Tightly Coupled Range Inertial Odometry and Mapping with Exact Point Cloud Downsampling

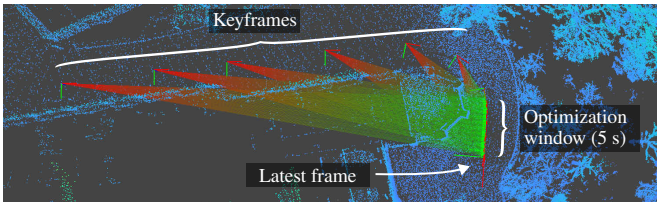
Kenji Koide[1], Aoki Takanose[1], Shuji Oishi[1], and Masashi Yokozuka[1]

***Abstract*— In this work, to facilitate the real-time processing of multi-scan registration error minimization on factor graphs, we devise a point cloud downsampling algorithm based on coreset extraction. This algorithm extracts a subset of the residuals of input points such that the subset yields exactly the same quadratic error function as that of the original set for a given pose. This enables a significant reduction in the number of residuals to be evaluated without approximation errors at the sampling point. Using this algorithm, we devise a complete SLAM framework that consists of odometry estimation based on sliding window optimization and global trajectory optimization based on registration error minimization over the entire map, both of which can run in real time on a standard CPU. The experimental results demonstrate that the proposed framework outperforms state-of-the-art CPU-based SLAM frameworks without the use of GPU acceleration.**

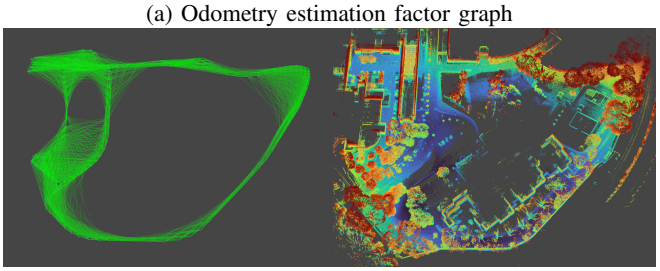
I. Introduction

Point cloud SLAM algorithms that directly compute and minimize point cloud registration errors on factor graphs have been gaining attention due to their precision and robustness. Methods such as odometry estimation with sliding window optimization [1], global trajectory optimization via global registration error minimization [2], and LiDAR-bundle adjustment [3] excel in optimizing sensor poses to maximize the consistency between multiple point clouds. They offer more accurate and reliable estimations compared to those obtained using traditional approaches such as filtering-based odometry estimation [4] and global trajectory optimization using relative pose constraints [5]. However, those methods are computationally intensive, as they simultaneously optimize multiple sensor poses with registration error factors involving a vast number of points. This makes real-time processing challenging without extensive approximations or hardware accelerators such as GPUs [1], [2], [6].

In our previous work, we introduced GLIM, a SLAM framework that utilizes GPU-accelerated registration error factors [7]. Its odometry estimation and global trajectory optimization rely on factor graphs that directly minimize point cloud registration errors across multiple frames, eliminating the need for conventional relative pose factors (i.e., *gtsam::BetweenFactor<Pose3>*). These registration-error-minimization-based algorithms demonstrated significant advantages, including extreme robustness against point cloud degeneration and interruptions as well as the ability to close loops even with minimal frame overlap. However, the reliance on GPU acceleration limits their broad applicability.



(a) Odometry estimation factor graph



(b) Global trajectory optimization factor graph

Fig. 1: Factor graphs for odometry estimation and global trajectory optimization. The proposed framework extensively uses registration error factors to directly minimize registration errors across multiple point clouds. The exact downsampling algorithm drastically reduces the linearization cost of registration error factors and enables the real-time processing of these dense factor graphs on standard CPUs.

In the present work, we aim to enable the real-time optimization of registration error minimization factor graphs using only a standard CPU[1]. To this end, we leverage a point cloud downsampling algorithm based on efficient coreset extraction [8]. This method extracts a subset of the residuals of input points, ensuring that the resulting subset preserves the exact quadratic error function of the original set for a given pose. This approach significantly reduces the number of residuals to be evaluated without introducing approximation errors at the sampling point. We also introduce a strategy to mitigate the computational cost of the downsampling by deferring the coreset extraction until it is required. By integrating this sampling algorithm, we develop a complete SLAM framework with registration error minimization factor graphs that can operate in real time on standard CPUs.

The contributions of this work are as follows:

- We extend the exact point cloud downsampling algorithm [8] by incorporating point correspondence up-

*This work was supported in part by JSPS KAKENHI Grant Number 23K16979 and projects, JPNP14004 and JPNP21004, subsidized by the New Energy and Industrial Technology Development Organization (NEDO).

[1]All the authors are with the Department of Information Technology and Human Factors, the National Institute of Advanced Industrial Science and Technology, Tsukuba, Ibaraki, Japan, k.koide@aist.go.jp

[1]See the project page for supplementary videos: https://staff.aist.go.jp/k.koide/projects/icra2025_es/.

dating to better capture the nonlinearity of the registration error function. We also introduce a factor linearization strategy that defers the execution of exact downsampling to reduce the computational overhead of the linearization process.

- Leveraging exact point cloud downsampling, we develop a complete SLAM framework that extensively uses registration error factors for both odometry estimation and global trajectory optimization, as shown in Fig. 1. Although the graph structures were originally designed for GPU processing [7], the exact downsampling algorithm enables them to operate in real time on a standard CPU.

II. Related Work

A. Odometry Estimation

Many existing LiDAR odometry algorithms rely on state filtering, which optimizes only the current sensor state while marginalizing past states as new observations arrive [4], [9]. The process is typically combined with scan-to-model matching, where past point clouds are accumulated into a single model point cloud (or local map), and the current sensor scan is aligned with this model [10]. Although this causal estimation approach is efficient and accurate in feature-rich environments, it struggles to propagate the uncertainty of past observations and sensor states, resulting in difficulties in scenarios with point cloud degeneration and interruptions.

To enhance robustness, LiDAR odometry algorithms based on sliding window optimization have been introduced [1], [7], [11], [12]. Unlike filtering-based approaches, these algorithms optimize both current and past sensor states within a sliding window and allow the correction of estimation drift by propagating uncertainties backward in time. This results in robustness to point cloud degeneration and rapid motion. However, the need to continuously optimize multiple sensor states requires significant computational resources, making real-time processing challenging [1]. Consequently, many of these methods rely on extensive downsampling [12] or feature extraction [11] to reduce the computational demands.

B. Global Trajectory Optimization

Pose graph optimization constructs a factor graph based on relative pose constraints and optimizes the global trajectory by minimizing errors in the pose space. This method has been widely adopted due to its computational efficiency [1], [5]. However, its accuracy is often constrained by the challenge of accurately representing the uncertainty of scan matching results using a Gaussian distribution [2].

Integrating point cloud registration errors into global trajectory optimization can improve the accuracy of trajectory estimation [7], [13]. However, computing these errors across an entire map is computationally expensive. Existing approaches rely on significant downsampling [14] or GPU computation [7] to manage the high computational load.

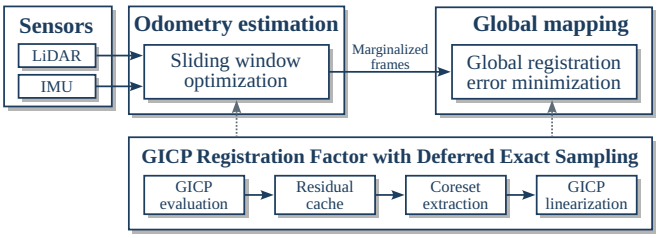


Fig. 2: Odometry estimation module, which estimates sensor ego-motion using sliding window factor graph optimization, and global mapping module, which constructs factor graph to directly minimize matching cost errors across entire map. Both modules utilize the GICP scan matching factor accelerated with the exact point cloud downsampling algorithm.

III. Methodology

Fig. 2 shows an overview of the proposed framework, which comprises odometry estimation and global trajectory estimation modules. Both modules are built with two key building blocks, namely the generalized iterative closest point (GICP) registration error factor accelerated with exact point cloud downsampling and voxel-based fast overlap estimation with occupancy bit chunks. We introduce these building blocks in Secs. III-A and III-B and then describe the proposed SLAM framework in Secs. III-C and III-D.

A. Registration Error Factor Accelerated with Exact Point Cloud Downsampling

Registration error function: To constrain the relative pose between two point clouds $\mathcal{P}_i$ and $\mathcal{P}_j$, we use the distribution-to-distribution distance metric in GICP [15]. GICP models each point $\boldsymbol{p}_k \in \mathcal{P}_j$ as a Gaussian distribution $\boldsymbol{p}_k = (\boldsymbol{\mu}_k, \boldsymbol{\Sigma}_k)$, which represents the local surface shape, and computes the distance between point $\boldsymbol{p}_k$ and its corresponding nearest point $\boldsymbol{p}'_k = (\boldsymbol{\mu}'_k, \boldsymbol{\Sigma}'_k)$ in the other point cloud as follows:

$$f^{\mathrm{GICP}}(\mathcal{P}_i, \mathcal{P}_j, \boldsymbol{T}_{ij}) = \sum_{\boldsymbol{p}_k \in \mathcal{P}_j} \| f^{\mathrm{D2D}}(\boldsymbol{p}'_k, \boldsymbol{p}_k, \boldsymbol{T}_{ij}) \|^2, \tag{1}$$

$$f^{\mathrm{D2D}}(\boldsymbol{p}'_k, \boldsymbol{p}_k, \boldsymbol{T}_{ij}) = \boldsymbol{\Phi}_k^\top \boldsymbol{d}_k, \tag{2}$$

$$\boldsymbol{d}_k = \boldsymbol{\mu}'_k - \boldsymbol{T}_{ij} \boldsymbol{\mu}_k, \tag{3}$$

$$\boldsymbol{\Phi}_k \boldsymbol{\Phi}_k^\top = (\boldsymbol{\Sigma}'_k + \boldsymbol{T}_{ij} \boldsymbol{\Sigma}_k \boldsymbol{T}_{ij}^\top)^{-1}, \tag{4}$$

where $\boldsymbol{T}_{ij} = \boldsymbol{T}_i^{-1} \boldsymbol{T}_j$ is the relative pose between $\mathcal{P}_i$ and $\mathcal{P}_j$. The decomposition $\boldsymbol{\Phi}_k \boldsymbol{\Phi}_k^\top$ of the information matrix can be efficiently obtained using Cholesky decomposition.

In the Gauss-Newton optimization, the residual function f^{GICP} is linearized at the current estimate $\breve{\boldsymbol{T}}_{ij}$ to form a quadratic error factor:

$$f^{\mathrm{GICP}}(\mathcal{P}_i, \mathcal{P}_j, \breve{\boldsymbol{T}}_{ij} \boxplus \Delta\boldsymbol{x}) \approx \Delta\boldsymbol{x}^\top \boldsymbol{H} \Delta\boldsymbol{x} + 2\boldsymbol{b}^\top \Delta\boldsymbol{x} + c, \tag{5}$$

where $\boldsymbol{H} = \boldsymbol{J}^\top \boldsymbol{J}$, $\boldsymbol{b} = \boldsymbol{J}^\top \boldsymbol{e}$, $c = \boldsymbol{e}^\top \boldsymbol{e}$, $\boldsymbol{J} = \frac{\partial \boldsymbol{e}}{\partial \boldsymbol{T}}$, and $\boldsymbol{e}$ is a stack of residuals given by f^{D2D}.

Exact point cloud downsampling: The linearization of f^{GICP} is computationally intensive because it requires the evaluation of residuals for all points in $\mathcal{P}_j$. A common approach to mitigate this computation burden is to

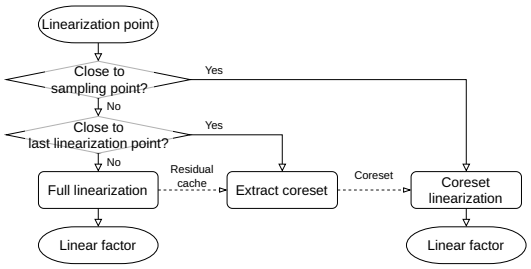


Fig. 3: Flowchart of exact point cloud downsampling with deferred sampling strategy.

decrease the number of points using random sampling [13] or geometry-aware feature selection [16], [17], [18]. However, the accuracy of these sampling methods largely depends on the number of sampled points and typically requires sampling several thousand points to maintain sufficient accuracy.

To drastically reduce the number of points (e.g., tens to hundreds) while retaining accuracy, we adopt the exact point cloud downsampling algorithm [8], which is based on the concept of coresets in computational geometry [19].

An exact coreset $\mathcal{X}'$ is a subset of input data $\mathcal{X}$ selected such that the result of an algorithm f on the coreset becomes the same as that on the original set: $f(\mathcal{X}') = f(\mathcal{X})$, where $\mathcal{X}' \subset \mathcal{X}$. Our downsampling algorithm extracts an exact coreset of the residuals of input points, ensuring that the original quadratic error function is precisely recovered. Specifically, given a relative pose (referred to as the sampling point) $\tilde{\boldsymbol{T}}_{ij}$, it extracts a subset of residuals $\tilde{\boldsymbol{e}} \subset \boldsymbol{e}$ and corresponding weights $\tilde{\boldsymbol{w}}$ such that the weighted subset yields exactly the same quadratic error function parameters $\tilde{\boldsymbol{H}}$, $\tilde{\boldsymbol{b}}$, and $\tilde{c}$ as those of the original set ($\boldsymbol{H}, \boldsymbol{b}$, and c) at $\tilde{\boldsymbol{T}}_{ij}$: $\tilde{\boldsymbol{H}} = \tilde{\boldsymbol{J}}^\top \tilde{\boldsymbol{W}} \tilde{\boldsymbol{J}} = \boldsymbol{H}$, $\tilde{\boldsymbol{b}} = \tilde{\boldsymbol{J}}^\top \tilde{\boldsymbol{W}} \tilde{\boldsymbol{e}} = \boldsymbol{b}$, $\tilde{c} = \tilde{\boldsymbol{e}} \tilde{\boldsymbol{W}} \tilde{\boldsymbol{e}} = c$, where $\tilde{\boldsymbol{J}} = \frac{\partial \tilde{\boldsymbol{e}}}{\partial \boldsymbol{T}_{ij}}$ and $\tilde{\boldsymbol{W}} = \mathrm{diag}(\tilde{\boldsymbol{w}})$. Due to space limitations, we refer the reader to [8], [19] for the detailed process of finding such a coreset[2].

When re-linearizing f^{GICP}, we evaluate only the selected subset $\tilde{\boldsymbol{e}}$ and compose a quadratic error factor. Since the subset $\tilde{\boldsymbol{e}}$ yields the same quadratic error function parameters as those for the original set $\boldsymbol{e}$, no approximation errors are introduced at the sampling point $\tilde{\boldsymbol{T}}_{ij}$. As the subset $\tilde{\boldsymbol{e}}$ can be much smaller than the original set (about 0.5% to 5% of $\boldsymbol{e}$), re-linearization is significantly faster compared to the original set. Unlike prior work [8], where f^{GICP} was re-linearized without updating point correspondences, our approach re-linearizes f^{GICP} with the coreset and updates point correspondences. While the coreset does not guarantee approximation accuracy for nonlinearity, in practice, it enables an accurate approximation of the nonlinear objective function, as demonstrated in Sec. IV-A.

Deferred sampling strategy: Although the extracted coreset provides good approximation accuracy around the sampling point $\tilde{\boldsymbol{T}}_{ij}$, the approximation error grows as the current estimate $\breve{\boldsymbol{T}}_{ij}$ deviates further from $\tilde{\boldsymbol{T}}_{ij}$. We thus re-extract a new coreset when the translation or rotation displacement between the current estimate $\breve{\boldsymbol{T}}_{ij}$ and the sampling point $\tilde{\boldsymbol{T}}_{ij}$ exceeds its threshold (e.g., 1.0 m or 1.0°, respectively).

However, naively re-extracting the coreset every time f^{GICP} is linearized would introduce overhead, especially during the initial optimization iterations when sensor pose estimates tend to change significantly, causing the newly extracted coresets to be quickly discarded in the next linearization. To mitigate this, we implement a deferred sampling strategy, as shown in Fig. 3

In this strategy, the GICP factor is first linearized at the current estimate $\breve{\boldsymbol{T}}_{ij}^0$ with all input points, and residuals $\boldsymbol{e}^0$ and their Jacobians $\boldsymbol{J}^0$ are obtained. We store $\boldsymbol{e}^0$ and $\boldsymbol{J}^0$ in cache memory and create a quadratic factor with the full residuals $\boldsymbol{e}^0$. During the next linearization, if the displacement between the current and previous linearization points, $(\breve{\boldsymbol{T}}_{ij}^0)^{-1}\breve{\boldsymbol{T}}_{ij}^1$, is below its threshold (e.g., 0.25 m or 0.25°), we perform exact downsampling on the cached residuals $\boldsymbol{e}^0$ and Jacobians $\boldsymbol{J}^0$ to extract the coreset $\tilde{\boldsymbol{e}}$. This coreset is then re-evaluated at the current linearization point $\breve{\boldsymbol{T}}_{ij}^1$ to form a new quadratic factor. As long as the current estimate $\breve{\boldsymbol{T}}_{ij}^t$ remains close to the sampling point $\tilde{\boldsymbol{T}}_{ij} = \breve{\boldsymbol{T}}_{ij}^0$, the coreset is reused to re-linearize the objective function f^{GICP}.

Since the displacements of sensor pose estimates tend to converge as the optimization proceeds, this deferred evaluation strategy significantly reduces unnecessary executions of exact downsampling.

B. Fast Overlap Estimation

We use a voxelmap-based overlap metric to manage the localization and mapping processes. To enhance processing speed, we employ a spatial voxel hashing algorithm [20] similar to those in VoxelMap [14] and Faster-LIO [21]. Additionally, we introduce a fast occupancy grid based on occupancy bit chunks to further improve efficiency. Inspired by the VDB structure in [22], we pack the binary occupancy states of $8\times8\times8$ voxels into a 512-bit chunk and store the occupancy chunks in a flat hash table. Hash collisions are resolved by open addressing. Unlike implementations [14], [21] that rely on closed addressing (e.g., with std::unordered_set), our implementation stores all voxel data in a compact contiguous memory region, making it more cache-friendly. We compute the overlap ratio by counting the number of source points that fall within occupied target voxels. We observed that this method yields up to a 2.5x processing speed improvement compared to conventional implementations [14], [21]. Note that we use this occupancy grid solely for overlap evaluation. The nearest neighbor search in GICP factors is performed using an exact nearest neighbor search, such as KdTree.

C. Odometry Estimation

Following [7], we employ keyframe-based odometry estimation with sliding window factor graph optimization, as shown in Fig. 4. Let $\boldsymbol{x}_t = [\boldsymbol{T}_t, \boldsymbol{v}_t, \boldsymbol{b}_t]$ be the sensor state at time t, where $\boldsymbol{T}_t \in SE(3)$ is the sensor pose, $\boldsymbol{v}_t \in \mathbb{R}^3$ is the velocity, and $\boldsymbol{b}_t \in \mathbb{R}^6$ is the IMU bias.

[2]The coreset extraction algorithm is available at `https://github.com/koide3/caratheodory2`.

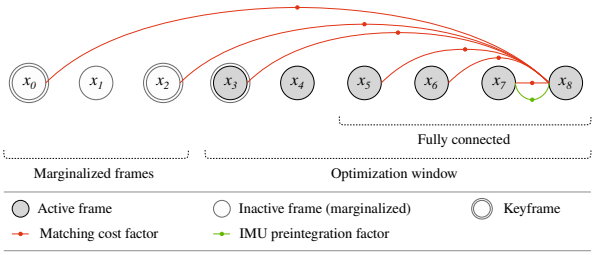


Fig. 4: Factor graph for odometry estimation[3].

We optimize the sensor states within an optimization window $\mathcal{X}^w$ that contains frames from the past 5 seconds. To bound the computational cost, frames that leave the window are marginalized from the graph. Every time a new sensor frame is inserted, we create a preintegrated IMU factor [23] between the current and previous frames. We also create GICP factors between the new frame and the preceding N^{pre} frames (e.g., three frames) to improve robustness to rapid sensor motion. To reduce estimation drift, GICP factors are also created between the latest frame and keyframes $\mathcal{X}^k$, which is a set of past frames selected such that they are well distributed in space (i.e., with maximal distance between them) while having sufficient overlap with the latest frame.

The objective function is summarized as follows:

$$g^{\text{LIO}}(\mathcal{X}^w) = \sum_{\boldsymbol{x}_j \in \mathcal{X}^w} \sum_{\boldsymbol{x}_i \in \mathcal{X}_j^p \cup \mathcal{X}^k} f^{\text{GICP}}(\mathcal{P}_i, \mathcal{P}_j, \boldsymbol{T}_i, \boldsymbol{T}_j) \quad (6)$$

$$+ \sum_{\boldsymbol{x}_i \in \mathcal{X}^w} f^{\text{IMU}}(\boldsymbol{x}_{i-1}, \boldsymbol{x}_i) + f^{\text{MG}}(\mathcal{X}^w), \quad (7)$$

where $\mathcal{X}_j^p = \{\boldsymbol{x}_{j-N^{\text{pre}}}, \ldots, \boldsymbol{x}_{j-1}\}$ is the preceding frames of $\boldsymbol{x}_j$, $\mathcal{X}^k$ is the keyframes, f^{IMU} is the IMU error factor, and f^{MG} is the error term to compensate for marginalized variables and factors. We use the iSAM2 optimizer [24] implemented in GTSAM [25] for efficient optimization and marginalization.

A key difference from [7] is that we use the GICP factor with exact nearest neighbor search (i.e., KdTree) instead of voxel-based nearest neighbor search. The use of exact nearest neighbor search improves the convergence of the optimization process. While this results in a high computational cost, exact downsampling enables real-time processing on a CPU and makes GPU computation unnecessary.

D. Global Trajectory Optimization

Frames marginalized from the odometry estimation graph are concatenated into a single point cloud at a certain interval (e.g., every 50 frames) to form a submap. The global mapping module then constructs a factor graph to optimize the submap poses to achieve a globally consistent sensor trajectory. As in [7], we create a GICP factor between every pair of submaps with an overlap that exceeds a threshold (e.g., 15%), resulting in a highly dense factor graph, as shown in Fig. 1 (b). The objective function is thus defined as

$$g^{\text{GM}}(\mathcal{X}^g) = \sum_{(\boldsymbol{T}^i, \boldsymbol{T}^j) \in \mathcal{X}^o} f^{\text{GICP}}(\mathcal{S}^i, \mathcal{S}^j, \boldsymbol{T}^i, \boldsymbol{T}^j), \quad (8)$$

[3]Reprinted from [7] with permission from Elsevier.

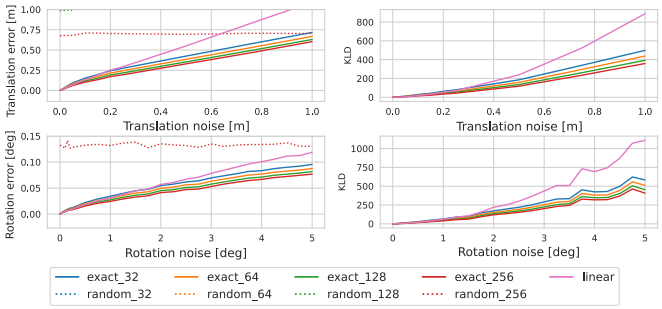


Fig. 5: Registration error function approximation errors. Exact downsampling showed zero errors at the sampling point (when noise = 0) and consistently showed errors smaller than those of the quadratic approximation (i.e., linear factor at the sampling point). Note that most of the random sampling results are not visible in this figure due to excessively large errors. The numerical values in the legend refer to the numbers of points sampled.

where $\mathcal{X}^g$ is the set of all submap poses, $\mathcal{X}^o$ is all pairs of submaps with an overlap, and $\mathcal{S}^i$ and $\boldsymbol{T}^i$ are the point cloud and the pose of submap i, respectively. The optimization is incrementally performed with the iSAM2 optimizer [24].

IV. Experiments

A. Approximation Accuracy of Exact Downsampling

Experimental setting: We evaluated the approximation accuracy of exact point cloud downsampling on the KITTI dataset [26]. We created pairs of consecutive frames from the first 500 frames of the KITTI 00 sequence and extracted an exact coreset for each pair. We then compared the linearization results of the registration error on the coreset and those on the original points under pose perturbation. The approximation error was measured using two metrics: 1) KL divergence (KLD) between the covariance matrices (i.e., $\text{KLD}(\boldsymbol{H}^{-1}, \tilde{\boldsymbol{H}}^{-1})$), 2) translation and rotation errors between the mean vectors (i.e., $\boldsymbol{\mu} = \boldsymbol{H}^{-1}\boldsymbol{b}$ and $\tilde{\boldsymbol{\mu}} = \tilde{\boldsymbol{H}}^{-1}\tilde{\boldsymbol{b}}$). For each frame pair, we repeated the evaluation 100 times, varying the magnitude of the displacement from the sampling point. As baselines, we also evaluated random sampling and quadratic approximation (i.e., linearized factor at the sampling point). The number of sampled points varied from 32 to 256. Note that random sampling often yields corrupted linearized factors with such a small number of sampled points.

Experimental results: Fig. 5 shows the evaluation results. Exact downsampling demonstrated zero approximation error for both metrics when the displacement from the sampling point was zero. This result confirms that the coresets precisely recovered the original quadratic function at the sampling point. Although quadratic approximation also showed zero error at the sampling point, its accuracy quickly deteriorated as the evaluation point moved away from the sampling point. This deterioration suggests that the registration error factor is highly nonlinear and cannot be adequately captured by linear approximations. Random sampling exhibited large approximation errors with this very

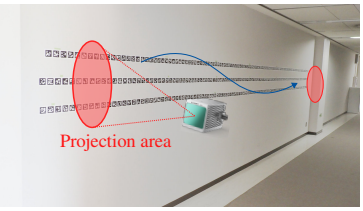


Fig. 6: Setup for flat wall experiment [7]. The LiDAR was moved between pillars, which induced severe point cloud degeneration for a few seconds.

TABLE I: Point Cloud Degeneration Test Results (ATEs [m])

Seq.	FLIO [4]	VoxelMap [14]	SLICT [1]	GLIM [7]	Proposed
01	0.815	0.577	0.947	0.118	0.424
02	0.822	0.146	0.331	0.299	0.092
03	0.873	0.950	1.088	0.040	0.114
04	1.137	0.586	0.729	0.389	0.448
05	1.048	0.786	0.747	0.228	0.311
06	15.551	0.807	0.311	0.056	0.068
07	0.635	0.366	0.659	0.017	0.014
08	0.297	0.279	0.983	0.146	0.045
Average	2.647	0.562	0.724	0.162	0.190

small number of sampled points. For example, random sampling with 256 points resulted in mean errors of 0.698 m and 0.132°and KLD errors of 2.7×10^3 and 4.1×10^3 on average. These results indicate that random sampling requires a significantly larger number of points (e.g., more than 1000) to reasonably approximate the original error function.

B. Robustness to Point Cloud Degeneration

Experimental setting: We evaluated the proposed odometry estimation method on the *flatwall* dataset[4] to demonstrate its robustness to point cloud degeneration. This dataset consists of eight sequences recorded with a Livox Avia. In each sequence, the LiDAR was moved between two pillars in a corridor while facing a flat wall, which induced severe point cloud degeneration for a few seconds. While the durations of the sequences are relatively short (8.8 to 18.5 s), this dataset poses a significant challenge for existing methods, as point cloud degeneration makes accurate odometry estimation difficult.

We compared the proposed method with four state-of-the-art tightly coupled LiDAR-IMU odometry methods, namely FAST-LIO2 (FLIO) [4], based on the iterated Kalman filter, VoxelMap [14], based on a voxel-based efficient plane representation, SLICT [1], which uses continuous-time sliding window optimization, and GLIM [7], which uses GPU-accelerated sliding window optimization.

Experimental results: Table I summarizes the absolute trajectory errors (ATEs) [27] for the evaluated methods. FAST-LIO2, VoxelMap, and SLICT were greatly affected by the severe degeneration of point clouds and thus exhibited large average ATEs (2.647, 0.562, and 0.724 m, respectively). The proposed method showed an average ATE of 0.190 m, which is comparable to that of GLIM (0.162 m), even though it does not rely on GPU acceleration. This performance is attributed to sliding window optimization, which effectively corrects trajectory drift during point cloud degeneration by propagating future observations to past states.

[4] https://staff.aist.go.jp/k.koide/projects/glimsupp/flatwall.html

TABLE II: ATEs [m] for Odometry Estimation Methods on MCD VIRAL Dataset

Sequence	FLIO [4]	DLIO [9]	SLICT [1]	GLIM [7]	Proposed
ntu_day_01	1.510	1.925	1.890	1.054	**0.918**
ntu_day_02	0.272	0.636	**0.168**	0.259	0.289
ntu_day_10	2.084	3.052	1.429	**1.099**	1.361
ntu_night_04	1.599	2.373	1.002	1.007	**0.961**
ntu_night_08	1.425	2.056	**0.822**	1.558	1.846
ntu_night_13	0.903	1.928	**0.574**	0.785	0.780
kth_day_06	1.005	0.562	0.633	**0.283**	0.466
kth_day_09	0.733	0.326	0.262	**0.194**	0.219
kth_day_10	2.176	0.665	0.737	**0.204**	0.296
kth_night_01	1.040	0.414	0.540	**0.317**	0.403
kth_night_04	0.567	0.376	0.441	**0.152**	0.256
kth_night_05	2.158	0.903	0.855	0.259	**0.185**
tuhh_day_02	0.273	0.283	0.236	**0.185**	0.268
tuhh_day_03	0.970	0.731	0.743	0.843	**0.459**
tuhh_day_04	**0.077**	0.232	0.084	0.124	0.083
tuhh_night_07	0.279	0.436	0.227	**0.120**	0.272
tuhh_night_08	0.749	0.685	0.740	0.605	**0.520**
tuhh_night_09	**0.057**	0.375	0.094	0.089	0.067
Average	0.993	0.998	0.638	0.508	0.536

Best values are shown in bold.

C. Quantitative Evaluation on MCD VIRAL dataset

Experimental setting: To quantitatively evaluate the accuracy of the proposed method in practical scenarios, we conducted experiments on the MCD VIRAL dataset [28]. This dataset consists of 18 sequences recorded with an Ouster OS1-128 and a Livox Mid70. Six sequences (ntu sequences) were recorded with an on-vehicle setup. The remaining sequences (kth and tuhh sequences) were recorded with a handheld sensor setup. The average and maximum durations of the sequences are respectively 537 and 969 s.

Odometry estimation accuracy: We compared the proposed odometry estimation with FAST-LIO2 [4], DLIO [9], SLICT [1], and GLIM [7]. Loop closure was disabled for all methods. Table II summarizes the ATEs for the evaluated methods. The results for FAST-LIO2, DLIO, and SLICT were taken from [28]. Note that our method used only point clouds obtained using Ouster OS1-128.

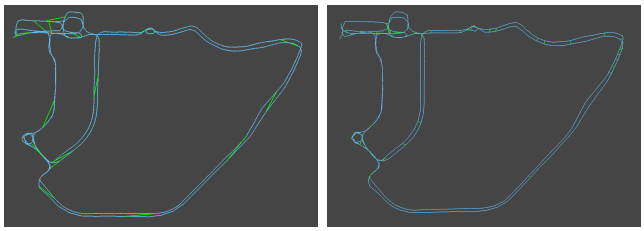
FAST-LIO2 and DLIO, based on state filtering, were greatly affected by the challenging setup, which involved high-speed motion and dynamic environment changes, and thus exhibited large average ATEs (0.993 and 0.998 m, respectively). Although SLICT exhibited a better ATE (0.638 m), it was computationally intensive and failed to achieve real-time processing. Among the compared methods, GLIM showed the best average ATE (0.508 m) owing to its robust sliding window optimization. However, it required GPU acceleration for real-time processing. The proposed method showed an average ATE (0.536 m) that was comparable to that of GLIM owing to sliding window optimization.

Global mapping accuracy: We compared the proposed SLAM framework including loop closure with three baseline methods. SLICT [1] uses conventional scan-matching-based loop detection and pose-graph-based trajectory optimization. PGO is conventional pose-graph-based trajectory optimization implemented in GLIM [7], similar to that of SLICT.

TABLE III: ATEs [m] for Global Trajectory Optimization Methods with Loop Closure on MCD VIRAL Dataset

Sequence	SLICT [1]	PGO [7]	GLIM [7]	Proposed
ntu_day_01	0.885	0.853	**0.686**	0.731
ntu_day_02	**0.179**	0.390	0.185	0.255
ntu_day_10	1.108	0.815	**0.625**	0.689
ntu_night_04	1.312	0.956	0.983	**0.881**
ntu_night_08	0.678	0.678	0.582	**0.560**
ntu_night_13	0.717	0.667	0.668	**0.619**
kth_day_06	0.330	0.364	**0.144**	0.227
kth_day_09	0.122	0.169	0.109	**0.098**
kth_day_10	0.250	0.327	0.220	**0.174**
kth_night_01	0.340	0.318	0.277	**0.248**
kth_night_04	0.200	0.292	0.128	**0.095**
kth_night_05	0.329	**0.185**	0.241	0.203
tuhh_day_02	0.190	0.215	0.125	**0.119**
tuhh_day_03	0.472	**0.360**	0.718	0.593
tuhh_day_04	0.069	0.111	0.069	**0.063**
tuhh_night_07	0.199	0.165	**0.129**	0.134
tuhh_night_08	**0.403**	0.512	0.552	0.500
tuhh_night_09	0.103	0.075	**0.055**	0.058
Average	0.438	0.414	0.361	0.347

Best values are shown in bold.



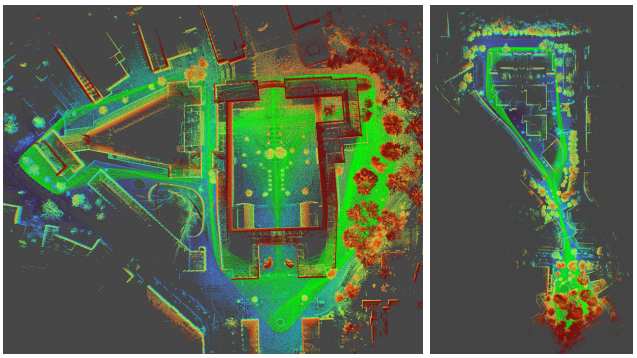
(a) SLICT (b) PGO

Fig. 7: Global optimization graphs. Blue and green lines respectively represent odometry and loop factors.

GLIM [7] uses the same global trajectory optimization factor graph structure as that of the proposed method with GPU-accelerated registration error factors.

Table III summarizes the ATEs of the compared methods with loop closure. Although SLICT had a greatly improved accuracy compared to that without loop closure, it showed the largest ATE among the compared methods (0.438 m). PGO showed the second largest average ATE (0.414 m). In particular, SLICT and PGO exhibited large ATEs in longer sequences (ntu sequences). Fig. 7 shows the pose graphs generated by SLICT and PGO for the ntu_day_01 sequence. Blue and green lines represent odometry and loop constraints, respectively. The generated pose graphs are very sparse, as pose-graph-based methods have difficulty closing loops for frames with a small overlap. This result suggests that it is difficult to accurately correct trajectories with large loops using pose graph optimization.

GLIM and the proposed method showed better ATEs compared to those of the pose-graph-based methods (0.361 and 0.347 m, respectively). Fig. 1 (b) and Fig. 8 show that the proposed method constructed dense registration error minimization factor graphs that enabled accurate correction of trajectories with large loops. The proposed method exhibited a slightly better average ATE compared to that of GLIM. Although the difference is not significant, we consider that the use of exact nearest neighbor search resulted in better convergence compared to that for GLIM, which is based on



(a) kth_day_06 (b) tuhh_day_03

Fig. 8: Example of mapping results and global trajectory optimization graphs.

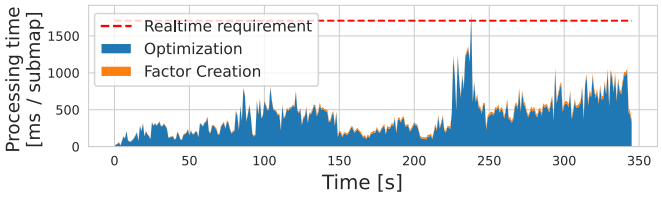


Fig. 9: Processing time of global mapping for ntu_day_01.

voxel-based approximate nearest neighbor search.

Processing time: We measured the processing times of the odometry estimation and the global mapping modules on the ntu_day_01 sequence, one of the longest sequences in the dataset. The timings were recorded on an Intel Core i7 8700K with 32 GB of RAM. Note that the factor linearization and the linear solver were made multi-threaded using Intel TBB.

The odometry estimation and the global mapping took approximately 51.9 ms (for each frame) and 433.9 ms (for each submap), respectively, both well within the real-time requirements (100 ms per frame and an average submap interval of 1.7 s). Fig. 9 shows how the processing time of the global mapping module changed as the mapping progressed. Thanks to the exact downsampling algorithm and incremental optimization with iSAM2, the global optimization quickly converged every time a new submap was inserted.

As an ablation study, we measured the processing times with exact downsampling disabled. The average processing times of odometry estimation and global trajectory optimization deteriorated significantly, falling below the real-time requirements (284.8 and 1831.8 ms, respectively). This result confirms that exact downsampling greatly accelerates the optimization process.

V. Conclusions

We proposed a range-inertial SLAM framework with GICP factors accelerated with the exact downsampling algorithm. We devised a deferred sampling strategy to mitigate the processing cost of exact downsampling. The experimental results demonstrated that the proposed method enables the real-time processing of registration error minimization factor graphs on a standard CPU and exhibits an accuracy comparable to those of GPU-based methods.

REFERENCES

[1] T.-M. Nguyen, D. Duberg, P. Jensfelt, S. Yuan, and L. Xie, "Slict: Multi-input multi-scale surfel-based lidar-inertial continuous-time odometry and mapping," *IEEE Robotics and Automation Letters*, vol. 8, no. 4, pp. 2102–2109, Apr. 2023.

[2] K. Koide, M. Yokozuka, S. Oishi, and A. Banno, "Globally consistent 3d LiDAR mapping with GPU-accelerated GICP matching cost factors," *IEEE Robotics and Automation Letters*, vol. 6, no. 4, pp. 8591–8598, Oct. 2021.

[3] Z. Liu and F. Zhang, "BALM: Bundle adjustment for lidar mapping," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 3184–3191, Apr. 2021.

[4] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "FAST-LIO2: Fast direct LiDAR-inertial odometry," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2053–2073, Aug. 2022.

[5] J. Behley and C. Stachniss, "Efficient surfel-based slam using 3d laser range data in urban environments." in *Robotics: Science and Systems*, vol. 2018, 2018, p. 59.

[6] X. Liu, Z. Liu, F. Kong, and F. Zhang, "Large-scale LiDAR consistent mapping using hierarchical LiDAR bundle adjustment," *IEEE Robotics and Automation Letters*, vol. 8, no. 3, pp. 1523–1530, Mar. 2023.

[7] K. Koide, M. Yokozuka, S. Oishi, and A. Banno, "GLIM: 3D range-inertial localization and mapping with gpu-accelerated scan matching factors," *Robotics and Autonomous Systems*, vol. 179, p. 104750, Sept. 2024.

[8] K. Koide, S. Oishi, M. Yokozuka, and A. Banno, "Exact point cloud downsampling for fast and accurate global trajectory optimization," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Oct. 2023.

[9] K. Chen, R. Nemiroff, and B. T. Lopez, "Direct lidar-inertial odometry: Lightweight lio with continuous-time motion correction," in *IEEE International Conference on Robotics and Automation*. IEEE, May 2023.

[10] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time." in *Robotics: Science and Systems*, vol. 2, no. 9. Berkeley, CA, 2014, pp. 1–9.

[11] X. Zuo, Y. Yang, P. Geneva, J. Lv, Y. Liu, G. Huang, and M. Pollefeys, "Lic-fusion 2.0: Lidar-inertial-camera odometry with sliding-window plane-feature tracking," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Oct. 2020.

[12] K. Li, M. Li, and U. D. Hanebeck, "Towards high-performance solid-state-lidar-inertial odometry and mapping," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5167–5174, July 2021.

[13] V. Reijgwart, A. Millane, H. Oleynikova, R. Siegwart, C. Cadena, and J. Nieto, "Voxgraph: Globally consistent, volumetric mapping using signed distance function submaps," *IEEE Robotics and Automation Letters*, vol. 5, no. 1, pp. 227–234, Jan. 2020.

[14] C. Yuan, W. Xu, X. Liu, X. Hong, and F. Zhang, "Efficient and probabilistic adaptive voxel mapping for accurate online lidar odometry," *IEEE Robotics and Automation Letters*, vol. 7, no. 3, pp. 8518–8525, July 2022.

[15] A. Segal, D. Haehnel, and S. Thrun, "Generalized-icp." in *Robotics: science and systems*, vol. 2, no. 4. Seattle, WA, 2009, p. 435.

[16] S. Yi, Y. Lyu, L. Hua, Q. Pan, and C. Zhao, "Light-loam: A lightweight lidar odometry and mapping based on graph-matching," *IEEE Robotics and Automation Letters*, vol. 9, no. 4, pp. 3219–3226, Apr. 2024.

[17] Y. Duan, J. Peng, Y. Zhang, J. Ji, and Y. Zhang, "Pfilter: Building persistent maps through feature filtering for fast and accurate lidar-based slam," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Oct. 2022.

[18] J. Jiao, Y. Zhu, H. Ye, H. Huang, P. Yun, L. Jiang, L. Wang, and M. Liu, "Greedy-based feature selection for efficient lidar slam," in *IEEE International Conference on Robotics and Automation*. IEEE, May 2021, pp. 5222–5228.

[19] A. Maalouf, I. Jubran, and D. Feldman, "Fast and accurate least-mean-squares solvers," in *Conference on Neural Information Processing Systems*, vol. 32. Curran Associates, Inc., 2019.

[20] M. Teschner, B. Heidelberger, M. Müller, D. Pomerantes, and M. H. Gross, "Optimized spatial hashing for collision detection of deformable objects." in *Vmv*, vol. 3, 2003, pp. 47–54.

[21] C. Bai, T. Xiao, Y. Chen, H. Wang, F. Zhang, and X. Gao, "Faster-LIO: Lightweight tightly coupled lidar-inertial odometry using parallel sparse incremental voxels," *IEEE Robotics and Automation Letters*, vol. 7, no. 2, pp. 4861–4868, Apr. 2022.

[22] K. Museth, "Vdb: High-resolution sparse volumes with dynamic topology," *ACM Transactions on Graphics*, vol. 32, no. 3, pp. 1–22, June 2013.

[23] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation," in *Robotics: Science and Systems XI*, ser. RSS2015. Robotics: Science and Systems Foundation, July 2015.

[24] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "isam2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering," in *IEEE International Conference on Robotics and Automation*. IEEE, May 2011.

[25] F. Dellaert and G. Contributors, "borglab/gtsam," May 2022. [Online]. Available: https://github.com/borglab/gtsam)

[26] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013.

[27] Z. Zhang and D. Scaramuzza, "A tutorial on quantitative trajectory evaluation for visual(-inertial) odometry," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Oct. 2018, pp. 7244–7251.

[28] T.-M. Nguyen, S. Yuan, T. H. Nguyen, P. Yin, H. Cao, L. Xie, M. Wozniak, P. Jensfelt, M. Thiel, J. Ziegenbein, and N. Blunder, "Mcd: Diverse large-scale multi-campus dataset for robot perception," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 6 2024.