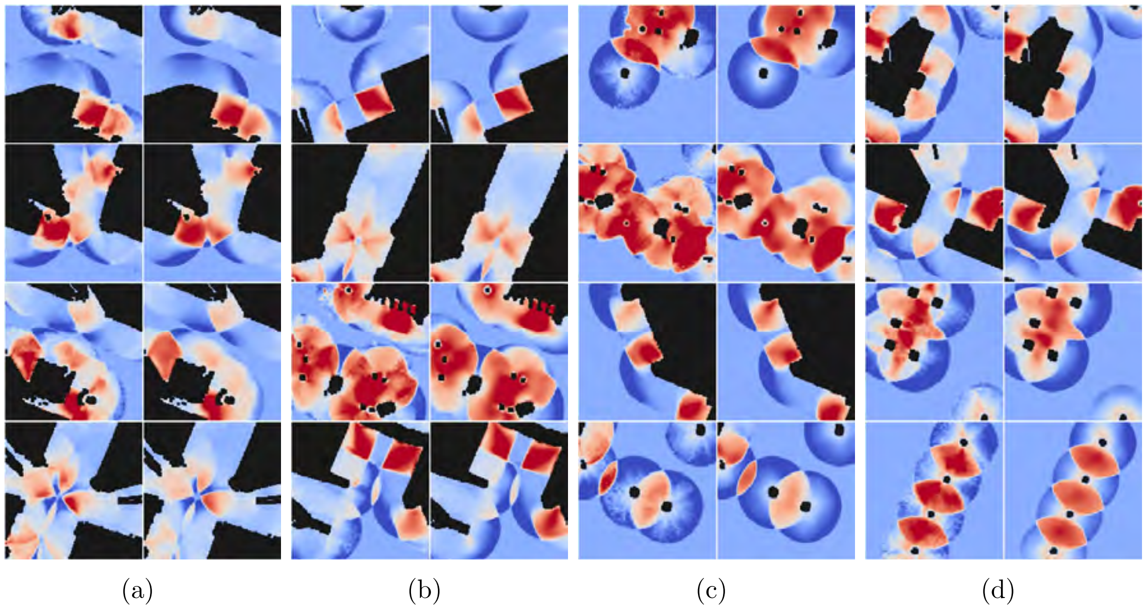


Figure 6.3: Local localizability map comparisons. In each sub-figure, the ground truth localizability map is shown in the left column. The predicted localizability map using our approach is shown in the right column. The same color scheme that is used in Figure 6.2 is applied here.

localizability map, which is a representation of $\underline{\lambda}(\mathbf{H}_k'\mathbf{R}_k^{-1}\mathbf{H}_k)$, for planning under uncertainty problems using a 2D occupancy grid map.

To train a neural network for solving such problem, we generate localizability maps using the approach introduced in [148]. To obtain the value of $\underline{\lambda}(\mathbf{H}_k'\mathbf{R}_k^{-1}\mathbf{H}_k)$

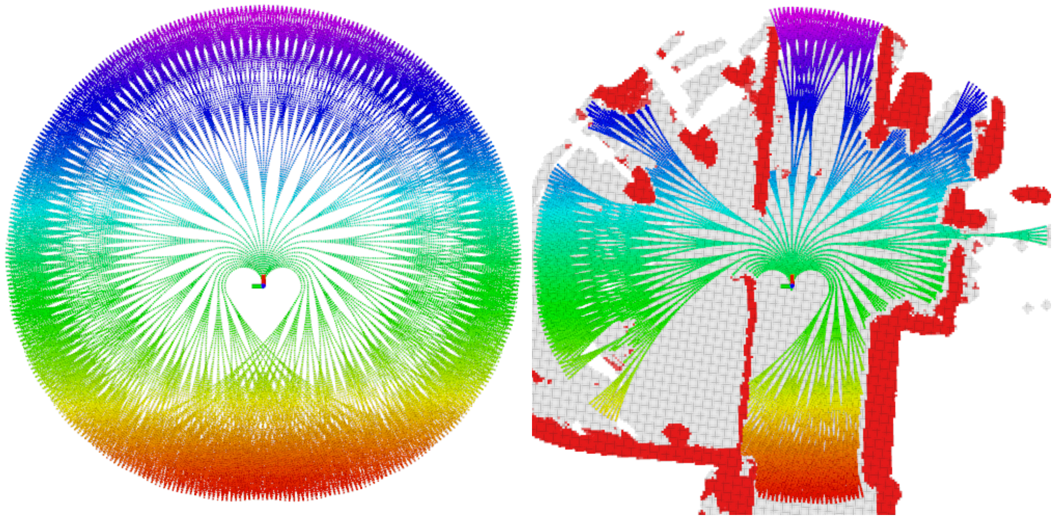

(a) Path library (b) Valid paths

Figure 6.4: Using a pre-computed path library for efficient and aggressive navigation. In (a), a path library generated using the dynamic model of a ground robot. In (b), valid paths after incorporating the obstacle information. Grey areas are traversable. Red areas are non-traversable.

for a cell in the localizability map, we first simulate a laser scan using a model of the lidar and the surrounding map. We then repeatedly translate/rotate the scan by a random transformation within a threshold. At last, we register the transformed scan to the map and obtain the converged registration covariance. This covariance will be treated as $\mathbf{R}$. We repeat this process for every cell in the map. Representative examples of localizability maps are shown in Figure 6.1.

Then we train a network that is described in Chapter 5. The inferred localizability map for Willow Garage is shown in 6.2. The 2D occupancy grid map of Willow garage is shown on the left. The ground truth localizability map that is obtained by calculating $\underline{\lambda}(\mathbf{H}_k'\mathbf{R}_k^{-1}\mathbf{H}_k)$ using the exhaustive approach for each cell is shown in the middle. The predicted localizability map is shown on the right. Note that calculation of the ground truth localizability map takes 27.7 hours on a computer that is equipped with i9-9900k CPU with multi-threading enabled. The inferred localizability map using the proposed approach only takes 1.7 seconds on a computer that is equipped with a Titan RTX GPU. Representative local localizability map comparisons are shown in Figure 6.3.

6.2.2 Aggressive Navigation for UGVs

Another direction for future work is enabling aggressive navigation for UGVs in outdoor environments. Similarly to the approach proposed in [149], we can generate a path library for a UGV based on its dynamic model. Creating such a path library may alleviate the computational burden that is encountered when using sampling-based planning algorithms. When using such a path library for motion planning, we only need to incorporate the new map information and prune the paths that are in collision with obstacles. This approach may be especially suitable for UGVs that are only equipped with limited computational resources or need to traverse aggressively in a cluttered environment.

Bibliography

[1] T. Shan and B. Englot, "Sampling-based minimum risk path planning in multiobjective configuration spaces," in *IEEE Conference on Decision and Control (CDC)*, pp. 814–821, IEEE, 2015.

[2] B. Englot, T. Shan, S. D. Bopardikar, and A. Speranzon, "Sampling-based minmax uncertainty path planning," in *IEEE Conference on Decision and Control (CDC)*, pp. 6863–6870, IEEE, 2016.

[3] T. Shan and B. Englot, "Belief roadmap search: Advances in optimal and efficient planning under uncertainty," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5318–5325, IEEE, 2017.

[4] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4758–4765, IEEE, 2018.

[5] T. Shan, J. Wang, B. Englot, and K. Doherty, "Bayesian generalized kernel inference for terrain traversability mapping," in *Conference on Robot Learning*, pp. 829–838, 2018.

[6] I. M. Mitchell and S. Sastry, "Continuous path planning with multiple constraints," in *IEEE International Conference on Decision and Control (CDC)*, vol. 5, pp. 5502–5507, IEEE, 2003.

[7] O. Castillo, L. Trujillo, and P. Melin, "Multiple objective genetic algorithms for path-planning optimization in autonomous mobile robots," *Soft Computing*, vol. 11, no. 3, pp. 269–279, 2007.

[8] P. Vadakkepat, K. C. Tan, and W. Ming-Liang, "Evolutionary artificial potential fields and their application in real time robot path planning," in *Proceedings of the 2000 Congress on Evolutionary Computation*, vol. 1, pp. 256–263, IEEE, 2000.

[9] P. Hansen, "Bicriterion path problems," in *Multiple criteria decision making theory and application*, pp. 109–127, Springer, 1980.

[10] J. M. Jaffe, "Algorithms for finding paths with multiple constraints," *Networks*, vol. 14, no. 1, pp. 95–116, 1984.

[11] E. Q. V. Martins, "On a multicriteria shortest path problem," *European Journal of Operational Research*, vol. 16, no. 2, pp. 236–245, 1984.

[12] S. M. LaValle and S. A. Hutchinson, "Optimal motion planning for multiple robots having independent goals," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 912–925, 1998.

[13] Z. Clawson, X. Ding, B. Englot, T. A. Frewen, W. M. Sisson, and A. Vladimirsky, "A bi-criteria path planning algorithm for robotics applications," *arXiv preprint arXiv:1511.01166*, 2015.

[14] X. D. Ding, B. Englot, A. Pinto, A. Speranzon, and A. Surana, "Hierarchical multi-objective planning: From mission specifications to contingency management," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3735–3742, IEEE, 2014.

[15] L. Kavraki, P. Svestka, and M. H. Overmars, "Probabilistic roadmaps for path planning in high-dimensional configuration spaces," *IEEE Transactions on Robotics and Automation*, 1994.

[16] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.

[17] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 846–894, 2011.

[18] G. A. Hollinger and G. S. Sukhatme, "Sampling-based robotic information gathering algorithms," *The International Journal of Robotics Research*, vol. 33, no. 9, pp. 1271–1287, 2014.

[19] S. D. Bopardikar, B. Englot, and A. Speranzon, "Multiobjective path planning: Localization constraints and collision probability," *IEEE Transactions on Robotics*, vol. 31, no. 3, pp. 562–577, 2015.

[20] B. D. Luders and J. P. How, "An optimizing sampling-based motion planner with guaranteed robustness to bounded uncertainty," in *American Control Conference*, pp. 771–777, IEEE, 2014.

[21] A. Bry and N. Roy, "Rapidly-exploring random belief trees for motion planning under uncertainty," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 723–730, IEEE, 2011.

[22] J. Kim, R. A. Pearce, and N. M. Amato, "Extracting optimal paths from roadmaps for motion planning," in *IEEE International Conference on Robotics and Automation (ICRA)*, vol. 2, pp. 2424–2429, IEEE, 2003.

[23] L. I. R. Castro, P. Chaudhari, J. Tumova, S. Karaman, E. Frazzoli, and D. Rus, "Incremental sampling-based algorithm for minimum-violation motion plan-

ning," in *IEEE Conference on Decision and Control (CDC)*, pp. 3217–3224, IEEE, 2013.

[24] D. Devaurs, T. Siméon, and J. Cortés, "Efficient sampling-based approaches to optimal path planning in complex cost spaces," in *Algorithmic Foundations of Robotics XI*, pp. 143–159, Springer, 2015.

[25] L. Jaillet, J. Cortés, and T. Siméon, "Sampling-based path planning on configuration-space costmaps," *IEEE Transactions on Robotics*, vol. 26, no. 4, pp. 635–646, 2010.

[26] Y. Gao, S.-d. Sun, and D.-f. He, "Global path planning of mobile robot based on particle filter," in *WRI World Congress on Computer Science and Information Engineering*, vol. 3, pp. 205–209, IEEE, 2009.

[27] A. Soltani and T. Fernando, "A fuzzy based multi-objective path planning of construction sites," *Automation in Construction*, vol. 13, no. 6, pp. 717–734, 2004.

[28] F. Guo, H. Wang, and Y. Tian, "Multi-objective path planning for unrestricted mobile," in *IEEE International Conference on Automation and Logistics*, pp. 1046–1051, IEEE, 2009.

[29] Q. Zhang, I. Rekleitis, and G. Dudek, "Uncertainty reduction via heuristic search planning on hybrid metric/topological map," in *Conference on Computer and Robot Vision*, pp. 222–229, IEEE, 2015.

[30] D. Meger, I. Rekleitis, and G. Dudek, "Heuristic search planning to reduce exploration uncertainty," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3392–3399, IEEE, 2008.

[31] H. Yang, J. Lim, and S.-e. Yoon, "Anytime rrbt for handling uncertainty and dynamic objects," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4786–4793, IEEE, 2016.

[32] B. Luders, M. Kothari, and J. How, "Chance constrained rrt for probabilistic robustness to environmental uncertainty," in *AIAA guidance, navigation, and control conference*, p. 8160, 2010.

[33] B. D. Luders, S. Karaman, and J. P. How, "Robust sampling-based motion planning with asymptotic optimality guarantees," in *AIAA Guidance, Navigation, and Control (GNC) Conference*, p. 5097, 2013.

[34] D. Lenz, M. Rickert, and A. Knoll, "Heuristic search in belief space for motion planning under uncertainties," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2659–2665, IEEE, 2015.

[35] R. Takei, W. Chen, Z. Clawson, S. Kirov, and A. Vladimirsky, "Optimal control with reset-renewable resources," *arXiv preprint arXiv:1110.6221*, 2011.

[36] S. D. Bopardikar, B. Englot, and A. Speranzon, "Multi-objective path planning in gps denied environments under localization constraints," in *American Control Conference*, pp. 1872–1879, IEEE, 2014.

[37] W. Stadler, "Fundamentals of multicriteria optimization," in *Multicriteria Optimization in Engineering and in the Sciences*, pp. 1–25, Springer, 1988.

[38] T. L. Veith, M. L. Wolfe, and C. D. Heatwole, "Optimization procedure for cost effective bmp placement at a watershed scale 1," *JAWRA Journal of the American Water Resources Association*, vol. 39, no. 6, pp. 1331–1343, 2003.

[39] C. Sun, S. G. Ritchie, K. Tsai, and R. Jayakrishnan, "Use of vehicle signature analysis and lexicographic optimization for vehicle reidentification on freeways," *Transportation Research Part C: Emerging Technologies*, vol. 7, no. 4, pp. 167–185, 1999.

[40] A. Engau and M. M. Wiecek, "Generating ε-efficient solutions in multiobjective programming," *European Journal of Operational Research*, vol. 177, no. 3, pp. 1566–1579, 2007.

[41] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," *Structural and Multidisciplinary Optimization*, vol. 26, no. 6, pp. 369–395, 2004.

[42] M. J. Rentmeesters, W. K. Tsai, and K.-J. Lin, "A theory of lexicographic multi-criteria optimization," in *IEEE International Conference on Engineering of Complex Computer Systems*, pp. 76–79, IEEE, 1996.

[43] G. D. Self, "Multicriterion optimization in engineering with fortran programs," 1987.

[44] F. Waltz, "An engineering approach: hierarchical optimization criteria," *IEEE Transactions on Automatic Control*, vol. 12, no. 2, pp. 179–180, 1967.

[45] H. Choset, K. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. Kavraki, and S. Thrun, "Principle of robot motion: Theory, algorithms, and application," 2005.

[46] S. Prentice and N. Roy, "The belief roadmap: Efficient planning in belief space by factoring the covariance," *The International Journal of Robotics Research*, vol. 28, no. 11-12, pp. 1448–1465, 2009.

[47] S. D. Bopardikar, B. Englot, A. Speranzon, and J. Van Den Berg, "Robust belief space planning under intermittent sensing via a maximum eigenvalue-based bound," *The International Journal of Robotics Research*, vol. 35, no. 13, pp. 1609–1626, 2016.

[48] M. Stilman, "Task constrained motion planning in robot joint space," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3074–3081, IEEE, 2007.

[49] K. Shi, J. Denny, and N. M. Amato, "Spark prm: Using rrts within prms to efficiently explore narrow passages," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4659–4666, IEEE, 2014.

[50] S. A. Wilmarth, N. M. Amato, and P. F. Stiller, "Maprm: A probabilistic roadmap planner with sampling on the medial axis of the free space," in *ICRA*, pp. 1024–1031, 1999.

[51] J. Denny, E. Greco, S. Thomas, and N. M. Amato, "Marrt: Medial axis biased rapidly-exploring random trees," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 90–97, IEEE, 2014.

[52] S. Rodriguez, X. Tang, J.-M. Lien, and N. M. Amato, "An obstacle-based rapidly-exploring random tree," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 895–900, IEEE, 2006.

[53] D. Devaurs, T. Siméon, and J. Cortés, "Enhancing the transition-based rrt to deal with complex cost spaces," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4120–4125, IEEE, 2013.

[54] A.-A. Agha-Mohammadi, S. Chakravorty, and N. M. Amato, “Firm: Sampling-based feedback motion-planning under motion uncertainty and imperfect measurements,” *The International Journal of Robotics Research*, vol. 33, no. 2, pp. 268–304, 2014.

[55] R. Pepy, M. Kieffer, and E. Walter, “Reliable robust path planner,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1655–1660, IEEE, 2008.

[56] J. Van Den Berg, P. Abbeel, and K. Goldberg, “Lqg-mp: Optimized path planning for robots with motion uncertainty and imperfect state information,” *The International Journal of Robotics Research*, vol. 30, no. 7, pp. 895–913, 2011.

[57] W. Burgard, O. Brock, and C. Stachniss, *The Stochastic Motion Roadmap: A Sampling Framework for Planning with Markov Motion Uncertainty.* MIT Press, 2008.

[58] V. A. Huynh, S. Karaman, and E. Frazzoli, “An incremental sampling-based algorithm for stochastic optimal control,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2865–2872, IEEE, 2012.

[59] H. Kurniawati and V. Yadav, “An online pomdp solver for uncertainty planning in dynamic environment,” in *Robotics Research*, pp. 611–629, Springer, 2016.

[60] P. J. Besl and N. D. McKay, “Method for registration of 3-d shapes,” in *Sensor fusion IV: Control Paradigms and Data Structures*, vol. 1611, pp. 586–606, International Society for Optics and Photonics, 1992.

[61] S. Rusinkiewicz and M. Levoy, "Efficient variants of the icp algorithm," in *3dim*, vol. 1, pp. 145–152, 2001.

[62] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image and Vision Computing*, vol. 10, no. 3, pp. 145–155, 1992.

[63] A. Segal, D. Haehnel, and S. Thrun, "Generalized-icp," vol. 2, no. 4, p. 435, 2009.

[64] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. W. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," vol. 11, no. 2011, pp. 127–136, 2011.

[65] A. Nüchter, "Parallelization of scan matching for robotic 3d mapping," in *European Conference on Mobile Robots*, 2007.

[66] D. Qiu, S. May, and A. Nüchter, "Gpu-accelerated nearest neighbor search for 3d registration," in *International Conference on Computer Vision Systems*, pp. 194–203, Springer, 2009.

[67] D. Neumann, F. Lugauer, S. Bauer, J. Wasza, and J. Hornegger, "Real-time rgb-d mapping and 3-d modeling on the gpu using the random ball cover data structure," in *IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pp. 1161–1167, IEEE, 2011.

[68] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz, "Learning informative point classes for the acquisition of object model maps," in *International Conference on Control, Automation, Robotics and Vision*, pp. 643–650, IEEE, 2008.

[69] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu, "Fast 3d recognition and pose using the viewpoint feature histogram," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2155–2162, IEEE, 2010.

[70] Y. Li and E. B. Olson, "Structure tensors for general purpose lidar feature extraction," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1869–1874, IEEE, 2011.

[71] J. Serafin, E. Olson, and G. Grisetti, "Fast and robust 3d feature extraction from sparse point clouds," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4105–4112, IEEE, 2016.

[72] M. Velas, M. Spanel, and A. Herout, "Collar line segments for fast odometry estimation from velodyne point clouds," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4486–4495, IEEE, 2016.

[73] R. Dubé, D. Dugas, E. Stumm, J. Nieto, R. Siegwart, and C. Cadena, "Segmatch: Segment based place recognition in 3d point clouds," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5266–5272, IEEE, 2017.

[74] M. Bosse and R. Zlot, "Keypoint design and evaluation for place recognition in 2d lidar maps," *Robotics and Autonomous Systems*, vol. 57, no. 12, pp. 1211–1224, 2009.

[75] R. Zlot and M. Bosse, "Efficient large-scale 3d mobile mapping and surface reconstruction of an underground mine," in *Field and Service Robotics*, pp. 479–493, Springer, 2014.

[76] B. Steder, G. Grisetti, and W. Burgard, “Robust place recognition for 3d range data based on point features,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 1400–1405, IEEE, 2010.

[77] W. S. Grant, R. C. Voorhies, and L. Itti, “Finding planes in lidar point clouds for real-time registration,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4347–4354, IEEE, 2013.

[78] J. Zhang and S. Singh, “Loam: Lidar odometry and mapping in real-time,” in *Robotics: Science and Systems*, vol. 2, p. 9, 2014.

[79] J. Zhang and S. Singh, “Low-drift and real-time lidar odometry and mapping,” *Autonomous Robots*, vol. 41, no. 2, pp. 401–416, 2017.

[80] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? the kitti vision benchmark suite,” in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3354–3361, IEEE, 2012.

[81] J. M. Santos, D. Portugal, and R. P. Rocha, “An evaluation of 2d slam techniques available in robot operating system,” in *IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*, pp. 1–6, IEEE, 2013.

[82] M. Whitty, S. Cossell, K. S. Dang, J. Guivant, and J. Katupitiya, “Autonomous navigation using a real-time 3d point cloud,” in *Australasian Conference on Robotics and Automation*, pp. 1–3, 2010.

[83] A. Rönnau, G. Liebel, T. Schamm, T. Kerscher, and R. Dillmann, “Robust 3d scan segmentation for teleoperation tasks in areas contaminated by radiation,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2419–2424, IEEE, 2010.

[84] Y. Tanaka, Y. Ji, A. Yamashita, and H. Asama, “Fuzzy based traversability analysis for a mobile robot on rough terrain,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3965–3970, IEEE, 2015.

[85] A. Chilian and H. Hirschmüller, “Stereo camera based navigation of mobile robots on rough terrain,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4571–4576, IEEE, 2009.

[86] D. Gingras, T. Lamarche, J.-L. Bedwani, and É. Dupuis, “Rough terrain reconstruction for rover motion planning,” in *Canadian Conference on Computer and Robot Vision*, pp. 191–198, IEEE, 2010.

[87] I. Bogoslavskyi, O. Vysotska, J. Serafin, G. Grisetti, and C. Stachniss, “Efficient traversability analysis for mobile robots using the kinect sensor,” in *European Conference on Mobile Robots*, pp. 158–163, IEEE, 2013.

[88] S. Kuthirummal, A. Das, and S. Samarasekera, “A graph traversal based algorithm for obstacle detection using lidar or stereo,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 3874–3880, IEEE, 2011.

[89] D. Wooden, M. Malchano, K. Blankespoor, A. Howardy, A. A. Rizzi, and M. Raibert, “Autonomous navigation for bigdog,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4736–4741, IEEE, 2010.

[90] 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 (ICRA)*, pp. 5539–5544, IEEE, 2013.

[91] J. Sock, J. Kim, J. Min, and K. Kwak, "Probabilistic traversability map generation using 3d-lidar and camera," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5631–5637, IEEE, 2016.

[92] S. Zhou, J. Xi, M. W. McDaniel, T. Nishihata, P. Salesses, and K. Iagnemma, "Self-supervised learning to visually detect terrain surfaces for autonomous robots operating in forested terrain," *Journal of Field Robotics*, vol. 29, no. 2, pp. 277–297, 2012.

[93] K. Zimmermann, P. Zuzanek, M. Reinstein, and V. Hlavac, "Adaptive traversability of unknown complex terrain with obstacles for mobile robots," in *IEEE international conference on robotics and automation (ICRA)*, pp. 5177–5182, IEEE, 2014.

[94] À. Santamaria-Navarro, E. H. Teniente, M. Morta, and J. Andrade-Cetto, "Terrain classification in complex three-dimensional outdoor environments," *Journal of Field Robotics*, vol. 32, no. 1, pp. 42–60, 2015.

[95] B. Suger, B. Steder, and W. Burgard, "Traversability analysis for mobile robots in outdoor environments: A semi-supervised learning approach based on 3d-lidar data," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3941–3946, IEEE, 2015.

[96] J. Ahtiainen, T. Stoyanov, and J. Saarinen, "Normal distributions transform traversability maps: Lidar-only approach for traversability mapping in outdoor environments," *Journal of Field Robotics*, vol. 34, no. 3, pp. 600–621, 2017.

[97] P. Krüsi, P. Furgale, M. Bosse, and R. Siegwart, "Driving on point clouds: Motion planning, trajectory optimization, and terrain assessment in generic

nonplanar environments," *Journal of Field Robotics*, vol. 34, no. 5, pp. 940–984, 2017.

[98] M. Liu and R. Siegwart, "Navigation on point cloud - a riemannian metric approach," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4088–4093, IEEE, 2014.

[99] F. Colas, S. Mahesh, F. Pomerleau, M. Liu, and R. Siegwart, "3d path planning and execution for search and rescue ground robots," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 722–727, IEEE, 2013.

[100] S. Vasudevan, F. Ramos, E. Nettleton, and H. Durrant-Whyte, "Gaussian process modeling of large-scale terrain," *Journal of Field Robotics*, vol. 26, no. 10, pp. 812–840, 2009.

[101] V. Guizilini and F. Ramos, "Variational hilbert regression with applications to terrain modeling," in *International Symposium on Robotics Research (ISRR)*, 2017.

[102] W. Yang, X. Zhang, Y. Tian, W. Wang, J.-H. Xue, and Q. Liao, "Deep learning for single image super-resolution: A brief review," *IEEE Transactions on Multimedia*, 2019.

[103] R. Keys, "Cubic convolution interpolation for digital image processing," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 29, no. 6, pp. 1153–1160, 1981.

[104] C. E. Duchon, "Lanczos filtering in one and two dimensions," *Journal of Applied Meteorology*, vol. 18, no. 8, pp. 1016–1022, 1979.

[105] C. Dong, C. C. Loy, K. He, and X. Tang, “Learning a deep convolutional network for image super-resolution,” in *European Conference on Computer Vision*, pp. 184–199, Springer, 2014.

[106] J.-B. Huang, A. Singh, and N. Ahuja, “Single image super-resolution from transformed self-exemplars,” in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5197–5206, 2015.

[107] J. Kim, J. Kwon Lee, and K. Mu Lee, “Deeply-recursive convolutional network for image super-resolution,” in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1637–1645, 2016.

[108] W. Shi, J. Caballero, F. Huszár, J. Totz, A. P. Aitken, R. Bishop, D. Rueckert, and Z. Wang, “Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network,” in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1874–1883, 2016.

[109] C. Ledig, L. Theis, F. Huszár, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, *et al.*, “Photo-realistic single image super-resolution using a generative adversarial network,” in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 4681–4690, 2017.

[110] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, “Generative adversarial nets,” in *Advances in Neural Information Processing Systems*, pp. 2672–2680, 2014.

[111] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

[112] Y. Zhang and T. Funkhouser, “Deep depth completion of a single rgb-d image,” in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 175–185, 2018.

[113] F. Ma, L. Carlone, U. Ayaz, and S. Karaman, “Sparse sensing for resource-constrained depth reconstruction,” in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 96–103, IEEE, 2016.

[114] J. Ku, A. Harakeh, and S. L. Waslander, “In defense of classical image processing: Fast depth completion on the cpu,” in *Conference on Computer and Robot Vision (CRV)*, pp. 16–22, IEEE, 2018.

[115] F. Ma, G. V. Cavalheiro, and S. Karaman, “Self-supervised sparse-to-dense: self-supervised depth completion from lidar and monocular camera,” *arXiv preprint arXiv:1807.00275*, 2018.

[116] V. Casser, S. Pirk, R. Mahjourian, and A. Angelova, “Depth prediction without the sensors: Leveraging structure for unsupervised learning from monocular videos,” *arXiv preprint arXiv:1811.06152*, 2018.

[117] C. Godard, O. Mac Aodha, and G. J. Brostow, “Unsupervised monocular depth estimation with left-right consistency,” in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 270–279, 2017.

[118] I. Laina, C. Rupprecht, V. Belagiannis, F. Tombari, and N. Navab, “Deeper depth prediction with fully convolutional residual networks,” in *International Conference on 3D vision (3DV)*, pp. 239–248, IEEE, 2016.

[119] N. Smolyanskiy, A. Kamenev, and S. Birchfield, “On the importance of stereo for accurate depth estimation: An efficient semi-supervised deep neural network

approach," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pp. 1007–1015, 2018.

[120] L. Yu, X. Li, C.-W. Fu, D. Cohen-Or, and P.-A. Heng, "Pu-net: Point cloud upsampling network," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2790–2799, 2018.

[121] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems*, pp. 5099–5108, 2017.

[122] A. Gelb, *Applied optimal estimation.* MIT press, 1974.

[123] Y. Okamoto and H. J. Maris, "A novel algorithm for calculation of the extreme eigenvalues of large hermitian matrices," *Computer Physics Communications*, vol. 76, no. 2, pp. 191–202, 1993.

[124] D. J. Webb and J. Van Den Berg, "Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics," in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5054–5061, IEEE, 2013.

[125] D. Fox, "Adapting the sample size in particle filters through kld-sampling," *The international Journal of robotics research*, vol. 22, no. 12, pp. 985–1003, 2003.

[126] M. Himmelsbach, F. V. Hundelshausen, and H.-J. Wuensche, "Fast segmentation of 3d point clouds for ground vehicles," in *IEEE Intelligent Vehicles Symposium*, pp. 560–565, IEEE, 2010.

[127] I. Bogoslavskyi and C. Stachniss, "Fast range image-based segmentation of sparse 3d laser scans for online operation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 163–169, IEEE, 2016.

[128] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. J. Leonard, and F. Dellaert, "isam2: Incremental smoothing and mapping using the bayes tree," *The International Journal of Robotics Research*, vol. 31, no. 2, pp. 216–235, 2012.

[129] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," vol. 3, no. 3.2, p. 5, 2009.

[130] W. R. Vega-Brown, M. Doniec, and N. G. Roy, "Nonparametric bayesian inference on multivariate exponential families," in *Advances in Neural Information Processing Systems*, pp. 2546–2554, 2014.

[131] A. Melkumyan and F. T. Ramos, "A sparse covariance function for exact gaussian process inference in large datasets," in *International Joint Conference on Artificial Intelligence*, 2009.

[132] K. Doherty, T. Shan, J. Wang, and B. Englot, "Learning-aided 3-d occupancy mapping with bayesian generalized kernel inference," *IEEE Transactions on Robotics*, 2019.

[133] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, pp. 2149–2154, IEEE, 2004.

[134] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[135] Y. Gal and Z. Ghahramani, “Dropout as a bayesian approximation: Representing model uncertainty in deep learning,” in *International Conference on Machine Learning*, pp. 1050–1059, 2016.

[136] R. M. Neal, *Bayesian learning for neural networks*, vol. 118. Springer Science & Business Media, 2012.

[137] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, “Driving in the matrix: Can virtual worlds replace human-generated annotations for real world tasks,” in *IEEE International Conference on Robotics and Automation (ICRA)*, pp. 746–753, 2017.

[138] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, “Carla: An open urban driving simulator,” *Conference on Robot Learning*, 2017.

[139] O. Ronneberger, P. Fischer, and T. Brox, “U-net: Convolutional networks for biomedical image segmentation,” in *International Conference on Medical Image Computing and Computer-assisted Intervention*, pp. 234–241, Springer, 2015.

[140] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” *arXiv preprint arXiv:1502.03167*, 2015.

[141] V. Nair and G. E. Hinton, “Rectified linear units improve restricted boltzmann machines,” in *International Conference on Machine Learning*, pp. 807–814, 2010.

[142] Y. Wang, W.-L. Chao, D. Garg, B. Hariharan, M. Campbell, and K. Q. Weinberger, “Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving,” in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 8445–8453, 2019.

[143] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013.

[144] F. Chen, S. Bai, T. Shan, and B. Englot, "Self-learning exploration and mapping for mobile robots via deep reinforcement learning," in *AIAA Scitech 2019 Forum*, p. 0396, 2019.

[145] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 2014.

[146] F. Chollet *et al.*, "Keras: The python deep learning library," *Astrophysics Source Code Library*, 2018.

[147] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, "Tensorflow: A system for large-scale machine learning," in *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pp. 265–283, 2016.

[148] R. Schirmer, P. Biber, and C. Stachniss, "Efficient path planning in belief space for safe navigation," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2857–2863, IEEE, 2017.

[149] J. Zhang, R. G. Chadha, V. Velivela, and S. Singh, "P-cap: Pre-computed alternative paths to enable aggressive aerial maneuvers in cluttered environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 8456–8463, IEEE, 2018.

Vita

Tixiao Shan

Education

- **Stevens Institute of Technology**, Ph.D., Mechanical Engineering, 2014-2019
- **Shanghai University**, M.S., Mechanical Electronics Engineering, 2011-2014
- **Qingdao University**, B.S., Mechanical and Electrical Engineering, 2007-2011

Research Experience

- **Research Assistant**, Stevens Institute of Technology, 2014-2019
- **Assistant Engineer Intern**, ABB Engineering Ltd., 2013-2014
- **Research Assistant**, Shanghai University, 2012-2014

Teaching Experience

- **Intro to Robotics**, Stevens Institute of Technology, 2017-2019
- **Control Systems**, Stevens Institute of Technology, 2017-2019
- **Engineering Design**, Stevens Institute of Technology, 2018-2019
- **Systems Laboratory**, Stevens Institute of Technology, 2016
- **CAD and CAM**, Shanghai University, 2012

Publications

- F. Chen, J. Wang, T. Shan, and B. Englot, "Autonomous Exploration Under Uncertainty via Graph Convolutional Networks," *International Symposium on Robotics Research*, 2019.

- J. Wang, T. Shan, and B. Englot, "Virtual Maps for Autonomous Exploration with Pose SLAM," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2019.
- K. Doherty, T. Shan, J. Wang, and B. Englot, "Learning-aided 3D Occupancy Mapping with Bayesian Generalized Kernel Inference," *IEEE Transactions on Robotics (T-RO)*, 2019
- J. Wang, T. Shan, M. Chandrasekaran, T. Osedach, and B. Englot, "Deep Learning for Detection and Tracking of Underwater Pipelines using Multibeam Imaging Sonar," *IEEE International Conference on Robotics and Automation (ICRA) Workshop*, 2019
- J. Wang, T. Shan, and B. Englot, "Underwater Terrain Reconstruction from Forward-Looking Sonar Imagery," *IEEE International Conference on Robotics and Automation (ICRA)*, 2019
- F. Chen, S. Bai, T. Shan and B. Englot. "Self-Learning Exploration and Mapping for Mobile Robots via Deep Reinforcement Learning." *International AIAA Information-Driven Decision and Control Conference (AIAA)*, 2019
- T. Shan, K. Doherty, J. Wang and B. Englot. "Bayesian Generalized Kernel Inference for Terrain Traversability Mapping." *Conference on Robot Learning (CoRL)*, 2018
- T. Shan and B. Englot. "LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain." *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4758-4765, 2018
- T. Shan and B. Englot. "Belief Roadmap Search: Advances in Optimal and Efficient Planning Under Uncertainty." *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 5318-5325, 2017
- B. Englot, T. Shan, S.D. Bopardikar, and A. Speranzon. "Sampling-based Min-

Max Uncertainty Path Planning." *IEEE International Conference on Decision and Control (CDC)*, pp. 6863-6870, 2016

- T. Shan and B. Englot. "Sampling-based Minimum Risk Path Planning in Multiobjective Configuration Spaces." *IEEE International Conference on Decision and Control (CDC)*, pp. 814-821, 2015
- T. Shan and B. Englot, "Tunable-Risk Sampling-Based Path Planning Using a Cost Hierarchy," *IEEE International Conference on Robotics and Automation (ICRA) Workshop*, pp. 2, 2015

Awards

- Fernando L. Fernandez Robotics and Automation Fellowship, 2019
- Stevens Innovation and Entrepreneurship Fellowship, 2014-2017
- China National Scholarship for Graduate Students, 2013
- Shanghai University Outstanding Students, 2013
- Qingdao University Scholarship, 2007-2010
- Qingdao University Outstanding Students, 2008-2010
- Tsingtao Brewery Education Scholarship, 2009