

Copyright Undertaking

This thesis is protected by copyright, with all rights reserved.

By reading and using the thesis, the reader understands and agrees to the following terms:

1. The reader will abide by the rules and legal ordinances governing copyright regarding the use of the thesis.
2. The reader will use the thesis for the purpose of research or private study only and not for distribution or further reproduction or any other purpose.
3. The reader agrees to indemnify and hold the University harmless from and against any loss, damage, cost, liability or expenses arising from copyright infringement or unauthorized usage.

IMPORTANT

If you have reasons to believe that any materials in this thesis are deemed not suitable to be distributed in this form, or a copyright owner having difficulty with the material being included in our database, please contact lbsys@polyu.edu.hk providing details. The Library will look into your claim and consider taking remedial action upon receipt of the written requests.

MULTI-SENSOR FUSION LIDAR SLAM FOR ADVANCED ROBOTICS

KAIDUO FANG

PhD

The Hong Kong Polytechnic University

2026

The Hong Kong Polytechnic University

Department of Electrical and Electronic Engineering

**Multi-Sensor Fusion LiDAR SLAM
for Advanced Robotics**

Kaiduo Fang

A thesis submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy

August 2025

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgement has been made in the text.

_____ (Signed)

Kaiduo FANG (Name of student)

Dedication

To my parents, my girl, and friends

*For supporting me
in this long journey*

Abstract

LiDAR Odometry (LO) and LiDAR-Inertial Odometry (LIO) are crucial for autonomous robotics but face challenges in accuracy (especially in dynamic or feature-scarce environments), computational efficiency for low-cost platforms, generalization across LiDAR types, and inconsistency in filter-based methods. While optimization-based LIO offers accuracy, it is computationally expensive. Extended Kalman Filter (EKF)-based LIO is more efficient but suffers from inconsistent estimation due to linearization. Invariant EKF (InEKF) addresses inconsistency through Lie group theory but remains under-explored for LIO.

This thesis presents novel frameworks addressing these gaps. First, we propose Inc-DLOM, a generalized, efficient LO framework using Voxel-to-Voxel (V2V) direct registration. It optimizes Generalized ICP (GICP) by downsampling point clouds into voxels storing centroids and covariances. This V2V formulation significantly improves computational efficiency while retaining essential information. An Incremental Voxel Mapping (IVM) method, based on spatial hashing, enables $\mathcal{O}(1)$ neighbor search and efficient map updates with single-scan insertion. Inc-DLOM flexibly adapts to point cloud data from various LiDAR architectures. Our evaluation primarily focuses on both mechanical rotating and solid-state LiDARs.

Building on Inc-DLOM, we introduce Invariant-DLIO, a tightly coupled LIO system resolving EKF inconsistency using Lie group-based InEKF. The state (rotation, velocity, translation) is defined on the $SE_2(3)$ manifold, extended with IMU biases ($SE_2(3) \times \mathbb{R}^6$). This structure yields approximately log-linear, trajectory-independent error dynamics, en-

hancing consistency. The correction step leverages direct registration in an iterated optimization scheme. Combined with IVM, this forms a lightweight, fast, and robust LIO system.

Both frameworks were extensively validated on public datasets and real-world robotic platforms across diverse environments, scenarios, and motion dynamics. Inc-DLOM demonstrated superior accuracy and computational efficiency compared to state-of-the-art open-source LO methods (e.g., HDL-SLAM, KISS-ICP). Invariant-DLIO outperformed leading open-source LIO methods in both accuracy and efficiency, confirming the benefits of InEKF for consistent estimation.

Finally, recognizing deployment challenges on resource-constrained hardware (e.g., Jetson Orin Nano), we explored CUDA-accelerated deployment of Inc-DLOM for autonomous driving digital twin applications, paving the way for real-time performance on low-cost platforms.

This work delivers significant advancements in LO/LIO through: 1) A general, efficient V2V direct registration LO (Inc-DLOM); 2) A consistent, tightly-coupled InEKF-based LIO (Invariant-DLIO); 3) Comprehensive validation showing state-of-the-art performance; 4) Initial exploration of GPU acceleration for broader deployment.

Acknowledgements

Support is the foundation upon which great achievements are built

First and foremost, my deepest gratitude goes to my parents; your unwavering love, patience, and steadfast support throughout this extended academic journey have been my anchor. Without your sacrifices and encouragement, this accomplishment would not have been possible.

Thanks to my supervisor, Dr. Ivan HO, for his exceptional mentorship. His insightful guidance and constant encouragement were invaluable in shaping this research. Special recognition is due to my major collaborator, Dr. Song Rui, whose intellectual partnership proved indispensable. Together we have accomplished a series of significant works that form the cornerstone of this dissertation.

To my colleagues and friends across three stages, at Jilin University, University of Michigan, and The Hong Kong Polytechnic University, thank you for enriching both my academic path and personal growth. Your camaraderie during challenging moments and shared triumphs have been vital throughout my journey. As we move forward, I extend my warmest wishes for your continued success and fulfillment.

Kaiduo Fang

Hong Kong

August 2025

Contents

Acknowledgements	iv
1 Introduction	2
1.1 Background and Motivation	2
1.2 Objectives	6
1.3 List of Publication	9
1.4 Outlines	10
2 Literature Review	12
2.1 Point Cloud Registration	12
2.2 LiDAR Odometry	15
2.3 LiDAR Inertial Odometry	17
2.4 Invariant Extended Kalman Filtering (InEKF)	19
3 Incremental Direct LiDAR Odometry and Mapping	21
3.1 Introduction	21
3.2 Proposed Method	22
3.2.1 Preprocess	23
3.2.2 Scan-to-Scan (S2S) Matching	24
3.2.2.1 Voxelization	24
3.2.2.2 Voxel-to-Voxel (V2V) GICP	25
3.2.3 Scan-to-Mapping (S2M) Matching	25

3.2.3.1	Global Voxel Information	25
3.2.3.2	Incremental Voxel Mapping	26
3.2.3.3	Incremental GICP	28
3.3	Experimental Results	30
3.3.1	Experimental Setup	30
3.3.2	Autonomous Driving Scenario	32
3.3.3	High Frequency Motion Scenario	35
3.3.4	Solid-State LiDAR Scenario	36
3.3.5	Magni Robot Scenario	37
3.4	Conclusion	37
4	Invariant Kalman Filtering Based LiDAR-Inertial Odometry	41
4.1	Introduction	41
4.2	Proposed Method	42
4.2.1	Theory Preliminaries	42
4.2.2	State Representation	44
4.2.3	IMU Kinematics Model and Propagation	46
4.2.4	Invariant Extended Kalman Filtering (InEKF) Observation	48
4.2.4.1	Point Cloud Voxelization	48
4.2.4.2	Correction	49
4.2.4.3	Incremental Update on InEKF	52
4.2.5	Scan-to-Mapping Refinement	53
4.2.5.1	Incremental Voxel Mapping	53
4.2.5.2	Incremental GICP	54
4.3	Experimental Results	55
4.3.1	Experiment Setup	55
4.3.2	Ablation Study	55
4.3.2.1	Effect of Incremental InEKF Update	55
4.3.2.2	Validation of IVM	58

4.3.2.3	Validation of InEKF	59
4.3.3	KITTI Odometry Evaluation	60
4.3.4	High-Frequency Motion Evaluation	61
4.3.5	Custom Robotic Dataset	62
4.3.5.1	Indoor Dataset	63
4.3.5.2	Campus Dataset	63
4.3.6	Computational Efficiency	64
4.4	Conclusion	64
5	Deploying SLAM: Applications	67
5.1	Motivation	67
5.2	PolyU-F1Tenth	68
5.3	Proposed Method	71
5.3.1	Point Cloud Preprocessing	72
5.3.2	CUDA-Preprocessing	73
5.3.2.1	Voxel-Based Downsampling via GPU Hash Table	74
5.3.2.2	CUDA-Based Matrix Exponential for SE(3)	74
5.3.3	Scan-to-Scan (S2S) Matching	75
5.3.3.1	Voxelization	75
5.3.3.2	Voxel-to-Voxel (V2V) GICP	76
5.3.4	CUDA Incremental Voxel Mapping (CUDA-IVM)	76
5.3.4.1	CPU IVM	76
5.3.4.2	CUDA Implementation	78
5.3.5	CUDA-Accelerated Incremental GICP (CUDA IGICP)	81
5.4	Experiments	83
5.4.1	CUDA Acceleration on Host Computer	83
5.4.2	Deployment for Jetson Series	86
5.4.3	Mapping Result for Different Datasets	87
5.4.3.1	PolyU Campus	87

<i>CONTENTS</i>	viii
5.4.3.2 Textureless Corridor	90
5.4.3.3 Open-Source Dataset	90
6 Conclusion	93
6.1 Contributions	93
6.2 Future Works	95
6.2.1 Graph Optimization	95
6.2.2 Autoware Integration	97
7 References	101

List of Figures

3.1	Inc-DLOM Framework.	23
3.2	The visualization of IVM. (a) The plot of the point cloud mapping. (b) The plot of IVM on the mapping.	28
3.3	Data Collection Robotic Platform	31
3.4	Textureless Corridor View	31
3.5	Inc-DLOM Trajectories on selected dataset sequences. The figure shows the estimated trajectories of dataset sequences, including multiple LiDAR types and different working scenarios. The plot of ground truths (dashed line) and estimated trajectories (blue) shows good estimation performance of our proposed method.	33
3.6	Performance on NCD-Math-Hard. (Left) Estimated trajectory (blue), comparing with the result from DLO (Orange). (Right) The mapping result from Inc-DLOM.	36
3.7	RS-L515 Mapping Comparison. (i) HDL-SLAM failed to track the motion of a solid-state LiDAR. (ii) The mapping result from DLO is better than (i), but the zoomed views show many twisted objects mapping views. (iii) Inc-DLOM can achieve good mapping results.	38
3.8	The point cloud mapping of different methods.	39
3.9	Zoomed views of different parts of point cloud mapping.	40

4.1	Invariant-DLIO takes the point cloud $\mathcal{P}_k$ and IMU raw data $\tilde{\omega}_t, \tilde{a}_t$ as inputs, and goes through the procedure of Voxelization and InEKF Propagation . InEKF Correction step takes voxelized cloud $\mathcal{P}_k$ and propagated state $\hat{\mathbf{X}}_t$ as inputs to estimate robot state $\tilde{\mathbf{X}}_{k+1}$, where the rotation part $\tilde{\mathcal{R}}_t$ and translational part $\tilde{p}_t$ are taken and formed $\Delta\mathcal{T}$ as the initial guess for Incremental GICP . The proposed <i>Incremental Voxel Mapping (IVM)</i> $\mathcal{M}$ will be updated by <i>Global Voxel Information</i> $\mathcal{V}_k^*$. The global pose $\mathcal{T}_k$ is refined by the proposed Incremental GICP	42
4.2	Data Collection Robotic Platform	56
4.3	Textureless Corridor View	56
4.4	Trajectories estimated by Invariant-DLIO, the color bar means the error plot.	57
4.5	Time Cost of Incremental InEKF Update.	58
4.6	The visualization of IVM. (a) The plot of the point cloud mapping. (b) The plot of IVM on the mapping.	59
4.7	Trajectory plot of Z-axis.	63
4.8	Computation Time per Frame.	64
4.9	The point cloud mapping of campus garden.	66
5.1	Approximate Geekbench 5 Scores for a Range of Desktop, Embedded, and Low-cost CPUs	68
5.2	F1Tenth Hardware [75]	69
5.3	PolyU-F1Tenth Hardware [75]	70
5.4	CUDA-Accelerated LO Framework.	72
5.5	Sequential IVM updating with input voxels.	77
5.6	CUDA-IVM updating with input voxels.	79
5.7	Processing Time per Frame	84
5.8	CUDA Acceleration on Submodules	85
5.9	Processing Time per Frame on NCLT_1	87

5.10	CUDA-Accelerated LO Mapping	88
5.11	DLO Mapping	89
5.12	Fast-LIO Mapping	89
5.13	KISS-ICP Mapping	90
5.14	Mapping in Long Corridor	91
5.15	High Fidelity Mapping of NCLT	91
5.16	High Fidelity Mapping of Newer College Dataset	92
6.1	An example of pose graph.	95
6.2	Dataset Trajectory	97
6.3	Mapping Result of Beijing Data	98
6.4	Tracking Result Under Narrow Corridor	98
6.5	Global Drifting on z-axis	98
6.6	Autoware Framework	99
6.7	Digital Twin Framework	100

List of Tables

1.1	Characteristics of Different LiDARs	5
3.1	Statistical Results of the IVM.	28
3.2	Detailed Time Costs of Subprocedures in Inc-DLOM.	29
3.3	Translational Drifting Error (%) / Rotational Drifting Error (deg /100m) Evaluation of KITTI Dataset	34
3.4	RMSE of ATE (m) Evaluation of KITTI Dataset	34
3.5	RMSE of ATE (m) Evaluation of NCLT Dataset and Newer College Dataset	35
4.1	Statistical Results of the IVM.	59
4.2	Translational Drifting Error (%) / Rotational Drifting Error (deg /100m) Evaluation of KITTI Dataset	60
4.3	RMSE of ATE (m) Evaluation of KITTI Dataset	61
4.4	RMSE of ATE (m) Evaluation of NCLT Dataset and Newer College Dataset	62

List of Abbreviations

ARM Advanced RISC Machines

ATE Absolute Trajectory Error

ATI-CTLO Adaptive Temporal Interval-based Continuous- Time LiDAR-only Odometry

AV Autonomous Vehicle

BA bundle adjustmen

CPU Central Processing Unit

CT-ICP Continuous Time ICP

CUDA Compute Unified Device Architecture

CUDA IGICP CUDA-Accelerated Incremental GICP

DLO Direct LiDAR Odometry

DoF Degree-of-Freedom

EKF Extended Kalman Filter

ESKF Error-State Kalman Filtering

Fast-LIO Fast LiDAR-Inertial Odometry)

Fast-VGICP Voxelized GICP

FLANN Fast Library for Approximate Nearest Neighbors

FoV Field-of-View

GB Gigabyte

GICP Generalized Iterative Closest Point

GNSS Global Navigation Satellite Systems

GPS Global Positioning System

GPU Graphics Processing Unit

ICP Iterative Closest Point

ikd-Tree Incremental KD-Tree

IMU Inertial Measurement Unit

Inc-DLOM Incremental Direct LiDAR Odometry and Mapping

InEKF Invariant Extended Kalman Filtering

INS Inertial Navigation Systems

Inv-DLIO Invariant Kalman Filtering Based Direct LiDAR Inertial Odometry

iSAM2 incremental smoothing and mapping with the Bayes tree

IVM Incremental Voxel Mapping

KB Kilobyte

KD Tree K-Dimensional Tree

LC-LIO loosely-coupled LiDAR-Inertial Odometry

LeGO-LOAM Lightweight and Groud-Optimized LiDAR Odometry and Mapping on
Variable Terrain

LiDAR Light Detection and Ranging

LINS Lidar-Inertial State Estimator

LIO LiDAR-Inertial Odometry

LIO-SAM Lidar Inertial Odometry via Smoothing and Mapping

LO LiDAR Odometry

LOAM LiDAR Odometry and Mapping in Real-time

MAP maximum a posterior

MB MegaByte

ms millisecond

MULLS Versatile LiDAR SLAM via Multi-metric Linear Least Square

MulRan Multimodal Range Dataset for Urban Place Recognition

NCD Newer College Dataset

NCLT Dataset The University of Michigan North Campus Long-Term Vision and LIDAR Dataset.

NDT Normal Distributions Transform

NICP Normal ICP

OpenMP Open Multi-Processing

PC Personal Computer

PCA Principal Component Analysis

PCL Point Cloud Library

PCR point cloud registration

PL-ICP Point-to-Line ICP

PVE-LIOM Pseudo-Visual Enhanced LiDAR-Inertial Odometry and Mapping

RF radio-frequency

RMSE root mean squared error

RS-L515 Intel Realsense L515

S2M Scan-to-Mapping

S2S Scan-to-Scan

SLAM Simultaneous Localization And Mapping

SuMa Surfel-based Mapping using 3D Laser Range Data

SWIR short-wave infrared

T-LOAM Truncated Least Squares LiDAR-Only Odometry and Mapping in Real Time

TBB Threading Building Blocks

UAV Unmanned Aircraft Vehicles

UGV Unmanned Ground Vehicles

V2V Voxel-to-Voxel

VG Voxel Grid Downsampling

VLP Velodyne Puck

VR Virtual Reality

VRAM Video Random Access Memory

Chapter 1

Introduction

1.1 Background and Motivation

Robust localization is a foundational requirement for various types autonomous systems, including Autonomous Vehicles (AV), Unmanned Aircraft Vehicles (UAV), etc., where safety and operational efficacy demand high precision in real-time ego-motion estimation [83, 76, 32]. While Global Navigation Satellite Systems (GNSS) offer global coverage [73, 48], their signals degrade or fail entirely in urban canyons, dense forests, tunnels, and indoor environments, creating critical operational voids [84, 49]. Alternative methods struggle to bridge this gap reliably: camera-based vision odometry often fails in low-light or textureless scenes and suffers from scale ambiguity[63], inertial navigation systems (INS) accumulate unbounded drift due to sensor noise[90], and radio-frequency (RF) techniques (e.g., trilateration, direction-of-arrival) depend on external infrastructure or anchors [86], limiting scalability. To overcome these issues, LiDAR (Light Detection and Ranging) offers a powerful solution, which is a sensor generating precise 3D point clouds measurements of the surrounding environment. Unlike passive sensors, LiDAR delivers metric measurements invariant to lighting, captures structural geometry in feature-poor settings (e.g., snow-covered landscapes), and penetrates obscurants like fog or smoke via short-wave infrared (SWIR) wavelengths [78, 93, 22, 34]. By enabling infrastructure-free,

high-fidelity scene reconstruction, LiDAR Odometry (LO) provides the resilience and accuracy needed for autonomy in GPS-denied or dynamically complex terrains, making it indispensable for next-generation navigation systems [97].

LiDAR Odometry (LO) addresses the problem of estimating egomotion by sequentially registering LiDAR point clouds [39, 51]. Robot motion, typically decomposed into rotational and translational components, is commonly expressed as a transformation matrix representing pose within a global frame. Contemporary LO methodologies [35], originating from foundational works such as LOAM [89], frequently adopt a dual-registration framework: *scan-to-scan* and *scan-to-map*. Scan-to-scan registration estimates the relative transformation between consecutive point cloud frames [36, 31, 15]. Concurrently, a persistently maintained global map, derived from historical scans, facilitates scan-to-map registration. This latter process refines pose estimation by aligning the current frame with the global map, initialized using the scan-to-scan transformation. To satisfy real-time computational constraints imposed by dense point cloud data, LO systems typically extract sparse feature points for efficient estimation [91, 7, 82]. However, feature-based approaches are inherently reliant on the discriminative quality of extracted features; consequently, their performance degrades significantly in geometrically degenerate or textureless environments [13]. Furthermore, the continuous expansion of the global map introduces escalating computational demands, ultimately impairing long-term operational efficiency. A critical additional limitation is the prevalent dependence of existing methods on specific LiDAR hardware characteristics. For instance, most LOAM-derived approaches assume the use of mechanical LiDARs, extensively leveraging fixed ring patterns during preprocessing; this inherent design impedes adaptation to novel sensor types. Collectively, these limitations, including performance degradation in challenging environments, computational inefficiency over time, and sensor-specific design, restricted the broad deployment of LiDAR Odometry across diverse robotic platforms.

Beyond the limitations of feature-based LO, alternative localization paradigms face their own challenges in complex environments. Probabilistic methods, such as particle

filters, provide robustness to uncertainty but are notoriously computationally expensive for high-dimensional state estimation in large-scale 3D spaces. Their performance heavily depends on the design of the motion and observation models, which are difficult to generalize across diverse sensor types and environments. Therefore, the core challenge persists: how to achieve real-time, accurate, and, crucially, generalizable LiDAR-based localization that is efficient, robust to environmental degradation, and independent of specific sensor hardware.

Consequently, enhancing the generalization capability of LO systems emerges as a critical research imperative. This generalization encompasses operational robustness across varied environments (e.g., textureless, indoor, outdoor) and adaptability to different LiDAR technologies, including variations in scanning patterns (rings), resolution, and sensing modalities (e.g., mechanical vs. solid-state).

As a complement, solid-state LiDARs can offer different application scenarios than traditional mechanical LiDARs [46, 50]. Unlike mechanical ones, which scan their surroundings through rotating structures, solid-state LiDARs generate point cloud data with a silicon chip [79, 12]. Due to different scanning principles, the point clouds generated from solid-state LiDARs have unique characteristics. For small-scale applications, such as Virtual Reality (VR) and indoor navigation, solid-state LiDARs are essential in the SLAM systems. However, most existing LO methods rely on data formats that are specific to mechanical LiDARs (such as ring, intensity, etc.), which makes them unavailable for solid-state LiDAR applications. In addition, different versions of mechanical LiDARs also have different characteristics, e.g., Field-of-View (FoV), number of channels, resolutions, etc., which are summarized in Table I [29, 10, 79, 64]. All of these mentioned properties introduce discrepancies in the point cloud data, and their implications for different LO methods can be significant, which makes the generalization of LO methods a key outstanding issue. Motivated by these challenges, the first part of this thesis investigates pathways towards generalized LiDAR Odometry, as will be elaborated in the research objectives.

Table 1.1: Characteristics of Different LiDARs

Sensor	Type	Frequency	FoV	Horizontal Resolution	Vertical Resolution
Velodyne HDL-64E	mechanical	5–15 Hz	$360^\circ \times 26.8^\circ$	0.08°	0.41°
Velodyne HDL-32E	mechanical	5–15 Hz	$360^\circ \times 41.33^\circ$	0.08°	1.33°
Ouster OS-0 128	mechanical	10 Hz	$360^\circ \times 90^\circ$	0.175°	0.70°
Realsense L515	solid-state	30 Hz	$70^\circ \times 55^\circ$	0.07°	0.07°

LiDAR-Inertial Odometry (LIO) is fundamentally motivated by the inherent limitations of standalone LiDAR Odometry (LO). While LO provides metric ego-motion estimates through point cloud registration, its performance critically depends on distinct geometric features and degrades significantly in geometrically degenerate or textureless environments. Furthermore, LO systems accumulate drift over time, struggle with motion distortion during rapid maneuvers, and often exhibit hardware dependencies tied to specific LiDAR characteristics. Inertial Measurement Units (IMUs) offer highly complementary data, providing high-frequency, metric motion estimates (linear acceleration and angular velocity) independent of external scene geometry. Methodologically, LIO systems are broadly categorized as optimization-based [71, 87, 94] or filtering-based [62, 1, 85]. Optimization-based LIO typically formulates measurements from LiDAR point clouds and IMU as factors within a graph-based optimization framework. However, this approach is often computationally expensive due to the need to fuse large numbers of measurements [71]. Consequently, filtering-based LIO methods, particularly those leveraging the Extended Kalman Filter (EKF), have gained attention for their superior computational efficiency while achieving comparable accuracy. Nevertheless, most existing EKF-based LIO implementations [85, 62] rely on hand-crafted feature extractors to select points for the correction step, reducing robustness when insufficient features are available. Moreover, a fundamental limitation of the EKF framework is its susceptibility to inconsistent estimation arising from linearization errors; this can cause state propagation interference and inaccurate covariance estimates, failing to reflect true estimator uncertainty [38]. To address these consistency issues, the Invariant Extended Kalman Filter

(InEKF) has emerged [3, 33]. Compared to the traditional EKF, InEKF preserves symmetry and exhibits log-linear autonomous error dynamics, leading to superior convergence and consistency. Despite these advantages, the application of InEKF to LiDAR-Inertial Odometry remains relatively unexplored, with prior work primarily focused on Visual-Inertial Odometry, highlighting a significant research gap. The second part of this thesis work is motivated from this research gap.

Despite extensive research on LiDAR Odometry (LO) and LiDAR-Inertial Odometry (LIO) methods, fundamental challenges persist in meeting computational efficiency requirements. Many state-of-the-art LO/LIO implementations rely on high-end CPUs and GPUs to achieve optimal accuracy and real-time performance. However, such hardware resources are often unavailable in embedded systems (e.g., NVIDIA Jetson Nano [41]) or shared with concurrent computationally intensive tasks in modern robotic pipelines, including deep learning-based perception (e.g., YOLO [65]), reinforcement learning for motion planning [28], and other resource-demanding processes. Consequently, localization algorithms cannot monopolize available computing resources, necessitating lightweight yet accurate and robust LO/LIO solutions. This challenge is further compounded in SLAM deployments, where point cloud processing modules, such as KD-Tree construction for neighborhood searches, historical map management, and pose optimization, impose significant CPU overhead. These operations exhibit substantially degraded performance on resource-constrained platforms compared to high-end workstations. To address these limitations, a promising direction involves offloading select computations to embedded GPUs. The final component of this thesis targets these critical deployment challenges, aiming to bridge the gap between algorithmic performance and practical computational constraints.

1.2 Objectives

The primary objectives of this research were to improve the localization performance for LiDAR perception based autonomous robots, regardless of LiDAR types, working sce-

narios, or platform properties.

Based on these objectives, the generalized LiDAR Odometry problem is firstly raised. The generalization includes working scenarios, point cloud properties, and motion frequencies. Traditional hand-crafted feature extractors cannot be reliable for point cloud preprocessing. The direct registration methods are utilized. To solve the conflict between real-time requirement and computational load, we proposed Incremental Direct LiDAR Odometry and Mapping (Inc-DLOM) in the first part of the thesis. Inc-DLOM is a direct registration-based LiDAR Odometry (LO) method, which is fast, computationally efficient, and able to provide accurate pose estimation performance for different types of LiDAR in multiple working scenarios. The main parts of Inc-DLOM are Scan-to-Scan and Scan-to-Mapping.

As the most existing LO methods, Scan-to-Scan (S2S) is the process handling the relative transformation between two adjacent frames, which is also known as the point cloud registration problem. Getting accurate estimation result of S2S is essential for LO framework, it can provide proper initial guess for Scan-to-Mapping or pose graph construction. Poor S2S results will lead to motion tracking failures, hence, selecting suitable method is important. As one of the most powerful and popular methods, Generalized Iterative Closest Point (GICP) is well-known for its accuracy, especially for real-world data, like point cloud collected by KITTI Dataset. But GICP's efficiency is also a outstanding issue, since the algorithm requires extensive usage of nearest neighbor searching, which is often difficult to achieve real-time performance for common computer CPUs. Out of the efficiency of GICP algorithm, a lightweight Generalized ICP (GICP) version [68], called Voxel-to-Voxel (V2V) GICP was proposed. The original point cloud data is transformed into a set of voxels, and the optimization object is also transformed into a "voxel-to-voxel" type, which improves the computational efficiency without reducing the necessary information in the original data.

Managing historical point cloud maps is essential for computational efficiency and accuracy. Existing methods often store and concatenate historical point cloud frames and ac-

cess them with data structures such as KD-Tree or Octo-Tree to perform nearest-neighbor searches [6, 17]. Although special data structures can reduce computational complexity, they are no longer efficient as the point cloud mapping expands. This is because they usually require continuous data structure reconstruction during the matching process [89], which can make the scan-to-mapping module slow and memory-intensive. Therefore, the appropriate mapping management procedure for LO and the framework to work with it remains an outstanding issue. To solve this issue, We proposed an efficient mapping method called Incremental Voxel Mapping (IVM) with pre-computed voxels. The IVM implementation is based on spatial hashing, which enables incremental insertion and update of the historical mapping. The mapping construction is quite efficient since every insertion contains only one point cloud. With spatial hashing, the IVM can perform neighbor searching at the computational complexity of $\mathcal{O}(1)$. Based on IVM's constant searching method, we proposed an efficient Scan-to-Mapping method, Incremental GICP. The newest point cloud scan is registered with the IVM, in which the optimization problem is constructed with $\mathcal{O}(1)$ operation.

As mentioned previously, most of the existing LIO methods are based on the combination of traditional error-state EKF and hand-crafted feature extractors [62, 2, 88, 1], which makes their performance unstable under various environments. In addition, the existing inconsistency still limits the performance of LIO methods. To solve these issues, we designed the tightly-coupled Invariant EKF state estimator for LiDAR-Inertial Odometry based on the invariant observer and Lie theory. The error dynamics include $SE_2(3)$ groups and IMU biases, which are approximately log-linear and trajectory-independent. The correction step of InEKF is designed based on V2V GICP's optimization objective, by integrating with IMU data in an iterative optimization manner. However, the existing EKF-based LIOs are sharing a similar updating procedure by stacking all measurements into matrices with large dimensions, and the efficiency will be affected by the increase in measurement numbers, which are often larger than 3000 with point cloud observations. Directly performing expensive matrix operations, e.g., inversion, multiplication, etc., will

be too time consuming for the tight real-time requirements. To target this issue, we also proposed an incremental construction method for InEKF implementation, which allows efficient matrix operations to improve LIO's speed. After that, the IVM-based Incremental GICP is also utilized to construct Scan-to-Mapping refinement procedure.

Finally, the third part of this thesis is motivated by the deployment on F1Tenth [75], a 1/10 scale autonomous vehicle simulation model. As a lightweight AV model, F1Tenth is using Jetson Orin Nano [41] as computing platform, which has far less computing power than ordinary host CPUs, e.g., Intel i7. The limitation of hardware raises challenges in the deployment of SLAM algorithms. Due to the large amount of data, the existing LO/LIO methods relies on the extensive usage of CPU processing, including KD-Tree for data query, multi-process of different submodules, different matrix operations, etc. The limitation of CPU performance will degrade the accuracy and efficiency of LO/LIO algorithms. Jetson Orin Nano contains lightweight CPU with ARM architecture, which makes the common multi-process methods applied for popular PC not applicable or optimal, such as Automatic Vectorization [56], OpenMP [18], and Intel TBB [66]. Jetson Orin Nano provides embedded GPU as the computing complement, mostly for AI-based technologies. As a natural consideration, GPU's application should be implemented for LO/LIO acceleration. An agile Autonomous Driving prototype, PolyU-F1Tenth, is implemented and deployed with rich sensor suite for advanced robotics applications. Inc-DLOM is also reimplemented with CUDA acceleration for the SLAM application, especially maintaining the accurate performance under insufficient computing resources.

1.3 List of Publication

The core parts of the thesis are published in papers [25] and [26]. We also presented part of the work in conferences [23] and [24].

Journal Papers

- K. Fang, R. Song and I. W. -H. Ho, “Inc-DLOM: Incremental Direct LiDAR Odometry and Mapping,” in *IEEE Access*, vol. 13, pp. 6527-6538, 2025, doi: 10.1109/ACCESS.2025.3526626.
- K. Fang, R. Song and I. Wang-Hei Ho, “Invariant-DLIO: Direct LiDAR–Inertial Odometry Based on Invariant Kalman Filtering,” in *IEEE Sensors Journal*, vol. 25, no. 11, pp. 20572-20583, 1 June1, 2025, doi: 10.1109/JSEN.2025.3558916.

Conference Papers

- K. Fang and I. W. -H. Ho, “Real-Time GICP: Direct LiDAR SLAM for CPU Environment,” in *TENCON 2022 - 2022 IEEE Region 10 Conference (TENCON)*, Hong Kong, Hong Kong, 2022, pp. 1-6, doi: 10.1109/TENCON55691.2022.9977999.
- K. Fang, and R. Song, “GMLO: Ground and Memory Optimized LiDAR Odometry,” in *Proceedings of the 9th EAI International Conference on Smart Grid and Innovative Frontiers in Telecommunications (EAI SmartGIFT 2024)*, Hong Kong SAR, China, 2024, pp. 1-9.

1.4 Outlines

Chapter 2 gives a review of the relevant literature. This includes a review of point cloud registration (PCR), LiDAR Odometry (LO), LiDAR Inertial Odometry (LIO), and Invariant Kalman Filtering problems of robotics navigation.

Chapter 3 presents a novel LiDAR Odometry framework, Inc-DLOM, based on a direct point cloud registration method. The framework contains two step pose refinement: a Scan-to-Scan matching for an initial guess and a Scan-to-Mapping refinement method. In the chapter, a lightweight voxel-based mapping storage and query method is also presented. Extensive experiments were conducted on both open-source datasets and customized robot data collections. The result showed that the proposed LO framework can reach state-of-the-art accuracy and efficiency in various sensor working scenarios.

Chapter 4 presents a novel LiDAR Inertial Odometry framework, Invariant-DLIO, based on the Inc-DLOM proposed in Chapter 3 and the theory of Invariant Extended Kalman Filtering (InEKF). The mathematical fundamentals of Lie groups are introduced. The InEKF-based point cloud observations are developed for the sensor fusion work of LiDAR and IMU. The experimental results of Invariant-DLIO are also given in this chapter.

Chapter 5 presents the deployment and extension of our LO and LIO framework on the NVIDIA Jetson Orin Nano platform [41], a resource-constrained embedded computing system. This chapter provides comprehensive technical insights into implementing our CUDA-accelerated Inc-DLOM solution, including the parallel CPU-GPU architectural modifications essential for real-time performance on edge devices.

Chapter 6 concludes the contributions of this thesis, and more future development directions are also given.

Chapter 2

Literature Review

2.1 Point Cloud Registration

Point cloud registration (PCR), the foundational task in LiDAR Odometry (LO) research, solves for the rigid-body transformation aligning two point clouds. During alignment, one point cloud is typically designated as the **target frame** and remains fixed, while the transformation required to align the source frame to the target is estimated. The most classical method of PCR is called Iterative Closest Point (ICP). The algorithm operates iteratively, in each loop, it establishes correspondences (typically nearest neighbors) between points in the source and target clouds, then computes the optimal rigid-body transformation (rotation and translation) minimizing the distance between these correspondences, until the optimization achieves convergence. In the ICP problem, given a target frame, $P = \{p_i\}$, $\forall i = 1, \dots, N$, and a source frame $Q = \{q_i\}$, $\forall i = 1, \dots, N$, the error metric is defined as

$$\mathcal{T} = \operatorname{argmin} \sum ||p_i - (R \cdot q_i + t)||_2^2, \quad (2.1)$$

where the transformation $\mathcal{T}$ has the form of

$$\mathcal{T} = \begin{bmatrix} R & t \\ 0 & 1 \end{bmatrix} \in \operatorname{SE}(3), R \in \operatorname{SO}(3), t \in \mathbb{R}^3. \quad (2.2)$$

Noted that the transformation used in this thesis follows the Lie Theory expression, including the Special Euclidean group, SE(3), and the Special Orthogonal group, SO(3). In the past decades, the quaternion expression has often been used for rotation representation, but the math terms would be over-complex for derivative computing and optimization smoothing. ICP algorithm formulates the basic math definitions for PCR problems, and multiple variants were investigated based on the modifications to the optimization objectives.

This work adopts the Lie group formulation for rigid transformations, operating on the Special Euclidean group SE(3) and its rotation subgroup SO(3). While quaternion representations have been widely used in prior decades, their mathematical complexity impedes efficient derivative computation and optimization smoothness. The Iterative Closest Point (ICP) algorithm establishes the foundational mathematical framework for point cloud registration (PCR), and numerous variants have been developed through modifications to its optimization objectives.

Point-to-Line ICP (PL-ICP) proposed a type of objective as follows:

$$\mathcal{T} = \operatorname{argmin} \Sigma ||\mathbf{n}_i \cdot [p_i - (R \cdot q_i + t)]||_2^2, \quad (2.3)$$

where $\mathbf{n}_i$ is the normal vector to the local surface at the projected target point. Compared with Vanilla ICP's point-to-point metric, PL-ICP is minimizing the sum of distances between the source point to the projected target point surface. The experiments show that PL-ICP can converge in fewer iterations and achieve faster computing speed than the vanilla ICP algorithm.

The Generalized Iterative Closest Point (GICP) algorithm is an enhanced modification of ICP, instead of treating every point in the point set as fixed, GICP considers points as samples from Gaussian distributions on surfaces, which are defined as: $p_i \sim$

$$\mathcal{N}(\mu_p^i, C_p^i), q_i \sim \mathcal{N}(\mu_q^i, C_q^i).$$

$$\mathcal{T} = \operatorname{argmin}_{T \in SE(3)} \sum_i \|q_i - \mathcal{T} \cdot p_i\|_{(C_i^Q + RC_i^Q R^T)^{-1}}^2 \quad (2.4)$$

After introducing the stochastic element to ICP algorithm, the optimization objective (2.1) has been modified to minimizing the Mahalanobis distance between the two distributions instead of the Euclidean distance between points. The new objective has been defined as (2.4), in which R is the rotation matrix to transform source point's covariance into target point cloud's local coordinate frame. Compared to other variants of ICP, GICP has better accuracy and faster convergence speed, which makes it one of the most effective PCR algorithms. Normal ICP (NICP) [69] is a powerful extension of GICP, this method extends the point data to $\mathbb{R}^6$, in which the first three dimensions data are point locations in the 3D Euclidean space and the last three dimensions are normal vectors of local surfaces. With the extended features, NICP is able to perform better than GICP for dense point cloud data, especially under small-scale indoor environments. However, both GICP and NICP are not suitable for real-time registration, due to the computing of local covariance requires extensive nearest neighboring search, which is extremely time-consuming. And they are usually used for offline PCR ground-truth generation. GenZ-ICP [45] proposes a method that combines different error metrics within the ICP algorithm and introduces an adaptive weighting-based optimization strategy, which allows it to achieve good results in degenerate environments. Despite its robustness and accuracy in real-world problems, ICP-based methods are often considered too time-consuming and computationally expensive for real-time robotics localization. IMLS-SLAM [20] is an example with accurate estimation but over 1.2 s per frame processing time, which can not meet real-time localization requirements.

2.2 LiDAR Odometry

Feature-based LiDAR Odometry (LO) methods complement ICP-based approaches. A prominent example is LiDAR Odometry and Mapping (LOAM) [89], which employs curvature-based feature extraction and estimates transformations using selected feature points rather than full point clouds. LOAM defines different cost functions for edge, planar, and corner features for pose estimation. To mitigate accumulated errors from sequential scan-matching, it maintains a historical point cloud map for loop closure and back-end optimization. LOAM's point cloud map data has a range of $250 \times 250 \times 50$ meters, which is a simple and direct mapping storage method. This foundational architecture—feature extraction, scan-matching, and mapping—persists across most contemporary LO methods, whether feature-based or direct.

Over the past decade, different LOAM variants have been developed, the similar feature-based scan-matching paradigm has been retained. LeGO-LOAM [70] introduced a lightweight, ground-optimized adaptation for computationally constrained platforms like small Unmanned Ground Vehicles (UGVs) operating under smooth motion conditions. It first segments raw point clouds via range-image processing into ground and object clusters. Feature extraction then occurs on these segmented subsets, reducing computational load versus LOAM. While retaining LOAM's scan-matching and mapping procedures, LeGO-LOAM implements a two-step Levenberg-Marquardt optimization that accelerates convergence and enhances processing speed. Empirical results confirm its superior accuracy and efficiency for small UGVs.

Beyond LeGO-LOAM, other LOAM-based methods include F-LOAM [80] for accelerated computation, PVE-LIOM [21] enhances feature extraction through 2D scan-image construction, T-LOAM [96] proposed cost functions based on the theory of truncated least squares, etc. ATI-CTLO dynamically adjusts temporal intervals using PCA-based point cloud processing, adapting to motion dynamics and environmental degeneracy [95]. Nevertheless, feature-based methods exhibit reduced robustness in textureless environments where reliable feature extraction is challenging. Their hand-crafted feature extractors also

incur significant computational overhead.

Efficient historical map management remains critical for LO performance. Current methods typically concatenate point cloud frames and employ KD-Trees or Octrees [6, 17] for nearest-neighbor searches. However, these structures degrade in efficiency as maps expand, requiring frequent reconstruction during matching [89]—rendering scan-to-map modules slow and memory-intensive. Thus, developing scalable map management strategies for LO frameworks persists as an open challenge.

Registration-based LO methods have gained recent research traction. HDL-SLAM—an early open-source implementation—performs scan-to-scan matching via user-selectable GICP or NDT algorithms. It maintains a pose graph where nodes (frames) are added upon exceeding rotation/translation thresholds, with edges encoding relative transformations between non-consecutive frames. While offering versatile scan-matching options and integration with Autoware (a prominent autonomous driving simulator), HDL-SLAM lacks robust loop closure detection and map management. Erroneous node/edge insertions consequently degrade pose-graph optimization.

Direct LiDAR Odometry (DLO) [14] leverages GICP-based direct registration with NanoFLANN-accelerated KD-Trees [8]. Its "NanoGICP" variant enhances scan-to-scan and scan-to-map refinement efficiency. To avoid computational bottlenecks in growing maps, DLO implements keyframe selection via point cloud spaciousness (mean distance to origin). This reduces scan-to-map correspondence points to hundreds rather than hundreds of thousands, enabling efficient CPU processing. Unlike range-based maps, DLO's spatially optimized keyframes provide broader overlap with current scans, improving refinement quality. However, computational constraints preclude motion distortion compensation, limiting applicability in high-frequency motion scenarios.

KISS-ICP simplifies classical point-to-point ICP by minimizing parameterization [77]. While typical ICP-based LO methods, e.g., MULLS (107 parameters) [60], SuMa (49 parameters) [4], CT-ICP (30 parameters) [19] require extensive tuning for generalization, KISS-ICP adaptively sets the parameters including voxel sizes, distance thresholds, con-

vergence criteria, etc., achieving cross-dataset robustness with fixed parameters. It incorporates an adaptive robust kernel and stores map data in hash tables, where each $v \times v \times v$ voxel retains $\leq N_{max}$ points. KISS-ICP achieves high efficiency by avoiding additional computations, such as normal vector estimation or local covariance calculations. However, KISS-ICP’s performance can still be corrupted by the dataset with significant noises and moving objects.

2.3 LiDAR Inertial Odometry

Integrating the Inertial Measurement Unit (IMU) data, including acceleration and rotational speed, is a natural complement for LO methods, the integration of IMU data can provide a reasonable initial guess for Scan-Matching problem [27]. Out of the purpose, many LO methods also support integrating IMU data, e.g., LOAM [89], LeGO-LOAM [70], and HDL-SLAM [43]. This LiDAR-IMU fusion method is also known as loosely-coupled LIO, which achieves pose level fusion of different sensor types [16, 47, 11]. While this decoupling offers computational efficiency and implementation ease, it inherently suffers from suboptimal fusion due to ignored correlations between raw sensor measurements, leading to accumulated drift during aggressive motion or in geometrically degenerate environments (e.g., tunnels, open skies). These limitations underscore the necessity of tightly-coupled LIO, which directly fuses raw LiDAR point clouds and IMU measurements within a unified optimization/filtering framework.

LIO-SAM (Lidar Inertial Odometry via Smoothing and Mapping) is a state-of-the-art tightly-coupled LIO framework that leverages a factor graph optimization back-end to achieve high-precision odometry and mapping [71]. It fuses raw LiDAR point clouds, IMU data, and GPS measurements into a unified optimization framework, where different factors, including IMU preintegration, LiDAR Odometry, GPS constraints, and loop-closure detections are all kept inside the factor graph for optimization. The LiDAR Odometry factors is generated based on the edge and planar cost functions similar with

the ones in LOAM, and the correspondences are searched by a fixed number of recent LiDAR point clouds, which construct a sub-keyframe point mapping. The back-end relies on incremental smoothing and mapping with the Bayes tree (iSAM2) while inserting new nodes. The implementation of LIO-SAM is purely open-source, with versatile modules for future development, and different experiments show that LIO-SAM can reach accurate performance for different LiDAR point cloud dataset. While LIO-SAM's open-source implementation demonstrates versatility and achieves high accuracy across different LiDAR datasets, its computational efficiency is not comparable to filtering-based LIO approaches, due to the computational load from graph optimization. Additional limitations include the requirements for 9-axis IMU measurements and primary validation with the VLP-16 LiDAR, restricting its adaptability to various robotic platforms and operational environments. Similarly, LIOM [87] uses the factor graph optimization based on the measurement of edge and plane points, and the IMU preintegration is also applied as the factor in the graph. LIO-mapping [94] is based on feature points measurement and IMU preintegration, which constructs a sliding-window method for graph optimization.

Compared to optimization-based LIO, such as LIO-SAM, methods based on filtering theory can achieve outstanding computational efficiency [52]. Filtering methods fuse high-rate IMU measurements (100-200 Hz) for prediction and lower-rate LiDAR point cloud for correction, avoiding computationally expensive batch optimization, like pose graph [59] or bundle adjustment (BA) [53, 58]. The smoothing-based methods have large computational complexity, often $\mathcal{O}(n^3)$ with the problem size, which can cause severe delay at LIO methods of thousands of point cloud measurements. Extended Kalman Filtering (EKF) is widely used for LIO methods [38], for its outstanding computational efficiency, natural fit for IMU propagation, and robust accuracy behavior, despite suffering from linearization errors and accumulating drift. LINS designed an iterated error-state Kalman Filter (ESKF) for state correction recursively by generating new feature correspondences in each iteration, and the feature extraction and mapping modules are based on LeGO-LOAM. LINS represent the robot state with the local centric formulation, the

state consists rotation (as quaternion), velocity, translation, IMU biases, and gravity direction, which forms a 18 Degree-of-Freedom (DoF) estimation problem. The iterated updated manner ensures LINS has good efficiency on the state correction step. However, the implementation of LINS is naive, all point cloud measurements are stacked to form a batch matrix, and requires time-cost matrix operations like inversion and multiplication. Such behaviors make LINS only achieve acceptable computation speed on VLP-16, a type of LiDAR producing extremely sparse point cloud data.

Fast-LIO (Fast LiDAR-Inertial Odometry) is a high-performance, filtering-based state estimation algorithm designed for real-time robotic navigation [85]. Fast-LIO formulates an ESKF directly on the $SE(3)$ manifold, ensuring numerical stability and reducing linearization errors through iterative updates during LiDAR scan registration. Incremental KD tree (ikd-Tree) is proposed for point cloud mapping storage in Fast-LIO, which enables dynamic map management with amortized constant-time complexity for point insertion, deletion, and nearest neighbor search. This tight coupling of an accurate manifold-based filter with an ultra-efficient map representation allows Fast-LIO to process dense LiDAR point clouds at high frequencies (e.g., >100 Hz for state estimation) with minimal latency, even on resource-constrained platforms, while maintaining low drift and robustness in geometrically degenerate environments. Its efficiency eliminates the need for computationally expensive keyframe selection or global optimization, which is typical of smoothing-based approaches.

2.4 Invariant Extended Kalman Filtering (InEKF)

Traditional EKF design often relies on the assumption that the estimation error is sufficiently small, leading to possible convergence failure when the estimation is far from its true state. Besides, the eigenvalues of the Kalman covariance matrix are also assumed to be bounded by strictly positive scalars. The non-trivial assumptions can make EKF diverge or provide non-optimal estimation. As the complement method, InEKF is built

upon symmetry-preserving observer theory, which was firstly proposed by Bonnabel [9]. Symmetry-preserving observer is a type of non-linear observer that maintains the symmetries inherent in the system's equations. The maintaining of system's symmetries can provide the invariance into estimation error dynamics under the changes corresponding to different Lie groups. The invariance property was later introduced into the implementation of the InEKF, where the usage of the Lie group representation with the symmetry property has also been recognized. The outstanding convergence and consistency performance of InEKF was first presented in [3]. Zhang et al. [92] analyzed the convergence and consistency of the InEKF SLAM with improved system performance. An InEKF algorithm based on contact point observation is proposed for legged-robot state estimation [33], where the observability was analyzed and the math formulation for different observations was also developed. Shi et al. proposed an InEKF and feature based LiDAR Inertial Odometry method [72]. Although InEKF theory has been rigorously proven from a mathematical point of view in previous works, and there are also applications for legged-robot state estimation[33], Inertial Navigation System (INS) [40], and Visual-Inertial Odometry (VIO) [54, 37], the InEKF-based LIO framework has not been fully investigated.

Chapter 3

Incremental Direct LiDAR Odometry and Mapping

The contents of this chapter are mostly from the paper [25]. Minor modifications were made, most of the contents and results are remaining the same.

3.1 Introduction

In most public LiDAR localization datasets, researchers have focused more on data under autonomous driving scenarios, e.g., KITTI [29], Oxford RobotCar [55], MulRan [42], etc., where the robotic platforms are usually commercial vehicles with smooth motion and non-drastic acceleration. Concurrently, LiDAR is also used to localize robot platforms with vigorous rotations and abrupt acceleration changes, like the Segway robot of NCLT [10]. Most existing LO methods do not perform well in these challenging environments, mainly because most of them do not consider point cloud distortions within their frameworks and appropriately address such issues. The method proposed in [81] utilized modified Fast-VGICP [44] and pose graph optimization, achieved good performance for different autonomous driving scenarios. However, the performance for the high-frequency robotic platform is still worth for improvement. Moreover, solid-state LiDARs, such as the Intel Realsense L515, possess different scanning paradigms and produce data characteristics

that diverge substantially from traditional mechanical LiDAR [79]. This discrepancy renders existing LO methods for mechanical LiDARs incompatible with solid-state LiDARs. Consequently, the generalization of LO approaches for different working scenarios and LiDAR types is worth investigating.

3.2 Proposed Method

In this section, the details of the proposed LO are presented. Inc-DLOM takes the point cloud $\mathcal{P}_k$ as input, and goes through the procedure of **Preprocess**. Then the deskewed and downsampled point cloud is utilized for **S2S matching** to compute the relative *S2S transformation (Scan-to-Scan Transform)* $\Delta\mathcal{T}$. In the procedure of **S2M matching**, $\mathcal{P}_k$ is generated by the *Voxelization* step, and taken by *V2V GICP*. The proposed *Incremental Voxel Mapping (IVM)* $\mathcal{M}$ will be updated by *Global Voxel Information* $\mathcal{P}_k^*$. The global pose $\mathcal{T}_k$ is computed by the proposed *Incremental GICP* method with the initial guess $\mathcal{T}_k^0$.

For the k -th point cloud input, noted as,

$$\mathcal{P}_k = \{\cdots, p_i, \cdots \mid \forall p_i \in \mathbb{R}^3\} \quad (3.1)$$

Inc-DLOM is estimating the global pose $\mathcal{T}_k$. $\forall \mathcal{T} \in \text{SE}(3)$ has the general form of

$$\mathcal{T} = \begin{bmatrix} \mathcal{R} & t \\ 0 & 1 \end{bmatrix} \quad (3.2)$$

where $\mathcal{R} \in \text{SO}(3)$ represents the rotation matrix, $t \in \mathbb{R}^3$ represents the translation vector. The point p can be transformed by $\mathcal{T}$ as,

$$\mathcal{T} \cdot p := \mathcal{R} \cdot p + t \quad (3.3)$$

These definitions will be adopted throughout the paper.

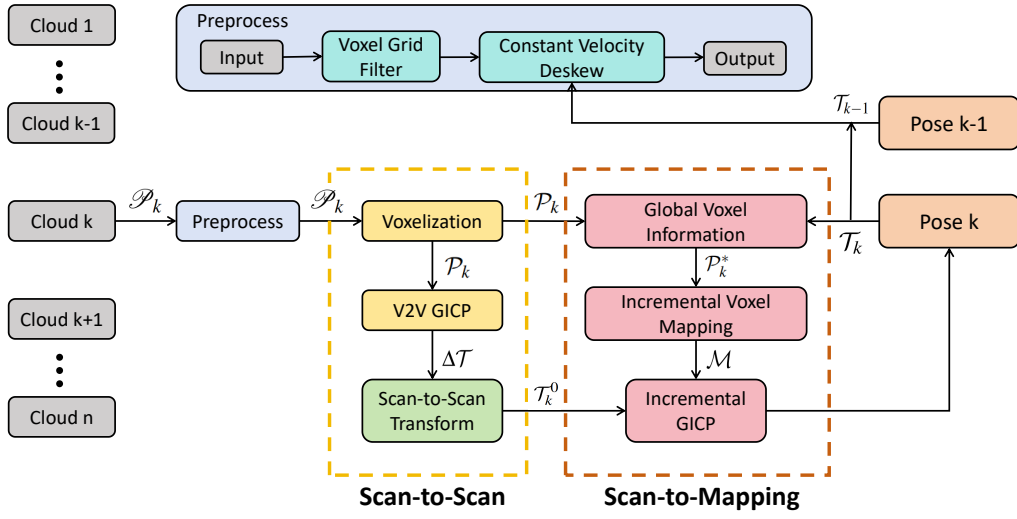


Figure 3.1: Inc-DLOM Framework.

3.2.1 Preprocess

The point cloud will first be preprocessed, including the downsampling by Voxel Grid Filter and the point cloud deskew with the constant velocity model, shown in the light blue part in Fig. 3.1. The point cloud data is recorded continuously while the robotic platforms perform ego-motions. During point cloud data acquisition, severe accelerations and rotations will cause LiDAR point cloud distortions and lead to registration errors. In our proposed LO method, point cloud deskew with the constant velocity model is also included in the Preprocess part.

For $\mathcal{P}_k$, the relative motion is defined as (5.4), which is derived from $\mathcal{T}_{k-1}$ and $\mathcal{T}_k$.

$$\Delta\mathcal{T} = \mathcal{T}_{k-1}^{-1} \cdot \mathcal{T}_k \quad (3.4)$$

Then the motion velocity $\xi \in \mathfrak{se}(3)$ is derived from (5.4) as follows:

$$\xi = \frac{\text{Log}(\Delta\mathcal{T})}{\Delta\tau} = \begin{bmatrix} \omega \\ \nu \end{bmatrix} \quad (3.5)$$

Where $\Delta\tau$ is the relative time of single point cloud sensing, which is usually around 0.1 s for mechanical LiDAR. The operator $\text{Log}(\cdot): \text{SE}(3) \rightarrow \mathbb{R}^6$ is the matrix logarithm for

SE(3). ω and ν represent the lie algebra for the rotation and translation parts respectively.

For the typical mechanical LiDAR sensing paradigm, points in one point cloud are measured at different relative timestamps. For $p_i \in \mathcal{P}$, the data is measured with a relative timestamp $s_i \in [0, \Delta\tau)$ from the first measurement. Then $\forall p_i \in \mathcal{P}_k$ is deskewed as:

$$p_i = \text{Exp}(s_i \cdot \xi) \cdot p_i \quad (3.6)$$

where the operator $\text{Exp}(\cdot) : \mathbb{R}^6 \rightarrow \text{SE}(3)$ is the matrix exponential for $\mathfrak{se}(3)$.

3.2.2 Scan-to-Scan (S2S) Matching

After the preprocess step, the k -th cloud data $\mathcal{P}_k$ is utilized to compute the relative transform with the previous cloud, which is denoted as $\mathcal{T}_{k-1}^k$. The process of Scan-to-Scan (S2S) matching consists two steps, voxelization and Voxel-to-Voxel GICP, for computing the voxels information and $\mathcal{T}_{k-1}^k$ respectively.

3.2.2.1 Voxelization

$\mathcal{P}_k$ will be transformed from a point set into a voxel set $\mathcal{P}_k$, defined as (5.11).

$$\mathcal{P}_k \rightarrow \mathcal{P}_k = \{\cdots, \mathcal{V}_i, \cdots\} = \bigcup_{i=0}^n \mathcal{V}_i, \quad \mathcal{V}_i = \{\mathcal{C}_i, p_i\} \quad (3.7)$$

In which, n is the size of $\mathcal{P}_k$, $\mathcal{V}_i$ is the i -th voxel in the point cloud data. $\mathcal{V}_i$ contains the local surface covariance $\mathcal{C}_i \in \mathbb{R}^{3 \times 3}$ and the coordinate $p_i \in \mathbb{R}^3$. It is worth noting that the number of voxels is much less than the original number of points since p_i that belongs to the subset of $\mathcal{P}_k$ is selected by Approximate Voxel Filter [67]. $\mathcal{C}_i$ is computed from nearby points searched by KD-Tree. In our implementation, NanoFLANN is used for improving the computational computation efficiency [8].

3.2.2.2 Voxel-to-Voxel (V2V) GICP

Other than aligning the points correspondences mentioned in the original GICP paper [68], the key idea of V2V GICP is the registration of the target set $\mathcal{P}^T$ and source set $\mathcal{P}^S$, where the optimization problem is defined as (3.8). $\mathbb{F}(\cdot, \cdot)$ is the cost function taking two voxels as inputs, $\mathcal{V}_i^S \in \mathcal{P}^S$ and its closest neighboring voxel $\mathcal{V}_i^T \in \mathcal{P}^T$. Σ is the voxel-to-voxel covariance defined in (5.13), in which $\mathcal{C}_i^T$ and $\mathcal{C}_i^S$ are voxel covariances of $\mathcal{V}_i^T$ and $\mathcal{V}_i^S$ respectively.

$$\mathcal{T}^* = \operatorname{argmin}_{\mathcal{T} \in \text{SE}(3)} \sum_i \mathbb{F}(\mathcal{V}_i^T, \mathcal{V}_i^S) = \operatorname{argmin}_{\mathcal{T} \in \text{SE}(3)} \sum_i \|p_i^T - \mathcal{T} \cdot p_i^S\|_{\Sigma^{-1}}^2 \quad (3.8)$$

$$\Sigma = \mathcal{C}_i^M + \mathcal{R} \cdot \mathcal{C}_i^S \cdot \mathcal{R}^T \quad (3.9)$$

3.2.3 Scan-to-Mapping (S2M) Matching

The proper mapping construction and storage method is critical for LO's performance since the pose estimation often requires scan-to-mapping matching to eliminate the accumulated error. However, the usage of local point mapping in the direct LO is often considered as more memory and time-consuming than the one in feature-based methods due to the amount of data. To address this issue, we propose a novel scan-to-mapping scheme that is computationally efficient and can support lightweight, real-time mapping access.

3.2.3.1 Global Voxel Information

As shown in Fig. 3.1, the voxel information (5.11) is recycled for the S2M scheme, which is transformed into the global frame to construct the historical mapping with the pose $\mathcal{T}_k$. The global voxel $\mathcal{V}_i^*$ is calculated as

$$\mathcal{V}_i^* = \mathcal{T}_k \cdot \mathcal{V}_i := \{\mathcal{R}_k \cdot \mathcal{C}_i \cdot \mathcal{R}_k^T, \mathcal{R}_k \cdot p_i + t_k\} = \{\mathcal{C}_i^*, p_i^*\} \quad (3.10)$$

3.2.3.2 Incremental Voxel Mapping

It is apparent that the size of historical mapping will grow as time passes for the LiDAR Odometry. Most keyframe-based LO methods build the keyframes based on self-motions. For general cases, the thresholds of rotation radians and translation distances are set for keyframe insertion to the historical mapping. Even if efficient data structures like KD-Tree can be used for accessing the points, the operations are still time-consuming and computationally intensive, because keyframe insertion requires continuous data structure re-built. For example, LOAM-based methods are often not able to support real-time scan-to-mapping operation for every incoming point cloud data, even though the mapping cloud is processed with voxel grid downsampling. Besides, the self-motion thresholds are hard to set without extensive try-outs.

Incremental Voxel Mapping (IVM) is implemented based on the spatial hash table to insert and update the historical voxel mapping incrementally to address the mentioned issues. Voxels are stored in the hash table and every one of them is associated with the spatial key. To access one specific voxel, only p_i^* is needed to compute the spatial key as

$$\text{key}(\mathcal{V}_i^*) = \text{floor}(p_i^*/S_{IVM}) \quad (3.11)$$

where S_{IVM} is the pre-defined voxel size parameter of IVM. Since the hash mapping of the data is unique, $\mathcal{V}_i^*$ can be accessed with $\mathcal{O}(1)$ time complexity, which is essential for efficiency. The hash function is defined as

$$h(p_i^*) = h(x, y, z) = ((x * a) \text{ xor } (y * b) \text{ xor } (z * c)) \% d \quad (3.12)$$

$$a = 73856093, b = 471943, c = 83492791, d = 10000000$$

where the input of $h(\cdot)$ is the 3-dim spatial voxel coordinate containing x, y, z , xor is exclusive OR, and $\%$ is the modulus operator. By the hash function in (5.7), every incoming set of $\mathcal{V}_i$ can be managed and stored in the hash table individually, which allows efficient

access in the following steps.

In our proposed method, IVM is continuously inserting the incoming set of $\mathcal{V}_i^*$ to expand the historical mapping, which is denoted as

$$\mathcal{M} = \sum_{j=0}^M \bigcup_{i=0}^n \mathcal{V}_i^* = \bigcup \mathcal{V}_{\mathcal{M}}, \quad (3.13)$$

where M is the number of point cloud frames, and the sum operation means the concatenation of different voxel sets. While expanding $\mathcal{M}$, the voxels to be inserted are classified into two types, new voxels $\mathcal{V}_n^*$ and historical voxels $\mathcal{V}_h^*$, depending on whether $\text{key}(\mathcal{V}^*)$ is inserted into IVM before. All $\mathcal{V}_n^*$ will be inserted into $\mathcal{M}$ with the hash table process. $\mathcal{V}_h^*$ will be merged into the voxel with the same spatial key, which is denoted as $\mathcal{V}_{\mathcal{M}}$. The voxel merging operation is noted as $\mathcal{V}_r^* = \mathcal{V}_h^* \oplus \mathcal{V}_{\mathcal{M}}$, which is defined as the follows:

$$p_r^* = \frac{p_h^* + p_{\mathcal{M}}}{2} \quad (3.14)$$

$$\begin{aligned} \mathcal{C}_r^* = & (\mathcal{C}_{\mathcal{M}} + \mathcal{C}_h^* + (p_{\mathcal{M}} - p_r^*)(p_{\mathcal{M}} - p_r^*)^T \\ & + (p_h^* - p_{\mathcal{M}})(p_h^* - p_{\mathcal{M}})^T)/2 \end{aligned} \quad (3.15)$$

The operation allows IVM to update $\mathcal{M}$ incrementally without storing the points data in voxels, which needs a large amount of memory usage as the mapping expands, as the traditional keyframe-based LO methods. Due to the parallelism of IVM, multi-processing can be utilized in our implementation to enhance computational efficiency. To illustrate the effect of the mapping procedure in Inc-DLOM, Fig. 4.6 shows the visualization of IVM in the KITTI 00.

Fig. 4.6 (a) shows the mapping of the first 150 point clouds without processing, and (b) shows the IVM of the mapping, from which we can see that the IVM is ranging the exact distance as the original point cloud mapping. Table 4.1 shows the statistical comparison of the original point mapping and IVM, the voxel size of IVM is 1.50m, which gives rise to the best estimation result of Inc-DLOM in the KITTI Dataset [29]. Through the Incre-

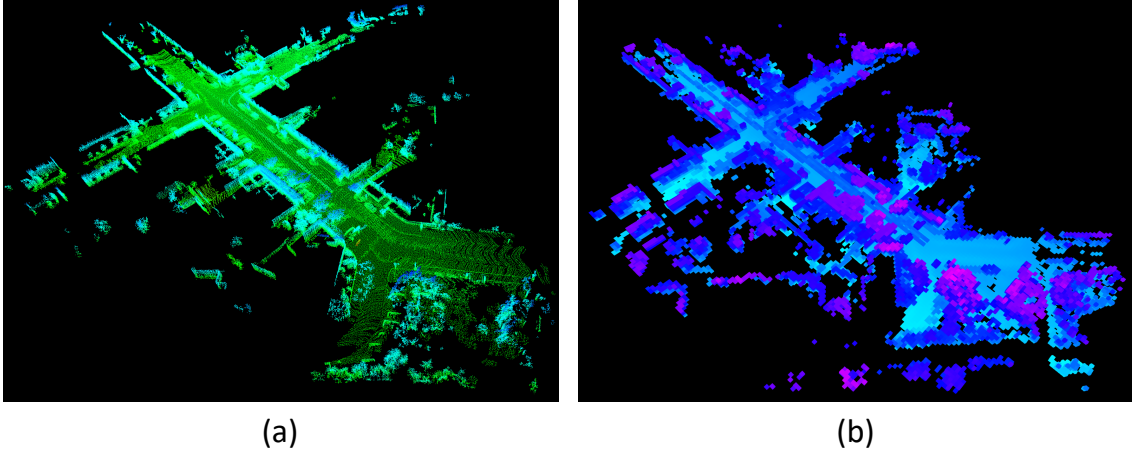


Figure 3.2: **The visualization of IVM.** (a) The plot of the point cloud mapping. (b) The plot of IVM on the mapping.

mental Voxel Mapping process, Inc-DLOM reduces the size of the mapping by hundreds of times, which is the reason why the scheme becomes extremely lightweight and fast. The IVM procedure proposed in this paper can preserve a huge amount of historical point cloud information with only limited memory costs. This property is essential for pose estimation since historical mapping will be used for Scan-to-Mapping pose refinement to eliminate accumulated drifting from Scan-to-Scan alignment, and IVM can maintain historical information to an extremely large range with reasonable memory costs.

Table 3.1: Statistical Results of the IVM.

	Point Mapping	Inc-Voxel Mapping
Points / Voxels No.	2535005	15852
Memory Usage	29.01 MB	0.18 MB

3.2.3.3 Incremental GICP

Incremental GICP is the core part of the S2M matching scheme, which is described in Algorithm 1. The algorithm takes the historical mapping described in Subsection 4.2.5.1, voxel set described in Subsection 5.3.3.1, and the initial guess $\mathcal{T}_k^0 \in \text{SE}(3)$ as inputs. $\mathcal{T}_k^0$ is computed from the previous pose and relative S2S transformation as (5.14).

$$\mathcal{T}_k^0 = \mathcal{T}_{k-1} \cdot \Delta\mathcal{T} \quad (3.16)$$

The difference between Incremental GICP and V2V GICP is mainly on lines 5-8, where getting $\mathcal{V}_{\mathcal{M}}$ by $\text{key}(\mathcal{V}_i)$ is the $\mathcal{O}(1)$ operation in IVM hash table instead of KD-Tree searching, and the cost function will be constructed if the voxel associated with $\text{key}(\mathcal{V}_i)$ can be searched in IVM.

The cost function of Incremental GICP is defined as (4.60), which is solved by Gauss-Newton iteratively.

$$\mathcal{T}_k = \underset{\mathcal{T} \in \text{SE}(3)}{\text{argmin}} \sum_i \mathbb{F}(\mathcal{V}_{\mathcal{M}}, \mathcal{V}_i) = \underset{\mathcal{T} \in \text{SE}(3)}{\text{argmin}} \sum_i \|p_{\mathcal{M}} - \mathcal{T} \cdot p_i\|_{\Sigma^{-1}}^2 \quad (3.17)$$

$$\Sigma = \mathcal{C}_{\mathcal{M}} + \mathcal{R} \cdot \mathcal{C}_i \cdot \mathcal{R}^T$$

A_i, b_i mentioned in line 9 of Algorithm 1 is the general format of the nonlinear least-square problem of $Ax = b$, where matrix L is the Cholesky square root of matrix Σ so that $\Sigma = L \cdot L^T$.

$$J_i = L^{-1} \cdot \begin{bmatrix} [\mathcal{T} \cdot p_i]_{\times}, & -I_{3 \times 3} \end{bmatrix} \quad (3.18)$$

$$A_i = J_i^T \cdot J_i, A_i \in \mathbb{R}^{6 \times 6} \quad (3.19)$$

$$b_i = -J_i^T \cdot L^{-1} \cdot (p_{\mathcal{M}} - \mathcal{T} \cdot p_i), b_i \in \mathbb{R}^{6 \times 1} \quad (3.20)$$

Table 3.2: Detailed Time Costs of Subprocedures in Inc-DLOM.

	Preprocess		S2S Matching		S2M Matching	
Procedures	Down-sampling	Points Deskew	Covs Computing	Optimization	IVM Updates	Optimization
Time (ms)	1.554	6.538	4.128	0.870	0.003	0.872

Incremental GICP allows the proposed LO to perform fast scan-to-mapping refinement, which improves the accuracy of the direct point cloud registration, while barely increasing the computation time. Table 3.2 shows the computational time cost for every procedure in Inc-DLOM for a single LiDAR point cloud frame measured in milliseconds (ms), here the voxel size in IVM is 1.0m, which is verified by the best accuracy test on

the NCLT Dataset [10]. The results are tested by the first 4000 frames in **NCLT_1**, mentioned in Section 3.3. From Table 3.2, we can see that the procedure of S2M matching only increases the computation time by less than 1 ms, which is negligible for real-world applications.

Algorithm 1: Procudure of Incremental GICP.

Input: historical mapping $\mathcal{M}$, point cloud voxels $\mathcal{P}_k$, initial guess $\mathcal{T}_k^0$
Output: global pose $\mathcal{T}_k$

```

1  $\mathcal{T}_k \leftarrow \mathcal{T}_k^0$ ;
2 while  $\mathcal{T}_k$  not converged do
3    $A \leftarrow 0^{6 \times 6}$ ,  $b \leftarrow 0^{6 \times 1}$ ;
4   for  $i$  in  $\{0, \dots, n\}$  do
5      $\mathcal{V}_{\mathcal{M}} \leftarrow \text{getVoxel}(\mathcal{M}, \text{key}(\mathcal{V}_i))$ ;
6     if  $\mathcal{V}_{\mathcal{M}} == \emptyset$  then
7       continue
8     end
9      $A_i, b_i \leftarrow \text{cost}(\mathcal{V}_{\mathcal{M}}, \mathcal{V}_i)$ ;
10     $A \leftarrow A + A_i$ ,  $b \leftarrow b + b_i$ ;
11  end
12   $\delta T \leftarrow \text{GaussNewton}(A, b)$ ;
13   $\mathcal{T}_k \leftarrow \mathcal{T}_k \cdot \delta T$ 
14 end
15 return  $\mathcal{T}_k$ 

```

3.3 Experimental Results

3.3.1 Experimental Setup

To evaluate Inc-DLOM's performance and generalization, we firstly performed experiments on multiple public datasets with different working scenarios, the evaluation goes on both accuracy and efficiency. Starting with the KITTI Odometry Benchmark [29], the classical dataset for the scenarios of autonomous driving localization. To test Inc-DLOM's performance on challenging scenarios, we evaluate the performance with the NCLT Dataset [10] and Newer College Dataset (NCD) [64], which are collected by a Segway robot and hand-held device of high-frequency self-motions. Fig. 3.5 shows the

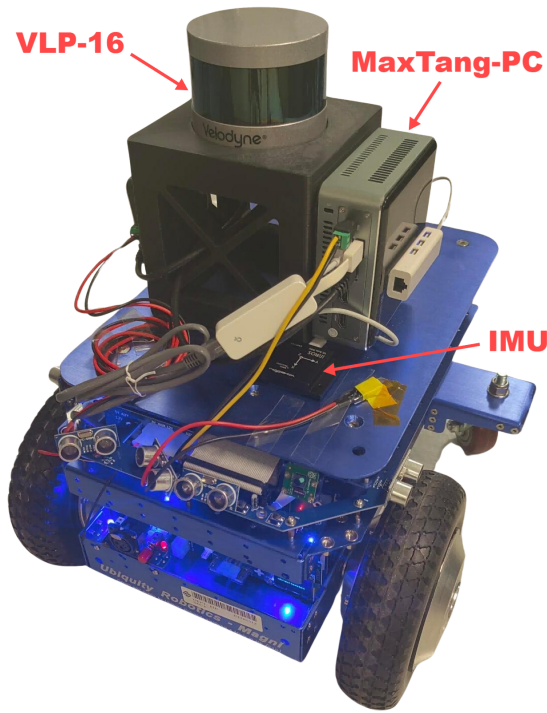


Figure 3.3: Data Collection Robotic Platform

trajectories estimated by Inc-DLOM in different data sequences.

In addition, we also collected the custom dataset with our robotic platform, deployed with VLP-16 LiDAR, IMU, MaxTang FP650 NUC [57], shown in Fig. 4.2. The data acquisition scenario is in the corridor of PolyU buildings. The selected corridors lack identifying objects and are filled with plain walls, repetitive doors and windows, as shown in Fig. 5.14.



Figure 3.4: Textureless Corridor View

The chassis is a Magni Silver robot [61], a typical differential drive robot. The control is highly dependent on the feedback from wheel encoders, where the measurements are noisy and easy to interfere by slippery, which makes the robot containing uneven motions even with certain control inputs. Both the textureless environment and irregular high-frequency motions formed a challenging working scenario.

Besides, our proposed Inc-DLOM is tested with the data collected by Intel RealSense L515 [79], which has a different scanning scheme than traditional mechanical LiDARs. The performance on solid-state LiDAR is essential for the generalization evaluation. Finally, the computational efficiency is also evaluated.

For all aspects of the experiments, we compare our Inc-DLOM with different state-of-the-art open-source LO systems, including feature-based LO methods: Fast-LIO [85], LINS [62], LIOM [87], and direct LO method: HDL-SLAM [43] and Direct LiDAR Odometry (DLO) [14]. All of the experiments are conducted with a platform based on an AMD Ryzen 9 7940HS CPU with 4.00 GHz frequency.

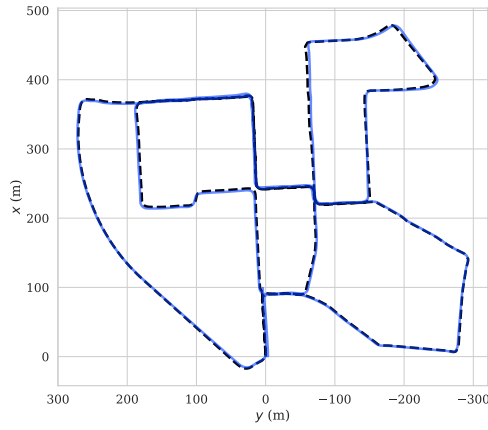
3.3.2 Autonomous Driving Scenario

The evaluation is firstly measured based on the error metrics defined by the KITTI Odometry Benchmark as translational drifting error in percentage and rotational drifting error in degree of 100 m for all possible subsequences of length (100,...,800) meters. Table 4.2 shows the evaluation results, the best results among the considered methods for each sequence are highlighted with black bold.

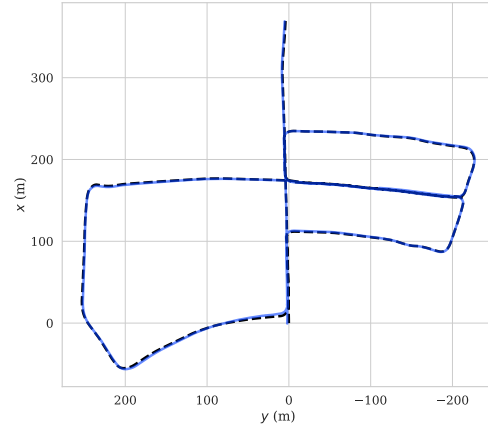
We also compared the Absolute Trajectory Error (ATE) [30], which is a more direct and intuitive error metric for localization performance evaluation. ATE is defined as (4.61),

$$ATE_i = \mathcal{T}_{est,i}^{-1} \cdot \mathcal{T}_{ref,i} \quad (3.21)$$

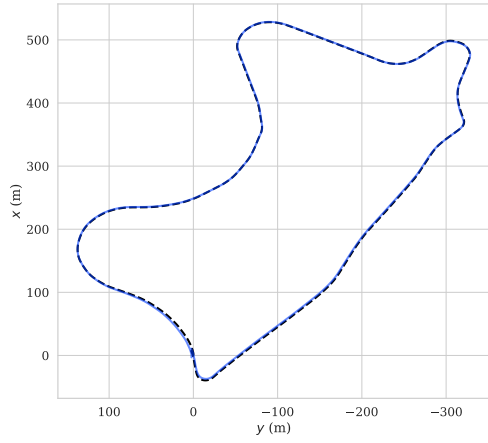
where $\mathcal{T}_{est,i}$ refers to the pose estimated and $\mathcal{T}_{ref,i}$ is the ground truth, and the root mean squared error (RMSE) is computed by (4.62). Noted that the RMSE value computed refers



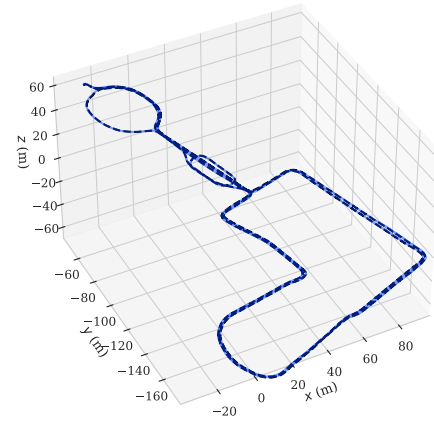
(a) KITTI-00



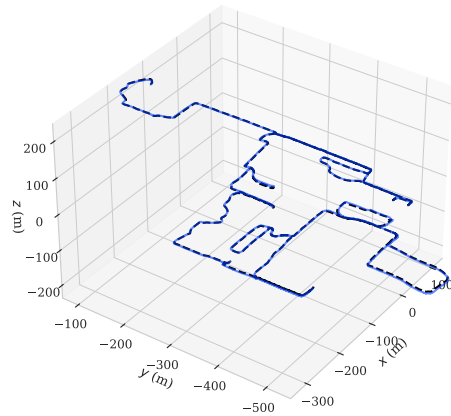
(b) KITTI-05



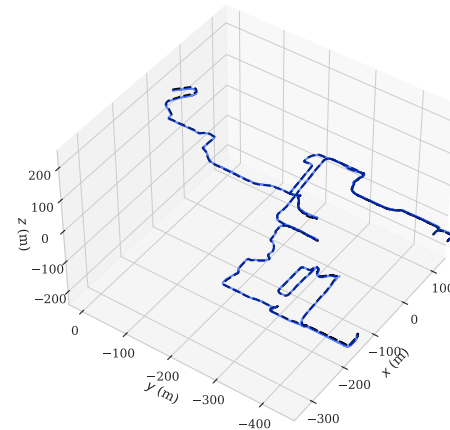
(c) KITTI-09



(d) NCD-1



(e) NCLT-2



(f) NCLT-3

Figure 3.5: Inc-DLOM Trajectories on selected dataset sequences. The figure shows the estimated trajectories of dataset sequences, including multiple LiDAR types and different working scenarios. The plot of ground truths (dashed line) and estimated trajectories (blue) shows good estimation performance of our proposed method.

Table 3.3: Translational Drifting Error (%) / Rotational Drifting Error (deg /100m) Evaluation of KITTI Dataset

	00	02	05	06	07	08	09	10
Inc-DLOM	0.58 / 0.33	0.94 / 0.35	0.38 / 0.24	0.23 / 0.14	0.14 / 0.16	0.99 / 0.39	0.28 / 0.23	0.22 / 0.26
Fast-LIO	3.24 / 1.35	1.84 / 0.55	0.51 / 0.26	1.34 / 0.48	0.46 / 0.29	2.68 / 1.07	0.36 / 0.34	0.51 / 2.07
LINS	8.46 / 3.45	10.08 / 3.04	7.64 / 3.56	1.12 / 0.86	4.29 / 2.12	8.75 / 3.53	11.49 / 3.60	9.35 / 3.27
HDL-SLAM	3.04 / 1.05	9.29 / 2.72	0.80 / 0.46	0.89 / 0.29	0.72 / 0.95	2.45 / 0.92	2.35 / 0.71	1.11 / 0.82
LIOM	9.75 / 2.37	18.85 / 5.08	9.75 / 2.37	17.24 / 6.33	0.20 / 0.48	21.69 / 9.06	10.96 / 6.54	4.75 / 2.60
DLO	2.37 / 0.53	6.36 / 0.94	1.50 / 0.43	0.48 / 0.22	0.95 / 0.49	4.96 / 1.51	3.12 / 0.46	4.06 / 0.74

to the translational part of ATE.

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=0}^N ATE_i^2} \quad (3.22)$$

Table 3.4: RMSE of ATE (m) Evaluation of KITTI Dataset

	00	02	05	06	07	08	09	10
Inc-DLOM	3.85	7.21	2.51	1.43	1.57	7.91	2.32	1.45
Fast-LIO	42.49	12.41	3.59	9.30	2.62	17.21	2.69	4.80
LINS	43.47	85.87	4.40	7.56	22.91	22.51	N.A.	52.62
HDL-SLAM	22.44	N.A.	4.96	5.68	4.41	19.26	21.15	5.58
LIOM	85.43	N.A.	57.90	N.A.	1.18	N.A.	69.19	22.60
DLO	5.68	8.04	2.58	1.64	1.82	21.61	6.92	2.35

We compared the translational part of the RMSE of ATE, shown in Table 4.3, among which the best results are highlighted with bold texts. From the evaluation, we can see that Inc-DLOM achieves the best performance in almost all sequences both in the drifting and ATE evaluations. Although LIOM can get better performance on Sequence 07 in the ATE evaluation, it is not comparable with Inc-DLOM in both drifting and ATE in all other sequences. Overall, Inc-DLOM is outperforming all chosen state-of-the-art LO methods.

3.3.3 High Frequency Motion Scenario

For robotic platforms with high-frequency self-motions, SLAM algorithms on autonomous driving scenarios may not be able to perform effectively, which makes it important to evaluate the SLAM performance in this aspect. The NCLT dataset is one of the typical datasets collected by a Segway robot with Velodyne HDL-32 LiDAR [10]. We have selected five sequences from the NCLT dataset, NCLT_1 (20130110), NCLT_2 (20120615), NCLT_3 (20120429), NCLT_4 (20120115), and NCLT_5 (20120511). The Newer College Dataset is collected by hand-held OS-128 LiDAR with long trajectories and high-frequency rotations [64], which is similar to the NCLT Dataset. We selected four sequences from the Newer College Dataset, NCD_1 (2020-03-10), NCD_2 (2021-11-30), NCD_3 (math-1), and NCD_4 (math-2). The evaluation of NCLT and NCD is shown in Table 4.4. From the table, we can see that our proposed Inc-DLOM outperforms all other benchmarks. LIOM and LINS are typical methods that fail on robotic platforms with high-frequency self-motions, while HDL-SLAM and DLO are direct LO methods for comparison.

Table 3.5: RMSE of ATE (m) Evaluation of NCLT Dataset and Newer College Dataset

	NCLT_1	NCLT_2	NCLT_3	NCLT_4	NCLT_5	NCD_1	NCD_2	NCD_3	NCD_4
Inc-DLOM	0.71	1.93	1.47	2.74	2.80	0.32	0.34	0.09	0.16
Fast-LIO	0.87	2.35	1.61	2.74	3.13	1.28	1.06	0.31	0.49
LINS	28.30	151.22	fail	fail	fail	fail	fail	0.86	1.14
HDL-SLAM	28.93	38.48	82.29	fail	fail	2.57	1.64	0.97	2.47
LIOM	fail	fail	fail	fail	fail	fail	fail	0.48	2.08
DLO	1.56	2.37	7.26	11.84	4.72	0.43	0.96	0.17	0.31

In addition, we also evaluated our method on an extremely challenging dataset sequence, NCD-Math-Hard, which is full of rapid rotations in different locations and easy to lead tracking failure for SLAM algorithms. Fig. 3.6 (i) shows the estimated trajectory of Inc-DLOM (blue), in comparison with ground truth (dashed) and DLO result (orange), where the other selected methods failed. From the figure, it is obvious that DLO can track the motions in the beginning, however, the performance is affected when the LiDAR is conducting rapid rotations. On the contrary, our proposed Inc-DLOM can achieve robust performance even with vigorous motions and the trajectory is highly consistent with the

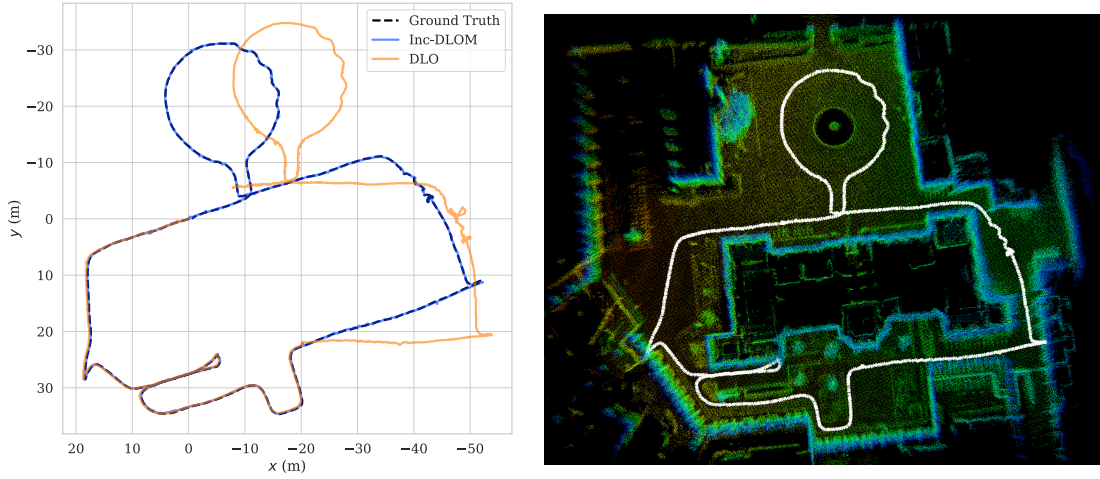


Figure 3.6: **Performance on NCD-Math-Hard.** (Left) Estimated trajectory (blue), comparing with the result from DLO (Orange). (Right) The mapping result from Inc-DLOM.

ground truth. Fig. 3.6 (ii) shows the mapping result from Inc-DLOM.

3.3.4 Solid-State LiDAR Scenario

We also evaluate our Inc-DLOM on a dataset collected by an Intel Realsense L515 solid-state LiDAR (RS-L515) [79]. Compared with mechanical LiDARs, RS-L515 produces point clouds with a frequency of 30 Hz, smaller FoV, and higher spatial density as shown in Table 1.1. The data has different ranging distances and the points are not forming the spatial rings as mechanical LiDARs do. These kinds of unique information will make classical LO methods not applicable, e.g., Fast-LIO, LINS, and LIOM, since their feature extraction methods are highly dependent on the properties of mechanical LiDARs.

Therefore, we compare the mapping result of HDL-SLAM, DLO, and Inc-DLOM on the dataset as shown in Fig. 3.7. The mapping visualization of Inc-DLOM is shown in Fig. 3.7 (iii), in which two areas with distinctive features are zoomed. DLO can handle the tracking scenario of solid-state LiDAR, but different zoomed views show that the mapping quality is not good. For point cloud mapping, shelves in the room are blurred and overlapped, and the view of the room also shows inconsistency. The mapping result of the proposed Inc-DLOM is much better as shown in Fig. 3.7 (iii), which demonstrates that it

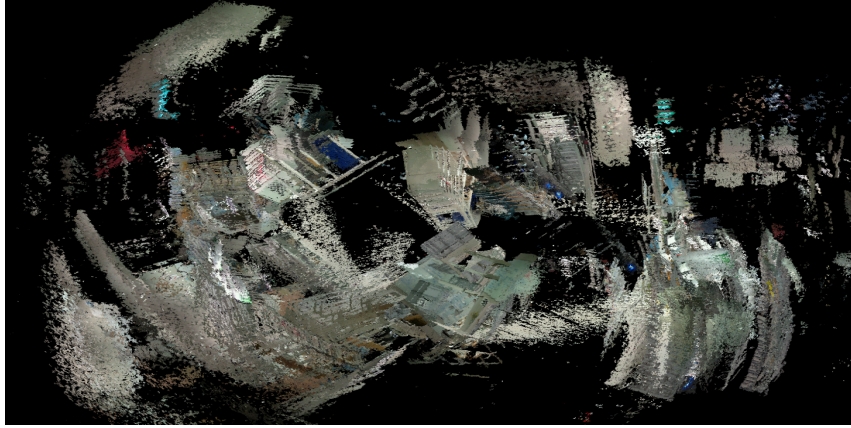
can successfully handle the general working scenarios on different robotics applications.

3.3.5 Magni Robot Scenario

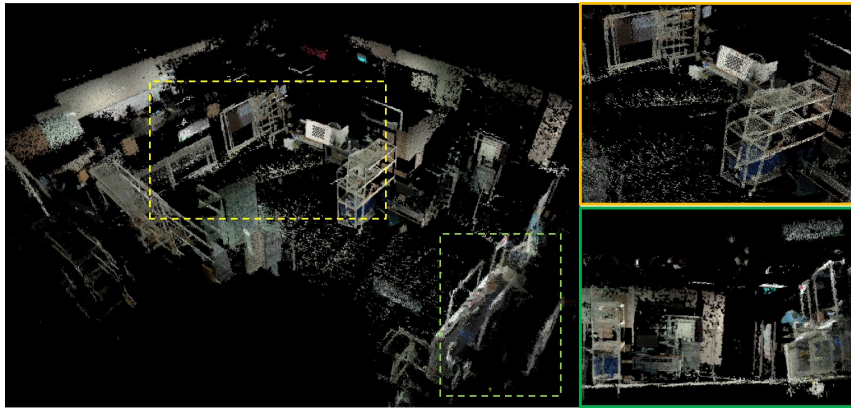
As described in 3.3.1, we collected the dataset in a textureless environment with 6820 frames of point clouds, associated with IMU data of 200 Hz frequency. Only Fast-LIO and DLO can generate proper mapping performance for the corridor among all comparison methods. The point cloud mapping concatenated by the estimated poses is shown as an xy-plane view in Fig. 3.8. DLO can estimate the robot's motions properly in most parts of the trajectory but still generates inconsistency in the mapping. Both Fast-LIO and Inc-DLOM can provide proper estimation results, as their mappings show a proper view of the corridor. However, Fast-LIO's mapping consists of many blurred parts, which means the estimation error is preserved in the estimation. In comparison, our proposed Inc-DLOM can provide clearer and better point cloud mapping. The zoomed views of different parts of the point cloud are shown in Fig. 3.9.

3.4 Conclusion

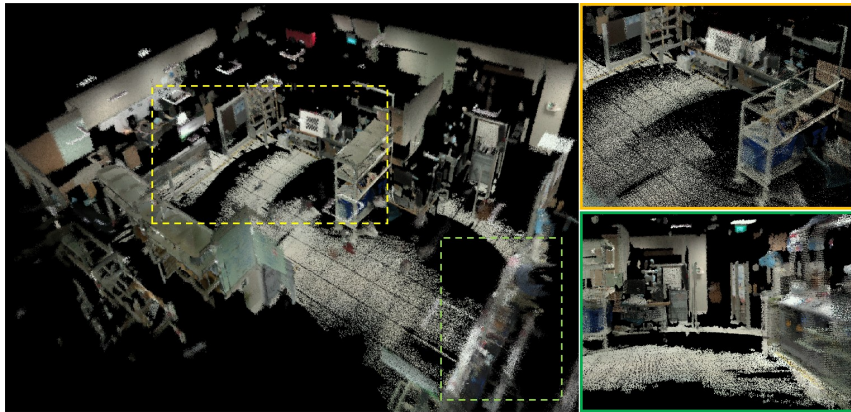
We proposed a novel LiDAR Odometry framework, Incremental Direct LiDAR Odometry (Inc-DLOM) in this paper, which is accurate, computationally efficient, and generalized for multiple robotic platform working scenarios. Inc-DLOM contains point cloud deskew via a constant motion model, Voxel-to-Voxel GICP with the high computational efficiency of scan-to-scan registration method, and a novel scan-to-mapping refinement scheme based on Incremental Voxel Mapping (IVM). Different mapping operations including search, query, and access are within $\mathcal{O}(1)$ time complexity, allowing scan-to-mapping refinement scheme to avoid extensive time-consuming neighboring searches in the traditional point mapping. Eventually, the various components are more tightly integrated by the proposed Inc-DLOM framework. Evaluation of Inc-DLOM on multiple aspects demonstrates that it provides state-of-the-art localization for generalized robotic



(i)

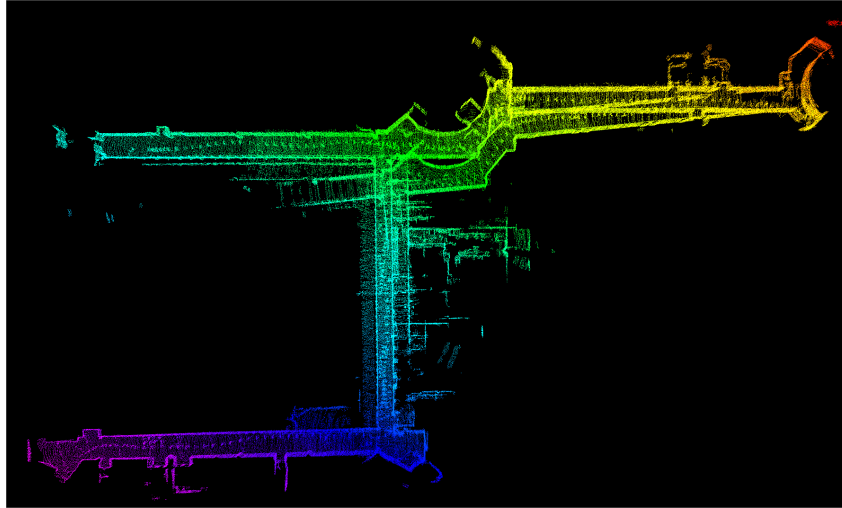


(ii)

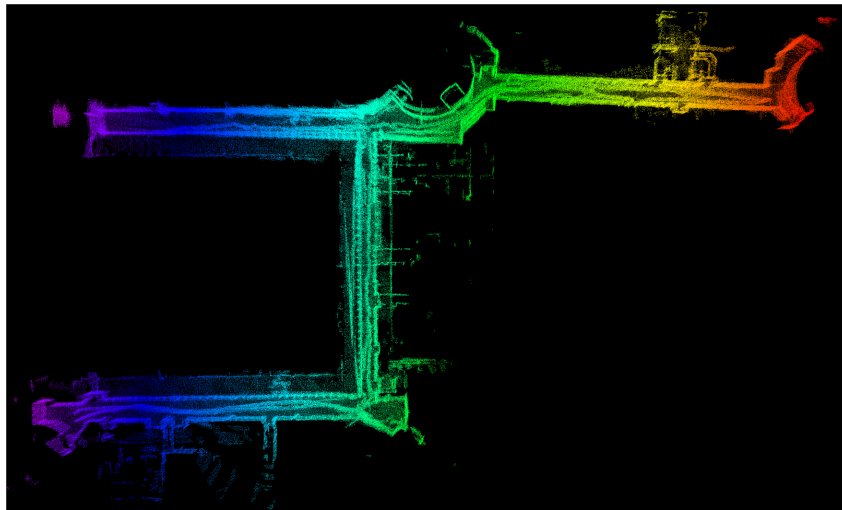


(iii)

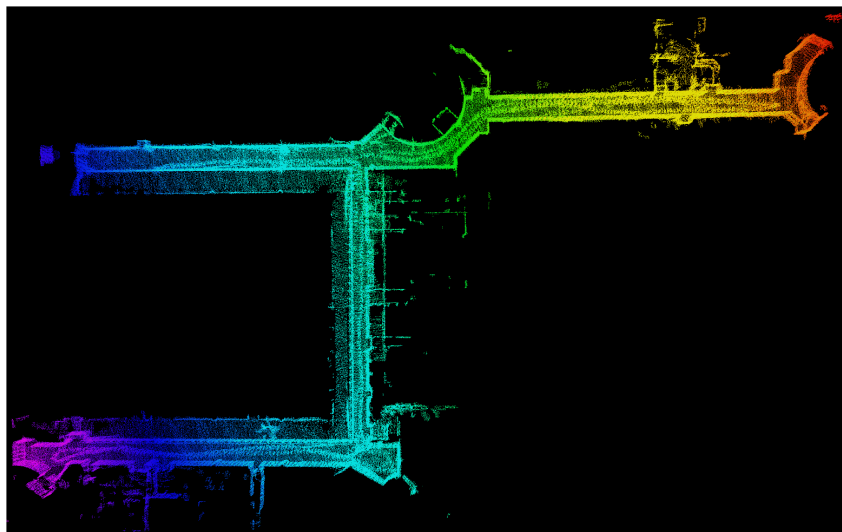
Figure 3.7: **RS-L515 Mapping Comparison.** (i) HDL-SLAM failed to track the motion of a solid-state LiDAR. (ii) The mapping result from DLO is better than (i), but the zoomed views show many twisted objects mapping views. (iii) Inc-DLOM can achieve good mapping results.



(a) DLO



(b) Fast-LIO



(c) Inc-DLOM

Figure 3.8: The point cloud mapping of different methods.

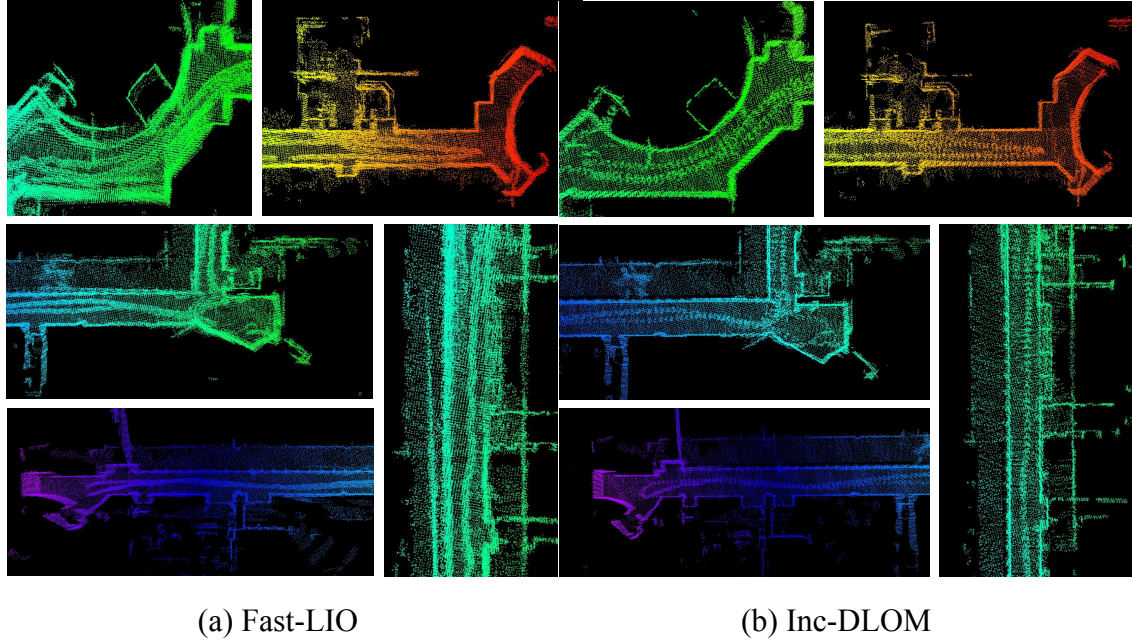


Figure 3.9: Zoomed views of different parts of point cloud mapping.

platforms with real-time performance. As future work, Inc-DLOM can be enhanced with additional sensor inputs, such as IMU, to implement a multiple-sensor fusion SLAM framework.

Chapter 4

Invariant Kalman Filtering Based LiDAR-Inertial Odometry

The contents of this chapter are mostly from the paper [26]. Minor modifications were made, most of the contents and results are remaining the same.

4.1 Introduction

The topic of LiDAR-Inertial Odometry (LIO) is raising the interest of researchers as one of the key areas for robotics navigation, among which Extended Kalman Filtering (EKF) based LIO has become the mainstream of LIO because of its excellent computational speed and good accuracy. However, the EKF-based methods cannot avoid the inconsistency from estimation error linearization. As the complement, Invariant Extended Kalman Filtering (InEKF) designed for state trajectories lying on the matrix Lie groups has been proposed and proved to be excellent in convergence and consistency. In this paper, we propose the method of Direct LiDAR-Inertial Odometry Based on Invariant Kalman Filtering (Invariant-DLIO), which contains the InEKF-based state estimator with the fusion of IMU measurements and LiDAR point clouds, where the error dynamics meets the properties of log-linear and trajectory-independent. A lightweight Scan-to-Mapping module is also designed for the refinement of pose estimation, where the mapping is updated and

operated with $\mathcal{O}(1)$ time complexity. Extensive experiments, including different public datasets and Magni robot data acquisition, are conducted in comparison with a series of state-of-the-art LIO/LO methods. Experimental results show that Invariant-DLIO achieves superior accuracy and efficiency.

4.2 Proposed Method

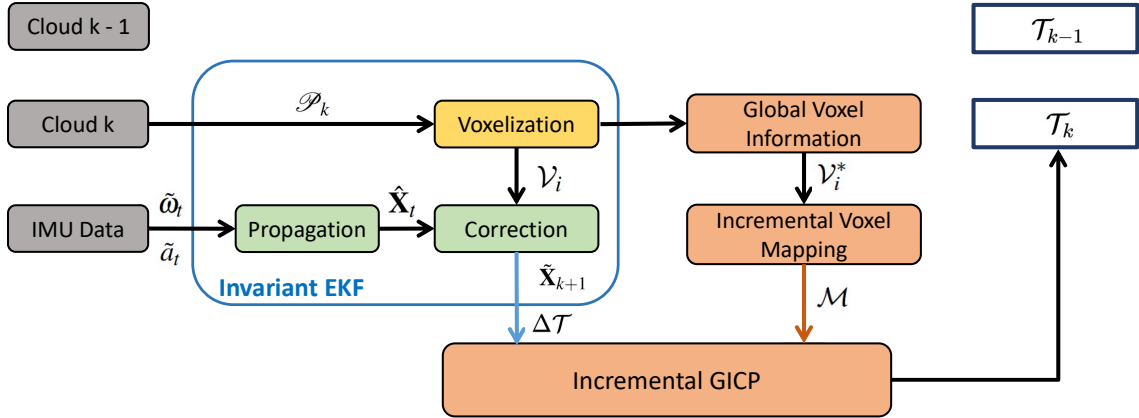


Figure 4.1: Invariant-DLIO takes the point cloud $\mathcal{P}_k$ and IMU raw data $\tilde{\omega}_t, \tilde{a}_t$ as inputs, and goes through the procedure of **Voxelization** and **InEKF Propagation**. **InEKF Correction** step takes voxelized cloud $\mathcal{P}_k$ and propagated state $\hat{\mathbf{x}}_t$ as inputs to estimate robot state $\tilde{\mathbf{x}}_{k+1}$, where the rotation part $\tilde{\mathcal{R}}_t$ and translational part $\tilde{p}_t$ are taken and formed $\Delta\mathcal{T}$ as the initial guess for **Incremental GICP**. The proposed *Incremental Voxel Mapping (IVM)* $\mathcal{M}$ will be updated by *Global Voxel Information* $\mathcal{V}_k^*$. The global pose $\mathcal{T}_k$ is refined by the proposed **Incremental GICP**.

4.2.1 Theory Preliminaries

The concept of the Lie group is widely used in existing SLAM research works. For the matrix Lie group $\mathcal{G}$ with the dimension of n , every element is associated with its Lie Algebra $\mathfrak{g}$. The hat-map operator is defined as (4.1).

$$(\cdot)^\wedge : \mathbb{R}^{\dim(\mathfrak{g})} \rightarrow \mathbb{R}^{\dim(\mathfrak{g}) \times \dim(\mathfrak{g})}; \quad \mathfrak{g} \rightarrow \mathfrak{g}^\wedge, \quad (4.1)$$

in which the matrix $\mathfrak{g}^\wedge$ meets the condition that $\mathcal{G} = \exp_m(\mathfrak{g}^\wedge)$, where $\exp_m$ is the matrix exponential. The inverse of the hat-map is called vee-map, defined as $(\tau^\wedge)^\vee = \tau$. The matrix Lie group $\mathcal{G}$ and associated Lie algebra $\mathfrak{g}$ can be transformed by the matrix exponential map and logarithm map as,

$$\begin{aligned} \exp : \mathfrak{g} &\rightarrow \exp_m(\mathfrak{g}^\wedge), \\ \log : \mathcal{G} &\rightarrow \log_m(\mathcal{G})^\vee. \end{aligned} \tag{4.2}$$

For convenience, we will define the capitalized Exp and Log map to directly associate the Lie algebra vectors with the Lie group matrices, which are defined as,

$$\begin{aligned} \text{Exp} : \mathfrak{g} &\rightarrow \text{Exp}(\mathfrak{g}) = \exp_m(\mathfrak{g}^\wedge), \\ \text{Log} : \mathcal{G} &\rightarrow \text{Log}(\mathcal{G}) = \log_m(\mathcal{G})^\vee. \end{aligned} \tag{4.3}$$

The dynamic system evolving on the Lie group is denoted by (4.4)

$$\frac{d}{dt}\mathcal{X}_t = f_{u_t}(\mathcal{X}_t, \mathbf{w}_t^f), \tag{4.4}$$

where $\mathcal{X}_t \in \mathcal{G}$ is an element belonging to a matrix Lie group $\mathcal{G}$, $f_{u_t}(\cdot)$ is the system modeling function, and $\mathbf{w}_t^f$ is the noise of the state process. The left- and right-invariant errors between the two independent trajectories $\mathcal{X}_t$ and $\bar{\mathcal{X}}_t$ are defined as (4.5),

$$\begin{aligned} \eta_t^l &= \mathcal{X}_t^{-1} \cdot \bar{\mathcal{X}}_t = (\mathcal{L}\mathcal{X}_t)^{-1} \cdot (\mathcal{L}\bar{\mathcal{X}}_t), \text{ (left - invariant)} \\ \eta_t^r &= \bar{\mathcal{X}}_t \cdot \mathcal{X}_t^{-1} = (\bar{\mathcal{X}}_t\mathcal{L}) \cdot (\mathcal{X}_t\mathcal{L})^{-1}, \text{ (right - invariant)} \end{aligned} \tag{4.5}$$

where $\mathcal{L} \in \mathcal{G}$ is one arbitrary group element.

The group-affine dynamic system $f_{u_t}(\cdot)$ satisfies the condition defined as (4.6), $\forall \mathcal{X}_1, \mathcal{X}_2 \in \mathcal{G}$,

$$f_{u_t}(\mathcal{X}_1\mathcal{X}_2) = f_{u_t}(\mathcal{X}_1)\mathcal{X}_2 + \mathcal{X}_1f_{u_t}(\mathcal{X}_2) - \mathcal{X}_1f_{u_t}(\mathbf{I})\mathcal{X}_2, \tag{4.6}$$

where $\mathbf{I} \in \mathcal{G}$ is the group identity element. Furthermore, the invariant error dynamics

defined as (4.5) are trajectory-independent and satisfy the conditions (4.7),

$$\begin{aligned}\frac{d}{dt}\eta_t^l &= g_{u_t}(\eta_t^l) = f_{u_t}(\eta_t^l) - f_{u_t}(\mathbf{I}) \cdot \eta_t^l, \\ \frac{d}{dt}\eta_t^r &= g_{u_t}(\eta_t^r) = f_{u_t}(\eta_t^r) - \eta_t^r \cdot f_{u_t}(\mathbf{I})\end{aligned}\tag{4.7}$$

Let the term ξ_t be the Lie algebra, then the matrix A_t is defined by (4.8),

$$g_{u_t}(\text{Exp}(\xi_t)) = (A_t \xi_t)^\wedge + \mathcal{O}(\|\xi_t\|^2),\tag{4.8}$$

where $\mathcal{O}(\|\xi_t\|^2)$ is the high-order term. If ξ_t is defined by the linear differential equation as (4.9),

$$\frac{d}{dt}\xi_t = A_t \xi_t,\tag{4.9}$$

then $\forall t > 0$, the nonlinear estimation error η_t can be recovered from the differential equation by Log-linear property by (4.10) as

$$\eta_t = \text{Exp}(\xi_t).\tag{4.10}$$

Compared with the traditional ways of Kalman Filtering's error definition, the invariant error with Log-linear property is not the typical Jacobian linearization, since the error can be recovered from its differential as (4.9), (4.10).

4.2.2 State Representation

For the robot state estimation, the variables we focus on are orientation, velocity, and position of the body frame. Based on two common Lie groups, Special Euclidean group $\text{SE}(3)$ and Special Orthogonal group $\text{SO}(3)$, the extended Lie group of $\text{SE}_2(3)$ is defined as (4.11),

$$\mathcal{X} = \begin{bmatrix} \mathcal{R} & v & p \\ \mathbf{0}_{1 \times 3} & 1 & 0 \\ \mathbf{0}_{1 \times 3} & 0 & 1 \end{bmatrix} \in \text{SE}_2(3),\tag{4.11}$$

where $\mathcal{R} \in \text{SO}(3)$, $v \in \mathbb{R}^3$, and $p \in \mathbb{R}^3$ represent the rotation, velocity, and position in the 3D Euclidean space respectively.

Implementing the IMU-based state estimator often requires modeling with gyroscope and accelerometer biases. However, the biases are not on any Lie groups, and the compound group of $\text{SE}_2(3)$ is needed for state representation, which is defined as (4.12),

$$\mathbf{X}_t = [\mathcal{X}_t, \mathbf{b}_t^g, \mathbf{b}_t^a] \in \text{SE}_2(3) \times \mathbb{R}^6. \quad (4.12)$$

For the compound group $\text{SE}_2(3) \times \mathbb{R}^6$, the operator $\boxplus$ and $\boxminus$ are defined as follows,

$$\begin{bmatrix} \mathcal{X} \\ \mathbf{b}_1 \end{bmatrix} \boxplus \begin{bmatrix} \xi \\ \mathbf{b}_2 \end{bmatrix} = \begin{bmatrix} \text{Exp}(\xi) \cdot \mathcal{X} \\ \mathbf{b}_1 + \mathbf{b}_2 \end{bmatrix} \quad (4.13)$$

$$\begin{bmatrix} \mathcal{X}_1 \\ \mathbf{b}_1 \end{bmatrix} \boxminus \begin{bmatrix} \mathcal{X}_2 \\ \mathbf{b}_2 \end{bmatrix} = \begin{bmatrix} \text{Log}(\mathcal{X}_1 \mathcal{X}_2^{-1}) \\ \mathbf{b}_1 - \mathbf{b}_2 \end{bmatrix} \quad (4.14)$$

which will be used for our InEKF estimator in this work. The error term used for our proposed InEKF is defined as augmented right-invariant error as (4.15),

$$\mathbf{e}_t^r = \mathbf{X}_t \boxminus \bar{\mathbf{X}}_t = \begin{bmatrix} \text{Log}(\mathcal{X}_t \cdot \bar{\mathcal{X}}_t^{-1}) \\ \bar{\mathbf{b}}_t^g - \mathbf{b}_t^g \\ \bar{\mathbf{b}}_t^a - \mathbf{b}_t^a \end{bmatrix} = \begin{bmatrix} \xi_{\mathbf{R}_t} \\ \xi_{v_t} \\ \xi_{p_t} \\ \theta_t^g \\ \theta_t^a \end{bmatrix} = \begin{bmatrix} \eta_t^r \\ \theta_t \end{bmatrix} \in \mathbb{R}^{15}. \quad (4.15)$$

where the term $\bar{\mathbf{X}}_t$ represents the optimal estimation.

4.2.3 IMU Kinematics Model and Propagation

The measurement from IMU contains angular velocity and linear acceleration, which are disrupted by biases and Brownian motion noises as follows,

$$\begin{aligned}\tilde{\omega}_t &= \omega_t + \mathbf{b}_t^g + \mathbf{w}_t^g \\ \tilde{a}_t &= a_t + \mathbf{b}_t^a + \mathbf{w}_t^a\end{aligned}\tag{4.16}$$

where $\mathbf{w}_t^g, \mathbf{w}_t^a$ are the sensor noises defined by the Gaussian process as $\mathbf{w}_t^g, \mathbf{w}_t^a \sim \mathcal{GP}(\mathbf{0}, \Sigma \delta(t-t'))$. Considering the IMU biases, the system dynamics model is defined as (4.17).

$$\begin{aligned}\frac{d}{dt}\mathcal{R}_t &= \mathcal{R}_t(\tilde{\omega}_t - \mathbf{b}_t^g - \mathbf{w}_t^g)_\times \\ \frac{d}{dt}v_t &= \mathcal{R}_t(\tilde{a}_t - \mathbf{b}_t^a - \mathbf{w}_t^a)_\times + \mathbf{g} \\ \frac{d}{dt}p_t &= v_t\end{aligned}\tag{4.17}$$

The error term of the propagated state is defined as

$$\hat{\mathbf{e}}_t^r = \mathbf{X}_t \boxminus \hat{\mathbf{X}}_t = \begin{bmatrix} \text{Log}(\mathcal{X}_t \cdot \hat{\mathcal{X}}_t^{-1}) \\ \hat{\mathbf{b}}_t^g - \mathbf{b}_t^g \\ \hat{\mathbf{b}}_t^a - \mathbf{b}_t^a \end{bmatrix} = \begin{bmatrix} \hat{\xi}_{\mathcal{R}_t} \\ \hat{\xi}_{v_t} \\ \hat{\xi}_{p_t} \\ \hat{\theta}_t^g \\ \hat{\theta}_t^a \end{bmatrix} = \begin{bmatrix} \hat{\eta}_t^r \\ \hat{\theta}_t \end{bmatrix} \in \mathbb{R}^{15}.\tag{4.18}$$

To derive the state transition matrix, we need to find out the first-order approximation of the error derivative. We have

$$\begin{aligned}\frac{d}{dt}\hat{\xi}_{\mathcal{R}_t} &= \frac{d}{dt}(\mathcal{R}_t \hat{\mathcal{R}}_t^T) = (\hat{\mathcal{R}}_t(\mathbf{w}_t^g - \theta_t^g))_\times \\ \frac{d}{dt}\hat{\xi}_{v_t} &= \frac{d}{dt}(v_t - \mathcal{R}_t \hat{\mathcal{R}}_t^T v_t) = (\mathbf{g})_\times \hat{\xi}_{\mathcal{R}_t} + \\ &\quad (\hat{v}_t)_\times \hat{\mathcal{R}}_t(\mathbf{w}_t^g - \theta_t^g) + \hat{\mathcal{R}}_t(\mathbf{w}_t^a - \theta_t^a) \\ \frac{d}{dt}\hat{\xi}_{p_t} &= \frac{d}{dt}(\hat{p}_t - \mathcal{R}_t \hat{\mathcal{R}}_t^T p_t) = \hat{\xi}_{v_t} + (\hat{p}_t)_\times \hat{\mathcal{R}}_t(\mathbf{w}_t^g - \theta_t^g)\end{aligned}\tag{4.19}$$

The linear system with invariant error dynamics is constructed by

$$\frac{d}{dt}\mathbf{e}_t^r = \frac{d}{dt}\begin{bmatrix} \eta_t^r \\ \theta_t \end{bmatrix} = \mathbf{A}_t \begin{bmatrix} \eta_t^r \\ \theta_t \end{bmatrix} + \mathbf{G}_t \mathbf{n}_t, \quad (4.20)$$

where $\mathbf{n}_t = [\mathbf{w}_t^g; \mathbf{w}_t^a; \mathbf{0}; \mathbf{w}_t^{bg}; \mathbf{w}_t^{ba}]$ represents the augmented noise terms and $\mathbf{A}_t, \mathbf{G}_t$ are the state-transition matrix and Jacobian matrix of the noise at time t , which are shown as (4.21, 4.22),

$$\mathbf{A}_t = \begin{bmatrix} 0 & 0 & 0 & -\hat{\mathcal{R}}_t & 0 \\ (\mathbf{g})_{\times} & 0 & 0 & -(\hat{v}_t)_{\times} \hat{\mathcal{R}}_t & -\hat{\mathcal{R}}_t \\ 0 & \mathbf{I} & 0 & -(\hat{p}_t)_{\times} \hat{\mathcal{R}}_t & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{15 \times 15} \quad (4.21)$$

$$\mathbf{G}_t = \begin{bmatrix} -\hat{\mathcal{R}}_t & 0 & 0 & 0 \\ -(\hat{v}_t)_{\times} \hat{\mathcal{R}}_t & -\hat{\mathcal{R}}_t & 0 & 0 \\ -(\hat{p}_t)_{\times} \hat{\mathcal{R}}_t & 0 & 0 & 0 \\ 0 & 0 & \mathbf{I} & 0 \\ 0 & 0 & 0 & \mathbf{I} \end{bmatrix} \in \mathbb{R}^{15 \times 12} \quad (4.22)$$

To discretize the IMU dynamics and propagate the error-state covariance, the continuous-time Riccati equation is utilized as (4.23),

$$\frac{d}{dt}\mathbf{P}_t = \mathbf{A}_t \mathbf{P}_t + \mathbf{P}_t \mathbf{A}_t^T + \bar{\mathbf{Q}}_t, \quad (4.23)$$

of which, the analytical solution of the discretization is shown below as (4.24),

$$\hat{\mathbf{P}}_{k+1} = \exp_m(\mathbf{A}_t \Delta t) \hat{\mathbf{P}}_k \exp_m(\mathbf{A}_t \Delta t)^T + \mathbf{G}_t \mathbf{Q}_t \mathbf{G}_t^T \Delta t, \quad (4.24)$$

where the term Δt represents the IMU sampling time, $\mathbf{Q}_t$ is the covariance of IMU data

noises computed by offline calibration, and $\mathbf{P}_k$ is the uncertainty of state estimation.

It is worth noting that the values related to state propagation of the matrix $\mathbf{A}_t$ (i.e., the 3×3 in the upper-left corner) are all constants, which means that the state propagated in the next step will be independent of the estimated state and not influenced by biased estimation results. Although the parts related to IMU biases propagation contains the estimated state, the quantities introduced is only with a very small scale. This property serves as the foundation of our proposed Invariant EKF.

4.2.4 Invariant Extended Kalman Filtering (InEKF) Observation

Typically, for the Lie group $\mathcal{X} \in \mathcal{G}$, the right-invariant observation is defined in (4.25),

$$\mathbf{Y} = \mathcal{X}^{-1}b, \quad (4.25)$$

where b is a known vector. However, this observation definition is only suitable when the dimension of $\mathbf{Y}$ is small, for the observation based on large-scale point clouds, a novel definition is needed.

4.2.4.1 Point Cloud Voxelization

The point cloud data $\mathcal{P}_k$ will be transformed from a point set into a voxel set $\mathcal{P}_k$, defined as (5.11).

$$\mathcal{P}_k \rightarrow \mathcal{P}_k = \{\dots, \mathcal{V}_i, \dots\} = \bigcup_{i=0}^n \mathcal{V}_i, \quad \mathcal{V}_i = \{\mathcal{C}_i, p_i\} \quad (4.26)$$

In which, n is the size of $\mathcal{P}_k$, and $\mathcal{V}_i$ is the i -th voxel in the point cloud data. The voxel $\mathcal{V}_i$ contains the local surface covariance $\mathcal{C}_i \in \mathbb{R}^{3 \times 3}$ and the central coordinate $p_i \in \mathbb{R}^3$. The voxels are generated by the Approximate Voxel Filter [67]. $\mathcal{C}_i$ is computed from nearby points searched by a KD-Tree. In our implementation, NanoFLANN and OpenMP are used for efficiency improvement [8].

4.2.4.2 Correction

The estimated state of InEKF is the relative transformation between two consecutive LiDAR frames k and $k + 1$, denoted as (4.27), and the state error is defined as (4.28).

$$\mathbf{X}_k^{k+1} = [\mathcal{X}_k^{k+1}, \mathbf{b}_k^g, \mathbf{b}_k^a] \quad (4.27)$$

$$\mathbf{e}_k^{k+1} = \hat{\mathbf{X}}_k^{k+1} \boxminus \mathbf{X}_k^{k+1} \quad (4.28)$$

For convenience, we will use the term $\mathbf{X}_{k+1}$ and $\mathbf{e}_{k+1}$ in the following parts. When measurements from the $(k + 1)$ th LiDAR frame arrive, we will have the prior knowledge from the IMU propagation as the propagated state $\hat{\mathbf{X}}_{k+1}$ and covariance $\hat{\mathbf{P}}_{k+1}$. A prior distribution is added to the state by propagation process, from which the error state is denoted as (4.29),

$$\hat{\mathbf{e}}_{k+1} = \mathbf{X}_{k+1} \boxminus \hat{\mathbf{X}}_{k+1} \sim \mathcal{N}(\mathbf{0}, \hat{\mathbf{P}}_{k+1}). \quad (4.29)$$

Combined with the point cloud measurements, a maximum a posteriori (MAP) estimation $\tilde{\mathbf{X}}_{k+1}$ of the state $\mathbf{X}_{k+1}$ will be generated from the prior distribution. The MAP estimation can be solved in an iterated manner from the prior state as the initial value, then the MAP error state $\tilde{\mathbf{e}}_k^{k+1}$ in every iteration step is defined as (4.30),

$$\tilde{\mathbf{e}}_{k+1} = \mathbf{X}_{k+1} \boxminus \tilde{\mathbf{X}}_{k+1} \quad (4.30)$$

Since the propagated state and measurement model have different distributions, the projection of $\hat{\mathbf{e}}_{k+1}$ to the tangent space of $\tilde{\mathbf{e}}_{k+1}$ is necessary as

$$\begin{aligned} \hat{\mathbf{e}}_{k+1} &= \mathbf{X}_{k+1} \boxminus \hat{\mathbf{X}}_{k+1} = (\tilde{\mathbf{X}}_{k+1} \boxplus \tilde{\mathbf{e}}_{k+1}) \boxminus \hat{\mathbf{X}}_{k+1} \\ &\approx (\tilde{\mathbf{X}}_{k+1} \boxminus \hat{\mathbf{X}}_{k+1}) + \mathbf{J}_{k+1} \tilde{\mathbf{e}}_{k+1} \end{aligned} \quad (4.31)$$

where

$$\mathbf{J}_{k+1} = \begin{bmatrix} \mathcal{J}_l^{-1}(\text{Log}(\tilde{\mathcal{X}}_{k+1}\hat{\mathcal{X}}_{k+1}^{-1})) & \mathbf{0} \\ \mathbf{0} & \mathbf{I}_{6 \times 6} \end{bmatrix} \quad (4.32)$$

the term $\mathbf{J}_{k+1}$ is the inverse Jacobian matrix of $\hat{\mathbf{e}}_{k+1}$ with respect to $\tilde{\mathbf{e}}_{k+1}$ evaluated at 0. Then the prior distribution of $\tilde{\mathbf{e}}_{k+1}$ can be written as

$$\tilde{\mathbf{e}}_{k+1} \sim \mathcal{N}(-\mathbf{J}_{k+1}^{-1}(\tilde{\mathbf{X}}_{k+1} \boxminus \hat{\mathbf{X}}_{k+1}), \mathbf{J}_{k+1}^{-1} \hat{\mathbf{P}}_{k+1} \mathbf{J}_{k+1}^{-T}). \quad (4.33)$$

To develop our direct-based InEKF, a modified Generalized-ICP (GICP) [68] optimization objective is proposed. Here we denote the k -th and $(k+1)$ -th LiDAR frame as the target set $\mathcal{P}^T$ and source set $\mathcal{P}^S$ respectively. $\forall \mathcal{V}_i^S \in \mathcal{P}^S$, we have the observation residual defined as (4.34)

$$\mathbf{z}_i = \|p_i^T - (\tilde{\mathcal{R}} \cdot p_i^S + \tilde{t})\|_{\Sigma^{-1}}^2 \quad (4.34)$$

$$\Sigma = \mathcal{C}_i^M + \tilde{\mathcal{R}} \cdot \mathcal{C}_i^S \cdot \tilde{\mathcal{R}}^T \quad (4.35)$$

where $\mathcal{V}_i^T \in \mathcal{P}^T$ is the closest neighboring voxel, Σ is the voxel-to-voxel covariance defined in (4.35), in which $\mathcal{C}_i^T$ and $\mathcal{C}_i^S$ are voxel covariances of $\mathcal{V}_i^T$ and $\mathcal{V}_i^S$ respectively. To fuse the residual as (4.34) with propagated state $\hat{\mathbf{X}}_{k+1}$ and covariance $\hat{\mathbf{P}}_{k+1}$, we need to linearize the measurement model $\mathbf{z}_i$ expressed in terms of the nominal state and measurement noises. For general cases, LiDAR will bring laser-beam noises into the points measuring. For every single voxel, we assume it has a measurement noise $\mathbf{n}_i \sim \mathcal{N}(\mathbf{0}, \mathbf{R}_i)$, and the voxel coordinate measurement can be expressed as

$$p_i = {}^{gt}p_i + \mathbf{n}_i, \quad (4.36)$$

where the notation ${}^{gt}(\cdot)$ represents the true point. Then the measurement model is defined as (4.37).

$$\mathbf{z}_i(\mathbf{X}_{k+1}, \mathbf{n}_i) = \|p_i^T - (\tilde{\mathcal{R}} \cdot (p_i^S - \mathbf{n}_i) + \tilde{t})\|_{\Sigma^{-1}}^2 \quad (4.37)$$

The linearization by first-order approximation is expressed as (4.38),

$$\mathbf{z}_i(\mathbf{X}_{k+1}, \mathbf{n}_i) \approx \mathbf{z}_i(\tilde{\mathbf{X}}_{k+1}, \mathbf{0}) + \mathbf{H}_i \tilde{\mathbf{e}}_{k+1} + \mathbf{v}_i = \tilde{\mathbf{z}}_i + \mathbf{H}_i \tilde{\mathbf{e}}_{k+1} + \mathbf{v}_i \quad (4.38)$$

$$\mathbf{H}_i = \frac{\partial \mathbf{z}_i(\mathbf{X}_{k+1}, \mathbf{n}_i)}{\partial \tilde{\mathbf{e}}_{k+1}} = L^{-1} \begin{bmatrix} (\tilde{p}_i^S)_\times & \mathbf{0}_{3 \times 3} & -\mathbf{I}_{3 \times 3} & \mathbf{0}_{3 \times 6} \end{bmatrix} \quad (4.39)$$

where $\mathbf{H}_i \in \mathbb{R}^{3 \times 15}$ is the Jacobian matrix of $\mathbf{z}_i(\mathbf{X}_{k+1}, \mathbf{n}_i)$ with respect to $\tilde{\mathbf{X}}_{k+1}$ evaluated at zero, $\mathbf{v}_i \in \mathcal{N}(0, \mathbf{R}_i)$ comes from the measurement noise. L is the matrix square root of Σ so that $\Sigma = LL^T$.

Combining the prior distribution (4.29) with the linearized measurement model (4.38) leads to a maximum a posterior (MAP) estimation as

$$\min_{\tilde{\mathbf{e}}_{k+1}} (||(\tilde{\mathbf{X}}_{k+1} \boxminus \hat{\mathbf{X}}_{k+1}) + \mathbf{J}_{k+1} \tilde{\mathbf{e}}_{k+1}||_{\hat{\mathbf{P}}_{k+1}}^2 + \sum_i^m ||\mathbf{z}_i + \mathbf{H}_i \tilde{\mathbf{e}}_{k+1}||_{\mathbf{R}_i}^2) \quad (4.40)$$

which is solved with InEKF in an iterated manner. Let the terms

$$\mathbf{P} = \mathbf{J}_{k+1}^{-1} \hat{\mathbf{P}}_{k+1} \mathbf{J}_{k+1}^{-T} \in \mathbb{R}^{15 \times 15} \quad (4.41)$$

$$\mathbf{H} = [\dots, \mathbf{H}_i^T, \dots]^T \in \mathbb{R}^{3m \times 15} \quad (4.42)$$

$$\mathbf{R} = \text{diag}(\dots, \mathbf{R}_i, \dots) \in \mathbb{R}^{3m \times 3m} \quad (4.43)$$

$$\tilde{\mathbf{z}} = [\tilde{\mathbf{z}}_1^T, \dots, \tilde{\mathbf{z}}_i^T, \dots, \tilde{\mathbf{z}}_m^T]^T \in \mathbb{R}^{3m} \quad (4.44)$$

where the term m represents the dimension of point cloud observations. The Kalman gain matrix and updated estimated state are defined as

$$\mathbf{K} = (\mathbf{H}^T \mathbf{R}^{-1} \mathbf{H} + \mathbf{P}^{-1})^{-1} \mathbf{H}^T \mathbf{R}^{-1} \in \mathbb{R}^{15 \times 3m} \quad (4.45)$$

$$\tilde{\mathbf{X}}_{k+1} \leftarrow \tilde{\mathbf{X}}_{k+1} \boxplus (-\mathbf{K} \tilde{\mathbf{z}} - (\mathbf{I} - \mathbf{K} \mathbf{H}) \mathbf{J}_{k+1}^{-1} (\tilde{\mathbf{X}}_{k+1} \boxminus \hat{\mathbf{X}}_{k+1})) \quad (4.46)$$

Note that the update step (4.46) will be ongoing while the convergence condition is not

met, and every updated $\tilde{\mathbf{X}}_{k+1}$ is reused to compute the residuals (4.38) in every iteration. After $\tilde{\mathbf{X}}_{k+1}$ is converged to the optimal solution, the covariance matrix will be updated as (4.47). The voxel correspondences will be re-generated for new iterations to update the state, till $\tilde{\mathbf{X}}_{k+1}$ achieves global convergence. Finally, the initial guess $\Delta\mathcal{T}$ for Incremental GICP is formed by the estimated state of InEKF as (4.48).

$$\tilde{\mathbf{P}}_{k+1} = (\mathbf{I} - \mathbf{KH})\mathbf{P}(\mathbf{I} - \mathbf{KH})^T \quad (4.47)$$

$$\Delta\mathcal{T} = \begin{bmatrix} \tilde{\mathcal{R}}_{k+1} & \tilde{p}_{k+1} \\ \mathbf{0} & 1 \end{bmatrix} \in \text{SE}(3) \quad (4.48)$$

4.2.4.3 Incremental Update on InEKF

However, the listed equations involve many terms, in which the dimensions will vary with the number of voxel measurements, e.g., $\tilde{\mathbf{z}}$, $\mathbf{H}$, etc. Since our proposed InEKF is based on direct-based methods, the matrix calculations (4.45) (4.46) will include multiplication with very large matrix, which will cause huge negative impact on LIO's efficiency.

To address the mentioned issue, we simplified the matrix calculations with incremental method instead of stacking them as batch calculation. Let the terms defined as

$$\mathbf{K}_1 = \sum_{i=1}^m \mathbf{H}_i^T \cdot \mathbf{R}_i^{-1} \cdot \mathbf{H}_i = \mathbf{H}^T \mathbf{R}^{-1} \mathbf{H} \in \mathbb{R}^{15 \times 15} \quad (4.49)$$

$$\mathbf{K}_2 = \sum_{i=1}^m \mathbf{H}_i^T \cdot \mathbf{R}_i^{-1} \cdot \tilde{\mathbf{z}}_i \in \mathbb{R}^{15 \times 1} \quad (4.50)$$

$$\mathbf{K}\tilde{\mathbf{z}} = (\mathbf{K}_1 + \mathbf{P}^{-1})^{-1} \mathbf{K}_2 \quad (4.51)$$

$$\mathbf{KH} = (\mathbf{K}_1 + \mathbf{P}^{-1})^{-1} \mathbf{K}_1 \quad (4.52)$$

then substitute (4.51), (4.52) into (4.46), we have the updated equations as (4.53) and (4.54):

$$\begin{aligned} \tilde{\mathbf{X}}_{k+1} \leftarrow \tilde{\mathbf{X}}_{k+1} \boxplus & \left(-(\mathbf{K}_1 + \mathbf{P}^{-1})^{-1} \mathbf{K}_2 \right. \\ & \left. - (\mathbf{I} - (\mathbf{K}_1 + \mathbf{P}^{-1})^{-1} \mathbf{K}_1) \mathbf{J}_{k+1}^{-1} (\tilde{\mathbf{X}}_{k+1} \boxminus \hat{\mathbf{X}}_{k+1}) \right) \end{aligned} \quad (4.53)$$

$$\tilde{\mathbf{P}}_{k+1} = (\mathbf{I} - (\mathbf{K}_1 + \mathbf{P}^{-1})^{-1} \mathbf{K}_1) \mathbf{P} (\mathbf{I} - (\mathbf{K}_1 + \mathbf{P}^{-1})^{-1} \mathbf{K}_1)^T \quad (4.54)$$

In our InEKF implementation, the terms $\mathbf{K}_1$ and $\mathbf{K}_2$ are computed firstly with multi-thread acceleration, which is extremely efficient for C++ environments. Then it is obvious that all matrix operations, such as plus, multiplication, and inversion, are all with very small sizes, which will not increase the computation cost for the LIO system. The related ablation study will be shown in the experimental section.

4.2.5 Scan-to-Mapping Refinement

4.2.5.1 Incremental Voxel Mapping

Incremental Voxel Mapping (IVM) is implemented based on spatial hashing to insert and update the historical voxel mapping incrementally. All voxels in IVM are stored in the hash table, and every one of them is associated with a unique key. Before inserting into IVM, the voxel data are transformed into the global frame by (4.55), which refers to *Global Voxel Information* in Fig. 4.1.

$$\mathcal{V}_i^* = \mathcal{T}_k \cdot \mathcal{V}_i := \{\mathcal{R}_k \cdot \mathcal{C}_i \cdot \mathcal{R}_k^T, \mathcal{R}_k \cdot p_i + t_k\} = \{\mathcal{C}_i^*, p_i^*\} \quad (4.55)$$

To access one specific voxel, only p_i^* is needed to compute the spatial key as

$$\text{key}(\mathcal{V}_i^*) = \text{floor}(p_i^*/S_{IVM}) \quad (4.56)$$

where S_{IVM} is the pre-defined voxel size parameter of IVM. Since the hash mapping of the data is unique, $\mathcal{V}_i^*$ can be accessed with $\mathcal{O}(1)$ time complexity, which is essential for efficiency.

In our proposed IVM, the incoming set of $\mathcal{V}_i^*$ is continuously inserted to expand the

historical mapping $\mathcal{M}$ denoted as

$$\mathcal{M} = \sum_{j=0}^M \bigcup_{i=0}^n \mathcal{V}_i^* = \bigcup \mathcal{V}_{\mathcal{M}}, \quad (4.57)$$

where M is the number of point cloud frames, and the sum operation means the concatenation of different voxel sets. While expanding $\mathcal{M}$, the voxels to be inserted are classified into two types, new voxels $\mathcal{V}_n^*$ and historical voxels $\mathcal{V}_h^*$, depending on whether $\text{key}(\mathcal{V}^*)$ is inserted into IVM before. All $\mathcal{V}_n^*$ will be inserted into $\mathcal{M}$ with the hash table process. $\mathcal{V}_h^*$ will be merged into the voxel with the same spatial key, which is denoted as $\mathcal{V}_{\mathcal{M}}$. The voxel merging operation is noted as $\mathcal{V}_r^* = \mathcal{V}_h^* \oplus \mathcal{V}_{\mathcal{M}}$, which is defined as follows:

$$p_r^* = \frac{p_h^* + p_{\mathcal{M}}}{2} \quad (4.58)$$

$$\begin{aligned} \mathcal{C}_r^* = & (\mathcal{C}_{\mathcal{M}} + \mathcal{C}_h^* + (p_{\mathcal{M}} - p_r^*)(p_{\mathcal{M}} - p_r^*)^T \\ & + (p_h^* - p_{\mathcal{M}})(p_h^* - p_{\mathcal{M}})^T)/2 \end{aligned} \quad (4.59)$$

4.2.5.2 Incremental GICP

Incremental GICP takes the historical mapping $\mathcal{M}$, voxel set $\mathcal{P}_k$, and the initial guess $\Delta\mathcal{T} \in \text{SE}(3)$ as inputs. The difference between Incremental GICP and original GICP is mainly in the correspondence process, where getting $\mathcal{V}_{\mathcal{M}}$ by $\text{key}(\mathcal{V}_i)$ is the $\mathcal{O}(1)$ operation in IVM hash table instead of KD-Tree searching, and the cost function will be constructed if the voxel associated with $\text{key}(\mathcal{V}_i)$ can be searched in IVM. The cost function of Incremental GICP is defined as (4.60), which is solved by Gauss-Newton iteratively.

$$\mathcal{T}_k = \underset{\mathcal{T} \in \text{SE}(3)}{\text{argmin}} \sum_i \mathbb{F}(\mathcal{V}_{\mathcal{M}}, \mathcal{V}_i) = \underset{\mathcal{T} \in \text{SE}(3)}{\text{argmin}} \sum_i \|p_{\mathcal{M}} - \mathcal{T} \cdot p_i\|_{\Sigma^{-1}}^2 \quad (4.60)$$

$$\Sigma = \mathcal{C}_{\mathcal{M}} + \mathcal{R} \cdot \mathcal{C}_i \cdot \mathcal{R}^T$$

4.3 Experimental Results

4.3.1 Experiment Setup

Extensive experiments are conducted on multiple public datasets to assess Invariant-DLIO's performance over different working scenarios fully. The accuracy is firstly tested on the KITTI Odometry Benchmark [29], the classical dataset for autonomous driving scenarios. It is worth noting that the IMU data are generated from raw data that is not calibrated and aligned with LiDAR point cloud timestamps. We have processed the IMU data with interpolation and alignment to the LiDAR point clouds. The fused data contains IMU data with 100 Hz and point cloud data with 10 Hz frequencies. Autonomous driving scenarios are commonly considered not extremely challenging due to the smooth motions of the vehicle. As a complement, experiments for the robotic platforms with high-frequency ego-motions are conducted. The University of Michigan North Campus Long-Term Dataset (NCLT) [10] and the Newer College Dataset (NCD) [64] are utilized as benchmarks.

In addition, we also collected the custom dataset with our robotic platform, deployed with VLP-16 LiDAR, IMU, MaxTang FP650 NUC [57], shown in Fig. 4.2. The dataset consists of indoor and outdoor scenarios.

Finally, the computational efficiency is also evaluated. For all aspects of the experiments, we compare our Invariant-DLIO with different state-of-the-art open-source LO systems, including feature-based LIO methods: Fast-LIO [85], LINS [62], LIOM [87], and direct LO method: HDL-SLAM [43] and Direct LiDAR Odometry (DLO) [14]. Fig. 4.4 shows the trajectories estimated by Invariant-DLIO in different data sequences.

4.3.2 Ablation Study

4.3.2.1 Effect of Incremental InEKF Update

As described in 4.2.4.3, we proposed an InEKF update construction method in the incremental way, which performs better than the traditional batch ways. For comparison, we

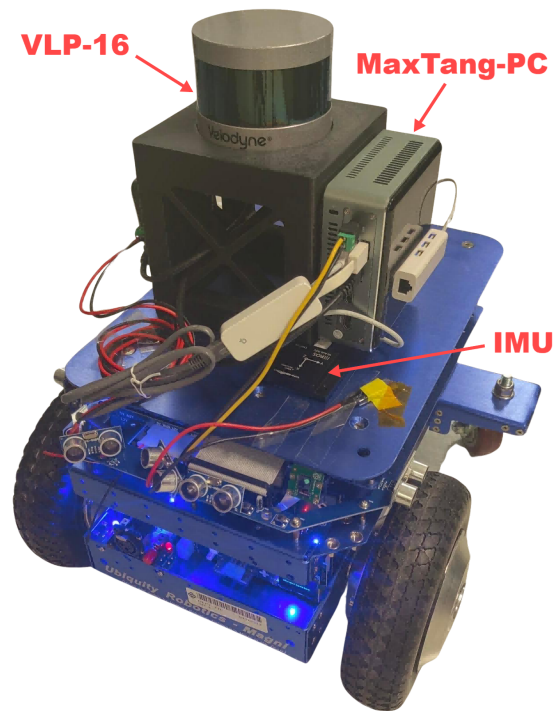


Figure 4.2: Data Collection Robotic Platform



Figure 4.3: Textureless Corridor View

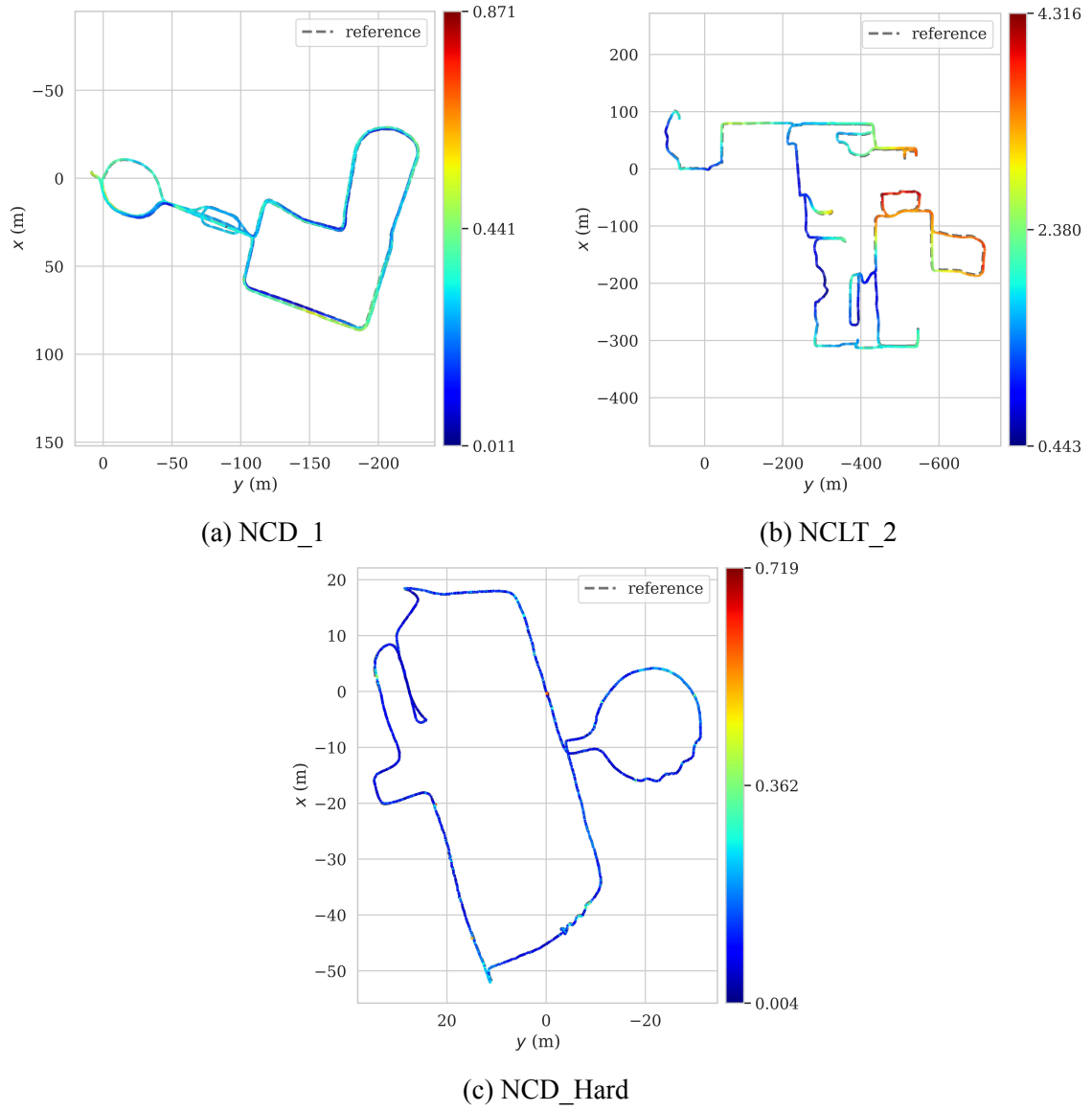


Figure 4.4: Trajectories estimated by Invariant-DLIO, the color bar means the error plot.

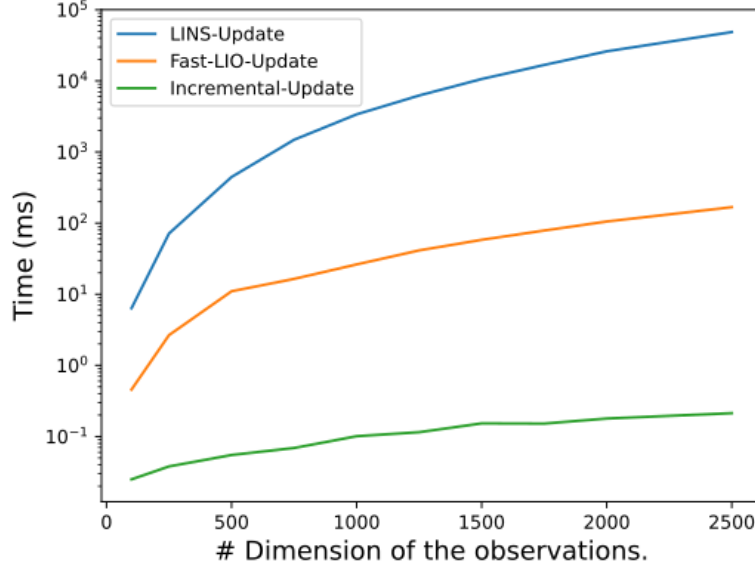


Figure 4.5: **Time Cost of Incremental InEKF Update.**

compared the time cost for a single iteration step of our proposed InEKF with the same part from LINS and Fast-LIO, shown in Fig. 4.5. From the plot, we can see our proposed Incremental InEKF update module shows the best of time efficiency within the single iteration step. LINS performs worst as its implementation uses batch update, which is only suitable for feature-based methods that have less measurement than direct-based methods. Fast-LIO performs better because the sparse matrices are used in the implementation.

4.3.2.2 Validation of IVM

To illustrate the effect of the IVM procedure in Invariant-DLIO, Fig. 4.6 shows the visualization of IVM in the KITTI 00, (a) shows the mapping of the first 150 point clouds without processing, and (b) visualizes the IVM, from which we can see that the IVM is ranging the exact distance as the original point cloud mapping. Table 4.1 shows the statistical comparison of the original point mapping and IVM, and the parameter $S_{IVM} = 1.50\text{m}$, which gives rise to the best estimation result in the KITTI Dataset [29]. The IVM algorithm can maximize the preservation of large-scale maps for Scan-to-Mapping refinement. Compared with traditional point mapping methods, IVM saves dozens of times of mem-

ory, and the voxel hash table allows $\mathcal{O}(1)$ voxel query operation. With these properties, Scan-to-Mapping is fast, lightweight, and accurate.

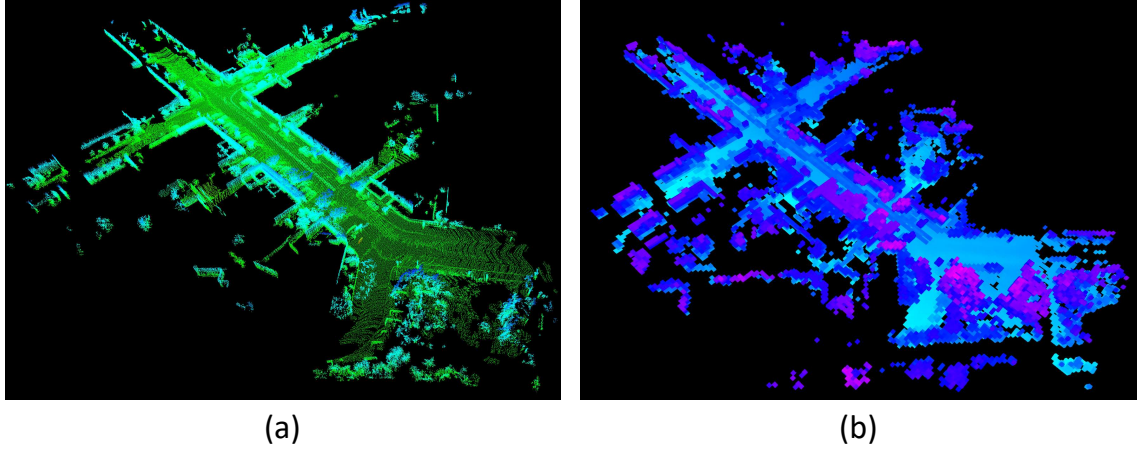


Figure 4.6: **The visualization of IVM.** (a) The plot of the point cloud mapping. (b) The plot of IVM on the mapping.

Table 4.1: Statistical Results of the IVM.

	Point Mapping	Inc-Voxel Mapping
Points / Voxels No.	2535005	15852
Memory Usage	29.01 MB	0.18 MB

4.3.2.3 Validation of InEKF

To evaluate the efficacy of the InEKF within our proposed Invariant-DLIO framework, a loosely-coupled LiDAR-Inertial Odometry (LC-LIO) system was implemented for comparative analysis. In this LC-LIO variant, the InEKF propagation and correction modules were omitted, with initial values for the Incremental Generalized Iterative Closest Point (GICP) algorithm instead derived from IMU pre-integration [27]. Experimental evaluations conducted on the KITTI and NCLT datasets demonstrate a marked degradation in the performance of LC-LIO compared with Invariant-DLIO. This discrepancy arises primarily from the absence of the InEKF procedure, which compromises the quality of initial alignment estimates required for robust Incremental GICP optimization. While the KITTI dataset, characterized by smoother vehicular motion profiles, exhibits less pronounced performance degradation, due to the inherent capability of Incremental GICP to

tolerate suboptimal initializations under moderate kinematic conditions, the limitations of LC-LIO become evident in high-frequency scenarios. Specifically, on the NCLT dataset, where motion dynamics are more aggressive and erratic, the lack of InEKF-derived corrections leads to significant drift and instability. These results underscore the critical role of the InEKF framework in ensuring reliable state estimation, particularly under challenging motion regimes where IMU pre-integration alone proves insufficient for accurate initialization.

4.3.3 KITTI Odometry Evaluation

To validate the performance in autonomous driving scenarios, we evaluated Invariant-DLIO on KITTI Odometry Benchmarks. The trajectories are first evaluated with translational/rotational drifting error metrics as defined in KITTI Odometry. Table 4.2 shows the evaluation results, the best results among the considered methods for each sequence are marked in red.

Table 4.2: Translational Drifting Error (%) / Rotational Drifting Error (deg / 100m) Evaluation of KITTI Dataset

	00	02	05	06	07	08	09	10
Inv-DLIO	0.58 / 0.32	0.91 / 0.32	0.39 / 0.24	0.23 / 0.17	0.12 / 0.11	0.89 / 0.34	0.26 / 0.21	0.19 / 0.20
LC-LIO	0.76 / 0.60	1.27 / 0.49	0.46 / 0.52	0.72 / 0.37	0.43 / 0.31	1.26 / 0.54	0.58 / 0.49	0.37 / 0.75
Fast-LIO	3.24 / 1.35	1.84 / 0.55	0.51 / 0.26	1.34 / 0.48	0.46 / 0.29	2.68 / 1.07	0.36 / 0.34	0.51 / 2.07
LINS	8.46 / 3.45	10.08 / 3.04	7.64 / 3.56	1.12 / 0.86	4.29 / 2.12	8.75 / 3.53	11.49 / 3.60	9.35 / 3.27
HDL-SLAM	3.04 / 1.05	9.29 / 2.72	0.80 / 0.46	0.89 / 0.29	0.72 / 0.95	2.45 / 0.92	2.35 / 0.71	1.11 / 0.82
LIOM	9.75 / 2.37	18.85 / 5.08	9.75 / 2.37	17.24 / 6.33	0.20 / 0.48	21.69 / 9.06	10.96 / 6.54	4.75 / 2.60
DLO	2.37 / 0.53	6.36 / 0.94	1.50 / 0.43	0.48 / 0.22	0.95 / 0.49	4.96 / 1.51	3.12 / 0.46	4.06 / 0.74

To fully assess the trajectories, Absolute Trajectory Error (ATE), a more intuitive error

metric, is utilized for evaluation defined as (4.61),

$$ATE_i = \mathcal{T}_{est,i}^{-1} \cdot \mathcal{T}_{ref,i} \quad (4.61)$$

where $\mathcal{T}_{est,i}$ refers to the pose estimated and $\mathcal{T}_{ref,i}$ is the ground truth, and the root mean squared error (RMSE) is computed by (4.62). It is worth noting that RMSE is computed by rotational part in degree unit or translational part in meter unit.

$$RMSE = \sqrt{\frac{1}{N} \sum_{i=0}^N ATE_i^2} \quad (4.62)$$

The translational part of ATE comparison results are shown in Table 4.3. From the results of Drifting error and ATE error, we can see that our proposed Invariant-DLIO is able to get the best and most robust performance in different sequences. LIOM can perform better on Sequence 07 in the ATE comparison, but it is not comparable with Invariant-DLIO in other sequences. Fast-LIO and DLO can achieve comparable performances in part of the dataset, but both of them are under-performing on some sequences.

Table 4.3: RMSE of ATE (m) Evaluation of KITTI Dataset

	00	02	05	06	07	08	09	10
Inv-DLIO	3.52	6.35	2.73	1.33	1.67	7.46	2.25	1.06
LC-LIO	4.69	8.17	2.89	2.49	2.05	13.80	4.21	3.65
Fast-LIO	42.49	12.41	3.59	9.30	2.62	17.21	2.69	4.80
LINS	43.47	85.87	4.40	7.56	22.91	22.51	132.83	52.62
HDL-SLAM	22.44	98.26	4.96	5.68	4.41	19.26	21.15	5.58
LIOM	85.43	137.34	57.90	175.68	1.18	143.78	69.19	22.60
DLO	5.68	8.04	2.58	1.64	1.82	21.61	6.92	2.35

4.3.4 High-Frequency Motion Evaluation

For general navigation cases, the data collected by robotic platforms with high-frequency motions are considered more challenging than autonomous driving scenarios, which are important for the assessment. The University of Michigan North Campus Long-Term

Table 4.4: RMSE of ATE (m) Evaluation of NCLT Dataset and Newer College Dataset

	NCLT_1	NCLT_2	NCLT_3	NCLT_4	NCLT_5	NCD_1	NCD_2	NCD_3	NCD_4
Inv-DLIO	0.73	1.84	1.57	2.64	2.92	0.33	0.24	0.08	0.17
LC-LIO	1.23	18.66	15.94	18.42	12.31	2.31	1.99	0.56	0.55
Fast-LIO	0.87	2.35	1.61	2.74	3.13	1.28	1.06	0.31	0.49
LINS	28.30	151.22	fail	fail	fail	fail	fail	0.86	1.14
HDL-SLAM	28.93	38.48	82.29	fail	fail	2.57	1.64	0.97	2.47
LIOM	fail	fail	fail	fail	fail	fail	fail	0.48	2.08
DLO	1.56	2.37	7.26	11.84	4.72	0.43	0.96	0.17	0.31

(NCLT) Dataset is a typical dataset collected by the Segway robot with Microstrain GX3 IMU and Velodyne HDL-32E LiDAR, and the data frequency of IMU and LiDAR are 100 Hz and 10 Hz respectively. In the experiments, five sequences from the NCLT dataset are selected, NCLT_1 (20130110), NCLT_2 (20120615), NCLT_3 (20120429), NCLT_4 (20120115), and NCLT_5 (20120511). For NCLT dataset, the parameter is set as $S_{IVM} = 1.0\text{m}$, based on the best estimation results. To fully assess the performance of Invariant-DLIO, we also used the Newer College Dataset, which is collected by hand-held OS-128 LiDAR with an internal IMU. Four sequences from the Newer College Dataset are selected for evaluation, NCD_1 (2020-03-10), NCD_2 (2021-11-30), NCD_3 (math-1), and NCD_4 (math-2). For NCD dataset, the parameter is set as $S_{IVM} = 1.2\text{m}$, based on the best estimation results. The ATE evaluation results are shown in Table 4.4. From the table, we can see that our proposed Invariant-DLIO outperforms all other benchmarks.

4.3.5 Custom Robotic Dataset

As described before, custom datasets are collected with a Magni Silver robot [61], a typical differential drive robot. The control is highly dependent on the feedback from wheel encoders, where the measurements are noisy and easy to interfere by slippery, which makes the robot containing uneven motions even with certain control inputs. In the corresponding experiments, the parameter is set as $S_{IVM} = 0.75\text{m}$, based on the best estimation results.

4.3.5.1 Indoor Dataset

We firstly collected the dataset with 6820 frames of point clouds, associated with IMU data of 200 Hz frequency. The data collection scene is a long, textureless corridor in PolyU. The mapping visualization comparison is similar with the performance of Inc-DLOM, as mentioned in 3.3.5. For brevity, the images are not shown in this section.

4.3.5.2 Campus Dataset

In addition, a dataset with 3000 frames of point cloud and 200 Hz IMU data is collected on the PolyU campus, which is an outdoor scenario. The point cloud mappings are shown in Fig. 4.9. The mapping results are similar to the indoor scenarios, DLO failed in tracking robot's movements, while Fast-LIO provides a proper but lower quality mapping than our proposed Invariant-DLIO. The campus dataset is collected on the podium level of PolyU, which is a horizontal plane. The proper estimated trajectory should be staying static on the z-axis. We compared the z-axis plots of Fast-LIO and our proposed Invariant-DLIO as shown in Fig.4.7, where Fast-LIO's estimation on the z-axis kept increasing. Due to the sparsity of LiDAR point clouds, drifting error on the z-axis is always the main difficulty for most LO or LIO methods. As a feature-based LIO, FAST-LIO's z-axis estimation error could be amplified by insufficient feature point extraction in vertical directions. In comparison, our proposed Invariant-DLIO is improving the estimation performance by direct registration methods, as the experimental result indicated.

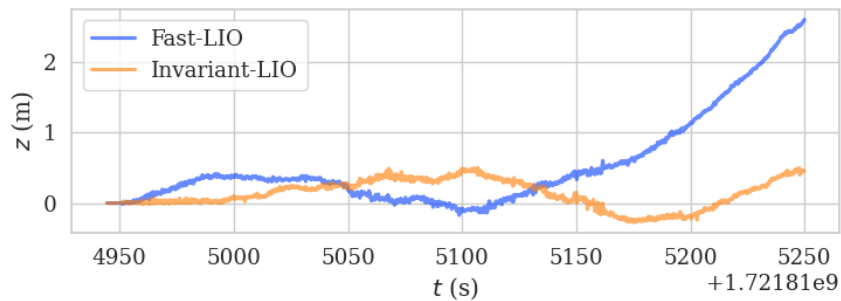


Figure 4.7: Trajectory plot of Z-axis.

4.3.6 Computational Efficiency

We also evaluated the average computational time per LiDAR frame of Invariant-DLIO in comparison with the chosen methods. The experiments are taken on one sequence from KITTI, NCLT, and Newer College Dataset respectively. The test computer is based on an AMD Ryzen 9 7940HS CPU with 4.00 GHz frequency, and the comparison result is shown in Fig. 4.8. The results show that Invariant-DLIO has the fastest processing speed and can achieve over 50 Hz in a low-cost CPU environment, and the efficiency performance is more stable than other benchmarks.

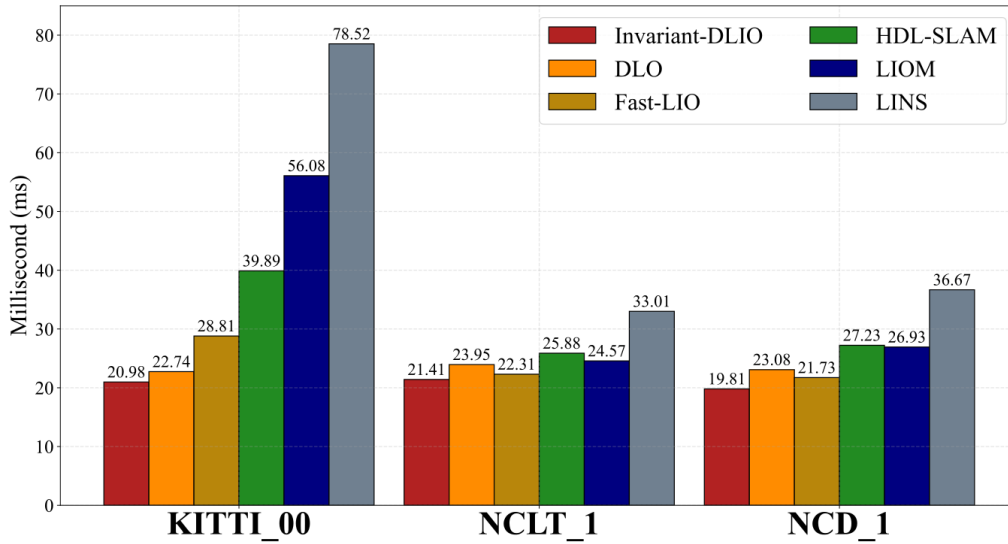
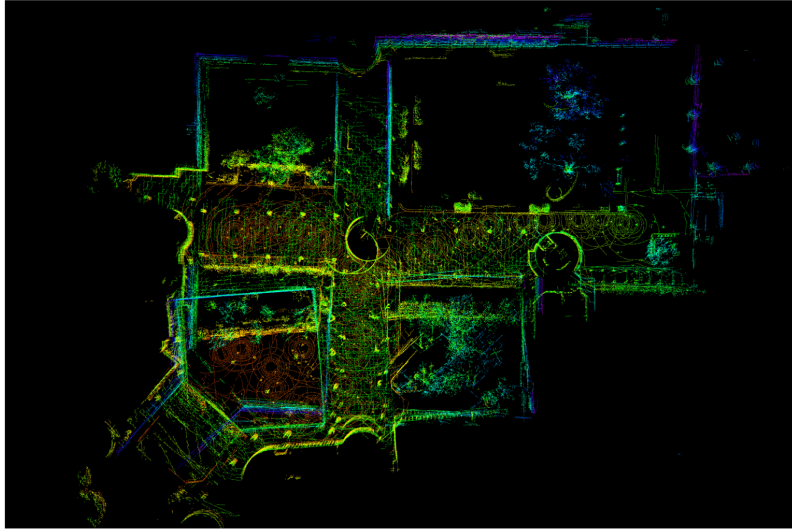


Figure 4.8: Computation Time per Frame.

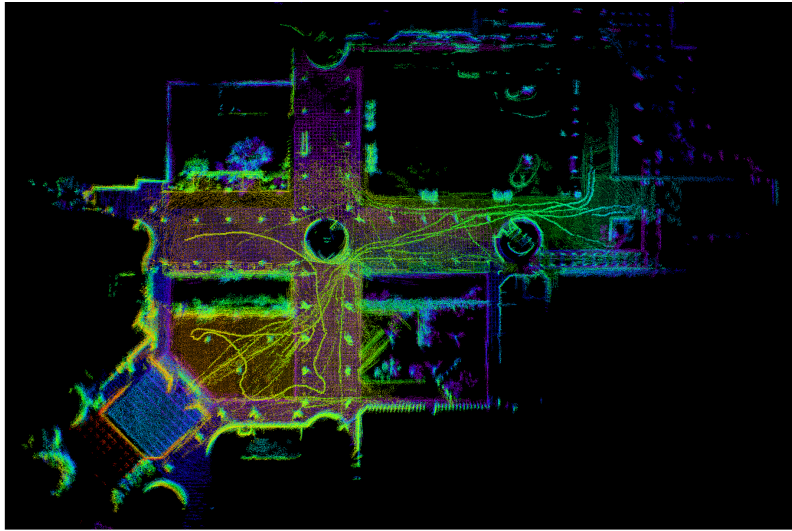
4.4 Conclusion

We proposed a novel LiDAR-Inertial Odometry framework, Invariant-DLIO in this paper, which is developed with InEKF theory and direct registration method. Invariant-DLIO also contains the module of Incremental Voxel Mapping, which is capable of preserving consistent historical mapping information and efficient correspondence searching with limited computational cost. Evaluation of Invariant-DLIO on different aspects demonstrates that the proposed method provides state-of-the-art localization performance in mul-

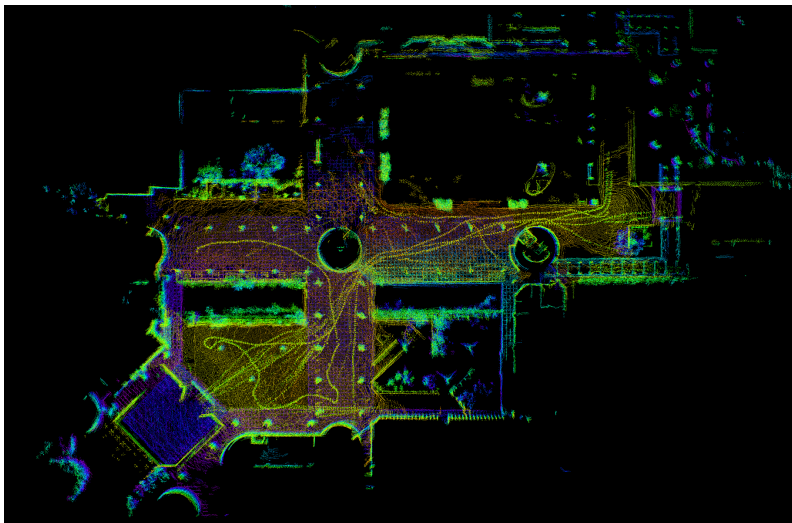
tiple working scenarios. As future works, the extrinsic calibration for IMU and LiDAR frames can be implemented into our proposed Invariant EKF estimator.



(a) DLO



(b) Fast-LIO



(c) Invariant-DLIO

Figure 4.9: The point cloud mapping of campus garden.

Chapter 5

Deploying SLAM: Applications

While the proposed direct LiDAR Odometry (LO) and LiDAR-Inertial Odometry (LIO) frameworks have been validated on conventional PC-based platforms, their robustness and computational efficiency remain unverified under embedded-system constraints, which is critical for real-world robotics applications. This chapter evaluates the portability of these frameworks to resource-limited embedded hardware, addressing the performance degradation risks inherent in such transitions.

5.1 Motivation

While numerous LiDAR Odometry (LO) and LiDAR-Inertial Odometry (LIO) methods achieve high accuracy and computational efficiency on high-end PC platforms, their evaluation predominantly relies on public datasets and desktop-grade hardware. This overlooks critical challenges in deploying these algorithms to resource-constrained embedded systems (e.g., NVIDIA Jetson series), where limited computational power induces severe performance degradation or systemic failure. Key operations, such as large-scale matrix inversion, KD-tree construction/query, and iterative point cloud processing, exhibit significantly reduced agility on embedded CPUs due to lower clock frequencies, restricted parallelization capabilities, and thermal throttling. As illustrated in Figure 5.1, embedded CPUs demonstrate substantially lower computational benchmarks (e.g., 4–8× lower

FLOPS than desktop counterparts), leading to unstable frame rates, increased trajectory drifting from delayed loop closures, and catastrophic tracking loss during high-speed maneuvers. Without special adaptations, embedded deployment risks compromising SLAM robustness despite of PC-proven efficiency.

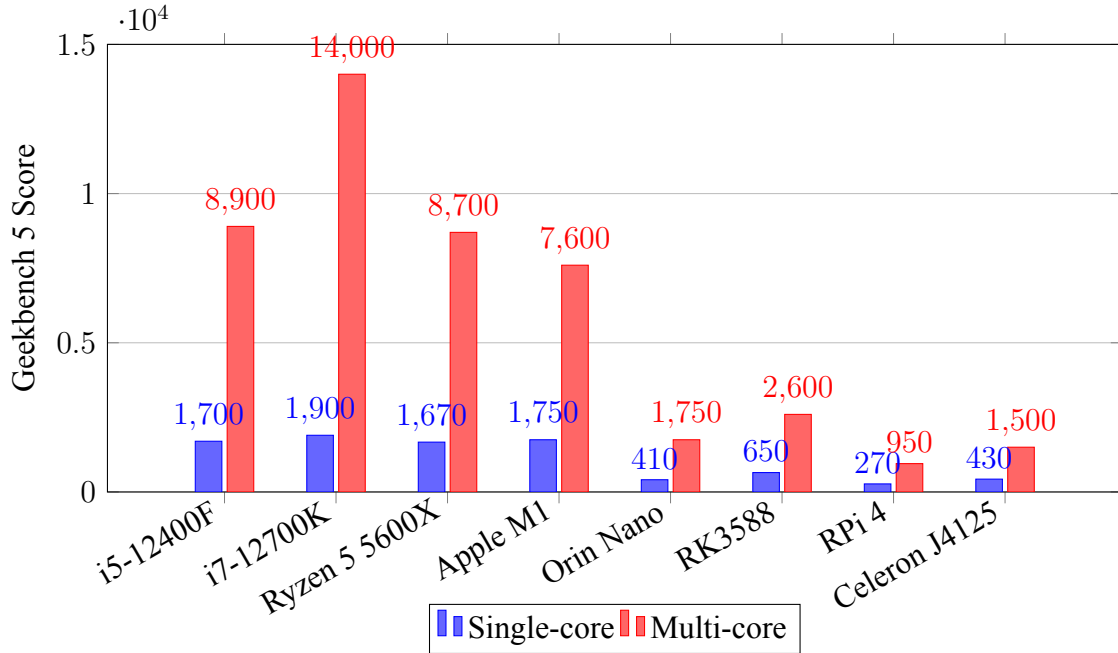


Figure 5.1: Approximate Geekbench 5 Scores for a Range of Desktop, Embedded, and Low-cost CPUs

The limitations mentioned inspired our future development of deploying our SLAM algorithm, especially on an embedded system, which is popular for autonomous robotics.

5.2 PolyU-F1Tenth

F1Tenth (or F1/10) is an open-source, 1/10th-scale autonomous racing platform and educational initiative originating from the University of Pennsylvania. It provides affordable hardware designs and software stacks, enabling students and researchers to build small autonomous race cars. The primary goals are to teach core robotics concepts (perception, planning, control, estimation) through hands-on projects and to foster cutting-edge research in autonomous navigation by hosting competitive racing events where these scaled vehicles must drive at high speeds safely and efficiently. It serves as a vital testbed for



Figure 5.2: F1Tenth Hardware [75]

developing and benchmarking algorithms relevant to full-scale autonomous vehicles in a safer and more accessible environment.

The implementation of F1Tenth is shown in Fig. 5.2. The platform employs a standardized, open-source sensor suite anchored by a Hokuyo 10LX 2D LiDAR (270deg FoV, 10m range, 40Hz), an Intel RealSense D455 depth camera. Sensor data is processed on embedded NVIDIA Jetson edge computers (e.g., Orin Nano), which handle SLAM, path planning, and control while adhering to strict computational constraints.

However, F1Tenth sensor suite contains limited measurement ranges, which limited its application scenarios at indoor, small-scale environments. In addition, embedded compute-

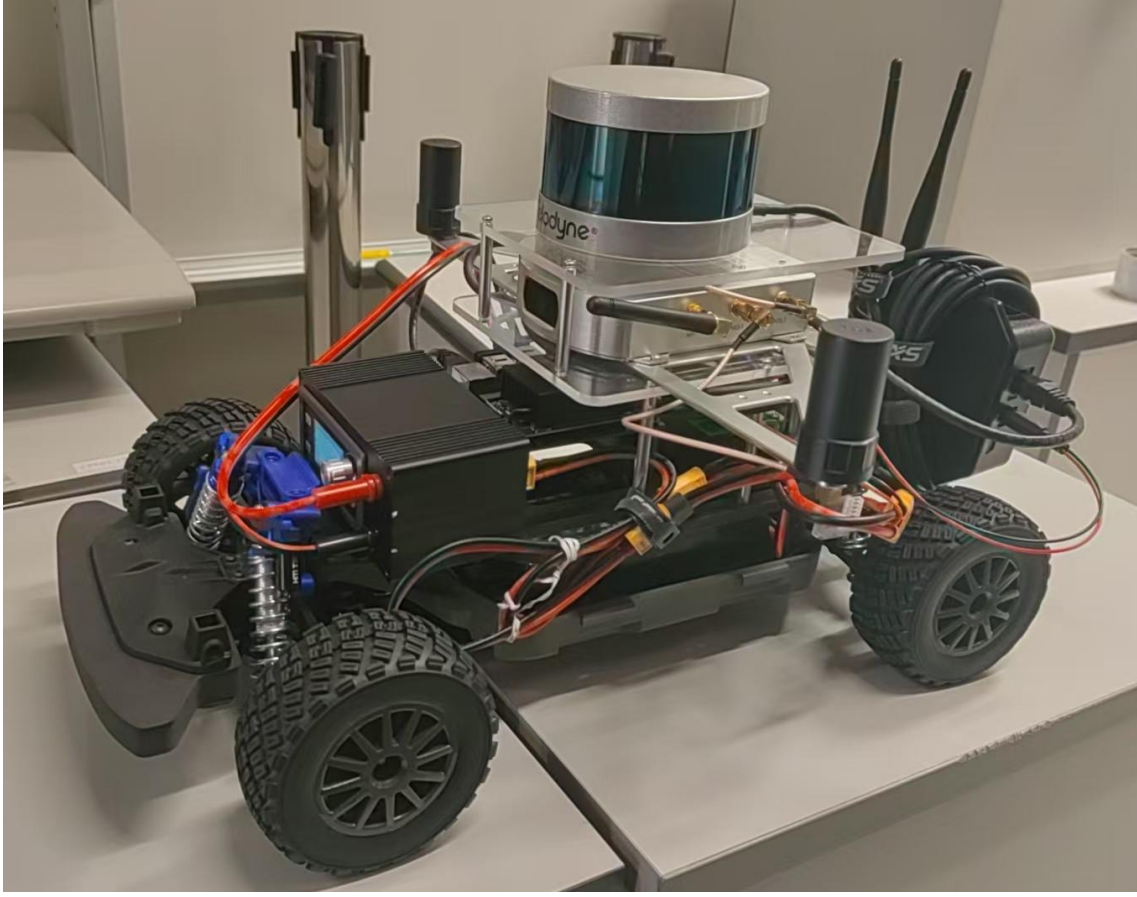


Figure 5.3: PolyU-F1Tenth Hardware [75]

power ceilings force trade-offs between measurement fidelity and behavior accuracy. Inspired by F1Tenth, we have developed our own designed PolyU-F1Tenth model based on the open-source configurations. To support outdoor environment for higher speed operation, the mechanical structure of the chassis is strengthened. Instead of 2D LiDAR, the Velodyne Pack 16 (VLP-16) is installed for enhanced perception. Fixposition Vision-RTK is also installed as a high-resolution GNSS assistance for outdoor scenarios. The implementation of PolyU-F1Tenth is shown in Fig. 5.3.

As benchmarked in Fig. 5.1, the NVIDIA Jetson Orin Nano deployed on PolyU-F1Tenth exhibits significantly lower computational throughput than desktop-grade CPUs. This deficit imposes severe latency on LiDAR Odometry (LO) processing, limiting state-of-the-art LO/LIO frameworks (including our proposed methods) to around 15 Hz mean update rates. At the platform’s native 10 Hz LiDAR acquisition frequency, processing

delays exceeding 100 ms per frame induce intermediate frame drops, triggering tracking failure during high-velocity operation. These limitations are compounded by the platform’s aggressive kinematics, which amplify pose estimation uncertainty under resource-constrained computation.

To mitigate these bottlenecks, we propose a heterogeneous CPU-GPU co-processing architecture leveraging the Jetson’s embedded GPU for parallelizable workloads. While GPU offloading offers potential acceleration for computationally intensive LO tasks, the architectural divergence between CPU and GPU execution models necessitates targeted algorithmic redesign. Without structural adaptations, including memory access optimization, kernel parallelization, and asynchronous pipeline scheduling, naive GPU porting may introduce overhead that degrades real-time performance.

5.3 Proposed Method

The proposed CUDA-Accelerated LiDAR Odometry is shown in Figure 5.10. Our proposed CUDA-accelerated LO is divided into CPU host and GPU device computation. The point cloud $\mathcal{P}_k$ is first copied into the GPU device and then undergoes **CUDA Preprocess**, which includes downsampling and constant velocity deskew. The processed cloud is transformed into the CPU host side for **Voxelization** to generate the set of voxels. **S2S matching** computes the relative transformation and the initial guess for S2M matching. The set of voxels is then copied into the GPU device for CUDA-based IVM updating. Incremental GICP is implemented with CUDA acceleration to compute the global pose.

For the k -th point cloud input, noted as,

$$\mathcal{P}_k = \{\cdots, p_i, \cdots \mid \forall p_i \in \mathbb{R}^3\} \quad (5.1)$$

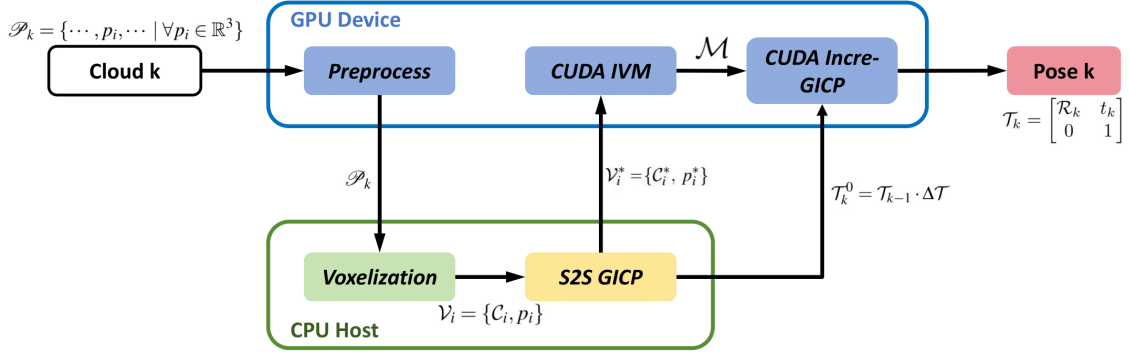


Figure 5.4: CUDA-Accelerated LO Framework.

Inc-DLOM is estimating the global pose $\mathcal{T}_k$. $\forall \mathcal{T} \in \text{SE}(3)$ has the general form of

$$\mathcal{T} = \begin{bmatrix} \mathcal{R} & t \\ 0 & 1 \end{bmatrix} \quad (5.2)$$

where $\mathcal{R} \in \text{SO}(3)$ represents the rotation matrix, $t \in \mathbb{R}^3$ represents the translation vector.

The point p can be transformed by $\mathcal{T}$ as,

$$\mathcal{T} \cdot p := \mathcal{R} \cdot p + t \quad (5.3)$$

These definitions will be adopted throughout the paper.

5.3.1 Point Cloud Preprocessing

The point cloud will first be preprocessed, including downsampling by Voxel Grid Filter and deskewing based on the constant velocity model. For $\mathcal{P}_k$, the motion velocity is defined as (5.4), which is derived from $\mathcal{T}_{k-1}$ and $\mathcal{T}_k$.

$$\Delta \mathcal{T} = \mathcal{T}_{k-1}^{-1} \cdot \mathcal{T}_k \quad (5.4)$$

Then the motion velocity $\xi \in \mathfrak{se}(3)$ is derived from (5.4) as follows:

$$\xi = \frac{\text{Log}(\Delta\mathcal{T})}{\Delta\epsilon} = \begin{bmatrix} \omega \\ \nu \end{bmatrix} \quad (5.5)$$

Where $\Delta\epsilon$ is the relative time of single point cloud sensing, which is usually around 0.1 s for mechanical LiDAR. The operator $\text{Log}(\cdot): \text{SE}(3) \rightarrow \mathbb{R}^6$ is the matrix logarithm for $\text{SE}(3)$. ω and ν represent the lie algebra for the rotation and translation parts respectively.

For the typical mechanical LiDAR sensing paradigm, points in one point cloud are measured at different relative timestamps. For $p_i \in \mathcal{P}$, the data is measured with a relative timestamp $s_i \in [0, \Delta\tau)$ from the first measurement. Then $\forall p_i \in \mathcal{P}_k$ is deskewed as:

$$p_i = \text{Exp}(s_i \cdot \xi) \cdot p_i \quad (5.6)$$

where the operator $\text{Exp}(\cdot): \mathbb{R}^6 \rightarrow \text{SE}(3)$ is the matrix exponential for $\mathfrak{se}(3)$.

However, the deskew step requires extensive usage of the matrix exponential operation, which will increase the processing time of a single point cloud. In our original Inc-DLOM implementation, this step will cost around 6 milliseconds. Besides, the Point Cloud Library (PCL) [67] implementation of voxel grid filter is also not efficient enough.

5.3.2 CUDA-Preprocessing

The inefficient PCL implementation and motion distortion motivate us to accelerate the preprocessing step. Although the original process is sequential and accelerated with CPU multi-threading, it remains inefficient for embedded platforms like Jetson.

5.3.2.1 Voxel-Based Downsampling via GPU Hash Table

Raw point cloud data, $p_i = (x_i, y_i, z_i)$, is first downsampled using a voxel grid. Each point is assigned to a voxel key:

$$key_i = \left(\left\lfloor \frac{x_i}{v_{\text{size}}} \right\rfloor, \left\lfloor \frac{y_i}{v_{\text{size}}} \right\rfloor, \left\lfloor \frac{z_i}{v_{\text{size}}} \right\rfloor \right)$$

A custom hash function is used to map voxel indices to GPU memory via a device-side hash table (`stdgpu::unordered_map` [74]), enabling parallel insertion with CUDA kernel functions. This step reduces computational cost while preserving structural fidelity. The spatial hash function is defined as

$$h(p_i) = ((x_i * a) \oplus (y_i * b) \oplus (z_i * c)) \% d, \quad (5.7)$$

$$a = 73856093, b = 471943, c = 83492791, d = 1 \times 10^7,$$

where *xor* is exclusive OR, and % is the modulus operator.

5.3.2.2 CUDA-Based Matrix Exponential for SE(3)

For each point, the relative time α_i within the scan is normalized to $[0, 1]$. The per-point pose $\mathbf{T}_i$ at time t_i is obtained by:

$$\mathbf{T}_i = \text{Exp}(s_i \cdot \text{Log}(\Delta \mathcal{T})), \quad (5.8)$$

$$s_i = \frac{t_i - t_{\min}}{t_{\max} - t_{\min}} \quad (5.9)$$

The matrix exponential $\text{Exp}(\cdot)$ for SE(3) is computed via closed-form expressions, which are highly amenable to CUDA parallelization:

1. SO(3) Exponential (Rodrigues' Formula):

$$\exp([\omega]_{\times}) = I + \frac{\sin \theta}{\theta} [\omega]_{\times} + \frac{1 - \cos \theta}{\theta^2} [\omega]_{\times}^2$$

where $\theta = \|\omega\|$. The code uses a Taylor expansion for small angles to ensure numerical stability.

2. SE(3) Exponential:

$$\text{Exp}(\hat{\xi}) = \begin{bmatrix} \exp([\omega]_{\times}) & V(\omega) \nu \\ 0 & 1 \end{bmatrix}$$

$$\text{where } V(\omega) = I + \frac{1-\cos\theta}{\theta^2}[\omega]_{\times} + \frac{\theta-\sin\theta}{\theta^3}[\omega]_{\times}^2.$$

CUDA device functions implement these formulas in single precision, processing each transformation independently and concurrently across thousands of threads. Then, a CUDA kernel is implemented for the motion-compensated transformation for each point as

$$\mathbf{p}'_i = \mathbf{T}_i \cdot \mathbf{p}_i, \quad (5.10)$$

where $\mathbf{T}_i$ is a 4×4 homogeneous matrix, and $\mathbf{p}_i$ is augmented as a homogeneous vector. Because the expensive matrix-vector operations are avoided, CUDA parallel process is suitable for the accelerated computation.

5.3.3 Scan-to-Scan (S2S) Matching

The S2S Matching process is described in the Chapter of Inc-DLOM. The details will be mentioned here briefly for consistency.

5.3.3.1 Voxelization

$\mathcal{P}_k$ will be transformed from a point set into a voxel set $\mathcal{P}_k$, defined as (5.11).

$$\mathcal{P}_k \rightarrow \mathcal{P}_k = \{\dots, \mathcal{V}_i, \dots\} = \bigcup_{i=0}^n \mathcal{V}_i, \quad \mathcal{V}_i = \{\mathcal{C}_i, p_i\}. \quad (5.11)$$

The point cloud data is downsampled into a set of voxels, where every voxel consists of local surface covariance $\mathcal{C}_i$ and central coordinate p_i . $\mathcal{C}_i$ is computed from 15

nearest neighboring points of p_i , and the nearest neighboring search is performed by NanoFLANN-based KD-Tree. Since the KD-Tree structure is not sufficient for GPU processing due to if-else conditioning, and copying data across host-device would be time-consuming, we kept this part under CPU computation.

5.3.3.2 Voxel-to-Voxel (V2V) GICP

V2V GICP transformed GICP’s objective into a “Voxel-to-Voxel” manner, which is defined as (5.12). $\mathcal{V}_i^S \in \mathcal{P}^S$ and its closest neighboring voxel $\mathcal{V}_i^T \in \mathcal{P}^T$ are taken for optimization inputs. Σ is the voxel-to-voxel covariance defined in (5.13).

$$\mathcal{T}^* = \arg \min_{\mathcal{T} \in \text{SE}(3)} \sum_i \|p_i^T - \mathcal{T} \cdot p_i^S\|_{\Sigma^{-1}}^2 \quad (5.12)$$

$$\Sigma = \mathcal{C}_i^M + \mathcal{R} \cdot \mathcal{C}_i^S \cdot \mathcal{R}^T \quad (5.13)$$

The initial guess (5.14) for Scan-to-Mapping registration is computed from V2V GICP.

$$\mathcal{T}_k^0 = \mathcal{T}_{k-1} \cdot \Delta \mathcal{T}. \quad (5.14)$$

5.3.4 CUDA Incremental Voxel Mapping (CUDA-IVM)

In this work, we present a highly parallel GPU implementation of incremental voxel-based mapping for GICP registration, leveraging a device-resident concurrent hash map to enable real-time scalability. Our approach utilizes the `stdgpu::unordered_map` [74] as the underlying data structure for storing and updating voxel information, supporting concurrent insertions and queries directly on the GPU.

5.3.4.1 CPU IVM

To illustrate the differences between CUDA-IVM and CPU IVM, the insertion and updating process is shown in Fig. 5.5. The figure showed an example case that a point cloud contains N points, in which different color represents different spatial key (described in

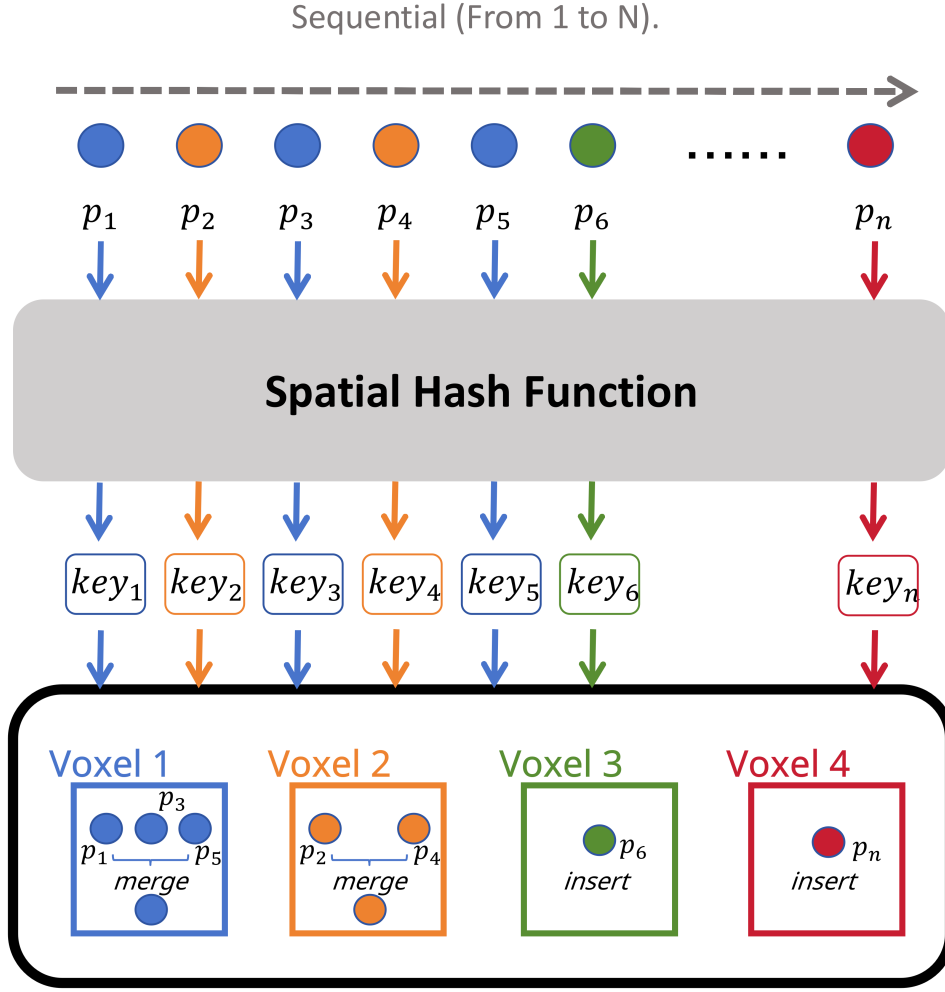


Figure 5.5: Sequential IVM updating with input voxels.

Section 5.3.2.1). The term sequential means that the deterministic order is assigned to different voxels, which is implemented by CPU multiprocessing in IVM.

As the example case shown in Fig. 5.5, the voxels will be inserted into IVM at their first appearance, after that new voxels will be merged into IVM's existing voxel. Given two voxels $\mathcal{V}_A = \{p_A, \mathcal{C}_A\}$ and $\mathcal{V}_B = \{p_B, \mathcal{C}_B\}$, the merging operation is described as follows, $\mathcal{V}_A$ and $\mathcal{V}_B$ contain m and n points. In our condition, $m = n = 15$, from GICP's nearest neighboring search number.

Mean Update

$$p_{\text{new}} = \frac{mp_A + np_B}{m + n} \quad (5.15)$$

Covariance Update

$$\mathcal{C}_{\text{new}} = \frac{1}{m+n} \left[m \left(\mathcal{C}_A + (p_A - p_{\text{new}})(p_A - p_{\text{new}})^\top \right) + n \left(\mathcal{C}_B + (p_B - p_{\text{new}})(p_B - p_{\text{new}})^\top \right) \right] \quad (5.16)$$

Covariance Regularization To ensure numerical stability and avoid degeneracy in the incremental update of voxel covariances, we perform a regularization step after each covariance update. Given the updated covariance matrix Σ , we compute its eigen-decomposition:

$$\mathcal{C}_{\text{new}} = U \Lambda U^\top \quad (5.17)$$

where U is the orthogonal matrix of eigenvectors and Λ is the diagonal matrix of eigenvalues $(\lambda_1, \lambda_2, \lambda_3)$.

We then replace the eigenvalues with a set of prescribed values to limit the condition number and ensure a minimum variance in all directions:

$$\tilde{\Lambda} = \text{diag}(\tilde{\lambda}_1, \tilde{\lambda}_2, \tilde{\lambda}_3) \quad (5.18)$$

where, for example, $\tilde{\lambda}_1 = \tilde{\lambda}_2 = 1$ and $\tilde{\lambda}_3 = 0.001$. The regularized covariance is then reconstructed as:

$$\mathcal{C}_{\text{new}} = U \tilde{\Lambda} U^\top \quad (5.19)$$

5.3.4.2 CUDA Implementation

The major challenge of implementing IVM for CUDA devices is the *race condition*, which commonly arises when multiple threads attempt to insert, update, or merge voxels associated with the same grid cell. Unlike the CPU, the GPU device performs *for-loop* in a random order, which means the race condition that several CPU threads update the same IVM cell is expected, leading to erroneous mapping construction and update.

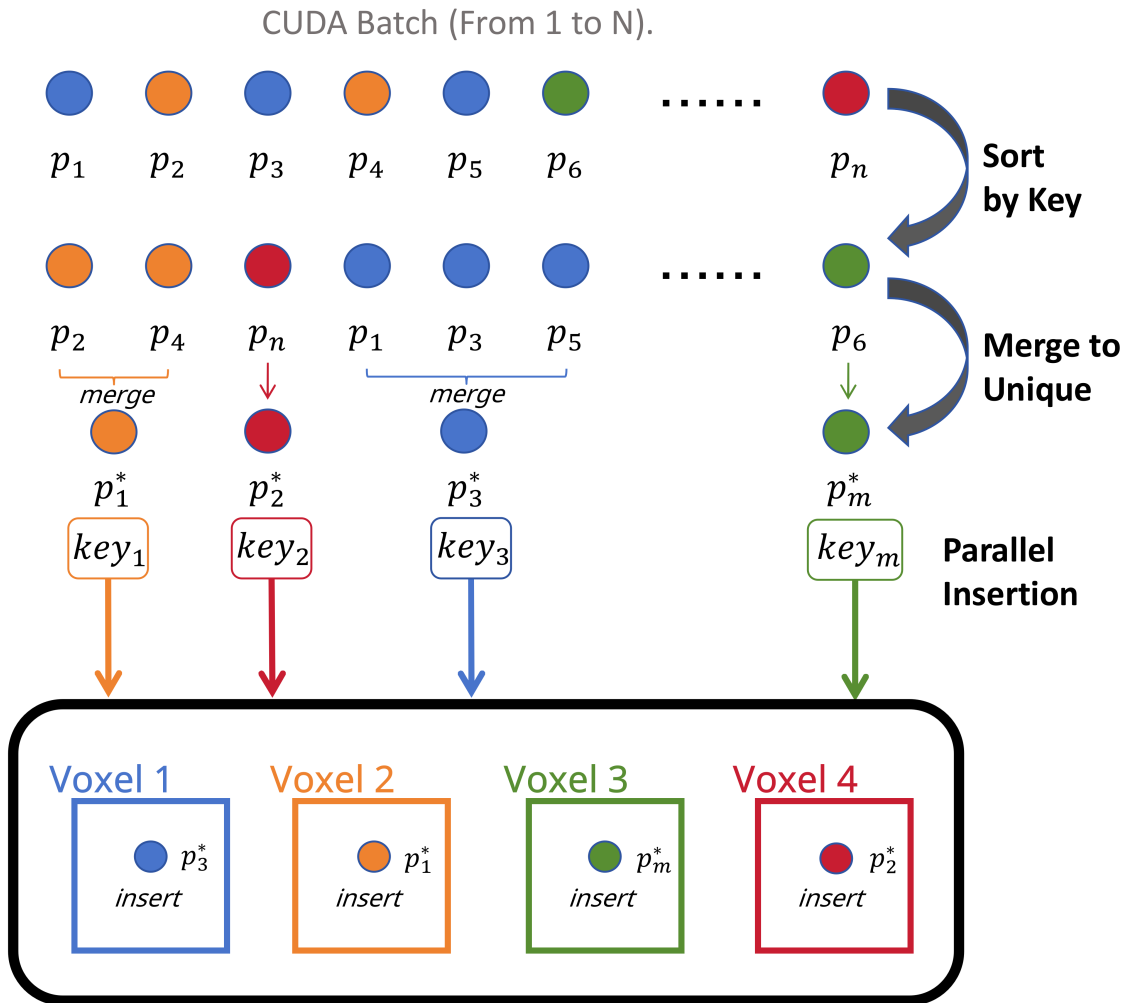


Figure 5.6: CUDA-IVM updating with input voxels.

To address the challenge of race conditions, we employ a carefully designed multi-stage processing pipeline, as shown in Fig. 5.6.

Sort by Key In our framework, each voxel $\mathcal{V}_i = \{p_i, \mathcal{C}_i\}$ is assigned a discrete voxel key $\mathbf{k}_i = (k_x, k_y, k_z)$, computed by quantizing its coordinates according to the voxel grid resolution. To enable efficient aggregation and update of voxels, we employ a key-based sorting mechanism defined as follows:

1. **Key Extraction:** For each point, the voxel key is computed as:

$$\mathbf{k}_i = \left(\left\lfloor \frac{p_{i,x}}{s} \right\rfloor, \left\lfloor \frac{p_{i,y}}{s} \right\rfloor, \left\lfloor \frac{p_{i,z}}{s} \right\rfloor \right)$$

where s is the voxel size.

2. **Lexicographical Ordering:** The keys are sorted according to lexicographical order, when $\mathbf{k}_a < \mathbf{k}_b$,

$$\begin{cases} k_{a,x} < k_{b,x}, & \text{or} \\ k_{a,x} = k_{b,x} \text{ and } k_{a,y} < k_{b,y}, & \text{or} \\ k_{a,x} = k_{b,x}, k_{a,y} = k_{b,y}, \text{ and } k_{a,z} < k_{b,z} \end{cases}$$

3. **Parallel Sorting:** The set of key-value pairs $(\mathbf{k}_i, \mathcal{V}_i)$ is sorted using parallel sorting primitives, `thrust::sort_by_key` [5] on the GPU. This ensures that all voxels mapping to the same key are stored contiguously in memory.
4. **Facilitating Aggregation:** The resulting sorted order allows efficient downstream aggregation and reduction operations, as all elements corresponding to the same voxel key can be processed collectively without additional synchronization.

This key-based sorting procedure is fundamental to our scalable and parallel incremental mapping pipeline, ensuring both deterministic processing and high throughput in large-scale point cloud registration tasks.

Merge to Unique Given a sequence of key-value pairs $(\mathbf{k}_i, \mathcal{V}_i)$ for $i = 1, \dots, N$, the *reduce by key* algorithm groups all values with identical keys and applies an associative reduction operator $\oplus$ to the values within each group, the *Merge to Unique* is summarized as:

1. **Reduction:** The values within each segment are combined using an associative reduction operator $\oplus$, which is described as 5.3.4.1,

$$\mathcal{V}_u = \mathcal{V}_{j_1} \oplus \mathcal{V}_{j_2} \oplus \dots \oplus \mathcal{V}_{j_m}$$

where $\mathcal{V}_u$ denotes the aggregated voxels for key $\mathbf{k}_u$.

2. **Output:** The algorithm outputs the set of unique keys $\{\mathbf{k}_u\}$ and their corresponding reduced voxels $\{\mathcal{V}_u\}$.

This operation efficiently aggregates all values associated with the same key, producing a set of unique keys and their corresponding reduced values.

Parallel Insertion After sorting and merging, the remaining voxels are unique in the IVM grids, allowing efficient CUDA-based insertion. The insertion operation is invariant to the data order and avoiding all race conditions.

5.3.5 CUDA-Accelerated Incremental GICP (CUDA IGICP)

We implement a GPU-accelerated version of the Generalized Iterative Closest Point (GICP) algorithm to efficiently align a source point cloud to a target map represented as a set of voxel-wise Gaussian distributions. The alignment proceeds as follows:

1. **Input Preparation:** The target map $\mathcal{M}$ is represented on the device as a concurrent hash map, the source voxel set is transformed from host (CPU) memory to device (GPU) memory for Scan-to-Map refinement, with the initial guess as (5.14).

2. **Iterative Optimization Loop:** For each iteration until convergence or a maximum number of iterations:

(a) **Transformation Update:** The current estimate $(\mathbf{R}, \mathbf{t})$ is extracted from $\mathbf{T}_k$. These are copied to the device for use in the CUDA kernels.

(b) **Correspondence and Cost Construction (CUDA Kernel):**

- Each source point is transformed into the target frame using the current estimate.
- The corresponding target voxel is retrieved from the map using a quantized voxel key.
- If a valid correspondence is found and is within a predefined distance threshold, the point-to-voxel cost terms are computed:
 - The combined covariance is formed as $\Sigma = \Sigma_{\text{tgt}} + \mathbf{R}\Sigma_{\text{src}}\mathbf{R}^\top$.
 - The Jacobian matrix J is constructed for the rigid transformation.
 - The local Hessian A_i and gradient b_i are computed as:

$$A_i = J^\top \Sigma^{-1} J, \quad b_i = -J^\top \Sigma^{-1} (\mathbf{p}_{\text{tgt}} - \mathbf{p}_{\text{src}})$$

- All A_i and b_i are stored in device memory.

(c) **Parallel Reduction (CUDA Kernel):** The per-point contributions A_i and b_i are reduced in parallel to obtain the global system:

$$A = \sum_i A_i, \quad b = \sum_i b_i$$

(d) **Transformation Update (Host):** The global system (A, b) is transferred to the host. The update step $\Delta\xi$ is obtained by solving $A\Delta\xi = b$ (e.g., via LDLT decomposition). The transformation estimate is updated via the exponential map:

$$\mathbf{T}_k \leftarrow \exp(\hat{\Delta\xi})\mathbf{T}_k$$

- (e) **Convergence Check:** The alignment loop terminates if the transformation change is below a preset threshold or if the maximum number of iterations is reached.
3. **Result:** The final transformation $\mathbf{T}_k$ aligns the source point cloud to the target map with high efficiency enabled by GPU parallelism.

This CUDA-based GICP implementation leverages device-side data structures, thread-level parallelism for correspondence search and cost computation, and fast parallel reduction to achieve real-time performance for large-scale point cloud registration.

5.4 Experiments

The experiments comprise two phases. First, the CUDA-accelerated LiDAR Odometry (LO) module is validated on a high-end host computer equipped with an Intel i7 CPU and an NVIDIA RTX 3060 GPU. This environment significantly exceeds the computational requirements of our lightweight SLAM framework, enabling a feasibility assessment. Second, the optimized LO module is deployed on a Jetson Orin Nano, the designated computing platform for the PolyU-F1Tenth vehicle. Evaluation employs datasets from diverse LiDAR sensors, supplemented by a custom dataset collected on the PolyU campus using the F1Tenth platform.

5.4.1 CUDA Acceleration on Host Computer

The original Inc-DLOM implementation has been redeveloped for CUDA acceleration to facilitate future development. The vanilla implementation’s dependency on the Point Cloud Library (PCL) [67] presented limitations for deployment on resource-constrained platforms like Jetson, where PCL’s computational overhead is significant. Additionally, version compatibility issues in PCL frequently introduced bugs. To address these challenges while maintaining data structure consistency, we eliminated PCL dependencies

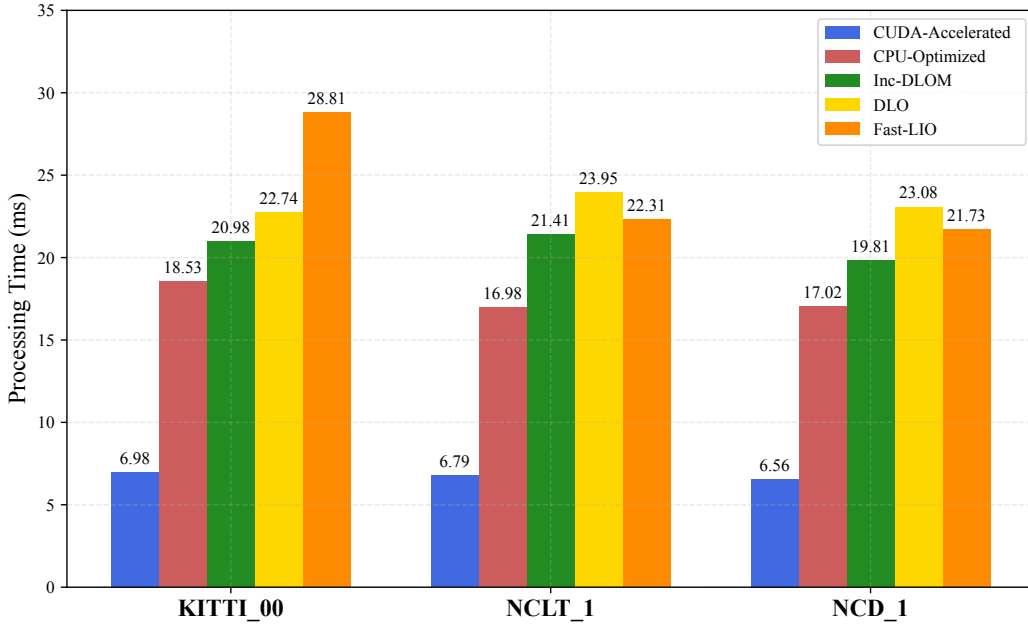


Figure 5.7: Processing Time per Frame

and reimplemented core functionality using exclusively standard library components, creating a CPU-optimized version. Our CUDA kernels and acceleration methods were subsequently developed based on this optimized CPU foundation. All evaluations were conducted on a system running Ubuntu 22.04 with CUDA 12.6, utilizing an Intel i5-12400F CPU and NVIDIA RTX 3060 GPU (6GB VRAM). Processing time comparisons across methods are shown in Fig. 5.7.

For processing speed assessment, we compared our previous Inc-DLOM implementation against Fast-LIO and DLO, excluding other benchmarks due to their relatively weaker performance. Our newly developed CPU-optimized and CUDA-accelerated versions were also evaluated. As Fig. 5.7 demonstrates, the CPU-optimized Inc-DLOM achieves significantly faster computation after removing unnecessary dependencies like PCL. The CUDA-accelerated LiDAR odometry (LO) reaches approximately 150 Hz processing frequency on PC hardware, representing state-of-the-art performance for LO systems. This advancement enables new possibilities for lightweight LiDAR odometry implementations.

To better illustrate the CUDA Acceleration, the computing time of different CUDA-Accelerated submodules are compared with the ones in CPU-Optimized version, including

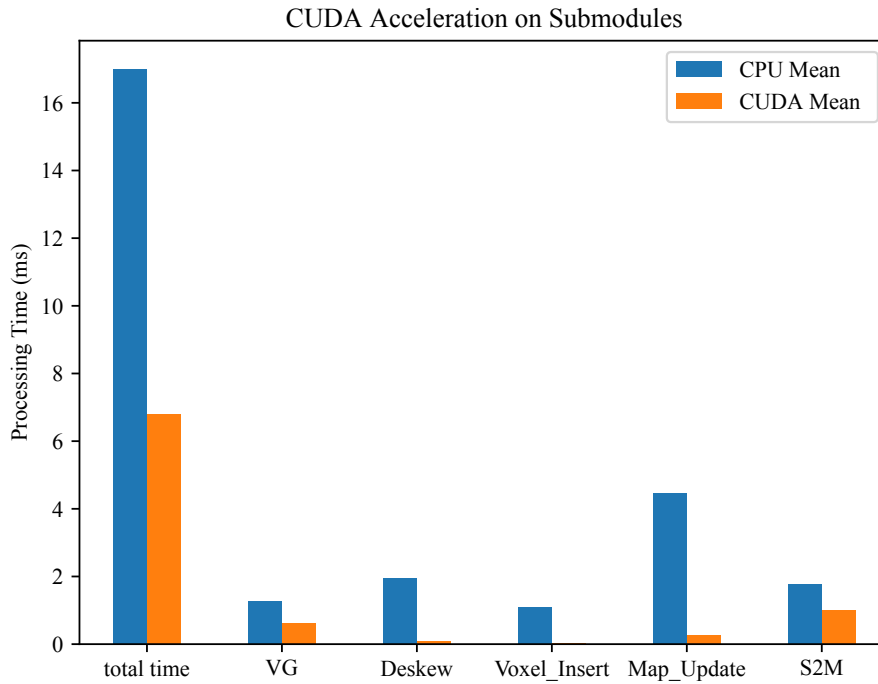


Figure 5.8: CUDA Acceleration on Submodules

Voxel Grid Downsampling (VG), point cloud deskew (Deskew), voxel insertion to IVM (Voxel_Insert), IVM update (Map_Update), and Scan-to-Mapping refinement (S2M). The processing time comparison is shown in Fig. 5.8.

As demonstrated in Fig. 5.8, CUDA acceleration yields substantial performance improvements across key computational procedures. Most notably, point cloud deskewing, voxel insertion into the Incremental Voxel Map (IVM), and IVM updating exhibit significant speedups. The former two operations benefit directly from parallel CUDA processing, where GPU architecture optimally accelerates the inherent parallelism in these tasks. The IVM update procedure was previously limited to single-threaded execution due to concurrent accessing issues during voxel insertion into the hash table, hindering effective use of OpenMP for acceleration. This limitation explains the particularly dramatic acceleration observed after CUDA implementation. Additionally, voxel grid filtering and scan-to-map (S2M) matching operations are accelerated through customized CUDA kernels.

5.4.2 Deployment for Jetson Series

While CUDA acceleration significantly improves processing speed on the host computer, its performance on embedded GPUs such as the Jetson Orin Nano is constrained by system architecture limitations. The parallel processing paradigm necessitates frequent data transfers between host CPU and GPU device, where the restricted memory bandwidth of embedded platforms creates a bottleneck. Additionally, the comparatively weaker embedded CPU impedes overall pipeline efficiency. To quantify these effects, we benchmarked both CUDA-accelerated and CPU-optimized LO implementations on Jetson Orin Nano using the NCLT_1 dataset. As shown in Fig. 5.9, conventional methods (FastLIO and DLO) exhibit significant processing time degradation as map scale increases. In contrast, all Inc-DLOM variants demonstrate more stable performance due to memory-efficient sparse voxel representation, with the CPU-optimized version achieving consistent 10 ms/frame operation. The CUDA-accelerated LO further reduces processing time by nearly 50% compared to the CPU-optimized version. However, this represents a less substantial improvement than observed on desktop systems.

Architectural constraints explain the performance difference. For example, Jetson Orin Nano's 51.2 GB/s memory bandwidth versus 360 GB/s on desktop RTX 3060, and the PCIe 4.0 x8 interface creates transfer bottlenecks, particularly affecting data-intensive operations like point cloud deskew and voxel insertion. Furthermore, the ARM Cortex-A78AE CPU's three times lower single-thread performance versus desktop i5-12400F impedes coordination tasks. Thermal constraints (15W TDP vs. 170W desktop GPU) prevent sustained peak compute. These limitations highlight the need for embedded-specific optimizations targeting zero-copy transfers and kernel fusion.

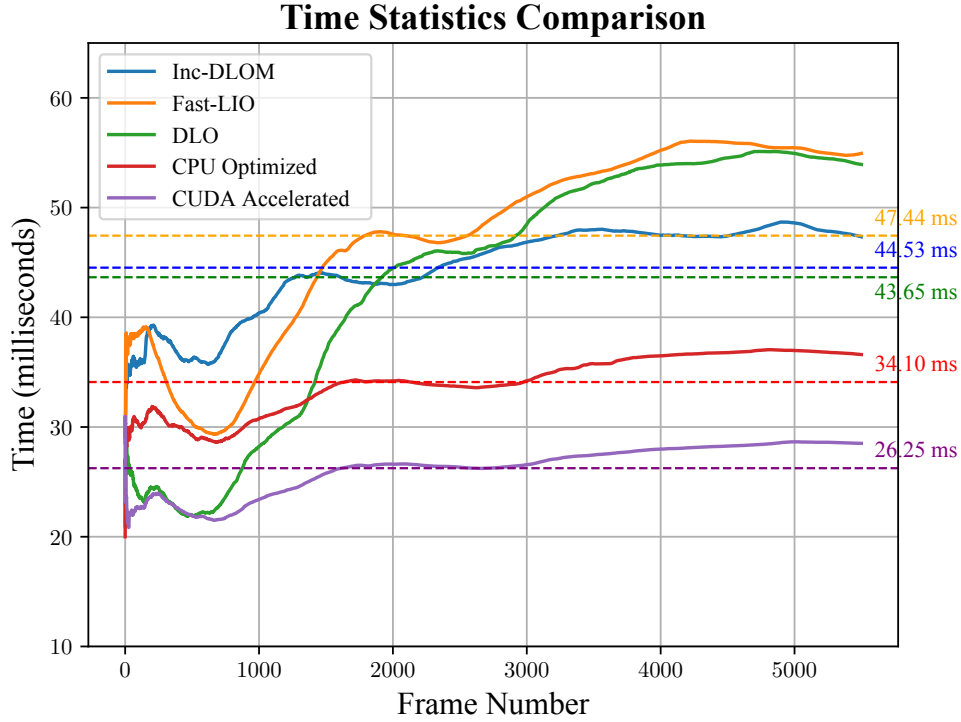


Figure 5.9: Processing Time per Frame on NCLT_1

5.4.3 Mapping Result for Different Datasets

5.4.3.1 PolyU Campus

In conclusion, our efficiency evaluation demonstrates that the CUDA-accelerated implementation successfully enables real-time deployment of our direct LiDAR odometry framework, achieving a consistent 30Hz processing frequency that satisfies the computational requirements for F1Tenth autonomous racing. To quantitatively validate performance under extreme operating conditions, we collected a dedicated dataset comprising 3,000 LiDAR point clouds captured during aggressive maneuvers. Compared to our previous validation platform (Magni robot), the F1Tenth vehicle exhibits significantly more challenging motion dynamics, with peak rotational velocities exceeding 4.8 rad/s and linear accelerations surpassing 2.5 g. These conditions induce severe point cloud distortion and scene deformation, increasing LO error potential for different LO methods.

We chose different LO/LIO methods for comparison with the proposed CUDA-accelerated LO mapping (Fig. 5.10), they are DLO [14], Fast-LIO [85], and KISS-ICP [77] as shown

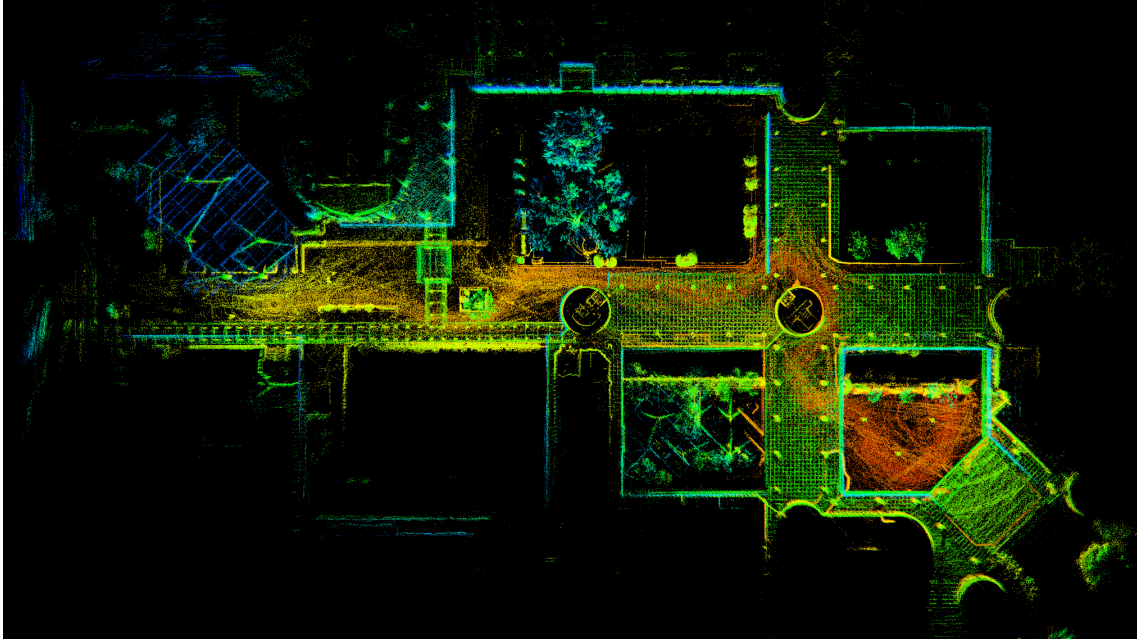


Figure 5.10: CUDA-Accelerated LO Mapping

in Figs. 5.10, 5.11, 5.12, and 5.13 , respectively.

KISS-ICP exhibits significant difficulties in tracking the motion of the F1Tenth vehicle, particularly during initial movement. This failure stems largely from the dynamic campus environment, where pedestrian movements introduce numerous outliers that overwhelm KISS-ICP’s outlier rejection. Furthermore, its reliance on a “point-to-point” optimization objective compounds the issue. The voxel map accumulates all environmental points, including persistent outliers, causing the scan-to-map refinement to fail as incorrect correspondences build up. DLO (Direct LiDAR Odometry) also struggles, with its map becoming severely distorted due to the vehicle’s aggressive motions; it demonstrably fails to track rotations effectively. While Fast-LIO produces a better map overall, it still exhibits distortion, particularly at the trajectory’s end during fast rotations. Additionally, Jetson’s insufficient computing power leads to random frame loss during pose computation.

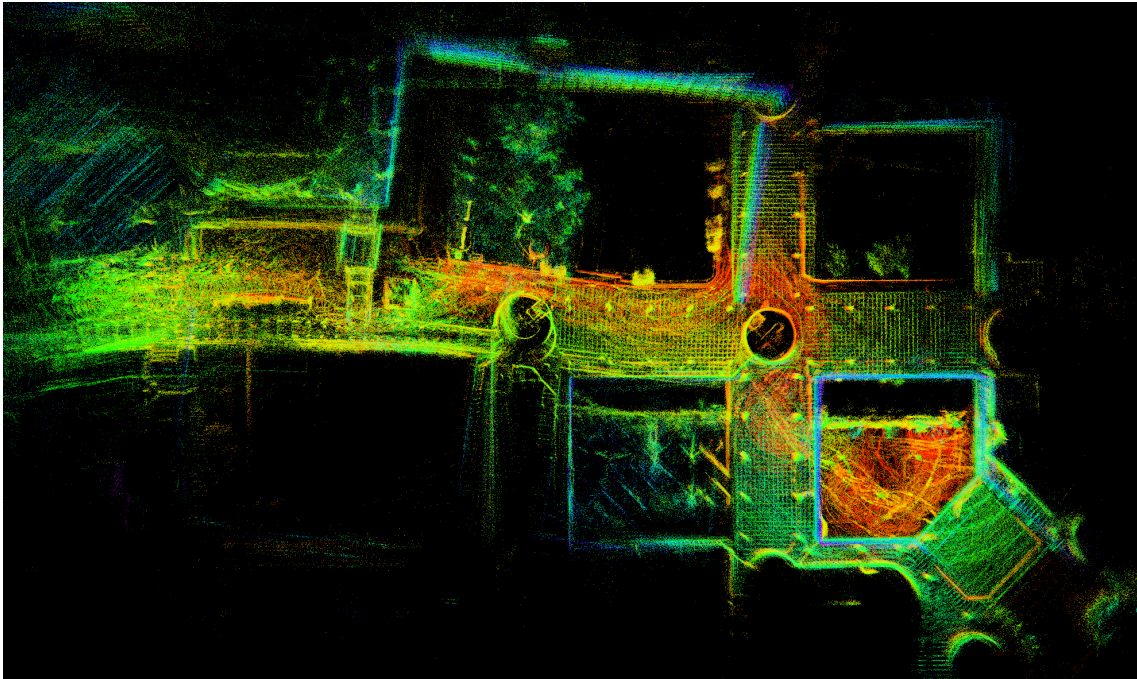


Figure 5.11: DLO Mapping

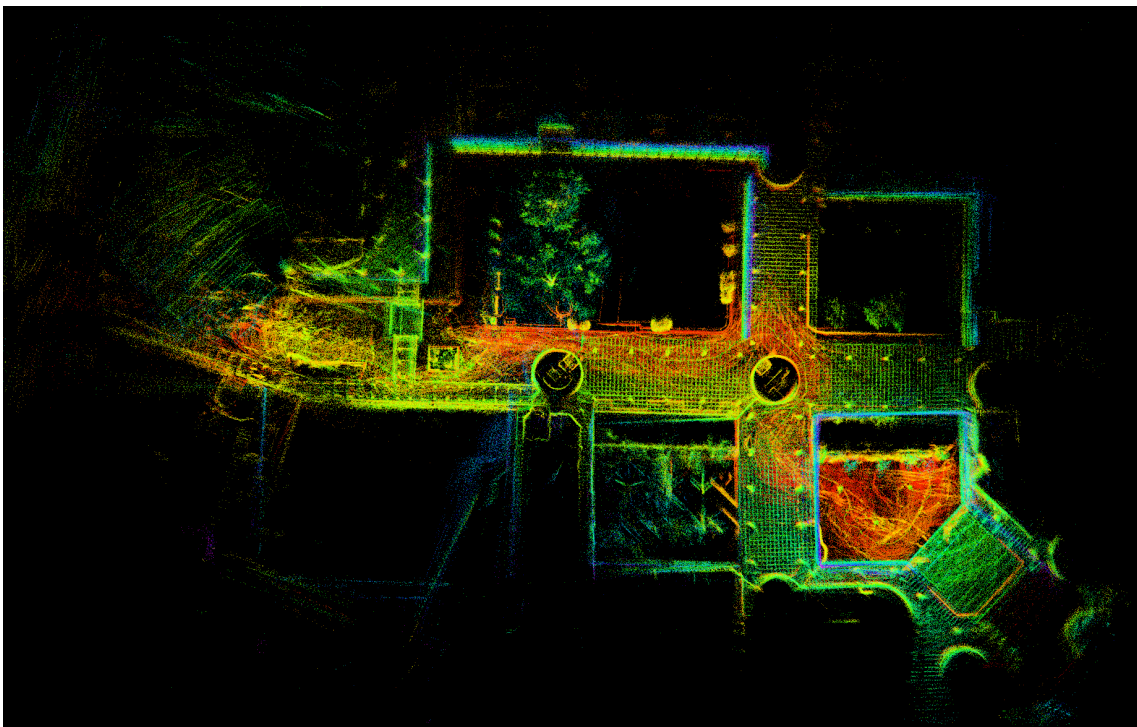


Figure 5.12: Fast-LIO Mapping

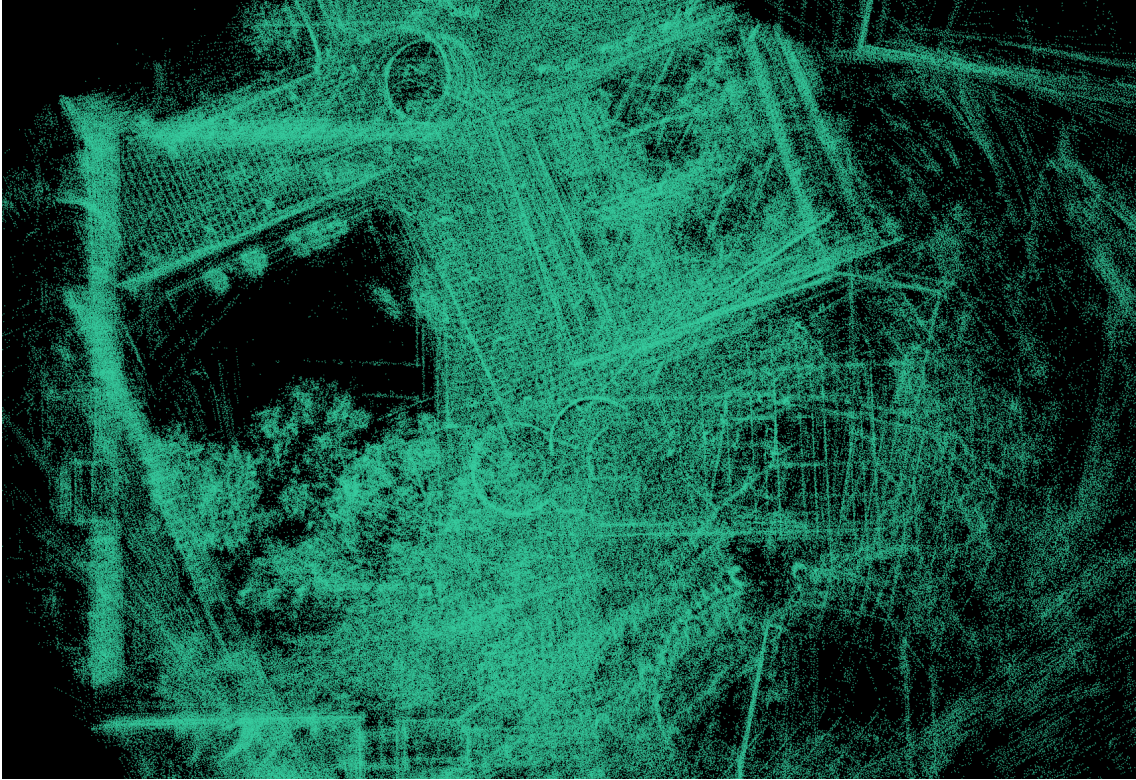


Figure 5.13: KISS-ICP Mapping

5.4.3.2 Textureless Corridor

To evaluate performance in textureless environments, we selected a long corridor within a residential apartment building. The F1Tenth was driven at a reduced speed, yet encountered more challenging degenerate conditions. The resulting map in Fig. 5.14 demonstrates the robust performance of the CUDA-Accelerated LO approach.

5.4.3.3 Open-Source Dataset

We also build the proper mapping by NCLT [10] and Newer College Dataset [64], all tests are performed with Jetson Orin Nano. The mapping results in Figs. 5.15 and 5.16 show that our proposed LO framework does not degenerate by reduced computing power.

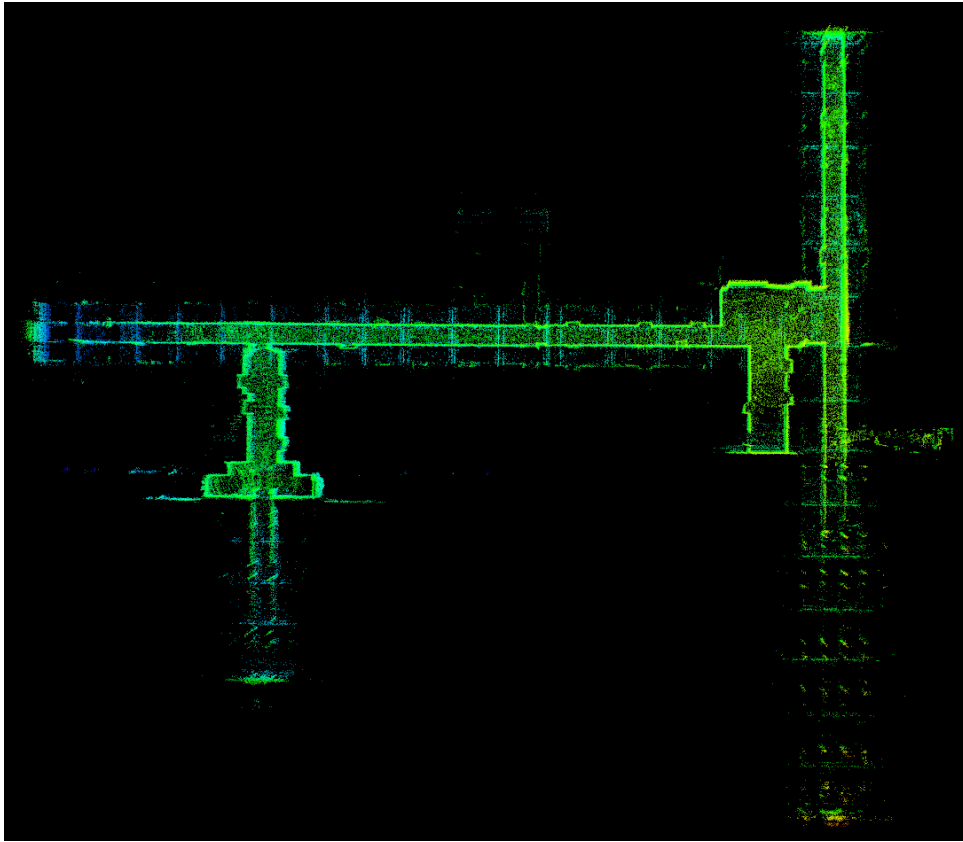


Figure 5.14: Mapping in Long Corridor

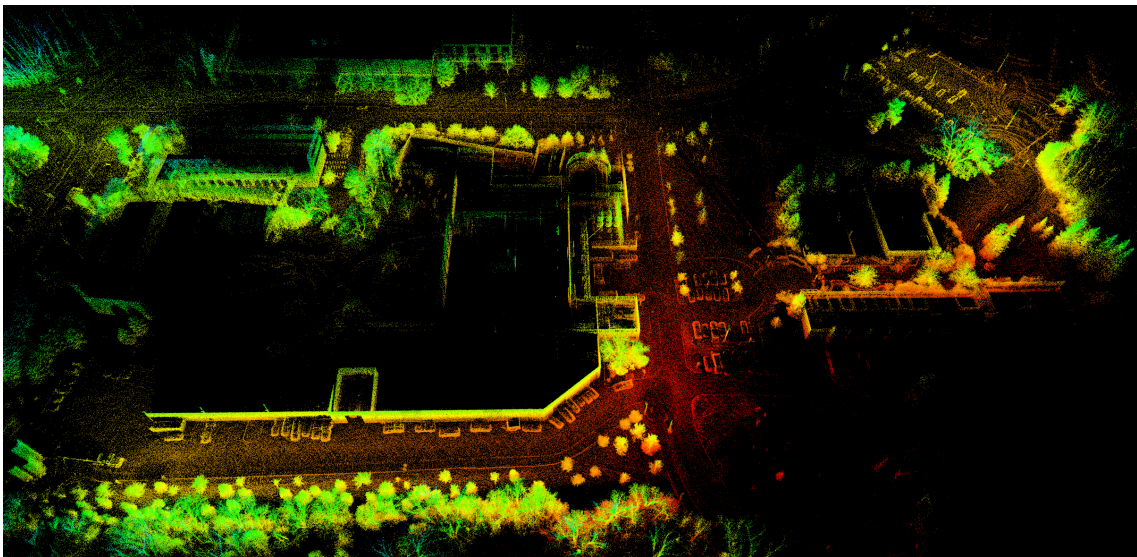


Figure 5.15: High Fidelity Mapping of NCLT

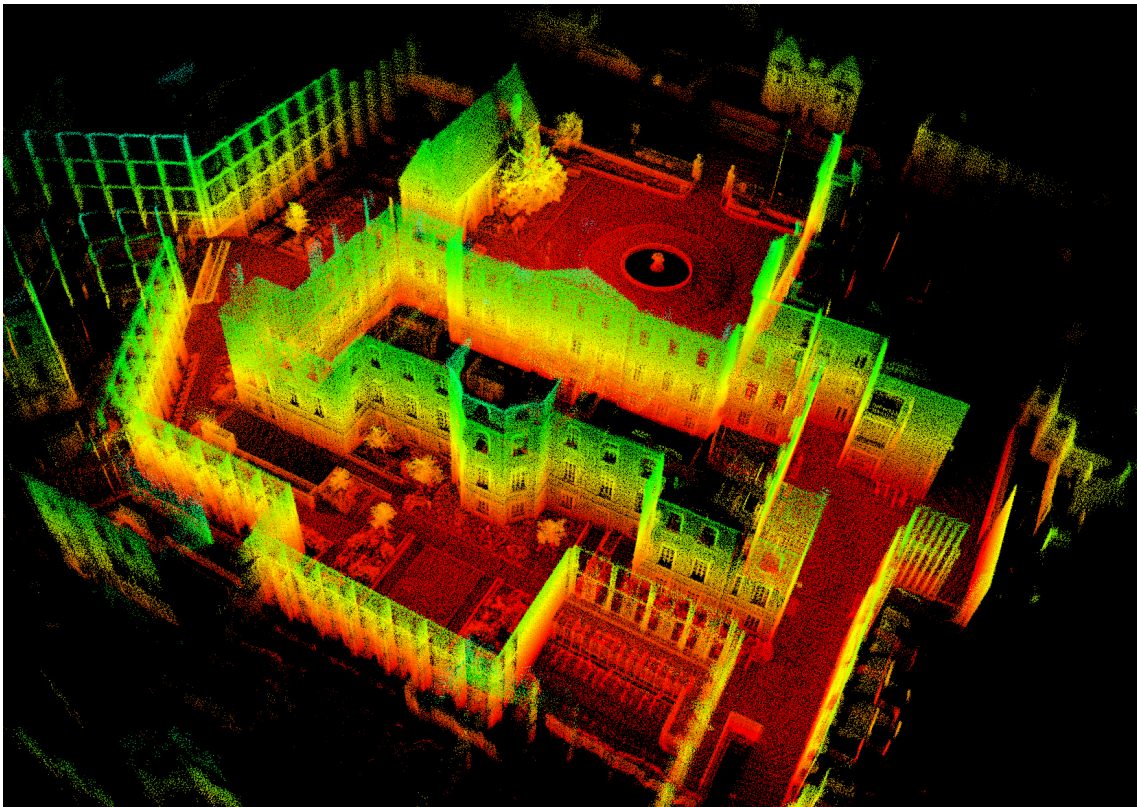


Figure 5.16: High Fidelity Mapping of Newer College Dataset

Chapter 6

Conclusion

6.1 Contributions

In summary, this thesis makes the following contributions:

1. Chapter 3 presents Inc-DLOM, a generalized high-performance LiDAR odometry framework. This approach utilizes direct point cloud registration, specifically the Generalized Iterative Closest Point (GICP) algorithm. As a direct registration method, GICP operates without requiring sensor-specific information such as ring numbers or intensity values, enabling the framework to handle diverse LiDAR sensors without feature extraction tailored to specific hardware types. To enhance processing speed, we introduce a voxel-to-voxel correspondence scheme that reduces the number of point correspondences while preserving covariance information. Additionally, we propose an Incremental Voxel Mapping (IVM) module based on spatial hash tables for efficient map storage and query operations. This design enables $\mathcal{O}(1)$ time complexity for voxel insertion and retrieval. Leveraging IVM's efficiency, our incremental GICP implementation achieves rapid convergence during scan-to-map refinement. The resulting LiDAR odometry solution demonstrates broad applicability across both mechanical and solid-state LiDARs without parameter retuning. Experimental validation on public datasets and custom platforms con-

firm state-of-the-art accuracy in both absolute trajectory error (ATE) and drifting root mean square error (RMSE), while maintaining computational efficiency exceeding 50 Hz for reliable real-time operation in unstructured environments.

2. Chapter 4 introduces Invariant DLIO, our novel LiDAR-Inertial Odometry framework. While Inc-DLOM delivers robust performance, fusion with IMU measurements provides valuable complementary data—particularly beneficial in degenerate environments and during aggressive motions—due to the IMU’s high-frequency dynamics. Existing LIO methods predominantly employ Error-State Kalman Filters (ESKF) or Extended Kalman Filters (EKF), which offer computational efficiency but suffer from linearization-induced biases. To address these limitations, we combine Inc-DLOM’s direct registration approach with Invariant Kalman Filtering (IKF), formulating state estimation on the $SE_2(3)$ Lie group. This design yields trajectory-independent error dynamics where IMU propagation exhibits log-linear properties, thereby avoiding the inconsistency issues inherent in traditional EKF approaches. The scan-to-map refinement module retains the IVM structure for efficient pose optimization. Our framework preserves the advantages of direct point cloud registration while achieving enhanced robustness through IMU integration, extending reliable localization capabilities across diverse robotic operating environments.
3. Chapter 5 details the deployment of our LO/LIO frameworks on embedded systems. While previous evaluations on high-end CPUs demonstrated efficiency across datasets and custom robotic platforms, performance constraints emerge on resource-limited devices like NVIDIA Jetson due to reduced computational capabilities. To overcome this limitation, we develop a heterogeneous CPU-GPU co-processing architecture with custom CUDA kernels that transform key Inc-DLOM operations into GPU-friendly computations. Our optimized implementation achieves **>150 Hz** on desktop GPUs and sustains **>30 Hz** on NVIDIA Jetson Orin Nano platforms—

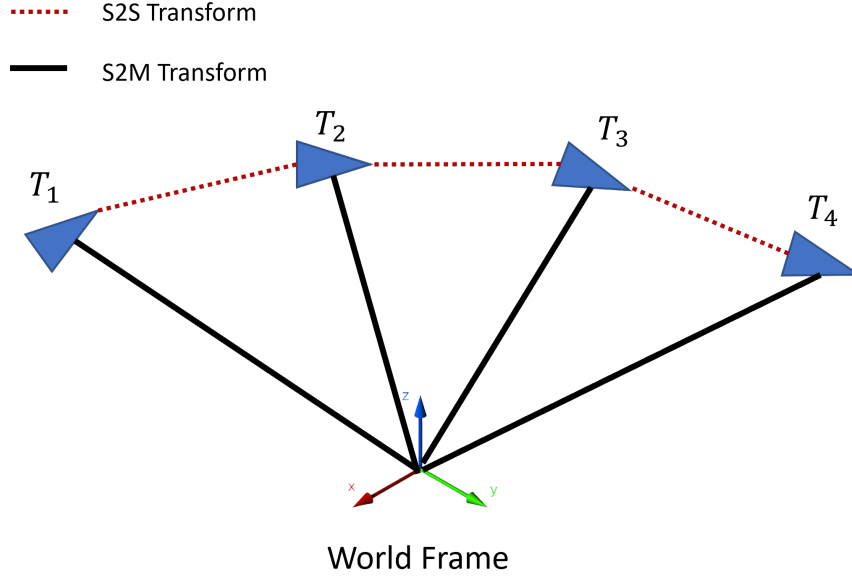


Figure 6.1: An example of pose graph.

doubling the throughput of comparable state-of-the-art methods. This enables real-time processing of high-density LiDAR data (e.g., 128-beam Ouster sensors) under stringent power ($<15\text{W}$) and latency constraints, significantly advancing complex SLAM capabilities for micro-mobile robots and edge AI applications.

6.2 Future Works

6.2.1 Graph Optimization

Currently, our global pose is computed by S2M module, with the initial guess computed from S2S module from consecutive point cloud frames. However, the back-end optimization could be investigated. Fig. 6.1 showed an example of building a pose graph for optimization. T_i represents robot pose nodes for i -th frame, red dashed lines showed scan-to-scan edges representing the relative transformation computed from S2S, and black lines show scan-to-map edges representing the global poses refined from S2M. The smoothing method can be utilized for pose optimization, instead of just using the S2S transforms as initial guesses. More forms of edge constraints could be added, such as IMU pre-

integration, robot dynamics constraints, and GNSS signal, etc., to enable a various type of sensor fusion framework.

During my internship at Autocity, in order to systematically evaluate the performance of the proposed LiDAR-Inertial Odometry (LIO) framework in real-world complex environments, I conducted tests using a data sequence collected by the company underneath the Guomao Bridge in Beijing. This scenario exhibits typical GNSS-denied characteristics: due to obstruction from the bridge structure, no reliable Global Navigation Satellite System signals are available throughout the entire sequence, as illustrated by the trajectory in Fig. 6.2, which remains entirely within a signal blind zone. This condition provides an ideal experimental setting to verify the long-term operational capability of a pure LiDAR-inertial odometry system without external absolute observation aids.

Testing the proposed LIO framework on this dataset yielded the mapping result shown in Fig. 6.3. Overall, the system demonstrates good global consistency and scene reconstruction capability. Notably, even when the robot passed through a narrow corridor beneath the bridge (shown in Fig. 6.4), where the point cloud data exhibited locally degenerate features, the Generalized Iterative Closest Point algorithm employed still maintained robust motion tracking and mapping results. This confirms the adaptability of the framework in environments with limited geometric features.

However, due to the inherent properties of LiDAR odometry, the system still reveals drift in the elevation direction (z -axis) in the absence of GNSS constraints. As shown in the x - z side view in Fig. 6.5, a noticeable cumulative deviation is evident in the vertical direction of the global point cloud map. This phenomenon primarily stems from the sparsity of LiDAR measurements in the vertical direction and the lack of observability in the z -axis for a robot platform dominated by planar motion. During long-duration, large-scale operation, pure LiDAR-inertial fusion struggles to fully suppress error accumulation in elevation.

This experimental result indicates that in transition zones between indoor and outdoor environments or in subterranean settings where GNSS signals are persistently unavailable,

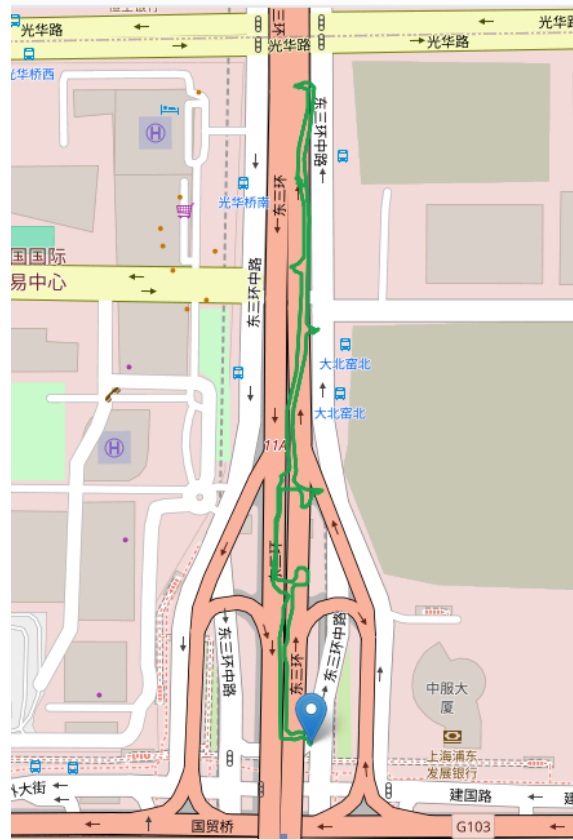


Figure 6.2: Dataset Trajectory

although LIO can maintain a high level of local accuracy and global map consistency, its z-axis drift remains a key limiting factor affecting the absolute accuracy of the global map. This further highlights the necessity of incorporating other auxiliary elevation constraints (such as barometers, terrain matching, or intermittent GNSS) to form a multi-sensor fusion optimization framework in similar application scenarios.

6.2.2 Autoware Integration

Autoware is an open-source software framework for autonomous driving systems, providing comprehensive capabilities in perception, planning, control, and localization. The framework of Autoware is shown in Fig. 6.6.

Localization in Autoware operates primarily through a multi-sensor fusion approach. The core module integrates LiDAR-based scan matching with global positioning system (GPS) and inertial measurement unit (IMU) data. LiDAR point clouds are aligned against

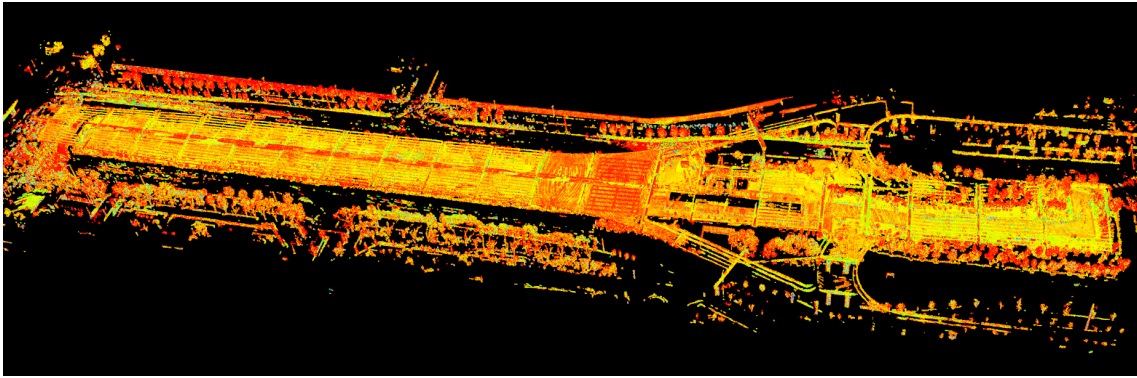


Figure 6.3: Mapping Result of Beijing Data

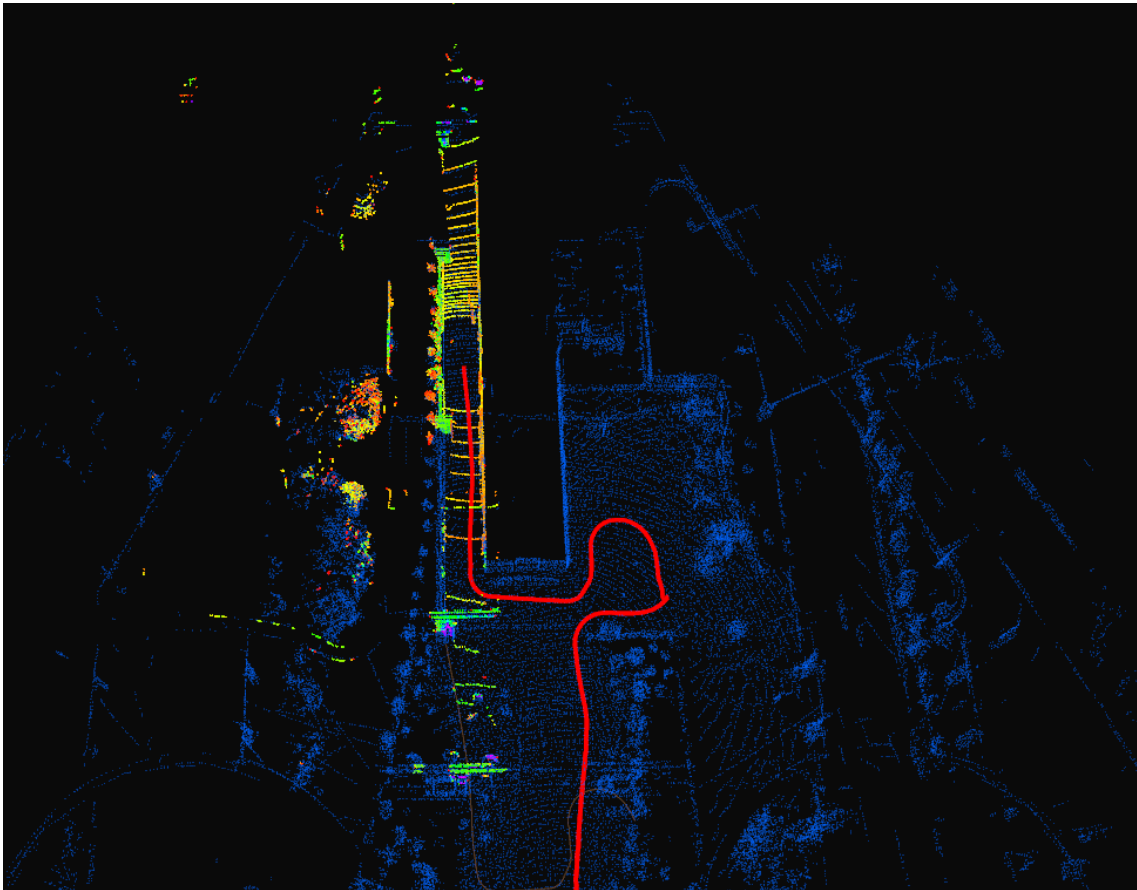


Figure 6.4: Tracking Result Under Narrow Corridor



Figure 6.5: Global Drifting on z-axis

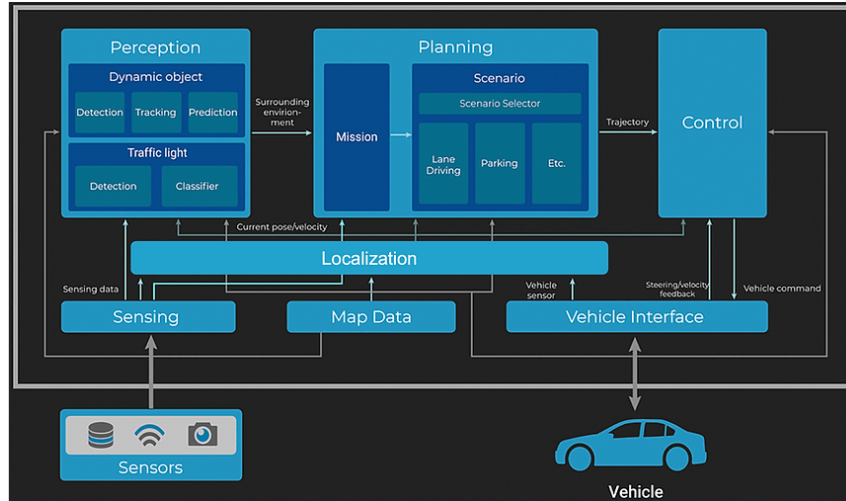


Figure 6.6: Autoware Framework

a pre-built high-definition (HD) map using probabilistic matching algorithms, typically Normal Distributions Transform (NDT). This alignment generates an initial pose estimate, which is further refined through Kalman filtering or particle filtering to fuse temporal IMU data and correct for GPS drift. The output is a robust, real-time 6-degree-of-freedom (6-DoF) pose estimate critical for path planning and vehicle control. However, the localization mode of Autoware requires the input of pre-built point cloud map, which limits the application scenarios of Autoware. Inspired of the limitation, one of the future development of this thesis is to integrate proposed LO/LIO framework into the localization pipeline, which allows switching to multiple driving scenarios.

Our research group has developed a digital twin platform for Connected and Autonomous Vehicles (CAVs), as illustrated in Fig. 6.7. This platform integrates capabilities such as V2X communication, transportation simulation, and human-driving behavior modeling. The underlying urban environment is constructed using open-source street map data from Tsim Sha Tsui, Hong Kong. A key direction for future development involves incorporating a simulated LiDAR sensor into the digital twin. This enhancement will enable the integration of our proposed LiDAR Odometry (LO) / LiDAR-Inertial Odometry (LIO) framework, facilitating the generation of high-fidelity point cloud maps within the simulated environment. Ultimately, this advancement will allow the digital twin to serve as a

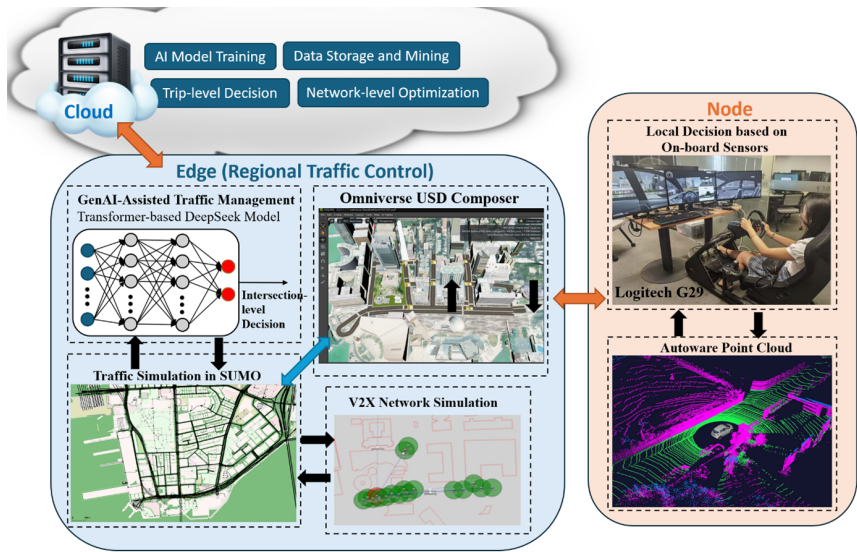


Figure 6.7: Digital Twin Framework

comprehensive testing and validation platform for Autoware-based autonomous driving systems.

Chapter 7

References

- [1] F. Aghili and C.-Y. Su. “Robust Relative Navigation by Integration of ICP and Adaptive Kalman Filter Using Laser Scanner and IMU”. In: *IEEE/ASME Transactions on Mechatronics* 21.4 (Aug. 2016), pp. 2015–2026.
- [2] Chung-Bai et al. “Faster-LIO: Lightweight Tightly Coupled Lidar-Inertial Odometry Using Parallel Sparse Incremental Voxels”. In: *IEEE Robotics and Automation Letters* 7.2 (2022), pp. 4861–4868. DOI: 10.1109/LRA.2022.3152830.
- [3] A. Barrau. “Non-linear state error based extended Kalman filters with applications to navigation”. PhD thesis. Mines ParisTech, 2015.
- [4] Jens Behley and Cyrill Stachniss. “Efficient Surfel-Based SLAM using 3D Laser Range Data in Urban Environments”. In: *Robotics: Science and Systems (RSS)*. 2018. DOI: 10.15607/RSS.2018.XIV.016. URL: <https://www.roboticsproceedings.org/rss14/p16.html>.
- [5] Nathan Bell and Jared Hoberock. “Chapter 26 - Thrust: A Productivity-Oriented Library for CUDA”. In: *GPU Computing Gems Jade Edition*. Ed. by Wen-mei W. Hwu. Applications of GPU Computing Series. Boston: Morgan Kaufmann, 2012, pp. 359–371. ISBN: 978-0-12-385963-1. DOI: <https://doi.org/10.1016/B978-0-12-385963-1.00026-5>. URL: <https://www.sciencedirect.com/science/article/pii/B9780123859631000265>.

- [6] J. L. Bentley. “Multidimensional binary search trees used for associative searching”. In: *Communications of the ACM* 18.9 (Sept. 1975), pp. 509–517.
- [7] Pierre Biasutti et al. “LU-Net: An Efficient Network for 3D LiDAR Point Cloud Semantic Segmentation Based on End-to-End-Learned 3D Features and U-Net”. In: *2019 IEEE/CVF International Conference on Computer Vision Workshop (ICCVW)*. 2019, pp. 942–950. DOI: 10.1109/ICCVW.2019.00123.
- [8] J. L. Blanco and P. K. Rai. *Nanoflann: A C++ header-only fork of FLANN, a library for nearest neighbor (NN) with kd-trees*. <https://github.com/jlblancoc/nanoflann>. 2014.
- [9] S. Bonnabel, P. Martin, and P. Rouchon. “Non-Linear Symmetry-Preserving Observers on Lie Groups”. In: *IEEE Transactions on Automatic Control* 54.7 (July 2009), pp. 1709–1713.
- [10] N. Carlevaris-Bianco, A. Ushani, and R. Eustice. “University of Michigan North Campus long-term vision and lidar dataset”. In: *International Journal of Robotics Research (IJRR)* 35.9 (2016), pp. 1023–1035.
- [11] Jiachong Chang et al. “An Anti-Spoofing Model Based on MVM and MCCM for a Loosely-Coupled GNSS/INS /LiDAR Kalman Filter”. In: *IEEE Transactions on Intelligent Vehicles* 9.1 (2024), pp. 1744–1755. DOI: 10.1109/TIV.2023.3335356.
- [12] K. Chang et al. “SOA-Integrated Widely Tunable Laser Array for All-Solid Lidar Application”. In: *IEEE Photonics Technology Letters* 36.23 (Dec. 2024), pp. 1381–1384. DOI: 10.1109/LPT.2024.3478811.
- [13] Jiahui Chen et al. “VPE-SLAM: Virtual Point Enhanced SLAM Using Solid-State LiDAR for Weak Feature Environments”. In: *IEEE Sensors Journal* 24.10 (2024), pp. 16397–16407. DOI: 10.1109/JSEN.2024.3382350.
- [14] K. Chen et al. “Direct LiDAR Odometry: Fast Localization With Dense Point Clouds”. In: *IEEE Robotics and Automation Letters* 7.2 (Apr. 2022), pp. 2000–2007.

- [15] Shoubin Chen et al. “NDT-LOAM: A Real-Time Lidar Odometry and Mapping With Weighted NDT and LFA”. In: *IEEE Sensors Journal* 22.4 (2022), pp. 3660–3671. DOI: 10.1109/JSEN.2021.3135055.
- [16] Songming Chen and Vincent Frémont. “A Loosely Coupled Vision-LiDAR Odometry using Covariance Intersection Filtering”. In: *2021 IEEE Intelligent Vehicles Symposium (IV)*. 2021, pp. 1102–1107. DOI: 10.1109/IV48863.2021.9575275.
- [17] Y. Cheng and X. Tong. “Application of Octree Map in Robotic Environment Reconstruction”. In: *2018 IEEE 8th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems (CYBER)*. Tianjin, China, 2018, pp. 843–848.
- [18] L. Dagum and R. Menon. “OpenMP: an industry standard API for shared-memory programming”. In: *IEEE Computational Science and Engineering* 5.1 (1998), pp. 46–55. DOI: 10.1109/99.660313.
- [19] Pierre Dellenbach et al. “CT-ICP: Real-time Elastic LiDAR Odometry with Loop Closure”. In: *2022 International Conference on Robotics and Automation (ICRA)*. 2022, pp. 5580–5586. DOI: 10.1109/ICRA46639.2022.9811849.
- [20] J. Deschaud. “IMLS-SLAM: Scan-to-Model Matching Based on 3D Data”. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 2480–2485.
- [21] Y. Dong et al. “PVE-LIOM: Pseudo-Visual Enhanced LiDAR-Inertial Odometry and Mapping”. In: *IEEE Transactions on Instrumentation and Measurement* 72 (2023), pp. 1–13.
- [22] Yu-Cheng Fan et al. “Environmental Detection for Autonomous Vehicles Based on Multi-View Image and 3D LiDAR Point Cloud Map”. In: *IEEE Access* 11 (2023), pp. 70408–70424. DOI: 10.1109/ACCESS.2023.3292118.

- [23] K. Fang and I. W.-H. Ho. “Real-Time GICP: Direct LiDAR SLAM for CPU Environment”. In: *TENCON 2022 - 2022 IEEE Region 10 Conference (TENCON)*. Hong Kong, Hong Kong, 2022, pp. 1–6. DOI: 10.1109/TENCON55691.2022.9977999.
- [24] K. Fang and R. Song. “GMLO: Ground and Memory Optimized LiDAR Odometry”. In: *Proceedings of the 9th EAI International Conference on Smart Grid and Innovative Frontiers in Telecommunications (EAI SmartGIFT 2024)*. Hong Kong SAR, China, 2024, pp. 1–9.
- [25] K. Fang, R. Song, and I. W.-H. Ho. “Inc-DLOM: Incremental Direct LiDAR Odometry and Mapping”. In: *IEEE Access* 13 (2025), pp. 6527–6538. DOI: 10.1109/ACCESS.2025.3526626.
- [26] K. Fang, R. Song, and I. W.-H. Ho. “Invariant-DLIO: Direct LiDAR–Inertial Odometry Based on Invariant Kalman Filtering”. In: *IEEE Sensors Journal* 25.11 (June 2025), pp. 20572–20583. DOI: 10.1109/JSEN.2025.3558916.
- [27] C. Forster et al. “On-Manifold Preintegration for Real-Time Visual–Inertial Odometry”. In: *IEEE Transactions on Robotics* 33.1 (Feb. 2017), pp. 1–21. DOI: 10.1109/TR0.2016.2597321.
- [28] Manikandan Ganesan et al. “A Comprehensive Review on Deep Learning-Based Motion Planning and End-to-End Learning for Self-Driving Vehicle”. In: *IEEE Access* 12 (2024), pp. 66031–66067. DOI: 10.1109/ACCESS.2024.3394869.
- [29] A. Geiger, P. Lenz, and R. Urtasun. “Are we ready for autonomous driving? The KITTI vision benchmark suite”. In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2012, pp. 3354–3361.
- [30] M. Grupp. *evo: Python package for the evaluation of odometry and SLAM*. <https://github.com/MichaelGrupp/evo>. 2017.
- [31] Hongliang Guo, Jiankang Zhu, and Yunping Chen. “E-LOAM: LiDAR Odometry and Mapping With Expanded Local Structural Information”. In: *IEEE Transactions*

- on Intelligent Vehicles* 8.2 (2023), pp. 1911–1921. DOI: 10 . 1109 /TIV . 2022 . 3151665.
- [32] M. Haris and H. Nam. “Path Planning Optimization of Smart Vehicle With Fast Converging Distance-Dependent PSO Algorithm”. In: *IEEE Open Journal of Intelligent Transportation Systems* 5 (2024), pp. 726–739. DOI: 10 . 1109 /OJITS . 2024 . 3486155.
- [33] R. Hartley et al. “Contact-aided invariant extended Kalman filtering for robot state estimation”. In: *The International Journal of Robotics Research* 39.4 (Jan. 2020), pp. 402–430.
- [34] Robin Heinzler et al. “CNN-Based Lidar Point Cloud De-Noising in Adverse Weather”. In: *IEEE Robotics and Automation Letters* 5.2 (2020), pp. 2514–2521. DOI: 10 . 1109 /LRA . 2020 . 2972865.
- [35] Wolfgang Hess et al. “Real-time loop closure in 2D LIDAR SLAM”. In: *2016 IEEE International Conference on Robotics and Automation (ICRA)*. 2016, pp. 1271–1278. DOI: 10 . 1109 /ICRA . 2016 . 7487258.
- [36] Kohei Honda et al. “Generalized LOAM: LiDAR Odometry Estimation With Trainable Local Geometric Features”. In: *IEEE Robotics and Automation Letters* 7.4 (2022), pp. 12459–12466. DOI: 10 . 1109 /LRA . 2022 . 3219022.
- [37] T. Hua et al. “PLV-IEKF: Consistent Visual-Inertial Odometry using Points, Lines, and Vanishing Points”. In: *2023 IEEE International Conference on Robotics and Biomimetics (ROBIO)*. Koh Samui, Thailand, 2023, pp. 1–7. DOI: 10 . 1109 /ROBIO58561 . 2023 . 10354852.
- [38] G. P. Huang, A. I. Mourikis, and S. I. Roumeliotis. “Observability-based rules for designing consistent EKF SLAM estimators”. In: *The International Journal of Robotics Research* 29.5 (2010), pp. 502–528.

- [39] Leyao Huang. “Review on LiDAR-based SLAM Techniques”. In: *2021 International Conference on Signal Processing and Machine Learning (CONF-SPML)*. 2021, pp. 163–168. DOI: 10.1109/CONF-SPML54095.2021.00040.
- [40] J. H. Hwang, J. Cha, and C. G. Park. “A Novel Federated Structure of Invariant EKF Using Left-/Right-Invariant Observations”. In: *IEEE Sensors Journal* 22.21 (Nov. 2022). 1 November 2022, pp. 20645–20654.
- [41] *Jetson Orin Nano Super*. NVIDIA, 2025. URL: <https://www.nvidia.cn/autonomous-machines/embedded-systems/jetson-orin/nano-super-developer-kit/> (visited on 05/14/2025).
- [42] G. Kim et al. “MulRan: Multimodal Range Dataset for Urban Place Recognition”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. Paris, France, 2020, pp. 6246–6253.
- [43] K. Koide, J. Miura, and E. Menegatti. “A portable three-dimensional LIDAR-based system for long-term and wide-area people behavior measurement”. In: *International Journal of Advanced Robotic Systems* (2019).
- [44] K. Koide et al. “Voxelized GICP for Fast and Accurate 3D Point Cloud Registration”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. Xi’an, China, 2021, pp. 11054–11059.
- [45] D. Lee, H. Lim, and S. Han. “GenZ-ICP: Generalizable and Degeneracy-Robust LiDAR Odometry Using an Adaptive Weighting”. In: *IEEE Robotics and Automation Letters* (2024). Early Access. DOI: 10.1109/LRA.2024.3498779.
- [46] Kailai Li, Meng Li, and Uwe D. Hanebeck. “Towards High-Performance Solid-State-LiDAR-Inertial Odometry and Mapping”. In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 5167–5174. DOI: 10.1109/LRA.2021.3070251.
- [47] Tao Li et al. “P3-LOAM: PPP/LiDAR Loosely Coupled SLAM With Accurate Covariance Estimation and Robust RAIM in Urban Canyon Environment”. In: *IEEE Sensors Journal* 21.5 (2021), pp. 6660–6671. DOI: 10.1109/JSEN.2020.3042968.

- [48] You Li et al. “Inertial Sensing Meets Machine Learning: Opportunity or Challenge?” In: *IEEE Transactions on Intelligent Transportation Systems* 23.8 (2022), pp. 9995–10011. DOI: 10.1109/TITS.2021.3097385.
- [49] You Li et al. “Toward Location-Enabled IoT (LE-IoT): IoT Positioning Techniques, Error Sources, and Error Mitigation”. In: *IEEE Internet of Things Journal* 8.6 (2021), pp. 4035–4062. DOI: 10.1109/JIOT.2020.3019199.
- [50] Jiarong Lin and Fu Zhang. “Loam livox: A fast, robust, high-precision LiDAR odometry and mapping package for LiDARs of small FoV”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. 2020, pp. 3126–3131. DOI: 10.1109/ICRA40945.2020.9197440.
- [51] Guangjie Liu et al. “Innovations and Refinements in LiDAR Odometry and Mapping: A Comprehensive Review”. In: *IEEE/CAA Journal of Automatica Sinica* 12.6 (2025), pp. 1072–1094. DOI: 10.1109/JAS.2025.125198.
- [52] Guangjie Liu et al. “Innovations and Refinements in LiDAR Odometry and Mapping: A Comprehensive Review”. In: *IEEE/CAA Journal of Automatica Sinica* 12.6 (2025), pp. 1072–1094. DOI: 10.1109/JAS.2025.125198.
- [53] Shangjia Liu et al. “GBA-LIOM: Global Bundle Adjustment Enhanced LiDAR–Inertial Odometry and Mapping”. In: *IEEE Transactions on Instrumentation and Measurement* 74 (2025), pp. 1–13. DOI: 10.1109/TIM.2025.3571159.
- [54] Y. Luo, J. Hu, and C. Guo. “Right Invariant SE2 (3) - EKF for Relative Navigation in Learning-based Visual Inertial Odometry”. In: *2022 5th International Symposium on Autonomous Systems (ISAS)*. Hangzhou, China, 2022.
- [55] W. Maddern et al. “1 Year, 1000km: The Oxford RobotCar Dataset”. In: *The International Journal of Robotics Research (IJRR)* (2016).
- [56] Olga V. Moldovanova and Mikhail G. Kurnosov. “Auto-Vectorization of Loops on Intel 64 and Intel Xeon Phi: Analysis and Evaluation”. In: *Parallel Computing*

- Technologies*. Ed. by Victor Malyshkin. Cham: Springer International Publishing, 2017, pp. 143–150. ISBN: 978-3-319-62932-2.
- [57] *MTN-FP650-Maxtang: Premier Supplier of A*. <https://www.maxtangpc.com/show-106-194-1.html>. (accessed Jul. 23, 2024).
- [58] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. “ORB-SLAM: A Versatile and Accurate Monocular SLAM System”. In: *IEEE Transactions on Robotics* 31.5 (2015), pp. 1147–1163. DOI: 10.1109/TR0.2015.2463671.
- [59] Seyed-Mahdi Nasiri, Reshad Hosseini, and Hadi Moradi. “Novel Parameterization for Gauss–Newton Methods in 3-D Pose Graph Optimization”. In: *IEEE Transactions on Robotics* 37.3 (2021), pp. 780–797. DOI: 10.1109/TR0.2020.3034021.
- [60] Yue Pan et al. “MULLS: Versatile LiDAR SLAM via Multi-metric Linear Least Square”. In: *2021 IEEE International Conference on Robotics and Automation (ICRA)*. 2021, pp. 11633–11640. DOI: 10.1109/ICRA48506.2021.9561364.
- [61] *Products Magni*. <https://www.ubiquityrobotics.com/products-magni/>.
- [62] C. Qin et al. “LINS: A Lidar-Inertial State Estimator for Robust and Efficient Navigation”. In: *2020 IEEE International Conference on Robotics and Automation (ICRA)*. Paris, France, 2020, pp. 8899–8906.
- [63] Tong Qin, Peiliang Li, and Shaojie Shen. “VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator”. In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 1004–1020. DOI: 10.1109/TR0.2018.2853729.
- [64] M. Ramezani et al. “The Newer College Dataset: Handheld LiDAR, Inertial and Vision with Ground Truth”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, NV, USA, 2020, pp. 4353–4360.
- [65] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 779–788. DOI: 10.1109/CVPR.2016.91.

- [66] Arch D. Robison. “Intel® Threading Building Blocks (TBB)”. In: *Encyclopedia of Parallel Computing*. Ed. by David Padua. Boston, MA: Springer US, 2011, pp. 955–964. ISBN: 978-0-387-09766-4. DOI: 10.1007/978-0-387-09766-4_51. URL: https://doi.org/10.1007/978-0-387-09766-4_51.
- [67] R. B. Rusu and S. Cousins. “3D is here: Point Cloud Library (PCL)”. In: *2011 IEEE International Conference on Robotics and Automation*. Shanghai, China, 2011, pp. 1–4. DOI: 10.1109/ICRA.2011.5980567.
- [68] A. Segal, D. Haehnel, and S. Thrun. “Generalized-ICP”. In: *Robotics: Science and Systems*. Robotics: Science and Systems Foundation, June 2009.
- [69] J. Serafin and G. Grisetti. “NICP: Dense normal based point cloud registration”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2015, pp. 742–749.
- [70] T. Shan and B. Englot. “LeGO-LOAM: Lightweight and Ground-Optimized Lidar Odometry and Mapping on Variable Terrain”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2018, pp. 4758–4765.
- [71] T. Shan et al. “LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping”. In: *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Las Vegas, NV, USA, 2020, pp. 5135–5142.
- [72] P. Shi et al. “Invariant Extended Kalman Filtering for Tightly Coupled LiDAR-Inertial Odometry and Mapping”. In: *IEEE/ASME Transactions on Mechatronics* 28.4 (Aug. 2023), pp. 2213–2224.
- [73] Akpojoto Siemuri et al. “A Systematic Review of Machine Learning Techniques for GNSS Use Cases”. In: *IEEE Transactions on Aerospace and Electronic Systems* 58.6 (2022), pp. 5043–5077. DOI: 10.1109/TAES.2022.3219366.
- [74] P. Stotko. “stdgpu: Efficient STL-like Data Structures on the GPU”. arXiv:1908.05936. Aug. 2019. URL: <https://arxiv.org/abs/1908.05936>.

- [75] F1TENTH Development Team. *F1TENTH Autonomous Racing Documentation*. <https://f1tenth.readthedocs.io/en/stable/>. Accessed: 2025. 07. 17. 2023. URL: <https://f1tenth.readthedocs.io/en/stable/>.
- [76] S. Thrun et al. “Stanley: The robot that won the DARPA grand challenge”. In: *Journal of Field Robotics* 23.9 (2006), pp. 661–692.
- [77] I. Vizzo et al. “KISS-ICP: In Defense of Point-to-Point ICP – Simple, Accurate, and Robust Registration If Done the Right Way”. In: *IEEE Robotics and Automation Letters* 8.2 (Feb. 2023), pp. 1029–1036.
- [78] Chen Wang, Yanfeng Gu, and Xian Li. “LPRnet: A Self-Supervised Registration Network for LiDAR and Photogrammetric Point Clouds”. In: *IEEE Transactions on Geoscience and Remote Sensing* 63 (2025), pp. 1–12. DOI: 10.1109/TGRS.2025.3541639.
- [79] H. Wang, C. Wang, and L. Xie. “Lightweight 3-D Localization and Mapping for Solid-State LiDAR”. In: *IEEE Robotics and Automation Letters* 6.2 (Apr. 2021), pp. 1801–1807.
- [80] H. Wang et al. “F-LOAM: Fast LiDAR Odometry and Mapping”. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2021, pp. 4390–4396.
- [81] J. Wang et al. “Feature- and Distribution-Based LiDAR SLAM With Generalized Feature Representation and Heuristic Nonlinear Optimization”. In: *IEEE Transactions on Instrumentation and Measurement* 72 (2023), pp. 1–15.
- [82] Yuehai Wang et al. “Point Cloud Completion Cascade Optimization Network Based on Feature Fusion”. In: *2023 IEEE 6th International Conference on Computer and Communication Engineering Technology (CCET)*. 2023, pp. 1–5. DOI: 10.1109/CCET59170.2023.10335158.

- [83] R. W. Wolcott and R. M. Eustice. “Visual localization within LiDAR maps for automated urban driving”. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 176–183.
- [84] Zongzhou Wu et al. “A Failure-Resistant, Lightweight, and Tightly Coupled GNSS/INS/Vision Vehicle Integration for Complex Urban Environments”. In: *IEEE Transactions on Instrumentation and Measurement* 73 (2024), pp. 1–13. DOI: 10.1109/TIM.2024.3406814.
- [85] W. Xu and F. Zhang. “FAST-LIO: A Fast, Robust LiDAR-Inertial Odometry Package by Tightly-Coupled Iterated Kalman Filter”. In: *IEEE Robotics and Automation Letters* 6.2 (Apr. 2021), pp. 3317–3324.
- [86] Zhu Xuefen et al. “Carrier frequency disturbance distributions on GPS during equatorial ionospheric scintillation”. In: *Journal of Systems Engineering and Electronics* 31.6 (2020), pp. 1308–1317. DOI: 10.23919/JSEE.2020.000101.
- [87] H. Ye, Y. Chen, and M. Liu. “Tightly Coupled 3D Lidar Inertial Odometry and Mapping”. In: *2019 International Conference on Robotics and Automation (ICRA)*. Montreal, QC, Canada, 2019, pp. 3144–3150.
- [88] Wenlu Yu et al. “I2EKF-LO: A Dual-Iteration Extended Kalman Filter Based LiDAR Odometry”. In: *2024 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2024, pp. 10453–10460. DOI: 10.1109/IROS58592.2024.10801455.
- [89] J. Zhang and S. Singh. “Low-drift and real-time lidar odometry and mapping”. In: *Autonomous Robots* 41.2 (2017), pp. 401–416.
- [90] Liang Zhang, Tao Zhang, and Hongyu Wei. “A Single Source-Aided Inertial Integrated Navigation Scheme for Passive Navigation of Autonomous Underwater Vehicles”. In: *IEEE Sensors Journal* 24.7 (2024), pp. 11237–11245. DOI: 10.1109/JSEN.2024.3367931.

- [91] Linshuai Zhang et al. “Research on Feature Extraction Method Based on Point Cloud Roughness”. In: *2022 IEEE International Conference on Mechatronics and Automation (ICMA)*. 2022, pp. 1568–1573. DOI: 10.1109/ICMA54519.2022.9856165.
- [92] T. Zhang et al. “Convergence and Consistency Analysis for a 3-D Invariant-EKF SLAM”. In: *IEEE Robotics and Automation Letters* 2.2 (Apr. 2017), pp. 733–740.
- [93] Xiao Zhang and Xinming Huang. “Real-Time Fast Channel Clustering for LiDAR Point Cloud”. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 69.10 (2022), pp. 4103–4107. DOI: 10.1109/TCSII.2022.3185228.
- [94] S. Zhao et al. “A Robust Laser-Inertial Odometry and Mapping Method for Large-Scale Highway Environments”. In: *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Macau, China, 2019, pp. 1285–1292.
- [95] B. Zhou et al. “ATI-CTLO: Adaptive Temporal Interval-Based Continuous-Time LiDAR-Only Odometry”. In: *IEEE Robotics and Automation Letters* 9.12 (Dec. 2024), pp. 11162–11169. DOI: 10.1109/LRA.2024.3486233.
- [96] P. Zhou et al. “T-LOAM: Truncated Least Squares LiDAR-Only Odometry and Mapping in Real Time”. In: *IEEE Transactions on Geoscience and Remote Sensing* 60 (2022), pp. 1–13.
- [97] Jun Zhu, Hongyi Li, and Tao Zhang. “Camera, LiDAR, and IMU Based Multi-Sensor Fusion SLAM: A Survey”. In: *Tsinghua Science and Technology* 29.2 (2024), pp. 415–429. DOI: 10.26599/TST.2023.9010010.