Loam_livox: A fast, robust, high-precision LiDAR odometry and mapping package for LiDARs of small FoV

Jiarong Lin and Fu Zhang

***Abstract*— LiDAR odometry and mapping (LOAM) has been playing an important role in autonomous vehicles, due to its ability to simultaneously localize the robot's pose and build high-precision, high-resolution maps of the surrounding environment. This enables autonomous navigation and safe path planning of autonomous vehicles. In this paper, we present a robust, real-time LOAM algorithm for LiDARs with small FoV and irregular samplings. By taking effort on both front-end and back-end, we address several fundamental challenges arising from such LiDARs, and achieve better performance in both precision and efficiency compared to existing baselines. To share our findings and to make contributions to the community, we open source our codes on Github[1].**

I. Introduction

With the ability to provide long range, highly accurate 3D measurements of the surrounding environment, light detection and ranging (LiDARs) is becoming an essential sensor in many robotic applications, such as autonomous driving vehicles [1], drones [2, 3], surveying, and mapping [4, 5]. To enable massive use in these areas, recent developments in LiDAR technologies have been focusing on lowering the device cost while increasing its reliability [6]. In this trend, one class of LiDARs that gains increasingly interests and developments are solid state LiDARs, which come with various implementations, such as micro-electro-mechanical-system (MEMS) scanning, optical phase array (OPA), Risley prism, etc. Being massively produced [2], these high performance and extremely low-cost LiDARs hold the potential to promote or radically change the robotics industry.

Despite their superiority in cost, reliability, and possibly performance against the conventional mechanical spinning LiDARs, such as Velodyne Puck [3], solid state LiDARs have many new features that bring significant challenges to the LiDAR navigation and mapping. These features are (to explain these features, we take the Livox MID40 LiDAR [2] as an example due to its wide availability):

Small FoV: solid state LiDARs usually have very small field of view (FoV). For examples, Livox MID40 has a front facing, conical shaped FoV spanning 38.4 degrees. Other solid state LiDARs such MEMS LiDARs also suffer from similar small FoV problem due to the large size of the MEMS mirror preventing large steering angles. Comparing

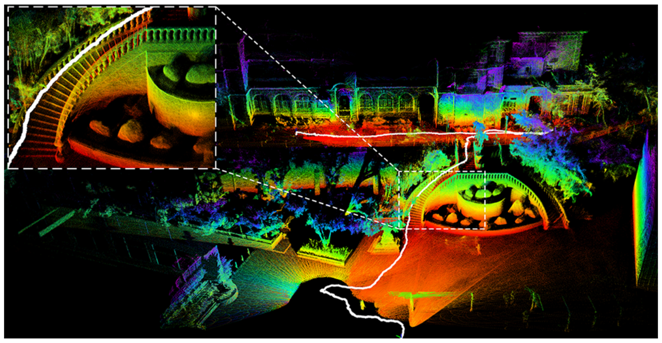


Fig. 1: The 3D map of the Chong Yuet Ming Cultural Center in the University of Hong Kong (HKU).

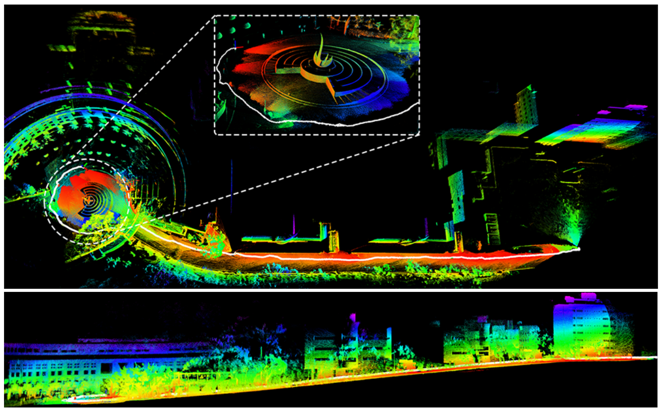


Fig. 2: The large scale mapping of the Hong Kong University of Science and Technology (HKUST) main campus, the upper and lower images are the bird-view and side-view, respectively. In above images, the white path is the trajectory of the LiDAR, points are colored by their heights. The accompanying video is available at `https://youtu.be/WHbbtU-Q9-k`

to conventional spinning LiDAR (see Fig. 3), the reduced FoV will lead to very fewer features in a frame, making the subsequent feature matching prone to degenerate and easily disturbed by moving objects. Although a larger FoV can be obtained by combining multiple LiDARs, it considerably increases the sensor cost and weight.

Irregular scanning pattern: existing spinning LiDARs have multiple laser-receiver pairs stacking in a vertical row. Rotating all pairs as a whole leads to a collection of parallel rings. This regular scanning greatly simplifies the feature extraction. For example, a corner is easily computed by differentiating the depth of points on the line. In constrat, the scanning pattern of solid state LiDARs is quite irregular. For example, the Livox MID40 has a rosette-like scanning pattern (see Fig. 4) where two neighboring scanning petals are separated far apart.

Non-repetitive scanning: to maximize the coverage ratio

J. Lin and F. Zhang are with the Department of Mechanical Engineering, Hong Kong University, Hong Kong SAR., China. {`jiarong.lin`, `fuzhang`}`@hku.hk`

[1]`https://github.com/hku-mars/loam_livox`

[2] `https://www.livoxtech.com/mid-40-and-mid-100`

[3]`https://velodynelidar.com/vlp-16.html`

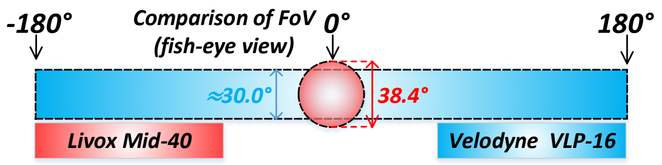


Fig. 3: FoV of Livox Mid-40 and Velodyne PUCK (VLP-16).

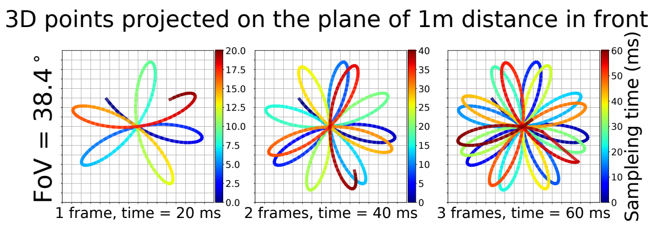


Fig. 4: The scanning trajectory of 3D points projected on the plane of $1m$ distance in front, where the color encodes the sampling time.

even when the LiDAR is static, non-repetitive scanning is usually adopted [7] where the scanning trajectory never repeats itself (see Fig. 4).

Motion blur: due to the continuous scanning of a single laser head, the 3D points measured in one frame are really sampled at different times as the LiDAR is continuously moving. The in-frame motion will distort the point clouds and cause motion blur although the motion blur also exists in conventional mechanical spinning LiDARs (and essentially all sequentially scanning LiDARs), it is usually less disastrous due to the simultaneous scanning of multiple lasers.

To address the problems mentioned above, we develop a software package named "Loam_Livox", which addresses many key issues including feature extraction and selection in a very limited FoV, robust outliers rejection, moving objects filtering and motion distortion compensation. Without other sensors such as IMU, GPS, and cameras, our algorithm calculates the LiDAR poses in real time (i.e. odometry) by registering its point cloud to a specified range of local map. Some of the results we obtained are shown in Fig. 1 and Fig. 2, where we can tell the precision of the algorithm from the level of details of the stairs and railing (Fig. 1), as well as versatility for large-scale mapping (Fig. 2).

II. Related work

State estimation and map-building are the fundamental prerequisites for intelligent robots. In the past recent years, we have seen great efforts being made in the field of simultaneous localization and mapping (SLAM), including both vision-based and laser-based approaches. In this paper, we mainly focus on the problem of laser-based SLAM.

Besl *et al* [8] first proposed the Iterative Closest Points (ICP) method for scan matching, which builds the basic operation for odometry. Building on this, Mendes *et al* [9] proposed a pose-graph SLAM to correct the drift in sequential scan matching, and demonstrated its effectiveness in a high definition LiDARs, Velodyne HDL 64.

While the ICP algorithm performs well for 3D scans with dense points, its effectiveness considerably degrades when the points in a scan are sparse where the two scans do not scan the same location on an object. To solve this problem, Pulli *et al* [10] proposed a "point-to-plane" error metric. This metric is used together with the "point-to-point" metric in [8] and called the generalized ICP in [11]. Zhang *et al* [12] and Shan *et al* [13] also used the "point-to-edge" metric in the context of LiDAR odometry and mapping.

Besides the geometry features mentioned above, 3D key-points based method [14]–[16] have also been proposed. These methods required less computation resources, by extracting keypoints from dense point cloud with detector like Point Feature Histograms (PFH) [14, 15], Viewpoint Feature Histograms (VFH) [16], etc. Considering the point cloud characteristic of our scenarios and the demands of real-time performance, we use point-to-edge and point-to-plane feature in our work, inspired by the work of [12, 13].

To eliminate the effects of motion blur caused by LiDAR movement, authors in [12] , [17] and [18] compensate the movements in front-end processing by linear interpolating the LiDAR pose. More recently, Gentil *et al* [19] formulates an optimization problem in the back-end processing to compensate the LiDAR movement. Compared to the previous work, the back-end processing method achieves better performance but cannot run in real-time.

While most of the previous work were based on spinning LiDARs, in this work, we focus on the odometry and mapping with solid-state LiDARs of small FOVs. Our contributions are: (1) we develop a complete LOAM algorithm for LiDARs with small FOVs. The algorithms is carefully engineered and made open source to benefit the community; (2) we further increase the accuracy and robustness of the LOAM algorithm by considering the low-level physical properties of LiDAR sensors in the front-end processing; (3) we propose a picewise processing technique to overcome the motion blur problem, and parallelize its implementation. Experiments show that the piecewise processing outperforms linear interpolation in terms of accuracy and running efficiency.

III. Points selection and feature extraction

The overview or our system is shown as Fig. 5, whose front-end processing comprises of the point selection and feature extraction. Considering the measuring mechanism of a LiDAR sensor its low-level physical properties (e.g., laser spot size, signal noise ratio), we perform a point level selection to extract the "good points".

A. Points selection

We compute the following features of each 3D point $\mathbf{p} = [x, y, z]$, where the $X-Y-Z$ axis correspond to the Front-Left-Up (FLU) of a LiDAR (see Fig. 6 (a)).

Depth D is the distance of the measured point to the LiDAR sensor.

$$D(\mathbf{p}) = \sqrt{x^2 + y^2 + z^2} \tag{1}$$

The laser deflection angle Φ is the angle between X axis and laser ray

$$\Phi(\mathbf{p}) = \tan^{-1}\left(\sqrt{(y^2 + z^2)/x^2}\right) \tag{2}$$

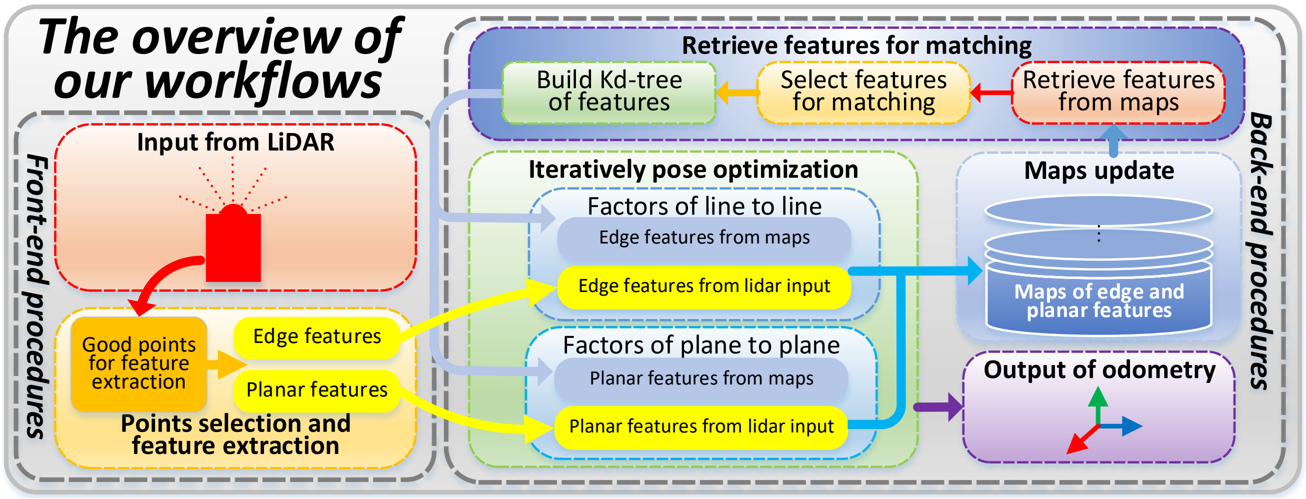

The overview of our workflows
Front-end procedures
Input from LiDAR
Good points for feature extraction
Edge features
Planar features
Points selection and feature extraction
Retrieve features for matching
Build Kd-tree of features
Select features for matching
Retrieve features from maps
Iteratively pose optimization
Factors of line to line
Edge features from maps
Edge features from lidar input
Factors of plane to plane
Planar features from maps
Planar features from lidar input
Maps update
Maps of edge and planar features
Output of odometry
Back-end procedures

Fig. 5: The overview of our workflows. Each new frame is matched directly with the global map to provide the odometry output. The matching result is in turn used to register the frame to the global map, leading to the same rate (i.e., 20 Hz) of odometry output and map update. In our implementation, only the feature points (i.e., edge points and plane points) are saved in memory and all the raw points are saved in hard disk for possible offline processing (e.g., offline global optimization).

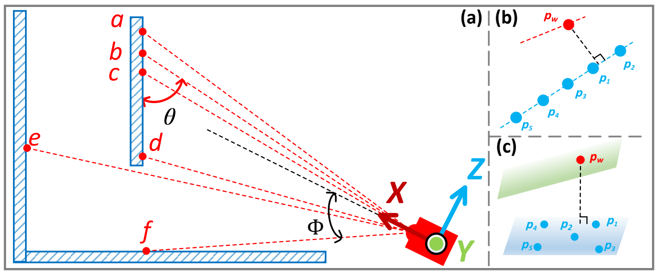


Fig. 6: (a) Illustration of incident angle θ, deflection angle Φ; (b) residual of edge-to-edge; (c) residual of plane-to-plane.

The intensity I is $I(\mathbf{p}) = R/D(\mathbf{p})^2$, where R is the object reflectivity, measured by the LiDAR sensor (some LiDARs, e.g., Velodyne Puck, returns the intensity instead of reflectivity. In this case, the intensity is directly available). Small intensity $I(\mathbf{p})$ means the point is either far from the LiDAR sensor (large $D(\mathbf{p})$) or the object reflectivity R is low.

The incident angle θ is the angle between the laser ray and the local plane around the measured point (Fig. 6 (a)).

$$\theta(\mathbf{p}_b) = \cos^{-1}\left(\frac{(\mathbf{p}_a - \mathbf{p}_c)\cdot \mathbf{p}_b}{|\mathbf{p}_a - \mathbf{p}_c|\,|\mathbf{p}_b|}\right) \tag{3}$$

To increase the localization and mapping accuracy, we remove any of the following points:

1. Points nearing to the fringe of the FoV. (e.g., $\Phi(\mathbf{p}) \geq 17^\circ$ for Livox MID40). In such area, scanning trajectory has large curvatures, leading to the feature extraction in Section III.B less reliable.
2. Points with too large or too small intensity (e.g. $I(\mathbf{p}) \leq 7\times10^{-3}$,or $I(\mathbf{p}) \geq 1\times10^{-1}$ for MID40). This is because intensity directly indicates the strength of the received laser signal. Too large intensity (signal) usually leads to saturation or distortion in the receiving circuitry and decreases the ranging accuracy. On the other hand, too small intensity (signal) usually leads to lower signal noise ratio, which also deteriorate the ranging accuracy.
3. Points with incident angles near to π or 0 (e.g. $\theta(\mathbf{p}) \leq 5^\circ$,or $\theta(\mathbf{p}) \geq 175^\circ$ for MID40), like point $\mathbf{p}_f$ in Fig. 6 (a). This is because the laser spot caused by the nonzero divergence angle of the laser beam will be considerably elongated. As a result, the measured range is the average of the area covered by the large spot instead of a specific point.
4. Points hidden behind an objects (e.g., $\mathbf{p}_e$ in Fig. 6 (a)), which will cause a false edge feature otherwise. A point $\mathbf{p}_e$ is a hidden point if:

$$|\mathbf{p}_e - \mathbf{p}_d| \geq 0.1|\mathbf{p}_e|, \text{ and} |\mathbf{p}_e| > |\mathbf{p}_d|$$

where $\mathbf{p}_d$ is the nearest measurement point in scanning order.

B. *Feature extraction*

After points selection, we perform feature extraction to extract features from the “good points”. We extract plane and edge features by computing the local smoothness of the point candidate as in [12]. Furthermore, to mitigate the matching degeneration due to the small number of features caused by the limited FoV and the point selection, we employ the LiDAR reflectivity as the 4-th dimensional measurement. If the reflectivity of a 3D point is considerably different its neighborhood points, we treat it as a point of edge feature (edge in the reflectivity due to materials change, in contrast to the edge in geometry due to shape change). Such points are beneficial in some of the degeneration cases like facing a wall with closed doors and windows.

IV. Iterative pose optimization

Due to the non-repetitive scanning mentioned in Section. I, the extracted feature cannot be constantly matched between two frames like in [12, 13, 19]. A simple example is that, even when the LiDAR is static, the scanned trajectory (and feature points) are different from the previous frame. In our work, we use an iterative pose optimization procedure to calculate the LiDAR pose. With the proper implementation detailed later, we achieve real-time odometry and mapping, both at 20Hz.

A. *Residual of edge-to-edge*

Denote $\boldsymbol{\mathcal{E}}_k$ and $\boldsymbol{\mathcal{E}}_m$ the set of all edge features (see Section. III-B) in the current frame and in the map, respectively. For each point in $\boldsymbol{\mathcal{E}}_k$, we find 5 nearest points from $\boldsymbol{\mathcal{E}}_m$ (see Fig. 6 (b)). To boost the searching speed, we build a *KD-tree* of $\boldsymbol{\mathcal{E}}_m$ (see Fig. 5). Moreover, the KD-tree is built by another parallel thread once the last registered frame/sub-frame is received (see Fig. 7). This makes the KD-tree immediately available when the new frame is received.

Let $\mathbf{p}_l$ be a point in $\boldsymbol{\mathcal{E}}_k$ of the current frame (k-th frame). Noticing that the point $\mathbf{p}_l$ in $\boldsymbol{\mathcal{E}}_k$ is in the local LiDAR frame while points of $\boldsymbol{\mathcal{E}}_m$ are registered in the global map, to find the nearest points of $\mathbf{p}_l$ in $\boldsymbol{\mathcal{E}}_m$, we need to project it to the global map by the following transformation.

$$\mathbf{p}_w = \mathbf{R}_k \mathbf{p}_l + \mathbf{t}_k \tag{4}$$

where $(\mathbf{R}_k, \mathbf{t}_k)$ is the LiDAR pose when the last point in current frame is sampled, and needs to be determined by the pose optimization. Here we use the LiDAR pose at the last point in a frame to represent the pose of the whole frame, and all points in that frame are projected to the global map using this pose. Also notice that the last point in the current frame is essentially the first point in the next frame.

Let $\mathbf{p}_i$ denote the i-th nearest points of $\mathbf{p}_w$ of $\boldsymbol{\mathcal{E}}_m$. To make sure that $\mathbf{p}_i$ is indeed on a line, we compute the mean $\boldsymbol{\mu}$ and covariance matrix $\mathbf{\Sigma}$ formed by the m nearest points of $\mathbf{p}_w$. We set m to 5 in our work. If the biggest of eigenvalue of $\mathbf{\Sigma}$ is three times larger than the second biggest eigenvalue, we assure that the nearest points of $\mathbf{p}_w$ form a line on which $\mathbf{p}_w$ should lie. The correspond point-to-edge residual is then computed as (Fig. 6 (b)) and then added to pose optimization.

$$\mathbf{r}_{p2e}\left(\mathbf{p}_w\right) = \frac{\left|\left(\mathbf{p}_w - \mathbf{p}_5\right) \times \left(\mathbf{p}_w - \mathbf{p}_1\right)\right|}{\left|\mathbf{p}_5 - \mathbf{p}_1\right|} \tag{5}$$

B. *Residual of plane-to-plane*

Similar to the edge feature points, for a point in the planar feature set $\boldsymbol{\mathcal{P}}_k$ of current frame, we find 5 nearest points in the planar feature set $\boldsymbol{\mathcal{P}}_m$ of the map (see Fig. 6(c)). We also assure these 5 nearest points are indeed within the same plane by computing their covariance matrix Σ. If the smallest eigenvalue of $\mathbf{\Sigma}$ is three times less than the second smallest eigenvalue, we compute the distance of the plane point in the current frame to the plane formed by the 5 points in the same plane, as follows, and add this residual to pose optimization.

$$\mathbf{r}_{p2p}\left(\mathbf{p}_w\right) = \frac{\left(\mathbf{p}_w - \mathbf{p}_1\right)^T \left(\left(\mathbf{p}_3 - \mathbf{p}_5\right) \times \left(\mathbf{p}_3 - \mathbf{p}_1\right)\right)}{\left|\left(\mathbf{p}_3 - \mathbf{p}_5\right) \times \left(\mathbf{p}_3 - \mathbf{p}_1\right)\right|} \tag{6}$$

C. *In-frame motion compensation*

As mentioned previously, the 3D points are sampled at different time of different poses (i.e., motor blur) as the LiDAR motion is continuously undergoing. To eliminate the effect of motion blur, we propose two methods as follows:

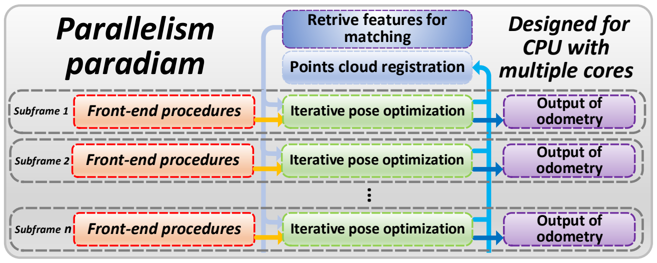

Parallelism paradiam
Retrive features for matching
Points cloud registration
Designed for CPU with multiple cores
Subframe 1
Front-end procedures
Iterative pose optimization
Output of odometry
Subframe 2
Front-end procedures
Iterative pose optimization
Output of odometry
Subframe n
Front-end procedures
Iterative pose optimization
Output of odometry

Fig. 7: Our parallel paradigm for CPU with multiple cores. Each sub-frame is matched with the global map independently on a dedicated thread and the resulting pose is published with the sub-frame timestamp as the odometry output. The matcheTd sub-frame is then registered to the global map and become a part of the map. Another dedicated thread receives the new registered sub-frame and build a KD three of the updated map to be used in the next frame.

1) Piecewise processing: A simple yet effective way to eliminate the effect of motion blur is piecewise processing. We divide an incoming frame into three sequential sub-frames. Then match these three sub-frames to the same map accumulated so far independently. During the scan matching of each sub-frame, all its points are projected to the global map using the LiDAR pose at the end point of that sub-frame. By doing so, the time interval of each sub-frame is 1/3 of the original frame. Although this method seems very simple, it works surprisingly well as shown in the results. Additionally, this piecewise processing has the benefits of utilizing the multi-core structure in modern CPUs by parallelizing the matching of each sub-frame (see Fig. 7).

2) Linear interpolation: Another commonly used motion compensation method is linear interpolation, as in [12]. Denote $(\mathbf{R}_k, \mathbf{t}_k)$ the LiDAR pose at the last point of the current frame and $(\mathbf{R}_{k-1}, \mathbf{t}_{k-1})$ the previous frame, $(\mathbf{R}_{k-1}^k, \mathbf{t}_{k-1}^k)$ the relative rotation and translation between the previous and the current frame, then:

$$\mathbf{R}_k = \mathbf{R}_{k-1}\mathbf{R}_{k-1}^k, \quad \mathbf{t}_k = \mathbf{R}_{k-1}\mathbf{t}_{k-1}^k + \mathbf{t}_{k-1}$$

Assume t_{k-1} is the sampling time of the last point in the previous frame. For any point sampled at time t in the current frame, we have $t \in [t_{k-1}, t_k]$. Compute $s = (t - t_{k-1})/(t_k - t_{k-1})$, then, the linearly interpolated pose at time t is:

$$\mathbf{R}_{k-1}^t = e^{\widehat{\boldsymbol{\omega}}\theta s}, \quad \mathbf{t}_{k-1}^t = s\mathbf{t}_{k-1}^k$$

where θ is the magnitude and $\boldsymbol{\omega}$ is the unit vector of of the rotation axis of $\mathbf{R}_{k-1}^k$, respectively. $\widehat{\boldsymbol{\omega}}$ is the skew symmetric matrix of $\boldsymbol{\omega}$. From the Rodrigue's formula, we have:

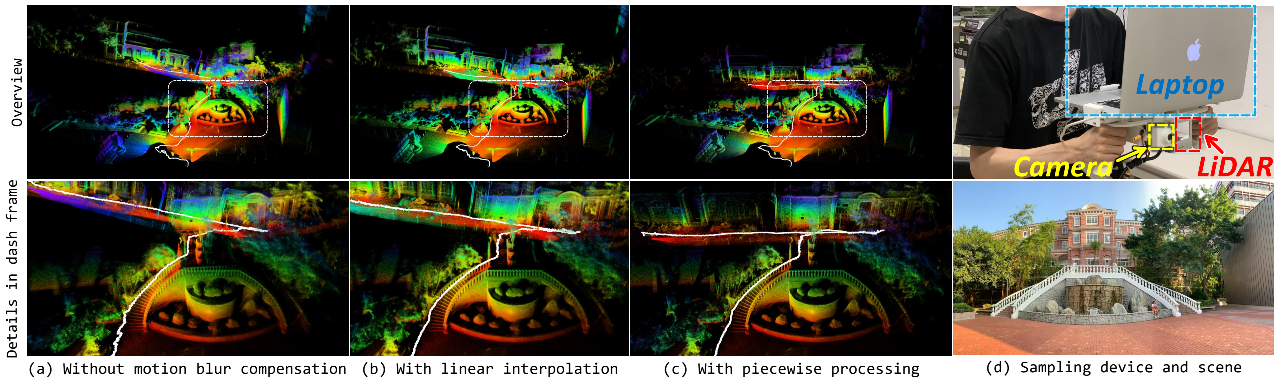
$$\mathbf{R}_{k-1}^t = \mathbf{I} + \widehat{\boldsymbol{\omega}}\sin(s\theta) + \widehat{\boldsymbol{\omega}}^2(1 - \cos(s\theta))$$

which implies that only $\sin(s\theta)$ and $\cos(s\theta)$ needs to be computed for each point of the current frame, while the rests remain constant. This saves some computations. With $\mathbf{R}_{k-1}^t$, the LiDAR pose at the current time is:

$$\mathbf{R}_t = \mathbf{R}_{k-1}\mathbf{R}_{k-1}^t, \quad \mathbf{t}_t = \mathbf{R}_{k-1}\mathbf{t}_{k-1}^t + \mathbf{t}_{k-1} \tag{7}$$

Then we can project the point at time t to the global map by the interpolated pose, as follows:

$$\mathbf{p}_w(t) = \mathbf{R}_t\mathbf{p}_l + \mathbf{t}_t \tag{8}$$



Overview
Details in dash frame
Laptop
Camera
LiDAR
(a) Without motion blur compensation
(b) With linear interpolation
(c) With piecewise processing
(d) Sampling device and scene

Fig. 8: The comparison of different motion compensation methods. The first column shows the results without any motion compensation, the second is with linear interpolation, and the third column is piecewise processing. The upper picture in the fourth column is our hand-held device for data collection, the lower picture is the RGB image of the mapped area.

Algorithm 1: Iterative LiDAR pose optimization

```
Input  : The edge set $\mathcal{E}_k$ and plane set $\mathcal{P}_k$ from the
         current (sub-) frame; The edge set $\mathcal{E}_m$ and
         plane set $\mathcal{P}_m$ from maps; The LiDAR pose
         of the previous frame $(\mathbf{R}_{k-1}, \mathbf{t}_{k-1})$.
Output: The pose of the current frame $(\mathbf{R}_k, \mathbf{t}_k)$.
Start  : $\mathbf{R}_k \leftarrow \mathbf{R}_{k-1}$, $\mathbf{t}_k \leftarrow \mathbf{t}_{k-1}$
for Iterative pose optimization is not converged do
    for $\mathbf{p}_l \in \mathcal{E}_k$ do
        Compute $\mathbf{p}_w$ via (4) (or (8)).
        Find 5 nearest points $\{\mathbf{p}_{1\sim5}\}$ of $\mathbf{p}_w$ in $\mathcal{E}_m$.
        if $\{\mathbf{p}_{1-5}\}$ are indeed in a line then
            Add point-to-edge residual $\mathbf{r}_{e2e}$ via (5).
    for $\mathbf{p}_l \in \mathcal{P}_k$ do
        Compute $\mathbf{p}_w$ via (4) (or (8)).
        Find 5 nearest points $\{\mathbf{p}_{1\sim5}\}$ of $\mathbf{p}_w$ in $\mathcal{P}_m$.
        if $\{\mathbf{p}_{1\sim5}\}$ are indeed a plane then
            Add point-to-plane residual $\mathbf{r}_{p2p}$ via (6).
    Perform pose optimization with 2 iterations.
    Recompute $\mathbf{r}_{e2e}$ and $\mathbf{r}_{p2p}$, then remove 20% of
     the biggest residual.
    for a maximal number of iterations do
        if the nonlinear optimization converges then
            Break;
```

D. Outliers rejection, dynamic objects filtering

To avoid moving object in real environments bringing down the accuracy of scan matching, we perform a dynamic objects filtering as follows: in each iteration of the iterative pose optimization, we refind the nearest neighbors of each feature point and add the edge-to-edge residual (5) and plane-to-plane residual (6) to the objective function, we first perform pose optimization with a small number of iterations (e.g., 2 used in our experiments). Using the optimization results, we compute the two residuals in (5) and (6), and remove the first 20% largest residuals. With the outliers removed, a full pose optimization is finally performed. The complete iterative pose optimization algorithm is summarized in Algorithm. 1.

V. Results

In this section, we evaluate our algorithm in three different folds. In Section. V-A, we evaluate the result of mapping by comparing the methods with different way of motion compensations. In Section. V-B, we evaluate the accuracy of position localization and rotation of our odometry by comparing it with GPS and motion capture systems, respectively. In Section. V-C, we evaluate the running performance by comparing the time consumption with currently available baseline.

A. Evaluation of mapping

The comparisons of the two motion compensation methods of are shown in Fig. 8, where we can see that, without any motion compensation (Fig. 8 (a)), the mapping is very blurry in local areas (e.g., stairs, railing) and distorted in larger scale (e.g., the building is curved). In contrast, with the motion compensation, both of the linear interpolation and piecewise processing effectively eliminate the motion blur, and the stair steps and railing are distinguishable one from another. However, the linear interpolation has a considerable long-term drift, as seen by the curved building in the upper figure of Fig. 8 (b). This is because the data are collected by hand-held devices and the movement could be quite jerky and cannot be accurately captured by simple linear interpolation.

B. Evaluation of odometry

We evaluate the localization of our algorithm by comparing with the measurement of GPS, shown in Fig. 9. We compute the distance of two positions of our odometry and then compare it with the measurement of GPS. The results on two datasets are 0.41% and 0.65%, respectively, implying that the localization is of high accuracy.

Furthermore, we evaluate the accuracy of rotation by comparing our result with "OptiTrack". The curves shown in Fig. 10 demonstrate that the trajectories of our odometry and motion capture system (mocap) are very close and the

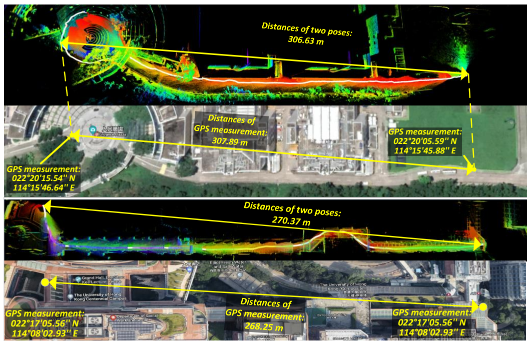

Distances of two poses: 306.63 m
Distances of GPS measurement: 307.89 m
GPS measurement: 022°20'05.59'' N 114°15'45.88'' E
GPS measurement: 022°20'15.54'' N 114°15'46.64'' E
Distances of two poses: 270.37 m
Distances of GPS measurement: 268.25 m
GPS measurement: 022°17'05.56'' N 114°08'02.93'' E
GPS measurement: 022°17'05.56'' N 114°08'02.93'' E

Fig. 9: The localization accuracy on two datasets: outdoor (upper) and indoor (lower). In each dataset, we compare our odometry results with Google maps and compute the traveled distance.

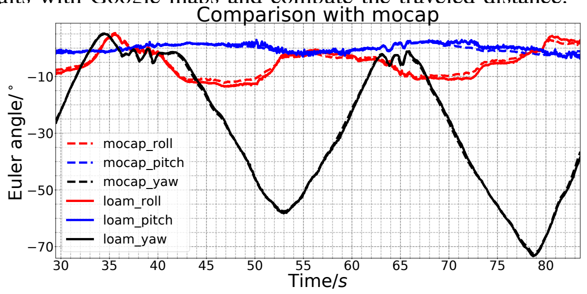


Fig. 10: The comparison between our results and motion capture (mocap) system, the dashed line is measured by mocap, and the solid line is the odometry output from our algorithm.

average error of Euler angles in all three directions is as low as 1.1°.

C. Evaluation of running performance

We evaluate the time consumption of our algorithm and the current baseline [4] (both algorithm eliminates the motion blur with piecewise processing and we replace the front-end procedure of A-LOAM to make it applicable with Livox MID40) on two platforms, the desktop PC (with *i7-9700K*) and onboard-computer (DJI manifold2[5] with *i7-8550U*). As shown in Table I, benefiting from the parallelization among sub-frame registration, as well as between feature matching and KD-tree building, our algorithms run *2∼3* times faster than the baseline.

VI. Conclusion and Discussion

This paper presented an odometry and mapping algorithm for LiDARs with small FOVs. The algorithm inherits the basic structure and techniques (e.g., feature extraction, mating, motion compensation by linear interpolation) of standard LOAM algorithm, but with several key new contributions, such as point selection, iterative pose optimization, and implementation parallelization. The developed algorithm has its odometry and mapping both running in real time (i.e., 20 Hz). While achieving a high level of accuracy in mapping and localization, the sequential scan matching is inherently drifting. Reducing this drift by using techniques like loop closure in concurrent work [20] and sliding window optimization will be further researched in the future.

[4]`https://github.com/HKUST-Aerial-Robotics/A-LOAM`
[5]`https://www.dji.com/cn/manifold-2`

	Desktop PC @4.0∼4.8 Ghz	Desktop PC parallel	Onboard PC @3.0∼3.5 Ghz	Onboard PC parallel
Ours	35.68 ms	17.24 ms	54.60 ms	32.54 ms
Baseline	109.00 ms	NaN	125.13 ms	NaN

TABLE I: The time consumption for each frame of our algorithm and the baseline[4], where “Desktop PC parallel” and “Onboard PC parallel” use 3 threads for point cloud registration.

References

[1] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, *et al.*, “Towards fully autonomous driving: Systems and algorithms,” in *2011 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2011, pp. 163–168.

[2] A. Bry, A. Bachrach, and N. Roy, “State estimation for aggressive flight in gps-denied environments using onboard sensing,” in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 1–8.

[3] F. Gao, W. Wu, W. Gao, and S. Shen, “Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments,” *Journal of Field Robotics*, vol. 36, no. 4, pp. 710–733, 2019.

[4] A. Nüchter, K. Lingemann, J. Hertzberg, and H. Surmann, “6d slam—3d mapping outdoor environments,” *Journal of Field Robotics*, vol. 24, no. 8-9, pp. 699–722, 2007.

[5] B. Schwarz, “Lidar: Mapping the world in 3d,” *Nature Photonics*, vol. 4, no. 7, p. 429, 2010.

[6] “Ces 2018: Waiting for the $100 lidar.” [Online]. Available: https://spectrum.ieee.org/cars-that-think/transportation/sensors/ces-2018-how-a-new-generation-lidars-is-redefining-the-car

[7] “Point cloud characteristics of livox-lidar.” [Online]. Available: https://www.livoxtech.com/3296f540ecf5458a8829e01cf429798e/downloads/Pointcloudcharacteristics.pdf

[8] P. J. Besl and N. D. McKay, “Method for registration of 3-d shapes,” in *Sensor fusion IV: control paradigms and data structures*, vol. 1611. International Society for Optics and Photonics, 1992, pp. 586–606.

[9] E. Mendes, P. Koch, and S. Lacroix, “Icp-based pose-graph slam,” in *2016 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR)*. IEEE, 2016, pp. 195–200.

[10] K. Pulli, “Multiview registration for large data sets,” in *Second International Conference on 3-D Digital Imaging and Modeling (Cat. No. PR00062)*. IEEE, 1999, pp. 160–168.

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

[12] J. Zhang and S. Singh, “Loam: Lidar odometry and mapping in real-time.” in *Robotics: Science and Systems*, vol. 2, 2014, p. 9.

[13] T. Shan and B. Englot, “Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain,” in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4758–4765.

[14] R. B. Rusu, Z. C. Marton, N. Blodow, and M. Beetz, “Learning informative point classes for the acquisition of object model maps,” in *2008 10th International Conference on Control, Automation, Robotics and Vision*. IEEE, 2008, pp. 643–650.

[15] R. B. Rusu, N. Blodow, and M. Beetz, “Fast point feature histograms (fpfh) for 3d registration,” in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 3212–3217.

[16] R. B. Rusu, G. Bradski, R. Thibaux, and J. Hsu, “Fast 3d recognition and pose using the viewpoint feature histogram,” in *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2010, pp. 2155–2162.

[17] S. Hong, H. Ko, and J. Kim, “Vicp: Velocity updating iterative closest point algorithm,” in *2010 IEEE International Conference on Robotics and Automation*. IEEE, 2010, pp. 1893–1898.

[18] D. Droeschel and S. Behnke, “Efficient continuous-time slam for 3d lidar-based online mapping,” in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 1–9.

[19] C. L. Gentil, T. Vidal-Calleja, and S. Huang, “In2laama: Inertial lidar localisation autocalibration and mapping,” *arXiv preprint arXiv:1905.09517*, 2019.

[20] J. Lin and F. Zhang, “A fast, complete, point cloud based loop closure for lidar odometry and mapping,” *arXiv preprint arXiv:1909.11811*, 2019.