LOAM: Lidar Odometry and Mapping in Real-time

Ji Zhang and Sanjiv Singh

***Abstract*— We propose a real-time method for odometry and mapping using range measurements from a 2-axis lidar moving in 6-DOF. The problem is hard because the range measurements are received at different times, and errors in motion estimation can cause mis-registration of the resulting point cloud. To date, coherent 3D maps can be built by off-line batch methods, often using loop closure to correct for drift over time. Our method achieves both low-drift and low-computational complexity without the need for high accuracy ranging or inertial measurements. The key idea in obtaining this level of performance is the division of the complex problem of simultaneous localization and mapping, which seeks to optimize a large number of variables simultaneously, by two algorithms. One algorithm performs odometry at a high frequency but low fidelity to estimate velocity of the lidar. Another algorithm runs at a frequency of an order of magnitude lower for fine matching and registration of the point cloud. Combination of the two algorithms allows the method to map in real-time. The method has been evaluated by a large set of experiments as well as on the KITTI odometry benchmark. The results indicate that the method can achieve accuracy at the level of state of the art offline batch methods.**

I. Introduction

3D mapping remains a popular technology [1]–[3]. Mapping with lidars is common as lidars can provide high frequency range measurements where errors are relatively constant irrespective of the distances measured. In the case that the only motion of the lidar is to rotate a laser beam, registration of the point cloud is simple. However, if the lidar itself is moving as in many applications of interest, accurate mapping requires knowledge of the lidar pose during continuous laser ranging. One common way to solve the problem is using independent position estimation (e.g. by a GPS/INS) to register the laser points into a fixed coordinate system. Another set of methods use odometry measurements such as from wheel encoders or visual odometry systems [4], [5] to register the laser points. Since odometry integrates small incremental motions over time, it is bound to drift and much attention is devoted to reduction of the drift (e.g. using loop closure).

Here we consider the case of creating maps with low-drift odometry using a 2-axis lidar moving in 6-DOF. A key advantage of using a lidar is its insensitivity to ambient lighting and optical texture in the scene. Recent developments in lidars have reduced their size and weight. The lidars can be held by a person who traverses an environment [6], or even attached to a micro aerial vehicle [7]. Since our method is intended to push issues related to minimizing drift in odometry estimation, it currently does not involve loop closure.

The method achieves both low-drift and low-computational complexity without the need for high accuracy ranging or inertial measurements. The key idea in obtaining this level of performance is the division of the typically complex problem of simultaneous localization and mapping (SLAM) [8], which seeks to optimize a large number of variables simultaneously, by two algorithms. One algorithm performs odometry at a high frequency but low fidelity to estimate velocity of the lidar. Another algorithm runs at a frequency of an order of magnitude lower for fine matching and registration of the point cloud. Although unnecessary, if an IMU is available, a motion prior can be provided to help account for high frequency motion. Specifically, both algorithms extract feature points located on sharp edges and planar surfaces, and match the feature points to edge line segments and planar surface patches, respectively. In the odometry algorithm, correspondences of the feature points are found by ensuring fast computation. In the mapping algorithm, the correspondences are determined by examining geometric distributions of local point clusters, through the associated eigenvalues and eigenvectors.

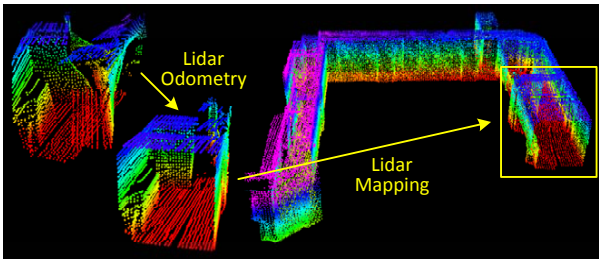


Fig. 1. The method aims at motion estimation and mapping using a moving 2-axis lidar. Since the laser points are received at different times, distortion is present in the point cloud due to motion of the lidar (shown in the left lidar cloud). Our proposed method decomposes the problem by two algorithms running in parallel. An odometry algorithm estimates velocity of the lidar and corrects distortion in the point cloud, then, a mapping algorithm matches and registers the point cloud to create a map. Combination of the two algorithms ensures feasibility of the problem to be solved in real-time.

By decomposing the original problem, an easier problem is solved first as online motion estimation. After which, mapping is conducted as batch optimization (similar to iterative closest point (ICP) methods [9]) to produce high-precision motion estimates and maps. The parallel algorithm structure ensures feasibility of the problem to be solved in real-time. Further, since the motion estimation is conducted at a higher frequency, the mapping is given plenty of time to enforce accuracy. When running at a lower frequency, the mapping algorithm is able to incorporate a large number of feature points and use sufficiently many iterations for convergence.

II. Related Work

Lidar has become a useful range sensor in robot navigation [10]. For localization and mapping, most applications use 2D lidars [11]. When the lidar scan rate is high compared to its extrinsic motion, motion distortion within the scans can

J. Zhang and S. Singh are with the Robotics Institute at Carnegie Mellon University. Emails: zhangji@cmu.edu and ssingh@cmu.edu.

The paper is based upon work supported by the National Science Foundation under Grant No. IIS-1328930.

often be neglected. In this case, standard ICP methods [12] can be used to match laser returns between different scans. Additionally, a two-step method is proposed to remove the distortion [13]: an ICP based velocity estimation step is followed by a distortion compensation step, using the computed velocity. A similar technique is also used to compensate for the distortion introduced by a single-axis 3D lidar [14]. However, if the scanning motion is relatively slow, motion distortion can be severe. This is especially the case when a 2-axis lidar is used since one axis is typically much slower than the other. Often, other sensors are used to provide velocity measurements, with which, the distortion can be removed. For example, the lidar cloud can be registered by state estimation from visual odometry integrated with an IMU [15]. When multiple sensors such as a GPS/INS and wheel encoders are available concurrently, the problem is usually solved through an extended Kalman filer [16] or a particle filter [1]. These methods can create maps in real-time to assist path planning and collision avoidance in robot navigation.

If a 2-axis lidar is used without aiding from other sensors, motion estimation and distortion correction become one problem. A method used by Barfoot et al. is to create visual images from laser intensity returns, and match visually distinct features [17] between images to recover motion of a ground vehicle [18]–[21]. The vehicle motion is modeled as constant velocity in [18], [19] and with Gaussian processes in [20], [21]. Our method uses a similar linear motion model as [18], [19] in the odometry algorithm, but with different types of features. The methods [18]–[21] involve visual features from intensity images and require dense point cloud. Our method extracts and matches geometric features in Cartesian space and has a lower requirement on the cloud density.

The approach closest to ours is that of Bosse and Zlot [3], [6], [22]. They use a 2-axis lidar to obtain point cloud which is registered by matching geometric structures of local point clusters [22]. Further, they use multiple 2-axis lidars to map an underground mine [3]. This method incorporates an IMU and uses loop closure to create large maps. Proposed by the same authors, Zebedee is a mapping device composed of a 2D lidar and an IMU attached to a hand-bar through a spring [6]. Mapping is conducted by hand nodding the device. The trajectory is recovered by a batch optimization method that processes segmented datasets with boundary constraints added between the segments. In this method, the measurements of the IMU are used to register the laser points and the optimization is used to correct the IMU biases. In essence, Bosse and Zlot's methods require batch processing to develop accurate maps and therefore are unsuitable for applications where maps are needed in real-time. In comparison, the proposed method in real-time produces maps that are qualitatively similar to those by Bosse and Zlot. The distinction is that our method can provide motion estimates for guidance of an autonomous vehicle. Further, the method takes advantage of the lidar scan pattern and point cloud distribution. Feature matching is realized ensuring computation speed and accuracy in the odometry and mapping algorithms, respectively.

III. Notations and Task Description

The problem addressed in this paper is to perform ego-motion estimation with point cloud perceived by a 3D lidar, and build a map for the traversed environment. We assume that the lidar is pre-calibrated. We also assume that the angular and linear velocities of the lidar are smooth and continuous over time, without abrupt changes. The second assumption will be released by usage of an IMU, in Section VII-B.

As a convention in this paper, we use right uppercase superscription to indicate the coordinate systems. We define a sweep as the lidar completes one time of scan coverage. We use right subscription k, $k \in Z^+$ to indicate the sweeps, and $\mathcal{P}_k$ to indicate the point cloud perceived during sweep k. Let us define two coordinate systems as follows.

- Lidar coordinate system $\{L\}$ is a 3D coordinate system with its origin at the geometric center of the lidar. The x-axis is pointing to the left, the y-axis is pointing upward, and the z-axis is pointing forward. The coordinates of a point i, $i \in \mathcal{P}_k$, in $\{L_k\}$ are denoted as $\boldsymbol{X}^L_{(k,i)}$.
- World coordinate system $\{W\}$ is a 3D coordinate system coinciding with $\{L\}$ at the initial position. The coordinates of a point i, $i \in \mathcal{P}_k$, in $\{W_k\}$ are $\boldsymbol{X}^W_{(k,i)}$.

With assumptions and notations made, our lidar odometry and mapping problem can be defined as

Problem: Given a sequence of lidar cloud $\mathcal{P}_k$, $k \in Z^+$, compute the ego-motion of the lidar during each sweep k, and build a map with $\mathcal{P}_k$ for the traversed environment.

IV. System Overview

A. Lidar Hardware

The study of this paper is validated on, but not limited to a custom built 3D lidar based on a Hokuyo UTM-30LX laser scanner. Through the paper, we will use data collected from the lidar to illustrate the method. The laser scanner has a field of view of $180°$ with $0.25°$ resolution and 40 lines/sec scan rate. The laser scanner is connected to a motor, which is controlled to rotate at an angular speed of $180°/s$ between $-90°$ and $90°$ with the horizontal orientation of the laser scanner as zero. With this particular unit, a sweep is a rotation from $-90°$ to $90°$ or in the inverse direction (lasting for 1s). Here, note that for a continuous spinning lidar, a sweep is simply a semi-spherical rotation. An onboard encoder measures the motor rotation angle with a resolution of $0.25°$, with which, the laser points are projected into the lidar coordinates, $\{L\}$.

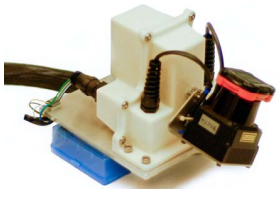


Fig. 2. The 3D lidar used in this study consists of a Hokuyo laser scanner driven by a motor for rotational motion, and an encoder that measures the rotation angle. The laser scanner has a field of view of $180°$ with a resolution of $0.25°$. The scan rate is 40 lines/sec. The motor is controlled to rotate from $-90°$ to $90°$ with the horizontal orientation of the laser scanner as zero.

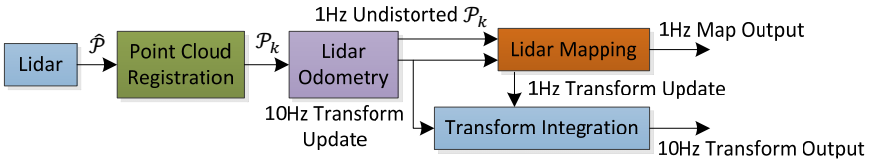


Fig. 3. Block diagram of the lidar odometry and mapping software system.

B. Software System Overview

Fig. 3 shows a diagram of the software system. Let $\hat{\mathcal{P}}$ be the points received in a laser scan. During each sweep, $\hat{\mathcal{P}}$ is registered in $\{L\}$. The combined point cloud during sweep k forms $\mathcal{P}_k$. Then, $\mathcal{P}_k$ is processed in two algorithms. The lidar odometry takes the point cloud and computes the motion of the lidar between two consecutive sweeps. The estimated motion is used to correct distortion in $\mathcal{P}_k$. The algorithm runs at a frequency around 10Hz. The outputs are further processed by lidar mapping, which matches and registers the undistorted cloud onto a map at a frequency of 1Hz. Finally, the pose transforms published by the two algorithms are integrated to generate a transform output around 10Hz, regarding the lidar pose with respect to the map. Section V and VI present the blocks in the software diagram in detail.

V. Lidar Odometry

A. Feature Point Extraction

We start with extraction of feature points from the lidar cloud, $\mathcal{P}_k$. The lidar presented in Fig. 2 naturally generates unevenly distributed points in $\mathcal{P}_k$. The returns from the laser scanner has a resolution of $0.25°$ within a scan. These points are located on a scan plane. However, as the laser scanner rotates at an angular speed of $180°/s$ and generates scans at 40Hz, the resolution in the perpendicular direction to the scan planes is $180°/40 = 4.5°$. Considering this fact, the feature points are extracted from $\mathcal{P}_k$ using only information from individual scans, with co-planar geometric relationship.

We select feature points that are on sharp edges and planar surface patches. Let i be a point in $\mathcal{P}_k$, $i \in \mathcal{P}_k$, and let $\mathcal{S}$ be the set of consecutive points of i returned by the laser scanner in the same scan. Since the laser scanner generates point returns in CW or CCW order, $\mathcal{S}$ contains half of its points on each side of i and $0.25°$ intervals between two points. Define a term to evaluate the smoothness of the local surface,

$$c = \frac{1}{|\mathcal{S}| \cdot ||\boldsymbol{X}^L_{(k,i)}||} || \sum_{j \in \mathcal{S}, j \neq i} (\boldsymbol{X}^L_{(k,i)} - \boldsymbol{X}^L_{(k,j)}) ||. \quad (1)$$

The points in a scan are sorted based on the c values, then feature points are selected with the maximum c's, namely, edge points, and the minimum c's, namely planar points. To evenly distribute the feature points within the environment, we separate a scan into four identical subregions. Each subregion can provide maximally 2 edge points and 4 planar points. A point i can be selected as an edge or a planar point only if its c value is larger or smaller than a threshold, and the number of selected points does not exceed the maximum.

While selecting feature points, we want to avoid points whose surrounded points are selected, or points on local planar surfaces that are roughly parallel to the laser beams (point B in Fig. 4(a)). These points are usually considered as unreliable. Also, we want to avoid points that are on boundary of occluded regions [23]. An example is shown in Fig. 4(b). Point A is an edge point in the lidar cloud because its connected surface (the dotted line segment) is blocked by another object. However, if the lidar moves to another point of view, the occluded region can change and become observable. To avoid the aforementioned points to be selected, we find again the set of points $\mathcal{S}$. A point i can be selected only if $\mathcal{S}$ does not form a surface patch that is roughly parallel to the laser beam, and there is no point in $\mathcal{S}$ that is disconnected from i by a gap in the direction of the laser beam and is at the same time closer to the lidar then point i (e.g. point B in Fig. 4(b)).

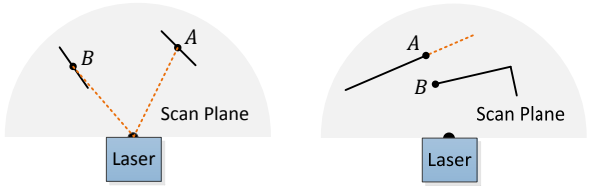
In summary, the feature points are selected as edge points starting from the maximum c value, and planar points starting from the minimum c value, and if a point is selected,

- The number of selected edge points or planar points cannot exceed the maximum of the subregion, and
- None of its surrounding point is already selected, and
- It cannot be on a surface patch that is roughly parallel to the laser beam, or on boundary of an occluded region.

An example of extracted feature points from a corridor scene is shown in Fig. 5. The edge points and planar points are labeled in yellow and red colors, respectively.

B. Finding Feature Point Correspondence

The odometry algorithm estimates motion of the lidar within a sweep. Let t_k be the starting time of a sweep k. At the end of each sweep, the point cloud perceived during the sweep,



(a) (b)

Fig. 4. (a) The solid line segments represent local surface patches. Point A is on a surface patch that has an angle to the laser beam (the dotted orange line segments). Point B is on a surface patch that is roughly parallel to the laser beam. We treat B as a unreliable laser return and do not select it as a feature point. (b) The solid line segments are observable objects to the laser. Point A is on the boundary of an occluded region (the dotted orange line segment), and can be detected as an edge point. However, if viewed from a different angle, the occluded region can change and become observable. We do not treat A as a salient edge point or select it as a feature point.

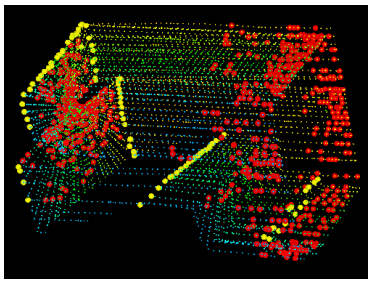


Fig. 5. An example of extracted edge points (yellow) and planar points (red) from lidar cloud taken in a corridor. Meanwhile, the lidar moves toward the wall on the left side of the figure at a speed of 0.5m/s, this results in motion distortion on the wall.

$\mathcal{P}_k$, is reprojected to time stamp t_{k+1}, illustrated in Fig. 6. We denote the reprojected point cloud as $\bar{\mathcal{P}}_k$. During the next sweep $k+1$, $\bar{\mathcal{P}}_k$ is used together with the newly received point cloud, $\mathcal{P}_{k+1}$, to estimate the motion of the lidar.

Let us assume that both $\bar{\mathcal{P}}_k$ and $\mathcal{P}_{k+1}$ are available for now, and start with finding correspondences between the two lidar clouds. With $\mathcal{P}_{k+1}$, we find edge points and planar points from the lidar cloud using the methodology discussed in the last section. Let $\mathcal{E}_{k+1}$ and $\mathcal{H}_{k+1}$ be the sets of edge points and planar points, respectively. We will find edge lines from $\bar{\mathcal{P}}_k$ as the correspondences for the points in $\mathcal{E}_{k+1}$, and planar patches as the correspondences for those in $\mathcal{H}_{k+1}$.

Note that at the beginning of sweep $k+1$, $\mathcal{P}_{k+1}$ is an empty set, which grows during the course of the sweep as more points are received. The lidar odometry recursively estimates the 6-DOF motion during the sweep, and gradually includes more points as $\mathcal{P}_{k+1}$ increases. At each iteration, $\mathcal{E}_{k+1}$ and $\mathcal{H}_{k+1}$ are reprojected to the beginning of the sweep using the currently estimated transform. Let $\tilde{\mathcal{E}}_{k+1}$ and $\tilde{\mathcal{H}}_{k+1}$ be the reprojected point sets. For each point in $\tilde{\mathcal{E}}_{k+1}$ and $\tilde{\mathcal{H}}_{k+1}$, we are going to find the closest neighbor point in $\bar{\mathcal{P}}_k$. Here, $\bar{\mathcal{P}}_k$ is stored in a 3D KD-tree [24] for fast index.

Fig. 7(a) represents the procedure of finding an edge line as the correspondence of an edge point. Let i be a point in $\tilde{\mathcal{E}}_{k+1}$, $i \in \tilde{\mathcal{E}}_{k+1}$. The edge line is represented by two points. Let j be the closest neighbor of i in $\bar{\mathcal{P}}_k$, $j \in \bar{\mathcal{P}}_k$, and let l be the closest neighbor of i in the two consecutive scans to the scan of j. $(j,\ l)$ forms the correspondence of i. Then, to verify both j and l are edge points, we check the smoothness of the local surface based on (1). Here, we particularly require that j and l are from different scans considering that a single scan cannot contain more than one points from the same edge line. There is only one exception where the edge line is on the

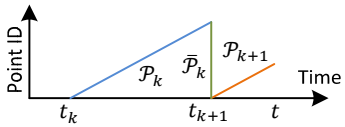


Fig. 6. Reprojecting point cloud to the end of a sweep. The blue colored line segment represents the point cloud perceived during sweep k, $\mathcal{P}_k$. At the end of sweep k, $\mathcal{P}_k$ is reprojected to time stamp t_{k+1} to obtain $\bar{\mathcal{P}}_k$ (the green colored line segment). Then, during sweep $k+1$, $\bar{\mathcal{P}}_k$ and the newly perceived point cloud $\mathcal{P}_{k+1}$ (the orange colored line segment) are used together to estimate the lidar motion.

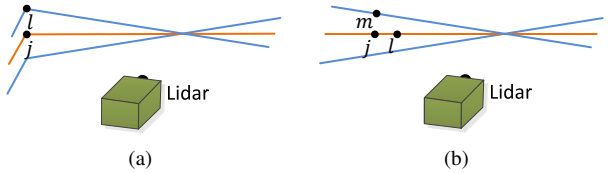


Fig. 7. Finding an edge line as the correspondence for an edge point in $\tilde{\mathcal{E}}_{k+1}$ (a), and a planar patch as the correspondence for a planar point in $\tilde{\mathcal{H}}_{k+1}$ (b). In both (a) and (b), j is the closest point to the feature point, found in $\bar{\mathcal{P}}_k$. The orange colored lines represent the same scan of j, and the blue colored lines are the two consecutive scans. To find the edge line correspondence in (a), we find another point, l, on the blue colored lines, and the correspondence is represented as $(j,\ l)$. To find the planar patch correspondence in (b), we find another two points, l and m, on the orange and blue colored lines, respectively. The correspondence is $(j,\ l,\ m)$.

scan plane. If so, however, the edge line is degenerated and appears as a straight line on the scan plane, and feature points on the edge line should not be extracted in the first place.

Fig. 7(b) shows the procedure of finding a planar patch as the correspondence of a planar point. Let i be a point in $\tilde{\mathcal{H}}_{k+1}$, $i \in \tilde{\mathcal{H}}_{k+1}$. The planar patch is represented by three points. Similar to the last paragraph, we find the closest neighbor of i in $\bar{\mathcal{P}}_k$, denoted as j. Then, we find another two points, l and m, as the closest neighbors of i, one in the same scan of j, and the other in the two consecutive scans to the scan of j. This guarantees that the three points are non-collinear. To verify that j, l, and m are all planar points, we check again the smoothness of the local surface based on (1).

With the correspondences of the feature points found, now we derive expressions to compute the distance from a feature point to its correspondence. We will recover the lidar motion by minimizing the overall distances of the feature points in the next section. We start with edge points. For a point $i \in \tilde{\mathcal{E}}_{k+1}$, if $(j,\ l)$ is the corresponding edge line, $j, l \in \bar{\mathcal{P}}_k$, the point to line distance can be computed as

$$d_{\mathcal{E}} = \frac{\left|(\tilde{\boldsymbol{X}}^L_{(k+1,i)} - \bar{\boldsymbol{X}}^L_{(k,j)}) \times (\tilde{\boldsymbol{X}}^L_{(k+1,i)} - \bar{\boldsymbol{X}}^L_{(k,l)})\right|}{\left|\bar{\boldsymbol{X}}^L_{(k,j)} - \bar{\boldsymbol{X}}^L_{(k,l)}\right|}, \quad (2)$$

where $\tilde{\boldsymbol{X}}^L_{(k+1,i)}$, $\bar{\boldsymbol{X}}^L_{(k,j)}$, and $\bar{\boldsymbol{X}}^L_{(k,l)}$ are the coordinates of points i, j, and l in $\{L\}$, respectively. Then, for a point $i \in \tilde{\mathcal{H}}_{k+1}$, if $(j,\ l,\ m)$ is the corresponding planar patch, $j, l, m \in \bar{\mathcal{P}}_k$, the point to plane distance is

$$d_{\mathcal{H}} = \frac{\left|(\tilde{\boldsymbol{X}}^L_{(k+1,i)} - \bar{\boldsymbol{X}}^L_{(k,j)}) \cdot ((\bar{\boldsymbol{X}}^L_{(k,j)} - \bar{\boldsymbol{X}}^L_{(k,l)}) \times (\bar{\boldsymbol{X}}^L_{(k,j)} - \bar{\boldsymbol{X}}^L_{(k,m)}))\right|}{\left|(\bar{\boldsymbol{X}}^L_{(k,j)} - \bar{\boldsymbol{X}}^L_{(k,l)}) \times (\bar{\boldsymbol{X}}^L_{(k,j)} - \bar{\boldsymbol{X}}^L_{(k,m)})\right|}, \quad (3)$$

where $\bar{\boldsymbol{X}}^L_{(k,m)}$ is the coordinates of point m in $\{L\}$.

C. Motion Estimation

The lidar motion is modeled with constant angular and linear velocities during a sweep. This allows us to linear interpolate the pose transform within a sweep for the points that are received at different times. Let t be the current time stamp, and recall that t_{k+1} is the starting time of sweep

$k+1$. Let $\boldsymbol{T}^L_{k+1}$ be the lidar pose transform between $[t_{k+1},\ t]$. $\boldsymbol{T}^L_{k+1}$ contains rigid motion of the lidar in 6-DOF, $\boldsymbol{T}^L_{k+1} = [t_x,\ t_y,\ t_z,\ \theta_x,\ \theta_y,\ \theta_z]^T$, where t_x, t_y, and t_z are translations along the x-, y-, and z- axes of $\{L\}$, respectively, and θ_x, θ_y, and θ_z are rotation angles, following the right-hand rule. Given a point i, $i \in \mathcal{P}_{k+1}$, let t_i be its time stamp, and let $\boldsymbol{T}^L_{(k+1,i)}$ be the pose transform between $[t_{k+1},\ t_i]$. $\boldsymbol{T}^L_{(k+1,i)}$ can be computed by linear interpolation of $\boldsymbol{T}^L_{k+1}$,

$$\boldsymbol{T}^L_{(k+1,i)} = \frac{t_i - t_{k+1}}{t - t_{k+1}} \boldsymbol{T}^L_{k+1}. \tag{4}$$

Recall that $\mathcal{E}_{k+1}$ and $\mathcal{H}_{k+1}$ are the sets of edge points and planar points extracted from $\mathcal{P}_{k+1}$, and $\tilde{\mathcal{E}}_{k+1}$ and $\tilde{\mathcal{H}}_{k+1}$ are the sets of points reprojected to the beginning of the sweep, t_{k+1}. To solved the lidar motion, we need to establish a geometric relationship between $\mathcal{E}_{k+1}$ and $\tilde{\mathcal{E}}_{k+1}$, or $\mathcal{H}_{k+1}$ and $\tilde{\mathcal{H}}_{k+1}$. Using the transform in (4), we can derive,

$$\boldsymbol{X}^L_{(k+1,i)} = \mathbf{R}\tilde{\boldsymbol{X}}^L_{(k+1,i)} + \boldsymbol{T}^L_{(k+1,i)}(1:3), \tag{5}$$

where $\boldsymbol{X}^L_{(k+1,i)}$ is the coordinates of a point i in $\mathcal{E}_{k+1}$ or $\mathcal{H}_{k+1}$, $\tilde{\boldsymbol{X}}^L_{(k+1,i)}$ is the corresponding point in $\tilde{\mathcal{E}}_{k+1}$ or $\tilde{\mathcal{H}}_{k+1}$, $\boldsymbol{T}^L_{(k+1,i)}(a:b)$ is the a-th to b-th entries of $\boldsymbol{T}^L_{(k+1,i)}$, and $\mathbf{R}$ is a rotation matrix defined by the Rodrigues formula [25],

$$\mathbf{R} = e^{\hat{\omega}\theta} = \mathbf{I} + \hat{\omega}\sin\theta + \hat{\omega}^2(1-\cos\theta). \tag{6}$$

In the above equation, θ is the magnitude of the rotation,

$$\theta = ||\boldsymbol{T}^L_{(k+1,i)}(4:6)||, \tag{7}$$

ω is a unit vector representing the rotation direction,

$$\omega = \boldsymbol{T}^L_{(k+1,i)}(4:6)/||\boldsymbol{T}^L_{(k+1,i)}(4:6)||, \tag{8}$$

and $\hat{\omega}$ is the skew symmetric matrix of ω [25].

Recall that (2) and (3) compute the distances between points in $\tilde{\mathcal{E}}_{k+1}$ and $\tilde{\mathcal{H}}_{k+1}$ and their correspondences. Combining (2) and (4)-(8), we can derive a geometric relationship between an edge point in $\mathcal{E}_{k+1}$ and the corresponding edge line,

$$f_{\mathcal{E}}(\boldsymbol{X}^L_{(k+1,i)}, \boldsymbol{T}^L_{k+1}) = d_{\mathcal{E}},\ i \in \mathcal{E}_{k+1}. \tag{9}$$

Similarly, combining (3) and (4)-(8), we can establish another geometric relationship between a planar point in $\mathcal{H}_{k+1}$ and the corresponding planar patch,

$$f_{\mathcal{H}}(\boldsymbol{X}^L_{(k+1,i)}, \boldsymbol{T}^L_{k+1}) = d_{\mathcal{H}},\ i \in \mathcal{H}_{k+1}. \tag{10}$$

Finally, we solve the lidar motion with the Levenberg-Marquardt method [26]. Stacking (9) and (10) for each feature point in $\mathcal{E}_{k+1}$ and $\mathcal{H}_{k+1}$, we obtain a nonlinear function,

$$\boldsymbol{f}(\boldsymbol{T}^L_{k+1}) = \boldsymbol{d}, \tag{11}$$

where each row of $\boldsymbol{f}$ corresponds to a feature point, and $\boldsymbol{d}$ contains the corresponding distances. We compute the Jacobian matrix of $\boldsymbol{f}$ with respect to $\boldsymbol{T}^L_{k+1}$, denoted as $\mathbf{J}$, where $\mathbf{J} = \partial f / \partial \boldsymbol{T}^L_{k+1}$. Then, (11) can be solved through nonlinear iterations by minimizing $\boldsymbol{d}$ toward zero,

$$\boldsymbol{T}^L_{k+1} \leftarrow \boldsymbol{T}^L_{k+1} - (\mathbf{J}^T\mathbf{J} + \lambda \mathrm{diag}(\mathbf{J}^{\mathrm{T}}\mathbf{J}))^{-1}\mathbf{J}^{\mathrm{T}}\boldsymbol{d}. \tag{12}$$

λ is a factor determined by the Levenberg-Marquardt method.

Algorithm 1: Lidar Odometry

1 **input** : $\bar{\mathcal{P}}_k$, $\mathcal{P}_{k+1}$, $\boldsymbol{T}^L_{k+1}$ from the last recursion
2 **output** : $\bar{\mathcal{P}}_{k+1}$, newly computed $\boldsymbol{T}^L_{k+1}$
3 **begin**
4 **if** at the beginning of a sweep **then**
5 $\boldsymbol{T}^L_{k+1} \leftarrow \mathbf{0}$;
6 **end**
7 Detect edge points and planar points in $\mathcal{P}_{k+1}$, put the points in $\mathcal{E}_{k+1}$ and $\mathcal{H}_{k+1}$, respectively;
8 **for** a number of iterations **do**
9 **for** each edge point in $\mathcal{E}_{k+1}$ **do**
10 Find an edge line as the correspondence, then compute point to line distance based on (9) and stack the equation to (11);
11 **end**
12 **for** each planar point in $\mathcal{H}_{k+1}$ **do**
13 Find a planar patch as the correspondence, then compute point to plane distance based on (10) and stack the equation to (11);
14 **end**
15 Compute a bisquare weight for each row of (11);
16 Update $\boldsymbol{T}^L_{k+1}$ for a nonlinear iteration based on (12);
17 **if** the nonlinear optimization converges **then**
18 Break;
19 **end**
20 **end**
21 **if** at the end of a sweep **then**
22 Reproject each point in $\mathcal{P}_{k+1}$ to t_{k+2} and form $\bar{\mathcal{P}}_{k+1}$;
23 Return $\boldsymbol{T}^L_{k+1}$ and $\bar{\mathcal{P}}_{k+1}$;
24 **end**
25 **else**
26 Return $\boldsymbol{T}^L_{k+1}$;
27 **end**
28 **end**

D. Lidar Odometry Algorithm

The lidar odometry algorithm is shown in Algorithm 1. The algorithm takes as inputs the point cloud from the last sweep, $\bar{\mathcal{P}}_k$, the growing point cloud of the current sweep, $\mathcal{P}_{k+1}$, and the pose transform from the last recursion, $\boldsymbol{T}^L_{k+1}$. If a new sweep is started, $\boldsymbol{T}^L_{k+1}$ is set to zero (line 4-6). Then, the algorithm extracts feature points from $\mathcal{P}_{k+1}$ to construct $\mathcal{E}_{k+1}$ and $\mathcal{H}_{k+1}$ in line 7. For each feature point, we find its correspondence in $\bar{\mathcal{P}}_k$ (line 9-19). The motion estimation is adapted to a robust fitting [27]. In line 15, the algorithm assigns a bisquare weight for each feature point. The feature points that have larger distances to their correspondences are assigned with smaller weights, and the feature points with distances larger than a threshold are considered as outliers and assigned with zero weights. Then, in line 16, the pose transform is updated for one iteration. The nonlinear optimization terminates if convergence is found, or the maximum iteration number is met. If the algorithm reaches the end of a sweep, $\mathcal{P}_{k+1}$ is reprojected to time stamp t_{k+2} using the estimated motion during the sweep. Otherwise, only the transform $\boldsymbol{T}^L_{k+1}$ is returned for the next round of recursion.

VI. Lidar Mapping

The mapping algorithm runs at a lower frequency then the odometry algorithm, and is called only once per sweep. At the end of sweep $k+1$, the lidar odometry generates a undistorted point cloud, $\bar{\mathcal{P}}_{k+1}$, and simultaneously a pose

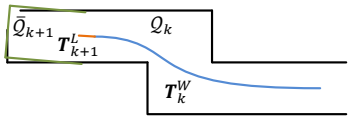


Fig. 8. Illustration of mapping process. The blue colored curve represents the lidar pose on the map, $\boldsymbol{T}^W_k$, generated by the mapping algorithm at sweep k. The orange color curve indicates the lidar motion during sweep $k+1$, $\boldsymbol{T}^L_{k+1}$, computed by the odometry algorithm. With $\boldsymbol{T}^W_k$ and $\boldsymbol{T}^L_{k+1}$, the undistorted point cloud published by the odometry algorithm is projected onto the map, denoted as $\bar{\mathcal{Q}}_{k+1}$ (the green colored line segments), and matched with the existing cloud on the map, $\mathcal{Q}_k$ (the black colored line segments).

transform, $\boldsymbol{T}^L_{k+1}$, containing the lidar motion during the sweep, between $[t_{k+1},\ t_{k+2}]$. The mapping algorithm matches and registers $\bar{\mathcal{P}}_{k+1}$ in the world coordinates, $\{W\}$, illustrated in Fig. 8. To explain the procedure, let us define $\mathcal{Q}_k$ as the point cloud on the map, accumulated until sweep k, and let $\boldsymbol{T}^W_k$ be the pose of the lidar on the map at the end of sweep k, t_{k+1}. With the outputs from the lidar odometry, the mapping algorithm extents $\boldsymbol{T}^W_k$ for one sweep from t_{k+1} to t_{k+2}, to obtain $\boldsymbol{T}^W_{k+1}$, and projects $\bar{\mathcal{P}}_{k+1}$ into the world coordinates, $\{W\}$, denoted as $\bar{\mathcal{Q}}_{k+1}$. Next, the algorithm matches $\bar{\mathcal{Q}}_{k+1}$ with $\mathcal{Q}_k$ by optimizing the lidar pose $\boldsymbol{T}^W_{k+1}$.

The feature points are extracted in the same way as in Section V-A, but 10 times of feature points are used. To find correspondences for the feature points, we store the point cloud on the map, $\mathcal{Q}_k$, in 10m cubic areas. The points in the cubes that intersect with $\bar{\mathcal{Q}}_{k+1}$ are extracted and stored in a 3D KD-tree [24]. We find the points in $\mathcal{Q}_k$ within a certain region around the feature points. Let $\mathcal{S}'$ be a set of surrounding points. For an edge point, we only keep points on edge lines in $\mathcal{S}'$, and for a planar point, we only keep points on planar patches. Then, we compute the covariance matrix of $\mathcal{S}'$, denoted as $\mathbf{M}$, and the eigenvalues and eigenvectors of $\mathbf{M}$, denoted as $\boldsymbol{V}$ and $\boldsymbol{E}$, respectively. If $\mathcal{S}'$ is distributed on an edge line, $\boldsymbol{V}$ contains one eigenvalue significantly larger than the other two, and the eigenvector in $\boldsymbol{E}$ associated with the largest eigenvalue represents the orientation of the edge line. On the other hand, if $\mathcal{S}'$ is distributed on a planar patch, $\boldsymbol{V}$ contains two large eigenvalues with the third one significantly smaller, and the eigenvector in $\boldsymbol{E}$ associated with the smallest eigenvalue denotes the orientation of the planar patch. The position of the edge line or the planar patch is determined by passing through the geometric center of $\mathcal{S}'$.

To compute the distance from a feature point to its correspondence, we select two points on an edge line, and three points on a planar patch. This allows the distances to be computed using the same formulations as (2) and (3). Then, an equation is derived for each feature point as (9) or (10), but different in that all points in $\bar{\mathcal{Q}}_{k+1}$ share the same time stamp, t_{k+2}. The nonlinear optimization is solved again by a robust fitting [27] through the Levenberg-Marquardt method [26], and $\bar{\mathcal{Q}}_{k+1}$ is registered on the map. To evenly distribute the points, the map cloud is downsized by a voxel grid filter [28] with the voxel size of 5cm cubes.

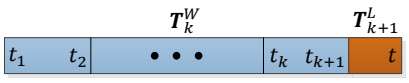


Fig. 9. Integration of pose transforms. The blue colored region illustrates the lidar pose from the mapping algorithm, $\boldsymbol{T}^W_k$, generated once per sweep. The orange colored region is the lidar motion within the current sweep, $\boldsymbol{T}^L_{k+1}$, computed by the odometry algorithm. The motion estimation of the lidar is the combination of the two transforms, at the same frequency as $\boldsymbol{T}^L_{k+1}$.

Integration of the pose transforms is illustrated in Fig. 9. The blue colored region represents the pose output from the lidar mapping, $\boldsymbol{T}^W_k$, generated once per sweep. The orange colored region represents the transform output from the lidar odometry, $\boldsymbol{T}^L_{k+1}$, at a frequency round 10Hz. The lidar pose with respect to the map is the combination of the two transforms, at the same frequency as the lidar odometry.

VII. Experiments

During experiments, the algorithms processing the lidar data run on a laptop computer with 2.5GHz quad cores and 6Gib memory, on robot operating system (ROS) [29] in Linux. The method consumes a total of two cores, the odometry and mapping programs run on two separate cores. Our software code and datasets are publicly available[1,2].

A. Indoor & Outdoor Tests

The method has been tested in indoor and outdoor environments. During indoor tests, the lidar is placed on a cart together with a battery and a laptop computer. One person pushes the cart and walks. Fig. 10(a) and Fig. 10(c) show maps built in two representative indoor environments, a narrow and long corridor and a large lobby. Fig. 10(b) and Fig. 10(d) show two photos taken from the same scenes. In outdoor tests, the lidar is mounted to the front of a ground vehicle. Fig. 10(e) and Fig. 10(g) show maps generated from a vegetated road and an orchard between two rows of trees, and photos are presented in Fig. 10(f) and Fig. 10(h), respectively. During all tests, the lidar moves at a speed of 0.5m/s.

To evaluate local accuracy of the maps, we collect a second set of lidar clouds from the same environments. The lidar is kept stationary and placed at a few different places in each environment during data selection. The two point clouds are matched and compared using the point to plane ICP method [9]. After matching is complete, the distances between one

[1] wiki.ros.org/loam_back_and_forth
[2] wiki.ros.org/loam_continuous

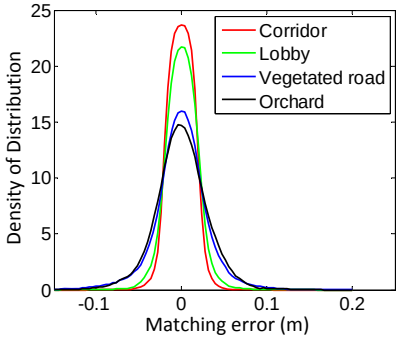


Fig. 11. Matching errors for corridor (red), lobby (green), vegetated road (blue) and orchard (black), corresponding to the four scenes in Fig. 10.

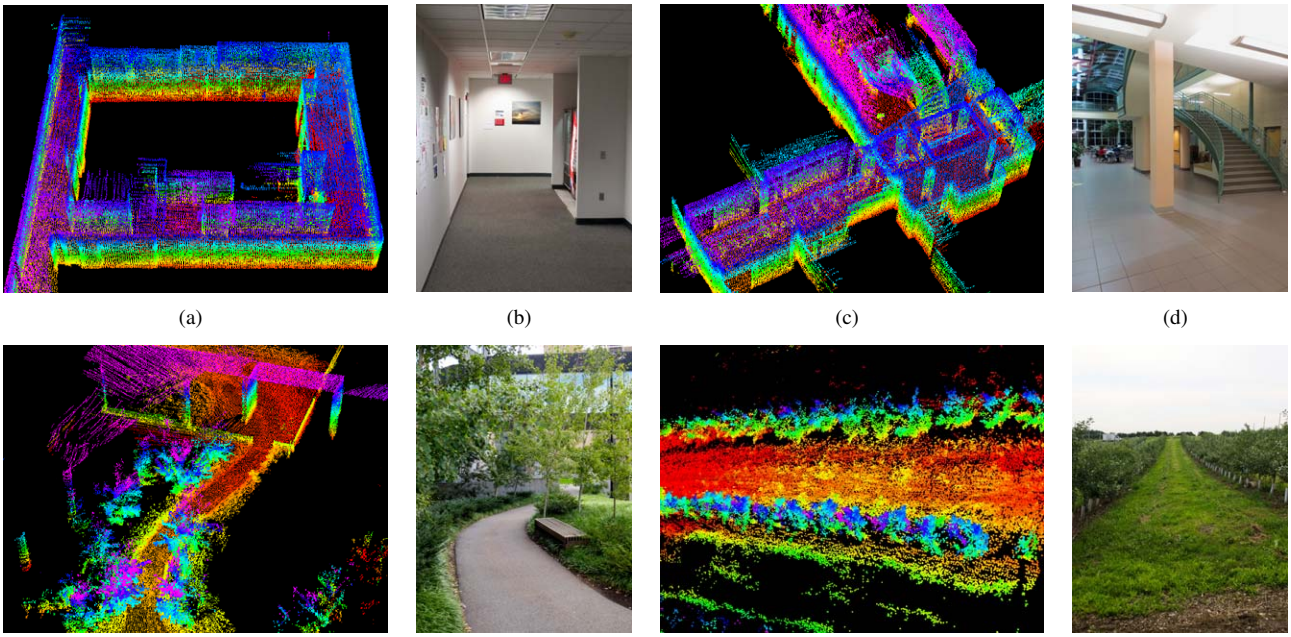


Fig. 10. Maps generated in (a)-(b) a narrow and long corridor, (c)-(d) a large lobby, (e)-(f) a vegetated road, and (g)-(h) an orchard between two rows of trees. The lidar is placed on a cart in indoor tests, and mounted on a ground vehicle in outdoor tests. All tests use a speed of 0.5m/s.

TABLE I

RELATIVE ERRORS FOR MOTION ESTIMATION DRIFT.

Environment	Test 1		Test 2	
	Distance	Error	Distance	Error
Corridor	58m	0.9%	46m	1.1%
Orchard	52m	2.3%	67m	2.8%

point cloud and the corresponding planer patches in the second point cloud are considered as matching errors. Fig. 11 shows the density of error distributions. It indicates smaller matching errors in indoor environments than in outdoor. The result is reasonable because feature matching in natural environments is less exact than in manufactured environments.

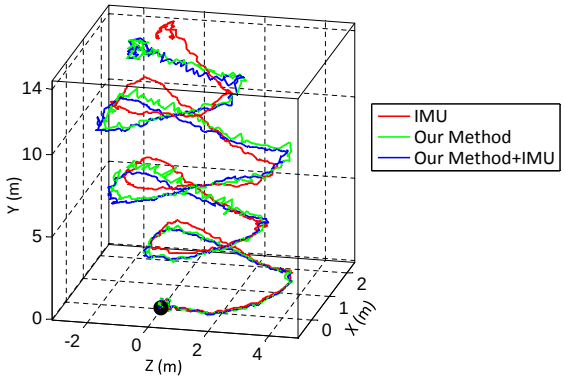
Additionally, we conduct tests to measure accumulated drift of the motion estimate. We choose corridor for indoor experiments that contains a closed loop. This allows us to start and finish at the same place. The motion estimation generates a gap between the starting and finishing positions, which indicates the amount of drift. For outdoor experiments, we choose orchard environment. The ground vehicle that carries the lidar is equipped with a high accuracy GPS/INS for ground truth acquisition. The measured drifts are compared to the distance traveled as the relative accuracy, and listed in Table I. Specifically, Test 1 uses the same datasets with Fig. 10(a) and Fig. 10(g). In general, the indoor tests have a relative accuracy around 1% and the outdoor tests are around 2.5%.

B. Assistance from an IMU

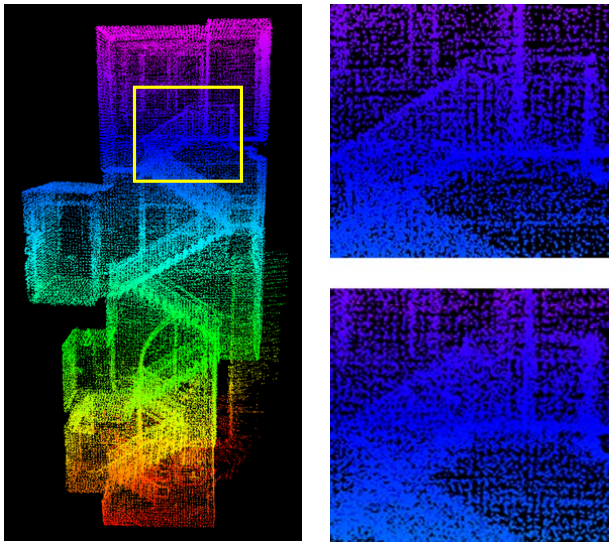
We attach an Xsens MTi-10 IMU to the lidar to deal with fast velocity changes. The point cloud is preprocessed in two ways before sending to the proposed method, 1) with orientation from the IMU, the point cloud received in one sweep is rotated to align with the initial orientation of the lidar in that sweep, 2) with acceleration measurement, the motion distortion is partially removed as if the lidar moves at a const velocity during the sweep. The point cloud is then processed by the lidar odometry and mapping programs.

The IMU orientation is obtained by integrating angular rates from a gyro and readings from an accelerometer in a Kalman filter [1]. Fig. 12(a) shows a sample result. A person holds the lidar and walks on a staircase. When computing the red curve, we use orientation provided by the IMU, and our method only estimates translation. The orientation drifts over 25° during 5 mins of data collection. The green curve relies only on the optimization in our method, assuming no IMU is available. The blue curve uses the IMU data for preprocessing followed by the proposed method. We observe small difference between the green and blue curves. Fig. 12(b) presents the map corresponding to the blue curve. In Fig. 12(c), we compare two closed views of the maps in the yellow rectangular in Fig. 12(b). The upper and lower figures correspond to the blue and green curves, respectively. Careful comparison finds that the edges in the upper figure are sharper.

Table II compares relative errors in motion estimation with and without using the IMU. The lidar is held by a person walking at a speed of 0.5m/s and moving the lidar up and down at a magnitude around 0.5m. The ground truth is manually measured by a tape ruler. In all four tests, using the proposed method with assistance from the IMU gives the highest accuracy, while using orientation from the IMU only leads to the lowest accuracy. The results indicate that the IMU is effective in canceling the nonlinear motion, with which, the proposed method handles the linear motion.



(a)



(b) (c)

Fig. 12. Comparison of results with/without aiding from an IMU. A person holds the lidar and walks on a staircase. The black dot is the starting point. In (a), the red curve is computed using orientation from the IMU and translation estimated by our method, the green curve relies on the optimization in our method only, and the blue curve uses the IMU data for preprocessing followed by the method. (b) is the map corresponding to the blue curve. In (c), the upper and lower figures correspond to the blue and green curves, respectively, using the region labeled by the yellow rectangular in (b). The edges in the upper figure are sharper, indicating more accuracy on the map.

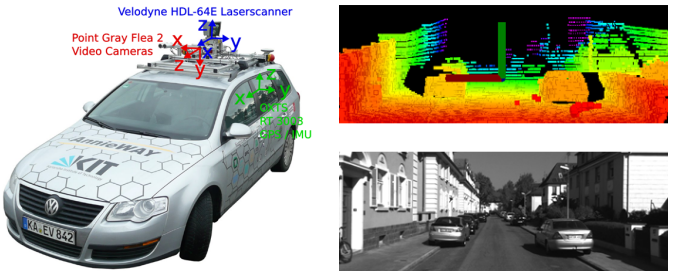
TABLE II

MOTION ESTIMATION ERRORS WITH/WITHOUT USING IMU.

Environment	Distance	Error		
		IMU	Ours	Ours+IMU
Corridor	32m	16.7%	2.1%	0.9%
Lobby	27m	11.7%	1.7%	1.3%
Vegetated road	43m	13.7%	4.4%	2.6%
Orchard	51m	11.4%	3.7%	2.1%

C. Tests with KITTI Datasets

We have also evaluated our method using datasets from the KITTI odometry benchmark [30], [31]. The datasets are carefully registered with sensors mounted on top of a passenger vehicle (Fig. 13(a)) driving on structured roads. The vehicle is equipped with a 360° Velodyne lidar, color/monochrome stereo cameras, and a high accuracy GPS/INS for ground truth



(a) (b)

Fig. 13. (a) Sensor configuration and vehicle used by the KITTI benchmark. The vehicle is mounted with a Velodyne lidar, stereo cameras, and a high accuracy GPS/INS for ground truth acquisition. Our method uses data from the Velodyne lidar only. (b) A sample lidar cloud (upper figure) and the corresponding visual image (lower figure) from an urban scene.

purposes. The lidar data is logged at 10Hz and used by our method for odometry estimation. Due to space issue, we are not able to include the results. However, we encourage readers to review our results on the benchmark website[3].

The datasets mainly cover three types of environments: "urban" with buildings around, "country" on small roads with vegetations in the scene, and "highway" where roads are wide and the surrounding environment is relatively clean. Fig. 13(b) shows a sample lidar cloud and the corresponding visual image from an urban environment. The overall driving distance included in the datasets is 39.2km. Upon uploading of the vehicle trajectories, the accuracy and ranking are automatically calculated by the benchmark server. Our method is ranked #1 among all methods evaluated by the benchmark irrespective of sensing modality, including state of the art stereo visual odometry [32], [33]. The average position error is 0.88% of the distance traveled, generated using trajectory segments at 100m, 200m, ..., 800m lengthes in 3D coordinates.

VIII. CONCLUSION AND FUTURE WORK

Motion estimation and mapping using point cloud from a rotating laser scanner can be difficult because the problem involves recovery of motion and correction of motion distortion in the lidar cloud. The proposed method divides and solves the problem by two algorithms running in parallel: the lidar odometry conduces coarse processing to estimate velocity at a higher frequency, while the lidar mapping performs fine processing to create maps at a lower frequency. Cooperation of the two algorithms allows accurate motion estimation and mapping in real-time. Further, the method can take advantage of the lidar scan pattern and point cloud distribution. Feature matching is made to ensure fast computation in the odometry algorithm, and to enforce accuracy in the mapping algorithm. The method has been tested both indoor and outdoor as well as on the KITTI odometry benchmark.

Since the current method does not recognize loop closure, our future work involves developing a method to fix motion estimation drift by closing the loop. Also, we will integrate the output of our method with an IMU in a Kalman filter to further reduce the motion estimation drift.

[3] www.cvlibs.net/datasets/kitti/eval_odometry.php

REFERENCES

[1] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics*. Cambridge, MA, The MIT Press, 2005.

[2] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, "iSAM2: Incremental smoothing and mapping using the bayes tree," *The International Journal of Robotics Research*, vol. 31, pp. 217–236, 2012.

[3] R. Zlot and M. Bosse, "Efficient large-scale 3D mobile mapping and surface reconstruction of an underground mine," in *The 7th International Conference on Field and Service Robots*, Matsushima, Japan, July 2012.

[4] K. Konolige, M. Agrawal, and J. Sol, "Large-scale visual odometry for rough terrain," *Robotics Research*, vol. 66, p. 201212, 2011.

[5] D. Nister, O. Naroditsky, and J. Bergen, "Visual odometry for ground vechicle applications," *Journal of Field Robotics*, vol. 23, no. 1, pp. 3–20, 2006.

[6] M. Bosse, R. Zlot, and P. Flick, "Zebedee: Design of a spring-mounted 3-D range sensor with application to mobile mapping," vol. 28, no. 5, pp. 1104–1119, 2012.

[7] S. Shen and N. Michael, "State estimation for indoor and outdoor operation with a micro-aerial vehicle," in *International Symposium on Experimental Robotics (ISER)*, Quebec City, Canada, June 2012.

[8] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: part i the essential algorithms," *IEEE Robotics & Automation Magazine*, vol. 13, no. 2, pp. 99–110, 2006.

[9] S. Rusinkiewicz and M. Levoy, "Efficient variants of the ICP algorithm," in *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*, Quebec City, Canada, June 2001.

[10] A. Nuchter, 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.

[11] S. Kohlbrecher, O. von Stryk, J. Meyer, and U. Klingauf, "A flexible and scalable SLAM system with full 3D motion estimation," in *IEEE International Symposium on Safety, Security, and Rescue Robotics*, Kyoto, Japan, September 2011.

[12] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing ICP variants on real-world data sets," *Autonomous Robots*, vol. 34, no. 3, pp. 133–148, 2013.

[13] S. Hong, H. Ko, and J. Kim, "VICP: Velocity updating iterative closest point algorithm," in *IEEE International Conference on Robotics and Automation (ICRA)*, Anchorage, Alaska, May 2010.

[14] F. Moosmann and C. Stiller, "Velodyne SLAM," in *IEEE Intelligent Vehicles Symposium (IV)*, Baden-Baden, Germany, June 2011.

[15] S. Scherer, J. Rehder, S. Achar, H. Cover, A. Chambers, S. Nuske, and S. Singh, "River mapping from a flying robot: state estimation, river detection, and obstacle mapping," *Autonomous Robots*, vol. 32, no. 5, pp. 1 – 26, May 2012.

[16] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: A factored solution to the simultaneous localization and mapping problem," in *The AAAI Conference on Artificial Intelligence*, Edmonton, Canada, July 2002.

[17] H. Bay, A. Ess, T. Tuytelaars, and L. Gool, "SURF: Speeded up robust features," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[18] H. Dong and T. Barfoot, "Lighting-invariant visual odometry using lidar intensity imagery and pose interpolation," in *The 7th International Conference on Field and Service Robots*, Matsushima, Japan, July 2012.

[19] S. Anderson and T. Barfoot, "RANSAC for motion-distorted 3D visual sensors," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Tokyo, Japan, Nov. 2013.

[20] C. H. Tong and T. Barfoot, "Gaussian process gauss-newton for 3D laser-based visual odometry," in *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013.

[21] S. Anderson and T. Barfoot, "Towards relative continuous-time SLAM," in *IEEE International Conference on Robotics and Automation (ICRA)*, Karlsruhe, Germany, May 2013.

[22] M. Bosse and R. Zlot, "Continuous 3D scan-matching with a spinning 2D laser," in *IEEE International Conference on Robotics and Automation*, Kobe, Japan, May 2009.

[23] Y. Li and E. Olson, "Structure tensors for general purpose LIDAR feature extraction," in *IEEE International Conference on Robotics and Automation*, Shanghai, China, May 9-13 2011.

[24] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf, *Computation Geometry: Algorithms and Applications, 3rd Edition*. Springer, 2008.

[25] R. Murray and S. Sastry, *A mathematical introduction to robotic manipulation*. CRC Press, 1994.

[26] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. New York, Cambridge University Press, 2004.

[27] R. Andersen, "Modern methods for robust regression." *Sage University Paper Series on Quantitative Applications in the Social Sciences*, 2008.

[28] R. B. Rusu and S. Cousins, "3D is here: Point Cloud Library (PCL)," in *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[29] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: An open-source robot operating system," in *Workshop on Open Source Software (Collocated with ICRA 2009)*, Kobe, Japan, May 2009.

[30] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? The kitti vision benchmark suite," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012, pp. 3354–3361.

[31] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The KITTI dataset," *International Journal of Robotics Research*, no. 32, pp. 1229–1235, 2013.

[32] H. Badino and T. Kanade, "A head-wearable short-baseline stereo system for the simultaneous estimation of structure and motion," in *IAPR Conference on Machine Vision Application*, Nara, Japan, 2011.

[33] A. Y. H. Badino and T. Kanade, "Visual odometry by multi-frame feature integration," in *Workshop on Computer Vision for Autonomous Driving (Collocated with ICCV 2013)*, Sydney, Australia, 2013.