Exact Point Cloud Downsampling for Fast and Accurate Global Trajectory Optimization

Kenji Koide[1], Shuji Oishi[1], Masashi Yokozuka[1], and Atsuhiko Banno[1]

***Abstract*— This paper presents a point cloud downsampling algorithm for fast and accurate trajectory optimization based on global registration error minimization. The proposed algorithm selects a weighted subset of residuals of the input point cloud such that the subset yields exactly the same quadratic point cloud registration error function as that of the original point cloud at the evaluation point. This method accurately approximates the original registration error function with only a small subset of input points (29 residuals at a minimum). Experimental results using the KITTI dataset demonstrate that the proposed algorithm significantly reduces processing time (by 87%) and memory consumption (by 99%) for global registration error minimization while retaining accuracy.**

I. Introduction

Global trajectory optimization is a crucial step for localization and mapping systems. Because it is unavoidable that trajectory errors accumulate in online odometry estimation that performs real-time optimization using local observations, it is necessary to correct estimation drift by considering the global consistency of the map.

Global registration error minimization is one of the most accurate approaches to global trajectory optimization [1]. Unlike the conventional pose graph optimization that minimizes the errors in the pose space [2], global registration error minimization directly minimizes the multi-frame point cloud registration errors over the entire map. This approach avoids the Gaussian approximation of the relative pose constraint and enables accurate trajectory optimization by jointly aligning all frames in the map [3]. However, it is known to be computationally expensive compared to pose graph optimization, as it requires a re-evaluation of registration error functions that involve residual computations for many points [4]. It also consumes a substantial amount of memory in order to remember the point correspondences between frames.

To mitigate the processing cost and memory consumption of global registration error minimization, we propose a point cloud downsampling algorithm based on an efficient and exact weighted coreset extraction algorithm [5]. A coreset is a subset of an input dataset selected such that the result of an algorithm on the coreset approximates that on the original set [6]. For example, considering a Hessian matrix calculated from a Jacobian matrix ($\boldsymbol{H} = \boldsymbol{J}^T\boldsymbol{J}$ and $\boldsymbol{J} \in \mathbb{R}^{N\times D}$), a weighted coreset $\boldsymbol{S} \in \mathbb{R}^{M\times D}$ approximates the original Hessian matrix with a subset of the Jacobian matrix ($\boldsymbol{S}^T\boldsymbol{S} \approx \boldsymbol{H}$ and $\boldsymbol{S} \subset \boldsymbol{J}$). In this work, we employ a novel coreset extraction algorithm, which finds a coreset to *exactly* represent the original Hessian matrix in a time linear to the number of points [5].

*This work was supported in part by JSPS KAKENHI Grant Number 23K16979 and a project commissioned 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

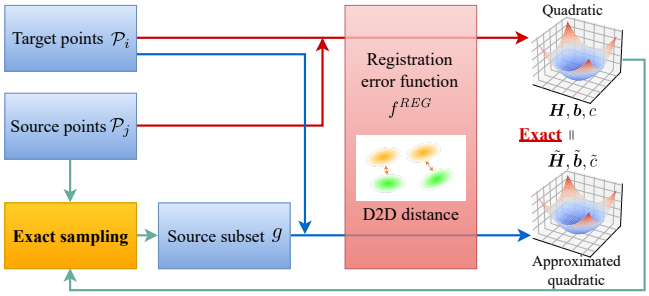


Fig. 1: The proposed algorithm extracts a weighted subset of input source points such that the subset yields the same quadratic registration error function as that of the original points at the evaluation point.

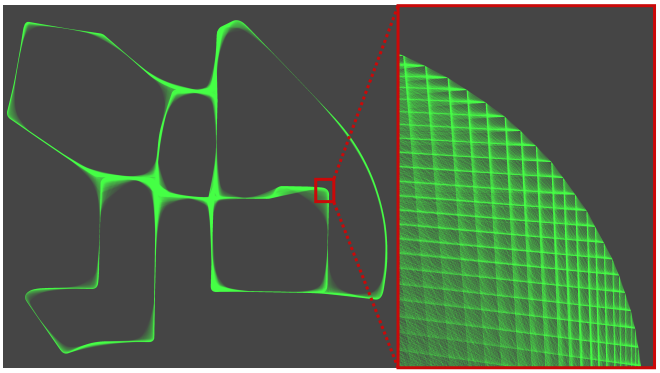


Fig. 2: Dense factor graph for global registration error minimization. The proposed algorithm reduces memory consumption by 99% and processing time by 87% for the optimization of the factor graph.

Using this algorithm, we find a coreset of the residuals of the input point cloud such that it exactly reconstructs the original quadratic registration error function at the evaluation point (see Fig. 1). The proposed method needs only 29 residuals at a minimum to compose an exact coreset for a quadratic error function with six-dimensional input. It is also able to find a larger coreset with M residuals (e.g., M=256) to enhance the nonlinearity approximation accuracy. Note that because one three-dimensional point yields three residuals for point cloud registration, the computation of 29 residuals

entails a processing cost approximately equal to that for 10 points.

For global trajectory optimization, we construct an extremely dense global registration error minimization factor graph, as shown in Fig. 2. The graph contains 4,540 pose variables and 585,417 registration error factors. We evaluate the residuals of approximately 10,000 points for each factor at every optimization iteration. The optimization requires approximately 22 GB of memory and 13 hours on a CPU with 128 threads. The proposed algorithm drastically decreases the number of residuals to be computed and reduces memory consumption and processing time to approximately 0.25 GB and 1.7 hours, respectively, while retaining estimation accuracy.

The main contributions of this work can be summarized as follows:

- We extend the exact coreset extraction algorithm in [5] so that it can extract coresets with an arbitrary number of residuals. We also made several modifications to improve the processing speed of the algorithm.
- Based on the extended exact coreset extraction algorithm, we propose a point cloud downsampling algorithm that exactly recovers the original quadratic registration error function at the evaluation point. It also shows good approximation accuracy for the nonlinearity of the original function.
- We release the code of the proposed coreset extraction algorithm as open source.

II. Related Work

A. Pose Graph Optimization

Pose graph optimization is the gold standard for global trajectory optimization for range-based SLAM [2]. It models the relative pose between frames estimated by scan matching as a Gaussian distribution and optimizes the sensor poses via maximum a posteriori estimation in the pose space. To model the Gaussian relative pose constraint, an explicit estimate of the representative relative pose value (i.e., mean) and its uncertainty (i.e., covariance) is required. However, scan matching becomes unreliable when two frames have only a small overlap, making it difficult to obtain an accurate relative pose estimate, which, in turn, makes it difficult to constrain the relative pose between small overlapping frames with pose graph optimization [3]. The uncertainty estimation of a scan matching result is also difficult in practice [7], and many works use inappropriate covariance matrices to construct the relative pose constraints (e.g., using constant covariance matrices [8] or a simple weighting scheme [9]). These difficulties in the modeling of the relative pose constraint lead to inaccurate estimation results.

B. Bundle Adjustment

Bundle adjustment (BA) is another approach to global trajectory optimization that simultaneously optimizes sensor poses and environment parameters. For range-based SLAM, BA is often formulated as the simultaneous estimation of pose variables and plane or edge environment parameters [10]. Because the plane and edge parameters can be estimated from sensor poses and point coordinates, they can be eliminated from the estimation variables, turning the BA problem into a direct eigenvalue minimization of the accumulated points [11]. While this approach ensures a consistent mapping result, it can suffer from high computation cost and a small convergence basin due to the non-least-squares error function. To mitigate these issues, it is necessary to combine range-based BA with hierarchical pose graph optimization [12].

C. Global Registration Error Minimization

Global registration error minimization is an approach to directly minimize multi-frame registration errors over the entire map [1]. Because it avoids the Gaussian approximation of pose graph optimization, it enables the construction of accurate relative pose constraints between point clouds with a very small overlap. Furthermore, it naturally propagates the per-point uncertainty of input data to the pose uncertainty via re-linearization of point cloud registration errors. While it shows a substantially better accuracy compared to pose graph optimization, it is computationally expensive because it re-evaluates point cloud registration errors between all point cloud pairs at every optimization iteration.

To mitigate the computation cost of global registration error minimization, one may consider reducing the number of input points by, for example, random sampling [4]. However, this may change the shape of the objective function, depending on the selected points, and can negatively and substantially affect the optimization result. Although there is another approach to speeding up global registration error minimization with GPU-acceleration, the computation cost is still high, and it requires submap-level registration [3].

D. Coreset Extraction

Geometric data summarization is an essential tool for handling big data in computational geometry, and coresets are one of the most important classes of summarization methods. A coreset is a subset of an input dataset selected such that the result of an algorithm on it approximates the result on the original set [6]. When a coreset produces the same output of an algorithm as that of the original set, it is called exact. For example, Caratheodory's theorem states that every point in a convex hull of points in $\mathbb{R}^D$ can be represented by a weighted sum of at most $D+1$ points [13]. This means there always exists an exact coreset with only $D+1$ points to represent a point in a convex hull (a.k.a., a Caratheodory set).

While Caratheodory provided an algorithm to find such a coreset, the time required was $O(N^2D^2)$, rendering it impractical for large-scale problems [13]. Although there were several subsequent works proposing efficient approximating algorithms to find a Caratheodory set, an exact algorithm with a linear time complexity had, until recently, been unavailable. However, in 2019, an epoch-making algorithm was proposed by Maalouf et al., one that finds an exact Caratheodory set in time linear to the number of input data

points [5]. The method was applied to find an exact coreset to reconstruct a Hessian matrix using only a small subset of the input data for accelerating least squares optimization.

Our aim in this work is to introduce this novel data summarization technique to drastically reduce the computation cost of point cloud registration while retaining the accuracy of the original set by extracting an exact coreset of input points. To the best of our knowledge, this is the first work that introduces this type of exact data summarization technique in the context of point cloud registration.

III. Methodology

A. Problem Setting

Given a sequence of point clouds $\mathcal{P}_i = \{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_N\}$ and an initial guess of their poses $\breve{\boldsymbol{T}}_i$, we estimate refined sensor poses $\boldsymbol{T}_i$ by minimizing the registration errors between all overlapping point clouds. The registration error function f^{REG} is defined as the sum of least square errors between corresponding points:

$$f^{REG}(\mathcal{P}_i, \mathcal{P}_j, \boldsymbol{T}_i, \boldsymbol{T}_j) = \sum_{\boldsymbol{p}_k \in \mathcal{P}_j} \| f^{DIST}(\boldsymbol{p}'_k, \boldsymbol{p}_k, \boldsymbol{T}_i, \boldsymbol{T}_j) \|^2, \quad (1)$$

$$= \boldsymbol{e}^T \boldsymbol{e}, \quad (2)$$

where $\boldsymbol{p}'_k \in \mathcal{P}_i$ is the corresponding point of $\boldsymbol{p}_k \in \mathcal{P}_j$, f^{DIST} is a distance function between points that returns a residual vector $\boldsymbol{e}_k \in \mathbb{R}^3$, and $\boldsymbol{e} = [\boldsymbol{e}_1^T, \boldsymbol{e}_2^T, \cdots]^T$ is a stack of $\boldsymbol{e}_k$.

In Gauss-Newton optimization, the error function f^{REG} is linearized at the current estimate $\breve{\boldsymbol{T}} = (\breve{\boldsymbol{T}}_i, \breve{\boldsymbol{T}}_j)$ and modeled in the quadratic form

$$f^{REG}(\mathcal{P}_i, \mathcal{P}_j, \breve{\boldsymbol{T}} \boxplus \Delta\boldsymbol{x}) \approx \Delta\boldsymbol{x}^T \boldsymbol{H} \Delta\boldsymbol{x} + 2\boldsymbol{b}^T \Delta\boldsymbol{x} + c, \quad (3)$$

where $c = f^{REG}(\mathcal{P}_i, \mathcal{P}_j, \breve{\boldsymbol{T}})$ is a constant, $\boldsymbol{J} = \partial \boldsymbol{e} / \partial \boldsymbol{T}$ is the Jacobian of residuals evaluated at $\breve{\boldsymbol{T}}$, and $\boldsymbol{b} = \boldsymbol{J}^T \boldsymbol{e}$ and $\boldsymbol{H} = \boldsymbol{J}^T \boldsymbol{J}$ are the information vector and matrix, respectively.

Because the linearization of f^{REG} involves residual computation for all input points in $\mathcal{P}_j$, it is computationally expensive. It also consumes a substantial amount of memory to remember the corresponding points for each point cloud pair.

To mitigate the memory consumption and processing cost of global registration error minimization, we introduce a downsampling function that takes as input the error function, point clouds, and current estimate of sensor poses, and outputs a weighted subset of residuals:

$$f^{DOWN}(f^{REG}, \mathcal{P}_i, \mathcal{P}_j, \breve{\boldsymbol{T}}_i, \breve{\boldsymbol{T}}_j) = (\boldsymbol{w}_{ij}, g_{ij}), \quad (4)$$

where $g_{ij}(\boldsymbol{e}) = \tilde{\boldsymbol{e}}$ is a function to select M residuals from $\boldsymbol{e}$ (i.e., $\tilde{\boldsymbol{e}} \subset \boldsymbol{e}$ and $\tilde{\boldsymbol{e}} \in \mathbb{R}^M$), and $\boldsymbol{w}_{ij} \in \mathbb{R}^M$ is the weights for the selected residuals. During optimization, we use the subset of residuals $\tilde{\boldsymbol{e}}$ instead of the original set to re-linearize the nonlinear error function f^{REG}.

To accurately approximate the original error function with only a small subset of the input data, we select a subset that will yield exactly the same quadratic error function as the original function at the evaluation point $\breve{\boldsymbol{x}}$:

$$\boldsymbol{H} = \tilde{\boldsymbol{H}}, \qquad \boldsymbol{b} = \tilde{\boldsymbol{b}}, \qquad c = \tilde{c}, \quad (5)$$

where

$$\tilde{\boldsymbol{e}} = g_{ij}(\boldsymbol{e}), \quad \tilde{\boldsymbol{J}} = \frac{\partial \tilde{\boldsymbol{e}}}{\partial \boldsymbol{T}}, \quad \boldsymbol{W}_{ij} = \mathrm{diag}(\boldsymbol{w}_{ij}), \quad (6)$$

$$\tilde{\boldsymbol{H}} = \tilde{\boldsymbol{J}}^T \boldsymbol{W}_{ij} \tilde{\boldsymbol{J}}, \quad \tilde{\boldsymbol{b}} = \tilde{\boldsymbol{J}}^T \boldsymbol{W}_{ij} \tilde{\boldsymbol{e}}, \quad \tilde{c} = \tilde{\boldsymbol{e}}^T \boldsymbol{W}_{ij} \tilde{\boldsymbol{e}}. \quad (7)$$

For simplicity and efficiency, we assume that the initial guess of sensor poses is reasonably accurate and use the point correspondences found at the initial state during optimization.

B. Registration Error Function

The proposed downsampling algorithm only requires the error function to be a first-order differentiable least squares function. It thus can be applied to most of the common registration error functions (e.g., point-to-point [14] and point-to-plane [15] distance functions). In this work, we use the generalized ICP (GICP) error function [16] based on the distribution-to-distribution distance.

GICP models each point as a Gaussian distribution $\boldsymbol{p}_k = (\boldsymbol{\mu}_k, \boldsymbol{C}_k)$ representing the local surface shape around the point and calculates the distribution-to-distribution distance between a point $\boldsymbol{p}_k \in \mathcal{P}_j$ and its corresponding point $\boldsymbol{p}'_k \in \mathcal{P}_i$ as follows:

$$f^{GICP}(\boldsymbol{p}'_k, \boldsymbol{p}_k, \boldsymbol{T}_i, \boldsymbol{T}_j) = \boldsymbol{d}^T \boldsymbol{\Omega} \boldsymbol{d}, \quad (8)$$

$$\boldsymbol{d} = \boldsymbol{\mu}'_k - \boldsymbol{T}_{ij} \boldsymbol{\mu}_k, \qquad \boldsymbol{\Omega} = \boldsymbol{C}'_k + \boldsymbol{T}_{ij} \boldsymbol{C}_k \boldsymbol{T}_{ij}^T, \quad (9)$$

where $\boldsymbol{T}_{ij} = \boldsymbol{T}_i^{-1} \boldsymbol{T}_j$ is the relative pose between $\boldsymbol{T}_i$ and $\boldsymbol{T}_j$.

For ease of the following downsampling process, we decompose $\boldsymbol{\Omega} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$ and use the following function f^{DIST} that gives the same least square errors as f^{GICP}:

$$f^{DIST}(\boldsymbol{p}'_k, \boldsymbol{p}_k, \boldsymbol{T}_i, \boldsymbol{T}_j) = \boldsymbol{\Phi}^T \boldsymbol{d}. \quad (10)$$

Because $\boldsymbol{\Omega}$ is a symmetric positive definite matrix, the decomposition $\boldsymbol{\Omega} = \boldsymbol{\Phi}\boldsymbol{\Phi}^T$ can efficiently be found via Cholesky decomposition. This modification does not change the final objective function and thus does not affect the optimization process.

C. Fast Coreset Extraction

To extract a weighted subset of input residuals that yields the same quadratic error function as that of the original set, we use an extended version of the efficient coreset set extraction algorithm in [5].

Caratheodory's theorem [13] states that every point in a convex hull of points in $\mathbb{R}^D$ can be represented as a weighted sum of a subset of at most $D+1$ points. Given a Jacobian matrix $\boldsymbol{J} = [\boldsymbol{a}_1^T, \cdots, \boldsymbol{a}_N^T]^T$, where $\boldsymbol{a}_k = \frac{\partial e_k}{\partial \boldsymbol{T}} \in \mathbb{R}^{1 \times D}$ represents derivatives of a residual e_k, the Hessian matrix $\boldsymbol{H} = \boldsymbol{J}^T \boldsymbol{J}$ can be given by $\sum_{k=1}^N \boldsymbol{a}_k^T \boldsymbol{a}_k$. We consider *flattened* vectors $\boldsymbol{h}_k \in \mathbb{R}^{D^2}$ of $\boldsymbol{a}_k^T \boldsymbol{a}_k$ and calculate their mean $\boldsymbol{\mu}^h = \frac{\sum_{k=1}^N \boldsymbol{h}_k}{N} = \frac{\sum_{k=1}^N \mathrm{flatten}(\boldsymbol{a}_k^T \boldsymbol{a}_k)}{N}$. Because $\boldsymbol{\mu}^h$ is always in the convex hull of $\boldsymbol{h}_k$, we can find a minimum exact coreset (a.k.a., the Caratheodory set) of $\boldsymbol{h}_k$ to represent $\boldsymbol{\mu}^h$ and scale it to recover the original Hessian matrix from at most $D^2 + 1$ row vectors in the original Jacobian matrix.

Algorithm 1 Caratheodory($\boldsymbol{P}, u, M$)

Input:
A set of points $\boldsymbol{P} = \{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_N\}$ in $\mathbb{R}^L$
A weighting function $u : \boldsymbol{P} \to [0, \infty]$
Target output size M
Output:
A subset of input points and weighting function $(\boldsymbol{S}, w)$ s.t. $\sum (u(\boldsymbol{p}_i) \cdot \boldsymbol{p}_i) = \sum (w(\boldsymbol{s}_i) \cdot \boldsymbol{s}_i)$ and $|\boldsymbol{S}| = M$

if $N \leq M$ **then**
  return $(\boldsymbol{P}, u)$
for $\boldsymbol{p}_i \in \{\boldsymbol{p}_2, \cdots, \boldsymbol{p}_N\}$ **do**
  $\boldsymbol{a}_i \leftarrow \boldsymbol{p}_i - \boldsymbol{p}_1$
$\boldsymbol{A} \leftarrow [\boldsymbol{a}_2 | \cdots | \boldsymbol{a}_N]$ ▷ $\boldsymbol{A} \in \mathbb{R}^{L \times (N-1)}$
Compute $\boldsymbol{v} = [v_2, \cdots, v_N]^T \neq 0$ s.t. $\boldsymbol{A}\boldsymbol{v} = 0$
$v_1 \leftarrow -\sum_{i=2}^{N} v_i$
$\alpha \leftarrow \min\{u(\boldsymbol{p}_i)/v_i | i \in \{1, \cdots, N\} \text{ and } v_i > 0\}$
$w(\boldsymbol{p}_i) \leftarrow u(\boldsymbol{p}_i) - \alpha v_i$ for every $i \in \{1, \cdots, N\}$
$\boldsymbol{S} \leftarrow \{\boldsymbol{p}_i | w(\boldsymbol{p}_i) > 0 \text{ and } i \in \{1, \cdots, N\}\}$
if $|\boldsymbol{S}| > M$ **then**
  return CARATHEODORY($\boldsymbol{S}, w, M$)
return $(\boldsymbol{S}, w)$

Algorithm 2 Fast-Caratheodory($\boldsymbol{P}, u, K, M$)

Input:
A set of points $\boldsymbol{P} = \{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_N\}$ in $\mathbb{R}^L$
A weighting function $u : \boldsymbol{P} \to [0, \infty]$
Number of clusters $K >= L + 2$
Target output size M
Output:
A subset of input points and weighting function $(\boldsymbol{C}, w)$ s.t. $\sum (u(\boldsymbol{p}_i) \cdot \boldsymbol{p}_i) = \sum (w(\boldsymbol{c}_i) \cdot \boldsymbol{c}_i)$

if $|N \leq M|$ **then**
  return $(\boldsymbol{P}, u)$
$\{\boldsymbol{P}_1, \cdots, \boldsymbol{P}_K\}$ = equally divided disjoint subsets of $\boldsymbol{P}$
for $i \in \{1, \cdots, K\}$ **do**
  $\boldsymbol{\mu}_i \leftarrow \frac{\sum_{\boldsymbol{p} \in \boldsymbol{P}_i} u(\boldsymbol{p}) \cdot \boldsymbol{p}}{\sum_{\boldsymbol{p} \in \boldsymbol{P}_i} u(\boldsymbol{p})}$ ▷ Compute mean of the cluster
  $u'(\boldsymbol{\mu}_i) \leftarrow \sum_{\boldsymbol{p} \in \boldsymbol{P}_i} u(\boldsymbol{p})$
$(\tilde{\boldsymbol{\mu}}, \tilde{w}) \leftarrow$ CARATHEODORY($\{\boldsymbol{\mu}_1, \cdots, \boldsymbol{\mu}_K\}, u'$)
$\boldsymbol{C} \leftarrow \cup_{\boldsymbol{\mu}_i \in \tilde{\boldsymbol{\mu}}} \boldsymbol{P}_i$ ▷ Recover points from selected clusters
for $\boldsymbol{\mu}_i \in \tilde{\boldsymbol{\mu}}$ and $\boldsymbol{p} \in \boldsymbol{P}_i$ **do**
  $w(\boldsymbol{p}) \leftarrow \frac{\tilde{w}(\boldsymbol{\mu}_i) u(\boldsymbol{p})}{\sum_{\boldsymbol{p} \in \boldsymbol{P}_i} u(\boldsymbol{p})}$
return FAST-CARATHEODORY($\boldsymbol{C}, w, K, M$)

Algorithms 1, 2, and 3 describe the proposed coreset extraction algorithm extending the algorithms in [13], [5]. Due to space limitations, we provide only a brief explanation of the algorithms and highlight the modifications we made. For details of the original algorithms, we refer the reader to [5]. The code for the proposed algorithm is available online so that the reader can confirm the details [1].

Algorithm 1 is Caratheodory's classic algorithm for finding a minimum exact coreset from a weighted input set [13]. This algorithm finds and eliminates redundant data points one by one. Because the time required to execute the algorithm is $O(N^2L^2)$, it cannot be directly applied to large datasets. To reduce the time complexity, Algorithm 2 divides the input set into K disjoint subsets (e.g., $K = 64$) and computes the means of the clusters. It then applies Algorithm 1 to extract a

[1] https://github.com/koide3/caratheodory2

Algorithm 3 Fast-Caratheodory-Quadratic($\boldsymbol{e}, \boldsymbol{J}, M$)

Input:
A residual vector $\boldsymbol{e} \in \mathbb{R}^N = [e_1, \cdots, e_N]^T$
Jacobian matrix $\boldsymbol{J} \in \mathbb{R}^{N \times D} = [\boldsymbol{a}_1, \cdots, \boldsymbol{a}_N]^T$
Target output size M
Output:
A function g to select a subset of $\boldsymbol{e}$
Weights $\boldsymbol{w}$ for the selected subset
g and $\boldsymbol{w}$ satisfy $\tilde{\boldsymbol{J}}^T \boldsymbol{W} \tilde{\boldsymbol{J}} = \boldsymbol{J}^T \boldsymbol{J}$, $\tilde{\boldsymbol{J}}^T \boldsymbol{W} \tilde{\boldsymbol{e}} = \boldsymbol{J}^T \boldsymbol{e}$, $\tilde{\boldsymbol{e}}^T \boldsymbol{W} \tilde{\boldsymbol{e}} = \boldsymbol{e}^T \boldsymbol{e}$, where $\tilde{\boldsymbol{e}} = g(\boldsymbol{e}), \tilde{\boldsymbol{J}} = g(\boldsymbol{J})$, and $\boldsymbol{W} = \text{diag}(\boldsymbol{w})$

for $i \in \{1, \cdots, N\}$ **do**
  $\boldsymbol{h}_i \leftarrow$ upper triangular elements of $\boldsymbol{a}_i^T \boldsymbol{a}_i$
  $\boldsymbol{b}_i \leftarrow \boldsymbol{a}_i^T e_i$
  $c_i \leftarrow e_i^2$
  $\boldsymbol{p}_i \leftarrow [\boldsymbol{h}_i^T, \boldsymbol{b}_i^T, c_i]^T$
  $u(\boldsymbol{p}_i) \leftarrow 1/N$
$\boldsymbol{P} \leftarrow \{\boldsymbol{p}_1, \cdots, \boldsymbol{p}_N\}$
$(\boldsymbol{C}, w) \leftarrow$ FAST-CARATHEODORY($\boldsymbol{P}, u, K, M$)
$g \leftarrow$ A function to select entries of $\boldsymbol{e}$ exist in $\boldsymbol{C}$
$\boldsymbol{w} \leftarrow [w(c_i) | c_i \in g(\boldsymbol{e})]$
return $(g, \boldsymbol{w})$

Caratheodory set of the cluster means and eliminates clusters that are not selected to be in the Caratheodory set. It recovers the set of input data from the clusters in the extracted Caratheodory set and repeats this process until the number of data points becomes less than or equal to a target output size. The clustering strategy significantly improves the time complexity and can find a Caratheodory set in time linear to the number of input data points $O(NL)$. Algorithm 3 takes as input a residual vector $\boldsymbol{e}$ and its Jacobian matrix $\boldsymbol{J}$, computes *flattened* vectors of $\boldsymbol{H}, \boldsymbol{b}, c$, and passes them to Algorithm 2 to obtain a weighted subset of input residuals to reconstruct the original quadratic function.

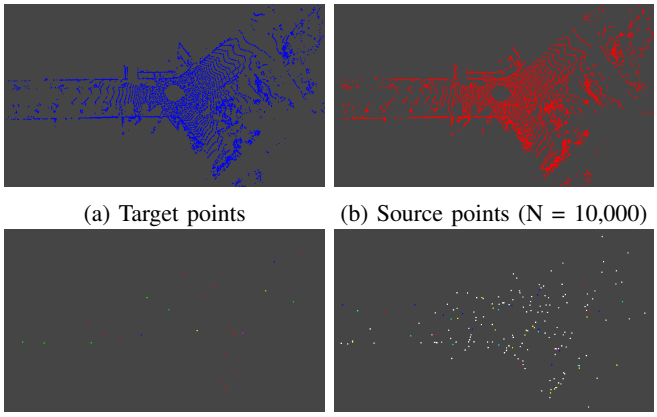
The modifications we made to fit the algorithms in [5] to our problem can be summarized as follows:

1) While the original algorithm selects only a minimum number of data points to compose a Caratheodory set, we modified Algorithms 1 and 2 so that they can change the target number of data points M to control the trade-off between approximation accuracy and computation speed. By increasing M, the algorithm extracts more data that are redundant to compose a Caratheodory set but help approximate the nonlinearity of the original registration error function. Note that the proposed algorithm does not extract exactly M points but rather extracts points in the range $[\max(M - K, 29), M]$. While we can easily modify Algorithm 2 to select exactly M points, we avoided doing so to save processing time.
2) In the original implementation of Algorithm 1, most of the computation time was taken by SVD to find $v \neq 0$ such that $\boldsymbol{A}v = 0$. To improve the processing speed, we use an efficient LU decomposition method to find the nullspace of $\boldsymbol{A}$ instead of SVD.
3) While [5] recovers only the Hessian matrix $\boldsymbol{H}$, we recover $\boldsymbol{b}$ and c in addition to $\boldsymbol{H}$ to approximate the quadratic function in Eq. 3. Although this modification increases the dimension of the input vectors from D^2

Algorithm 4 Exact-Downsampling($\mathcal{P}_i, \mathcal{P}_j, \breve{\boldsymbol{T}}_i, \breve{\boldsymbol{T}}_j, f^{REG}, M$)

Input:
Point clouds $\mathcal{P}_i$ and $\mathcal{P}_j$
Poses $\breve{\boldsymbol{T}}_i$ and $\breve{\boldsymbol{T}}_j$ to evaluate errors between $\mathcal{P}_i$ and $\mathcal{P}_j$
Registration error function f^{REG}
Target output size M
Output:
A function g to select a subset of residuals of input points
A weight vector $\boldsymbol{w}$

Shuffle points in $\mathcal{P}_j$
$\boldsymbol{e}^T\boldsymbol{e} \leftarrow f^{REG}(\mathcal{P}_i, \mathcal{P}_j, \boldsymbol{x}_i, \boldsymbol{x}_j)$
$\boldsymbol{J} \leftarrow \partial \boldsymbol{e}/\partial \boldsymbol{T}_j$
$(g, \boldsymbol{w}) \leftarrow$ FAST-CARATHEODORY-QUADRATIC$(\boldsymbol{e}, \boldsymbol{J}, M)$
return $(g, \boldsymbol{w})$



(a) Target points (b) Source points (N = 10,000)

(c) Sampled residuals (M = 29) (d) Sampled residuals (M = 512)

Fig. 3: Example of downsampling results. Source points (b) and sampled points with different target numbers of residuals (M=29 (c), and M=512 (d)) all yield the same quadratic registration error function for the target points at the evaluation point. The colors of the sampled points represent the selected axes of residuals (R=X, G=Y, B=Z).

to $D^2 + D + 1$, we can omit the non-upper-triangular elements of $\boldsymbol{H}$, since it is symmetric. As a result, for a function with six-dimensional input, the input vector dimension becomes $21+6+1 = 28$, and we need only $28 + 1 = 29$ residuals to exactly recover the original quadratic function at a minimum.

As we will show in Sec. IV-A, modifications 2 and 3 reduce processing time by a factor of 10. Note that because the registration error function is symmetric for $\boldsymbol{T}_i$ and $\boldsymbol{T}_j$, the subset extracted for $\boldsymbol{T}_j$ exactly reconstructs $\boldsymbol{H}_{ii}, \boldsymbol{H}_{ij}$, and $\boldsymbol{b}_i$ as well.

D. Point Cloud Downsampling

The proposed exact point cloud downsampling algorithm is summarized as in Algorithm 4. Because Algorithm 2 reduces the dataset size by dropping contiguous items, we first shuffle the input points to avoid bias in the sampling result. We then evaluate the residuals $\boldsymbol{e}$ and Jacobian $\boldsymbol{J}$ of the registration error function at $\breve{\boldsymbol{T}}_i$ and $\breve{\boldsymbol{T}}_j$. We input $\boldsymbol{e}$ and $\boldsymbol{J}$ to Algorithm 3 and obtain a subset of the residuals and weights.

During Gauss-Newton optimization, we iteratively re-linearize the error function f^{REG} with the extracted subset to obtain a quadratic error function.

Fig. 3 (c) and (d) show examples of downsampling results. The colors of the points indicate the selected axes of the sampled points (R=X, G=Y, B=Z). Because the proposed algorithm works on a per-residual basis, it happens that only one or two axes of a point are selected (Fig. 3 (c)). Because the algorithm tends to drop contiguous residuals, most of the selected points have all selected axes when the number of target residuals is sufficiently large (Fig. 3 (d)). Both the sampled points with $M = 29$ and $M = 512$ shown in Fig. 3 (c) and (d) exactly recover the quadratic registration error function between the original input point clouds shown in Fig. 3 (a) and (b).

IV. EXPERIMENTS

A. Numerical Validation

We first verified the proposed algorithm through numerical validation. We began by randomly generating $\boldsymbol{J} \in \mathbb{R}^{N\times 6}$ and $\boldsymbol{e} \in \mathbb{R}^N$ ($N = 30{,}000 \approx 10{,}000$ points) and applying the proposed downsampling algorithm to obtain a weighted subset of the residuals $(g, \boldsymbol{w})$. We then calculated the quadratic function approximation error as $\max(\|\boldsymbol{J}^T\boldsymbol{J} - \tilde{\boldsymbol{J}}^T\boldsymbol{W}\tilde{\boldsymbol{J}}\|, \|\boldsymbol{J}^T\boldsymbol{e} - \tilde{\boldsymbol{J}}^T\boldsymbol{W}\tilde{\boldsymbol{e}}\|, \boldsymbol{e}^T\boldsymbol{e} - \tilde{\boldsymbol{e}}^T\boldsymbol{W}\tilde{\boldsymbol{e}})$, where $\tilde{\boldsymbol{J}} = g(\boldsymbol{J}), \tilde{\boldsymbol{e}} = g(\boldsymbol{e})$, and $\boldsymbol{W} = \mathrm{diag}(\boldsymbol{w})$. We repeated this numerical validation 100 times [2].

To evaluate the speed gain of the modifications proposed in Sec. III-C, we ran the algorithm with three configurations:

1) Naive implementation of [5] with the expanded input dimension $D^2 + D + 1 = 43$
2) Naive implementation of [5] with the compact input dimension without the non-upper-triangular elements of the Hessian $((D + 1) \times (D/2) + D + 1 = 28)$
3) Using the compact input dimension (28) and LU decomposition instead of SVD

All three configurations produced approximation errors below 10^{-10} for all trials, showing the validity of the proposed algorithm. Configuration 1 took approximately 125 ms for each trial. With configuration 2, the processing time decreased to 76 ms by using the compact input representation. Finally, with configuration 3, the processing time further decreased to 7 ms, or roughly 5.6% of the time for configuration 1.

We then evaluated the change in processing time for the proposed algorithm when the target output size varies from 29 to 1024. Again, all the selected subsets showed reconstruction errors below 10^{-10} for all the settings. Table I summarizes the average processing times with several target numbers of residuals M. As shown, when the target number of residuals was increased, the processing time also grew, albeit gradually. While all the settings exactly reconstructed the original quadratic error function at the evaluation point, as we will show in the following Sec. IV-B, a larger number

[2]The code for reproducing this experiment is available at the GitHub repository.

TABLE I: Processing time for the exact sampling algorithm

Number of residuals (M)	29	64	128	256	512	1024
Processing time [ms]	6.87	13.39	19.34	25.03	29.99	34.19

TABLE II: Hessian approximation accuracy evaluation result

Method	Num. of points / residuals	Normalized KLD
Random sampling	10 points ≈ 30 residuals	0.996 ± 0.026
	64 points ≈ 192 residuals	0.437 ± 0.213
	256 points ≈ 768 residuals	0.107 ± 0.062
	1024 points ≈ 3072 residuals	0.024 ± 0.013
	10149 points (original points)	0.000 ± 0.000
Exact sampling	29 - 3072 residuals	0.000 ± 0.000

of residuals leads to better approximation accuracy under pose displacements. Thus, setting the target number of residuals parameter involves a trade-off between approximation accuracy and processing speed.

B. Nonlinearity Approximation Accuracy Analysis

We next compared the approximation accuracy of the proposed exact sampling with that of the commonly used random sampling approach. Here, we took two consecutive frames in the sequence 00 of the KITTI dataset [17], as shown in Fig. 3. We aligned the frames by using standard GICP scan matching and then sampled subsets of the input points and residuals using random sampling and the proposed exact sampling method while changing the target output size.

We computed the Hessian matrices of f^{REG} by using the sampled subsets and compared them with that computed using the original input points. To compare Hessian matrices, we used the normalized KLD metric defined as follows:

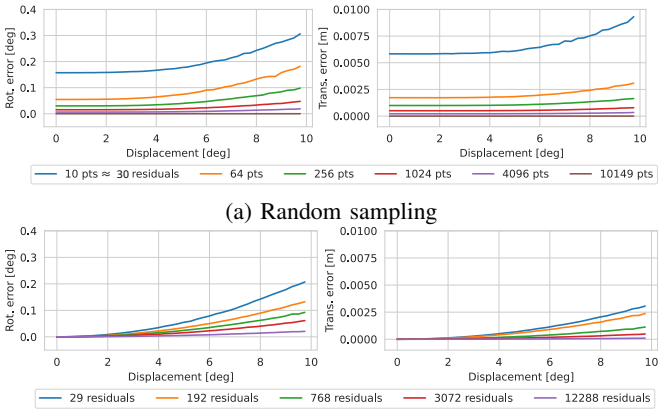
$$\mathrm{KLD}(\boldsymbol{H}, \tilde{\boldsymbol{H}}) = \frac{1}{2}\left(\log(\frac{|\boldsymbol{H}|}{|\tilde{\boldsymbol{H}}|}) + \mathrm{tr}(\boldsymbol{H}^{-1}\tilde{\boldsymbol{H}})\right), \quad (11)$$

$$\mathrm{NormedKLD}(\boldsymbol{H}, \tilde{\boldsymbol{H}}) = 1 - \exp(-\mathrm{KLD}(\boldsymbol{H}, \tilde{\boldsymbol{H}})). \quad (12)$$

We repeated the evaluation 100 times while shuffling the order of the points.

Table II summarizes the normalized KLDs for the random and exact sampling results. The random sampling results showed large errors when the number of points was small. In particular, when the number of points was set to 10, we observed that the Hessian matrix computed from the sampled points sometimes degenerated. As the number of points increased, the approximation accuracy of the random sampling results improved. When and only when the target number of points was set to the same number as the number of input points, the approximation error became zero. This result suggests that random sampling would not be a good choice for sampling point cloud registration errors, since it substantially changes the objective function shape when the number of sampling points is small. On the other hand, as verified in Sec. IV-A, the proposed algorithm exactly recovered the original Hessian matrix with a small number of residuals and thus the normalized KLD was always zero at the evaluation point.

We then evaluated the changes in function approximation accuracy of the two sampling methods under pose displacements. We applied random rotation noise and re-evaluated the registration error function using the sampled subsets to obtain $\boldsymbol{H}$ and $\boldsymbol{b}$. We then computed the displacement vector claimed by the linearized system ($\Delta\boldsymbol{x} = \boldsymbol{H}^{-1}\boldsymbol{b}$) and compared it with that computed using the original points.



(a) Random sampling

(b) Exact sampling

Fig. 4: Errors of the displacement vector $\Delta\boldsymbol{x} = \boldsymbol{H}^{-1}\boldsymbol{b}$ of random and exact sampling results under rotation noise.

Fig. 4 summarizes the displacement vector errors. Because the residuals sampled by the proposed algorithm exactly recovered the original quadratic error function, the proposed algorithm showed zero displacement vector errors with all the settings when the noise was zero. As the rotation noise became larger and the current estimate separated further from the evaluation point, the displacement vector errors became gradually worse due to the nonlinearity of the registration error function. However, we can see that the results of the proposed exact sampling show substantially smaller errors compared to the random sampling results with equivalent output size settings.

C. Application to Global Trajectory Optimization

We applied the proposed exact downsampling to global trajectory optimization on the longest four sequences (00, 02, 05, 08) in the KITTI dataset [17] in order to compare its trajectory estimation accuracy and associated processing time with those of the conventional pose graph optimization.

Evaluation protocol: To obtain an initial guess of sensor poses, we applied GICP scan matching between consecutive frames (i.e., odometry estimation). Meanwhile, we ran Scan-Context [18] to detect loop candidates and validated them via GICP scan matching. We constructed a pose graph with relative pose constraints between the consecutive frames and loop pairs and performed optimization using the Levenberg-Marquardt optimizer in GTSAM [19]. The objective function of pose graph optimization is defined as follows:

$$f^{PGO}(\mathcal{X}) = \sum_{(\boldsymbol{T}_i, \boldsymbol{T}_j, \hat{\boldsymbol{T}}_{ij})} \|f^{RP}(\boldsymbol{T}_i, \boldsymbol{T}_j, \hat{\boldsymbol{T}}_{ij})\|^2, \quad (13)$$

$$f^{RP}(\boldsymbol{T}_i, \boldsymbol{T}_j, \hat{\boldsymbol{T}}_{ij}) = \rho\left(\boldsymbol{d}_{ij}^T \boldsymbol{H}_{ij} \boldsymbol{d}_{ij}\right), \quad (14)$$

$$\boldsymbol{d}_{ij} = \log\left(\hat{\boldsymbol{T}}_{ij}^{-1} \boldsymbol{T}_i^{-1} \boldsymbol{T}_j\right), \quad (15)$$

TABLE III: ATEs [m] of pose graph optimization and global registration error minimization results

Method	Sequence			
	00	02	05	08
Pose graph optimization (Identity)	1.6257 ± 0.7529	23.9856 ± 8.2752	1.5149 ± 0.6760	9.3636 ± 3.2890
Pose graph optimization (Hessian)	1.3777 ± 0.6051	9.3406 ± 3.1490	1.6249 ± 0.8489	5.0532 ± 2.5991
Pose graph optimization (Hessian + Dense)	1.1846 ± 0.5625	9.3393 ± 3.1486	1.6240 ± 0.8488	5.0520 ± 2.5993
Registration error minimization + Exact sampling (M=29)	0.9553 ± 0.4650	8.9679 ± 3.0856	0.2917 ± 0.1060	4.4394 ± 2.5294

where $\mathcal{X}$ is the set of sensor poses, $\hat{\boldsymbol{T}}_{ij}$ is the relative pose measurement given by scan matching, and ρ is Cauchy's robust kernel. For $\boldsymbol{H}_{ij}$, we used two settings: 1) setting the identity matrix $\boldsymbol{I}_{6\times6}$ to $\boldsymbol{H}_{ij}$, and 2) setting the Hessian matrix at the last iteration of GICP scan matching to $\boldsymbol{H}_{ij}$.

After performing pose graph optimization, we found all overlapping point cloud pairs by voxel-based overlap assessment [3] and constructed a densely connected graph with factors between all detected overlapping pairs, as shown in Fig. 2. With this dense graph structure, we again optimized the sensor poses using pose graph optimization and global registration error minimization. The objective function for global registration error minimization is defined as

$$f^{GR}(\mathcal{X}) = \sum_{(\boldsymbol{T}_i,\boldsymbol{T}_j)\in\mathcal{X}^O} \|f^{REG}(\mathcal{P}_i,\mathcal{P}_j,\boldsymbol{T}_i,\boldsymbol{T}_j)\|^2, \qquad (16)$$

where $\mathcal{X}^O$ is the set of point cloud pairs with an overlap.

For pose graph optimization, we applied GICP scan matching for all frame pairs and created relative pose constraints from the scan matching results. For global registration error minimization, we applied the exact sampling for all frame pairs with the target number of residuals $M = 29$ and optimized the sensor poses by minimizing the global registration errors computed with the sampled residuals. For the sequence 00, we also ran the same algorithm with M=256 and M=1024 and the original input points to determine the effect that changing the number of samples has on processing time and estimation accuracy.

Comparison with pose graph optimization: Table III summarizes the absolute trajectory errors (ATEs) [20] for pose graph optimization and global registration error minimization. We used the *evo* toolkit [3] to measure the ATEs.

For most of the sequences, pose graph optimization with identity Hessian matrices produced much worse results than when Hessian matrices composed of the scan matching results were used. This suggests the importance of appropriately modeling the relative pose constraints for better estimation results.

Although pose graph optimization on the dense factor graph showed better ATEs than those for the sparse graph, the accuracy gain was not very significant. Fig. 5 shows the dense pose graph optimization graph. The colors of the factors indicate the magnitudes of the relative pose errors. We can see that the factors between frames at a distance tend to show large errors because the GICP scan matching failed on small overlapping frames. This suggests that increasing the number of factors does not always improve the estimation accuracy of pose graph optimization.

[3] https://github.com/MichaelGrupp/evo

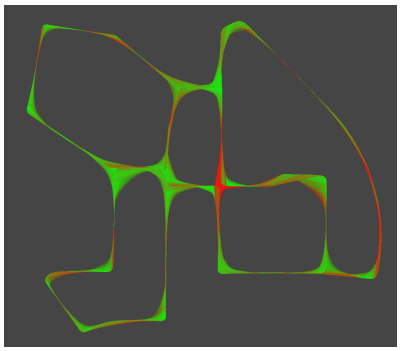


Fig. 5: Dense pose graph optimization result. Color indicates the magnitude of the relative pose errors of each factor (Green: small error, Red: large error).

TABLE IV: ATEs of global registration error minimization with different sampling settings for the sequence 00

Method	ATE [m]	Optimization time [h]
Exact sampling (M=29)	0.9553 ± 0.4650	1.67
Exact sampling (M=256)	0.9549 ± 0.4646	1.88
Exact sampling (M=1024)	0.9549 ± 0.4647	2.36
Original points (10,000 points)	0.9549 ± 0.4647	13.21

The global registration error minimization approach greatly improved the ATEs of all the sequences compared to those of pose graph optimization. Because this approach directly minimizes registration errors over the entire map while avoiding the Gaussian approximation of the relative pose constraint, it can accurately constrain the relative pose between frames with small overlap.

Effect of the exact sampling: Table IV summarizes the ATEs of global registration error minimization with different sampling settings. While the ATEs improved as the number of residuals increased, the smallest number of residuals M=29 showed an ATE very close to that of the original points. Because we started the optimization from a sufficiently good initial guess, the approximation errors were sufficiently small for the small displacements with the minimum number of residuals (M=29) as shown in Sec. IV-B. We consider that, if the initial guess had larger errors, the results from the smallest sampling setting might deteriorate and the other sampling settings with more residual samples would show better results.

Memory consumption: Table V summarizes the memory consumption of global registration error minimization for each sampling setting. For the case in which the original

TABLE V: Memory consumption

Sampling method	Num. residuals	Memory consumption [GB] Correspondences	Sampled residuals
Exact sampling	29	0.11	0.14
	256	0.43	1.15
	1024	1.52	4.87
All residuals	∼30,000	25.13	0.00

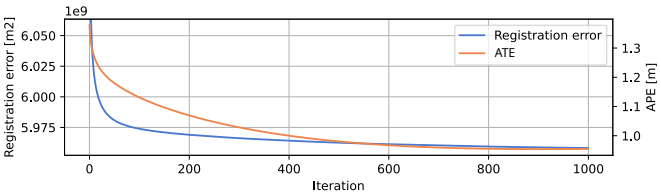


Fig. 6: Global registration error and ATE change during global registration error minimization.

points are used, where we need to remember the point correspondences for each factor, approximately 25 GB of memory was required. With exact sampling, where, in addition to remembering the point correspondences, we need to remember the selected residual samples, total memory consumption was substantially smaller. For the setting M=29, only 0.11 GB was needed for the point correspondences and only 0.14 GB was needed for the sampled residuals.

Processing time: Fig. 6 shows the decreasing pattern of both the global registration error and the ATE during optimization. With this large-scale factor graph optimization, more than several hundred iterations were needed before convergence. Fig. 7 shows a breakdown of the optimization times for the different sampling settings. With the original points, the optimization took approximately 13.2 hours on a CPU with 128 threads. We can see that the linearization and cost evaluation of the registration error function consumed a majority of the optimization time. Although the proposed exact sampling introduced additional processing time for downsampling of approximately 130 s, it drastically decreased the processing time for linearization and cost evaluation and reduced the total optimization time (1.67 hours with M=29, 2.36 hours with M=1024). Note that while we used the multi-threaded MULTIFRONTAL_CHOLESKY linear solver in GTSAM, we observed that it did not fully utilize all the available CPU cores. We consider that the total optimization time can be further improved by using a linear solver dedicated to multi-threading or GPU computing.

V. Conclusion

This work proposed a point cloud downsampling algorithm based on an algorithm for efficient exact coreset extraction. Experimental results showed that this approach drastically reduces the linearization cost of the point cloud registration error function without sacrificing accuracy. We plan to apply the proposed algorithm to extremely large city- or nation-scale mapping problems and other types of data (e.g., dense full BA for visual SLAM).

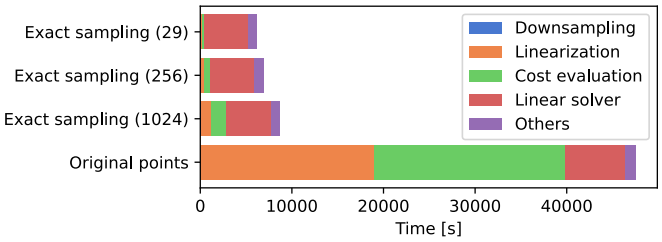


Fig. 7: Optimization time breakdown.

References

[1] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous robots*, vol. 4, pp. 333–349, 1997.

[2] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[3] 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.

[4] 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.

[5] 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.

[6] J. M. Phillips, "Coresets and sketches," in *Handbook of discrete and computational geometry*. Chapman and Hall/CRC, 2017, pp. 1269–1288.

[7] D. Landry, F. Pomerleau, and P. Giguere, "CELLO-3d: Estimating the covariance of ICP in the real world," in *IEEE International Conference on Robotics and Automation*. IEEE, May 2019, pp. 8190–8196.

[8] 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.

[9] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus, "LIO-SAM: Tightly-coupled lidar inertial odometry via smoothing and mapping," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Oct. 2020, pp. 5135–5142.

[10] D. Wisth, M. Camurri, and M. Fallon, "VILENS: Visual, inertial, lidar, and leg odometry for all-terrain legged robots," *IEEE Transactions on Robotics*, vol. 39, no. 1, pp. 309–326, Feb. 2023.

[11] 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.

[12] 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.

[13] W. Cook and R. Webster, "Caratheodory's theorem," *Canadian Mathematical Bulletin*, vol. 15, no. 2, pp. 293–293, 1972.

[14] D. Chetverikov, D. Svirko, D. Stepanov, and P. Krsek, "The trimmed iterative closest point algorithm," in *Object recognition supported by user interaction for service robots*. IEEE, 2002, pp. 545–548.

[15] 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.

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

[17] 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.

[18] G. Kim and A. Kim, "Scan context: Egocentric spatial descriptor for place recognition within 3d point cloud map," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, Oct. 2018, pp. 4802–4809.

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

[20] 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.