VT&R3: Generalizing the Teach and Repeat Navigation Framework

by

Yuchen Wu

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science

Department of Institute for Aerospace Studies
University of Toronto

VT&R3: Generalizing the Teach and Repeat Navigation Framework

Yuchen Wu
Master of Applied Science

Department of Institute for Aerospace Studies
University of Toronto
2022

Abstract

Visual Teach & Repeat (VT&R) achieves long-range and long-term autonomous path following through visual topometric mapping and localization. The system requires only a stereo camera for sensing, making it well-suited for many mobile robot applications involving GPS-denied environments. In this thesis, we present VT&R3, a new version of VT&R that generalizes its underlying teach and repeat navigation framework to work with lidar, radar, and possibly any rich sensor. We develop lidar and radar topometric mapping and localization pipelines that enable lidar/radar-based teach and repeat. We further extend the lidar pipeline with the ability to detect obstacles in changing environments, making obstacle avoidance during path following possible. We validate our pipelines through extensive evaluation using real-world datasets, including runtime analysis showing their online operation capability. Our results demonstrate surprising accuracy and robustness of lidar localization across varying weather and seasonal conditions, while radar localization remains competitive and has advantages in computation and storage requirements.

Acknowledgements

First and foremost, my deepest gratitude goes to my supervisor, Professor Tim Barfoot, for helping me through an important transition in my life and for the expert guidance in completing this thesis.

I thank all the members of the Autonomous Space Robotics Laboratory for building a friendly and collaborative environment. Special thanks to Ben and Daniel for early collaboration on the VT&R3 project, to Mona for conducting over a hundred kilometers of VT&R3 field testing, to Hugues for sharing your work on lidar odometry & mapping, and to Keenan and David for working together on lidar/radar topometric mapping and localization.

I acknowledge all the VT&R2 developers for creating such an impressive navigation system. All the works in this thesis are based on your accomplishments, for which I am very grateful.

Finally, thank you to my parents for the unconditional support and love.

Contents

1 Introduction **1**

1.1 Motivation . . . 1

1.2 Contributions . . . 2

2 Visual Teach & Repeat 3 **3**

2.1 History . . . 3

2.1.1 The Predecessor of VT&R . . . 3

2.1.2 Visual Teach & Repeat 1 . . . 4

2.1.3 Visual Teach & Repeat 2 . . . 4

2.1.4 Moving Forward . . . 8

2.2 The Main Idea: Multi-Experience Topometric Map . . . 9

2.2.1 Relative Pose Graph as the Topometric Map . . . 9

2.2.2 The Multi-Experience Extension . . . 10

2.2.3 Privileged vs Non-Privileged Submap . . . 10

2.3 System Overview . . . 12

2.4 State Estimation . . . 13

2.4.1 State Estimation Interfaces . . . 14

2.4.2 Pipeline and Parallelization . . . 16

2.5 Global Planning . . . 18

2.6 Local Planning . . . 19

2.7 Task Planning . . . 19

2.8 The Graphical User Interface . . . 21

2.8.1 The Network of Paths . . . 21

2.8.2 The Robot Marker . . . 21

2.8.3 The Teach & Repeat Pass Manager . . . 22

2.9 Summary . . . 23

3 Lidar & Radar Topometric Mapping and Localization **24**

3.1 Introduction . . . 24

3.2 Related Work . . . 25

3.3 Methodology . . . 26

3.3.1 Pipeline Overview . . . 26

3.3.2 Keypoint Extraction . . . 27

3.3.3 Continuous-Time Iterative Closest Point . . . 28

3.3.4 Doppler-Compensated CT-ICP . . . 29
3.3.5 Localization ICP . . . 30
3.4 Evaluation . . . 30
3.4.1 Dataset . . . 30
3.4.2 Metrics . . . 32
3.4.3 Results . . . 33
3.5 Summary . . . 36

4 Lidar Map Maintenance and Obstacle Detection 38
4.1 Introduction . . . 38
4.2 Related Work . . . 39
4.2.1 Change Detection . . . 39
4.2.2 Map Maintenance . . . 40
4.2.3 Obstacle Detection . . . 42
4.3 Methodology . . . 42
4.3.1 Assumptions . . . 42
4.3.2 Pipeline Overview . . . 43
4.3.3 Dynamic Points Removal . . . 45
4.3.4 Multi-Experience Mapping . . . 46
4.3.5 Obstacle Detection . . . 48
4.3.6 Safe Corridor Specification and Visualization . . . 48
4.3.7 Local Planning Integration . . . 49
4.4 Evaluation . . . 51
4.4.1 Dataset . . . 51
4.4.2 Evaluation Strategy and Metrics . . . 52
4.4.3 Results . . . 54
4.5 Summary . . . 60

5 Conclusion and Future Work 62
5.1 Conclusion . . . 62
5.2 Future Work . . . 63
5.2.1 Visual Teach & Repeat 3 . . . 63
5.2.2 Lidar & Radar Topometric Mapping and Localization . . . 63
5.2.3 Lidar Map Maintenance and Obstacle Detection . . . 64

Bibliography 65

List of Tables

3.1 Metric localization RMSE results on sequence: 2020-11-26 34
3.2 Topometric localization and mapping computation and storage requirements 36

4.1 Obstacle detection results: precision, recall and f-score 58
4.2 Map maintenance and obstacle detection computation requirements 58

List of Figures

2.1 VT&R2 field testing result in Sudbury (Ontario, Canada), 2016 7
2.2 VT&R3 topometric map structure 9
2.3 VT&R3 multi-experience pose graph structure 10
2.4 VT&R3 topometric map construction process 11
2.5 VT&R3 system overview 12
2.6 VT&R3 odometry & mapping pose graph 14
2.7 VT&R3 inter-experience localization pose graph 15
2.8 VT&R3 state estimation pipeline and parallelization diagram 16
2.9 VT&R3 global planning illustration 18
2.10 VT&R3 local planning illustration 19
2.11 VT&R3 HFSM diagram for task planning 20
2.12 VT&R3 GUI overview 21
2.13 VT&R3 GUI states during a teach pass 22
2.14 VT&R3 GUI states during a repeat pass 23

3.1 Lidar/radar topometric mapping and localization pipeline 27
3.2 CT-ICP factor graph 28
3.3 The *Boreas* data collection platform 31
3.4 The 8km Glen Shields route where the *Boreas* dataset is collected 31
3.5 Lidar/radar teach & repeat reference and test sequences 32
3.6 Lidar/radar teach & repeat localization error histograms 33
3.7 Illustration of the noisy lidar data for localization during the 2021-01-26 sequence . . 34
3.8 Lidar/radar localization error during the snowstorm sequence 35
3.9 Aligned point clouds from radar-to-radar and radar-to-lidar localization 35

4.1 Lidar map maintenance and obstacle detection pipeline 44
4.2 2D ray-tracing example 46
4.3 Updated VT&R3 GUI for safe corridor specification and visualization 49
4.4 Obstacle and safe corridor aware local planning demonstration 50
4.5 Clearpath Grizzly UGV for data collection 51
4.6 Dataset routes and environment characteristics 52
4.7 An example of synthesizing fake obstacles into a real lidar scan 53
4.8 Qualitative result of dynamic points removal 54
4.9 Qualitative result of multi-experience mapping 55

4.10 The final long-term static map of the three environments used for evaluation 56
4.11 Ground slope and roughness histogram of the three environments for evaluation . . . 57
4.12 Precision-recall curve of the proposed obstacle detection method and its variants . . 57
4.13 Qualitative result of obstacle detection using fake obstacles 59
4.14 Qualitative result of online real obstacle detection 60

Acronyms

BFAR	Bounded False Alarm Rate
BFS	Breadth-First Search
CT-ICP	Continuous-Time Iterative Closest Point
CFAR	Constant False Alarm Rate
FOV	Field of View
GNSS	Global Navigation Satellite System
GP	Gaussian Process
GPS	Global Positioning System
GUI	Graphical User Interface
HFSM	Hierarchical Finite State Machine
ICP	Iterative Closest Point
IMU	Inertial Measurement Unit
lidar	Light Detection and Ranging
MCL	Multi-Channel Localization
MEL	Multi-Experience Localization
MPC	Model Predictive Control
NDT	Normal Distributions Transform
PCA	Principle Component Analysis
radar	Radio Detection and Ranging
RMSE	Root Mean Squared Error
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
STPG	Spatial-Temporal Pose Graph
SVD	Singular Value Decomposition
SURF	Speeded Up Robust Features
TEB	Time Elastic Band
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
UKF	Unscented Kalman Filter
UTIAS	University of Toronto Institute for Aerospace Studies
VT&R	Visual Teach & Repeat

Notation

a	This font is used for quantities that are real scalars.
$\mathbf{a}$	This font is used for quantities that are real column vectors.
$\mathbf{A}$	This font is used for quantities that are real matrices.
$\underrightarrow{\boldsymbol{\mathcal{F}}}_a$	A vectrix representing a reference frame in three dimensions.
$SE(3)$	The special Euclidean group, a matrix Lie group used to represent poses.
$\mathfrak{se}(3)$	The Lie algebra associated with $SE(3)$.
$\mathbf{T}_{ba}$	A matrix in $SE(3)$ that transforms vectors from frame $\underrightarrow{\boldsymbol{\mathcal{F}}}_a$ to frame $\underrightarrow{\boldsymbol{\mathcal{F}}}_b$.
$\exp(\cdot^\wedge)$	A Lie algebra operator mapping from $\mathfrak{se}(3)$ to $SE(3)$
$\ln(\cdot)^\vee$	A Lie algebra operator mapping from $SE(3)$ to $\mathfrak{se}(3)$

Chapter 1

Introduction

1.1 Motivation

Autonomous navigation based on the teach and repeat framework has shown to be a viable solution for many mobile robot applications in transportation, mining, and planetary exploration. The idea is to constrain the robot to navigate along a set of human-taught paths, simplifying general navigation to path following and thereby avoiding challenging terrain assessment and path planning problems in unstructured environments. A particularly successful implementation of this framework is the Visual Teach & Repeat (VT&R) system developed at the Autonomous Space Robotics Laboratory (ASRL) [1]. The system connects all taught paths into a large (kilometer-scale) network where the robot navigates freely via accurate (centimeter-level) path following. It handles unstructured and changing environments with a demonstrated operational window of multiple months in natural outdoor scenarios. Most impressive is that the system can achieve such long-range and long-term operational capability using a single stereo camera for sensing.

In this thesis, we continue to exploit the potential of teach and repeat navigation through an architectural re-design of VT&R focusing on generalizability. Today's mobile robots are equipped with various sensors (camera, lidar, radar, IMU, etc.). Relying on a single stereo camera is an impressive feature of VT&R but also prevents the system from taking advantage of other sensor modalities. Switching to a new sensor or sensor suite requires different algorithms to solve problems such as mapping and localization, but the navigation framework inherently generalizes to any sensor/robot combination. The primary goal of this thesis is to design a new teach and repeat system on top of VT&R that reflects the generalizability of its underlying framework, enabling easy adaptation to a wide variety of sensor/robot combinations.

A major benefit of having such a system is that it accelerates the development and evaluation of new solutions to problems related to teach and repeat navigation. Previously, localization issues arose in the long-term operation of VT&R, leading to a large amount of work improving the robustness of visual localization under significant appearance changes [2, 3, 4]. Safety concerns in collaborative environments motivated research on terrain assessment preventing the robot from running into humans during autonomous path following [5, 6]. The secondary goal of this thesis is to look into similar problems while using a lidar or radar as the primary sensor instead of a stereo camera. Our work towards this goal demonstrates the generalizability improvement of the re-designed system

and, more importantly, gets VT&R ready for new robots and use cases.

1.2 Contributions

The contributions of this thesis are:

Visual Teach & Repeat 3 (VT&R3): our newest version of VT&R with a complete software upgrade and architectural re-design towards generalizability. VT&R3 inherits the key feature of its predecessor, VT&R2 [1], in using a single stereo camera for teach and repeat navigation. It additionally provides standardized interfaces for state estimation and global/local planning that allow easily adding new algorithms for use with alternative sensor/robot combinations. VT&R3 has already been used in multiple publications [7, 8, 9] and is open-source at `https://github.com/utiasASRL/vtr3`.

Radar & Lidar Topometric Mapping and Localization: a topometric mapping and localization pipeline using data from a lidar or radar, using VT&R3's state estimation interface. The pipeline is capable of performing radar-only, lidar-only, or cross-modal radar-to-lidar localization. The performance of all three variants is evaluated and compared using our public available Boreas dataset [10], resulting in the following paper:

> Keenan Burnett, Yuchen Wu, David J Yoon, Angela P Schoellig, and Timothy D Barfoot. "Are We Ready for Radar to Replace Lidar in All-Weather Mapping and Localization?" *arXiv:2203.10174*, 2022.

Lidar Map Maintenance and Obstacle Detection: an extension to the lidar-only topometric mapping and localization pipeline with 1) a map maintenance module that produces clean and up-to-date lidar maps in changing environments and 2) an obstacle detection module that labels points from live lidar scans as belonging to obstacles or not. The proposed methods for both map maintenance and obstacle detection are validated using real-world datasets. The extended pipeline will be used by VT&R3 to enable obstacle avoidance during path following when using a lidar as the primary sensor.

The details of each contribution will be presented in Chapter 2, 3 and 4, respectively. A conclusion of the presented work and discussion on future directions will be given in Chapter 5.

Chapter 2

Visual Teach & Repeat 3

In this chapter, we present the main contribution of this thesis: Visual Teach & Repeat 3, a mobile robot navigation system implementing the teach and repeat navigation framework, designed to be used by various sensor/robot combinations. We start this chapter with a review of the VT&R development history that provides the relevant background information. We then explain the core idea of VT&R3 in representing the environment with a *multi-experience topometric map*. An overview the VT&R3 system is presented afterward, followed by more detailed descriptions of its main components.

2.1 History

2.1.1 The Predecessor of VT&R

Early autonomous navigation systems related to the teach and repeat framework can be found in the work of Matsumoto et al. [11] and Howard [12]. In [11], the teach pass is called a *recording run*, during which the robot records a sequence of camera images and the relational information suggesting how to move from one image to the next. The robot can then repeat the path by matching against these images and leveraging the relational information. Howard [12] introduced *manifold map*, which can be seen as a form of the *topometric map* later used by VT&R. The advantages of using such a representation for incremental mapping (i.e., path teaching) and retro-traversing (i.e., path repeating) were discussed and demonstrated.

The VT&R system originates from the work by Marshall et al. [13, 14] in developing an automated system for the "load-haul-dump" (LHD) cycle of under tramming vehicles. The system operates in three steps: *teaching*, *route profiling*, and *playback*. In the *teaching* step, the vehicle is manually driven on the desired route and logs data from onboard sensors (front and rear laser range finders, a hinge angle encoder, and a drive shaft encoder). Then, *route profiling* is performed offline using the logged data to generate a path profile, a speed profile, a pause profile, and a sequence of local maps along the path represented by overlapping occupancy grids. After that, the system can play the route profile back autonomously. An UKF-based localizer and a feedback linearization controller work together to compute and reject lateral and heading errors, allowing the robot to follow the profiled path accurately at the desired speed.

The system was deployed on multiple LHD vehicles and tested extensively in real underground

mine environments. It demonstrated superior performance along straight but narrow passages compared to human operators. Path tracking error was reported to be less than 50cm and 0.1rad in all test sequences.

2.1.2 Visual Teach & Repeat 1

The encouraging results from [14] led to the development of VT&R1 presented by Furgale and Barfoot [15] in 2010, which is a complete navigation system capable of long-range path following in nonplanar terrain using a stereo camera as the only sensor. VT&R1 works similarly to [14]: online data collection along the desired path followed by offline map generation, after which the path can be repeated autonomously. The main difference lies in the algorithm employed for mapping and localization in order to work with a stereo camera and nonplanar terrain, where VT&R1 uses a SURF variant of the stereo visual odometry algorithm. The local maps generated by VT&R1 consist of triangulated SURF features as 3D landmarks instead of occupancy grids, allowing localization to be performed in 3D. Local maps are topologically connected to form a hybrid topological/metric representation (a.k.a. a *topometric* map) of the environment. The same controller in [14] is used, which operates on the approximated ground plane of each local map. Furgale and Barfoot [15] envisioned VT&R1 being used for planetary exploration tasks and thereby tested the system at a Mars analog site on Devon Island in the Canadian High Arctic. Over 32 km of testing was reported, with 99.6% of the distance traveled completed autonomously.

The significance of VT&R1 lies in its demonstration and promotion of using overlapping locally consistent maps to achieve long-range autonomy. However, its practicality is limited due to the inability to support long-term operation. The visual features extracted from the stereo images were found to be highly susceptible to environment appearance changes, which led to a rapid increase in localization failures as the time gap between path teaching and repeating increases. The issue was partially addressed by interleaving frame-to-frame odometry and frame-to-map localization during path repeating so that odometry carries the robot forward over areas with poor localization. Nonetheless, the operational window of VT&R1 is limited to a few hours. Besides that, the mandatory offline map generation step causes extra inconveniences in working with the system. Addressing these limitations and inconveniences is the primary motivation for developing VT&R2.

2.1.3 Visual Teach & Repeat 2

VT&R2 came out in 2017 with significant improvements in long-term operation capability and ease of use. Its development also led to several publications. The improvements over VT&R1 are summarized as follows.

Building Network of Paths

VT&R2 inherits the hybrid topological/metric mapping approach from VT&R1 but represents the map as a relative pose graph [16] with metric information associated to vertices. Map building occurs online instead of offline, allowing a path to be repeated immediately after its initial demonstration. Each taught path is a chain of poses in the graph that can be connected to other paths on both ends. Path teaching can thus be split into multiple sessions where each session introduces a new path branching off and optionally merging into the existing paths. Path following has shown to be

smooth across the branching and merging points [17], meaning that the robot can seamlessly repeat paths built from multiple sessions. As a result, the relative pose graph structure can be viewed as a large-scale network of traversable paths, where the robot can autonomously navigate to any location in the network. A graphical user interface was also developed to simplify the process of 1) teaching new paths while connecting them to the existing network and 2) specifying the goal locations to which the robot should navigate.

Long-Term Visual Localization

The problem of localization failure due to environment changes is addressed in VT&R2 via two algorithmic improvements to VT&R1's visual localization pipeline: Multi-Channel Localization (MCL) [18] and Multi-Experience Localization (MEL) [4].

Multi-Channel Localization (MCL) should more precisely be described as a localization framework with the idea of fusing localization information from multiple sources into a single estimation problem in a tightly coupled way. A channel is a stream of information used to localize the robot. In VT&R2, it explicitly refers to a stream of stereo images that come from a single stereo camera and has passed through some transformation (e.g., RGB-to-grayscale, RGB-to-color-constant). The visual localization pipeline in VT&R1 is adapted to perform independent feature extraction and tracking for each channel and then fuse the data correspondences from all channels for state estimation. VT&R2 uses two instantiations of this framework: a light-resistant variant using color-constant images [3] and a duo-stereo variant using two stereo cameras facing forward and backward [2], respectively. The light-resistant variant adds one more channel for each camera that operates on the color-constant transformed image of its output. Color-constant transformation makes assumptions about the sensor and the environment and can produce an image whose observed color is largely independent of the varying illumination. The duo-stereo variant adds extra channels to handle inputs from a second stereo camera. It uses the increased field of view to get more visual features for localization. The MCL framework has been shown to extend the operation window of VT&R1 from a few hours to multiple days.

Multi-Experience Localization (MEL) uses additional views of the same path from repeat passes to handle gradual appearance change of the environment. An *experience* is a collection of environment information obtained from a single teach or repeat pass. In VT&R2, the environment information consists of only the visual features and their 3D locations. A multi-experience extension of the relative pose graph, termed Spatial-Temporal Pose Graph (STPG), is developed to store extra experiences from repeat passes and metrically relate them to the privileged experience from the teach pass. Through the STPG, the live experience can be matched against all past experiences simultaneously to solve a single estimation problem. These additional experiences are expected to capture the environment's appearance change and increase the chance of successful localization. To limit the otherwise unbounded computation cost of MEL, VT&R2 implements two algorithms to select a subset of experiences deemed most relevant to the live experience. The first algorithm computes Bag of Words (BoW) descriptors in addition to the visual features for each experience, whose similarity comparison can be performed very fast and used to select the most relevant experiences [19]. The second algorithm borrows the collaborative filtering technique from recommender systems, which intuitively selects experiences with the most similar feature matching history to the live experience [20]. VT&R2 with MEL is capable of operating across seasonal changes as long as

sufficient experiences exist to bridge the appearance gap.

Place-Dependent Terrain Assessment

Terrain assessment is introduced in VT&R2 to ensure safe path repeating. While general terrain assessment in an unstructured, outdoor environment is challenging, VT&R2 exploits the fact that the robot never leaves previously driven paths, thereby reformulating terrain assessment as a change detection problem. The traversability ahead of the robot is determined by its geometric similarity to previous views collected at the same place, and any significant change will cause the robot to stop and require human intervention. The algorithm was initially presented in [5] and later refined in [6]. Specifically, the incoming stereo images are processed into point clouds and converted to 2D feature grid maps. In [5], each map cell contains the average height of points in this cell, and the cell-wise height difference determines the similarity. In [6], the cells instead contain the distribution of points modeled by a Mixture-of-Gaussian to account for overhanging vegetation and ground surface roughness. This algorithm produces accurate terrain assessment results in a highly cluttered environment where general assessment approaches typically produce many false positives (i.e., traversable terrain is considered not traversable) and lead to overly conservative behavior.

Learning-Based Path Tracking

VT&R2 switches to a learning-based MPC path tracking algorithm that effectively uses past tracking experiences to reduce tracking error and increase tracking speed. This algorithm was initially proposed in [21] and later extended in [22] with the addition of worst-case tracking error estimation and constraint satisfaction. Its idea is to split the robot model in MPC into two parts: a fixed, *a priori* kinematic model and a learned disturbance model that captures environment disturbances and robot dynamics. The disturbance model is represented by a Gaussian Process trained on previous tracking experiences as a function of robot state, input, and other relevant variables. Using a Gaussian Process enables interpolating the learned disturbances with uncertainty, which is the key to predicting future states and generating worst-case scenarios. At each control iteration, the nominal state sequence is predicted using a sigma-point transform, and the worst-case scenarios are defined as sequences bounding the 3σ confidence region. A constrained optimization problem is then formulated and solved to find the control input ensuring that the worst-case tracking error stays below a specified limit. This algorithm demonstrated robust and accurate path tracking along difficult off-road paths in combination with VT&R2's high-performance visual localizer.

An extended field test of VT&R2 was conducted in 2016 at an untended gravel pit in Sudbury, Canada [1]. During the test, the system repeatedly traversed a 5km network of paths over 11 days (Figure 2.1), accumulating over 140km of autonomous driving with an autonomy rate of 99.6%. This high autonomy rate over long-range and long-term operation proved VT&R2's suitability and maturity even for industrial-grade applications. It was eventually offered by Clearpath Robotics as a general-purpose navigation package for their UGVs.

Figure 2.1: An orthomosaic map of the 5 km network of paths at the Ethier Sand and Gravel in Sudbury, Ontario, Canada, constructed during VT&R2 field testing. Image credit: Paton et al. [1]

2.1.4 Moving Forward

The success of VT&R2 did not mark the end of its development. There have been many other works centered around VT&R2, which either proposed algorithmic improvements or attempted to deploy VT&R2 on sensor-robot configurations other than stereo cameras and ground vehicles. For example, alternative visual localization algorithms have been proposed, including place-and-time-dependent feature descriptors [23], direct methods that rely on whole-image registration [24], or an end-to-end deep neural network from supervised training [25]. Guo and Barfoot [26] proposed an algorithm solving a robust variant of the Canadian Traveler Problem (CTP), which can be applied to VT&R2 for planning on the network of paths to minimize the risk of localization failure and path obstruction. Extensions to VT&R2's learning-based path tracker were presented in [27] and [28], which learn multiple disturbance models to account for varying robot dynamics under different conditions and select the most similar ones at runtime. A monocular camera version of VT&R2 was developed in [29], which leverages a local ground planarity assumption to obtain the scale information. Besides that, VT&R2 was also successfully deployed on a multirotor UAV for emergency return under GPS failure [30]. However, although these algorithms and deployments bring valuable contributions to VT&R2, they are unfortunately never added to the official codebase and eventually abandoned during post VT&R2 development. Our learned lesson is that, despite being successful in many aspects, VT&R2 suffers from poor generalizability to new algorithms and sensor-robot configurations. It is, in particular, missing the necessary interfaces that draw a boundary between the generic navigation algorithms and the tasks specific to the teach and repeat framework, such as constructing the network of paths and system state transitions.

VT&R2 is also not the only implementation of the teach and repeat framework in recent years. Other vision-based implementations have been presented in [31, 32, 33, 34] for evaluating various visual localization techniques dealing with appearance and viewpoint changes. However, most of these implementations do not build a network of paths, operate in a 3D environment, or even perform accurate metric localization. The maturity of these systems is not comparable to VT&R2. A lidar-based teach and repeat system is developed in [35], which is most similar to VT&R2 except that it uses ICP-based point cloud registration for odometry, mapping, and localization. Using a lidar sensor makes their system inherently robust to lighting changes, and the increased Field of View (FOV) allows the system to deviate from the reference path for obstacle avoidance. Unfortunately, their system is not open-source, and its development was discontinued after last use in [36]. More recently, Baril et al. [37] also developed a lidar teach and repeat system similar to that in [35] and used it to study the robustness of lidar localization to dense vegetation and snow precipitation and accumulation.

Overall, the teach and repeat navigation framework has demonstrated its value for both academic research and industrial applications. However, what is currently missing is an implementation that fully exploits its potential. VT&R2 comes close to meeting this requirement but does not generalize well to new algorithms and sensor-robot configurations. The implementation is also too old to use modern robotics tools, e.g., ROS2 and powerful machine learning libraries. Therefore, as presented in this thesis, VT&R3 is developed to fill these gaps. VT&R3 re-designs VT&R from the ground up to work with the newest software and hardware. It inherits the main features from VT&R2 without being restricted to a specific set of algorithms and sensor-robot configurations. This is achieved by re-formulating the estimation, planning, and control problems in VT&R to be more generic and

defining interfaces that separate them from teach and repeat navigation logics. The generalizability of VT&R3 will be demonstrated in Chapter 3 and 4.

So far, VT&R3 has been used in [8, 9] that uses GPS for odometry and relative localization, and in [7] that uses deep-learned visual features for localization across extreme change lighting conditions. Several partners, such as Defence Research and Development Canada (DRDC) and Clearpath Robotics, are also integrating VT&R3 into their own systems.

2.2 The Main Idea: Multi-Experience Topometric Map

This section presents the enabling idea of VT&R3: using a *multi-experience topometric map* to represent its operating environment. This representation allows VT&R3 to 1) incrementally construct a large-scale network of connected, traversable paths from multiple teach passes, 2) metrically relate additional views of the paths from repeat passes to their respective initial views using relative $SE(3)$ transformations, and 3) operate freely with any combination of sensors and algorithms for mapping and localization. It can be viewed as a generalization to the VT&R2 STPG data structure, and the differences are highlighted below.

2.2.1 Relative Pose Graph as the Topometric Map

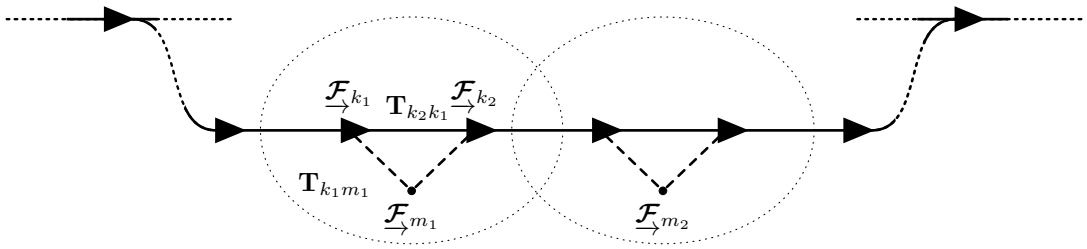


Figure 2.2: This figure illustrates VT&R3's topometric map structure. $\underrightarrow{\mathcal{F}}_k$ are frames of vertices (black triangles), and $\underrightarrow{\mathcal{F}}_m$ are frames of local maps (dotted ellipses). Each edge (solid line) stores the relative transformation between the two frames of its connected vertices (e.g., $\mathbf{T}_{k_2k_1}$). Each vertex is also associated with its spatially closest local map and stores the map-to-vertex transformation (e.g., $\mathbf{T}_{k_1m_1}$). Note that both vertex-to-vertex and map-to-vertex transformations can have an associated covariance matrix representing the uncertainty of the transformation.

At its core, VT&R3 follows VT&R2 to use a relative pose graph as its topometric environment representation. Each vertex in graph defines a local reference frame, which is referred to as a *vertex frame*, and each edge stores the relative $SE(3)$ transformation (with uncertainty [38]) between the frames of its linked vertices. In addition, each vertex has an associated local map that serves as a metric representation of its vicinity. The local maps define their respective reference frames with known relative transformation to all vertex frames with which they are associated. Note that this is different from VT&R2, where a one-to-one association between vertices and local maps is required, and the local maps must be expressed in the vertex frame. These restrictions from VT&R2 are relaxed so that one local map can be used by multiple vertices, allowing VT&R3 to generate sparsely distributed local maps of larger size without losing dense robot pose information along the path. Furthermore, VT&R3 has no restriction on what to store in the local maps, while VT&R2 only allows

point-based, sparse visual features. Figure 2.2 depicts the VT&R3 relative pose graph structure and its association between vertices and local maps.

2.2.2 The Multi-Experience Extension

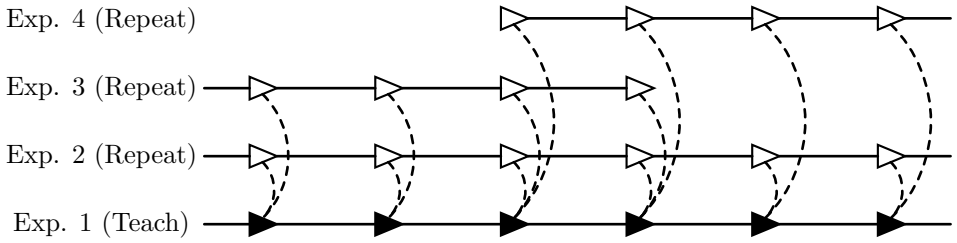


Figure 2.3: This figure illustrates VT&R3's multi-experience pose graph structure. Each row of vertices represents an experience. The initial experience (bottom row) comes from a teach pass, while additional experiences come from repeat passes. *Intra-experience* edges (solid line) connect vertices from the same experience, and *inter-experience* edges (dashed line) connect vertices from different experiences.

The notion of *experience* [39] arises naturally in a teach and repeat navigation framework. Since VT&R3 operates by alternating between teaching a new path and repeating an existing segment of the taught paths, each individual teach or repeat pass can be considered as adding a new experience to the topometric map, which results in a new chain of vertices in the pose graph as shown in Figure 2.3. The multi-experience extension to the pose graph, which used to be called STPG in VT&R2, keeps track of the respective experience of each vertex and distinguishes two types of edges: *temporal* edges link vertices from the same experience, and *spatial* edges link vertices from different experiences. This multi-experience extension is kept unchanged in VT&R3, except that *temporal* and *spatial* edges are renamed to *intra-experience* and *inter-experience* edges, respectively, to avoid confusion.

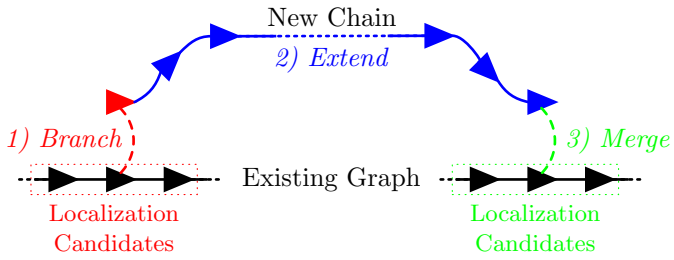
2.2.3 Privileged vs Non-Privileged Submap

Given a multi-experience topometric map built from several teach and repeat passes, the submap that contains all teach experiences is considered as the *privileged submap*, which forms the network of paths demonstrated to the robot. Planning and localization during repeats are both performed using this submap. The *non-privileged submap*, on the other hand, consists of all experiences from repeat passes and is used to provide additional views of the taught paths in a multi-experience localization framework.

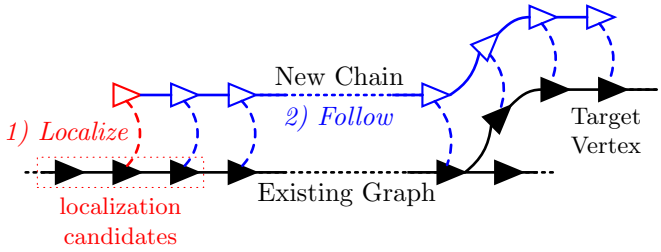
Privileged Submap Construction

The privileged submap is constructed incrementally by repeatedly branching off from and optionally merging into the existing submap. From the pose graph perspective, each teach pass adds a new chain to the graph that connects to the existing graph on one or both ends. Figure 2.4a shows the construction process of one teach pass, divided into three stages: *Branch*, *Extend*, and *Merge*.

Branch: prior to each teach pass (except the first one), a local window of vertices from the existing privileged submap should be specified as candidates for a branching point. When the teach



(a) Privileged Submap Construction (Teach Pass)



(b) Non-Privileged Submap Construction (Repeat Pass)

Figure 2.4: This figure illustrates the construction of (a) the privileged submap and (b) the non-privileged submap during each of the teach and repeat passes, respectively. Each teach pass is divided into three stages: *Branch*, *Extend*, and *Merge*, while each repeat pass is divided into two stages: *Localize* and *Follow*. Vertices and edges are colored according to the stage at which they are added to the graph. See Section 2.2.3 for more details.

pass begins, a new vertex is added to the graph immediately, initializing the new chain. Inter-experience localization is then performed to determine the robot's relative $SE(3)$ transformation to the spatially closest candidate using its associated local map. The result adds an inter-experience edge linking the new vertex to the closest candidate. Once this process is completed, the *Branch* stage ends, and the user can start driving the robot to extend the new chain.

Extend: while a new path is being taught, the robot's motion to the most recently created vertex is constantly estimated using a platform-specific odometry & mapping algorithm. A new vertex is appended to the end of the chain whenever the robot's motion exceeds a specified threshold. The $SE(3)$ transformation on the intra-experience edges as well as local maps are both outputs from the odometry & mapping algorithm.

Merge: the newly taught path may be merged back into the privileged submap to form a loop closure if the robot has traveled close to an existing location of the submap. *Merge* is similar to *Branch* where inter-experience localization is performed to determine the robot's spatially closest vertex from a list of user-specified candidates. Upon successful localization, an inter-experience edge linking the latest vertex from the current chain to the closest vertex is added to the privileged submap.

Note that the merging stage is optional, but the branching stage is currently non-skippable to ensure that the privileged submap is always connected.

Non-Privileged Submap Construction

The non-privileged submap can be seen as chains attached to the privileged submap, each constructed from one repeat pass. Assuming that a path to repeat is given as a chain of vertices from the privileged submap, the non-privileged submap construction can be divided into two stages: *Localize* and *Follow*, as shown in Figure 2.4b.

Localize: is essentially the same as the *Branch* stage above, where the robot is localized against a list of candidate vertices around the initial vertex of the repeat path. Successful localization results in the first inter-experience edge being added to the graph, connecting the new chain to the repeat path. VT&R3 then takes over the control of the robot to start following the path.

Follow: vertices and intra-experience edges are created from odometry & mapping similar to the *Extend* stage above. In the meantime, inter-experience localization constantly runs to determine the robot's pose with respect to the repeat path. Each successful localization either adds or refines the inter-experience edge connecting the latest vertex in the extending chain to its spatially closest vertex from the repeat path. The *Follow* stage ends when the robot reaches the target vertex, which is the end of the path.

Note that there is no problem extending a non-privileged chain across the branching and merging points in the privileged submap.

2.3 System Overview

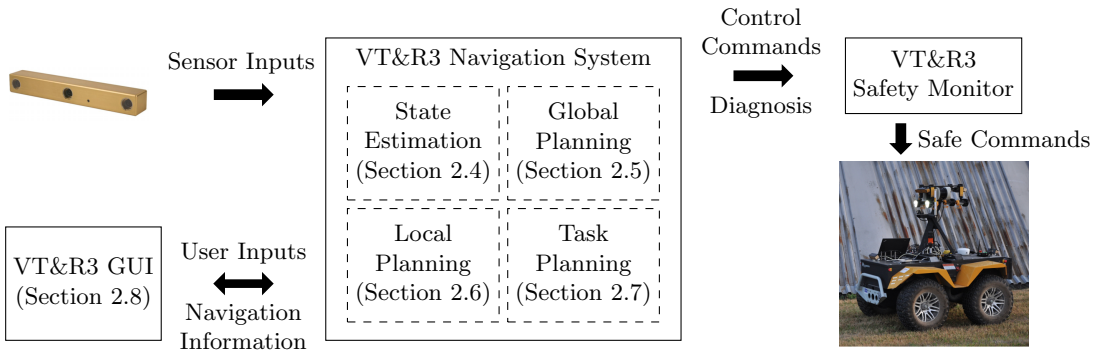


Figure 2.5: This figure provides a diagram overview of the VT&R3 system. At high level, VT&R3 has three components: VT&R3 Navigation System, VT&R3 GUI, and VT&R3 Safety Monitor. The navigation system can be divided further into four modules: state estimation, global planning, local planning, and task planning.

An overview of the VT&R3 system is shown in Figure 2.5. At a high level, it can be divided into three components: a navigation system, a graphical user interface, and a safety monitor.

The navigation system is the heart of VT&R3. It is a C++ program that employs a suite of state estimation, planning and control algorithms to complete tasks involved in the teach and repeat navigation framework. During execution, it receives sensory data from the robot's onboard sensors, exchanges information with the graphical user interface for visualization and accepting user commands, and sends control commands with diagnostics to the safety monitor. Communication between the navigation system and other components is done via the ROS2 middleware that defines

a secure and low-latency message-passing interface built on the Data Distribution Service (DDS) communication standard. Thanks to the strong ecosystem and community support of ROS2, VT&R3 can be deployed on various sensor-robot platforms for a wide range of applications.

Multiple modules exist in the navigation system to further split the tasks of state estimation, planning, and control. Each module defines its interface to communicate with other modules. Message passing through the interface is done via conventional C++ approaches, i.e., memory copy, references/pointers. During execution, each module runs in a separate software thread, while some modules may spawn more threads internally or dedicate some work to GPU for improved performance. The specific tasks of each module are detailed in Section 2.4-2.7.

The graphical user interface is a single-page web application that provides convenient tools for non-expert users to interact with the navigation system. Its server-side program (a.k.a the backend) is written in Python, and the client-side program (a.k.a the frontend) is written in JavaScript. The backend serves as a message converter to bridge the communication between the navigation system and the frontend. User inputs to the frontend are sent to the backend using WebSocket, a bi-directional real-time communication protocol built on a TCP connection. The backend converts the user inputs to ROS2 messages and publishes them to the navigation system. Meanwhile, incoming ROS2 messages from the navigation system pass reversely to the frontend. The main functionalities of the frontend are detailed in Section 2.8.

The safety monitor is an optional but recommended component for VT&R3. It runs as an independent process to ensure safe operation of the robot in case of system failure. The safety monitor should always inspect outcoming control commands from the navigation system before being sent to the robot. VT&R3 comes with a few template safety monitor implementations in C++ that can perform safety checks based on velocity limits and robot localization status. However, the specific functionalities of the safety monitor are platform and application dependent and thereby not discussed in this chapter.

2.4 State Estimation

In VT&R3, there are two state estimation problems to solve: an odometry & mapping problem and an inter-experience localization problem. As discussed in Section 2.2.3, odometry & mapping is used to construct new chains in VT&R3's multi-experience topometric map during both teach and repeat passes; inter-experience localization is required to determine the robot's pose metrically with respect to the repeat path during repeat passes while also connect a newly demonstrated path to the existing network via branching and merging during teach passes.

As mentioned earlier in Section 2.1.4, although there are numerous solutions to both problems with new algorithms constantly developed, few use a relative pose graph as the underlying data structure [16, 40]. Instead, many estimation algorithms prefer using a global map. Such differences lead to extra complexities in their integration into the VT&R system. To this end, the VT&R3 state estimation module defines a standard interface that exchanges information commonly producible by all kinds of odometry & mapping and localization algorithms regardless of their internal mechanisms. It then transforms the information into what is needed for VT&R map construction so that all extra complexities are handled in one place and hidden from the algorithm developers.

Furthermore, the state estimation module constructs a data processing pipeline that can par-

allelize the odometry & mapping process and the inter-experience localization process with most concurrency issues taken care of. It also leverages an asynchronous multi-thread execution model for additional time-insensitive tasks, such as pose graph optimization and local map maintenance. The number of processing threads and amount of parallelization are both configurable, allowing VT&R3 to scale flexibly with the available computation resources of the host system.

2.4.1 State Estimation Interfaces

Odometry & Mapping

The odometry & mapping interface requires its specialization to produce the following information from a sequence of sensor data:

1. a processing result flag f_s^O indicating whether the current sensor data is processed successfully;
2. a vertex creation flag f_v indicating whether a new vertex $\underrightarrow{\mathcal{F}}_{k+1}$ should be appended to the current chain;
3. an $SE(3)$ transformation $\hat{\mathbf{T}}_{rk}$ (optionally with uncertainty) between the current robot frame $\underrightarrow{\mathcal{F}}_r$ and the robot frame at the just-created vertex $\underrightarrow{\mathcal{F}}_k$;
4. if f_v is true, then a local map for the new vertex $\underrightarrow{\mathcal{F}}_{k+1}$ and the map-to-vertex tranformation $\mathbf{T}_{k+1,m}$.

The above information can be obtained from both global and relative odometry & mapping. By decoupling vertex frames and local map frames, this interface generalizes to both kinds of algorithms. For pure relative algorithms, such as VT&R2 visual odometry, the vertex frame $\underrightarrow{\mathcal{F}}_k$ coincides the map frame $\underrightarrow{\mathcal{F}}_m$ with one-to-one association, so that $\hat{\mathbf{T}}_{rk}$ comes directly from camera frame-to-frame registration. For global algorithms, the map frame $\underrightarrow{\mathcal{F}}_m$ is initialized at the beginning and kept fixed. In this case, performing frame to map registration results in an estimate of $\hat{\mathbf{T}}_{rm}$. $\hat{\mathbf{T}}_{rk}$ thereby can be obtained by

$$\mathbf{T}_{rk} = \mathbf{T}_{rm}\mathbf{T}_{km}^{-1}. \tag{2.1}$$

The vertex creation flag should be set to true when $\mathbf{T}_{rk}$ exceeds a specified threshold, and the local metric map can be a cropped version of the global map centered at the robot. An example is given in Figure 2.6, assuming that a global map is cropped periodically to be local maps, and the vertices are associated with their closest local maps, respectively.

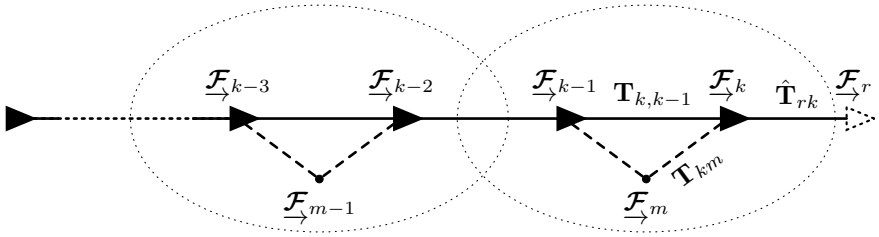


Figure 2.6: This figure shows an example of the pose chain and its relation to local maps during odometry & mapping, assuming the employed algorithm creates vertices more frequently than local maps. $\underrightarrow{\mathcal{F}}_r$ is the moving robot frame, $\underrightarrow{\mathcal{F}}_k$ are vertex frames, and $\underrightarrow{\mathcal{F}}_m$ are local map frames. Note that odometry & mapping is performed during both teach and repeat passes so that this chain can become either privileged or non-privileged part of the overall topometric map.

Upon each return from the odometry & mapping interface, the state estimation module caches

the result and sends $\hat{\mathbf{T}}_{rk}$ and f_s^O to the GUI and system monitor as live robot state updates and diagnostic information, respectively. If f_v is true, a new vertex with frame $\underrightarrow{\mathcal{F}}_{k+1}$ is appended to the current chain. By default, $\underrightarrow{\mathcal{F}}_{k+1}$ is same as $\underrightarrow{\mathcal{F}}_r$ at vertex-creation, so that the newly appended vertex is connected to its predecessor with $\mathbf{T}_{k+1,k} = \hat{\mathbf{T}}_{rk}$. The new vertex and edge are marked privileged if the system is in teach phase or non-privileged otherwise. After that, a vertex-creation callback is invoked with the new vertex instance passed as an argument for local map association and saving.

Inter-Experience Localization

Given 1) sensor input that has passed through odometry & mapping, 2) a target vertex $\underrightarrow{\mathcal{F}}_l$ with its associated local map and 3) an initial guess $\check{\mathbf{T}}_{rl}$, the inter-experience localization interface asks its specialization for a flag f_s^L indicating whether the localization is successful and a refined vertex-to-robot transformation $\hat{\mathbf{T}}_{rl}$. As discussed in Section 2.2.3, inter-experience localization is required at the *Branch* and *Merge* stage of the teach pass, and is required throughout the repeat pass. A successful localization (i.e., f_s^L is true) always results in the addition or refinement of an inter-experience edge in the pose graph. The resulting edge is marked privileged or non-privileged, depending on the current operating phase. It requires some explanation on how the target vertex $\underrightarrow{\mathcal{F}}_l$ and the initial guess $\check{\mathbf{T}}_{rl}$ are obtained.

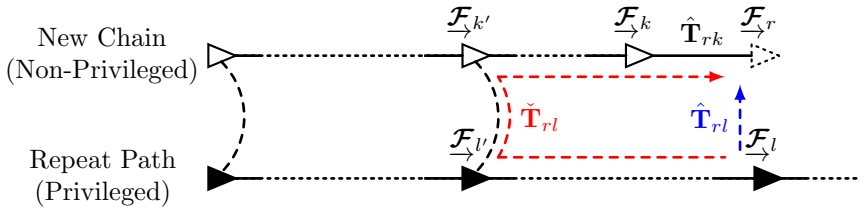


Figure 2.7: This figure shows the relevant pose graph structure during the *Follow* stage of a repeat pass. $\underrightarrow{\mathcal{F}}_r$ is the moving robot frame, $\underrightarrow{\mathcal{F}}_k$ and $\underrightarrow{\mathcal{F}}_{k'}$ are vertex frames of the current extending chain, and $\underrightarrow{\mathcal{F}}_l$ and $\underrightarrow{\mathcal{F}}_{l'}$ are vertex frames from the repeat path, which are from the privileged pose graph. $\underrightarrow{\mathcal{F}}_{k'}$ is the latest vertex frame that has been successfully localized to a vertex on the repeat path, $\underrightarrow{\mathcal{F}}_{l'}$. $\underrightarrow{\mathcal{F}}_l$ is the spatially closest vertex from the repeat path to $\underrightarrow{\mathcal{F}}_r$. An estimate of the transformation from $\underrightarrow{\mathcal{F}}_l$ to $\underrightarrow{\mathcal{F}}_r$, $\check{\mathbf{T}}_{rl}$, is generated by compounding transformations through $\underrightarrow{\mathcal{F}}_{m'}$, $\underrightarrow{\mathcal{F}}_{k'}$, and $\underrightarrow{\mathcal{F}}_k$. The employed inter-experience localization algorithm may use $\check{\mathbf{T}}_{rl}$ as both an initial guess and a prior to compute the a refined estimate $\hat{\mathbf{T}}_{rl}$.

Branch, *Merge* and *Localize*: a sequence of connected vertices are specified as candidates for localization. Until localization is successful (i.e., f_s^L is returned true), all candidates are passed through the interface iteratively with $\check{\mathbf{T}}_{rl}$ simply set to identity. Once f_s^L is returned true, the next candidate, which is supposed to be spatially closest to the robot, is selected by leveraging the refined transformation $\hat{\mathbf{T}}_{rl}$ and traversing through the pose graph edges that connect the candidates (see Figure 2.4). For example, assuming the previous candidate $\underrightarrow{\mathcal{F}}_{l'}$ has been localized successfully with result $\hat{\mathbf{T}}_{rl'}$, the initial estimate for the next candidate $\underrightarrow{\mathcal{F}}_l$ will be

$$\check{\mathbf{T}}_{rl} = \hat{\mathbf{T}}_{rl'}\mathbf{T}_{l'l}. \tag{2.2}$$

Follow: all vertices along the repeat path are considered candidates for localization. The next

target vertex is chosen by leveraging the last successful localization result, the latest odometry estimate, and traversing through the pose graph edges, which is illustrated in Figure 2.7. Given the pose graph in Figure 2.7, the initial estimate is

$$\check{\mathbf{T}}_{rl} = \mathbf{T}_{rk}\mathbf{T}_{kk'}\mathbf{T}_{k'l'}\mathbf{T}_{l'l}. \tag{2.3}$$

If localization is successful, an edge between the vertex of $\underrightarrow{\mathcal{F}}_l$ and the latest vertex of the current branch, $\underrightarrow{\mathcal{F}}_k$, is added if not exist, and the transformation on edge is set to

$$\hat{\mathbf{T}}_{kl} = \hat{\mathbf{T}}_{rk}^{-1}\hat{\mathbf{T}}_{rl}. \tag{2.4}$$

2.4.2 Pipeline and Parallelization

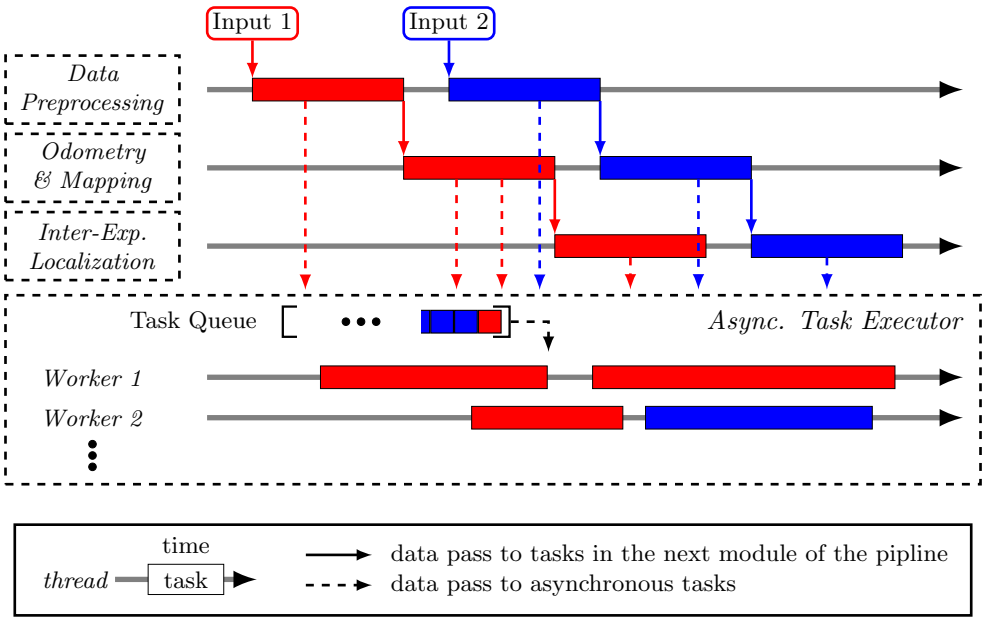


Figure 2.8: This figure illustrates the data processing pipeline and parallelization scheme of VT&R3's state estimation module. Time-sensitive tasks run on three dedicated threads for *data preprocessing*, *odometry & mapping*, and *inter-experience localization*, respectively. An asynchronous task executor is developed for time-insensitive tasks where the tasks are queued and executed later via a thread pool.

Processing Time-Sensitive Tasks

The data processing pipeline and its parallelization are presented in Figure 2.8. The pipeline consists of three modules: *Preprocessing*, *Odometry & Mapping*, and *Inter-Experience Localization*, each running on a dedicated thread. The second and third modules perform exactly as was described in Section 2.4.1. The *Preprocessing* module is prepended for potentially time-consuming tasks that can be separated from *Odometry & Mapping*, such as feature extraction from raw images or point clouds. With this setup, at most three sensor frames can be processed in parallel on a multi-core processor.

It is worth mentioning that this design of parallelization does not introduce any concurrency issues owing to the following guarantees:

1. the sensor input passes through each module sequentially, so it is never accessed by more than one thread at a time;
2. each of the three modules runs on its dedicated thread so that the resulting change to the pose graph from each module always happens sequentially;
3. *Odometry & Mapping* and *Inter-Experience Localization* of different inputs may run in parallel but never simultaneously access the same part of the pose graph.

For 3), note that *Odometry & Mapping* only adds vertices and intra-experience edges to the current branch. In contrast, *Inter-Experience Localization* only constructs inter-experience edges linking existing vertices.

The *Data Preprocessing* and *Odometry & Mapping* module should generally run at least as fast as the frame rate of the sensor to provide the most frequent updates to the robot's state. *Inter-Experience Localization* during path following may otherwise run slower if short-term odometry is accurate enough for the robot to follow the reference path. For example, VT&R2 performs localization only when a new vertex is created. In VT&R3, all modules run as frequently as possible by default but may choose to drop the present frame based on any criterion. An input frame is also discarded if it arrives at a module that is still busy with the previous one. Data buffers of arbitrary size are optionally added before each module to allow frames to be queued when the next module is not ready. The buffer helps avoid frame dropping due to slight run-time variations. However, one should note that if a module constantly runs slower than the sensor frame rate, the buffer will eventually get filled up so that either the pipeline throttles or the older frames are dropped.

Processing Time-Insensitive Tasks

While odometry & mapping and inter-experience localization are the two essential tasks of VT&R that have real-time requirements, there are many tasks for which a reasonable delay is acceptable. Examples of such tasks include: 1) local pose graph refinement such as sliding-window bundle adjustment in VT&R2, 2) local map maintenance such as dynamic object classification and removal from a point cloud map (Chapter 4), and 3) disk IO tasks such as saving or pre-loading local maps.

To handle these time-insensitive tasks, VT&R3 provides an asynchronous task executor to which a task of any kind can be submitted for execution via a pool of threads apart from the three dedicated threads for the data processing pipeline, also depicted in Figure 2.8. A unique id of the task is assigned upon submission, which may be used by subsequent tasks to specify as a prerequisite. Cyclic dependencies are prevented by design because only submitted tasks can be specified as prerequisites. Moreover, each task can specify a priority level at submission. The task executor internally uses a dependency graph to track task priorities and prerequisites. Whenever a worker thread is free, it will pick up the highest priority task that has all prerequisites satisfied. A bound on the number of queued tasks can also be specified. When the bound is reached and a new task is submitted, the lowest priority task or one of the tasks that recursively depends on it will be dropped.

Note that, although using the asynchronous task executor is a convenient and scalable way of handling time-insensitive tasks, there are two caveats to be aware of:

1. *latency*: it is currently impossible to guarantee a task's execution within a specific time after

submission. If the submitted tasks require more computation resources than is available, low-priority tasks will be queued forever and discarded.

2. *concurrency*: it is currently impossible to mark two or multiple tasks as mutually exclusive, i.e., only one task among them can be executed at a time. Tasks that access shared resources should thereby handle concurrency issues themselves.

2.5 Global Planning

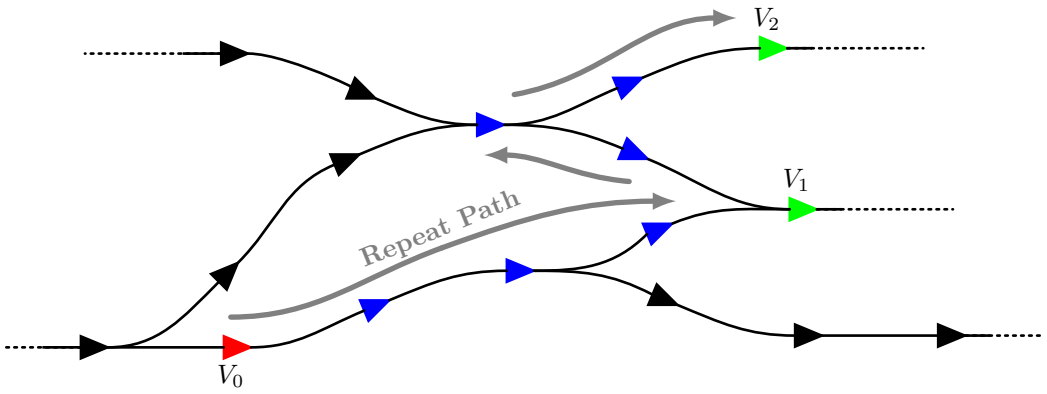


Figure 2.9: This figure shows an example of VT&R3's global planning problem. V_0 is the current closest vertex to the robot. V_1 and V_2 are user-specified vertices to visit in order during the repeat pass. The employed global planning algorithm should return an ordered list of vertices (blue) that starts at V_0, passes through V_1, and ends at V_2, which forms the repeat path and is sent to the state estimation module.

Since the robot is restricted to navigate along previously taught paths, global planning in VT&R amounts to finding the minimum-cost path under certain metrics between two vertices in the pose graph of the privileged submap. At the beginning of each repeat pass, a specialized global planning algorithm is invoked through an interface with 1) the closest vertex to the robot, V_0, 2) a list of vertices to be visited sequentially $V_1, ..., V_{K-1}$, and 3) access to the privileged subgraph. It should then return a chain of connected vertices from the subgraph that starts from V_0, passes through $V_1, ..., V_{K-1}$ iteratively, and ends at V_K. An example demonstrating the expected input and output when $K = 2$ is shown in Figure 2.9. The returned chain is then sent to the state estimation module as the repeat path shown in Figure 2.7.

The default global planning method in VT&R3 uses breadth-first search, which can be viewed as finding the optimal path using the same metric from which the graph is constructed. The returned path minimizes the total distance traveled when vertices are added with roughly constant linear and angular spacing. Note that the search is not directional, which assumes that any path in the network can be repeated in both directions, i.e., in the same and reversed direction it is taught.

Additionally, VT&R3 allows one to specify cost on edges using arbitrary heuristics and provides an implementation of Dijkstra's algorithm for convenience. For vision-based multi-experience localization in VT&R2, two edge costs that have been found particularly useful are time-since-last-traversal and time-of-day similarity.

2.6 Local Planning

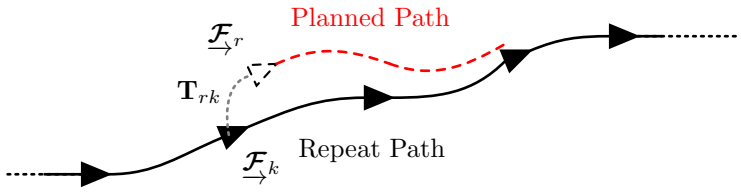


Figure 2.10: This figure illustrates VT&R3's local planning problem. $\underrightarrow{\mathcal{F}}_r$ is the moving robot frame with an estimated transformation from the spatially closest frame $\underrightarrow{\mathcal{F}}_k$ along the repeat path ($\mathbf{T}_{rk}$), which is constantly updated by the state estimation module. The employed algorithm should plan a local path and compute the corresponding control commands that drive the robot to stay close to the repeat path while, optionally, avoiding obstacles and untraversable terrain.

As shown in Figure 2.10, local planning in VT&R refers to finding the optimal local path under certain criteria and the corresponding control signals, given the repeat path, the current robot pose with respect to the path, and optionally local terrain and obstacle information. The state estimation module provides the current robot pose as a relative transformation between the current robot frame and the closest vertex frame in the repeat path. Local terrain information is optionally produced as byproducts of the state estimation module. VT&R3 has no restriction on the representation of terrain or obstacle information as long as the local planning algorithm can consume it.

Regarding implementation, the local planning module runs on a standalone thread during the *Follow* stage of the repeat phase. Communication between the state estimation module and the local planning module is done via a shared memory buffer due to its asynchronous nature. Specifically, the robot state and terrain information is typically updated at the sensor's frame rate while retrieved at the local planning rate required by the robot. The module constantly invokes the selected local planning algorithm through an interface with access to the shared memory. It then returns the control signal to be applied to the robot.

The current default local planning is an adapted version of the Time Elastic Band (TEB) local planner [41], which uses the TEB approach for time-optimal nonlinear model predictive control in $SE(2)$. At each control iteration, the closest vertex frame is used as the planning frame in which the planned trajectory will be represented. A local window of the repeat path is transformed into the planning frame using the relative transformation on edges. This window starts with the nearest vertex ahead of the robot and ends with a vertex up to a certain distance away. After that, both the robot and the transformed repeat path segment are projected into the horizontal plane of the planning frame and passed into the TEB planner. The current algorithm does not consider any terrain or obstacle information, so the trajectory is optimized to stay on the repeat path.

2.7 Task Planning

Task planning refers to the problem of structuring and controlling the switching between different tasks performed by the navigation system, triggered by certain events such as user commands and robot state changes. VT&R3 addresses this problem using a 2-level Hierarchical Finite State Machine (HFSM) as shown in Figure 2.11. At the higher level, it defines three states: *Idle*, *Teach* and *Repeat*.

Teach and *Repeat* are considered super-states since each of these contains multiple sub-states. One can notice that, except for *Plan*, these substates match exactly the different stages of the submap construction process previously described in Section 2.2.3, which suggests what tasks are performed at each substate. *Plan* is dedicated to global planning (Section 2.5), which computes the repeat path from user-specified target vertices. The associated tasks of each sub-state have been detailed in the previous sections.

A HFSM is chosen due to its modularity and simplicity in both interpretation and implementation. The downside is that HFSM is not a flexible and extensible solution for task planning compared to its alternatives, such as Behavior Trees [42]. However, since VT&R3 is designed specifically for teach and repeat navigation, the system states and behaviors are not expected to change across different applications. Therefore, the task planning module does not provide any interface for altering the HFSM or replacing it with other solutions.

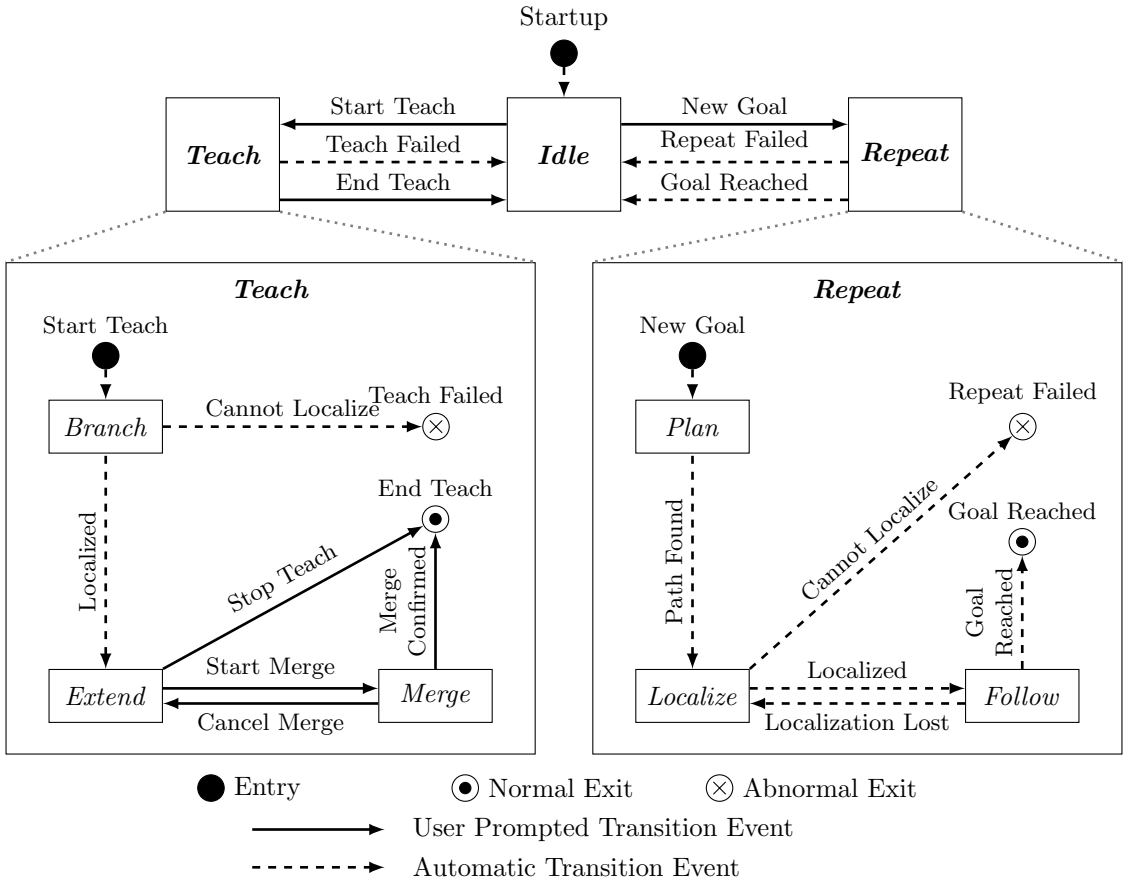


Figure 2.11: This figure shows the state transition diagram of VT&R3's HFSM, which is used for task planning. Normally, a complete teach pass should sequentially go through *Branch*, *Extend*, and optionally *Merge*. A complete repeat pass should go through *Plan*, *Localize*, and *Follow*. Additional state transitions are added to handle localization failures, resulting in early exiting.

2.8 The Graphical User Interface

The VT&R3 graphical user interface is designed to provide intuitive means for the user to interact with the system. A screenshot of the interface is shown in Figure 2.12. Its main components and their functionalities are as follows.

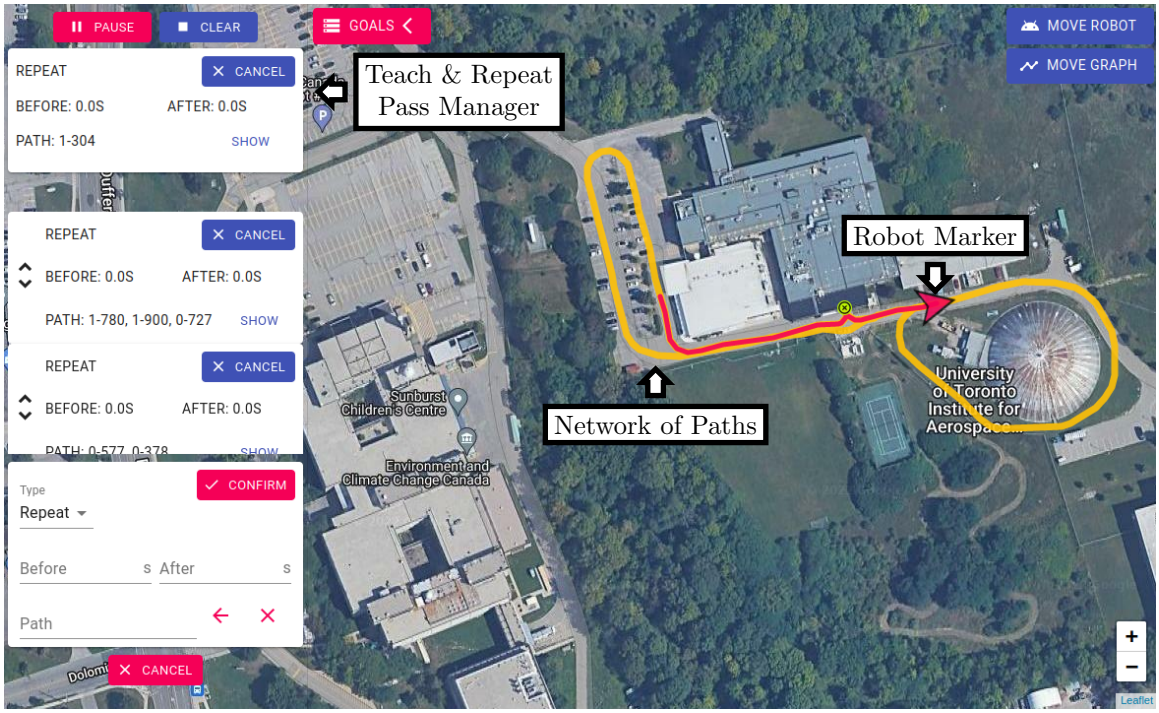


Figure 2.12: This figure shows a screenshot of the VT&R3 GUI, with the main components highlighted. See Section 2.8 for the details of each component.

2.8.1 The Network of Paths

The polylines in Figure 2.12 display the network of paths constructed so far. It is built from the pose graph of the privileged submap by first relaxing the graph in a global frame and then projecting it vertically to 2D. The user can use this visualization to specify 1) the repeat path, 2) the window for loop closure, and 3) the robot's nearest vertex. In addition, a tiled map is displayed in the background allowing the user to associate vertices to their real-world location. However, the network and the tiled map must be manually aligned because GPS is not required and is usually not used for VT&R3 operation. The GUI provides a convenient way to align the two by adjusting the network's relative location, orientation, and scale to the tiled map, with more details in the online wiki.

2.8.2 The Robot Marker

The robot marker, shown as an arrowhead in Figure 2.12, indicates the robot's current pose relative to the network. The marker is semitransparent when the robot is not metrically localized to the network and turns solid after successful localization. When the system is in *Idle* state, the user can change the robot's topological location by dragging the marker to the desired location on the

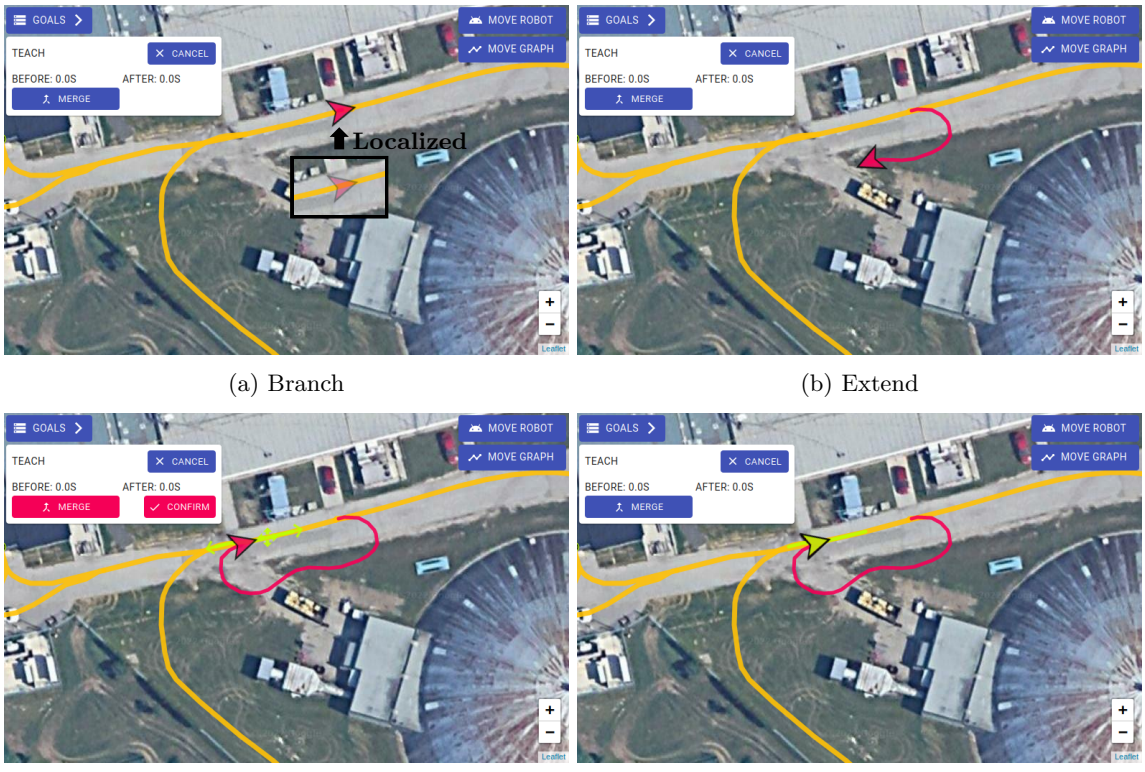
network of paths (see the online wiki). During path repeating, the location and orientation of the robot marker are constantly updated to reflect the live robot state as estimated by VT&R3.

2.8.3 The Teach & Repeat Pass Manager

The teach & repeat pass manager is a floating window located at the left of the GUI screen, as shown in Figure 2.12. It displays the current active teach or repeat pass and a list of queued passes to be executed afterward. New passes can be added using the pass submission pane at the bottom.

Teach Pass

Figure 2.13 shows the GUI states through out a teach pass. As soon as a teach pass becomes active, the VT&R3 system enters the *Branch* state where the robot is metrically localized to a local window of vertices centered at its current nearest vertex. Upon successful localization, the robot marker turns solid, and the system transitions into the *Extend* state, suggesting that the user can start driving the robot.



(a) Branch

(b) Extend

(c) Merge (Window Selection)

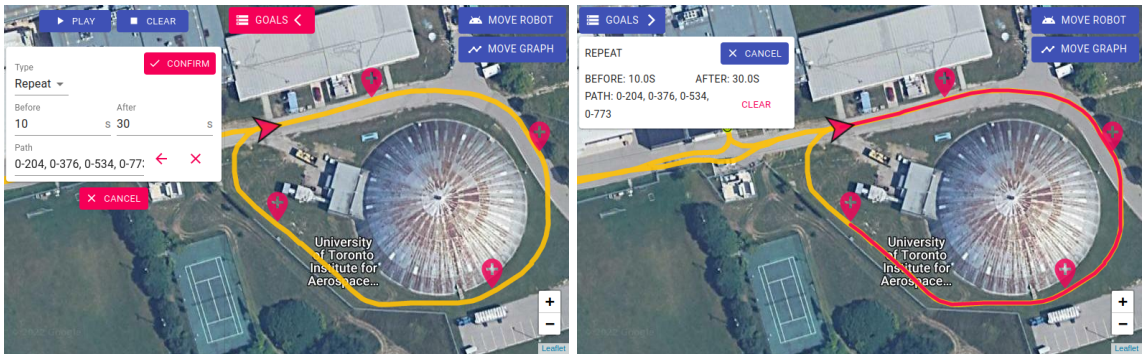
(d) Merge (Confirmation)

Figure 2.13: VT&R3 GUI states throughout a teach pass. (a) The robot marker turns from semi-transparent to solid after localization. (b) A new path is extended from the branching point. (c) The user initiates a *merge* and choose the localization window. (d) The user confirms the merge result by clicking the green arrowhead.

When the robot has been driven close to an existing area of the network, the user can initiate the *Merge* process (Figure 2.13c). A small region close to the robot should be selected and sent to the system as candidate vertices for localization. Successful localization pops up a green arrowhead

marker (Figure 2.13d) suggesting the estimated robot pose relative to the selected region. The user can then click on the arrowhead to confirm the result. The pose estimate is constantly updated, allowing the user to make minor adjustments before they are satisfied with the result. Once confirmed, the VT&R3 system switches back to *Idle* state, and the privileged pose graph is relaxed with the new loop closure constraint and then re-projected on the GUI.

Repeat Pass



(a) Waypoints Selection (b) Path Repeating

Figure 2.14: VT&R3 GUI states throughout a repeat pass. (a) The user specifies the waypoints to be visited. (b) VT&R3 plans a path through the waypoints and starts repeating the path.

Adding a repeat pass requires the user to specify a list of vertices to be visited in order, which can be done by simply clicking on the network polylines (Figure 2.14a). An arbitrary delay can be specified before or after the execution of each repeat pass, which is useful when the robot needs to stop at different places to perform other tasks. Once a repeat is added, no further action from the user is required since all the system state transitions happen automatically during the repeat pass.

2.9 Summary

In summary, we presented Visual Teach & Repeat 3 (VT&R3) in this chapter that is designed to meet the primary goal of this thesis, namely, a teach and repeat system that reflects the generalizability of the underlying framework for easy adaptation to various sensor/robot combinations (Section 1.1). VT&R3 achieves this goal by re-formulating the estimation, planning, and control problems involved in teach and repeat navigation to be more generic and defining interfaces that separate framework specific logics from sensor/robot dependent algorithms. VT&R3 also takes advantage of the popular ROS2 library, which enables safe, reliable and efficient communication with any ROS2-enabled sensors and robots. VT&R3 is an open-source project under permissive licensing at `https://github.com/utiasASRL/vtr3`.

We mentioned that VT&R3 has been used in [8, 9] exploring the use of GPS for odometry and relative localization. These works demonstrate the benefit of VT&R3's generalizable implementation in that it immediately allows using a different sensor or fusing multiple sensors (e.g., GPS and stereo camera in [8]) for teach and repeat navigation. In Chapter 3 and 4, we will futher take advantage of VT&R3 to develop and evaluate navigation algorithms using a radar/lidar as the primary sensor.

Chapter 3

Lidar & Radar Topometric Mapping and Localization

In this chapter, we present the second contribution of this thesis: a lidar/radar topometric mapping and localization pipeline leveraging VT&R3's state estimation interface described in Section 2.4. Our pipeline can be configured to perform radar-only, lidar-only, or cross-modal radar-to-lidar localization. We conduct an extensive comparison between these three variants using a public dataset and present our findings on the accuracy and robustness of each variant across varying seasonal and weather conditions.

3.1 Introduction

Many autonomous driving companies leverage detailed semantic maps to drive safely. These maps may include the locations of lanes, pedestrian crossings, traffic lights, and more. In this case, the vehicle no longer has to detect each of these features from scratch in real-time. Instead, given the vehicle's current position, the semantic map can be used as a prior to simplify the perception task. However, it then becomes critical to know the pose of the robot within the map with sufficient accuracy and reliability.

Dense lidar maps can be built using offline batch optimization while incorporating IMU measurements for improved local alignment and GPS for improved global alignment [43]. Highly accurate localization can be subsequently performed by aligning a live lidar scan with a pre-built map with reasonable robustness to weather conditions [44, 45]. Vision-based mapping and localization is an alternative that can be advantageous in the absence of environment geometry. However, robustness to large appearance changes (e.g., lighting) is a difficult and ongoing research problem [7]. Radar-based systems present another compelling alternative.

Models of atmospheric attenuation show that radar can operate under certain adverse weather conditions where lidar cannot [46, 47]. These conditions may include heavy rain (>25mm/hr), dense fog (>0.1g/m^3), or a dust cloud (>10g/m^3). Existing literature does not describe the operational envelope of current lidar or radar sensors for the task of localization. Prior works have assumed that lidar localization is susceptible to moderate rain or snow, necessitating the use of radar. In this chapter, we attempt to shed some light on this topic by comparing the performance of three

topometric mapping and localization pipelines developed using VT&R3's state estimation interface (Section 2.4). These pipelines perform radar-only, lidar-only, and cross-modal radar-to-lidar localization, respectively. We compare these pipelines across varying seasonal and weather conditions using our publicly available Boreas dataset [10].

3.2 Related Work

Automotive radar sensors now offer range and azimuth resolutions approximately on par with mechanically actuated radar. It is possible to replace a single 360-degree rotating radar with several automotive radar paneled around a vehicle [48]. Each target will then enjoy a relative (Doppler) velocity measurement, which can be used to estimate ego-motion [49]. However, recent work [50, 51] seems to indicate that the target extraction algorithms built into automotive radar may not necessarily be optimal for mapping and localization. Thus, sensors that expose the underlying signal data offer greater flexibility since the feature extraction algorithm can be tuned for the desired application.

Extracting keypoints from radar data and subsequently performing data association has proven challenging. The first works to perform radar-based localization relied on highly reflective objects installed within a demonstration area [52, 53]. These reflective objects were thus easy to discriminate from background noise. Traditional radar filtering techniques such as Constant False Alarm Rate (CFAR) [54] have proven to be difficult to tune for radar-based localization. Setting the threshold too high results in insufficient features, which can cause localization to fail, while setting the threshold too low results in a noisy radar point cloud and a registration process that is susceptible to local minima.

Several promising methods have been proposed to improve radar-based localization. Jose and Adams [55] demonstrated a feature detector that estimates the probability of target presence while augmenting their SLAM formulation to include radar cross section as an additional discriminating feature. Chandran and Newman [56] maximized an estimate of map quality to recover both the vehicle motion and radar map. Rouveure et al. [57] and Checchin et al. [58] eschewed sparse feature extraction entirely by matching dense radar scans using 3D cross-correlation and the Fourier-Mellin transform. Callmer et al. [59] demonstrated large-scale radar SLAM by leveraging vision-based feature descriptors. Mullane et al. [60] proposed to use a random-finite-set formulation of SLAM in situations of high clutter and data association ambiguity. Vivet et al. [61] and Kellner et al. [49] proposed to use relative Doppler velocity measurements to estimate the instantaneous motion. Schuster et al. [62] demonstrated a landmark-based radar SLAM that uses their Binary Annular Statistics Descriptor (BASD) to match keypoints. Rapp et al. [63] used Normal Distributions Transform (NDT) to perform probabilistic ego-motion estimation with radar.

Cen and Newman [64] demonstrated low-drift radar odometry over a large distance that inspired a resurgence of research into radar-based localization. Several datasets have been created to accelerate research in this area including the Oxford Radar RobotCar dataset [65], MulRan [66], and RADIATE [67]. We have recently released our own dataset, the Boreas dataset[1], which includes over 350km of data collected on a repeated route over the course of 1 year.

More recent work in radar-based localization has focused on either improving aspects of radar

[1] `https://www.boreas.utias.utoronto.ca/`

odometry [68, 69, 70, 71, 72, 73, 50, 74, 75, 76, 77], developing better SLAM pipelines [78, 79, 80], or performing place recognition [81, 82, 83]. Barnes et al. [71] trained an end-to-end correlation-based radar odometry pipeline. Barnes and Posner [72] demonstrated radar odometry using deep learned features and a differentiable SVD-based estimator. In [74], we quantified the importance of motion distortion in radar odometry and showed that Doppler effects should be removed during mapping and localization. Subsequently, in [75], we demonstrated unsupervised radar odometry, which combined a learned front-end with a classic probabilistic back-end.

Alhashimi et al. [77] present the current state of the art in radar odometry. Their method builds on prior work by Adolfsson et al. [76] by using a feature extraction algorithm called BFAR to add a constant offset b to the usual CFAR threshold: $T = a \cdot Z + b$. The resulting radar point clouds are registered to a sliding window of keyframes using an ICP-like optimizer while accounting for motion distortion.

Other related work has focused on localizing radar scans to satellite imagery [84, 85, 86], or to pre-built lidar maps [87, 88]. Localizing live radar scans to existing lidar maps built in ideal conditions is a desirable option as we still benefit from the robustness of radar without incurring the expense of building brand new maps. However, the global localization errors reported in these works are in the range of 1m or greater. We demonstrate that we can successfully localize live radar scans to a pre-built lidar map with a relative localization error of around 0.1m.

In this work, we implement topometric mapping and localization following the VT&R strategy. No GPS or IMU measurements are used. Hong et al. [80] recently compared the performance of their radar SLAM to SuMa, surfel-based lidar SLAM [45]. On the Oxford RobotCar dataset [65], they show that SuMa outperforms their radar SLAM. However, in their experiments, SuMa often fails partway through a route. Our interpretation is that SuMa losing track is more likely due to an implementation detail inherent to SuMa itself rather than a shortcoming of all lidar-based SLAM systems. It should be noted that Hong et al. did not tune SuMa beyond the original implementation, which was tested on a different dataset. In addition, Hong et al. tested SuMa using 32-beam lidar, whereas the original implementation used a 64-beam lidar. Furthermore, Hong et al. only provide a qualitative comparison between their radar SLAM and SuMA in rain, fog, and snow, whereas our work provides a quantitative comparison across varying weather conditions. In some of the qualitative results they presented, it is unclear whether SuMa failed due to adverse weather or due to geometric degeneracy in the environment, which is a separate problem. Importantly, our results seem to conflict with theirs by showing that lidar localization can operate successfully in even moderate to heavy snowfall. Although, it is possible that topometric mapping and localization is more robust to adverse weather since it uses odometry to obtain a very good initial guess/prior for localization to a map constructed in nominal weather.

3.3 Methodology

3.3.1 Pipeline Overview

As shown in Figure 3.1, our radar/lidar topometric mapping and localization pipeline leverages VT&R3's state estimation interface and its pipeline and parallelization scheme, which has been detailed in Section 2.4. Readers are referred to Section 2.4.1 for the desired input/ouput of the

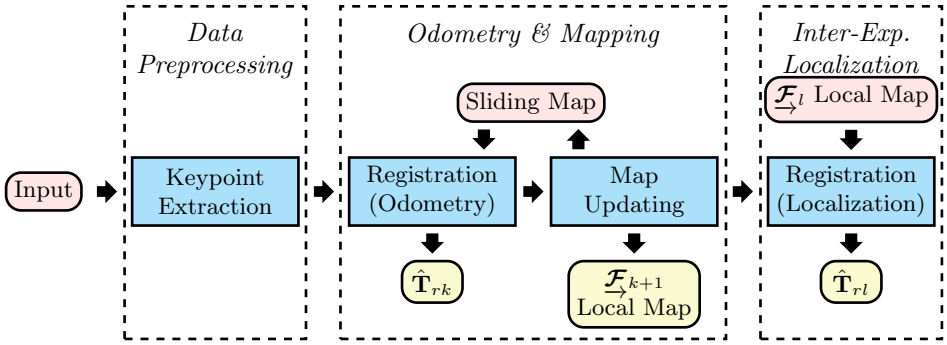


Figure 3.1: The data processing pipeline diagram for lidar/radar topometric mapping and localization, leveraging VT&R3's state estimation interface (Section 2.4) and its pipeline and parallelization scheme (Figure 2.8). See Section 3.3.1 for details.

pipeline and Figure 2.8 for the pipeline execution model. In VT&R3, the pipeline is divided into three modules: *Data Preprocessing*, *Odometry & Mapping*, and *Inter-Experience Localization*. In this work, we assign the following tasks to each module:

Keypoint Extraction: extracts keypoints from raw sensor data (either lidar or radar scans) to be used for scan-to-map registration. More sensor-specific information is given in Section 3.3.2.

Registration (Odometry): estimates the robot pose with respect to the latest vertex frame, $\hat{\mathbf{T}}_{rk}$, by registering the live scan to the current sliding map. The sliding map is a point cloud accumulated from previous scans and is constantly updated to be bounded in size and centered at the robot's location. The sliding map frame, denoted $\underrightarrow{\mathcal{F}}_{m}$, is initialized at the beginning and kept fixed. The live scan is also motion-compensated and sent to *Map Updating* and *Registration (Localization)*. We present the details of our motion-compensated odometry algorithm, CT-ICP, in Section 3.3.3.

Map Updating: adds the motion-compensated scan to the sliding map and removes points too far away from the robot. If either translation or rotation of $\hat{\mathbf{T}}_{rk}$ exceeds a threshold of 10m/30°, we ask VT&R3 to add a new vertex $\underrightarrow{\mathcal{F}}_{k+1}$ to the pose graph with $\mathbf{T}_{k+1,k} = \hat{\mathbf{T}}_{rk}$. The associated local map is the current sliding map transformed to the frame $\underrightarrow{\mathcal{F}}_{k+1}$ of the newly added vertex .

Registration (Localization): localizes the robot frame, $\underrightarrow{\mathcal{F}}_{r}$, against the local map of the spatially closest vertex frame, $\underrightarrow{\mathcal{F}}_{l}$, on the repeat path (i.e., $\hat{\mathbf{T}}_{rl}$ as shown in Figure 2.7). The way we find the closest vertex is described in Section 2.4.1, which leverages past localization results and odommetry updates. We present the details of the ICP optimization in Section 3.3.5.

3.3.2 Keypoint Extraction

Lidar

For each incoming lidar scan, we first perform voxel downsampling with voxel size $dl = 0.3$m. Only one point that is closest to the voxel center is kept. Next, we extract plane features from the downsampled point cloud by applying Principle Component Analysis (PCA) to each point and its neighbors from the raw scan. We define a feature score from PCA to be

$$s = 1 - \lambda_{\min}/\lambda_{\max}, \tag{3.1}$$

where $\lambda_{\min}$ and $\lambda_{\max}$ are the minimum and maximum eigenvalues, respectively. The downsampled point cloud is then filtered by this score, keeping no more than 20,000 points with scores above 0.95. Note that we use this score as a simple heuristic to remove points that are not on a planar surface. Specifically, a point on a planar surface may have one eigenvalue that is significantly smaller than the other two. With a threshold of 0.95, we only retain points whose largest eigenvalue is more than 20 times greater than the smallest eigenvalue. This is a sufficient but not necessary condition for a point to be not on a planar surface.

Radar

For each radar scan, we first extract point targets from each azimuth using the Bounded False Alarm Rate (BFAR) detector as described in [77]. BFAR adds a constant offset b to the usual Cell-Averaging CFAR threshold: $T = a \cdot Z + b$. We use the same (a, b) parameters as [77]. For each azimuth, we also perform *peak detection* by calculating the centroid of contiguous groups of detections as is done in [64]. We obtained a modest performance improvement by retaining the maximum of the left and right sub-windows relative to the cell under test as in (greatest-of) GO-CFAR [54]. These polar targets are then transformed into Cartesian coordinates and are passed to the *Odometry & Mapping* module without further filtering.

3.3.3 Continuous-Time Iterative Closest Point

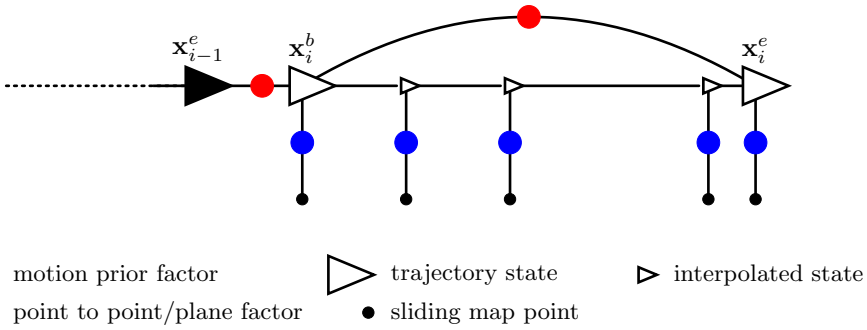


Figure 3.2: Factor graph of our CT-ICP algorithm. $\mathbf{x}_i^b$ and $\mathbf{x}_i^e$ are states of the underlying discrete trajectory at the beginning and end of the current scan i. $\mathbf{x}_{i-1}^e$ is the state at the end of the previous scan $i-1$. The interpolated states depend on the state variable $\mathbf{x}_i^b$ and $\mathbf{x}_i^e$.

Our CT-ICP algorithm combines the iterative data association of ICP with a continuous-time trajectory represented as exactly sparse Gaussian Process (GP) regression [89]. Our trajectory is $\mathbf{x}(t) = \{\mathbf{T}(t), \boldsymbol{\varpi}(t)\}$, where $\mathbf{T}(t) \in SE(3)$ is the robot pose (i.e., the relative transformation from the sliding map frame, $\underrightarrow{\mathcal{F}}_m$, to the robot frame, $\underrightarrow{\mathcal{F}}_r$), and $\boldsymbol{\varpi}(t) \in \mathbb{R}^6$ is the body-centric velocity. Following Anderson and Barfoot [89], our motion prior is

$$\begin{aligned} \dot{\mathbf{T}}(t) &= \boldsymbol{\varpi}(t)^{\wedge}\mathbf{T}(t), \\ \dot{\boldsymbol{\varpi}} &= \mathbf{w}(t), \quad \mathbf{w}(t) \sim \mathcal{GP}(\mathbf{0}, \mathbf{Q}_c\delta(t-\tau)), \end{aligned} \tag{3.2}$$

where $\mathbf{w}(t) \in \mathbb{R}^6$ is a zero-mean, white-noise Gaussian Process. This prior is applied in a piecewise fashion between an underlying discrete trajectory of pose-velocity state pairs, $\mathbf{x}_i = \{\mathbf{T}_i, \boldsymbol{\varpi}_i\}$, that

correspond to the representative timestamps of the ith sensor scan. Currently, we choose two representative timestamps for each scan, one at the beginning ($\mathbf{x}_i^b$) and the other at the end ($\mathbf{x}_i^e$). We seek to align the latest sensor scan i to the sliding map in frame $\underrightarrow{\mathcal{F}}_m$ by optimizing for the states $\mathbf{x}_i$.

We define a nonlinear optimization problem for $\mathbf{x}_i$, locking all previous states. The cost function is

$$J_{\text{odom}} = \phi_{\text{motion}} + \underbrace{\sum_{j=1}^{M} \left(\frac{1}{2} \mathbf{e}_{\text{odom},j}^T \mathbf{R}_j^{-1} \mathbf{e}_{\text{odom},j} \right)}_{\text{measurements}}. \tag{3.3}$$

Readers are referred to Anderson and Barfoot [89] for the squared-error cost expression of the motion prior, ϕ_{motion}.

Each measurement error term is

$$\mathbf{e}_{\text{odom},j} = \mathbf{D} \left(\mathbf{p}_m^j - \mathbf{T}(t_j)^{-1} \mathbf{T}_{rs} \mathbf{q}_j \right), \tag{3.4}$$

where $\mathbf{q}_j$ is a homogeneous point with corresponding timestamp t_j from the ith sensor scan, $\mathbf{T}_{rs}$ is the extrinsic calibration from the sensor frame $\underrightarrow{\mathcal{F}}_s$ to the robot frame $\underrightarrow{\mathcal{F}}_r$, $\mathbf{T}(t_j)$ is a pose from our trajectory queried at t_j. Note that, through GP interpolation, $\mathbf{T}(t_j)$ depends on the state variables $\mathbf{x}_i^b$ and $\mathbf{x}_i^e$ [89]. $\mathbf{p}_m^j$ is a homogeneous point from the sliding map associated to $\mathbf{q}_j$ and expressed in $\underrightarrow{\mathcal{F}}_m$, and $\mathbf{D}$ is a constant projection that removes the 4th homogeneous element. We define $\mathbf{R}_j^{-1}$ as either a constant diagonal matrix for radar data (point-to-point) or by using the outer product of the corresponding surface normal estimate for lidar data (point-to-plane). A factor graph of our CT-ICP formulation is shown in Figure 3.2.

We optimize for $\mathbf{x}_i$ iteratively using Gauss-Newton, but with nearest-neighbour data association after every Gauss-Newton iteration. CT-ICP is therefore performed with the following steps:

1. Temporarily transform all points $\mathbf{q}_j$ to frame $\underrightarrow{\mathcal{F}}_m$ using the latest trajectory estimate (motion undistortion).
2. Associate each point to its nearest neighbour in the map to identify its corresponding map point $\mathbf{p}_m^j$ in $\underrightarrow{\mathcal{F}}_m$.
3. Formulate the cost function J_{odom} in Equation (3.3) and perform a single Gauss-Newton iteration to update $\mathbf{x}_i$.
4. Repeat steps 1 to 3 until convergence.

Since VT&R3's odometry & mapping interface requires a single pose estimate per scan with respect to the most latest vertex frame $\underrightarrow{\mathcal{F}}_k$. We interpolate to get the robot pose with respect to the sliding map, $\hat{\mathbf{T}}_{rm}$, at the middle of the scan. The frame-to-vertex transformation is then obtained by

$$\hat{\mathbf{T}}_{rk} = \hat{\mathbf{T}}_{rm} \mathbf{T}_{km}^{-1}, \tag{3.5}$$

where $\mathbf{T}_{km}^{-1}$ is cached from previous estimates.

3.3.4 Doppler-Compensated CT-ICP

In [74], it is showed that Navtech radar sensors are susceptible to Doppler distortion and that this effect becomes significant during mapping and localization. A relative velocity between the

sensor and the surrounding environment causes the received frequency to be altered according to the Doppler effect. If the velocity in the sensor frame is known, this effect can be compensated for using a simple additive correction factor $\Delta\mathbf{q}_j$. In this work, we include this correction factor, which depends on a continuous-time interpolation of the estimated body-centric velocity $\boldsymbol{\varpi}(t)$ at the measurement time of each target t_j, in the measurement error term for radar CT-ICP:

$$\mathbf{e}_{\text{odom},j} = \mathbf{D}\left(\mathbf{p}_m^j - \mathbf{T}(t_j)^{-1}\mathbf{T}_{rs}(\mathbf{q}_j + \Delta\mathbf{q}_j)\right) \tag{3.6}$$

$$\text{where } \Delta\mathbf{q}_j = \mathbf{D}^T\beta\mathbf{a}_j\mathbf{a}_j^T\mathbf{D}\mathbf{q}_j^{\odot}\text{Ad}(\mathbf{T}_{sr})\boldsymbol{\varpi}(t_j), \tag{3.7}$$

where β is the Doppler distortion constant inherent to the sensor [74], and $\mathbf{a}_j$ is a 3×1 unit vector in the direction of $\mathbf{q}_j$.

3.3.5 Localization ICP

We use ICP to localize the motion-compensated live scan of the robot frame, $\underrightarrow{\mathcal{F}}_r$, against the local map of the spatially closest vertex frame, $\underrightarrow{\mathcal{F}}_l$, of the repeat path. The resulting relative transformation is $\hat{\mathbf{T}}_{rl}$, as required by VT&R3's inter-experience localization interface, as shown in Figure 2.7. The nonlinear cost function is

$$J_{\text{loc}} = \phi_{\text{pose}} + \sum_{j=1}^{M}\left(\frac{1}{2}\mathbf{e}_{\text{loc},j}^T\mathbf{R}_j^{-1}\mathbf{e}_{\text{loc},j}\right), \tag{3.8}$$

where we use $\check{\mathbf{T}}_{rl}$ from Equation (2.3) as an initial guess and a prior:

$$\phi_{\text{pose}} = \frac{1}{2}\ln(\check{\mathbf{T}}_{rl}\mathbf{T}_{rl}^{-1})^{\vee T}\mathbf{Q}_{rl}^{-1}\ln(\check{\mathbf{T}}_{rl}\mathbf{T}_{rl}^{-1})^{\vee}. \tag{3.9}$$

The covariance $\mathbf{Q}_{rl}$ can be computed by compounding the edge covariances corresponding to the relative transformations in Equation (2.3). Since all point clouds are already motion-compensated, the measurement error term is simply

$$\mathbf{e}_{\text{loc},j} = \mathbf{D}\left(\mathbf{p}_l^j - \mathbf{T}_{rl}^{-1}\mathbf{T}_{rs}\mathbf{q}_j\right), \tag{3.10}$$

where $\mathbf{q}_j$ is a homogeneous point from the motion-compensated live scan, and $\mathbf{p}_l^j$ is the nearest neighbour local map point to $\mathbf{q}_j$ after being transformed to the localization vertex frame $\underrightarrow{\mathcal{F}}_l$. See Section 3.3.3 for how we define $\mathbf{T}_{rs}$, $\mathbf{D}$, and $\mathbf{R}_j^{-1}$.

3.4 Evaluation

3.4.1 Dataset

Our experimental platform, depicted in Figure 3.3, includes a 128-beam Velodyne Alpha-Prime lidar, a FLIR Blakfly S monocular camera, a Navtech radar, and an Applanix POS LV GNSS-INS. Our lidar has a 40° vertical field of view, 0.1° vertical angular resolution, 0.2° horizontal angular resolution, and produces roughly 220k points per revolution at 10Hz up to 300m. The Navtech is a frequency modulated continuous wave (FMCW) radar with a 0.9° horizontal angular resolution and

Figure 3.3: Our platform, *Boreas*, includes a Velodyne Alpha-Prime (128-beam) lidar, a FLIR Blackfly S camera, a Navtech CIR304-H radar, and an Applanix POS LV GNSS-INS.

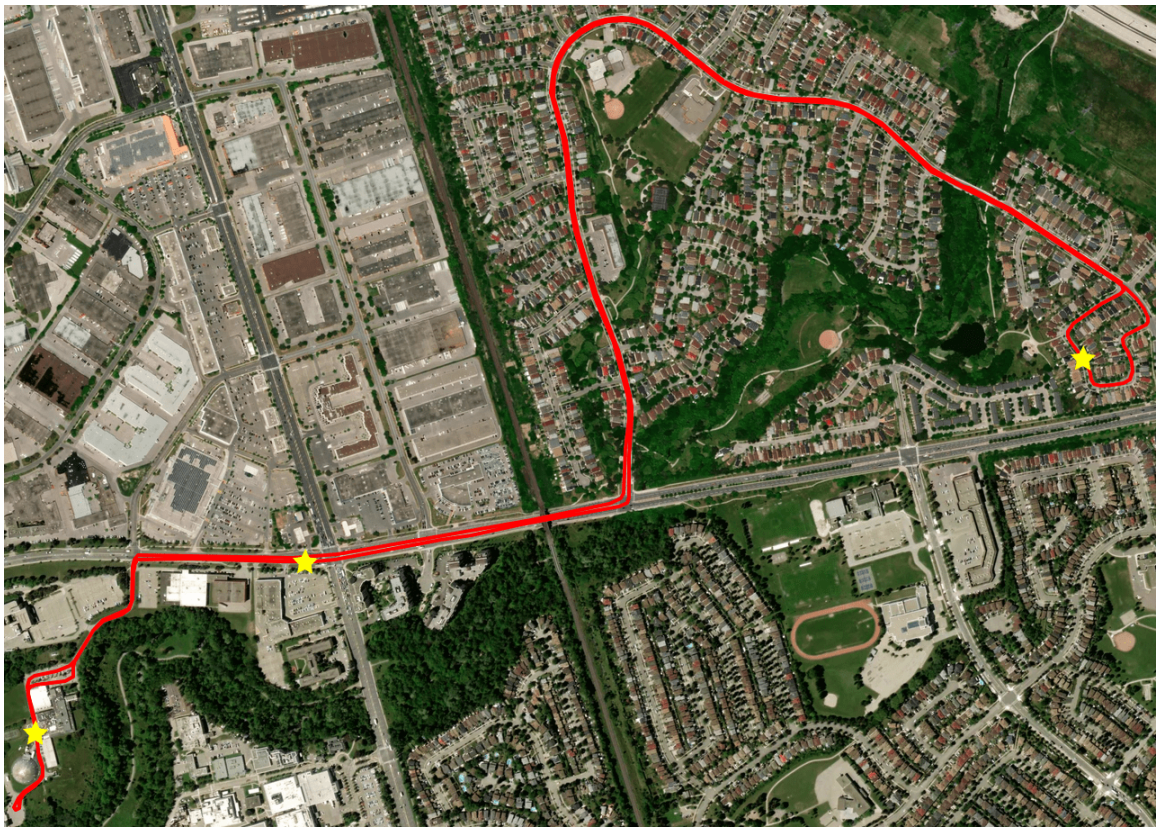


Figure 3.4: The 8km Glen Shields route in Toronto. The yellow stars correspond to UTIAS, Dufferin, and Glen Shields (left to right) as in Figure 3.5.

5.96cm range resolution, which provides measurements up to 200m at 4Hz. The test sequences used in this paper are part of our Boreas dataset, which contains over 350km of driving data collected by driving a repeated route over the course of one year. We intend to support live and open benchmarks for odometry, metric localization, and 3d object detection. Ground truth poses are obtained by post-processing GNSS, IMU, and wheel encoder measurements. An RTX subscription was used to achieve cm-level accuracy without a base station. RTX uses data from a global network of tracking stations to calculate corrections. The residual error of the post-processed poses reported by Applanix is typically 2-4cm in nominal conditions.

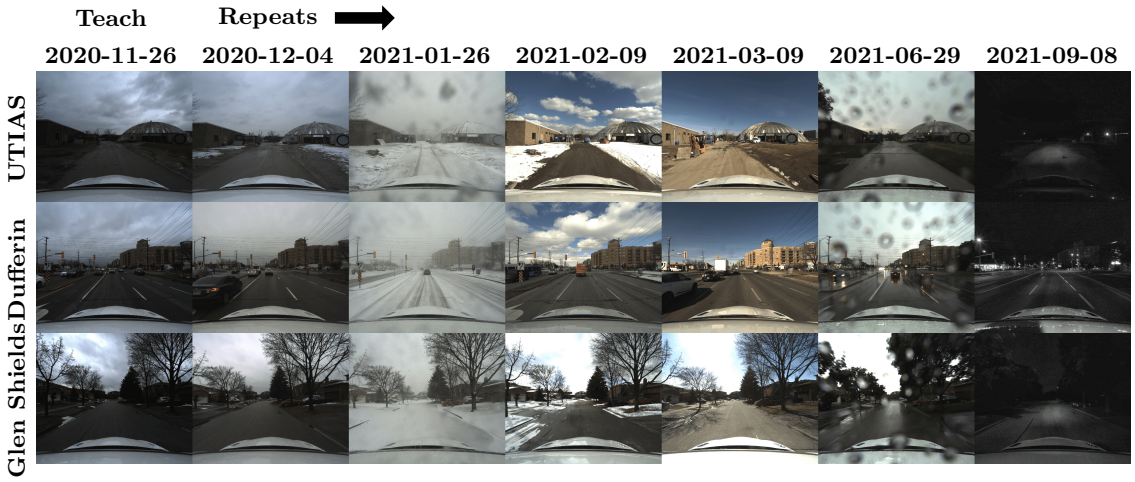


Figure 3.5: Our test sequences were collected by driving a repeated route over the source of one year. In our experiments, we use 2020-11-26 as our reference sequence for building maps. The remaining six sequences, which include sequences with rain and snow, are used to benchmark localization performance, which amounts to 48km of test driving. These camera images are provided for context.

In this experiment, we used seven sequences of the Glen Shields route (shown in Figure 3.4) chosen for their distinct weather conditions. These sequences are depicted in Figure 3.5. During the teach pass, a map is constructed using the reference sequence 2020-11-26. The radar-only and lidar-only pipelines use their respective sensor types to construct the map. No GPS or IMU information is required during the map-building process. Note that our test sequences include a significant amount of seasonal variation with ten months separating the initial teach pass and the final repeat pass. Sequences 2021-06-29 and 2021-09-08 include trees with full foliage while the remaining sequences lack this. 2021-01-26 was collected during a snowstorm, 2021-06-29 was collected in the rain, and 2021-09-08 was collected at night.

3.4.2 Metrics

During each of the repeat passes, our pipeline outputs a relative localization estimate between the live robot frame $\underrightarrow{\mathcal{F}}_r$ and a vertex frame $\underrightarrow{\mathcal{F}}_l$ in the repeat path, i.e., $\hat{\mathbf{T}}_{rl}$. We then compute RMSE values for the relative translation and rotation error as in [10]. We separate translational error into lateral and longitudinal components. Since the Navtech radar is a 2D sensor, we restrict our comparison to $SE(2)$ by omitting vertical (z-axis) errors and reporting heading error as the rotation error.

3.4.3 Results

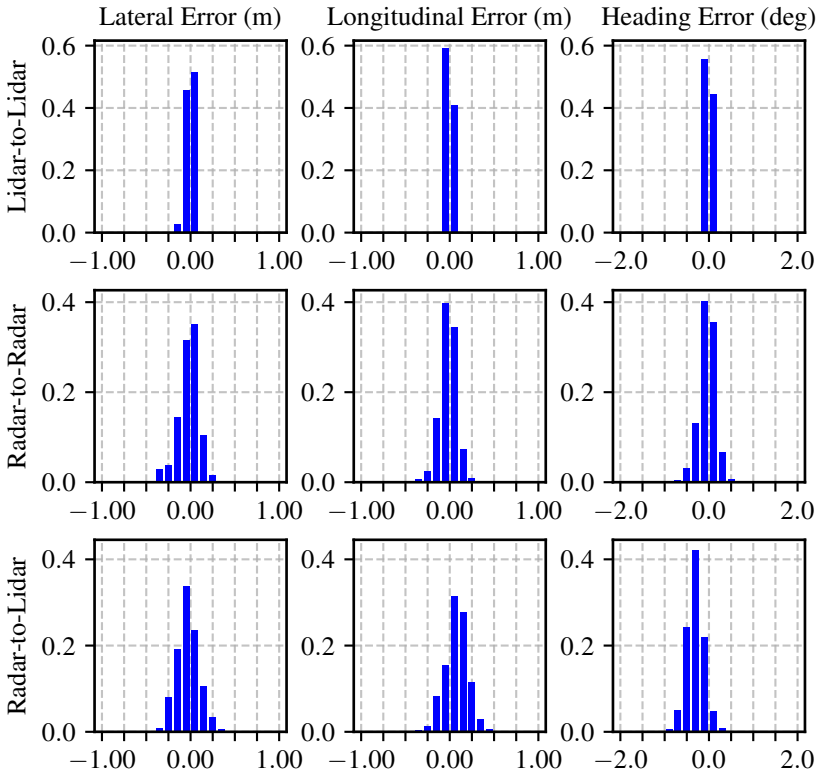


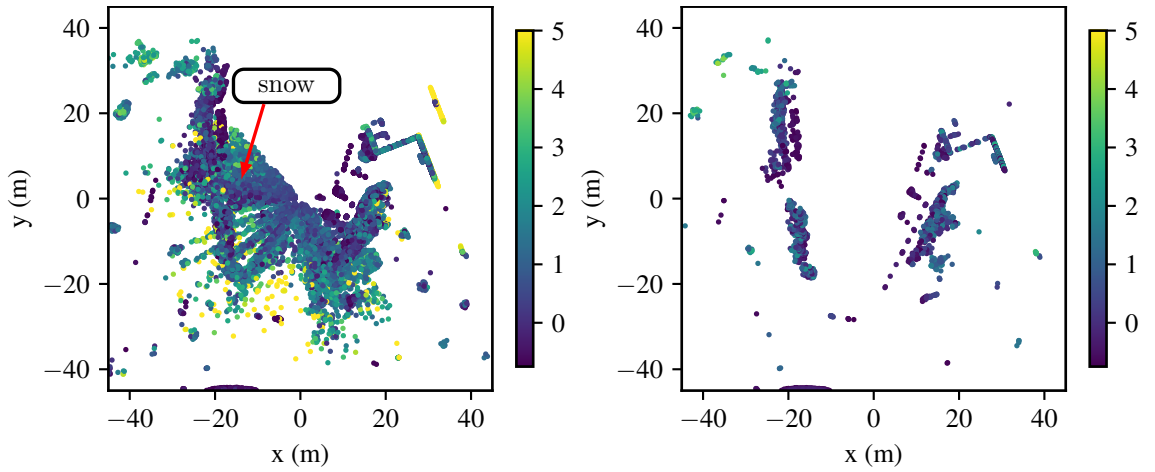
Figure 3.6: These histograms show the spread of the localization error for lidar-to-lidar, radar-to-radar, and radar-to-lidar, during the snowstorm sequence 2021-01-26.

Figure 3.6 depicts the spread of localization error during sequence 2021-01-26. Note that, although lidar-to-lidar localization is the most accurate, radar-to-radar localization remains reasonably competitive. When localizing radar scans to lidar maps, both the longitudinal error and heading error incur a bias. The longitudinal bias could be due to some residual Doppler distortion effects, and the heading bias could be the result of an error in the radar-to-lidar extrinsic calibration. A video showcasing radar mapping and localization can be found at this link[2].

Figure 3.8 shows the localization errors as a function of time during the snowstorm sequence 2021-01-26. Surprisingly, lidar localization appears to be unperturbed by the adverse weather conditions. During the snowstorm sequence, the lidar point cloud becomes littered with detections associated with snowflakes and a large section of the horizontal field of view becomes blocked by a layer of ice as shown in Figure 3.7 (a). However, in Figure 3.7 (b), we show that in actuality, these snowflake detections have little impact on ICP registration after filtering by normal score and only retaining the inliers of truncated least squares. Charron et al. [90] previously demonstrated that snowflake detections can be removed from lidar point clouds, although we do not use their method here. The robustness of our lidar pipeline to both weather and seasonal variations is reflected in the RMSE results displayed in Table 3.1.

Our results show that, contrary to the assertions made by prior works, lidar localization can be robust to moderate levels of precipitation and seasonal variation. Clearly, more work is required by

[2] `https://youtu.be/okS7pF6xX7A`



(a) Radar-to-Radar (b) Radar-to-Lidar

Figure 3.7: This figure illustrates the noisy lidar data that was used to localize during the 2021-01-26 sequence. Points are colored by their z-height. In (a) the ground plane has been removed and the point cloud has been randomly downsampled by 50% to highlight the snowflake detections. Note that a large forward section of the lidar's field of view is blocked by a layer of ice. However, as the results in Table 3.1 and Figure 3.8 show, lidar localization remains quite robust under these adverse conditions. (b) shows the lidar point cloud after filtering points by their normal score as in Equation 3.1 and only retaining the inliers of truncated least squares. Note that the snowflake detections seem to disappear, illustrating the robustness of our lidar pipeline.

Table 3.1: Metric Localization RMSE Results
Reference Sequence: 2020-11-26

	Lidar-to-Lidar		
	lateral (m)	longitudinal (m)	heading (deg)
2020-12-04	0.057	0.082	0.025
2021-01-26	0.047	0.034	0.034
2021-02-09	0.044	0.037	0.029
2021-03-09	0.049	0.040	0.022
2021-06-29	0.052	0.058	0.042
2021-09-08	0.060	0.041	0.030
mean	**0.052**	**0.049**	**0.030**
	Radar-to-Radar		
	lateral (m)	longitudinal (m)	heading (deg)
2020-12-04	0.130	0.127	0.223
2021-01-26	0.118	0.098	0.201
2021-02-09	0.111	0.089	0.203
2021-03-09	0.129	0.093	0.213
2021-06-29	0.163	0.155	0.241
2021-09-08	0.155	0.148	0.231
mean	**0.134**	**0.118**	**0.219**
	Radar-to-Lidar		
	lateral (m)	longitudinal (m)	heading (deg)
2020-12-04	0.132	0.179	0.384
2021-01-26	0.130	0.153	0.360
2021-02-09	0.126	0.152	0.389
2021-03-09	0.140	0.152	0.360
2021-06-29	0.159	0.161	0.415
2021-09-08	0.169	0.165	0.401
mean	**0.143**	**0.161**	**0.385**

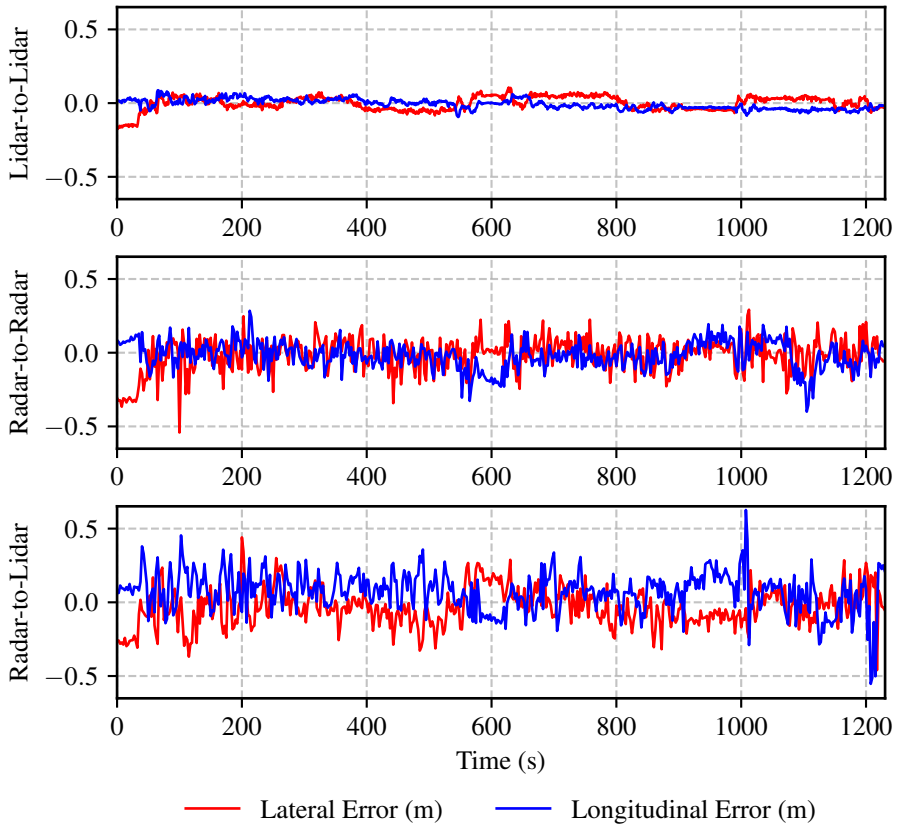
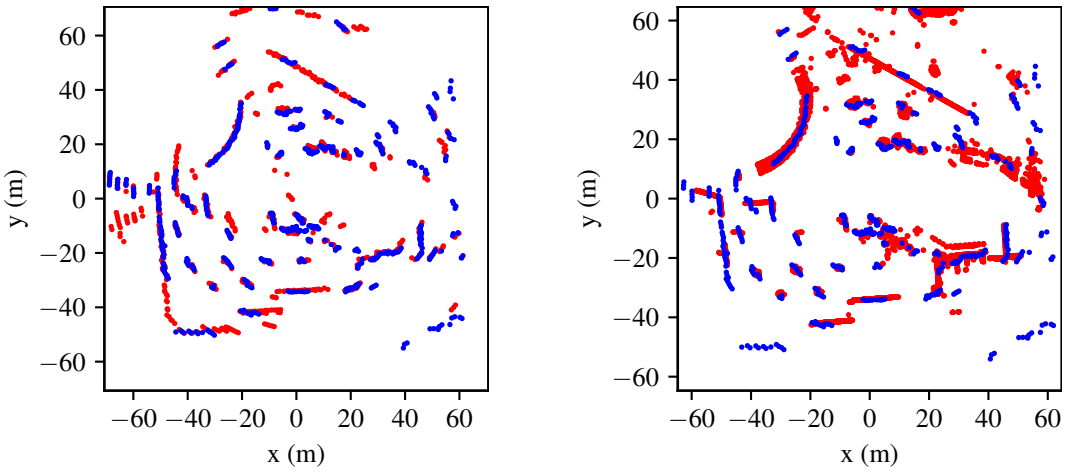


Figure 3.8: Here we plot metric localization errors during the snowstorm sequence 2021-01-26. Note that the lidar localization estimates remain accurate even with 1/4 of its field of view being blocked by a layer of ice as shown in Figure 3.7.



(a) Radar-to-Radar

(b) Radar-to-Lidar

Figure 3.9: This figure shows the live radar point cloud (blue) registered to a submap (red) built during the teach pass. In (a) we are performing radar-to-radar localization and so the submap is made up of radar points. In (b) we are localizing a radar point cloud (blue) to a previously built lidar submap.

the community to identify operational conditions where radar localization has an obvious advantage. These conditions may include very heavy precipitation, dense fog, or dust clouds. Nevertheless, we demonstrated that our radar localization is reasonably competitive with lidar. Furthermore, radar localization may still be important as a redundant backup system in autonomous vehicles. Figure 3.9 illustrates the live scan and submap during radar-to-radar and radar-to-lidar localization.

It is important to recognize that the results reported in this work are taken at a snapshot in time. Radar localization is not as mature of a field as lidar localization and radar sensors themselves still have room to improve. Note that incorporating IMU or wheel encoder measurements would improve the performance of all three compared systems. The detector we used, BFAR [77], did not immediately work when applied to a new radar with different noise characteristics. It is possible that a learning-based approach to keypoint extraction and matching may improve performance. Switching to a landmark-based pipeline or one based on image correlation may also be interesting avenues for comparison.

Radar-to-lidar localization is attractive because it allows us to use existing lidar maps, which many autonomous driving companies already have, while taking advantage of the robustness of radar sensing. Radar-based maps are not as useful as lidar maps since they lack sufficient detail to be used to create semantic maps.

Computation and Storage Requirements

Table 3.2: Pipeline Computation and Storage Requirements

	Teach (FPS)	Repeat (FPS)	Storage (MB/km)
Lidar-to-Lidar	3.6	3.0	86.4
Radar-to-Radar	**5.3**	**5.1**	**5.6**
Radar-to-Lidar	N/A	**5.1**	86.4

In Table 3.2, we show the computation and storage requirements of the different pipelines discussed in this work. Recall that we perform odometry & mapping during both teach and repeat passes and additionally localize against pre-built maps during repeat passes. We used a Lenovo P53 laptop with Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz and 32GB of memory. Our radar maps use significantly less storage (5.6MB/km) than our lidar maps (86.4MB/km).

3.5 Summary

In this work, we developed and compared the performance of three topometric mapping and localization pipelines that are different in using what sensor for mapping and localization, respectively. Specifically, we attempted to localize radar data against radar maps and either radar or lidar data against lidar maps. Our experiments showed that lidar-only mapping and localization is quite robust to adverse weather such as a snowstorm with a partial sensor blockage due to ice, suggesting that the sensitivity of lidar localization to moderate precipitation has been exaggerated in prior works. However, our radar-only pipeline was able to achieve competitive accuracy with a much smaller map, making it a viable alternative to lidar pipelines. More experiments are needed to identify conditions where the performance of radar pipelines exceeds that of lidar.

Furthermore, all three pipelines are available to be used with VT&R3, enabling lidar/radar-based teach and repeat navigation when combined with other navigation modules for planning and control. In the next chapter, we extend our lidar-only pipeline to detect obstacles in changing environments.

Chapter 4

Lidar Map Maintenance and Obstacle Detection

In this chapter, we present the third contribution of this thesis: an extended lidar mapping and localization pipeline with map maintenance and obstacle detection capabilities for obstacle-aware navigation in changing environments. Map maintenance filters out temporary objects from lidar local maps, while obstacle detection treats such objects as obstacles and labels them in live lidar scans. We demonstrate the performance of the resulting pipeline using real-world datasets and propose an idea of combining it with a local planner for obstacle avoidance during path repeating (future work).

4.1 Introduction

VT&R achieves autonomous navigation in complex unstructured environments using human-taught paths, which is powerful but relies on the assumption that the taught paths are always traversable. This assumption results in safety concerns in changing environments where obstacles may be introduced to the paths. In VT&R2, this problem is addressed by having a lookahead terrain assessment module that stops the robot when the path ahead looks significantly different from past experiences. However, it is still frustrating to see the robot constantly stop in front of obstacles that could have been avoided by temporarily deviating from the repeat path. One of our goals for VT&R3 is to prevent the taught paths from limiting the robot's path-planning capabilities. Past experiences on taught paths should be used to infer the environment dynamics and safety boundaries, facilitating planning and generation of new paths.

In this work, we take a step toward this goal by introducing map maintenance and obstacle detection capabilities to our lidar-only pipeline presented in Chapter 3. We classify objects in the environment into three categories: 1) *dynamic objects*, objects observed to be moving as the robot traverses through the environment; 2) *short-term static objects*, objects that appear, move, or disappear between experiences; and 3) *long-term static objects*, objects that persist across multiple experiences. Map maintenance identifies and retains only the constantly observed long-term static objects in the lidar local maps. Obstacle detection treats both dynamic and short-term static objects as obstacles and produces pointwise obstacle information from each incoming lidar scan. Both are

achieved via simple change detection with respect to past experiences, requiring minimal assumptions and prior knowledge about the environment. In particular, our obstacle detection approach does not treat the ground plane any differently from other objects/structures in the environment, as opposed to existing methods relying on ground segmentation as a preprocessing step [91, 92]. The performance of our extended pipeline has been evaluated on datasets of typical VT&R operating environments, from highly structured to highly unstructured.

Furthermore, we propose a way to use the detected obstacle information for safe obstacle avoidance during path repeating. We extend the VT&R3 GUI allowing a *safe corridor* to be specified along the taught paths within which the terrain is always traversable unless occupied by a dynamic or short-term static obstacle. We convert both the safe corridor and the obstacle information into 2D cost maps and pass them to VT&R3's default TEB local planner [41]. These cost maps encourage local deviation from the repeat path to avoid obstacles while restricting the maximum deviation below the safe corridor boundaries.

4.2 Related Work

4.2.1 Change Detection

We are concerned with detecting changes in the environment from two sets of lidar measurements or their derived representations (i.e., point clouds, voxel grids), with one considered a query and the other as a reference. The problem is frequently encountered in lidar-based localization, mapping, and object recognition tasks. Regarding existing solutions, the most direct approach is accumulating measurements from both sets into point clouds, computing the nearest neighbor distances between points from two clouds, and classifying changes by thresholding the distance [93]. Despite its simplicity, decent results can be achieved if the point clouds are dense and there is no need to reason about occlusions.

Some methods take one step further by clustering point clouds into more compact representations for improved efficiency and outlier robustness. Andreasson et al. [94] represents the reference point cloud by the Normal Distributions Transform (NDT). Change is detected by measuring the Mahalanobis distance between a query point and its nearest Gaussian. A similar approach is proposed in [95], which extracts Gaussian mixture models from both query and reference point cloud and quantifies change based on the Earth Mover's Distance. Vieira et al. [96] convert both point clouds into 3D density functions, which cluster points via implicit surfaces defined by the constant density boundaries. The difference in densities then determines the changed space.

Alternatively, if a probabilistic sensor model is specified, the likelihood of query measurements can be computed from reference measurements with change detected via Bayesian inference. An example is presented in [97], which models a lidar range measurement as a Gaussian centered at the predicted distance given no change and a uniform distribution otherwise. Another example can be found in [98], which detects change between two surface reconstructions from an RGB-D camera. The sensor model, in their case, accounts for not only depth measurement but also color and surface orientation information, demonstrating the advantage of Bayesian inference in fusing information from multiple sensing modalities.

However, the approaches above still cannot distinguish real changes from occlusions because

unmeasured space is implicitly assumed unoccupied. To avoid spurious changes due to occlusions, unmeasured space and measured free space must be handled differently, necessitating ray-tracing techniques.

Previous works have used ray-tracing for both 2D and 3D grid-based mapping, classifying cells as occupied, free and unknown [99, 100]. Such approaches can be used directly for change detection by comparing the resulting maps from the query and reference measurements at the cell level. To avoid the unfavorable grid-based discretization, Hebel et al. [101] apply ray-tracing directly to a point cloud representation for point-wise classification. In this approach, a voxel discretization of the point cloud is still required to facilitate the search of neighboring points along each laser beam. However, occupancy information is assigned to points instead of voxels according to their distances to the laser beam and end-point. Information from multiple beams is fused using the Dempster-Shafer theory of evidence, a common alternative to the Bayesian approach. Points lying in the free space of all rays with strong evidence are classified as change. A similar approach is proposed in [102] with minor differences in modeling the free and occupied space of laser beams.

Applying ray-tracing through voxel grids for neighboring points search is both computation and memory expensive. Underwood et al. [103] propose to project both rays and points into spherical coordinates where neighborhood search is done with $O(1)$ complexity. The method is mainly designed for a spinning-lidar configuration where all rays can be assumed to originate from the same location. Although used by multiple previous works (e.g., [104, 105, 106, 107]), including our work in this chapter, one should note that the above assumption is not strictly valid when the sensor platform is moving and causing motion distortion of lidar scans. This issue is mentioned in [108], which proposes another approach that accounts for platform motion while still exploiting some aspects of the lidar scan pattern. However, its computation advantage compared to voxel-based algorithms is no longer evident.

It is also worth mentioning that penetrable objects (e.g., vegetation) cause additional complexities in change detection because such objects define space that is not completely occupied or free. In representing environments with such objects, previous works often resort to real-valued grid maps whose values represent, for example, the probability of reflection [109] or the expected laser ray length [110]. A change detection method operating on such real-valued maps is proposed in [111].

In this work, we use change detection from lidar measurements as a tool for long-term mapping and obstacle detection. Our approach is closest to those of [93] for direct point cloud comparison, and [103] for ray-tracing, which are the relatively simple methods that avoid sensor and environment modeling. Such approaches are sufficient for a working system in our environments of interest and robot platform. However, one should be aware of implicit assumptions about the environment and the sensor characteristics mentioned above.

4.2.2 Map Maintenance

Map maintenance in changing environments has been an active area of research for decades. A particularly long-standing topic is detecting and removing dynamic objects from the map. The popular occupancy grid algorithm [99] can implicitly tolerate a moderate number of dynamic objects due to its additive map updating rule. A cell is considered free as long as it is more often observed free. Similarly, Hahnel et al. [109] proposed an Expectation-Maximization approach to differentiate between static and dynamic cells in grid mapping. They showed that the closed-form result could

be obtained directly from counting occupied and free observations. An alternative approach was to track moving objects from feature correspondences and exclude them from map building [112]. The main ideas behind these approaches are still seen in recent publications, although the details on map representation and moving object detection are very different.

Previous works most similar to ours in this aspect are [104] and [105]. We represent the map by sparse point clouds and identify dynamic objects from scan-to-map inconsistencies using ray-tracing. Moreover, both [104] and our method compare observations from multiple experiences to detect short-term static objects. The main difference is that [104] lacks the notion of dealing with long-term environmental changes. It keeps both static and dynamic points in a single map and assumes that a point can never be static once confirmed to be dynamic. In contrast, our method immediately discards the classified dynamic points from each experience. Static points are added to the map only if observed in multiple experiences and thereby considered long-term static.

Handling long-term environmental changes is another topic that has received a fair amount of attention. Existing work often assumes that the environment is observed intermittently over an extended period and focuses on dealing with changes between observations. Dynamic objects are assumed to be removed in advance using the methods above. Current approaches can be divided into two classes. One constantly updates a single map to stay updated with the current environment [113, 114, 115]. The other assumes the environment has multiple configurations and thus keeps multiple maps to match them individually [116, 117, 39].

Our multi-experience mapping algorithm (more details below in Section 4.3.4) falls into the first class and shares similarities with the Meta-Room algorithm presented by Ambrus et al. [115]. Both algorithms aim at mapping the current long-term static objects of the environment. The main difference lies in the assumption about the presence of unobserved objects due to occlusions. In [115], unobserved objects are always assumed to be present and not removed from the map. However, in our method, unobserved objects are treated the same as disappeared objects. As a result, their algorithm may incorrectly keep disappeared objects while our algorithm can incorrectly remove objects still in the environment. We adopt this different assumption to avoid expensive ray-tracing operations to reason about occlusions. Our algorithm is still robust to temporary occlusion because an object must be absent from multiple experiences before it is removed. Furthermore, we argue that if an object is constantly occluded, it should be removed from the map even if it is long-term static because it no longer benefits our use of the map for localization and obstacle detection.

Regarding methods belonging to the second class, an example is the Multi-Experience Localization (MEL) algorithm in VT&R2 [4]. MEL is inspired by the experience-based navigation framework of [39], which can be seen as a generalization of the mapping algorithms in [116] and [117]. The motivation behind this class of methods is that the environment may change too significantly, so generating maps using only its most stable components becomes insufficient. However, it is often a problem for visual maps but not lidar maps. In Chapter 3, we have shown that accurate lidar localization can be achieved across a year without any map updating. Furthermore, lidar localization is also too expensive to run across multiple maps in real-time. We thereby prefer maintaining a single map in this work instead of keeping multiple maps.

It is also worth pointing out some works that attempt to model the environment dynamics with parameters learned from observations [118, 119, 120]. Leveraging such models allows one to produce more compact maps and predict the environment state at a future time, which addresses the latency

problem in the above map updating methods. For example, Krajnik et al. [120] discretize the environment into voxel grids and model the occupancy of each cell by a superposition of periodical functions. Our method also stores past observations, which could be used for model learning. However, it is currently not considered because 1) the amount of data required is prohibitively large, and 2) the underlying model assumptions restrict the type of environments they can be applied to.

4.2.3 Obstacle Detection

Obstacle detection in this work can be related to the problem of unsupervised object discovery from change detection, which is often handled jointly with long-term mapping [121, 115, 122, 123]. Assuming again that the environment is observed intermittently over time, the object of interest are those that appear, disappear or move between observations so that can be extracted using methods from Section 4.2.1. They are excluded from mapping but, in this case, segmented and associated with known object classes. The same approaches can be applied to obstacle detection if one assumes that 1) the static map is always obstacle-free and 2) every obstacle can be observed and cause discrepancies to the map. Segmentation and association are not strictly required because object-level information is not mandatory for local path planning to avoid obstacles.

Our method is also inspired by previous terrain assessment works for VT&R2, which also rely on change detection [5, 6]. In [5], the terrain ahead of the robot is organized into robot-sized patches, each of which is a 2D grid storing the estimated terrain height. These patches are compared with those from previous experiences, and any significant difference causes the respective terrain to be marked untraversable. This approach is improved in [6], with each cell storing a 1D GMM with two Gaussians in the vertical direction. One Gaussian models the terrain height and roughness, and the other addresses noises due to over-hanging vegetations. With this representation, they can adaptively detect small changes in smooth terrain while being robust to noises in rough terrain. Our method learns from [6] to explicitly account for surface roughness in change detection using a Gaussian model, improving the detection of small obstacles in structured environments. However, other than that, our approach is very different because changes are identified pointwise from a live lidar scan, and we do not assume the terrain ahead of the robot to be the only surface of interest.

4.3 Methodology

4.3.1 Assumptions

Before describing our method, we formally state our assumptions about the environment and discuss their necessity and validity.

Assumption 1 *A safe corridor with place-dependent width is specified along all taught paths. Inside this corridor, the terrain formed by long-term static structures is always traversable unless blocked by dynamic or short-term static obstacles.*

Above is the key assumption allowing us to simplify obstacle detection to a change detection problem and enable the robot to avoid obstacles by temporarily deviating from the repeat path. The long-term static structures in the environment can be identified by comparing observations from different times, which is essentially a change detection problem. Given a map that contains only the

long-term static structures, dynamic or short-term static objects cause discrepancies between the live observation and the map, which is again identifiable from change detection. The above assumption ensures that only such objects can cause the terrain to be untraversable, so treating them uniformly as obstacles is sufficient. The safe corridor specification makes this assumption more realistic and defines the maximum path deviation for obstacle avoidance.

Assumption 2 *The environment geometry is well-constrained by the long-term static structures, ensuring sufficiently accurate inter-experience localization in the presence of short-term static and dynamic objects.*

Accurate localization implies proper alignment between the local maps from different experiences, which is a prerequisite for change detection. In Chapter 3, we have shown that lidar localization remains accurate across seasons and is robust to moderate adverse conditions. Previous works have also shown centimeter-level localization error in both structured and unstructured environments [35]. Our method cannot distinguish actual environment changes from those due to point cloud misalignments. However, the above assumption allows us to treat them as noise and ignore them in our change detection algorithm.

Assumption 3 *The environment can be approximated by local planar surfaces with different roughness, and penetrable objects can be safely ignored.*

The assumption about penetrable objects is necessary because our method considers a space in the environment at a time to be either occupied or free without ambiguity. We trade the ability to handle penetrable objects for more reliable detection of solid obstacles, which is more important in our case. The primary type of penetrable objects in our operating environments is vegetation such as tall grass and tree canopy, which can indeed be ignored from mapping and obstacle detection. As mentioned in Section 4.2.1, it is possible to learn a place-dependent sensor model that accounts for penetrable objects [110]. However, such a model cannot distinguish penetrable objects from space frequently occupied by dynamic objects, so is not considered in this work.

The assumption about local planar surfaces is needed because we use point-to-surface distance as a metric to classify changes between two point clouds, and the result is used for map maintenance and obstacle detection (more details in Section 4.3.4). This assumption may sound restrictive. However, since we ignore penetrable objects and allow for a certain degree of surface roughness, we show in our results that this assumption is still reliable even in highly unstructured environments that seem to lack any apparent surface feature.

4.3.2 Pipeline Overview

As shown in Figure 4.1, our data processing pipeline is an extension to the lidar-only pipeline presented in Section 3.3.1 with three new tasks: *Obstacle detection*, *Dynamic Points Removal* and *Multi-Experience Mapping*. The pipeline again leverages VT&R3's pipeline and parallelization scheme (Section 2.4.2). *Obstacle detection* is placed in the *Inter-Experience Localization* module to be performed immediately after *Registration (Localization)* on the same software thread. *Dynamic Points Removal* and *Multi-Experience Mapping* do not have real-time requirements, so they are submitted to the *Asynchronous Task Executor*. A summary of each task is given below.

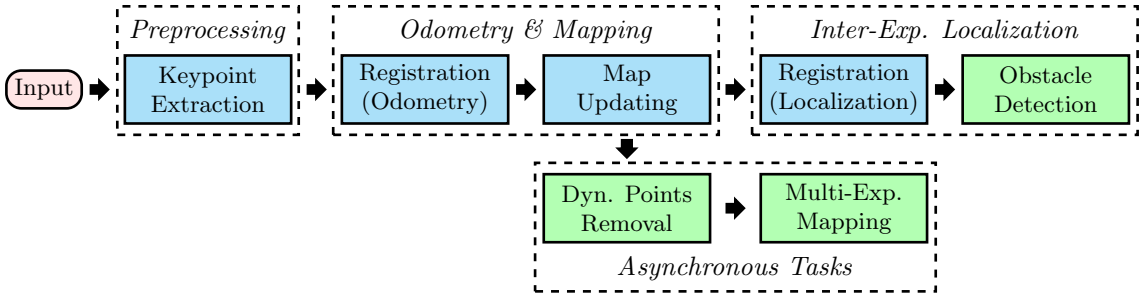


Figure 4.1: The data processing pipeline diagram for lidar map maintenance and obstacle detection, extending the mapping and localization pipline in Figure 3.1. Three new tasks (highlighted in green) are added to the pipeline: dynamic points removal, multi-experience mapping, and obstacle detection. See Section 4.3.2 for details about each task.

Keypoint Extraction: extracts keypoints from raw lidar scans to be used for scan-to-map registration. We use the same strategy presented in Section 3.3.2 without changing any parameters.

Registration (Odometry): stays the same as in Section 3.3.1, which estimates the robot pose with respect to the latest vertex frame via scan-to-map point cloud registration using our motion-compensated odometry algorithm, CT-ICP (Section 3.3.3). Recall that the map herein is a sliding map that is initialized at the start of the current teach/repeat pass and is updated to center at the robot location. A motion-compensated scan is also generated to be used by subsequent tasks.

Map Updating: updates the sliding map by including the motion-compensated live scan while removing points too far away from the robot. The thresholds for vertex and local map creation are changed compared to those in Section 3.3.1. We use a different threshold of 0.3m/10° for vertex creation and 5m/30° for local map creation. Specifically, whenever the robot's pose with respect to the last vertex exceeds the vertex creation threshold, we ask VT&R3 to add a new vertex to the pose graph. Additionally, we store the motion-compensated scan in the new vertex, which will be used to identify dynamic points from the local map (more details in Section 4.3.3). We separately keep track of the robot's motion since the last local map and generate a new one if it exceeds the local map creation threshold. Each newly added vertex will be associated with the latest local map.

Dynamic Points Removal: is performed on every local map produced by *Map Updating*. It detects and removes points from dynamic objects in the local map using a scan-to-map ray-tracing approach presented in Section 4.3.3. The updated local map is sent to *Multi-Experience Mapping*.

Multi-Experience Mapping: uses the output from *Dynamic Points Removal* to initialize and maintain a number of multi-experience local maps from which the points on long-term static objects can be extracted. We present the details of our multi-experience mapping algorithm in Section 4.3.4.

Registration (Localization): also stays the same as in Section 3.3.1, which localizes the robot to the closest vertex frame from the repeat path using the ICP algorithm presented in Section 3.3.5. Note that the local map in this case will have both dynamic and short-term static points removed . The aligned scan is sent to *Obstacle Detection* to label points on obstacles.

Obstacle Detection: performs point-wise obstacle detection from the aligned scan using the method in Section 4.3.5. The output is an annotated scan with points on obstacles distinguished from points on long-term static structures. It can also convert the output to a 2D cost map to be used by VT&R3's TEB local planner. The details will be given in Section 4.3.7.

4.3.3 Dynamic Points Removal

Identifying points on dynamic objects can be achieved by detecting changes across successive lidar scans. We follow previous work by Underwood et al. [103] to use a ray-tracing approach for change detection. The underlying idea is that each scan provides two kinds of information: occupied space where the points are located and free space where the lidar rays pass through. A point measured in one scan is likely to come from a dynamic object if it lies in the free space of other scans. Given the local map produced from *Map Updating*, which contains points accumulated from multiple scans, the task is then to find those points whose position is observed to be free space in some scans viewing the same space and remove them from the map. The detailed steps are as follows.

First, we find the lidar scans that are spatially close to the current local map and from the same experience. It can be done via a Breadth-First Search (BFS) on the pose graph, starting from the closest vertex to the center of the local map and up to a certain depth. Recall that *Registration (Odometry)* produces motion-compensated lidar scans, which will be stored in the pose graph vertices. The lidar scans from all visited vertices are retrieved for ray-tracing.

Then, for each lidar scan, we project it into a frustum grid to encode free space information [107]. A frustum grid is a 2D grid along the ϕ and θ dimensions of the scan's spherical coordinates, and each cell stores the measured range coordinate ρ. The resolution of the grid, $d\phi, d\theta$, is the same as the resolution of the scan in the horizontal and vertical direction, respectively.

After that, we transform the local map into the lidar scan frame, where the transformation is found by compounding the scan/map-to-vertex transformations and the vertex-to-vertex transformations stored in the pose graph. For each point in the transformed local map with its Cartesian coordinate $\mathbf{q} \in \mathbb{R}^3$ and surface normal $\mathbf{n} \in \mathbb{R}^3, \|\mathbf{n}\| = 1$, we consider it as lying in the free space of the lidar scan if the following conditions are met:

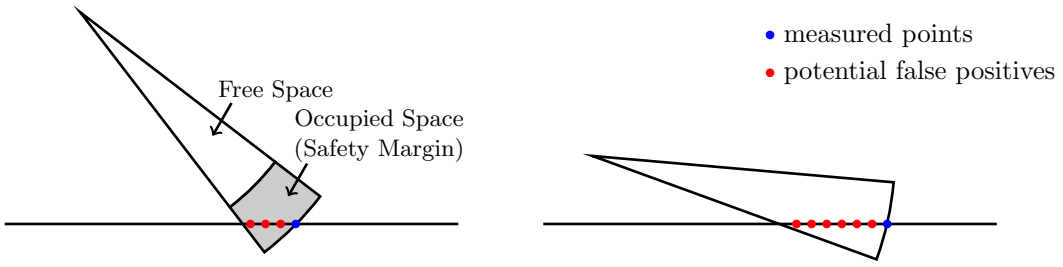
$$\text{Condition 1:} \quad \|\mathbf{q}\| < \rho - \rho \max(d\theta, d\phi)/2, \tag{4.1a}$$

$$\text{Condition 2:} \quad \arccos \left| \mathbf{n} \cdot \frac{\mathbf{q}}{\|\mathbf{q}\|} \right| < \pi/4, \tag{4.1b}$$

where ρ is the corresponding range value given $\mathbf{q}$'s coordinate in the frustum grid. Similar conditions have been used in [107]. In Condition 1, the term $\rho\max(d\theta, d\phi)/2$ is a safety margin to avoid false positives due to the finite resolution of the frustum grid. It is equal to the largest half size of the frustum at range ρ and ensures that points from a planar surface whose incidence angle is approximately less than $\pi/4$ will not be misclassified as free space. Condition 2 also avoids false positives for points with greater incidence angles. 2D examples illustrating the rationale of these two conditions are shown in Figure 4.2.

Condition 2 prevents us from treating the ground as dynamic because the ground is always near-parallel to the lidar rays. However, the downside is that we can no longer correctly classify points from a true dynamic object if it is always observed with a high incidence angle, such as car tops. We consider this a trade-off between false positive and false negative, and Condition 2 favors having fewer false negatives. In our situation, false negatives are less detrimental because they are usually classified as short-term static in *Multi-Experience Mapping* (Section 4.3.4) and thereby still removed from the final map of long-term static structures.

Once the local map is ray-traced against all lidar scans, we count the number of times each point in the map is observed in the free space of a lidar scan. We remove points that lie in the free space



(a) Condition 1 (b) Condition 2

Figure 4.2: This figure illustrates the idea of the two conditions (4.1a) and (4.1b) in preventing false positives in the ray-tracing algorithm using 2D examples. The safety margin in Condition 1 avoids potential false positives due to measurement noises and small incidence angle. Condition 2 conservatively prevents any positive classification when the incidence angle is large.

of at least two scans.

4.3.4 Multi-Experience Mapping

To remove short-term static obstacles and handle long-term changes, we extend our local map from the privileged experience to store observations from additional, non-privileged experiences. Each point in this extended *multi-experience local map* is associated with a bit-vector of maximum size $K = 128$, denoting whether its underlying surface is observed over the last K experiences. Each multi-experience local map is initialized with a regular local map from the teach pass and constantly updated with nearby local maps from the subsequent repeat passes.

Given a newly generated local map $\mathcal{M}$ with dynamic points removed, we first follow the same procedure as in Section 4.3.3 to find nearby multi-experience local maps $\{\widetilde{\mathcal{M}}_i, i = 1, ..., M\}$ and the corresponding transformations for map-to-map alignment, i.e. by performing a BFS on the pose graph and compounding vertex-to-vertex/map transformations.

Then, for each selected multi-experience local map $\widetilde{\mathcal{M}}_i$, if it has not been updated by any local map from the same experience as $\mathcal{M}$ but already contains K experiences, we discard the oldest experience and then remove points that are marked unobserved in the rest $K - 1$ experiences. For each remaining point $\mathbf{p} \in \widetilde{\mathcal{M}}_i$, we check if it is also observed in $\mathcal{M}$ and update its bit-vector accordingly.

Determining whether $\mathbf{p} \in \widetilde{\mathcal{M}}_i$ is observed in $\mathcal{M}$ is essentially a change detection problem. A simple metric to use is $\mathbf{p}$'s distance to the nearest point in $\mathcal{M}$ [93]. However, point-to-point distance is sensitive to point cloud density and cannot detect small changes. When Assumption 3 is true, point-to-surface distance is a better choice, in which case we instead aim to classify whether the underlying surface of $\mathbf{p}$ is observed in $\mathcal{M}$. We also follow [5] to account for the surface roughness in our metric so that we can detect small changes in structured environments while ignoring them in unstructured environments. We discuss our parameterization of a surface first and then the classification metric.

We define the underlying surface of a point $\mathbf{p}$ using three parameters: a point on the surface $\mathbf{c} \in \mathbb{R}^3$, the surface normal $\mathbf{n} \in \mathbb{R}^3$, $\|\mathbf{n}\| = 1$ and a roughness value λ. These parameters are estimated using $\mathbf{p}$ and its neighboring points, where $\mathbf{c}$ is the centroid, and λ is the smallest eigenvalue from PCA with $\mathbf{n}$ being its corresponding eigenvector. With this information, we could consider the

likelihood of a point $\mathbf{q} \in \mathbb{R}^3$ being on the surface of $\mathbf{p}$ as a Gaussian along its normal direction,

$$p(\mathbf{q} \mid \mathbf{c_p}, \mathbf{n_p}, \lambda_\mathbf{p}) = \mathcal{N}\left((\mathbf{q} - \mathbf{c_p}) \cdot \mathbf{n_p} \mid 0, \lambda_\mathbf{p}\right). \tag{4.2}$$

Equation (4.2) could be used directly as a classification metric. However, we notice that it is error-prone when the neighborhood of $\mathbf{p}$ cannot be approximated by a surface. A typical example is when $\mathbf{p}$ comes from vegetation (e.g., tall grass, tree canopy), where the estimated roughness λ is often large, making $\mathbf{p}$ more likely considered as observed in subsequent experiences. This is undesirable because we will end up accumulating points from or near vegetation in the multi-experience local map, which affects our ability to detect obstacles.

To address this issue, we choose to introduce a prior on $\lambda_\mathbf{p}$ to prevent it from getting too large. For simplicity, we use the conjugate prior of a Gaussian with a fixed mean, which is an Inverse-Gamma distribution

$$p(\lambda_\mathbf{p} \mid \alpha_0, \beta_0) = \text{Inv-Gamma}(\lambda_\mathbf{p} \mid \alpha_0, \beta_0), \tag{4.3}$$

where α_0 and β_0 are tunable shape and scale parameter, respectively. Let $P = \{\mathbf{p}_i, i = 1, ..., N\}$ be the points used to estimate the surface parameters of $\mathbf{p}$, the posterior is

$$\begin{aligned} p(\lambda_\mathbf{p} \mid \alpha_0, \beta_0, P) &= \text{Inv-Gamma}(\lambda_\mathbf{p} \mid \alpha_n, \beta_n), & (4.4)\\ \text{where} \quad \alpha_n &= \alpha_0 + \frac{N}{2}, & (4.5)\\ \beta_n &= \beta_0 + \frac{1}{2}\sum_{i=1}^{N} |(\mathbf{p}_i - \mathbf{c_p}) \cdot \mathbf{n_p}|^2. & (4.6) \end{aligned}$$

From Equation (4.5) and (4.6), we can see that β_0 and α_0 essentially play the role of a pseudo-measurement and its weight compared to real measurements. Also note that $\mathbf{c_p}$ and $\mathbf{n_p}$ are still estimated directly from P.

Given the posterior of $\lambda_\mathbf{p}$ from Equation (4.4), we can compute the probablity of a point $\mathbf{q}$ being on the underlying surface of $\mathbf{p}$ using the posterior predictive distribution, which is a t-distribution [124]

$$\begin{aligned} p(\mathbf{q} \mid \alpha_0, \beta_0, P) &= \int p(\mathbf{q} \mid \lambda_\mathbf{p})\, p(\lambda_\mathbf{p} \mid \alpha_0, \beta_0, P)\, d\lambda_\mathbf{p} & (4.7)\\ &= t\left(\frac{\alpha_n(\mathbf{q} - \mathbf{c_p}) \cdot \mathbf{n_p}}{\beta_n} \;\middle|\; 2\alpha_n\right). & (4.8) \end{aligned}$$

where $2\alpha_n$ is the degree of freedom.

Using Equation (4.8), we consider the underlying surface of $\mathbf{p}$ as observed in $\mathcal{M}$ if the following condition is satisfied:

$$\exists \mathbf{q} \in \mathcal{M}, \|\mathbf{q} - \mathbf{p}\| < r \text{ such that } p(\mathbf{q} \mid \alpha_0, \beta_0, P) > \gamma. \tag{4.9}$$

where r is the neighborhood search radius and γ is a classification threshold.

Finally, we merge the local map $\mathcal{M}$ into each of the multi-experience local map $\widetilde{\mathcal{M}}_i$ to add points observed for the first time. Voxel-based downsampling is performed afterward to keep the size of the map bounded. Importantly, if a point from $\mathcal{M}$ and an existing point lies in the same voxel, the

existing point is kept to prevent the map from drifting over experiences due to sensor noises and alignment errors.

Overall, our multi-experience local map keeps up with environment changes by constantly getting updated by the latest observations. To extract points from long-term static structures, we simply count the number of experiences each point is observed and filter out those below a certain threshold.

4.3.5 Obstacle Detection

With Assumption 1, obstacle detection simplifies to a change detection problem between the environment's long-term static structures and the current observation. The former can be obtained from the multi-experience local map, and the latter comes from the live lidar scan. If a point from the live scan does not belong to any mapped structure, it is likely from an obstacle.

As shown in the pipeline diagram (Figure 4.1), before *Obstacle Detection*, each lidar scan $\mathcal{S}$ has passed through *Registration (Localization)* where it is aligned to the nearest local map $\mathcal{M}$, which in this case contains only the long-term static structures. We perform change detection between $\mathcal{S}$ and $\mathcal{M}$ using a similar condition as (4.9). Specifically, each point $\mathbf{q} \in \mathcal{S}$ may be from an obstacle if the following condition is true:

$$\exists \mathbf{p} \in \mathcal{M}, \|\mathbf{q} - \mathbf{p}\| < r \text{ such that } p(\mathbf{q} \mid \alpha_0, \beta_0, P) > \gamma. \tag{4.10}$$

The same parameters $\alpha_0, \beta_0, r, \gamma$ in (4.9) are used. Note that P is the set of points for estimating the surface parameters of $\mathbf{p}$.

After all points in $\mathcal{S}$ have been classified, we remove isolated points classified as *positive* (i.e., coming from an obstacle) because they are likely noise or from penetrable objects. For each *positive* point $\mathbf{q}'$, we compute a *support* of this classification using $\mathbf{q}'$'s neighboring points as

$$\mathit{support}(\mathbf{q}') = \sum_{\mathbf{p}_i \in \mathcal{B}_r(\mathbf{q}')} \exp\left(-\|\mathbf{q}' - \mathbf{p}_i\|^2\right) \times \mathbb{1}(\mathbf{p}_i = \mathit{positive}), \tag{4.11}$$

where $\mathcal{B}_r(\mathbf{q}')$ is the spherical neighborhood of $\mathbf{q}'$ with radius r (same r as in (4.10)). $\mathbb{1}(\mathbf{p}_i = \mathit{positive})$ is a binary indicator function that evaluates to 1 if $\mathbf{p}_i$ is *positive* and 0 otherwise. Points with a support value below a threshold λ_f have their classification flipped to *negative*. We stress that, using Equation (4.11), neighboring points considered *negative* do not negatively contribute to the calculation of the *support*. This is necessary to prevent points near the intersection of obstacles and long-term static structures from being misclassified as *negative*.

4.3.6 Safe Corridor Specification and Visualization

Due to Assumption 1, it is necessary to develop a method such that the width of the safe corridor can be easily specified during online operation. To this end, we extend the VT&R3 GUI with a slider at the bottom for specifying the safe corridor width, as shown in Figure 4.3. The slider's value can be set as either continuous to reflect the width in meters directly or discrete to indicate different path types with predefined corridor widths. During the teach pass, the user can adjust the safe corridor width at the current location by simply dragging the cursor on the slider. The selected value is then sent to the navigation system and associated with newly created vertices. The network

Figure 4.3: This figure shows the updated VT&R3 GUI for specifying and visualizing the safe corridor information. The slider at the bottom is used to specify the safe corridor width during each teach pass, and the paths are colored accordingly.

of paths displayed in the GUI is also colored accordingly to reflect the safe corridor width.

4.3.7 Local Planning Integration

In VT&R2, the path tracker commands the robot to follow the repeat path as closely as possible and will stop the robot for human intervention if the terrain ahead looks significantly different from past experiences [6]. However, with the *safe corridor* assumption (Assumption 1), the robot may occasionally leave the repeat path to avoid obstacles as long as it stays inside the safe corridor. In this section, we propose a way to achieve this behavior with VT&R3's default local planning solution.

Recall from Section 2.6 that VT&R3 adopts the TEB local planner as its default local planning algorithm. This planner essentially performs nonlinear MPC via online optimization of a trajectory in a three-dimensional spatiotemporal space (x, y, t). Trajectory optimization minimizes a cost function that encodes the desired behavior. Normally, the cost function comprises terms that 1) minimize the distance to waypoints along the repeat path, 2) minimize time to the last waypoint in the planning horizon, and 3) encourage compliance with certain smoothness and speed constraints. To enable obstacle avoidance, we convert the obstacle detection result and safe corridor information into two 2D cost maps and feed them into the cost function as additional terms.

Safe Corridor Cost Map ($\mathcal{C}_{SC}$): the safe corridor cost map is centered at the origin of the current localization frame (i.e. the closest vertex frame on the repeat path). The cell value is defined to be

$$\mathcal{C}_{SC}(i,j) = \min\left(1, \max\left(0, 1 - \frac{d_s - d_1 - d(i,j)}{d_0}\right)\right), \tag{4.12}$$

where $d(i,j)$ is the distance from the center of cell (i,j) to the repeat path, d_s is the specified corridor width. d_1 and d_0 are tuned minimum distance to the corridor boundary and influence distance, respectively.

Obstacle Cost Map ($\mathcal{C}_{Ob}$): for the obstacle cost map, we project *positive* points in the lidar scan onto the horizontal plane of the current localization frame. Then, we construct the cost map with its center being the localization frame origin. The value in each cell is set to

$$\mathcal{C}_{Ob}(i,j) = \min\left(1, \max\left(0, 1 - \frac{d(i,j) - d_1}{d_0}\right)\right), \tag{4.13}$$

with $d(i,j)$ being the distance from the center of cell (i,j) to the closest point in the projected lidar scan with only *positive* points. d_1 and d_0 are defined similarly as in (4.12).

We emphasize using two cost maps instead of merging them because the values are updated at different rates. The obstacle cost map is updated per lidar scan, but the safe corridor cost map is only updated when switching to a new localization frame. Both cost maps are incorporated by the TEB planner into its trajectory cost function. During optimization, the cost value from $\mathcal{C}_{SC}/\mathcal{C}_{Ob}$ at a trajectory pose is defined as a bilinear interpolation of the nearby cell values. The gradient is computed using a finite-difference approximation by the g2o library [125], which is the backend optimizer of the TEB planner.

The TEB planner additionally allows for simultaneous optimization of multiple trajectories from different homotopy classes to overcome sub-optimality issues due to trajectory initialization. We can enable this feature following the idea from Thomas et al. [126] by generating fake point obstacles at local maxima in $\mathcal{C}_{Ob}$ for homotopy class discovery.

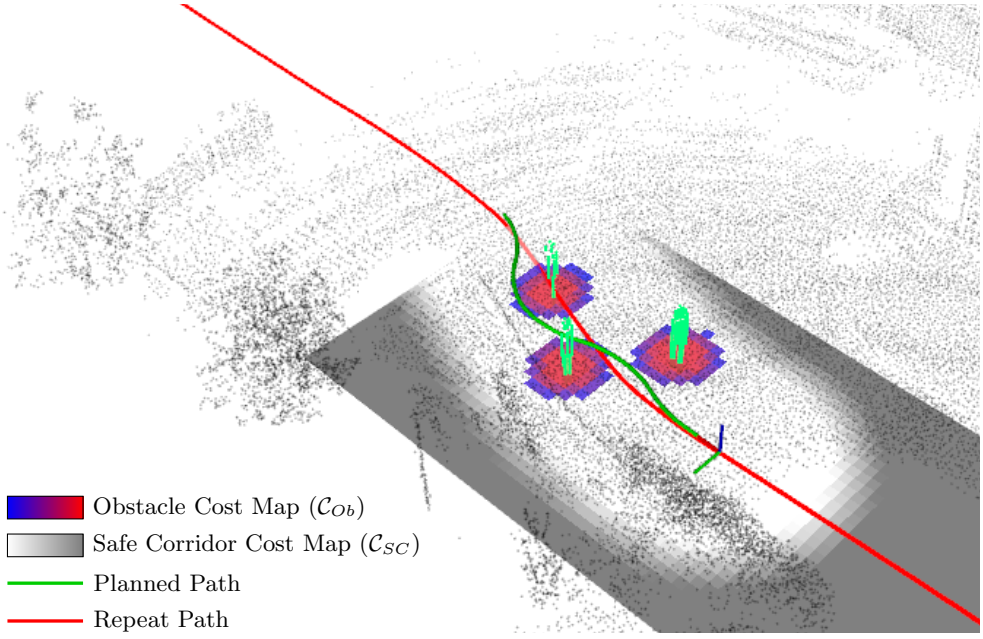


Figure 4.4: This figure demonstrates the proposed method of incorporating obstacle and safe corridor information into VT&R3 local planning. The planned path locally deviates from the repeat path to avoid obstacles while staying inside the safe corridor.

Figure 4.4 shows an example of what the two cost maps may look like and how they affect the

planned trajectory. Note that, although we have implemented this obstacle-avoiding local planning feature in VT&R3, a thorough evaluation is outside the scope of this chapter and left for future work.

4.4 Evaluation

4.4.1 Dataset

Figure 4.5: This picture shows our Clearpath Grizzly UGV used for data collection, equipped with a Waymo Laser Bear Honeycomb lidar.

The dataset for evaluation is collected at UTIAS using a Clearpath Grizzly UGV equipped with a Waymo Laser Bear Honeycomb lidar. Figure 4.5 shows a picture of our data collection vehicle. It is typically driven at 0.5-1m/s during data collection. The Honeycomb lidar has a 210° horizontal FOV and a 95° vertical FOV (21° up, 74° down). The vertical and horizontal resolutions are set to 0.76° and 1.2°, respectively. It spins at 5Hz and produces up to 40k points with a range up to 50m per revolution. Since the lidar fires two laser beams on opposite sides of the sensor, each revolution covers its FOV twice.

The dataset consists of repeated traversals of three different routes at the UTIAS campus, each representing a typical environment we are interested in and is depicted in Figure 4.6. We name the three routes *Parking Lot Route*, *MarsDome Route*, and *Grove Route*, respectively, and the characteristics of each route are as follows:

Parking Lot Route: is a 270m route at the UTIAS parking lot. The route is 100 percent on-road so that the ground is always level and smooth, which is easy for obstacle detection. However, the parking lot is an ideal environment for evaluating our long-term map maintenance algorithm since it presents the two types of obstacles we aim to detect and remove from the map: dynamic obstacles, which are pedestrians and moving vehicles, and short-term static obstacles, which are the parked cars that may appear or disappear between traversals.

MarsDome Route: is a 70m route inside the UTIAS MarsDome. The MarsDome is a 40-meter diameter dome filled with sand, gravel, and rocks to form hilly features. It is used to examine our ob-

stacle detection method's ability to handle slopped ground surfaces, which is typical in unstructured environments.

Grove Route: is a 330m route in a grove to the south of the UTIAS campus. The environment has no artificial structures but only trees. The ground is rougher than that in the parking lot and MarsDome and is covered by short grass, resulting in the most challenging condition for obstacle detection. The tree canopy is a typical penetrable structure we wish to filter out from obstacle detection.

From each route, we intermittently collected many sequences of data in December 2021 and January 2022.

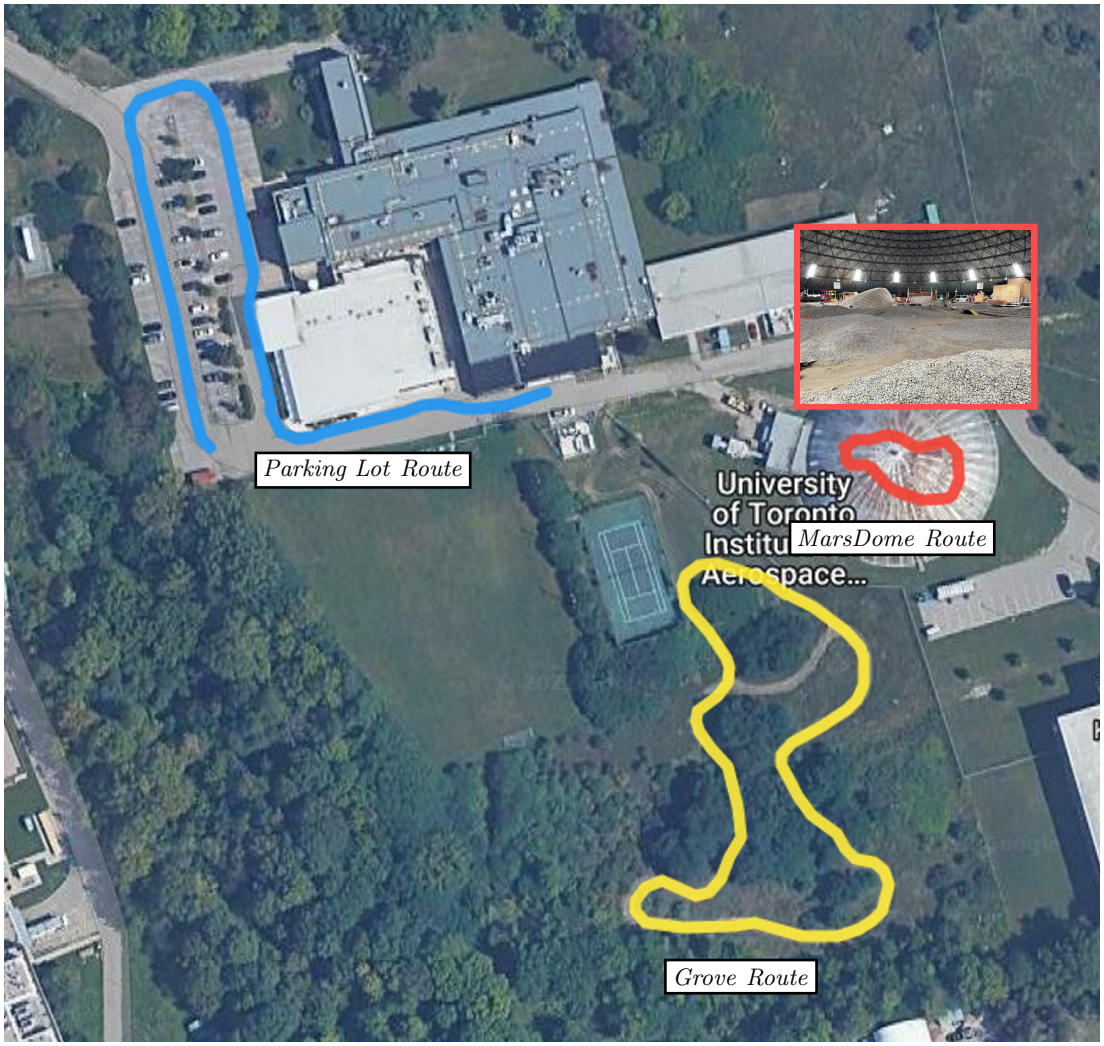


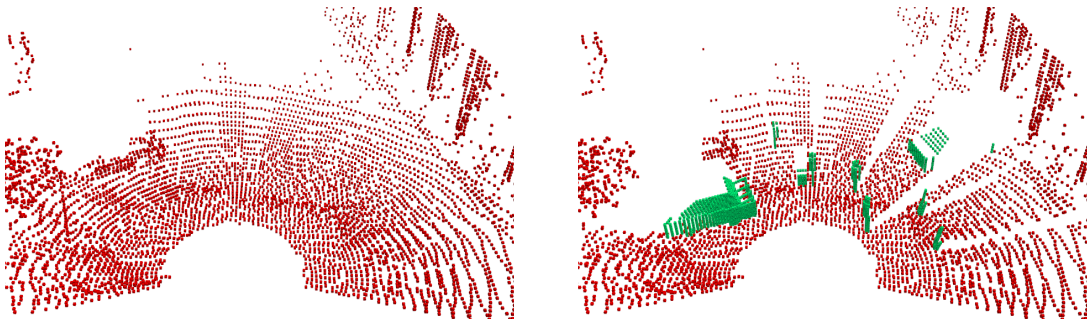
Figure 4.6: This figure shows the three data collection routes plotted on top of a satellite image and a picture taken inside the MarsDome.

4.4.2 Evaluation Strategy and Metrics

The main difficulty of using a real-world dataset for evaluation is that there is no direct access to ground truth information about whether a point in the lidar scan belongs to an obstacle or not.

Previous works typically resort to hand-labeling the ground truth, which inevitably introduces biases, especially in unstructured environments where many ambiguities exist. Another issue with our dataset is the lack of actual obstacles in the environment of *MarsDome Route* and *Grove Route*. In such environments, the usual obstacles of interest, apart from humans, are wild animals, large rocks, and fallen trees. However, their presence during data collection is rare unless manually introduced to the scene. Adding and removing such obstacles by hand is unfortunately too inefficient to handle the large number of obstacles we would like for evaluation. For example, in [6], approximately every 20 meters along the path has only one obstacle. In contrast, we would like 10-20 actual obstacles in each lidar scan to have a reasonable ratio of *positive* and *negative* points. Although we could evaluate our approach in a simulated environment, it does not capture all the real-world complexities, so it cannot prove our approach's practicality to be used by VT&R3.

To address the problem of missing ground truth labels and lacking obstacles, we adopt a strategy that mixes real-world structures with simulated obstacles. Specifically, we generate and place many virtual obstacles (e.g., humans, cars, rocks) at various locations in the environment. These obstacles are represented by CAD models approximating their shape. During pipeline execution, localization and mapping are performed with the original lidar scans from the dataset. However, before passing each scan for obstacle detection, these virtual obstacles are synthesized into the scan based on the current robot location. Points on obstacles are obtained via ray-tracing from the lidar origin according to the sensor resolution, and points behind obstacles are removed. Figure 4.7 shows an example scan with and without adding virtual obstacles. With these fake obstacles and by picking sequences from our dataset with no real obstacles, which is true for many sequences in the dataset, ground-truth labels can be obtained freely. We can then evaluate our obstacle detection algorithm using standard precision and recall metrics with the ground-truth labels[1]. Note that the positive class, in our case, are points belonging to obstacles.



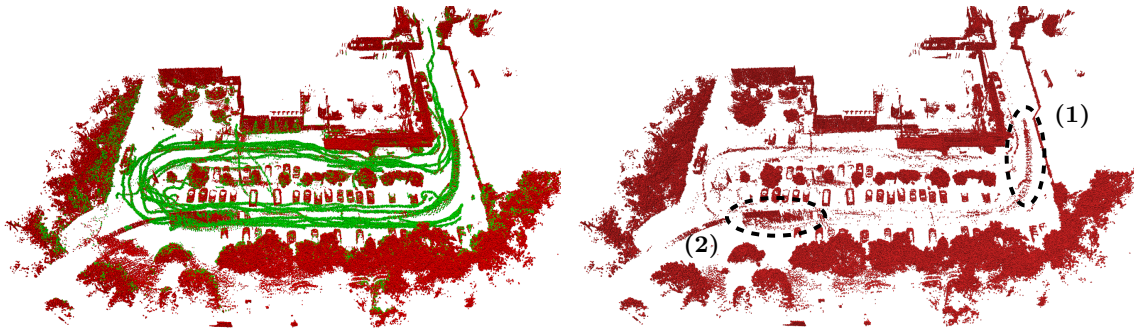
(a) Original Scan (b) Modified Scan with Fake Obstacles

Figure 4.7: This figure shows an example of synthesizing fake obstacles into a real lidar scan. (a) is the original scan. (b) is the scan with points on fake obstacles added and points behind fake obstacles removed. The fake obstacles include humans, cars, cubic boxes, and pylons.

[1]For map maintenance, we perform a qualitative-only evaluation because similar techniques have been used in other contexts and evaluated extensively in simulated and real environments [115]. Our evaluation herein is intended to demonstrate their applicability to VT&R and performance in the environments of our interest. We perform both quantitative and qualitative evaluations for obstacle detection to demonstrate our method's robustness to environments with sloped, rough ground surfaces and vegetations.

4.4.3 Results

Dynamic Points Removal



(a) Dynamic Points in Green (b) Dynamic Points Removed

Figure 4.8: Both subfigures show a *global map* of the *Parking Lot Route* aggregated from local maps of 25 teach/repeat passes, providing the qualitative result of dynamic points removal. The map in (a) has dynamic points highlighted in green, and the map (b) has dynamic points removed. We mark two areas in (a) where our method results in false negatives due to (1) high incidence angle and (2) non-detectable motion.

We evaluate our dynamic points removal approach (Section 4.3.3) using 25 sequences of data from the *Parking Lot Route* with one used as the teach sequence and others being repeat sequences. Figure 4.8 shows our method's classification between dynamic and static (both short-term and long-term) points. For better visualization, local maps from all locations and sequences are transformed into a common frame and aggregated to form a *global map*. Observations on the ground are also removed by keeping only points above the robot chassis.

Although having no ground-truth labels, we can infer them visually according to the points' location and pattern. Points on the road that form trails are clear from pedestrians and moving vehicles and thereby dynamic. The rest of the points tend to be static except those on tree canopy, which are ambiguous.

From Figure 4.8a, we see that most dynamic points are correctly classified, although with a fair amount of false negatives (i.e., dynamic points that are incorrectly classified as static). In contrast, there are few misclassifications of static points from buildings or parked cars. The results suggest that our algorithm trades false negatives for fewer false positives, which is intended as discussed in Section 4.3.3.

In Figure 4.8b, we mark misclassified points at two locations that correspond to typical failure modes of ray-tracing: high incidence angle and non-detectable motion, respectively. Points from location (1) come from the top of a car whose surface normal is nearly parallel to the laser rays. Points from location (2) are from a long truck that moves in parallel with the robot in that area so that the space on one side of the truck is never seen as free.

Multi Experience Mapping

Figure 4.9 shows the resulting classification between short-term and long-term static points by comparing local maps from multiple experiences using the method from Section 4.3.4. The plot is

(a) Short-Term Static Points in Green (b) Short-Term Static Points Removed

Figure 4.9: This figure resembles Figure 4.8, providing the qualitative result of short-term static points removal via multi-experience mapping. The map in (a) has short-term static points highlighted in green, and the map in (b) has short-term static points removed.

generated similarly as in Figure 4.8 by aggregating the multi-experience local maps from all locations and removing points on the ground. The same 25 sequences of data as above are used for evaluation.

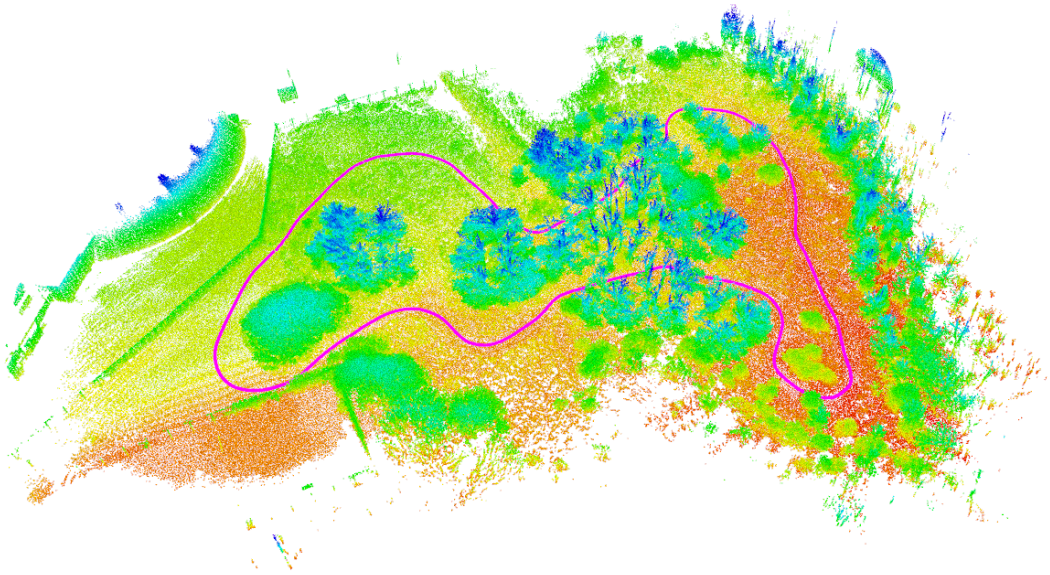
We treat points observed in less than 20 out of 25 experiences as short-term static points, resulting in a perfect classification of parked cars that are not present in all experiences. Furthermore, previously misclassified dynamic points are also treated as short-term static since they rarely appear in every experience. This behavior is favorable because it prevents such points from polluting the final map for localization and obstacle detection. It is also why we can afford a certain amount of false negatives from ray-tracing. However, we stress that it does not entirely preclude the necessity of attempting to remove dynamic points beforehand. In highly dynamic environments, some spaces may be frequently occupied by dynamic obstacles and contain points in all experiences. In this case, such points may be considered long-term static if we count their presence in different experiences.

The ground-truth label of points on tree canopy is again ambiguous, and their classification result from our method is difficult to characterize. The tree canopy is a typical structure that does not conform to our Assumption 3 since it is penetrable and cannot be approximated by local surfaces. Roughly, we see from Figure 4.9a that more points on the tree canopy seem to be classified as short-term static than long-term static. However, we posit that most positives are false due to our keypoint extraction strategy and poor surface normal estimates. We could have completely removed points on the tree canopy by simply clustering the classified positives. However, we observed that it sometimes leads to a degradation of accuracy and a higher chance of failure in localization, suggesting that the distribution of points on the tree canopy still provides some useful geometry information.

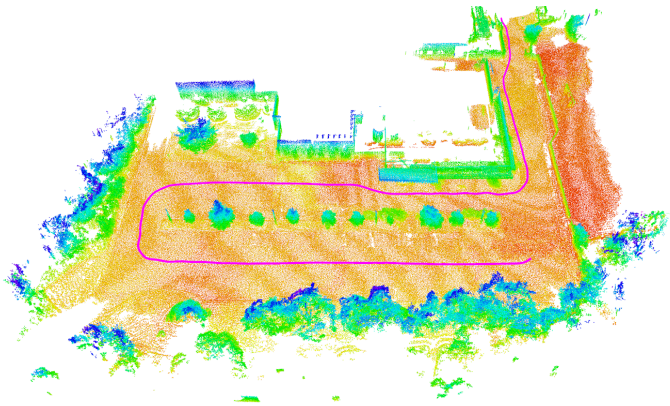
Figure 4.10 shows the aggregated long-term static global map with ground included and points colored according to their heights. Overall, we obtain a high density of points on all of the apparent long-term static structures, such as the ground and buildings. The intactness of the ground near the taught path is especially important because it is always observed in the lidar scan, which will be classified as an obstacle otherwise.

Obstacle Detection

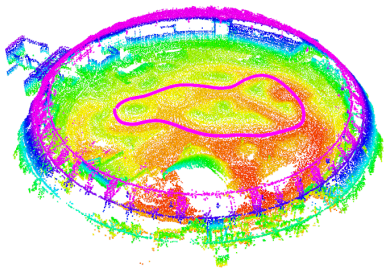
Figure 4.11 shows the spread of the ground roughness and slope along the three routes as a quantitative reflection of their difficulty for obstacle detection. The values are obtained from the underlying



(a) *Grove Route*



(b) *Parking Lot Route*



(c) *MarsDome Route*

Figure 4.10: This figure shows the final long-term static local maps of all three environments in our dataset, aggregated in a global frame. Points are colored by height, and the path is drawn in pink. These maps capture the observed long-term static structures in the environment. The map of *Parking Lot Route* is built from 25 sequences, and the other two maps are built from 5 sequences.

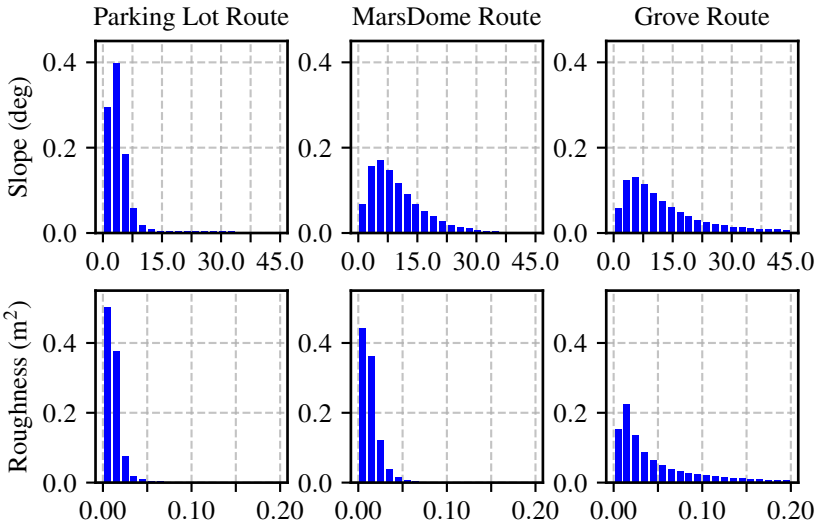


Figure 4.11: These histograms show the spread of the slope and roughness of the ground in the three environments for evaluation, as a quantitative reflection of the environments' difficulty for obstacle detection.

surface of each point close to the ground in the global maps from Figure 4.10. Slope is the angle between the surface normal and the horizontal plane defined by its closest vertex frame, and roughness is the variance of the point's neighbors along the surface normal. Comparing the *MarsDome Route* to the *Parking Lot Route*, the hilly feature in the former causes a higher variation in slope but a negligible difference in roughness. The roughness of the *Grove Route* is most spread out, and its variation in slope is comparable to that of the *MarsDome Route*. Moreover, note that the *Grove Route* is also populated by high vegetation, further complicating the obstacle detection conditions.

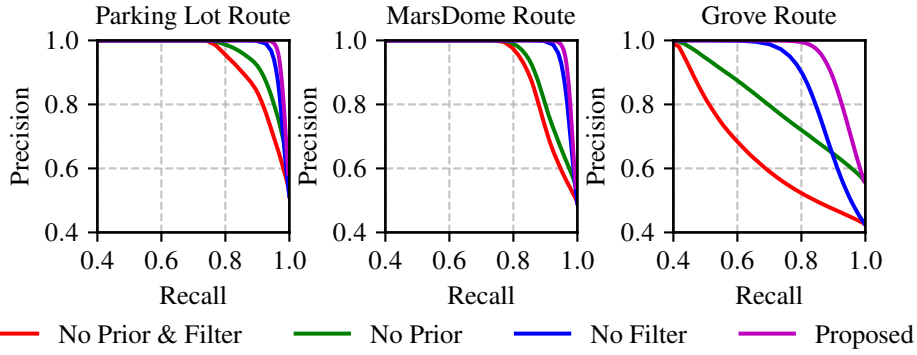


Figure 4.12: The precision-recall curve of the proposed obstacle detection method and its variants. The proposed method achieves the best performance in all three environments.

We evaluate our obstacle detection method (Section 4.3.5) using the strategy and metrics presented in Section 4.4.2. Five sequences from each route that contain no real obstacles are used for evaluation. We use four sequences to construct the multi-experience local maps and perform obstacle detection on the last sequence with fake obstacles added.

Figure 4.12 shows the resulting precision-recall curves of our method when varying the classification threshold γ in Equation (4.10), and Table 4.1 shows the corresponding precision and recall

Table 4.1: Obstacle Detection Results: Precision, Recall and F-Score

	Parking Lot Route			**MarsDome Route**			**Grove Route**		
	Prec.	Recall	F-Score	Prec.	Recall	F-Score	Prec.	Recall	F-Score
No Prior & Filter	0.90	0.84	0.87	0.95	0.81	0.87	0.62	0.65	0.64
No Prior	0.93	0.88	0.91	0.95	0.83	0.89	0.69	0.82	0.75
No Filter	0.99	0.92	0.95	0.98	0.92	0.95	0.92	0.78	0.84
Proposed	0.98	0.95	**0.97**	0.98	0.94	**0.96**	0.95	0.86	**0.90**

values that give the highest f-score. As for comparison, we provide results that do not use a prior, filter, or both. No prior refers to replacing $p(\mathbf{q} \mid \alpha_0, \beta_0, P)$ in our condition (4.10) with $p(\mathbf{q} \mid \mathbf{c_p}, \mathbf{n_p}, \lambda_\mathbf{p})$ (see Equation (4.2)) for classification. We further provide qualitative results in Figure 4.13, which shows a lidar scan from a *Grove Route* sequence after obstacle detection using our method and its no prior or filter variants, comparing to ground truth labels.

In all three environments, our method achieves an f-score no less than 0.90 with both precision and recall values higher than 0.85. There is little difference between the result from the *Parking Lot Route* and that from the *MarsDome Route*. It is expected because our method does not have any requirement or assumption on the overall planarity of the ground. The performance does deteriorate in the *Grove Route* due to vegetations that inject more noise to our surface parameter estimates. Adding a prior to surface roughness significantly improves the performance in all scenarios. As mentioned in Section 4.3.4, the prior can help avoid false negatives when the obstacle is near vegetation or other objects that cannot be approximated by local surfaces. An example can be seen in Figure 4.13d where the upper half of two fake humans in the scan is incorrectly classified as *negative* because the points are associated to over-hanging vegetations that give erroneously high roughness estimates. Applying a filter offers the most notable improvement in the *Grove Route*, where most outliers belong to vegetation, as shown by the difference in the precision-recall curves and Figure 4.13e.

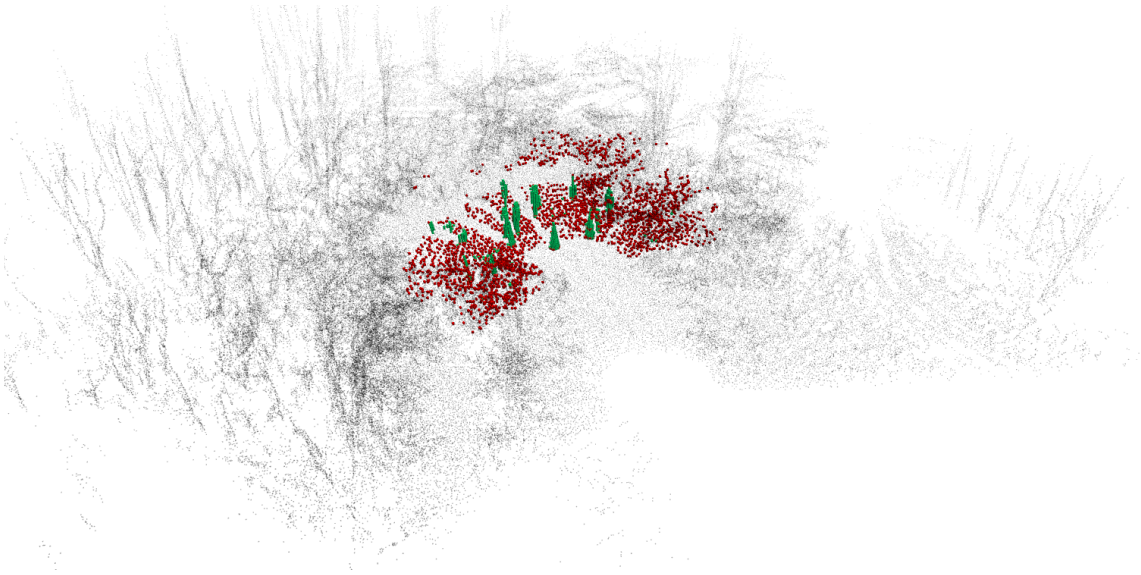
Finally, Figure 4.14 shows an example of detecting real obstacles using our method during VT&R3 online operation. Three traffic barrels were placed in front of the robot and correctly detected as obstacles in the lidar scan. Points from the ground, trees and tall grass surrounding the robot were labeled *negative* as expected.

Computation Requirements

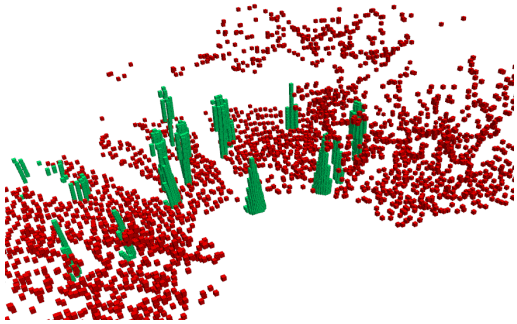
Table 4.2: Pipeline Computation Requirements

Task	**ΔT (ms)**
Keypoint Extraction	33
Registration (Odometry)	152
Map Updating	20
Registration (Localization)	128
Obstacle Detection	10
Dynamic Points Removal	329
Multi-Experience Mapping	14

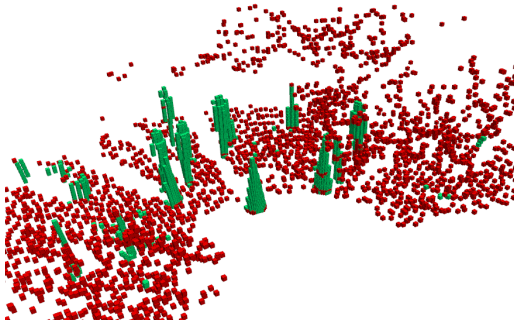
In Table 4.2, we show the computation requirements of each task in the pipeline diagram (Figure 4.1). $\mathbf{\Delta T}$ is the average running time per execution over an entire sequence of data from the *Parking Lot Route*. We used a Lenovo P53 laptop with Intel(R) Core(TM) i7-9750H CPU @ 2.60GHz



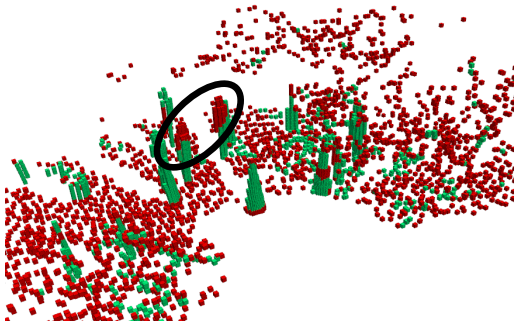
(a) Labeled Scan & Local Map



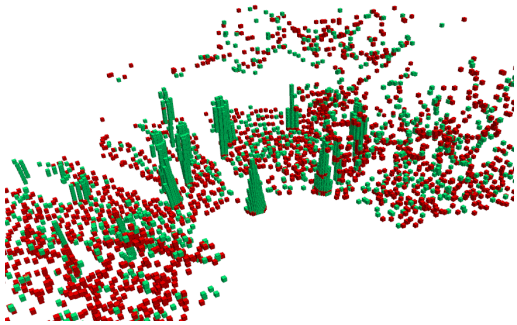
(b) Ground Truth



(c) Proposed



(d) No Prior



(e) No Filter

Figure 4.13: This figure provides qualitative results of the proposed obstacle detection method and its variants on fake obstacles. (a) displays the labeled scan and its aligned local map for reference. (b) shows the ground-truth labels for comparison. (c), (d) and (e) are the results of the proposed method and when no prior or filter is applied, respectively. In (d), points in the circled area are labeled *negative* because they are incorrectly considered part of the overhanging vegetation.

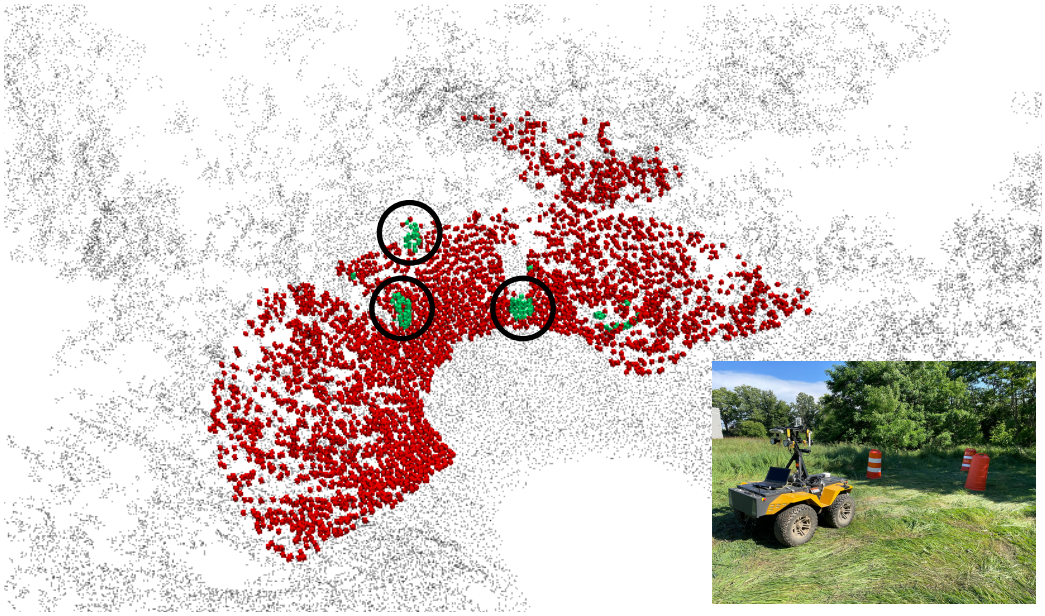


Figure 4.14: This figure shows qualitative results of online real obstacle detection using the proposed method. Green points in the circled area are from the traffic barrels in front of the robot, introduced as obstacles.

and 32GB of memory. The pipeline relies on VT&R3's parallelization scheme to run at frame rate (5Hz). The bottleneck is the *Odometry & Mapping* module, where *Registration (Odometry)* takes 152ms, and *Map Updating* takes 20ms. Note that *Dynamic Points Removal* is an asynchronous task executed once per each local map instead of each lidar scan, so it does not have to run at the frame rate.

4.5 Summary

In this work, we looked into the problem of safe lidar-based teach and repeat navigation in changing environments where obstacles may appear on the taught paths. We assumed that only dynamic and short-term static objects in the environment could be obstacles and developed methods that remove such objects from local maps while detecting them from live lidar scans. Our methods rely on simple change detection algorithms that work without requiring prior knowledge of these objects.

For map maintenance, we provided qualitative results on a parking lot dataset. We showed that both dynamic objects (e.g., moving cars, pedestrians) and short-term static objects (e.g., parked cars) were cleanly removed, while the long-term static objects (e.g., buildings, ground plane) were kept intact. For obstacle detection, we performed both qualitative and quantitative evaluations using real-world data from multiple environments with fake obstacles inserted. Our method achieved high precision and recall in both structured and unstructured environments and was particularly robust to noisy measurements from vegetations (given that we do not treat vegetations as obstacles). Qualitative result of online, real obstacle detection was also reported. The full pipeline was able to process data from a Honeycomb Lidar at frame rate (5Hz) using VT&R3's parallelization strategy.

We additionally proposed to specify a *safe corridor* along taught paths within which the robot could deviate from the repeat path to avoid obstacles. We presented an idea of converting both

the safe corridor and obstacle information into 2D cost maps and passing them to VT&R3's default TEB local planner to obtain a safe, obstacle-free trajectory. We reserve a thorough evaluation of this idea for future work.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

The motivation behind this thesis is the desire to bring the success of VT&R to not just mobile robots using a stereo camera as the primary sensor but also those equipped with other sensors, such as lidar, radar, GPS, or a combination of them. Designing VT&R to work with different sensors allows us to exploit the strength of each sensor, enjoy the power of sensor fusion, and prepare the system for more applications.

The primary goal of this thesis is thereby to re-design VT&R so that it is not tailored to a stereo camera setup, which is where the idea of generalizability lies. To this end, we re-formulated the state estimation, planning, and control problems in VT&R to be more generic and defined interfaces allowing algorithms for different sensors to be added easily. We also re-built the system's sensor/robot communication interfaces using the ROS2 library due to its reliability, efficiency, and popularity. We named our new system Visual Teach & Repeat 3 (VT&R3) and made it an open-source project, hoping to get external contributions and benefit the broader robotics community.

With VT&R3 at hand, we developed and evaluated methods for lidar/radar-based teach and repeat navigation, serving as demonstrations of the improved generalizability of our new system. In Chapter 3, we developed pipelines for lidar/radar topometric mapping and localization using VT&R3's state estimation interface and two algorithms for map building and localization, respectively. Experiments showed that using lidar data for both map building and localization resulted in the best localization accuracy and surprising robustness to adverse weather conditions such as snowstorm. However, pipelines using radar data also demonstrated competitive accuracy and robustness with some advantages in computation and storage, making them viable alternatives.

In Chapter 4, we further extended the lidar topometric mapping and localization pipeline with methods that remove temporary objects (e.g., pedestrians, cars) from the local maps while detecting them as obstacles from live scans. Our proposed methods do not generalize beyond teach and repeat navigation but have the advantage of not requiring prior knowledge of those objects. We validated these methods using real-world data from multiple structured/unstructured environments and demonstrated online operation of the extended pipeline on our testing robot. Using this pipeline for online obstacle avoidance in lidar-based teach and repeat is still a work in progress.

5.2 Future Work

In this section we discuss potential future directions to further improve or extend our work in Chapter 2, 3 and 4.

5.2.1 Visual Teach & Repeat 3

Handling Failures

Recall VT&R3's state transition diagram in Figure 2.11. Apart from the normal transitions accomplishing teach and repeat logics, there are only a few ones triggered by localization failures that result in early exiting from the current teach/repeat pass. However, other failures may occur from, for example, odometry and local planning. Our system is currently missing a systematic way to handle or recover from these failures. Future work should look into this issue and possibly define more states and transitions to address these failures. If state management using a HFSM becomes challenging, we should consider more flexible and scalable solutions, such as Behavior Trees [42].

Topological Localization

Currently, the user must manually specify the robot's closest vertex in the pose graph every time VT&R3 is re-started. However, it can be avoided if vertex-level topological localization is supported. It is easy when a GPS is available, and each vertex is associated with a GPS coordinate. If a camera is used, one can store the captured image to vertices and use any visual place recognition approach to re-locate the robot. Future work should explore these options and define a topological localization interface for VT&R3.

5.2.2 Lidar & Radar Topometric Mapping and Localization

Radar Localization Advantages

Our experiments in Section 3.4 show that lidar localization outperforms radar localization in accuracy on all test sequences, including those with rain and snow. However, according to previous studies on lidar/radar operating conditions [46, 47], it is likely that we have not reached the operational boundary of either lidar or radar. More experiments are needed, possibly with data collected during more adverse conditions. Meanwhile, we believe that radar localization accuracy can be further improved. Future work should explore alternative keypoint extraction algorithms and possibly landmark/correlation-based pipelines.

Computation Requirements

As shown in Table 3.2, our lidar-only pipeline cannot run at the frame rate of a Velodyne Alpha-Prime lidar (10Hz), even though we have exhausted all cores of our CPU. Compared to existing real-time ICP-based lidar odometry & mapping methods, our method is slow because 1) we pass more keypoints to our odometry algorithm, CT-ICP, and 2) the algorithm itself requires more computation for motion compensation (mainly to get the error Jacobians during Gauss-Newton optimization). It should be possible to select fewer keypoints without sacrificing localization accuracy, and computational improvements to CT-ICP should be explored.

5.2.3 Lidar Map Maintenance and Obstacle Detection

Local Planning Integration

Combining the pipeline in Chapter 4 with a local planner and demonstrating obstacle avoidance capability is the most important work to be done. An issue with our idea in Section 4.3.7 is that the cost function encodes objectives inherently conflict with each other. For example, even with no obstacles, the optimized trajectory will cut corners of the repeat path due to balancing trajectory length and distances to waypoints. Future work should look into ways that better reconcile these conflicting objectives. The TEB planner is itself a tentative default for VT&R3 that should be replaced eventually.

General Terrain Assessment

A strong assumption of our method is that only dynamic and short-term objects can be obstacles. Breaking this assumption can easily cause damage to the robot or other assets. Imagine that some long-term static objects happen to block the taught paths. Our method will only detect them during the first few times and eventually treat them as safe to pass through. Avoiding this assumption necessitates the use of more general terrain assessment methods. Future work should explore appropriate methods that can take advantage of the gathered environment information from past teach/repeat passes.

Bibliography

[1] Michael Paton, Kirk MacTavish, Laszlo-Peter Berczi, Sebastian Kai van Es, and Timothy D Barfoot. "I can see for miles and miles: An extended field test of visual teach and repeat 2.0". In: *FSR*. 2018.

[2] M. Paton, F. Pomerleau, and T. D. Barfoot. "Eyes in the Back of Your Head: Robust Visual Teach Repeat Using Multiple Stereo Cameras". In: *CRV*. 2015.

[3] Michael Paton, Kirk MacTavish, Chris J. Ostafew, and Timothy D. Barfoot. "It's Not Easy Seeing Green: Lighting-resistant Stereo Visual Teach Repeat Using Color-Constant Images". In: *ICRA*. 2015.

[4] Michael Paton, Kirk MacTavish, Michael Warren, and Timothy D. Barfoot. "Bridging the appearance gap: Multi-experience localization for long-term visual teach and repeat". In: *IROS*. 2016.

[5] L. Berczi and T. D. Barfoot. "It's like Déjà Vu All over Again: Learning Place-Dependent Terrain Assessment for Visual Teach and Repeat". In: *IROS*. 2016.

[6] Laszlo-Peter Berczi and Timothy D. Barfoot. "Looking High and Low: Learning Place-Dependent Gaussian Mixture Height Models for Terrain Assessment". In: *IROS*. 2017.

[7] Mona Gridseth and Timothy D Barfoot. "Keeping an Eye on Things: Deep Learned Features for Long-Term Visual Localization". In: *RA-L* (2021).

[8] Benjamin Congram and Timothy D. Barfoot. "Relatively Lazy: Indoor-Outdoor Navigation Using Vision and GNSS". In: *arXiv* (2021).

[9] Benjamin Congram and Timothy D. Barfoot. "Experimental Comparison of Visual and Single-Receiver GPS Odometry". In: *arXiv* (2021).

[10] Keenan Burnett, David J Yoon, Yuchen Wu, Andrew Zou Li, Haowei Zhang, Shichen Lu, Jingxing Qian, Wei-Kang Tseng, Andrew Lambert, Keith YK Leung, Angela P Schoellig, and Timothy D Barfoot. "Boreas: A Multi-Season Autonomous Driving Dataset". In: *arXiv* (2022).

[11] Y. Matsumoto, M. Inaba, and H. Inoue. "Visual Navigation Using View-Sequenced Route Representation". In: *ICRA*. 1996.

[12] Andrew Howard. "Multi-Robot Mapping Using Manifold Representations". In: *ICRA*. 2004.

[13] Joshua Marshall and Timothy Barfoot. "Design and Field Testing of an Autonomous Underground Tramming System". In: *FSR* (2007).

[14] Joshua Marshall, Timothy Barfoot, and Johan Larsson. "Autonomous Underground Tramming for Center-Articulated Vehicles". In: *JFR* (2008).

[15] Paul Furgale and Timothy D. Barfoot. "Visual teach and repeat for long-range rover autonomy". In: *JFR* (2010).

[16] Gabe Sibley, Christopher Mei, Ian Reid, and Paul Newman. "Vast-Scale Outdoor Navigation Using Adaptive Relative Bundle Adjustment". In: *IJRR* (2010).

[17] Sebastian K. Van Es and Timothy D. Barfoot. "Being in Two Places at Once: Smooth Visual Path Following on Globally Inconsistent Pose Graphs". In: *CRV*. 2015.

[18] Michael Paton, François Pomerleau, Kirk MacTavish, Chris J. Ostafew, and Timothy D. Barfoot. "Expanding the Limits of Vision-based Localization for Long-term Route-following Autonomy". In: *JFR* (2017).

[19] K. MacTavish, M. Paton, and T. D. Barfoot. "Visual Triage: A Bag-of-Words Experience Selector for Long-Term Visual Route Following". In: *ICRA*. 2017.

[20] Kirk MacTavish, Michael Paton, and Timothy D. Barfoot. "Selective Memory: Recalling Relevant Experience for Long-Term Visual Localization". In: *JFR* (2018).

[21] Chris J. Ostafew, Angela P. Schoellig, Timothy D. Barfoot, and Jack Collier. "Learning-Based Nonlinear Model Predictive Control to Improve Vision-based Mobile Robot Path Tracking: Learning-based Nonlinear Model Predictive Control to Improve Vision-based Mobile Robot Path Tracking". In: *JFR* (2015).

[22] Chris J. Ostafew, Angela P. Schoellig, and Timothy D. Barfoot. "Robust Constrained Learning-based NMPC Enabling Reliable Mobile Robot Path Tracking". In: *IJRR* (2016).

[23] N. Zhang, M. Warren, and T. D. Barfoot. "Learning Place-and-Time-Dependent Binary Descriptors for Long-Term Visual Localization". In: *ICRA*. 2018.

[24] Mona Gridseth and Timothy D Barfoot. "Towards Direct Localization for Visual Teach and Repeat". In: *CRV*. 2019.

[25] Mona Gridseth and Timothy D. Barfoot. "DeepMEL: Compiling Visual Multi-Experience Localization into a Deep Neural Network". In: *ICRA*. 2020.

[26] H. Guo and T. D. Barfoot. "The Robust Canadian Traveler Problem Applied to Robot Routing". In: *ICRA*. 2019.

[27] Christopher D. McKinnon and Angela P. Schoellig. "Experience-Based Model Selection to Enable Long-Term, Safe Control for Repetitive Tasks Under Changing Conditions". In: *IROS*. 2018.

[28] Christopher D. McKinnon and Angela P. Schoellig. "Learn Fast, Forget Slow: Safe Predictive Learning Control for Systems With Unknown and Changing Dynamics Performing Repetitive Tasks". In: *RA-L* (2019).

[29] Lee Clement, Jonathan Kelly, and Timothy D. Barfoot. "Robust Monocular Visual Teach and Repeat Aided by Local Ground Planarity and Color-constant Imagery". In: *JFR* (2017).

[30] Michael Warren, Melissa Greeff, Bhavit Patel, Jack Collier, Angela P. Schoellig, and Timothy D. Barfoot. "There's No Place Like Home: Visual Teach and Repeat for Emergency Return of Multirotor UAVs During GPS Failure". In: *RA-L* (2019).

[31] Trung Nguyen, George K. I. Mann, Raymond G. Gosine, and Andrew Vardy. "Appearance-Based Visual-Teach-And-Repeat Navigation Technique for Micro Aerial Vehicle". In: *JINT* (2016).

[32] Tomáš Krajník, Pablo Cristóforis, Keerthy Kusumam, Peer Neubert, and Tom Duckett. "Image Features for Visual Teach-and-Repeat Navigation in Changing Environments". In: *Robotics and Autonomous Systems* (2017).

[33] Tristan Swedish and Ramesh Raskar. "Deep Visual Teach and Repeat on Path Networks". In: *CVPR*. 2018.

[34] Luis G. Camara, Tomas Pivonka, Martin Jilek, Carl Gabert, Karel Kosnar, and Libor Preucil. "Accurate and Robust Teach and Repeat Navigation by Visual Place Recognition: A CNN Approach". In: *IROS*. 2020.

[35] Philipp Krüsi, Bastian Bücheler, François Pomerleau, Ulrich Schwesinger, Roland Siegwart, and Paul Furgale. "Lighting-invariant adaptive route following using iterative closest point matching". In: *JFR* (2015).

[36] Philipp Krüsi, Paul Furgale, Michael Bosse, and Roland Siegwart. "Driving on Point Clouds: Motion Planning, Trajectory Optimization, and Terrain Assessment in Generic Nonplanar Environments". In: *JFR* (2017).

[37] Dominic Baril, Simon-Pierre Deschênes, Olivier Gamache, Maxime Vaidis, Damien LaRocque, Johann Laconte, Vladimír Kubelka, Philippe Giguère, and François Pomerleau. "Kilometer-Scale Autonomous Navigation in Subarctic Forests: Challenges and Lessons Learned". In: *arXiv* (2021).

[38] Timothy D. Barfoot and Paul T. Furgale. "Associating Uncertainty With Three-Dimensional Poses for Use in Estimation Problems". In: *IEEE Transactions on Robotics* (2014).

[39] Winston Churchill and Paul Newman. "Experience-Based Navigation for Long-Term Localisation". In: *IJRR* (2013).

[40] R. Brooks. "Visual Map Making for a Mobile Robot". In: *ICRA*. 1985.

[41] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. "Integrated Online Trajectory Planning and Optimization in Distinctive Topologies". In: *Robotics and Autonomous Systems* (2017).

[42] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI: An Introduction*. 2018.

[43] Jesse Levinson and Sebastian Thrun. "Robust vehicle localization in urban environments using probabilistic maps". In: *ICRA*. 2010.

[44] Ryan W Wolcott and Ryan M Eustice. "Fast LIDAR localization using multiresolution Gaussian mixture maps". In: *ICRA*. 2015.

[45] Jens Behley and Cyrill Stachniss. "Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments." In: *RSS*. 2018.

[46] Graham M Brooker, Steven Scheding, Mark V Bishop, and Ross C Hennessy. "Development and application of millimeter wave radar sensors for underground mining". In: *Sensors* (2005).

[47] Graham Brooker, Mark Bishop, and Steve Scheding. “Millimetre waves for robotics”. In: *Australian Conference for Robotics and Automation*. 2001.

[48] Holger Caesar, Varun Bankiti, Alex H Lang, Sourabh Vora, Venice Erin Liong, Qiang Xu, Anush Krishnan, Yu Pan, Giancarlo Baldan, and Oscar Beijbom. “nuscenes: A multimodal dataset for autonomous driving”. In: *CVPR*. 2020.

[49] Dominik Kellner, Michael Barjenbruch, Jens Klappstein, Jürgen Dickmann, and Klaus Dietmayer. “Instantaneous ego-motion estimation using doppler radar”. In: *ITSC*. 2013.

[50] Pou-Chun Kung, Chieh-Chih Wang, and Wen-Chieh Lin. “A Normal Distribution Transform-Based Radar Odometry Designed For Scanning and Automotive Radars”. In: *ICRA*. 2021.

[51] Pengen Gao, Shengkai Zhang, Wei Wang, and Chris Xiaoxuan Lu. “Accurate Automotive Radar Based Metric Localization with Explicit Doppler Compensation”. In: *arXiv* (2021).

[52] Steve Clark and Hugh Durrant-Whyte. “Autonomous land vehicle navigation using millimeter wave radar”. In: *ICRA*. 1998.

[53] MWM Gamini Dissanayake, Paul Newman, Steve Clark, Hugh F Durrant-Whyte, and Michael Csorba. “A solution to the simultaneous localization and map building (SLAM) problem”. In: *T-RO* (2001).

[54] Hermann Rohling. “Radar CFAR thresholding in clutter and multiple target situations”. In: *IEEE Transactions on Aerospace and Electronic Systems* (1983).

[55] Ebi Jose and Martin David Adams. “Relative radar cross section based feature identification with millimeter wave radar for outdoor slam”. In: *IROS*. 2004.

[56] Manjari Chandran and Paul Newman. “Motion estimation from map quality with millimeter wave radar”. In: *IROS*. 2006.

[57] R Rouveure, MO Monod, and P Faure. “High resolution mapping of the environment with a ground-based radar imager”. In: *RADAR*. 2009.

[58] Paul Checchin, Franck Gérossier, Christophe Blanc, Roland Chapuis, and Laurent Trassoudaine. “Radar scan matching slam using the fourier-mellin transform”. In: *FSR*. 2010.

[59] Jonas Callmer, David Törnqvist, Fredrik Gustafsson, Henrik Svensson, and Pelle Carlbom. “Radar SLAM using visual features”. In: *EURASIP Journal on Advances in Signal Processing* (2011).

[60] John Mullane, Ba-Ngu Vo, Martin D Adams, and Ba-Tuong Vo. “A random-finite-set approach to Bayesian SLAM”. In: *T-RO* (2011).

[61] Damien Vivet, Paul Checchin, and Roland Chapuis. “Localization and mapping using only a rotating FMCW radar sensor”. In: *Sensors* (2013).

[62] Frank Schuster, Christoph Gustav Keller, Matthias Rapp, Martin Haueis, and Cristóbal Curio. “Landmark based radar SLAM using graph optimization”. In: *ITSC*. 2016.

[63] Matthias Rapp, Michael Barjenbruch, Markus Hahn, Jürgen Dickmann, and Klaus Dietmayer. “Probabilistic ego-motion estimation using multiple automotive radar sensors”. In: *Robotics and Autonomous Systems* (2017).

[64] Sarah H Cen and Paul Newman. “Precise ego-motion estimation with millimeter-wave radar under diverse and challenging conditions”. In: *ICRA*. 2018.

[65] Dan Barnes, Matthew Gadd, Paul Murcutt, Paul Newman, and Ingmar Posner. “The Oxford Radar RobotCar Dataset: A Radar Extension to the Oxford RobotCar Dataset”. In: *ICRA*. 2020.

[66] Giseop Kim, Yeong Sang Park, Younghun Cho, Jinyong Jeong, and Ayoung Kim. “MulRan: Multimodal Range Dataset for Urban Place Recognition”. In: *ICRA*. 2020.

[67] Marcel Sheeny, Emanuele De Pellegrin, Saptarshi Mukherjee, Alireza Ahrabian, Sen Wang, and Andrew Wallace. “RADIATE: A radar dataset for automotive perception in bad weather”. In: *ICRA*. 2021.

[68] Sarah H Cen and Paul Newman. “Radar-only ego-motion estimation in difficult settings via graph matching”. In: *ICRA*. 2019.

[69] Roberto Aldera, Daniele De Martini, Matthew Gadd, and Paul Newman. “What could go wrong? introspective radar odometry in challenging environments”. In: *ITSC*. 2019.

[70] Roberto Aldera, Daniele De Martini, Matthew Gadd, and Paul Newman. “Fast radar motion estimation with a learnt focus of attention using weak supervision”. In: *ICRA*. 2019.

[71] Dan Barnes, Rob Weston, and Ingmar Posner. “Masking by Moving: Learning Distraction-Free Radar Odometry from Pose Information”. In: *CoRL*. 2020.

[72] Dan Barnes and Ingmar Posner. “Under the Radar: Learning to Predict Robust Keypoints for Odometry Estimation and Metric Localisation in Radar”. In: *ICRA*. 2020.

[73] Yeong Sang Park, Young-Sik Shin, and Ayoung Kim. “PhaRaO: Direct Radar Odometry using Phase Correlation”. In: *ICRA*. 2020.

[74] Keenan Burnett, Angela P Schoellig, and Timothy D Barfoot. “Do We Need to Compensate for Motion Distortion and Doppler Effects in Spinning Radar Navigation?” In: *RA-L* (2021).

[75] Keenan Burnett, David J Yoon, Angela P Schoellig, and Timothy D Barfoot. “Radar Odometry Combining Probabilistic Estimation and Unsupervised Feature Learning”. In: *RSS*. 2021.

[76] Daniel Adolfsson, Martin Magnusson, Anas Alhashimi, Achim J Lilienthal, and Henrik Andreasson. “CFEAR Radarodometry - Conservative Filtering for Efficient and Accurate Radar Odometry”. In: *IROS*. 2021.

[77] Anas Alhashimi, Daniel Adolfsson, Martin Magnusson, Henrik Andreasson, and Achim J Lilienthal. “BFAR-Bounded False Alarm Rate detector for improved radar odometry estimation”. In: *arXiv* (2021).

[78] Martin Holder, Sven Hellwig, and Hermann Winner. “Real-time pose graph SLAM based on radar”. In: *IV*. 2019.

[79] Ziyang Hong, Yvan Petillot, and Sen Wang. “RadarSLAM: Radar based Large-Scale SLAM in All Weathers”. In: *IROS*. 2020.

[80] Ziyang Hong, Yvan Petillot, Andrew Wallace, and Sen Wang. “RadarSLAM: A robust simultaneous localization and mapping system for all weather conditions”. In: *IJRR* (2022).

[81] Daniele De Martini, Matthew Gadd, and Paul Newman. "kRadar++: Coarse-to-Fine FMCW Scanning Radar Localisation". In: *Sensors* (2020).

[82] Ştefan Săftescu, Matthew Gadd, Daniele De Martini, Dan Barnes, and Paul Newman. "Kidnapped Radar: Topological Radar Localisation using Rotationally-Invariant Metric Learning". In: *ICRA*. 2020.

[83] Matthew Gadd, Daniele De Martini, and Paul Newman. "Look Around You: Sequence-based Radar Place Recognition with Learned Rotational Invariance". In: *PLANS*. 2020.

[84] Tim Y Tang, Daniele De Martini, Dan Barnes, and Paul Newman. "RSL-Net: Localising in Satellite Images From a Radar on the Ground". In: *RA-L* (2020).

[85] Tim Y Tang, Daniele De Martini, Shangzhe Wu, and Paul Newman. "Self-Supervised Localisation between Range Sensors and Overhead Imagery". In: *RSS*. 2020.

[86] Tim Y Tang, Daniele De Martini, and Paul Newman. "Get to the Point: Learning Lidar Place Recognition and Metric Localisation Using Overhead Imagery". In: *RSS*. 2021.

[87] Huan Yin, Yue Wang, Li Tang, and Rong Xiong. "Radar-on-lidar: metric radar localization on prior lidar maps". In: *RCAR*. 2020.

[88] Huan Yin, Runjian Chen, Yue Wang, and Rong Xiong. "Rall: end-to-end radar localization on lidar map using differentiable measurement model". In: *T-ITS* (2021).

[89] Sean Anderson and Timothy D. Barfoot. "Full STEAM Ahead: Exactly Sparse Gaussian Process Regression for Batch Continuous-Time Trajectory Estimation on SE(3)". In: *IROS*. 2015.

[90] Nicholas Charron, Stephen Phillips, and Steven L Waslander. "De-noising of lidar point clouds corrupted by snowfall". In: *CRV*. 2018.

[91] Alireza Asvadi, Cristiano Premebida, Paulo Peixoto, and Urbano Nunes. "3D Lidar-based Static and Moving Obstacle Detection in Driving Environments: An Approach Based on Voxels and Multi-Region Ground Planes". In: *Robotics and Autonomous Systems* (2016).

[92] Lihao Wang, Chengfeng Zhao, and Jun Wang. "A LiDAR-Based Obstacle-Detection Framework for Autonomous Driving". In: *ECC*. 2020.

[93] D Girardeau-Montaut, M Roux, R Marc, and G Thibault. "Change Detection on Points Cloud Data acquired with a Ground Laser Scanner". In: *ISPRS Workshop, Enschede, the Netherlands*. 2005.

[94] Henrik Andreasson, Martin Magnusson, and Achim Lilienthal. "Has Somethong Changed Here? Autonomous Difference Detection for Security Patrol Robots". In: *IROS*. 2007.

[95] Pedro Nunez, Paulo L. J. Drews, Rui P. Rocha, Mario F. M. Campos, and Jorge Dias. "Novelty Detection and 3D Shape Retrieval Based on Gaussian Mixture Models for Autonomous Surveillance Robotics". In: *IROS*. 2009.

[96] Antonio W. Vieira, Paulo L. J. Drews, and Mario F. M. Campos. "Efficient Change Detection in 3D Environment for Autonomous Surveillance Robots Based on Implicit Volume". In: *ICRA*. 2012.

[97] Ralf Kaestner, Sebastian Thrun, Michael Montemerlo, and Matt Whalley. "A Non-rigid Approach to Scan Alignment and Change Detection Using Range Sensor Data". In: *FSR*. 2006.

[98] Evan Herbst, Peter Henry, Xiaofeng Ren, and Dieter Fox. “Toward Object Discovery and Modeling via 3-D Scene Comparison”. In: *ICRA*. 2011.

[99] A. Elfes. “Using Occupancy Grids for Mobile Robot Perception and Navigation”. In: *Computer* (1989).

[100] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. “OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees”. In: *Autonomous Robots* (2013).

[101] Marcus Hebel, Michael Arens, and Uwe Stilla. “Change Detection in Urban Areas by Direct Comparison of Multi-view and Multi-temporal ALS Data”. In: *PIA*. 2011.

[102] Wen Xiao, Bruno Vallet, Mathieu Brédif, and Nicolas Paparoditis. “Street Environment Change Detection from Mobile Laser Scanning Point Clouds”. In: *P&RS* (2015).

[103] J. P. Underwood, D. Gillsjö, T. Bailey, and V. Vlaskine. “Explicit 3D Change Detection Using Ray-Tracing in Spherical Coordinates”. In: *ICRA*. 2013.

[104] F. Pomerleau, P. Krüsi, F. Colas, P. Furgale, and R. Siegwart. “Long-Term 3D Map Maintenance in Dynamic Environments”. In: *ICRA*. 2014.

[105] F. Ferri, M. Gianni, M. Menna, and F. Pirri. “Dynamic Obstacles Detection and 3D Map Updating”. In: *IROS*. 2015.

[106] Lorenz Wellhausen, Renaud Dubé, Abel Gawel, Roland Siegwart, and Cesar Cadena. “Reliable Real-Time Change Detection and Mapping for 3D LiDARs”. In: *SSRR*. 2017.

[107] Hugues Thomas, Ben Agro, Mona Gridseth, Jian Zhang, and Timothy D. Barfoot. “Self-Supervised Learning of Lidar Segmentation for Autonomous Indoor Navigation”. In: *ICRA*. 2021.

[108] David Yoon, Tim Tang, and Timothy Barfoot. “Mapless Online Detection of Dynamic Objects in 3D Lidar”. In: *CRV*. 2019.

[109] D. Hahnel, R. Triebel, W. Burgard, and S. Thrun. “Map Building with Mobile Robots in Dynamic Environments”. In: *ICRA*. 2003.

[110] Alexander Schaefer, Lukas Luft, and Wolfram Burgard. “An Analytical Lidar Sensor Model Based on Ray Path Information”. In: *RA-L* (2017).

[111] L. Luft, A. Schaefer, Tobias Schubert, and W. Burgard. “Detecting Changes in the Environment Based on Full Posterior Distributions Over Real-Valued Grid Maps”. In: *RA-L* (2018).

[112] Chieh-Chih Wang and C. Thorpe. “Simultaneous Localization and Mapping with Detection and Tracking of Moving Objects”. In: *ICRA*. 2002.

[113] Aisha Walcott-Bryant, Michael Kaess, Hordur Johannsson, and John J. Leonard. “Dynamic Pose Graph SLAM: Long-term Mapping in Low Dynamic Environments”. In: *IROS*. 2012.

[114] Ahmad Aijazi, Paul Checchin, and Laurent Trassoudaine. “Automatic Removal of Imperfections and Change Detection for Accurate 3D Urban Cartography by Classification and Incremental Updating”. In: *Remote Sensing* (2013).

[115] Rares Ambrus, Nils Bore, John Folkesson, and Patric Jensfelt. “Meta-Rooms: Building and Maintaining Long Term Spatial Models in a Dynamic World”. In: *IROS*. 2014.

[116] Cyrill Stachniss and Wolfram Burgard. “Mobile Robot Mapping and Localization in Non-Static Environments”. In: (2005).

[117] Kurt Konolige and James Bowman. “Towards Lifelong Visual Maps”. In: *IROS*. 2009.

[118] Daniel Meyer-Delius, Maximilian Beinhofer, and Wolfram Burgard. “Occupancy Grid Models for Robot Mapping in Changing Environments”. In: *AAAI* (2012).

[119] Tomasz Kucner, Jari Saarinen, Martin Magnusson, and Achim J. Lilienthal. “Conditional Transition Maps: Learning Motion Patterns in Dynamic Environments”. In: *IROS*. 2013.

[120] Tomas Krajnik, Jaime Pulido Fentanes, Grzegorz Cielniak, Christian Dondrup, and Tom Duckett. “Spectral Analysis for Long-Term Robotic Mapping”. In: *ICRA*. 2014.

[121] Ross Finman, Thomas Whelan, Michael Kaess, and John J. Leonard. “Toward Lifelong Object Segmentation from Change Detection in Dense RGB-D Maps”. In: *ECMR*. 2013.

[122] Rares Ambrus, Johan Ekekrantz, John Folkesson, and Patric Jensfelt. “Unsupervised Learning of Spatial-Temporal Models of Objects in a Long-Term Autonomy Scenario”. In: *IROS*. 2015.

[123] Marius Fehr, Fadri Furrer, Ivan Dryanovski, Jürgen Sturm, Igor Gilitschenski, Roland Siegwart, and Cesar Cadena. “TSDF-based Change Detection for Consistent Long-Term Dense Reconstruction and Dynamic Object Discovery”. In: *ICRA*. 2017.

[124] Kevin P. Murphy. *Conjugate Bayesian analysis of the Gaussian distribution*. Tech. rep. 2007.

[125] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. “G2o: A General Framework for Graph Optimization”. In: *ICRA*. 2011.

[126] Hugues Thomas, Matthieu Gallet de Saint Aurin, Jian Zhang, and Timothy D. Barfoot. “Learning Spatiotemporal Occupancy Grid Maps for Lifelong Navigation in Dynamic Scenes”. In: *arXiv* (2021).