VoxelMap++: Mergeable Voxel Mapping Method for Online LiDAR(-Inertial) Odometry

Chang Wu, *Member, IEEE*, Yuan You, Yifei Yuan, Xiaotong Kong, Ying Zhang, Qiyan Li, and Kaiyong Zhao

***Abstract*—This letter presents VoxelMap++: a voxel mapping method with plane merging which can effectively improve the accuracy and efficiency of LiDAR(-inertial) based simultaneous localization and mapping (SLAM). This map is a collection of voxels that contains one plane feature with 3DOF representation and corresponding covariance estimation. Considering map will contain a large number of coplanar features (kid planes), these kid planes' can be regarded as the measurements with covariance of a larger plane (father plane). Thus, we have designed a plane merging module based on the union-find. This merging module is capable of distinguishing co-plane relationship within various voxels, then merge these kid planes to estimate the father plane by minimizing the trace of covariance. After merging, the father plane exhibits more accurate compare to kids plane, with decreasing of uncertainty, which improve the accuracy of LiDAR(-inertial) odometry. Experiments on different environments demonstrate the superior of VoxelMap++ compared with other state-of-the-art methods (see our attached video). Our implementation is open-sourced on GitHub which is applicable for both non-repetitive scanning LiDARs and traditional scanning LiDAR.**

***Index Terms*—Mapping, union-find, localization, simultaneous localization and mapping (SLAM).**

I. Introduction

Recently, with the rapid development of 3D LiDAR technology, LiDAR(-inertial) odometry has been wildly used in various applications such as autonomous vehicles [3], [4], UAVs [5] and mobile robot [7] because of the robustness and accuracy. The representation of the historical map is a crucial component in LiDAR(-inertial) odometry. Most state-of-the-art methods require fast and accurate registration, as well as efficient incremental updates during robot motion. The most commonly mapping method in LiDAR-based simultaneous localization and mapping (SLAM) is KD-Tree point cloud due to its ease of use, e.g. LOAM [8], Lego-LOAM [9], LIO-SAM [10], LIO-Mapping [11], LINS [12], LILI-OM [13], FAST-LIO [1], Point-LIO [5]. However, in the process of point cloud registration, it is inevitable to frequently search for K nearest neighbor points to fit the plane which is a time consume task for KD-Tree. The time complexity of the nearest neighbor search in KD-Tree is logarithmic, which poses a significant challenge on real-time performance. To overcome the drawback of map based on point clouds, FAST-LIO2 [16] develops an incremental kd-tree structure to organize the point cloud map efficiently; MINS [17] maintains a plane patch point cloud to decrease the numbers of point in map, which contain the center point p and Hesse normal n of plane. With the efforts of these researchers, point cloud maps have gradually become practical, but this is accompanied by a series of complex map management methods. However, preserving the uncertainty of point clouds is still impossible. The covariance for each point necessitates representation by a 3×3 matrix. However, this leads to an impractical fourfold increase in memory usage.

Apart from KD-Tree-based point cloud mapping, recent advancements include voxel mapping methods using spatial hashing, e.g., LIO-Livox, Faster-LIO [14], BALM [15], and VoxelMap [2]. Constant time complexity in hash querying ensures real-time system performance. Notably, VoxelMap introduces an efficient, probabilistic adaptive voxel mapping for LiDAR odometry, outperforming other methods in robustness and efficiency. In LiDAR raw processing, similar to FAST-LIO, VoxelMap omits plane/edge feature extraction. Post-motion compensation, all points are used for scan-matching to estimate the robot state using a tightly-coupled iterated Kalman filter. For real-time performance, VoxelMap integrates a caching mechanism for voxel plane parameters, eliminating redundant operations and enhancing efficiency. In terms of accuracy and robustness, VoxelMap excels with a covariance estimation method derived from the LiDAR sensor model, improving both robustness and positioning accuracy by addressing unreliable point-to-plane registration. Consequently, VoxelMap, an extension of FAST-LIO, is widely adopted in resource-limited embedded platforms.

However, VoxelMap is still undergoing refinement to achieve an optimal level of perfection. The utilization of a point and normal vector in a redundant 6DOF plane representation, will lead to unnecessary resource wastage. Additionally, VoxelMap ignores the interrelationship among neighboring voxels, resulting in the entire plane (such as the ground) on the map being divided into many subplanes. Moreover, the point clouds used to fitting one single plane are limited which makes the estimation

Manuscript received 4 August 2023; accepted 2 November 2023. Date of publication 17 November 2023; date of current version 28 November 2023. This letter was recommended for publication by Associate Editor J. Zhang and Editor J. Civera upon evaluation of the reviewers' comments. This work was supported by the Science and Technology Project Fund of Sichuan Province, China, under Grant 2018sz0364. *(Chang Wu and Yuan You are co-first authors.) (Corresponding author: Chang Wu.)*

Chang Wu, Yifei Yuan, Xiaotong Kong, Ying Zhang, and Qiyan Li are with the Institute of Information and Communication Engineering, University of Electronic Science and Technology of China (UESTC), Chengdu 611731, China (e-mail: changwu@uestc.edu.cn; yuanyf1998@126.com; m13086663208@163.com; 15847693106@163.com; qyan.lieestd@gmail.com).

Yuan You and Kaiyong Zhao are with the XGRIDS, Shenzhen 518051, China (e-mail: yuan.you@xgrids.com; kyzhao@xgrids.com).

This letter has supplementary downloadable material available at https://doi.org/10.1109/LRA.2023.3333736, provided by the authors.

Digital Object Identifier 10.1109/LRA.2023.3333736

https://www.bilibili.com/video/BV16h4y1r7Cm/

https://github.com/uestc-icsp/VoxelMapPlus_Public

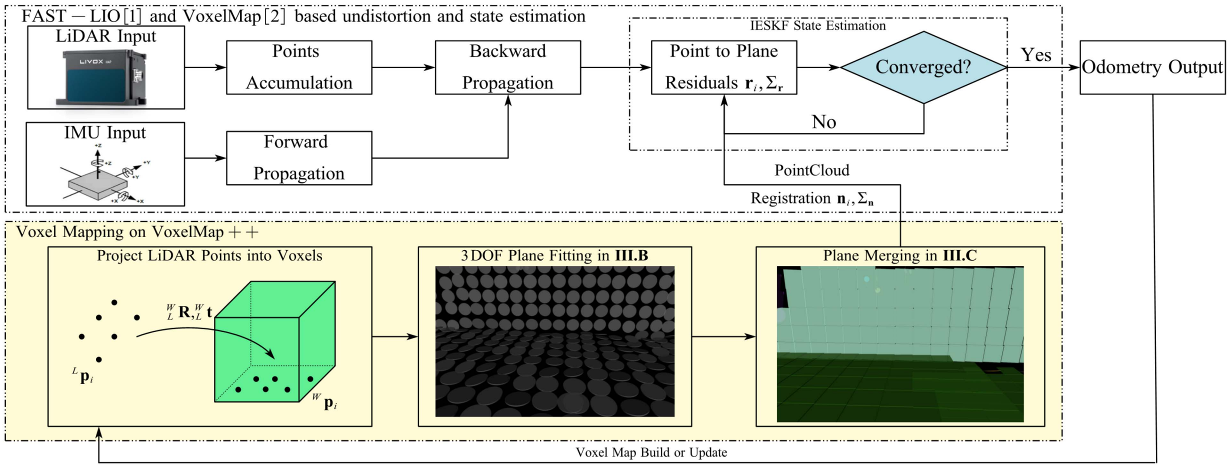

FAST − LIO [1] and VoxelMap [2] based undistortion and state estimation
LiDAR Input
IMU Input
Points Accumulation
Backward Propagation
Forward Propagation
IESKF State Estimation
Point to Plane Residuals $\mathbf{r}_i, \Sigma_\mathbf{r}$
Converged?
Yes
No
Odometry Output
PointCloud Registration $\mathbf{n}_i, \Sigma_\mathbf{n}$
Voxel Mapping on VoxelMap ++
Project LiDAR Points into Voxels
$^W_L\mathbf{R}, ^W_L\mathbf{t}$
$^L\mathbf{p}_i$
$^W\mathbf{p}_i$
3DOF Plane Fitting in III.B
Plane Merging in III.C
Voxel Map Build or Update

Fig. 1. System overview of VoxelMap++, the main contribution of this article is the mapping block denoted in yellow.

of plane not accurate enough. Simply increase the number of points used in the fitting plane can improve the accuracy. But plenty of memory and computing resources is used in the map update which is not worth the loss.

If the coplanar relationships can be effectively distinguished, then merging small planes within the voxels into a larger plane will greatly improve the accuracy of the plane, while also enhancing the positioning accuracy of the LIDAR(-inertial) odometry. At the same time, thousands of memory resources of small plane can be freed after merged. Thus, to further improve the peformance of LiDAR(-inertial) odometry, this letter proposes a novel online mergeable voxel mapping method with 3DOF plane representation termed VoxelMap++. Concretely, the contributions of this letter include:

- We enhance the plane fitting and covariance estimation method from 6DOF through least squares estimation, focusing on engineering implementation rather than precision. This enhancement significantly improves the efficiency of covariance estimation and reduces memory usage.
- We propose a novel online voxel merging method by using union-find. Each coplanar feature (kid plane) in the voxel will be considered as the measurements of a large plane (father plane). The merging module not only enhances the accuracy of plane fitting but also reduces uncertainty in the overall map, thereby improving state estimation accuracy.
- We compared VoxelMap++ with other state-of-art algorithms in both public structured urban datasets and other private challenging scenarios to demonstrate the superiority of the algorithm in accuracy and efficiency.
- We make the VoxelMap++ adapt to different kinds of LiDARs (multi-spinning LiDARs and non-conventional solid-state LiDARs) and open-sourced on our GitHub for sharing our findings and making contributions to the community.

II. System Overview

The pipeline of VoxelMap++ is illustrated in Fig. 1. The LiDAR raw points preprocessing method and state estimation method based on iterated error-state Kalman filter (IESKF) are similar to FAST-LIO and VoxelMap.

If an IMU is present, the discrete kinematic model will be used in Forward Propagation to predict the state vector and LiDAR raw points will be undistorted by compensating the relative motion in Backward Propagation which is described in FAST-LIO [1] and FAST-LIO2 [16]. Without IMU, the module about Forward Propagation and Backward Propagation can simplify to the motion compensation based on linear interpolation mentioned in Loam-Livox [23]. Considering motion compensation has been thoroughly studied, we recommend that readers find the details in the references [1].

After LiDAR raw processing, point-to-plane registration is used to estimate the system state which is illustrated in Section III-C. In registration, spatial hash is used to query the child nodes of union-find first. Then, the father node is found through the child nodes by using the pointer. Finally, the 3DOF plane representation of the father node is used for registration which is more accurate than child node because of plane merging in Section III-B.

After state estimation, each point in the new scan will be projected into the corresponding voxel, then construct or update the voxel map which is organized by a Hash table (key is voxel id and value is plane fitting results $\mathcal{P}$). The voxel map construction and update are introduced in Section III-B2. These new points will be incrementally used for 3DOF plane fitting and covariance estimation which will be introduced in Section III-A.

Then, the converged plane will be used for plane merging which will be introduced in Section III-B3. In this module, kid plane $\mathcal{P}^k$ in the voxels will be merged into father plane $\mathcal{P}^f$ based on union-find. Meanwhile, the plane estimation result of $\mathcal{P}^f$ will be more accurate compared with $\mathcal{P}^k$ which will improve the accuracy of LiDAR(-inertial) odometry.

III. Methodology

A. 3DOF Plane Fitting and Covariance Estimation

Like other implementations of LiDAR(-inertial) odometry [1], [2], [14], [17], we only use the plane features in the environment due to its vast availability. In each voxel, we maintain a 3DOF plane fitting results $\mathbf{n} = [a, b, d]^T$ and its covariance $\Sigma_\mathbf{n}$. In this subsection, we will illuminate how to fit the plane and estimate its covariance in 3DOF rather than other redundant representations.

Firstly, according to the analysis of the measurement noises for LiDAR in [18], we can calculate the covariance of each LiDAR point ${}^{L}\mathbf{p}_i$ in the local LiDAR frame (1), then use pose estimation result $({}^{W}_{L}\mathbf{R}, {}^{W}_{L}\mathbf{t})$ to transform it to the world frame (2). Thus, we can further estimate the LiDAR points covariance $\Sigma_{{}^{W}\mathbf{p}_i}$ in world frame (3). Detailed derivation of (1), (2), and (3) can be found in [2].

$$\mathbf{A}_i = [\boldsymbol{\omega}_i \quad -d_i\lfloor\boldsymbol{\omega}_i\rfloor_{\times}\mathbf{N}(\boldsymbol{\omega}_i)]$$

$$\Sigma_{{}^{L}\mathbf{p}_i} = \mathbf{A}_i \begin{bmatrix} \Sigma_{d_i} & \mathbf{0}_{1\times 2} \\ \mathbf{0}_{2\times 1} & \Sigma_{\boldsymbol{\omega}_i} \end{bmatrix} \mathbf{A}_i^T \tag{1}$$

$${}^{W}\mathbf{p}_i = {}^{W}_{L}\mathbf{R}\,{}^{L}\mathbf{p}_i + {}^{W}_{L}\mathbf{t} \tag{2}$$

$$\Sigma_{{}^{W}\mathbf{p}_i} = {}^{W}_{L}\mathbf{R}(\Sigma_{{}^{L}\mathbf{p}_i} + \lfloor{}^{L}\mathbf{p}_i\rfloor \Sigma_{{}^{W}_{L}\mathbf{R}} \lfloor{}^{L}\mathbf{p}_i\rfloor^{T}_{\times}){}^{W}_{L}\mathbf{R}^T + \Sigma_{{}^{W}_{L}\mathbf{t}} \tag{3}$$

Assuming a group of coplanar LiDAR points ${}^{W}\mathbf{p}_i, (i = 1, \ldots, N)$ with covariance $\Sigma_{{}^{W}\mathbf{p}_i}$ have been determined using (3). Then, we leverage the linear least-squares method to calculate the 3DOF representation of plane [17]. Considering a plane can be extracted from this point cloud parameterized as (4) by normalizing along the z-component (main-axis). However, this expression is singular when the z-component of the plane normal is close to zero. This issue can be easily resolved by calculating the projection of the point cloud to each coordinate axis, and the one with the smallest projection variance is the main axis.

$$ax + by + z + d = 0 \tag{4}$$

Since all ${}^{W}\mathbf{p}_i$ are coplanar satisfied (4), the least squares optimization function can be constructed as (5). After a series of identical deformations, a closed-form solution of $\mathbf{n}$ can be obtained (6).

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_n & y_n & 1 \end{bmatrix} \mathbf{n} = \begin{bmatrix} -z_1 \\ -z_2 \\ \vdots \\ -z_n \end{bmatrix} \tag{5}$$

$$\mathbf{n} = \frac{\mathbf{A}^*}{|\mathbf{A}|}\mathbf{e} \tag{6}$$

$\mathbf{A}^*$ is the adjugate matrix of $\mathbf{A}$, the expression of $\mathbf{A}$ and $\mathbf{e}$ is shown as (7).

$$\mathbf{A} = \begin{bmatrix} \sum x_i x_i & \sum x_i y_i & \sum x_i \\ \sum x_i y_i & \sum y_i y_i & \sum y_i \\ \sum x_i & \sum y_i & N \end{bmatrix}$$

$$\mathbf{e} = [-\sum x_i z_i \quad -\sum y_i z_i \quad -\sum z_i]^T \tag{7}$$

Based on the closed-form solution of $\mathbf{n}$, the uncertainty of the plane is hence

$$\boldsymbol{\Sigma}_{\mathbf{n}} = \sum_{i}^{N} \frac{\partial \mathbf{n}}{\partial {}^{W}\mathbf{p}_i} \boldsymbol{\Sigma}_{{}^{W}\mathbf{p}_i} \frac{\partial \mathbf{n}}{\partial {}^{W}\mathbf{p}_i}^T \tag{8}$$

where the covariance matrix $\boldsymbol{\Sigma}_{\mathbf{n}}$ is the uncertainty of plane 3DOF representation, which is calculated from the covariance matrix of the points on the plane $\boldsymbol{\Sigma}_{{}^{W}\mathbf{p}_i}$. For more detailed derivation about Jacobian, we refer our readers to our Supplementary Material[3].

Note that all elements in $\mathbf{A}$, $\mathbf{A}^*$ and $\mathbf{e}$ are the summation results, so the idea of dynamic programming can be used to update the 3DOF plane representation incrementally in practice which will reduce the complexity of fitting the plane during update introduced in Section III-B2.

B. Mergeable Voxel Mapping Method

1) Motivation: In the mapping method based on the voxels, scholars always ignore the relationship between voxels no matter how the voxel maps are segmented or how small the voxels are. It is undeniable that simply treating all voxels as independent individuals is very easy for engineering practice and has strong robustness. However, if we can effectively identify the relationships between voxels (e.g. whether they belong to the same plane), then using this relationship can reasonably improve the accuracy of voxel maps and further promote the performance of LiDAR(-inertial) odometry. Meanwhile, merging the voxels with unified properties (such as coplanar) can also effectively save memory usage which is undoubtedly meaningful for commercial low-cost robots.

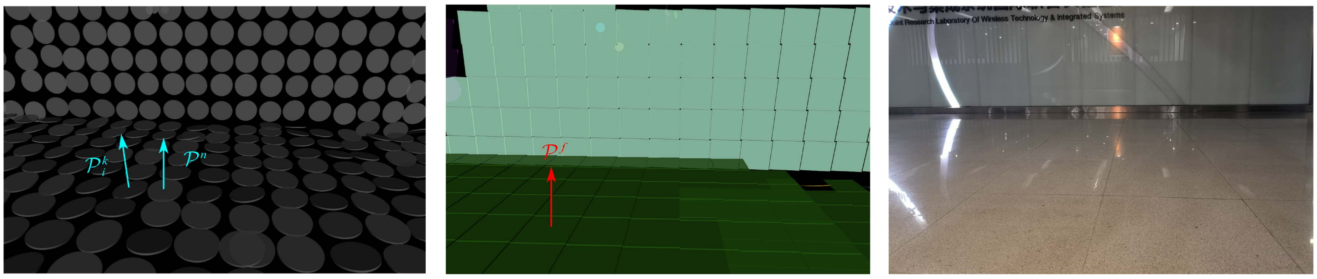
Based on the above ideas, we have modified the process of construction and update of voxel maps on the basis of previous research [2]. Then creatively proposed a voxel merging method based on union-find. Firstly, we distinguish the co-plane voxels based on the Chi-square test, these planes in the voxels are denoted as $\mathcal{P}^k$. Then, $\mathcal{P}^k$ are regarded as measurements with covariance of one large plane denoted as $\mathcal{P}^f$. Finally, we estimate the plane 3DOF representation and covariance of the $\mathcal{P}^f$ by minimizing the trace of the covariance matrix. $\mathcal{P}^f$ are more accurate than the estimation results of $\mathcal{P}^k$, thereby improving the accuracy of LiDAR(-inertial) odometry.

2) Voxel Map Construction and Update: Our method is based on spatial hashing to guarantee the real-time performance of the system. Precisely, our voxel map is maintained by a hash table, the 3-dimensional space is discretized into each small voxel, the key of the hash table is the identification of each voxel while the value of the hash table is the node of union-find which represents the plane feature in this voxel.

We set the side length of each voxel to 0.5 m rather than the default 3.0 m in VoxelMap. It is reasonable because we replace the octo-tree with a node of union-find for plane merging which will be introduced in Section III-B3. Due to the lack of the feature of voxel segmentation, if we want to achieve the same resolution, it is necessary to reduce the side length of the voxel. This mainly affects the memory usage of the hash table. However, by merging $\mathcal{P}^k$ in different voxels, we can free up the vast majority of voxels to reduce memory usage. In experiments shown in Section IV-C, we found that the memory usage of VoxelMap++ is even lower than that of VoxelMap since we do not need to maintain the planar representation in each voxel.

After the arrival of the first frame of LiDAR points, we check all of the voxels that contain enough points, then we use PCA to judge whether the points in each voxel can form a plane. After that, we calculate and store the plane 3DOF representation $\mathbf{n} = [a, b, d]^T$ and covariance $\Sigma_{\mathbf{n}}$ based on (6), and (8). For

[3][Online]. Available: https://github.com/uestc-icsp/VoxelMapPlus_Public/tree/main/doc



(a) plane estimation w/o merging (b) plane estimation w/ merging (c) reality

Fig. 2. Example of planes w/ and w/o merging. In (a), every single plane $\mathcal{P}_i^k$ has its own distinguish parameters which is not accurate due to the error of the sensor. In (b), the planes with the same color belong to one $\mathcal{P}^f$ and share the same plane representation. Compared with the reality structure in (c), the proposed plane merging method can reduce the noise of plane fitting in the large plane.

the LiDAR points that come after the first scan, their poses in the world frame will be obtained through state estimation. Then these points are registered into the map. If there already exists a voxel in the position of the point and this voxel has not converged, then this point will be added to it and the parameters of the voxel will be updated incrementally based on (6), and (8). Otherwise, we will construct a new voxel in that position.

To avoid the growing processing time with the arrival of more lidar points, the update of each voxel will stop when there are more than 50 points because the uncertainty of the parameters of the plane converges when the number of points reaches 50 [2]. When the voxel stops updating, we discard all points in the voxel to save memory and only retain the plane parameters $\mathbf{n} = [a, b, d]^T$ and their uncertainty $\Sigma_{\mathbf{n}}$. Finally, the newly converging set of planes $\mathcal{S}$ will serve as input of the plane merging algorithm in Section III-B3.

3) Plane Merging Based on Union-Find: After voxel map update, converged plane $\mathcal{P}^k$ will be merged with their neighborhood. The plane merging algorithm is designed based on union-find shown as Algorithm 1. $(.^f)$ denoted the pointer of the father node based on union-find. Left $(\&)$ represents the address retrieval operation. $t(.)$ denotes the calculation of the trace about the covariance matrix. One demo of the effect about plane merging module is shown in Fig. 2. Fig. 2(a) is the figure of planes without merging and Fig. 2(b) is the planes after merging. Planes with the same color belong to one plane and share the fitting results $\mathbf{n}$ and covariance $\Sigma_{\mathbf{n}}$. It is vividly shown that the plane merging module can effectively illuminate the Gauss noise of the plane in one single voxel.

$$\mathbf{r} = \mathbf{n}_{\mathcal{P}_i^k.f} - \mathbf{n}_{\mathcal{P}^n.f}$$
$$\gamma = \mathbf{r}(\mathbf{\Sigma}_{\mathbf{n}_{\mathcal{P}_i^k.f}} + \mathbf{\Sigma}_{\mathbf{n}_{\mathcal{P}^n.f}})^{-1}\mathbf{r}^T \tag{9}$$

$$\mathbf{n}_{New\mathcal{P}_i^k} = \frac{t(\mathbf{\Sigma}_{\mathbf{n}_{\mathcal{P}^n}.f})\mathbf{n}_{\mathcal{P}_i^k.f} + t(\mathbf{\Sigma}_{\mathbf{n}_{\mathcal{P}_i^k}.f})\mathbf{n}_{\mathcal{P}^n.f}}{t(\mathbf{\Sigma}_{\mathbf{n}_{\mathcal{P}_i^k}.f}) + t(\mathbf{\Sigma}_{\mathbf{n}_{\mathcal{P}^n}.f})} \tag{10}$$

$$\mathbf{\Sigma}_{\mathbf{n}_{New\mathcal{P}_i^k}} = \frac{t(\mathbf{\Sigma}_{\mathbf{n}_{\mathcal{P}^n}.f})^2\mathbf{\Sigma}_{\mathbf{n}_{\mathcal{P}_i^k}.f} + t(\mathbf{\Sigma}_{\mathbf{n}_{\mathcal{P}_i^k}.f})^2\mathbf{\Sigma}_{\mathbf{n}_{\mathcal{P}^n}.f}}{\left(t(\mathbf{\Sigma}_{\mathbf{n}_{\mathcal{P}_i^k}.f}) + t(\mathbf{\Sigma}_{\mathbf{n}_{\mathcal{P}^n}.f})\right)^2} \tag{11}$$

In 1-3, the new node in union-find which is the converged plane will be initialized. In 4-21, these new nodes will merge with other

Algorithm 1: Plane Merging Based on Union-Find.

```
    Require: converging planes set $\mathcal{S}$
 1: for $\mathcal{P}_i^k \in \mathcal{S}$ do
 2:   Initialize $\mathcal{P}_i^k$'s father node. $\mathcal{P}_i^k.f = \&\mathcal{P}_i^k$
 3: end for
 4: for $\mathcal{P}_i^k \in \mathcal{S}$ do
 5:   for $\mathcal{P}^n \in \mathcal{P}_i^k.neighbor$ do
 6:     Calculate the similarity $\gamma$ of $\mathcal{P}_i^k.f$ with $\mathcal{P}^n.f$ (9)
 7:     if $\gamma > \chi^2_{0.95}(1)$ then
 8:       continue
 9:     end if
10:     estimate ${}^{New}\mathcal{P}_i^k$ based on $\Sigma_{\mathbf{n}_{New\mathcal{P}_i^k}}$ (9), and (10)
11:     if $\mathcal{P}^n.f == \mathcal{P}^n$ then
12:       $\mathcal{P}^n.f = \mathcal{P}_i^k.f$
13:     else
14:       $Node = max\_kids(\mathcal{P}^n.f, \mathcal{P}_i^k.f)$
15:       $\mathcal{P}_i^k.f = \mathcal{P}^n.f = Node$
16:       pruning on each kids node about $\mathcal{P}_i^k.f$ and $\mathcal{P}^n.f$.
17:     end if
18:     $\mathcal{P}_i^k.f = \&{}^{New}\mathcal{P}_i^k$
19:     free the memory resource in $\mathcal{P}_i^k$ and $\mathcal{P}^n$
20:   end for
21: end for
```

nodes based on the union-find. In 5-20, each neighbor plane of $\mathcal{P}_i^k$ has been queried by calculating the spatial hash key values of the neighbor voxel. Then, perform the plane merging about this plane $\mathcal{P}_i^k$ and neighbor plane $\mathcal{P}^n$. In 6-9, we will calculate the similarity γ of $\mathcal{P}_i^k$ with $\mathcal{P}^n$ shown in (9) which is the random variable satisfy the distribution about $\chi^2(1)$. These two planes will be merged when the Mahalanobis distance is less than the threshold given by the 95% of the $\chi^2(1)$ distribution. In 10, we regard the $\mathcal{P}_i^k.f$ and $\mathcal{P}^n.f$ as the measurements with covariance of the larger plane ${}^{New}\mathcal{P}_i^k.f$, and estimate it covariance and 3DOF representation according to (9), and (10) which is a simple weighted average algorithm by minimizing the trace of $\mathbf{\Sigma}_{\mathbf{n}_{N\mathcal{P}_i^k}}$.
In 11-12, if neighbor's father node is its self, it is two simple cases of merging which is shown as Fig. 3(a), (b). In 13-15, if neighbor's father node is not its self, it is another complex case about plane merging which is shown in Fig. 3(c). In 16, we perform pruning after merging to maintain the maximum

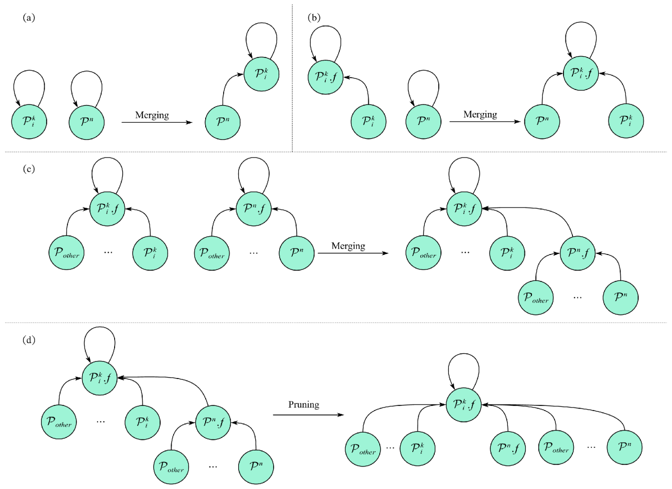


Fig. 3. Plane Merging and Node pruning method. (a) and (b) show the situation of merging when a neighbor's parent node is itself, and (c) illustrates when the neighbor's parent node is different. The pruning process, demonstrated in (d), is intended to reduce subsequent time consumption.

depth of union-find is 2 to avoid additional time consumption which is shown in Fig. 3(d). In 18, we assign the results of the larger plane ${}^{N}\mathcal{P}_i^k.f$ to $\mathcal{P}_i^k.f$, then $\mathcal{P}^n.f$ and $\mathcal{P}_i^k.f$ have been merged successfully. Finaly, in 19, we free the memory usage about kids plane which is not the plane about root node in the voxels. After merging, hundreds of voxels will share the same plane representation in $\mathcal{P}^k.f$, which effectively reduce memory usage and improve the accuracy about plane estimation.

C. State Estimation Based on IESKF

The state estimation is based on an iterated extended Kalman filter follow to FAST-LIO [1] and VoxelMap [2]. However, there are some differences in the calculation of the residuals and measurement covariance because of the difference in 3DOF plane representation. Assume that we are given a state estimation prior $\hat{\mathbf{x}}_k$ with covariance $\hat{P}_k$ based on Forward propagation. This prior will be updated with the point-to-plane distance registration to form a maximum a posteriori (MAP) estimation. Specifically, the i-th valid point-to-plane match leads to the observation equation shown as

$$\begin{aligned} z_i = h_i(\mathbf{x}_i, \mathbf{n}_i) &= 0 \qquad (12)\\ &\approx h_i(\hat{\mathbf{x}}_i^\kappa, 0) + \mathbf{H}_i^\kappa(\mathbf{x}_i \boxminus \hat{\mathbf{x}}_i^\kappa) + \mathbf{v}_i \end{aligned}$$

$$h_i(\hat{\mathbf{x}}_i^\kappa, 0) = \frac{\Omega^T({}^W_L\mathbf{R}^L\mathbf{p}_i + {}^W_L\mathbf{t}) + d}{||\Omega||} \qquad (13)$$

where $h_i(\hat{\mathbf{x}}_i^\kappa, 0)$ is the point-to-plane distance in the κ iteration and Ω is the normal vector $[a, b, 1]^T$ of registration plane $\mathcal{P}_i^k.f$. $\mathcal{P}_i^k.f$ can be determined by spatial hash querying $\mathcal{P}^k$ from map firstly, then finding the father nodes of $\mathcal{P}^k$ based on union-find. The $\mathbf{v}_j \sim (0, \mathbf{R}_j)$ is the observation noise propagate from the ${}^L\mathbf{p}_i$ and $\mathbf{n}_{\mathcal{P}_i^k.f}$ which is shown as (14).

$$\Sigma_{\mathbf{n}_{\mathcal{P}_i^k.f}, {}^L\mathbf{p}_i} = \begin{bmatrix} \Sigma_{\mathbf{n}_{\mathcal{P}_i^k.f}} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \Sigma_{{}^L\mathbf{p}_i} \end{bmatrix}$$

$$\mathbf{R}_i = \mathbf{J}_{\mathbf{v}_i}\Sigma_{\mathbf{n}_{\mathcal{P}_i^k.f}, {}^L\mathbf{p}_i}\mathbf{J}_{\mathbf{v}_i}^T$$

$$\mathbf{J}_{\mathbf{v}_i} = [\mathbf{J}_{\mathbf{n}_i}, \mathbf{J}_{{}^L\mathbf{p}_i}], \mathbf{J}_{{}^L\mathbf{p}_i} = \frac{1}{||\Omega||}\Omega^T \times {}^W_L\mathbf{R}$$

$$\mathbf{J}_{\mathbf{n}_i} = \frac{1}{||\Omega||}\begin{bmatrix} x_i(1 - \frac{1}{||\Omega||^2}h_i(\mathbf{x}_i, 0)) \\ y_i(1 - \frac{1}{||\Omega||^2}h_i(\mathbf{x}_i, 0)) \\ 1 \end{bmatrix}^T \qquad (14)$$

Finally, combining the state prior with all effective measurements, we can obtain the MAP estimation:

$$\min_{\hat{\mathbf{x}}_k^\kappa} \left\{ ||\hat{\mathbf{x}}_k^\kappa \boxminus \hat{\mathbf{x}}_k||^2_{\hat{\mathbf{P}}_k} + \sum_{i=1}^{N} ||h_i(\hat{\mathbf{x}}_k^\kappa, 0) + \mathbf{H}_i^\kappa(\mathbf{x}_k \boxminus \hat{\mathbf{x}}_k^\kappa)||_{\mathbf{R}_i} \right\} \qquad (15)$$

where the first part is the state prior and the second part is the measurement observation. The detail solution is based on an iterated extended Kalman filter can refer [1], [2], [25].

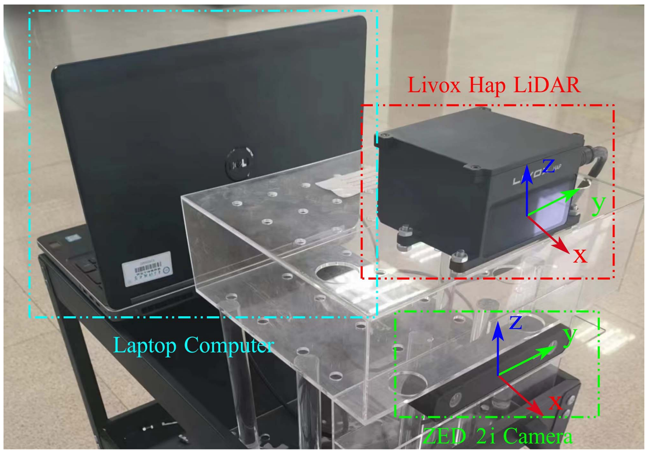


Fig. 4. Our data collection platform has Livox HAP LiDAR and ZED 2i camera with inner IMU, these devices are well strapdown on the trolley.

IV. Experiments

We implemented the proposed VoxelMap++ system in C++ and Robots Operating System (ROS) on a laptop computer with 2.9 GHz 8 cores and 16Gib memory. The experiment data not only includes structure ubran open source datasets M2DGR [19] and KITTI Vision Benchmark [22], but also our own private challenging datasets in unstructured and degenerated environment.

The M2DGR datasets used a Velodyne VLP-32 C with 360° $\times$ 40° FOV to scan the surrounding environment and obtain the 3D point cloud in 10 Hz. It also used a VI-sensor Realsense d435i to obtain inertial data at 200 Hz. The LiDAR and IMU in KITTI is collected by Velodyne HDL-64E S2 and OXTS RT3003 in urban environments.

Our own datasets are collected via Livox HAP LiDAR with 120° $\times$ 25° FOV and a ZED 2i camera with built-in Next-Gen IMU. The frequency of LiDAR point cloud and IMU is 10 Hz and 500 Hz, respectively. The sensors platform is shown as Fig. 4.

TABLE I
ACCURACY ATE IN METERS COMPARISON ON STRUCTURED URBAN SEQUENCES

Method	A-LOAM	LeGO-LOAM	LIO-SAM	LINS	FAST-LIO2	Faster-LIO	Point-LIO	VoxelMap	Ours (3DOF)	Ours (full)	Distance/m
Street02(a)	5.299	20.021	4.063	5.636	2.323	2.666	2.040	1.740	1.961	**1.16**0	1484
Street06(b)	0.628	1.246	0.417	1.742	0.457	0.413	**0.395**	0.490	0.519	0.412	479
Street07(c)	28.940	35.437	28.642	12.009	11.751	11.736	10.374	13.761	15.278	**12.853**	1104
Roomdark06(d)	0.314	0.373	0.324	2.205	0.314	0.312	0.301	0.294	0.282	**0.253**	72
Hall05(e)	1.065	1.030	1.047	1.010	1.022	1.031	1.008	0.937	0.929	**0.899**	79
Door01(f)	0.274	0.253	0.268	0.258	0.256	0.252	0.223	0.236	0.238	**0.217**	285
Lift04	1.323	1.370	X	**1.318**	X	X	X	X	X	X	142
KITTI 00	19.417	6.346	8.017	5.837	3.851	4.602	4.654	3.900	3.967	**3.5471**	3724
KITTI 04	0.593	0.834	0.743	0.647	0.555	0.752	0.541	**0.2716**	0.305	0.2827	393
KITTI 06	1.189	0.867	0.872	1.324	1.296	1.044	1.567	**0.5848**	0.813	0.7961	1232
KITTI 07	1.301	0.795	**0.639**	1.634	0.883	1.456	0.749	0.7651	0.775	0.7095	694
KITTI 10	1.969	1.863	1.352	1.759	1.572	1.575	1.078	1.1741	0.960	**0.7981**	919

The bold values represent the algorithms with the lowest positioning error in the current sequence.

A. Experiment on Structured Urban Datasets

M2DGR and KITTI covers the environment of structured urban, this two datasets is collected with a full sensor-suite including an IMU and a LiDAR, while it is also equipped with a GNSS-IMU system with real-time kinematic signals to get the dataset's ground truth.

Table I gives detailed information about tests route and evaluation results on A-LOAM, LeGO-LOAM, LIO-SAM, LINS, FAST-LIO2, VoxelMap, latest L-SLAM algorithms Point-LIO and our proposed VoxelMap++. The full version of the algorithm is denoted as Ours (full). To verify the influence about merging, we elliminate the plane merging module termed as Ours (3DOF). However, 6DOF representation is not applicable in our merging strategy, we don't experiment the case about 6DOF plane merging. It is clear that these sequences include various environments for SLAM including long distance and short distance, indoors and outdoors, straight line and zigzag route. These scenarios are sufficient to illustrate the structured urban environment.

We use the absolute trajectory error (ATE) [20] to illuminate the accuracy, where the results for A-LOAM, LeGO-LOAM, LIO-SAM and LINS are directly drawn from M2DGR. Performance is calculated based on evo [21] which is an open-source trajectory error evaluation toolkit. Ours(full) has better accuracy than other algorithms including VoxelMap under most circumstances. Without plane merging, the accuracy of VoxelMap and Ours(3DOF) is basically the same on different datasets, indicating the correctness of the 3DOF plane representation. It should be noted that for sequence lift04, the positioning of many algorithms have failed. This is because the lift04 sequence includes rising in a closed elevator which means the map has changed drastically and will cause catastrophic error in point cloud registration. In VoxelMap and VoxelMap++, the voxel on the elevator gates has converged before closed which means the map will not change anymore resulting in the continuous error in scan-to-map registration. In other methods based on point cloud mapping, the wrong point cloud registration in scan-to-map will also occur, but the inter-frame constraints based on scan-to-scan effectively alleviate state divergence. This phenomenon illuminates that scan-to-map registration is not applicable in some specific dynamic scenes in which the geometric constraints change rapidly in a short period of time. In order to determine the performance of our method in normal dynamic scenarios, we also conducted a series of extended experiments under dynamic scenarios in our Supplementary Material^5.

The reason our method is more accurate than VoxelMap and other state-of-art algorithms is that we adopt the plane merging block in voxel map update, which means more points will be used to describe a single plane. In point cloud mapping methods such as FAST-LIO2, only 5 points are used to fit the plane commonly because of the limitation of the real-time performance of the system, which makes the estimation of the plane not accurate enough for positioning. In VoxelMap, points within a voxel are used for plane fitting, but ignore the co-plane relationship between voxels. Therefore, its accuracy improvement is also limited. In VoxelMap++, our plane merging block based on union-find Section III-B3 can better estimate the plane fitting result and covariance by making good use of the coplanar information about the entire voxel map. In some cases, the laser points on the entire wall or the floor can be used to estimate one single huge plane.

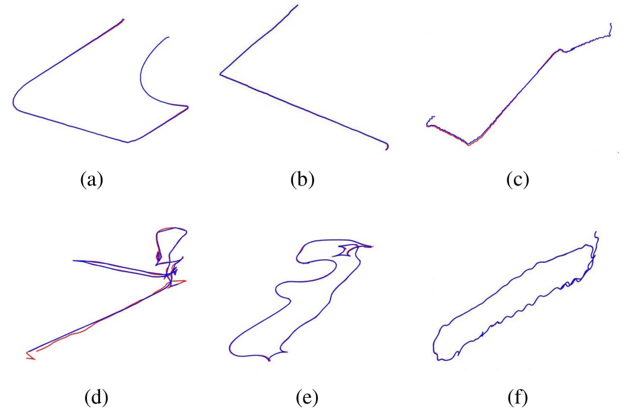


Fig. 5. (a)–(f) shows the estimated trajectories of our method (blue) and ground-truth (red) on the structure urban environment of the M2DGR dataset.

Fig. 5 shows the LiDAR trajectories of our method and their ground truth on all sample sequences. It is obvious that the trajectories of our method are very close to the ground truth, which means our method has high accuracy. Notice that we use the same parameters in all sequences experiment rather than deliberately tuning the parameter to show better results.

B. Experiment on Challenging Scenarios

In order to further test the performance of our algorithm in other challenging scenarios, such as unstructured scenes or indoors with long corridors. We design a series of experiments on these scenarios based on our own data collection platform. Due to the lack of hardware for GNSS-IMU systems with real-time kinematic signals or other ground truth. We use end-to-end errors, which are defined as the difference between the start point

Fig. 6. Outdoor unstructured scenarios of our experiment.

TABLE II
END-TO-END ERROR(METERS) ON UNSTRUCTURED SEQUENCES

Sequence	loopE	loopF	loopG	loopH	loopI
LIO-Livox	5.9141	2.8160	3.6342	2.2157	1.2061
FAST-LIO2	3.2134	0.0830	4.2893	0.8175	0.1591
Faster-LIO	5.7331	1.3926	1.6632	1.7050	0.4856
VoxelMap	1.6847	0.9577	0.4433	**0.0492**	0.0894
VoxelMap++	**0.0336**	**0.0441**	**0.0389**	0.0734	**0.0406**
Route Length(m)	329.4	373.6	311.8	519.6	284.4

The bold values represent the algorithms with the lowest positioning error in the current sequence.

and terminal point, to compare the accuracy of different algorithms. In order to estimate the end-to-end error, we will always push the cart with the data collection platform in the path around challenging scenarios and return to the origin finally. Because the LiDAR in the platform is the non-repetitive scanning Livox HAP, we can only use the state-of-art algorithm which is applicable for this sensor for comparison, such as FAST-LIO2, Faster-LIO, VoxelMap, and Livox's official LIO-Livox. We recommend readers to view the visualization results of challenging scenarios about VoxelMap++ in Supplementary Material[5].

1) Unstructured Forests and Grassland: Fig. 6 shows the unstructured scenarios of our experiment. we conducted experiments in a forest and grassland at the library's entrance in UESTC.

As shown in Table II, both VoxelMap and VoxelMap++ are more robust and accurate than other state-of-the-art methods in unstructured scenarios. This promotion is mainly due to the covariance estimation method in the point cloud $\Sigma_{^{W}\mathbf{p}_i}$, plane features $\Sigma_{\mathbf{n}}$ and observation noise $\mathbf{R}$. Our proposed VoxelMap++ is more accurate than VoxelMap because the plane merging module can make the estimation of the plane more accurate and the covariance declines in the huge plane. This means that the IESKF tends to rely on large planes with low observation covariance rather than chaotic features in forests and grasslands.

Other methods such as FAST-LIO2 regard the noise in each point-to-plane distance as a constant related to the accuracy of sensors, which means the features in the messy objects such as tree leaves and weed have the same influence as planar points. These features in messy objects can not satisfy the assumption of a point in the plane anymore. Therefore, these features should be directly deleted or adjusted covariance scientifically. The covariance propagation in VoxelMap is a useful strategy.

The error detection of coplanar recognition based on the chi-square test in Section III-B is relatively low, therefore it also has robustness in unstructured scenarios like VoxelMap.

2) Indoor Degenerated Corridor: Fig. 7 shows the indoor degenerated scenarios with long corridors of our experiment. The degenerate directions are denoted as the red arrow. The definition of degeneracy refers to degeneracy factor [24] which is the minimum eigenvalue of $\mathbf{H}^T\mathbf{H}$ and the associated eigenvector represent the degenerate direction.

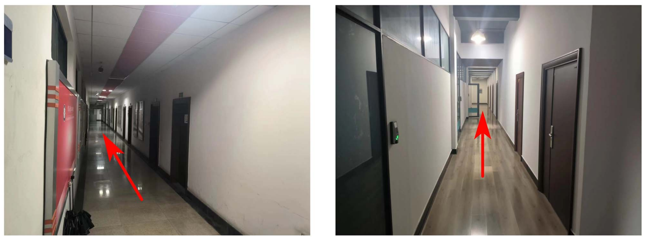


Fig. 7. Indoor degenerated scenarios of our experiment.

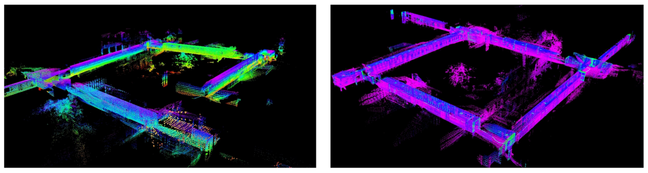


Fig. 8. Mapping result in loop1. Left is the result of VoxelMap, right is the result of VoxelMap++.

TABLE III
END-TO-END ERROR(METERS) ON INDOOR CORRIDOR SEQUENCES

Sequence	loop1	loop2	loop3	loop4	loop5
LIO-Livox	7.1192	18.5059	4.2932	10.0270	19.6729
FAST-LIO2	2.3812	2.8175	4.2703	13.2844	7.1070
Faster-LIO	2.3488	1.1595	8.1295	1.6455	10.1804
VoxelMap	9.3457	1.1388	3.8561	5.3355	14.2049
VoxelMap++	**0.5305**	**0.5671**	**0.2543**	**1.3094**	**1.4478**
Route Length(m)	202.9	219.7	315.2	317.1	405.0

The bold values represent the algorithms with the lowest positioning error in the current sequence.

In the corridor, because of the lack of plane constraints in the forward direction which means that the measurement Jacobian will lack the normal vector like [1,0,0] in body frame. It is inevitable to cause significant cumulative errors in the x-axis. Meanwhile, when turning at an intersection, the LiDAR will scan the next corridor without passing through, which means that the performance of the LIO attitude at this time will largely depend on the forward propagation of the IMU. This can easily lead to attitude errors, resulting in significant linear cumulative errors. In VoxelMap++, due to the advantage of plane merging, the entire floor and ceiling will be merged into two large planes, thus constraining the drift on the pitch and avoiding linear cumulative errors which are shown in Fig. 8. This phenomenon is similar to the ground segmentation in LeGO-LOAM, but it is further extended to all coplanar planes. In the direction of degeneracy occurred, VoxelMap++ can merge the coplanar walls at the end of the corridor, thereby enhancing their constraints to alleviate degradation by reducing the covariance about these planes.

As Table III shows, other SLAM algorithms are more prone to cumulative errors in corrider. Our proposed VoxelMap++ achieves much higher accuracy than others, mainly due to the plane merging can estimate the plane representation more accurately in real-time.

TABLE IV
RESOURCES USAGE (AVG. COMP. TIME (MS)/ MEM USAGE (MB))

	small scale (1,000m^2)	large scale (90,000m^2)
FAST-LIO2	11.5985/201.86	35.4858/228.23
Faster-LIO	7.3461/135.11	15.2910/200.21
VoxelMap	5.6830/158.06	14.5815/201.22
VoxelMap++	**4.8763/126.45**	**13.8323/195.91**

The bold values represent the algorithms with the lowest positioning error in the current sequence.

C. Consumption of Resources

Another advantage of our proposed VoxelMap++ is less CPU and memory resource usage compared with other state-of-art methods which is shown in Table IV. This promotion means our method can be applied to the resource constraint embedding system. 50 is used as a threshold for plane update both in VoxelMap and VoxelMap++. Other parameters are the same as the open-source repository.

From the perspective of memory usage, compared with the method based on the point cloud, we only need to maintain a hash table with save the plane representation which means we will not maintain millions of point clouds in the map. Compared with Voxelmap, the plane merging method in Section III-B3 will save the memory usage of voxels as much as possible. Ultimately, the map in VoxelMap++ will be represented as several large planes $\mathcal{P}_i^k.f$ and other planes $\mathcal{P}_j^k$ cannot be merged, rather than hundreds of plane in VoxelMap.

From the perspective of CPU usage, compared with the method based on the point cloud, we use spatial hashing O(1) instead of KNN search O(log (N)) based on KD-Tree. The plane parameters can be directly obtained by querying the value in the hash table instead of repeatedly performing plane fitting. Both these advantages make the CPU usage of VoxelMap and VoxelMap++ much lower than that of pointcloud-based methods such as FAST-LIO2. Compared with VoxelMap, we perform 3DOF plane representation rather than 6DOF which means our proposed VoxelMap++ consumes less computational resources in the plane associate operation. Meanwhile, since all inputs in the plane fitting method (6), and (8) are summation. We can incrementally update the value instead of recalculating which is a useful trick in programming.

V. Conclusion and Limitation

This letter proposes a mergeable voxel mapping method for online LiDAR(-inertial) odometry. Compared with other methods, this method maintains the plane feature with 3DOF representation and corresponding covariance which effectively improves the calculation speed and saves memory usage. In order to improve the accuracy of plane fitting, we make full use of the relationship between voxels and then merge the coplanar voxel based on union-find after plane fitting converges. This letter also shows how to implement the proposed mapping method in an iterated extended Kalman filter-based LiDAR(-inertial) odometry. The experiment on structured open-source datasets and our own challenging datasets shows that our method can achieve better performance than other state-of-the-art methods.

However, our method also has some drawbacks. For example, the robustness in dynamic scenes, such as closing elevators, will drop significantly. Thus, we will consider optimizing this method to identify the changing voxel in the future.

References

[1] W. Xu and F. Zhang, “FAST-LIO: A. fast, robust LiDAR-inertial odometry package by tightly-coupled iterated Kalman filter,” *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 3317–3324, Apr. 2021.
[2] C. Yuan, W. Xu, X. Liu, X. Hong, and F. Zhang, “Efficient and probabilistic adaptive Voxel mapping for accurate online LiDAR odometry,” *IEEE Robot. Autom. Lett.*, vol. 7, no. 3, pp. 8518–8525, Jul. 2022.
[3] G. Wan et al., “Robust and precise vehicle localization based on multi-sensor fusion in diverse city scenes,” in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 4670–4677.
[4] W. Ding, S. Hou, H. Gao, G. Wan, and S. Song, “LiDAR inertial odometry aided robust LiDAR localization system in changing city scenes,” in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 4322–4328.
[5] D. He, W. Xu, N. Chen, F. Kong, C. Yuan, and F. Zhang, “Point-LIO: Robust high-bandwidth light detection and ranging inertial odometry,” *Adv. Intell. Syst.*, vol. 5, no. 7, 2023, Art. no. 2200459.
[6] K. Chen, B. T. Lopez, A.-A. Agha-mohammadi, and A. Mehta, “Direct LiDAR odometry: Fast localization with dense point clouds,” *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 2000-2007, Apr. 2022.
[7] A. Reinke et al., “LOCUS 2.0: Robust and computationally efficient LiDAR odometry for real-time 3D mapping,” *IEEE Robot. Automat. Lett.*, vol. 7, no. 4, pp. 9043–9050, Oct. 2022.
[8] J. Zhang and S. Singh, “LOAM: LiDAR odometry and mapping in real-time,” in *Proc. Robot.: Sci. Syst.*, 2014, pp. 1–9.
[9] T. Shan and B. Englot, “LeGO-LOAM: Lightweight and ground-optimized LiDAR odometry and mapping on variable terrain,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 4758–4765.
[10] T. Shan et al., “LIO-SAM: Tightly-coupled LiDAR inertial odometry via smoothing and mappi NG,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2020, pp. 5135–5142.
[11] H. Ye, Y. Chen, and M. Liu, “Tightly coupled 3D LiDAR inertial odometry and mapping,” in *Proc. IEEE Int. Conf. Robot. Automat.*, 2019, pp. 3144–3150.
[12] C. Qin, H. Ye, C. E. Pranata, J. Han, S. Zhang, and M. Liu, “LINS: A. LiDAR-inertial state estimator for robust and efficient navigation,” in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 8899–8906.
[13] K. Li, M. Li, and U. D. Hanebeck, “Towards high-performance solid-state-LiDAR-inertial odometry and mapping,” *IEEE Robot. Automat. Lett.*, vol. 6, no. 3, pp. 5167–5174, Jul. 2021.
[14] 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 Robot. Automat. Lett.*, vol. 7, no. 2, pp. 4861–4868, Apr. 2022.
[15] Z. Liu and F. Zhang, “BALM: Bundle adjustment for LiDAR mapping,” *IEEE Robot. Automat. Lett.*, vol. 6, no. 2, pp. 3184–3191, Apr. 2021.
[16] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, “Fast-LIO2: Fast direct LiDAR-inertial odometry,” *IEEE Trans. Robot.*, vol. 38, no. 4, pp. 2053–2073, Aug. 2022.
[17] W. Lee, Y. Yang, and G. Huang, “Efficient multi-sensor aided inertial navigation with online calibration,” in *Proc. IEEE Int. Conf. Robot. Automat.*, 2021, pp. 5706–5712.
[18] C. Yuan, X. Liu, X. Hong, and F. Zhang, “Pixel-level extrinsic self calibration of high resolution LiDAR and camera in targetless environments,” *IEEE Robot. Automat. Lett.*, vol. 6, no. 4, pp. 7517–7524, Oct. 2021.
[19] J. Yin, A. Li, T. Li, W. Yu, and D. Zou, “M2DGR: A multi-sensor and multi-scenario SLAM dataset for ground robots,” *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 2266–2273, Apr. 2022.
[20] Z. Zhang and D. Scaramuzza, “A tutorial on quantitative trajectory evaluation for visual (-inertial) odometry,” in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 7244–7251.
[21] Michael Grupp. evo, “Python package for the evaluation of odometry and SLAM,” 2017. [Online]. Available: https://github.com/MichaelGrupp/evo
[22] A. Geiger, P. Lenz, and R. Urtasun, “Are we ready for autonomous driving? The KITTI vision benchmark suite,” in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2012, pp. 3354–3361.
[23] J. Lin and F. Zhang, “Loam livox: A fast robust high-precision LiDAR odometry and mapping package for LiDARs of small FoV,” in *Proc. IEEE Int. Conf. Robot. Automat.*, 2020, pp. 3126–3131.
[24] J. Zhang, M. Kaess, and S. Singh, “On degeneracy of optimization-based state estimation problems,” in *Proc. IEEE Int. Conf. Robot. Automat.*, 2016, pp. 809–816.
[25] D. He, W. Xu, and F. Zhang, “Embedding manifold structures into Kalman filters,” 2021, *arXiv:2102.03804*.