LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping

Preprint · July 2020

CITATIONS
0

READS
1,523

6 authors, including:

Tixiao Shan
Massachusetts Institute of Technology
15 PUBLICATIONS **134** CITATIONS
SEE PROFILE

Brendan Englot
Stevens Institute of Technology
62 PUBLICATIONS **1,019** CITATIONS
SEE PROFILE

Wei Wang
Massachusetts Institute of Technology
57 PUBLICATIONS **379** CITATIONS
SEE PROFILE

LIO-SAM: Tightly-coupled Lidar Inertial Odometry via Smoothing and Mapping

Tixiao Shan, Brendan Englot, Drew Meyers, Wei Wang, Carlo Ratti, and Daniela Rus

***Abstract*— We propose a framework for tightly-coupled lidar inertial odometry via smoothing and mapping, LIO-SAM, that achieves highly accurate, real-time mobile robot trajectory estimation and map-building. LIO-SAM formulates lidar-inertial odometry atop a factor graph, allowing a multitude of relative and absolute measurements, including loop closures, to be incorporated from different sources as factors into the system. The estimated motion from inertial measurement unit (IMU) pre-integration de-skews point clouds and produces an initial guess for lidar odometry optimization. The obtained lidar odometry solution is used to estimate the bias of the IMU. To ensure high performance in real-time, we marginalize old lidar scans for pose optimization, rather than matching lidar scans to a global map. Scan-matching at a local scale instead of a global scale significantly improves the real-time performance of the system, as does the selective introduction of keyframes, and an efficient sliding window approach that registers a new keyframe to a fixed-size set of prior "sub-keyframes." The proposed method is extensively evaluated on datasets gathered from three platforms over various scales and environments.**

I. Introduction

State estimation, localization and mapping are fundamental prerequisites for a successful intelligent mobile robot, required for feedback control, obstacle avoidance, and planning, among many other capabilities. Using vision-based and lidar-based sensing, great efforts have been devoted to achieving high-performance real-time simultaneous localization and mapping (SLAM) that can support a mobile robot's six degree-of-freedom state estimation. Vision-based methods typically use a monocular or stereo camera and triangulate features across successive images to determine the camera motion. Although vision-based methods are especially suitable for place recognition, their sensitivity to initialization, illumination, and range make them unreliable when they alone are used to support an autonomous navigation system. On the other hand, lidar-based methods are largely invariant to illumination change. Especially with the recent availability of long-range, high-resolution 3D lidar, such as the Velodyne VLS-128 and Ouster OS1-128, lidar becomes more suitable to directly capture the fine details of an environment in 3D space. Therefore, this paper focuses on lidar-based state estimation and mapping methods.

Many lidar-based state estimation and mapping methods have been proposed in the last two decades. Among them, the lidar odometry and mapping (LOAM) method proposed in [1] for low-drift and real-time state estimation and mapping is among the most widely used. LOAM, which uses a lidar and an inertial measurement unit (IMU), achieves state-of-the-art performance and has been ranked as the top lidar-based method since its release on the KITTI odometry benchmark site [2]. Despite its success, LOAM presents some limitations - by saving its data in a global voxel map, it is often difficult to perform loop closure detection and incorporate other absolute measurements, e.g., GPS, for pose correction. Its online optimization process becomes less efficient when this voxel map becomes dense in a feature-rich environment. LOAM also suffers from drift in large-scale tests, as it is a scan-matching based method at its core.

In this paper, we propose a framework for tightly-coupled lidar inertial odometry via smoothing and mapping, LIO-SAM, to address the aforementioned problems. We assume a nonlinear motion model for point cloud de-skew, estimating the sensor motion during a lidar scan using raw IMU measurements. In addition to de-skewing point clouds, the estimated motion serves as an initial guess for lidar odometry optimization. The obtained lidar odometry solution is then used to estimate the bias of the IMU in the factor graph. By introducing a global factor graph for robot trajectory estimation, we can efficiently perform sensor fusion using lidar and IMU measurements, incorporate place recognition among robot poses, and introduce absolute measurements, such as GPS positioning and compass heading, when they are available. This collection of factors from various sources is used for joint optimization of the graph. Additionally, we marginalize old lidar scans for pose optimization, rather than matching scans to a global map like LOAM. Scan-matching at a local scale instead of a global scale significantly improves the real-time performance of the system, as does the selective introduction of keyframes, and an efficient sliding window approach that registers a new keyframe to a fixed-size set of prior "sub-keyframes." The main contributions of our work can be summarized as follows:

- A tightly-coupled lidar inertial odometry framework built atop a factor graph, that is suitable for multi-sensor fusion and global optimization.
- An efficient, local sliding window-based scan-matching approach that enables real-time performance by registering selectively chosen new keyframes to a fixed-size set of prior sub-keyframes.
- The proposed framework is extensively validated with tests across various scales, vehicles, and environments.

T. Shan, D. Meyers, W. Wang, and C. Ratti are with the Department of Urban Studies and Planning, Massachusetts Institute of Technology, USA, {shant, drewm, wweiwang, ratti}@mit.edu.

B. Englot is with the Department of Mechanical Engineering, Stevens Institute of Technology, USA, benglot@stevens.edu.

T. Shan, W. Wang, and D. Rus are with the Computer Science & Artificial Intelligence Laboratory, Massachusetts Institute of Technology, USA, {shant, wweiwang, rus}@mit.edu.

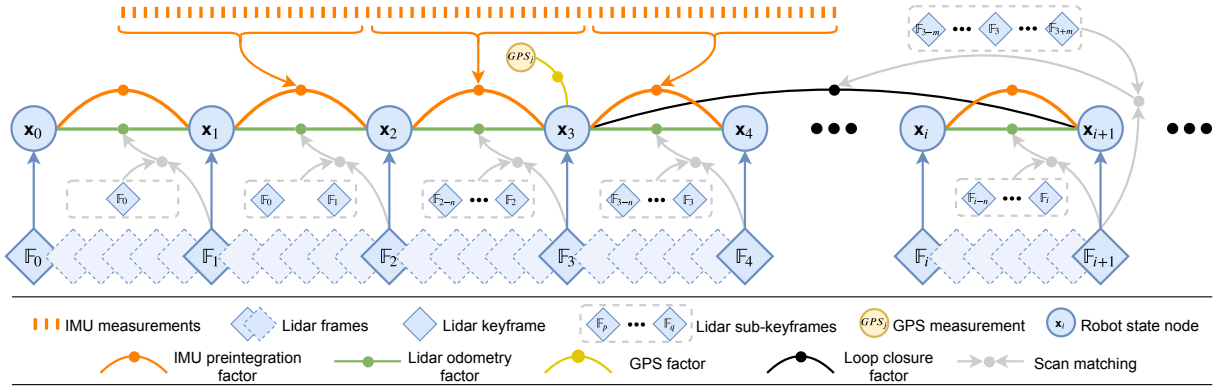


Fig. 1: The system structure of LIO-SAM. The system receives input from a 3D lidar, an IMU and optionally a GPS. Four types of factors are introduced to construct the factor graph.: (a) IMU preintegration factor, (b) lidar odometry factor, (c) GPS factor, and (d) loop closure factor. The generation of these factors is discussed in Section III.

II. Related Work

Lidar odometry is typically performed by finding the relative transformation between two consecutive frames using scan-matching methods such as ICP [3] and GICP [4]. Instead of matching a full point cloud, feature-based matching methods have become a popular alternative due to their computational efficiency. For example, in [5], a plane-based registration approach is proposed for real-time lidar odometry. Assuming operations in a structured environment, it extracts planes from the point clouds and matches them by solving a least-squares problem. A collar line-based method is proposed in [6] for odometry estimation. In this method, line segments are randomly generated from the original point cloud and used later for registration. However, a scan's point cloud is often skewed because of the rotation mechanism of modern 3D lidar, and sensor motion. Solely using lidar for pose estimation is not ideal since registration using skewed point clouds or features will eventually cause large drift.

Therefore, lidar is typically used in conjunction with other sensors, such as IMU and GPS, for state estimation and mapping. Such a design scheme, utilizing sensor fusion, can typically be grouped into two categories: loosely-coupled fusion and tightly-coupled fusion. In LOAM [1], IMU is introduced to de-skew the lidar scan and give a motion prior for scan-matching. However, the IMU is not involved in the optimization process of the algorithm. Thus LOAM can be classified as a loosely-coupled method. A lightweight and ground-optimized lidar odometry and mapping (LeGO-LOAM) method is proposed in [7] for ground vehicle mapping tasks [8]. Its fusion of IMU measurements is the same as LOAM. A more popular approach for loosely-coupled fusion is using extended Kalman filters (EKF). For example, [9]-[13] integrate measurements from lidar, IMU, and optionally GPS using an EKF in the optimization stage for robot state estimation.

Tightly-coupled systems usually offer improved accuracy and are presently a major focus of ongoing research [14]. In [15], preintegrated IMU measurements are exploited for de-skewing point clouds. A robocentric lidar-inertial state estimator, R-LINS, is presented in [16]. R-LINS uses an error-state Kalman filter to correct a robot's state estimate recursively in a tightly-coupled manner. Due to the lack of other available sensors for state estimation, it suffers from drift during long-during navigation. A tightly-coupled lidar inertial odometry and mapping framework, LIOM, is introduced in [17]. LIOM, which is the abbreviation for LIO-mapping, jointly optimizes measurements from lidar and IMU and achieves similar or better accuracy when compared with LOAM. Since LIOM is designed to process all the sensor measurements, real-time performance is not achieved - it runs at about $0.6\times$ real-time in our tests.

III. Lidar Inertial Odometry via Smoothing and Mapping

A. System Overview

We first define frames and notation that we use throughout the paper. We denote the world frame as $\mathbf{W}$ and the robot body frame as $\mathbf{B}$. We also assume the IMU frame coincides with the robot body frame for convenience. The robot state $\mathbf{x}$ can be written as:

$$\mathbf{x} = [\ \mathbf{R}^{\mathbf{T}},\ \mathbf{p}^{\mathbf{T}},\ \mathbf{v}^{\mathbf{T}},\ \mathbf{b}^{\mathbf{T}}\ ]^{\mathbf{T}}, \tag{1}$$

where $\mathbf{R} \in SO(3)$ is the rotation matrix, $\mathbf{p} \in \mathbb{R}^3$ is the position vector, $\mathbf{v}$ is the speed, and $\mathbf{b}$ is the IMU bias. The transformation $\mathbf{T} \in SE(3)$ from $\mathbf{B}$ to $\mathbf{W}$ is represented as $\mathbf{T} = [\mathbf{R} \mid \mathbf{p}]$.

An overview of the proposed system is shown in Figure 1. The system receives sensor data from a 3D lidar, an IMU and optionally a GPS. We seek to estimate the state of the robot and its trajectory using the observations of these sensors. This state estimation problem can be formulated as a maximum a posteriori (MAP) problem. We use a factor graph to model this problem, as it is better suited to perform inference when compared with Bayes nets. With the assumption of a Gaussian noise model, the MAP inference for our problem is equivalent to solving a nonlinear least-squares problem [18]. Note that without loss of generality, the proposed system can also incorporate measurements from other sensors, such as elevation from an altimeter or heading from a compass.

We introduce four types of *factors* along with one *variable* type for factor graph construction. This variable, representing the robot's state at a specific time, is attributed

to the $nodes$ of the graph. The four types of factors are: (a) IMU preintegration factors, (b) lidar odometry factors, (c) GPS factors, and (d) loop closure factors. A new robot state node $\mathbf{x}$ is added to the graph when the change in robot pose exceeds a user-defined threshold. The factor graph is optimized upon the insertion of a new node using incremental smoothing and mapping with the Bayes tree (iSAM2) [19]. The process for generating these factors is described in the following sections.

B. IMU Preintegration Factor

The measurements of angular velocity and acceleration from an IMU are defined using Eqs. 2 and 3:

$$\hat{\boldsymbol{\omega}}_t = \boldsymbol{\omega}_t + \mathbf{b}_t^{\boldsymbol{\omega}} + \mathbf{n}_t^{\boldsymbol{\omega}} \tag{2}$$

$$\hat{\mathbf{a}}_t = \mathbf{R}_t^{\mathbf{BW}}(\mathbf{a}_t - \mathbf{g}) + \mathbf{b}_t^{\mathbf{a}} + \mathbf{n}_t^{\mathbf{a}}, \tag{3}$$

where $\hat{\boldsymbol{\omega}}_t$ and $\hat{\mathbf{a}}_t$ are the raw IMU measurements in $\mathbf{B}$ at time t. $\hat{\boldsymbol{\omega}}_t$ and $\hat{\mathbf{a}}_t$ are affected by a slowly varying bias $\mathbf{b}_t$ and white noise $\mathbf{n}_t$. $\mathbf{R}_t^{\mathbf{BW}}$ is the rotation matrix from $\mathbf{W}$ to $\mathbf{B}$. $\mathbf{g}$ is the constant gravity vector in $\mathbf{W}$.

We can now use the measurements from the IMU to infer the motion of the robot. The velocity, position and rotation of the robot at time $t + \Delta t$ can be computed as follows:

$$\mathbf{v}_{t+\Delta t} = \mathbf{v}_t + \mathbf{g}\Delta t + \mathbf{R}_t(\hat{\mathbf{a}}_t - \mathbf{b}_t^{\mathbf{a}} - \mathbf{n}_t^{\mathbf{a}})\Delta t \tag{4}$$

$$\begin{aligned}\mathbf{p}_{t+\Delta t} = \mathbf{p}_t + \mathbf{v}_t\Delta t + \frac{1}{2}\mathbf{g}\Delta t^2 \\ + \frac{1}{2}\mathbf{R}_t(\hat{\mathbf{a}}_t - \mathbf{b}_t^{\mathbf{a}} - \mathbf{n}_t^{\mathbf{a}})\Delta t^2\end{aligned} \tag{5}$$

$$\mathbf{R}_{t+\Delta t} = \mathbf{R}_t\, \mathbf{exp}((\hat{\boldsymbol{\omega}}_t - \mathbf{b}_t^{\boldsymbol{\omega}} - \mathbf{n}_t^{\boldsymbol{\omega}})\Delta t), \tag{6}$$

where $\mathbf{R}_t = \mathbf{R}_t^{\mathbf{WB}} = \mathbf{R}_t^{\mathbf{BW}^\mathsf{T}}$. Here we assume that the angular velocity and the acceleration of $\mathbf{B}$ remain constant during the above integration.

We then apply the IMU preintegration method proposed in [20] to obtain the relative body motion between two timesteps. The preintegrated measurements $\Delta\mathbf{v}_{ij}$, $\Delta\mathbf{p}_{ij}$, and $\Delta\mathbf{R}_{ij}$ between time i and j can be computed using:

$$\Delta\mathbf{v}_{ij} = \mathbf{R}_i^\mathsf{T}(\mathbf{v}_j - \mathbf{v}_i - \mathbf{g}\Delta t_{ij}) \tag{7}$$

$$\Delta\mathbf{p}_{ij} = \mathbf{R}_i^\mathsf{T}(\mathbf{p}_j - \mathbf{p}_i - \mathbf{v}_i\Delta t_{ij} - \frac{1}{2}\mathbf{g}\Delta t_{ij}^2) \tag{8}$$

$$\Delta\mathbf{R}_{ij} = \mathbf{R}_i^\mathsf{T}\, \mathbf{R}_j. \tag{9}$$

Due to space limitations, we refer the reader to the description from [20] for the detailed derivation of Eqs. 7 - 9. Besides its efficiency, applying IMU preintegration also naturally gives us one type of constraint for the factor graph - IMU preintegration factors. The IMU bias is jointly optimized alongside the lidar odometry factors in the graph.

C. Lidar Odometry Factor

When a new lidar scan arrives, we first perform feature extraction. Edge and planar features are extracted by evaluating the roughness of points over a local region. Points with a large roughness value are classified as edge features. Similarly, a planar feature is categorized by a small roughness value. We denote the extracted edge and planar features from a lidar scan at time i as F_i^e and F_i^p respectively. All the features extracted at time i compose a lidar *frame* $\mathbb{F}_i$, where $\mathbb{F}_i = \{\mathrm{F}_i^e, \mathrm{F}_i^p\}$. Note that a lidar frame $\mathbb{F}$ is represented in $\mathbf{B}$. A more detailed description of the feature extraction process can be found in [1], or [7] if a range image is used.

Using every lidar frame for computing and adding factors to the graph is computationally intractable, so we adopt the concept of *keyframe* selection, which is widely used in the visual SLAM field. Using a simple but effective heuristic approach, we select a lidar frame $\mathbb{F}_{i+1}$ as a keyframe when the change in robot pose exceeds a user-defined threshold when compared with the previous state $\mathbf{x}_i$. The newly saved keyframe, $\mathbb{F}_{i+1}$, is associated with a new robot state node, $\mathbf{x}_{i+1}$, in the factor graph. The lidar frames between two keyframes are discarded. Adding keyframes in this way not only achieves a balance between map density and memory consumption but also helps maintain a relatively sparse factor graph, which is suitable for real-time nonlinear optimization. In our work, the position and rotation change thresholds for adding a new keyframe are chosen to be $1m$ and $10°$.

Let us assume we wish to add a new state node $\mathbf{x}_{i+1}$ to the factor graph. The lidar keyframe that is associated with this state is $\mathbb{F}_{i+1}$. The generation of a lidar odometry factor is described in the following steps:

1) Sub-keyframes for voxel map: We implement a sliding window approach to create a point cloud map containing a fixed number of recent lidar scans. Instead of optimizing the transformation between two consecutive lidar scans, we extract the n most recent keyframes, which we call the *sub-keyframes*, for estimation. The set of sub-keyframes $\{\mathbb{F}_{i-n}, ..., \mathbb{F}_i\}$ is then transformed into frame $\mathbf{W}$ using the transformations $\{\mathbf{T}_{i-n}, ..., \mathbf{T}_i\}$ associated with them. The transformed sub-keyframes are merged together into a voxel map $\mathbf{M}_i$. Since we extract two types of features in the previous feature extraction step, $\mathbf{M}_i$ is composed of two sub-voxel maps that are denoted $\mathbf{M}_i^e$, the edge feature voxel map, and $\mathbf{M}_i^p$, the planar feature voxel map. The lidar frames and voxel maps are related to each other as follows:

$$\begin{aligned}&\mathbf{M}_i = \{\mathbf{M}_i^e, \mathbf{M}_i^p\} \\ &where: \mathbf{M}_i^e = {}'\mathrm{F}_i^e \cup' \mathrm{F}_{i-1}^e \cup ... \cup' \mathrm{F}_{i-n}^e \\ &\qquad\quad \mathbf{M}_i^p = {}'\mathrm{F}_i^p \cup' \mathrm{F}_{i-1}^p \cup ... \cup' \mathrm{F}_{i-n}^p.\end{aligned}$$

${}'\mathrm{F}_i^e$ and ${}'\mathrm{F}_i^p$ are the transformed edge and planar features in $\mathbf{W}$. $\mathbf{M}_i^e$ and $\mathbf{M}_i^p$ are then downsampled to eliminate the duplicated features that fall in the same voxel cell. In this paper, n is chosen to be 25. The downsample resolutions for $\mathbf{M}_i^e$ and $\mathbf{M}_i^p$ are $0.2m$ and $0.4m$, respectively.

2) Scan-matching: We match a newly obtained lidar frame $\mathbb{F}_{i+1}$, which is also $\{\mathrm{F}_{i+1}^e, \mathrm{F}_{i+1}^p\}$, to $\mathbf{M}_i$ via scan-matching. Various scan-matching methods, such as [3] and [4], can be utilized for this purpose. Here we opt to use the method proposed in [1] due to its computational efficiency and robustness in various challenging environments.

We first transform $\{\mathrm{F}_{i+1}^e, \mathrm{F}_{i+1}^p\}$ from $\mathbf{B}$ to $\mathbf{W}$ and obtain $\{{}'\mathrm{F}_{i+1}^e, {}'\mathrm{F}_{i+1}^p\}$. This initial transformation is obtained by using the predicted robot motion, $\tilde{\mathbf{T}}_{i+1}$, from the IMU. For

each feature in ${}'\mathrm{F}^e_{i+1}$ or ${}'\mathrm{F}^p_{i+1}$, we then find its edge or planar correspondence in $\mathbf{M}^e_i$ or $\mathbf{M}^p_i$. For the sake of brevity, the detailed procedures for finding these correspondences are omitted here, but are described thoroughly in [1].

3) Relative transformation: The distance between a feature and its edge or planar patch correspondence can be computed using the following equations:

$$\mathbf{d}_{e_k} = \frac{\left|(\mathbf{p}^e_{i+1,k} - \mathbf{p}^e_{i,u}) \times (\mathbf{p}^e_{i+1,k} - \mathbf{p}^e_{i,v})\right|}{\left|\mathbf{p}^e_{i,u} - \mathbf{p}^e_{i,v}\right|} \tag{10}$$

$$\mathbf{d}_{p_k} = \frac{\left|\begin{matrix}(\mathbf{p}^p_{i+1,k} - \mathbf{p}^p_{i,u}) \\ (\mathbf{p}^p_{i,u} - \mathbf{p}^p_{i,v}) \times (\mathbf{p}^p_{i,u} - \mathbf{p}^p_{i,w})\end{matrix}\right|}{\left|(\mathbf{p}^p_{i,u} - \mathbf{p}^p_{i,v}) \times (\mathbf{p}^p_{i,u} - \mathbf{p}^p_{i,w})\right|}, \tag{11}$$

where k, u, v, and w are the feature indices in their corresponding sets. For an edge feature $\mathbf{p}^e_{i+1,k}$ in ${}'\mathrm{F}^e_{i+1}$, $\mathbf{p}^e_{i,u}$ and $\mathbf{p}^e_{i,v}$ are the points that form the corresponding edge line in $\mathbf{M}^e_i$. For a planar feature $\mathbf{p}^p_{i+1,k}$ in ${}'\mathrm{F}^p_{i+1}$, $\mathbf{p}^p_{i,u}$, $\mathbf{p}^p_{i,v}$, and $\mathbf{p}^p_{i,w}$ form the corresponding planar patch in $\mathbf{M}^p_i$. The GaussNewton method is then used to solve for the optimal transformation by minimizing:

$$\min_{\mathbf{T}_{i+1}} \left\{ \sum_{\mathbf{p}_{i+1,k} \in {}'\mathrm{F}^e_{i+1}} \mathbf{d}_{e_k} + \sum_{\mathbf{p}_{i+1,k} \in {}'\mathrm{F}^p_{i+1}} \mathbf{d}_{p_k} \right\}.$$

At last, we can obtain the relative transformation $\Delta\mathbf{T}_{i,i+1}$ between $\mathbf{x}_i$ and $\mathbf{x}_{i+1}$, which is the lidar odometry factor linking these two poses:

$$\Delta\mathbf{T}_{i,i+1} = \mathbf{T}_i^{\mathsf{T}} \mathbf{T}_{i+1} \tag{12}$$

We note that an alternative approach to obtain $\Delta\mathbf{T}_{i,i+1}$ is to transform sub-keyframes into the frame of $\mathbf{x}_i$. In other words, we match $\mathbb{F}_{i+1}$ to the voxel map that is represented in the frame of $\mathbf{x}_i$. In this way, the *real* relative transformation $\Delta\mathbf{T}_{i,i+1}$ can be directly obtained. Because the transformed features ${}'\mathrm{F}^e_i$ and ${}'\mathrm{F}^p_i$ can be reused multiple times, we instead opt to use the approach described in Sec. III-C.1 for its computational efficiency.

D. GPS Factor

Though we can obtain reliable state estimation and mapping by utilizing only IMU preintegration and lidar odometry factors, the system still suffers from drift during long-duration navigation tasks. To solve this problem, we can introduce sensors that offer absolute measurements for eliminating drift. Such sensors include an altimeter, compass, and GPS. For the purposes of illustration here, we discuss GPS, as it is widely used in real-world navigation systems.

When we receive GPS measurements, we first transform them to the local Cartesian coordinate frame using the method proposed in [21]. Upon the addition of a new node to the factor graph, we then associate a new GPS factor with this node. If the GPS signal is not hardware-synchronized with the lidar frame, we interpolate among GPS measurements linearly based on the timestamp of the lidar frame.

(a) Handheld device (b) Clearpath Jackal (c) Duffy 21

Fig. 2: Datasets are collected on 3 platforms: (a) a custom-built handheld device, (b) an unmanned ground vehicle - Clearpath Jackal, (c) an electric boat - Duffy 21.

We note that adding GPS factors constantly when GPS reception is available is not necessary because the drift of lidar inertial odometry grows very slowly. In practice, we only add a GPS factor when the estimated position covariance is larger than the received GPS position covariance.

E. Loop Closure Factor

Thanks to the utilization of a factor graph, loop closures can also be seamlessly incorporated into the proposed system, as opposed to LOAM and LIOM. For the purposes of illustration, we describe and implement a naive but effective Euclidean distance-based loop closure detection approach. We also note that our proposed framework is compatible with other methods for loop closure detection, for example, [22] and [23], which generate a point cloud descriptor and use it for place recognition.

When a new state $\mathbf{x}_{i+1}$ is added to the factor graph, we first search the graph and find the prior states that are close to $\mathbf{x}_{i+1}$ in Euclidean space. As is shown in Fig. 1, for example, $\mathbf{x}_3$ is one of the returned candidates. We then try to match $\mathbb{F}_{i+1}$ to the sub-keyframes $\{\mathbb{F}_{3-m}, ..., \mathbb{F}_3, ..., \mathbb{F}_{3+m}\}$ using the scan-matching method discussed in Section III-C. Note that $\mathbb{F}_{i+1}$ and the past sub-keyframes are first transformed into $\mathbf{W}$ before scan-matching. We obtain the relative transformation $\Delta\mathbf{T}_{3,i+1}$ and add it as a loop closure factor to the graph. Throughout this paper, we choose the index m to be 12, and the search distance for loop closures is set to be $15m$ from a new state $\mathbf{x}_{i+1}$.

In practice, we find adding loop closure factors is especially useful for correcting the drift in a robot's altitude, when GPS is the only absolute sensor available. This is because the elevation measurement from GPS is very inaccurate - giving rise to altitude errors approaching $100m$ in our tests, in the absence of loop closures.

IV. Experiments

We now describe a series of experiments to qualitatively and quantitatively analyze our proposed framework. The sensor suite used in this paper includes a Velodyne VLP-16 lidar, a MicroStrain 3DM-GX5-25 IMU, and a Reach M GPS. For validation, we collected 5 different datasets across various scales, platforms and environments. These datasets are referred to as *Rotation*, *Walking*, *Campus*, *Park* and *Amsterdam*, respectively. The sensor mounting platforms are shown in Fig. 2. The first three datasets were collected using

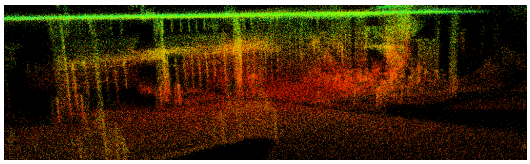
a custom-built handheld device on the MIT campus. The Park dataset was collected in a park covered by vegetation, using an unmanned ground vehicle (UGV) - the Clearpath Jackal. The last dataset, Amsterdam, was collected by mounting the sensors on a boat and cruising in the canals of Amsterdam. The details of these datasets are shown in Table I.

TABLE I: Dataset details

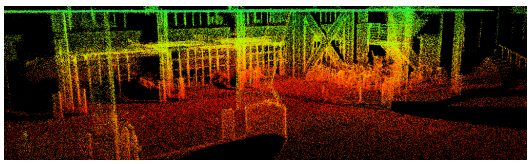
Dataset	Scans	Elevation change (m)	Trajectory length (m)	Max rotation speed (°/s)
Rotation	582	0	0	213.9
Walking	6502	0.3	801	133.7
Campus	9865	1.0	1437	124.8
Park	24691	19.0	2898	217.4
Amsterdam	107656	0	19065	17.2

We compare the proposed LIO-SAM framework with LOAM and LIOM. In all the experiments, LOAM and LIO-SAM are forced to run in real-time. LIOM, on the other hand, is given infinite time to process every sensor measurement. All the methods are implemented in C++ and executed on a laptop equipped with an Intel i7-10710U CPU using the robot operating system (ROS) [24] in Ubuntu Linux. We note that only the CPU is used for computation, without parallel computing enabled. Our implementation of LIO-SAM is freely available on Github[1]. Supplementary details of the experiments performed, including complete visualizations of all tests, can be found at the link below[2].

(a) Test environment



(b) LOAM



(c) LIO-SAM

Fig. 3: Mapping results of LOAM and LIO-SAM in the *Rotation* test. LIOM fails to produce meaningful results.

A. Rotation Dataset

In this test, we focus on evaluating the robustness of our framework when only IMU preintegration and lidar odometry factors are added to the factor graph. The *Rotation* dataset is collected by a user holding the sensor suite and performing a series of aggressive rotational maneuvers while standing still. The maximum rotational speed encountered in this test is 133.7 $^\circ/s$. The test environment, which is populated with structures, is shown in Fig. 3(a). The maps obtained from LOAM and LIO-SAM are shown in Figs. 3(b) and (c) respectively. Because LIOM uses the same initialization pipeline from [25], it inherits the same initialization sensitivity of visual-inertial SLAM and is not able to initialize properly using this dataset. Due to its failure to produce meaningful results, the map of LIOM is not shown. As is shown, the map of LIO-SAM preserves more fine structural details of the environment compared with the map of LOAM. This is because LIO-SAM is able to register each lidar frame precisely in $SO(3)$, even when the robot undergoes rapid rotation.

B. Walking Dataset

This test is designed to evaluate the performance of our method when the system undergoes aggressive translations and rotations in $SE(3)$. The maximum translational and rotational speed encountered is this dataset is 1.8 m/s and 213.9 °/s respectively. During the data gathering, the user holds the sensor suite shown in Fig. 2(a) and walks quickly across the MIT campus (Fig. 4(a)). In this test, the map of LOAM, shown in Fig. 4(b), diverges at multiple locations when aggressive rotation is encountered. LIOM outperforms LOAM in this test. However, its map, shown in Fig. 4(c), still diverges slightly in various locations and consists of numerous blurry structures. Because LIOM is designed to process all sensor measurements, it only runs at $0.56\times$ real-time while other methods are running in real-time. Finally, LIO-SAM outperforms both methods and produces a map that is consistent with the available Google Earth imagery.

C. Campus Dataset

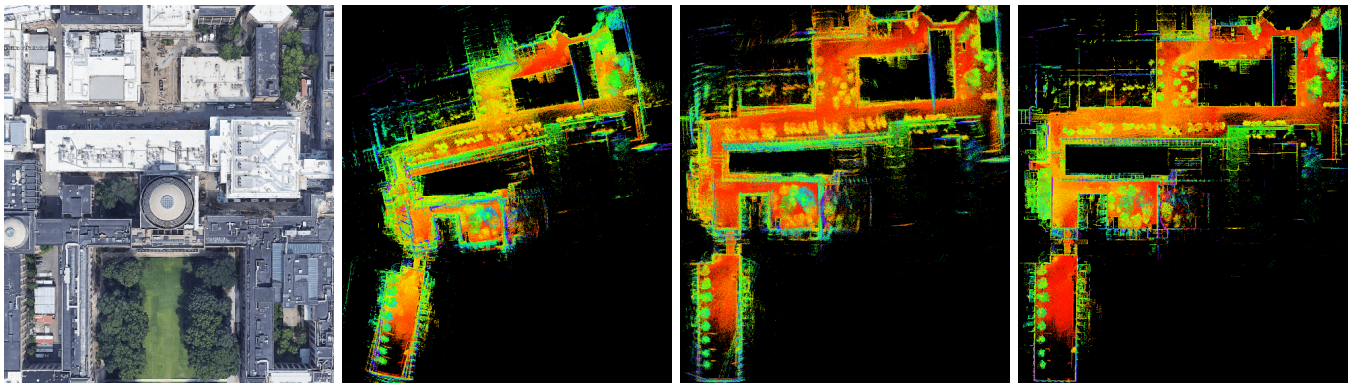
TABLE II: End-to-end translation error (meters)

Dataset	LOAM	LIOM	LIO-odom	LIO-GPS	LIO-SAM
Campus	192.43	Fail	9.44	6.87	0.12
Park	121.74	34.60	36.36	2.93	0.04
Amsterdam	Fail	Fail	Fail	1.21	0.17

This test is designed to show the benefits of introducing GPS and loop closure factors. In order to do this, we purposely disable the insertion of GPS and loop closure factors into the graph. When both GPS and loop closure factors are disabled, our method is referred to as *LIO-odom*, which only utilizes IMU preintegration and lidar odometry factors. When GPS factors are used, our method is referred to as *LIO-GPS*, which uses IMU preintegration, lidar odometry, and GPS factors for graph construction. LIO-SAM uses all factors when they are available.

To gather this dataset, the user walks around the MIT campus using the handheld device and returns to the same position. Because of the numerous buildings and trees in the

[1] https://github.com/TixiaoShan/LIO-SAM
[2] https://youtu.be/A0H8CoORZJU



(a) Google Earth (b) LOAM (c) LIOM (d) LIO-SAM

Fig. 4: Mapping results of LOAM, LIOM, and LIO-SAM using the *Walking* dataset. The map of LOAM in (b) diverges multiple times when aggressive rotation is encountered. LIOM outperforms LOAM. However, its map shows numerous blurry structures due to inaccurate point cloud registration. LIO-SAM produces a map that is consistent with the Google Earth imagery, without using GPS.

mapping area, GPS reception is rarely available and inaccurate most of the time. After filtering out the inconsistent GPS measurements, the regions where GPS is available are shown in Fig. 5(a) as green segments. These regions correspond to the few areas that are not surrounded by buildings or trees.

The estimated trajectories of LOAM, LIO-odom, LIO-GPS, and LIO-SAM are shown in Fig. 5(a). The results of LIOM are not shown due to its failure to initialize properly and produce meaningful results. As is shown, the trajectory of LOAM drifts significantly when compared with all other methods. Without the correction of GPS data, the trajectory of LIO-odom begins to visibly drift at the lower right corner of the map. With the help of GPS data, LIO-GPS can correct the drift when it is available. However, GPS data is not available in the later part of the dataset. As a result, LIO-GPS is unable to close the loop when the robot returns to the start position due to drift. On the other hand, LIO-SAM can eliminate the drift by adding loop closure factors to the graph. The map of LIO-SAM is well-aligned with Google Earth imagery and shown in Fig. 5(b). The relative translational error of all methods when the robot returns to the start is shown in Table II.

D. Park Dataset

In this test, we mount the sensors on a UGV and drive the vehicle along a forested hiking trail. The robot returns to its initial position after 40 minutes of driving. The UGV is driven on three road surfaces: asphalt, ground covered by grass, and dirt-covered trails. Due to its lack of suspension, the robot undergoes low amplitude but high frequency vibrations when driven on non-asphalt roads.

To mimic a challenging mapping scenario, we only use GPS measurements when the robot is in widely open areas, which is indicated by the green segments in Fig. 6(a). Such a mapping scenario is representative of a task in which a robot must map multiple GPS-denied regions and periodically returns to regions with GPS availability to correct the drift.

Similar to the results in the previous tests, LOAM, LIOM, and LIO-odom suffer from significant drift, since no absolute correction data is available. Additionally, LIOM only runs at $0.67\times$ real-time, while the other methods run in real-time. Though the trajectories of LIO-GPS and LIO-SAM coincide in the horizontal plane, their relative translational errors are different (Table II). Because no reliable absolute elevation measurements are available, LIO-GPS suffers from drift in altitude and is unable to close the loop when returning to the robot's initial position. LIO-SAM has no such problem, as it utilizes loop closure factors to eliminate the drift.

E. Amsterdam Dataset

Finally, we mounted the sensor suite on a boat and cruised along the canals of Amsterdam for 3 hours. Although the movement of the sensors is relatively smooth in this test, mapping the canals is still challenging for several reasons. Many bridges over the canals pose degenerate scenarios, as there are few useful features when the boat is under them, similar to moving through a long, featureless corridor. The number of planar features is also significantly less, as the ground is not present. We observe many false detections from the lidar when direct sunlight is in the sensor field-of-view, which occurs about 20% of the time during data gathering. We also only obtain intermittent GPS reception due to the presence of bridges and city buildings overhead.

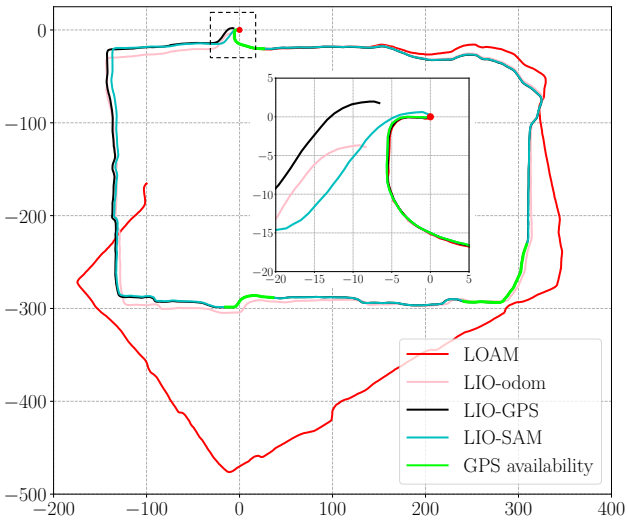
Due to these challenges, LOAM, LIOM, and LIO-odom all fail to produce meaningful results in this test. Similar to the problems encountered in the *Park* dataset, LIO-GPS is unable to close the loop when returning to the robot's initial position because of the drift in altitude, which further motivates our usage of loop closure factors in LIO-SAM.

F. Benchmarking Results

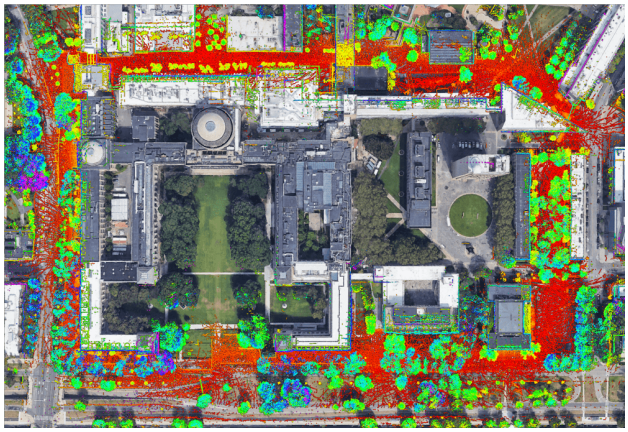
TABLE III: RMSE translation error w.r.t GPS

Dataset	LOAM	LIOM	LIO-odom	LIO-GPS	LIO-SAM
Park	47.31	28.96	23.96	1.09	0.96

Since full GPS coverage is only available in the $Park$ dataset, we show the root mean square error (RMSE) results w.r.t to the GPS measurement history, which is treated as

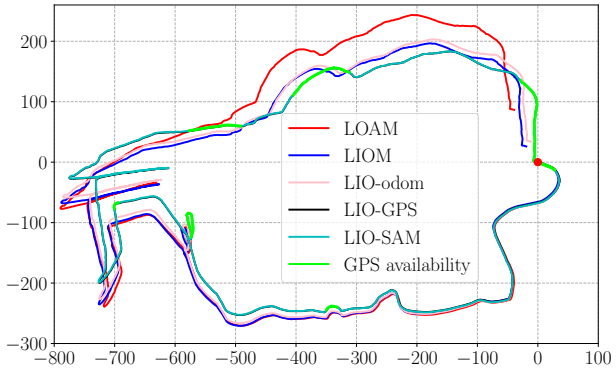


(a) Trajectory comparison

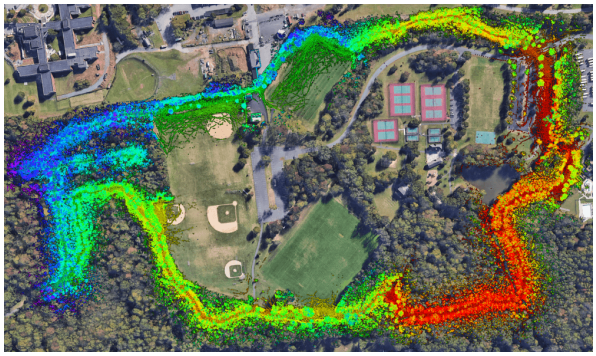


(b) LIO-SAM map aligned with Google Earth

Fig. 5: Results of various methods using the *Campus* dataset that is gathered on the MIT campus. The red dot indicates the start and end location. The trajectory direction is clock-wise. LIOM is not shown because it fails to produce meaningful results.



(a) Trajectory comparison



(b) LIO-SAM map aligned with Google Earth

Fig. 6: Results of various methods using the *Park* dataset that is gathered in Pleasant Valley Park, New Jersey. The red dot indicates the start and end location. The trajectory direction is clock-wise.

TABLE IV: Runtime of mapping for processing one scan (ms)

Dataset	LOAM	LIOM	LIO-SAM	Stress test
Rotation	83.6	Fail	41.9	13×
Walking	253.6	339.8	58.4	13×
Campus	244.9	Fail	97.8	10×
Park	266.4	245.2	100.5	9×
Amsterdam	Fail	Fail	79.3	11×

ground truth. This RMSE error does not take the error along the z axis into account. As is shown in Table III, LIO-GPS and LIO-SAM achieve similar RMSE error with respect to the GPS ground truth. Note that we could further reduce the error of these two methods by at least an order of magnitude by giving them full access to all GPS measurements. However, full GPS access is not always available in many mapping settings. Our intention is to design a robust system that can operate in a variety of challenging environments.

The average runtime for the three competing methods to register one lidar frame across all five datasets is shown in Table IV. Throughout all tests, LOAM and LIO-SAM are forced to run in real-time. In other words, some lidar frames are dropped if the runtime takes more than $100ms$ when the lidar rotation rate is 10Hz. LIOM is given infinite time to process every lidar frame. As is shown, LIO-SAM uses significantly less runtime than the other two methods, which makes it more suitable to be deployed on low-power embedded systems.

We also perform stress tests on LIO-SAM by feeding it the data faster than real-time. The maximum data playback speed is recorded and shown in the last column of Table IV when LIO-SAM achieves similar performance without failure compared with the results when the data playback speed is $1\times$ real-time. As is shown, LIO-SAM is able to process data faster than real-time up to $13\times$.

We note that the runtime of LIO-SAM is more significantly influenced by the density of the feature map, and less affected by the number of nodes and factors in the factor graph. For instance, the *Park* dataset is collected in a feature-rich environment where the vegetation results in a large quantity of features, whereas the Amsterdam dataset yields a sparser feature map. While the factor graph of the Park test consists of 4,573 nodes and 9,365 factors, the graph in the Amsterdam test has 23,304 nodes and 49,617 factors. Despite this, LIO-SAM uses less time in the *Amsterdam* test

Fig. 7: Map of LIO-SAM aligned with Google Earth.

as opposed to the runtime in the Park test.

V. Conclusions and Discussion

We have proposed LIO-SAM, a framework for tightly-coupled lidar inertial odometry via smoothing and mapping, for performing real-time state estimation and mapping in complex environments. By formulating lidar-inertial odometry atop a factor graph, LIO-SAM is especially suitable for multi-sensor fusion. Additional sensor measurements can easily be incorporated into the framework as new factors. Sensors that provide absolute measurements, such as a GPS, compass, or altimeter, can be used to eliminate the drift of lidar inertial odometry that accumulates over long durations, or in feature-poor environments. Place recognition can also be easily incorporated into the system. To improve the real-time performance of the system, we propose a sliding window approach that marginalizes old lidar frames for scan-matching. Keyframes are selectively added to the factor graph, and new keyframes are registered only to a fixed-size set of sub-keyframes when both lidar odometry and loop closure factors are generated. This scan-matching at a local scale rather than a global scale facilitates the real-time performance of the LIO-SAM framework. The proposed method is thoroughly evaluated on datasets gathered on three platforms across a variety of environments. The results show that LIO-SAM can achieve similar or better accuracy when compared with LOAM and LIOM. Future work involves testing the proposed system on unmanned aerial vehicles.

References

[1] J. Zhang and S. Singh, "Low-drift and Real-time Lidar Odometry and Mapping," *Autonomous Robots*, vol. 41(2): 401-416, 2017.

[2] A. Geiger, P. Lenz, and R. Urtasun, "Are We Ready for Autonomous Driving? The KITTI Vision Benchmark Suite", *IEEE International Conference on Computer Vision and Pattern Recognition*, pp. 3354-3361, 2012.

[3] P.J. Besl and N.D. McKay, "A Method for Registration of 3D Shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14(2): 239-256, 1992.

[4] A. Segal, D. Haehnel, and S. Thrun, "Generalized-ICP," *Proceedings of Robotics: Science and Systems*, 2009.

[5] W.S. Grant, R.C. Voorhies, and L. Itti, "Finding Planes in LiDAR Point Clouds for Real-time Registration," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4347-4354, 2013.

[6] M. Velas, M. Spanel, and A. Herout, "Collar Line Segments for Fast Odometry Estimation from Velodyne Point Clouds," *IEEE International Conference on Robotics and Automation*, pp. 4486-4495, 2016.

[7] T. Shan and B. Englot, "LeGO-LOAM: Lightweight and Ground-optimized Lidar Odometry and Mapping on Variable Terrain," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4758-4765, 2018.

[8] T. Shan, J. Wang, K. Doherty, and B. Englot, "Bayesian Generalized Kernel Inference for Terrain Traversability Mapping," *In Conference on Robot Learning*, pp. 829-838, 2018.

[9] S. Lynen, M.W. Achtelik, S. Weiss, M. Chli, and R. Siegwart, "A Robust and Modular Multi-sensor Fusion Approach Applied to MAV Navigation," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 3923-3929, 2013.

[10] S. Yang, X. Zhu, X. Nian, L. Feng, X. Qu, and T. Mal, "A Robust Pose Graph Approach for City Scale LiDAR Mapping," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1175-1182, 2018.

[11] M. Demir and K. Fujimura, "Robust Localization with Low-Mounted Multiple LiDARs in Urban Environments," *IEEE Intelligent Transportation Systems Conference*, pp. 3288-3293, 2019.

[12] Y. Gao, S. Liu, M. Atia, and A. Noureldin, "INS/GPS/LiDAR Integrated Navigation System for Urban and Indoor Environments using Hybrid Scan Matching Algorithm," *Sensors*, vol. 15(9): 23286-23302, 2015.

[13] S. Hening, C.A. Ippolito, K.S. Krishnakumar, V. Stepanyan, and M. Teodorescu, "3D LiDAR SLAM integration with GPS/INS for UAVs in urban GPS-degraded environments," *AIAA Infotech@Aerospace Conference*, pp. 448-457, 2017.

[14] C. Chen, H. Zhu, M. Li, and S. You, "A Review of Visual-Inertial Simultaneous Localization and Mapping from Filtering-Based and Optimization-Based Perspectives," *Robotics*, vol. 7(3):45, 2018.

[15] C. Le Gentil,, T. Vidal-Calleja, and S. Huang, "IN2LAMA: Inertial Lidar Localisation and Mapping," *IEEE International Conference on Robotics and Automation*, pp. 6388-6394, 2019.

[16] C. Qin, H. Ye, C.E. Pranata, J. Han, S. Zhang, and Ming Liu, "R-LINS: A Robocentric Lidar-Inertial State Estimator for Robust and Efficient Navigation," *arXiv:1907.02233*, 2019.

[17] H. Ye, Y. Chen, and M. Liu, "Tightly Coupled 3D Lidar Inertial Odometry and Mapping," *IEEE International Conference on Robotics and Automation*, pp. 3144-3150, 2019.

[18] F. Dellaert and M. Kaess, "Factor Graphs for Robot Perception," *Foundations and Trends in Robotics*, vol. 6(1-2): 1-139, 2017.

[19] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J.J. Leonard, and F. Dellaert, "iSAM2: Incremental Smoothing and Mapping Using the Bayes Tree," *The International Journal of Robotics Research*, vol. 31(2): 216-235, 2012.

[20] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "On-Manifold Preintegration for Real-Time Visual-Inertial Odometry," *IEEE Transactions on Robotics*, vol. 33(1): 1-21, 2016.

[21] T. Moore and D. Stouch, "A Generalized Extended Kalman Filter Implementation for The Robot Operating System," *Intelligent Autonomous Systems*, vol. 13: 335-348, 2016.

[22] G. Kim and A. Kim, "Scan Context: Egocentric Spatial Descriptor for Place Recognition within 3D Point Cloud Map," *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4802-4809, 2018.

[23] J. Guo, P. VK Borges, C. Park, and A. Gawel, "Local Descriptor for Robust Place Recognition using Lidar Intensity," *IEEE Robotics and Automation Letters*, vol. 4(2): 1470-1477, 2019.

[24] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A.Y. Ng, "ROS: An Open-source Robot Operating System," *IEEE ICRA Workshop on Open Source Software*, 2009.

[25] T. Qin, P. Li, and S. Shen, "Vins-mono: A Robust and Versatile Monocular Visual-Inertial State Estimator," *IEEE Transactions on Robotics*, vol. 34(4): 1004-1020, 2018.