Article

The International Journal of Robotics Research
2025, Vol. 44(9) 1553–1599

DOI: 10.1177/02783649251316881
journals.sagepub.com/home/ijr

A flexible framework for accurate LiDAR odometry, map manipulation, and localization

Jose Luis Blanco-Claraco

Abstract

Light Detection and Ranging (LiDAR)-based simultaneous localization and mapping (SLAM) is a core technology for autonomous vehicles and robots. One key contribution of this work to 3D LiDAR SLAM and localization is a fierce defense of view-based maps (pose graphs with time-stamped sensor readings) as the fundamental representation of maps. As will be shown, they allow for the greatest flexibility, enabling the posterior generation of arbitrary metric maps optimized for particular tasks, for example, obstacle avoidance and real-time localization. Moreover, this work introduces a new framework in which mapping pipelines can be defined without coding, defining the connections of a network of reusable blocks much like deep-learning networks are designed by connecting layers of standardized elements. We also introduce tightly-coupled estimation of linear and angular velocity vectors within the Iterative Closest Point (ICP)-like optimizer, leading to superior robustness against aggressive motion profiles without the need for an IMU. Extensive experimental validation reveals that the proposal compares well to, or improves, former state-of-the-art (SOTA) LiDAR odometry systems, while also successfully mapping some hard sequences where others diverge. A proposed self-adaptive configuration has been used, without parameter changes, for all 3D LiDAR datasets with sensors between 16 and 128 rings, and has been extensively tested on 83 sequences over more than 250 km of automotive, handheld, airborne, and quadruped LiDAR datasets, both indoors and outdoors. The system flexibility is demonstrated with additional configurations for 2D LiDARs and for building 3D NDT-like maps. The framework is open-sourced online: https://github.com/MOLAorg/mola.

Keywords

SLAM, LiDAR-odometry, localization

Received 21 August 2024; Revised 22 November 2024; Accepted 27 December 2024
Senior Editor: Luca Carlone
Associate Editor: Yulun Tian

1. Introduction

3D LiDAR is among the most widely-used sensors in contemporary mobile robots, autonomous vehicles, and Unmanned Aerial Vehicles (UAV) (Elhousni and Huang, 2020; Lee et al., 2023). While vision and RADAR are also relevant technologies and may become essential for many tasks, LiDAR will likely remain a core sensor for years to come (Roriz et al., 2022) due to a number of intrinsic advantages: direct and accurate 3D sensing with wide field of view, robustness against varying sunlight conditions, reduced computation demand to obtain dense point clouds, etc. (as illustrated in Figure 1).

Among all the potential applications of LiDAR sensing, this work focuses on the problems of pose tracking and mapping. We call *localization* to pose tracking on a prebuilt environment model that remains unmodified. Positioning in unknown environments becomes the well-known Simultaneous Localization and Mapping (*SLAM*) problem (Cadena et al., 2016). When only the most recently-observed parts of the environment are kept, the term *odometry* is used instead, hence we will leave the category SLAM for those approaches that are aware of loop closures (Grisetti et al., 2010). That is, odometry methods provide local accurate estimation of relative motion in the short term, while SLAM must also ensure global consistency of both, the trajectory and the world model.

Therefore, when LiDAR is the unique sensor in an odometry system it is called LiDAR odometry (LO). The main component presented in this work falls into that category. LO can be used alone or as part of a larger system,

Department of Engineering, University of Almería, La Cañada, Spain

Corresponding author:
José Luis Blanco-Claraco, Department of Engineering, University of Almería, La Cañada, Spain.
Email: jlblanco@ual.es

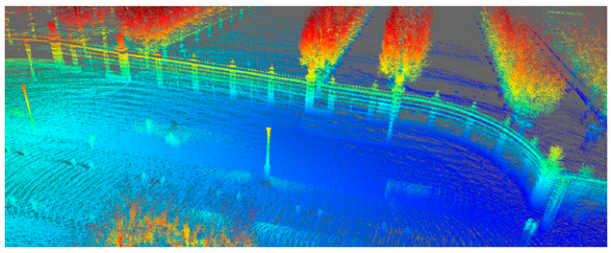


Figure 1. Example point cloud built from the presented LiDAR odometry system: a view of *Esplanade Gaston Monnerville* (Paris) from the Luxembourg Garden automotive dataset in Dellenbach et al. (2022). This is part of a multi-layer metric map, illustrated in Figure 10.

leading to full metric SLAM, for example, Cartographer in Hess et al. (2016), or multi-robot SLAM systems, for example, multi S-Graphs in Fernandez-Cortizas et al. (2024). In the literature, we can find how additional sensors have been successfully integrated with LO, leading to LiDAR Inertial Odometry (LIO) (Shan et al., 2020; Xu et al., 2022a), Visual Inertial Odometry (VIO) (Forster et al., 2016; Leutenegger et al., 2015), or Visual Lidar-Inertial Odometry (VLIO) (Shan et al., 2021). However, we will show that, for most conditions found in practice, our system is able to achieve good performance and accuracy relying on LiDAR alone, notwithstanding that even better results might arise from multimodal sensor fusion.

The present work defends that LO and SLAM systems can be built in a way that maximizes the reusability of their components, making them more flexible and easier to modify by end users, while also allowing resulting world models ("maps") to be built minimizing the loss of information with respect to raw sensor data. Two core ideas arise from these goals: (i) odometry and SLAM pipelines, together with their associated data structures, should be refactored into their smallest reusable components, allowing building whole systems from scratch by wiring them together and setting their parameters much like Deep Neural Networks (DNN) are designed by stacking prebuilt elements; (ii) disregarding the specific map type or features used by a particular LO or SLAM system, the most versatile map data structure is a *view-based map* (dubbed "simple maps" in our framework), where a sparse set of key-frames are retained together with their fundamental information, for example, raw sensor observations and kinematic state of the vehicle. The first usage of the term view-based SLAM can be found in the 2000s (Eustice et al., 2005; Konolige et al., 2010), although the idea can be traced back to the ground-breaking work Lu and Milios (1997) which served as the seed for an extremely successful approach to mapping dubbed Graph-SLAM (Grisetti et al., 2010). A more recent application of this idea of keeping the raw sensor information in graph-based SLAM can be found within Google's Cartographer (Hess et al., 2016) open-source implementation (refer to "assets writer"), although not mentioned in the original publication.

Therefore, building upon the two ideas above, this work introduces an open-source software ecosystem to the community. Note that none of the presented algorithms are based on learning methods, although the modular nature of the framework allows their integration as future works. To sum up, the key contributions of this work are:

- A software ecosystem providing reusable components for building point cloud processing pipelines without coding, including generation of voxel maps, occupancy 2D or 3D grid maps, and feature detection.
- A reference pipeline for LO with 3D LiDARs, well tested against more than 80 public dataset sequences.
- A method to easily integrate vehicle velocity as an unknown within ICP loops, allowing better point cloud undistortion, subject to a constant velocity assumption during each sensor sweep.
- A simple method to let ICP take into account prior uncertainty, including that from a constant velocity motion model or other sources of odometry, for example, wheel encoders or an IMU.
- A loop-closure algorithm to generate globally consistent maps from the LO outcome, optionally including georeferencing.
- A uniform API (Application Programming Interface) to access a large variety of robotics datasets, simplifying and easing benchmarking SLAM methods.

The rest of the article is organized as follows. Section 2 reviews previous works closely related to the presented contributions. Next, the proposed architecture and their individual parts are introduced in detail in Section 3. Experimental validation is provided in three parts: Section 4 presents extensive quantitative performance metrics of the proposed SLAM system, Section 5 illustrates in a qualitative way some use cases of our system, including situations and datasets where other SOTA systems diverge, and Section 6 shows localization results with previously-built maps. Furthermore, ablation studies are presented in Section 7 and we wrap up with conclusions in Section 8.

2. Related work

This section analyzes many of the past research directly related to the main topic of this paper: LiDAR odometry or SLAM. LiDAR works have been divided into odometry, loop-closure, and localization, with an additional brief subsection on graph SLAM. We will put a focus on those ideas that have been recurrently proposed in the literature due to their good results, especially those in which our system is based on. An extensive review of these fields is out of the scope of the present work; interested readers may refer to existing surveys such as Cadena et al. (2016), Bresson et al. (2017), or Xu et al. (2022b); or to Durrant-Whyte and Bailey (2006) and Bailey and Durrant-Whyte (2006) for a summary of

foundational ideas on which modern SLAM methods were built upon.

2.1. LiDAR odometry (LO)

The goal of a LO system is taking successive LiDAR scans and estimating the vehicle pose changes in between them. At the core of most proposed systems there exist modified versions of the Iterative Closest Point (ICP) algorithm (Besl and McKay, 1992), which solves that nonlinear problem in an iterative fashion, finding successive approximations for the sough relative pose between two input point clouds, looking for potential point-to-point pairings at each iteration. That process is often called scan matching or point cloud registration.

We can extract four key findings from past related works. First of all, existing approaches can be classified according to whether they try to match the latest observation against other past individual observations (the scan-to-scan approach), or against an incrementally-built local map (the scan-to-model or scan-to-map approach). It has been experimentally found that scan-to-scan works extremely well for 2D LiDARs, whereas the uneven distribution of measured points in 3D LiDARs favors the scan-to-model approach, as demonstrated with recent successful works like Cartographer (Hess et al., 2016), IMLS-SLAM (Deschaud, 2018), or KISS-ICP (Vizzo et al., 2023). Following this line, the system proposed in this work hence also adheres to the scan-to-model design (see Section 3.4).

Second, it is well known that trying to use ICP to register the raw points from a 3D LiDAR leads to extremely poor results. This can be explained by the uneven distribution of sampled points in the environment, with a higher density of near points that then dominate the cost function of the ICP optimization stage. This fact, together with the common use of LiDARs with a circular scanning pattern, leads to degenerate ICP solutions where nearby rings are aligned with nearby rings, disregarding the contribution from points sampled from distant parts of the environment. To overcome this problem, downsampling has been extensively used in the literature. One of the first successful LO systems, LOAM, introduced in Zhang and Singh (2014), splits the input clouds into two distinct subsets (corners and planar patches), performing a subsampling to keep only a maximum of 2 edge or 4 planar points per planar scan. This seminal work led to many variations, such as Lego-LOAM in Shan and Englot (2018) or PaGO-SLAM in Seo et al. (2022) which exploited an explicit ground plane segmentation. One of the most prominent derived works is Fast-LIO2 (Xu et al., 2022a), which leverages tightly-coupled IMU-LiDAR integration with efficient data structures for fast and accurate mapping without the need to extract features. Other recent works (Deschaud, 2018; Vizzo et al., 2023) employ voxel-based sampling, where raw points are first classified into 3D voxel data structures and then a single representative point is extracted from each occupied voxel following a given heuristic criterion (e.g., the average and the first point). LOCUS 2.0, introduced in Reinke et al. (2022), proposes an adaptive multi-level sampling of point clouds to try to maintain a constant number of points per scan. Following the same goal, another recent work dubbed SIMPLE (Bhandari et al., 2024) also performs spatial subsampling, but implemented via KD-trees instead of voxels. Efficient sampling of point clouds for registration is a research field on its own. The libpointmatcher project, started with Pomerleau et al. (2013), implements different strategies, such as those proposed in Rusinkiewicz and Levoy (2001) or in Labussière et al. (2020). Latest works tend to use simpler sampling schemes, but we have experimentally verified that even tiny implementation differences in the way points are sampled (e.g., using different hash functions in unordered voxel containers for successive downsampling stages, which determines which is the first point to fall into a given voxel) have a relevant impact in the accuracy of the obtained trajectories. Therefore, our system offers simple voxel-based sampling schemes as the default choice, but it also offers other possibilities for the community to experiment and benchmark them, while also being easily extensible with new sampling algorithms (see Section 3.3), including those based on deep learning, for example, following Lang et al. (2019).

Third, all modern approaches to LO have replaced the original closed-form least-squares formulation in the original ICP paper (Besl and McKay, 1992) with less efficient but more robust alternatives. In particular, instead of leaving the cost function to uniquely depend on the L2 distances between pointwise pairings, two enhancements are nowadays common: (i) employing robustified least squares, that is, robust kernel or loss functions as in Chebrolu et al. (2021), Deschaud (2018), and Vizzo et al. (2023), and (ii) using more geometric constraints apart of point-to-point pairings. On this latter point, the most common alternative is using point-to-plane pairings, as used in the original LOAM (Zhang and Singh, 2014) or, more recently, in Fast-LIO2 (Xu et al., 2022b) and in MAD-SLAM (Ferrari et al., 2024). LOCUS 2.0 (Reinke et al., 2022) introduces the estimated normals in the cost function, while another prominent work in this sense, MULLS (Pan et al., 2021), includes several potential geometric pairings simultaneously. A word is in order about SIMPLE (Bhandari et al., 2024), which employs a different approach: its cost function avoids the determination of point-to-point pairings by just adding Gaussian-like "rewards" for each nearby point, which can be seen as a one-to-many point pairing policy. Learning from all this body of past works, our framework provides a generic ICP-like optimization algorithm at its core, with user-extensible cost functions, predefined cost functions for point-to-point, point-to-plane or point-to-line (among others), different robust kernels, and the possibility of multiple correspondences (see Section 3.6).

The fourth lesson learned from past works comes from KISS-ICP (Vizzo et al., 2023), which demonstrated that dynamically adaptable parameters are beneficial for two reasons: (i) they can improve the overall accuracy since

parameters adapt automatically as the environment or the motion profile change over time, and (ii) they reduce the need for manually tuning parameters. From that work we have adopted (and modified) their idea of using an adaptive threshold for determining pairings, and extended the idea to many other parameters along the entire processing pipeline (see Section 3.5).

2.2. LiDAR localization

Localization with a predefined, static map and a 3D LiDAR is a topic with less research activity than LO and SLAM, despite the fact that one of the fundamental reasons one may want to build a map is to use it for navigation, which implies the ability to keep the vehicle localized in the map frame of reference. As in any other estimation problem, we could split classic approaches (those not based on deep learning) into those relying on *parametric* or on *non-parametric* distributions. The latter typically involves using particle filtering (Thrun et al., 2001), which are an excellent approach for vehicles constrained to trajectories on SE(2), but whose performance may degrade due to the curse of dimensionality for SE(3) motion. Still, it was shown in Blanco-Claraco et al. (2019) how it is possible to keep a vehicle well localized in SE(3) using a particle filter, from raw 3D LiDAR data and wheels odometry, much faster than real time. This line of work was enhanced by detecting and tracking features (pole-like objects) in Schaefer et al. (2021), still using particle filtering. On the other hand, parametric methods for localization, such as Hess et al. (2016) or Lee and Ryu (2024), typically rely on an ICP-like optimization loop to find the best vehicle pose at a given instant.

Since we believe that both approaches have their applicability niches, both solutions are provided in the present framework (see Section 6).

2.3. Graph optimization

The problem of finding the global optimal for a set of rigid transformations between a sequence of key-frames is a very well-studied one in the robotics community, either in its planar SE(2) or spatial SE(3) form. It has been dubbed Graph-SLAM (Grisetti et al., 2007, 2010, 2012), pose-graph optimization (Tian et al., 2021), or synchronization if mostly interested in the rotational parts (Carlone et al., 2015; Wang and Singer, 2013).

The goal of all these methods is to find the poses between the key-frames that minimize the mismatch of (typically sparse) relative pose observations. These techniques have an obvious direct application to optimization-based (Strasdat et al., 2012) odometry frameworks that follow the frame-to-frame paradigm. Despite the present work follows the frame-to-map model instead, we mention graph optimization here since it becomes essential when handling loop closures.

The most relevant graph optimization libraries are reviewed next, with a focus on those tailored to graph SLAM rather than bundle adjustment. One of the earliest works in this field is probably TORO (Grisetti et al., 2007b, 2007a, 2009), which implemented efficient gradient descent for pose networks of arbitrary topology (multiple loop closures). Next, G2O was presented in Kümmerle et al. (2011), becoming extremely popular, to the point that it is still used nowadays in SOTA frameworks. Three G2O features that explain this success are: (i) its modular design that allowed for arbitrary graphs to be defined, including pose-only graphs and those required for visual bundle adjustment (Triggs et al., 2000), (ii) the popularization of perturbation-based optimization on Lie groups (the $\boxplus$ notation) (Blanco-Claraco, 2021; Kümmerle et al., 2011; Sola et al., 2018), and (iii) the support for robust kernels (Chebrolu et al., 2021), essential for outlier rejection.

Afterward, two other popular frameworks were introduced: GTSAM (Dellaert, 2021), and CERES (Agarwal et al., 2022). Both are frameworks for solving large, non-linear optimization problems, and both feature robust kernels and some degree of automated calculation of cost function derivatives. In our loop closure subsystem (described in Section 3.12), we chose GTSAM as the graph optimization back-end.

2.4. LiDAR-based loop closure and SLAM

In the context of SLAM, loop closure can be defined as the detection of when the robot has revisited a placed that is already known, often after traversing a path that resembles a closed loop. *Topological* loop closure just detects such events, while *metrical* loop closure methods also identify the relative pose between the old and the new observations.

What does actually mean a loop closure in practice, and the way the problem is approached, are both strongly influenced by the employed sensors. For visual SLAM with typical pin-hole cameras, moving a camera around a small room would lead to loop closures, while for a 3D LiDAR with 360° of horizontal field of view one might never consider a loop closure situation if moving in a large warehouse if there are no severe occlusions. This variety of situations makes this research field extremely active for each sensor niche (Arshad and Kim, 2021; Huang, 2019; Tsintotas et al., 2022; Wang et al., 2019).

Regarding the topic of this work, LiDAR sensors, a recent work by Kim et al. (2021) introduced a new rotation-invariant descriptor for performing topological and (partial) metric localization. Similar ideas are proposed in related works like LiDAR-IRIS (Wang et al., 2020) which uses a polar representation of 3D point clouds, or in Gupta et al. (2024).

The system presented in this work, however, does not rely on topological loop closure detection (although it would be integrated as future works): instead, we rely on pure ICP-based alignment with a special ICP pipeline different than the one used for regular LO, fed by initial

guesses from consumer-grade GNSS, if available. This simple approach has demonstrated good enough for all the datasets explored in this work.

3. Proposed architecture

This section describes in detail the different parts of the proposed system. The next subsection starts providing a top-level overview of the key elements, and subsequently they are explained individually from a scientific and algorithmic point of view.

3.1. Overview

Pursuing the goals defined in the Introduction, we define the basic blocks illustrated in Figure 2. Among them, three essential types of procedural blocks or algorithms are defined. First, we have the odometry and loop-closure subsystems. The idea of splitting SLAM into odometry ("local mapping") and loop-closure ("global mapping") can be traced back to the visual SLAM community with PTAM (Klein and Murray, 2007) and has been adopted by most subsequent successful solutions for visual SLAM (RTAB-Map Labbe and Michaud (2014), ORB-SLAM Campos et al. (2021)) or LiDAR SLAM (Cartographer Hess et al. (2016)). These two blocks are discussed in Sections 3.4 and 3.12, respectively.

Second, we define generic metric mapping pipelines as one of the core elements in our framework, since they are used internally as building blocks for both, odometry and loop-closure, apart of having additional applications. They can be thought of as arbitrary combinations of algorithm as processing nodes in a data-driven network, with data being metric maps. Their final goal is taking raw sensory data (or a former metric map) as inputs, and processing them to create a new metric map (or to modify the existing one). They are introduced in Section 3.3.

The input and output of the algorithms defined in Figure 2 are data-related elements:

3.1.1. Raw sensor data source. This represents the source of sensor data, which can be either a real live device or an offline dataset. Real-world applications will normally use the former, while benchmarking and development predominantly use the latter for convenience. Available options in our framework are enumerated in Section 3.13.

3.1.2. Metric maps. Most existing methods in the literature use one single metric map representation (e.g., point clouds in Deschaud (2018) or Vizzo et al. (2023)) while others maintain two (e.g., corner and plane points in LOAM (Zhang and Singh, 2017) and most derived works). The present work proposes allowing the definition of an arbitrary set of metric map representations and data structure implementations. Some of them will be more useful during real-time mapping, others during map postprocessing or loop closing, others for obstacle avoidance, etc. More on metric maps follows up in Section 3.2.

3.1.3. View-based maps. Dubbed "simple-maps" in our implementation, they contain a subset of all incoming raw sensor data, synchronized and paired with the estimated pose according to an odometry or SLAM method to form key-frames. More details are given in Section 3.9.

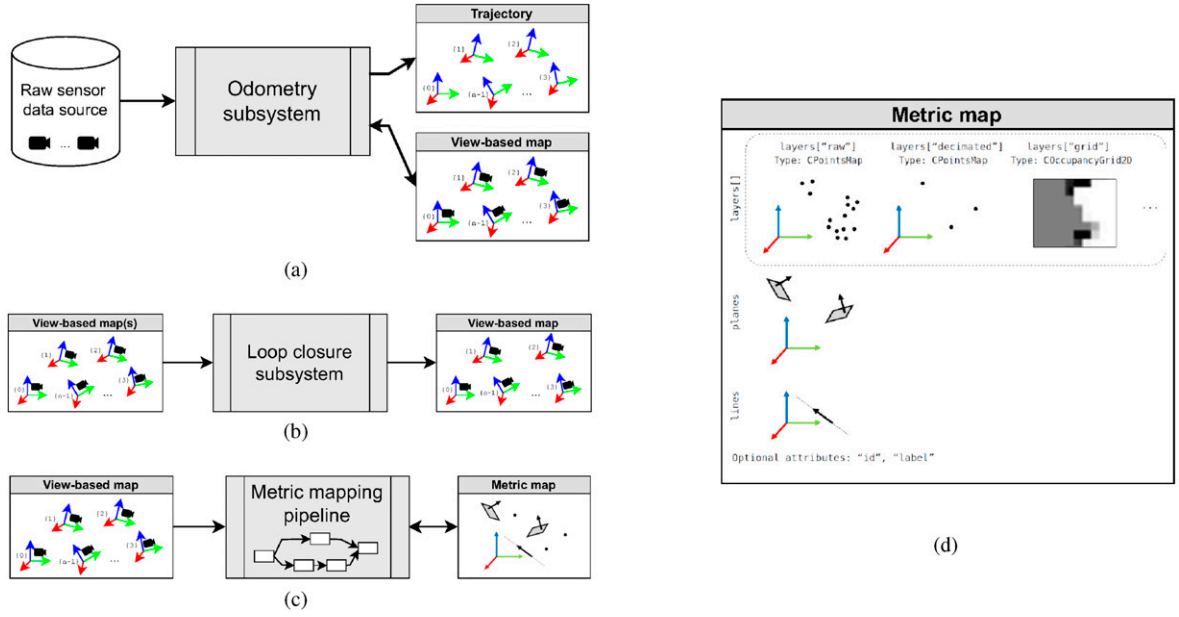


Figure 2. Main top-level data structures and procedural components of the presented framework: (a) odometry and (b) loop-closure subsystems, (c) metric map generation or update pipelines, and (d) a scheme of the contents of a metric map. Refer to discussion in Section 3.1. The contents of each subsystem are detailed in subsequent figures.

3.1.4. Trajectories. A trajectory $\mathbf{t}$ is simply a collection of n SE(3) timestamped poses, for time steps t_i with $i = 1, \ldots, n$:

$$\mathbf{t} = \{(t_1, \mathbf{T}_1), \ldots, (t_n, \mathbf{T}_n)\} \quad (1)$$

where following the RIGID notation for poses (Nadeau, 2024), ${}_w\mathbf{T}_i \equiv \mathbf{T}_i$ stands for the ith SE(3) pose of the vehicle local reference frame with respect to the world (w), that is, in global coordinates.

3.2. Metric maps

A metric map in the presented framework comprises a superposition of multiple underlying individual maps called *layers*, plus optional sets of planes and lines, as sketched in Figure 2(d).

The idea is using such compound metric maps as the fundamental data containers used as both inputs and outputs of metric map pipelines (see Section 3.3) and as inputs to the ICP-like optimization pipeline (see Section 3.6). The existence of different map layers is justified by two facts:

- Layer map types are independent, for example, some layers may be point clouds, others are occupancy voxel maps, each with a different resolution, etc.
- Layers have diverse semantics, for example, corners, planes, vehicles, pole-like objects, a horizontal slice of a 3D map useful for detecting walls, etc.

Such layer differentiation is useful during mapping, but also in posterior post-processing stages. Our framework allows the definition of new metric map types by users by means of a virtual base interface that abstracts all common operations of maps such as clearing, inserting sensor observations, evaluating the likelihood of a given observation, searching for nearest neighbors, etc. Having such abstract interface does not only allow for the extensibility and generic programming in our framework, but also enables performing performance benchmarking to take optimal decisions about what map type to use for each application. Next, we enumerate the metric map types that have been used in the experiments presented in this work:

1. Point clouds with contiguous storage layout in memory for each point component: This group of types include maps for simple clouds where each point only has (X, Y, Z) components, with an intensity (I) component (X, Y, Z, I), and with LiDAR ring (R) identifier and per-point timestamps (T), that is, (X, Y, Z, I, R, T) components.
2. Point clouds with a hashed voxels storage layout: this map type implements 3D voxels indexed by voxel coordinates using the optimal hash function introduced in Teschner et al. (2003), with a hard-coded maximum number of points. This is the most common map layer type in most parts of the proposed system, and it is based on similar successful implementations in past works (Deschaud, 2018; Vizzo et al., 2023).
3. Occupancy 3D voxel map: this type of map employs the re-implementation in Faconti (2023) of Volumetric Dynamic B + Trees (VDB), originally introduced by *DreamWorks Animation* in Museth (2013). It offers an efficient hierarchical data structure to cover large 3D volumes with high-resolution voxels. VDB could be used to store any arbitrary data within voxels, but in our framework, we so far only offer two map types: one to store the occupancy of each voxel, and another one that also includes its color. This map type has been used to represent both, 2D and 3D occupancy grid maps, especially if they are sparse.
4. 3D-NDT (Normal Distribution Transform) maps, as proposed in Magnusson et al. (2007), where a 3D sparse voxel map stores fitted Gaussian distributions for points in each voxel. In our implementation, voxels with a sufficient number of points whose spatial distribution clearly defines a plane store the mean and covariance matrix of such points, whereas voxels with too few samples or which are not planar enough retain all individual points. The idea is to enable our map to establish both, point-to-point pairings (for voxels which are not planar) and point-to-plane pairings (for planar ones). In particular, the NDT representation for a voxel is used if $\sigma_1/\sigma_2 < \tau$, with $\{\sigma_1, \sigma_2, \sigma_3\}$ the three eigenvalues of the voxel point distribution sorted in ascending order and τ a threshold (0.05 in our experiments). As the map is populated with incoming observations, individual voxels may switch between the two representations.
5. 2D grid maps with a plain contiguous memory layout: The most efficient data structure for 2D occupancy grids that neither, are very sparse, nor need to grow the map limits often. These maps have been popular in mobile robotics since Elfes (1987).

3.3. Metric map generation and update pipelines

Metric map pipelines are at the core of the flexibility offered by the present framework. Such pipelines are defined as a bipartite, directed graphs with two kinds of nodes: data nodes (metric maps, as defined in Section 3.2) and action nodes (either filters or generators). Directed edges may only exist between data nodes and action nodes. Follow Figure 3 as a reference for the discussion below, keeping in mind that despite map layers are represented there as point clouds, arbitrary map types are allowed as long as they are compatible with their action nodes.

Generators (see Figure 3(a)) are especial actions as they are the entry points for sensor data, transforming raw range data, images, depth images, etc. into more elaborated data structures, for example, point clouds with different per-point field annotations, or plane patches normals. They can also generate more than one output metric map layers, becoming the ideal location where to run classification tasks, for example, detecting edges or special objects in depth cameras or LiDAR observations. We provide hooks for

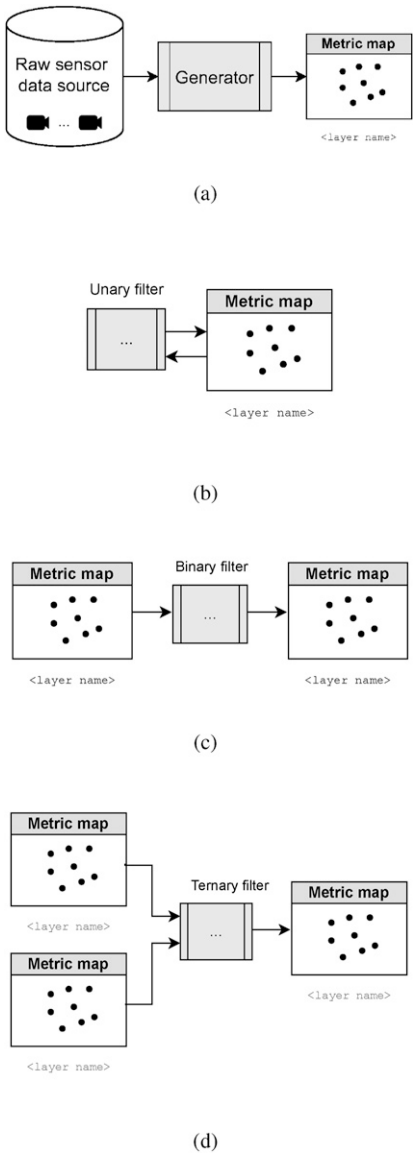


Figure 3. Metric map pipelines represent map processing networks as bipartite graphs with two kinds of nodes: data (map layers) and actions (generators or filters). Different connection patterns exist: (a) generators converting raw observations into map layers, and filters of different arity such as (b) unary, (c) binary, or (d) ternary map filters. See discussion in Section 3.3.

implementing custom detection algorithms that need to exploit the particular data structure for each possible input sensor, for example, edge detection algorithms from LiDAR range images. A generic default generator exists which just creates a map layer of a given type (by default, a point cloud) and inserts the observation into it by using the abstract interface mentioned in Section 3.2. For 2D or 3D LiDARs, this simply creates an unfiltered point cloud with all sensed points in a given scan.

Filters are the other kind of action in this framework, and are broadly defined as any arbitrary algorithm that takes at least one map layer as input and generates or modifies other (or the same input) layers. Our work already ships several such filters, while also offering a plug-in system to add user-defined ones. Some of the most important filters, used in the LiDAR odometry pipeline presented in Section 3.7, are briefly introduced next:

3.3.1. De-skewing. Most common LiDAR sensors today are rotating scanners, where a significant time elapses between the first and the last points in a complete 360° scan (typically between 50 ms and 200 ms). De-skewing or undistortion is a well-known scan data pre-processing stage where the *estimated* vehicle motion is used to interpolate the sensor trajectory in SE(3) over time during the scan period in order to project all sensed points to 3D space in the most precise way. Given a set of raw (distorted) points ${}_s\widetilde{\mathbf{R}} = \{{}_s\widetilde{\mathbf{r}}_i\}_{i=1}^N$ in the sensor ("s") frame of reference[1], with associated *relative* time-stamps t_i such as for $t = 0$ the sensor pose in the map frame is $\mathbf{T}_s = \{\mathbf{p}_s, \mathbf{R}_s\}$ (with $\mathbf{p}_s \in \mathbb{R}^3$ the translation and $\mathbf{R}_s \in SO(3)$ the orientation), with linear velocity ${}_s\boldsymbol{v}_s$ and angular velocity ${}_s\boldsymbol{\omega}_s$ (both in the sensor frame "s"), simple extrapolation on SE(3) is performed to obtain the corrected point cloud $\mathbf{R} = \{\mathbf{r}_i\}_{i=1}^N$ in the map frame:

$$\begin{bmatrix}\mathbf{r}_i \\ 1\end{bmatrix} = \begin{bmatrix}\mathbf{R}_s & \mathbf{p}_s \\ \mathbf{0}_{1\times 3} & 1\end{bmatrix} \begin{bmatrix}(t_i\ {}_s\boldsymbol{\omega}_s)^\wedge & t_i\ {}_s\boldsymbol{v}_s \\ \mathbf{0}_{1\times 3} & 1\end{bmatrix} \begin{bmatrix}{}_s\widetilde{\mathbf{r}}_i \\ 1\end{bmatrix} \quad (2)$$

with $\cdot^\wedge$ the hat or wedge operator, mapping SO(3) tangent space vectors to rotation matrices (Blanco-Claraco, 2021).

3.3.2. Time-stamp adjustment. Note that the exact moment used as a reference for relative time-stamps in the de-skewing filter above is not defined. Different conventions can be found in existing implementations, with the most common ones being using the first point or the one in the middle as references. This decision is coupled with the exact moment at which the vehicle pose and velocity are to be estimated by the LO or SLAM system, and for vehicles moving fast the difference between criteria may be in the order of tens of centimeters. Further complications arise from the diversity of per-point timestamp formatting in public datasets and sensor drivers, ranging from relative times in seconds, relative times in the fixed range [0,1], to just using absolute time stamps. Therefore, we provide this filter to adjust the timestamps in all incoming LiDAR scans to ensure they observe one of a list of possible criteria.

3.3.3. Down-sampling. As shown while discussing related LO works (Section 2.1), sub-sampling point clouds is essential to achieve stable ICP-based mapping. Our framework offers several down-sampling filters, ranging from simply picking the first point within each 3D voxel for a given uniform volumetric resolution, to keeping the average of each such voxels, or using nonlinear spatial tessellation.

3.3.4. Spatial filtering. We also provide ternary filters to split an incoming map layer in two categories: those points passing a given criterion and those that do not. Filters

include applying 3D bounding boxes (i.e., points within the box pass the test), checking for a minimum distance to a given point (e.g., the vehicle), or splitting by LiDAR ring number.

3.3.5. Map insertion. This fundamental binary filter takes an input point cloud layer and an output layer of an arbitrary type and uses the abstract API of metric maps to merge the cloud as if it was a single observation (e.g., from a 2D or 3D LiDAR) into the target. A primary use of this filter is local map updating in our LO system, but it is also key for map building in post-processing pipelines. Note that this operation generalizes several particular algorithms depending on the class of the target map: point cloud insertion into a voxel-based point cloud, ray-tracing on a 2D gridmap, raytracing and occupancy update for 3D volumetric grids, etc.

3.3.6. Dynamic object removal. This filter takes an input layer with an occupancy voxel map and uses its occupancy information to remove points from another point-cloud layer belonging to voxels with a reduced occupancy likelihood, meaning that those locations were sensed probably due to dynamic objects moving during the mapping process. Note that for a voxel to have a reduced occupancy probability it needs to have been observed more often free than occupied.

3.4. LiDAR odometry module

At this point, most basic blocks have been defined so we can introduce the architecture of the proposed LO module. Please use Figure 4 as a guide to the following discussion. In the figure, $\mathbf{q}[k]$ stands for the kinematic state of the vehicle at the discrete time step k, and it includes the vehicle body ("b") pose $\mathbf{T}_b \in SE(3)$ with respect to the local map and its linear $\boldsymbol{v}_b$ and angular $\boldsymbol{\omega}_b$ velocities. Also, keep in mind that such architecture remains the same for 2D or 3D mapping, for local maps as 2D gridmaps or as 3D point-clouds, etc. since those details are provided inside the configurable "pipeline" modules described later on.

Starting by the left of Figure 4, raw sensor observations arrive from either an offline or live source (block #1 in the diagram; see Section 3.13) and fed three subsystems: (i) the kinematic state estimation subsystem (block #2), mainly in charge of fusing sensors (wheel odometry, IMU, etc.) and past LO outputs ($\mathbf{q}[k-1]$) to make *short-term predictions* about the vehicle pose and velocity ($\mathbf{q}'[k]$); (ii) a first metric mapping pipeline (block #3; see *Section* 3.3), typically in charge of building filtered and down-sampled versions of the incoming LiDAR scans into the *observation* metric map data structure (see Section 3.2); and (iii) the view-based map (block #16), one of the main LO outputs.

As discussed earlier, this work follows a scan-to-map design pattern since it has proven benefits for 3D LiDAR mapping. Hence, after using raw observations to build a metric map (block #7), it is aligned against a local metric map (block #9), which is initialized via a special instance of a metric map pipeline (block #5) that normally will just create empty metric map layers. This block is run only once

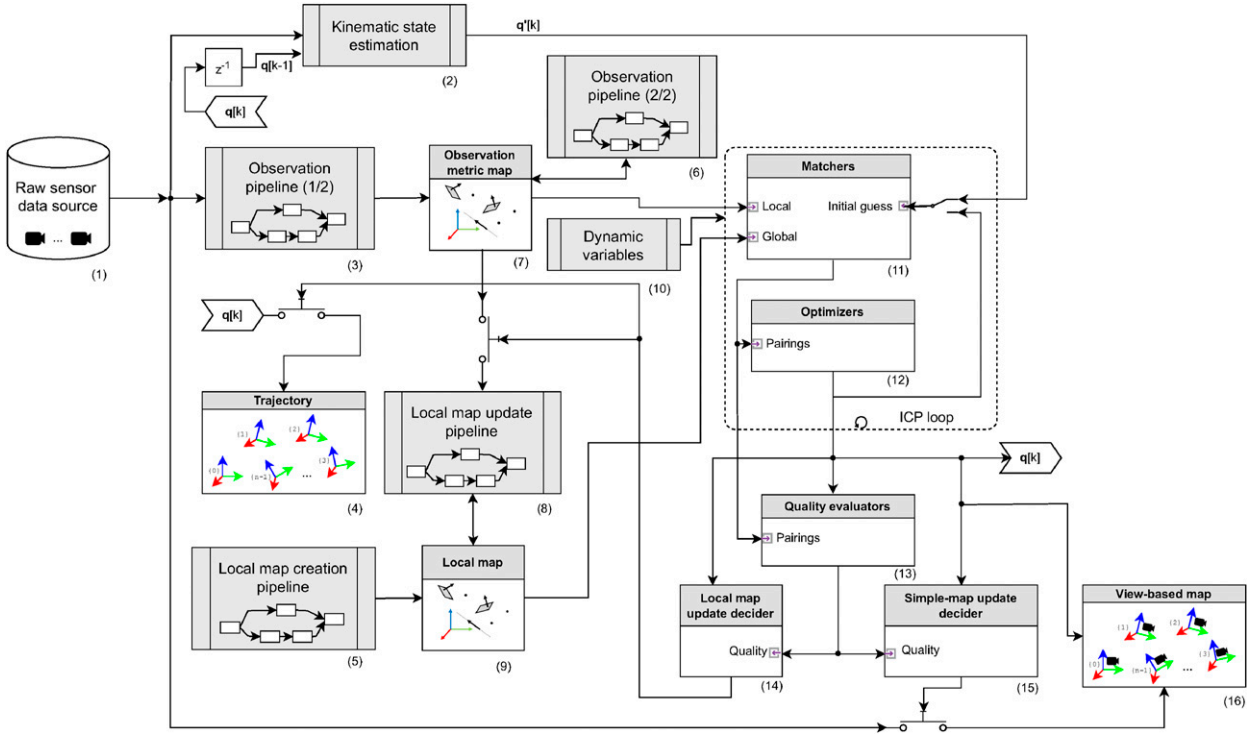


Figure 4. Architecture of the proposed generic optimization-based LiDAR odometry system. Each process is highly configurable and parameterizable. See discussion in Section 3.4.

at system start up or when it is reset. The local metric map data structure (block #9), is selectively updated using a custom pipeline algorithm (block #8) using one or more layers from the observation metric map (block #7) when certain criteria are met, as determined by a so-called local map update decider (block #14). An independent decider (block #15), normally with different criteria, is used to determine what frames become key-frames in the output view-based map. In general, such maps would benefit from sparser key-frames to reduce the size and memory requirements of final maps, while the local map tends to benefit from more frequent updates from incoming sensor frames. However, note that most past works (e.g., Deschaud, 2018; Vizzo et al., 2023) update the local map for every single incoming sensor frame. This may be optimal in driving scenarios, but our experimental results show that this policy may make the system in other scenarios, as analyzed in Section 7.2.

Pose tracking itself is achieved by means of an ICP-like iterative optimizer (blocks #11 and #12), in charge of finding the SE(3) pose $\mathbf{q}[k]$ that best explains the observation given a local map. This optimization procedure is detailed in Section 3.6. For now, it is enough to abstract it as an algorithm taking as inputs: (i) the initial guess $\mathbf{q}'[k]$ from the kinematic state estimator, (ii) a local and a global metric map (with different geometric entities and map layers depending on the application), and (iii) a set of dynamic variables (block #10) to adapt the behavior of the internal ICP pipeline (see Section 3.5).

Once the optimizer converges, a set of quality evaluation algorithms (block #13; see Section 3.6.4) are used to estimate the quality of the optimization. This metric, together with other conditions (not shown in the diagram for the sake of clarity) are then used to conditionally trigger the two decider blocks: (i) one to accept the alignment as good enough to append the obtained pose to the estimated trajectory (block #4) and to update the local metric map (block #9) via the purposely designed pipeline (block #8), and (ii) another one (block #15) to add new key-frames to the output map (block #16). Note that a localization-only mode exist (see Section 6), which disables both deciders while still producing an estimate of the vehicle state for each time step in $\mathbf{q}[k]$ by aligning the current observation against a prebuilt local map.

When the system is initialized, all blocks in Figure 4 are populated with objects created dynamically using class factories as described in a human-readable configuration file (a YAML file, Evans et al. (2017)), turning our framework into a flexible system easily re-configurable and extensible by end users, enabling systematic investigations on the effects of changes in individual sub-systems.

Subsequent sections give further details on the ICP loop and the particular configuration used for 3D LiDAR mapping.

3.5. Adaptive behavior: Dynamic variables

All software blocks in the LO system (Figure 4) have parameters that are initialized at system start-up. Some such parameters, however, would benefit from being dynamically adapted depending on sensed conditions like the size of the environment (indoors vs outdoors) or the motion profile (driving, handheld, drone, etc.). For example, take the voxel-based down-sampling mechanism used in IMLS-SLAM (Deschaud, 2018), which used a fixed resolution of 1.0 m, adequate for driving datasets. If the SLAM system is initialized in a small office-like environment instead, a much smaller resolution (i.e., less down-sampling) would be needed indeed. Instead of relying on manual tuning of all those parameters, our system would automatically detect the approximate size of the environment and initialize the local map and down-sampling resolutions accordingly.

To make this adaptation possible, our framework offers two features: (i) supporting mathematical expressions and basic scripting directly in the parameter specification files, and (ii) defining dynamic variables that can be used in such expressions. On the former, we use the C++ Mathematical Expression Toolkit Library (ExprTk) (Partow, 2015) due to its lightweight and efficient implementation. On the latter, a set of variables are defined so they can be used as symbols in the expressions, which are re-evaluated for each time step, and even for each ICP iteration within a given time step. Next, we review the most relevant such variables and how they are updated.

3.5.1. Maximum sensor range. This is a fundamental variable as it roughly indicates the size of the environment and can be used as a hint to the required down-sampling resolution. We define two such variables, the immediate maximum sensor range $r_{max}[k]$ for time-step k, and the low-pass filtered version $\widehat{r}_{max}[k]$ using a first-order IIR (Infinite Impulse Response) discrete-time filter with z-transform transfer function:

$$H(z) = \frac{1-\alpha}{z-\alpha} \quad \text{, with:} \quad \alpha = e^{-T\omega_c} \tag{3}$$

where α is a fixed parameter (typically in the range [0.9,0.99]) that defines the low-pass cut frequency ω_c for a sensor rate $1/T$.

3.5.2. Related to robot kinematic state. The latest vehicle state $\mathbf{q}[k]$, including pose and linear and angular velocities in the map frame, are available as dynamic variables. For example, the local map update decider (block #14 in Figure 4) checks for a minimum linear and angular distances between map updates. Our reference pipeline for 3D LiDAR mapping uses a heuristic formula for these distances that takes into account both, the maximum sensor range, and the instantaneous robot angular velocity, reducing the number of map updates while the sensor is experimenting higher accelerations. The insight behind this policy is that, despite scan deskewing, sensed points may be less accurate under these conditions since the assumption of constant velocity during each scan sweep is probably less accurate for higher angular velocities. Variables exposing the instantaneous robot pose ($\mathbf{q}[k]$) are also fundamental in the context of map post-processing, for example, for distance-based filtering while removing interferences from the vehicle itself or the person carrying a hand-held sensor.

3.5.3. Adaptive matching threshold. We have incorporated the idea of a dynamic threshold, proposed in KISS-ICP, to parameterize the data association and optimization stages of the ICP optimization loop according to perceived changes. As justified geometrically in Vizzo et al. (2023), given translational and rotational corrections $\mathbf{\Delta p} \in \mathbb{R}^3$ and $\mathbf{\Delta R} \in SO(3)$, respectively, of the ICP optimized pose with respect to the initial guess from the kinematic state prediction (Section 3.8), the expected maximum point-to-point error δ can be approximated with:

$$\delta = \|\mathbf{\Delta p}\| + 2r_{max} \sin \frac{1}{2} \left\| (\log \mathbf{\Delta R})^{\vee} \right\| \tag{4}$$

where log $\mathbf{R}$ is the matrix logarithm, the averaged maximum sensor range r_{max} is described in Section 3.5.1, and $(\cdot)^{\vee}$ is the *vee*-operator, the inverse of the wedge operator used in equation (2); that is, the 3 × 3 rotation correction matrix $\mathbf{\Delta R}$ is first mapped to the tangent space via the matrix logarithm giving a 3 × 3 skew-symmetric matrix with only three degrees of freedom, then the *vee*-operator simply takes out those three values as a 3 × 1 vector.

With the idea of obtaining a threshold σ to scale data association and optimization within ICP out of sequences of δ values over time, KISS-ICP proposes taking the standard deviation of all δ values, saturating with a minimum σ_{min} to prevent degeneration after long runs with good motion model predictions. While quite successful for driving scenarios, this approach lacks the ability to quickly react against abrupt changes in conditions (e.g., large occlusions, crossing a door moving from outdoors to an indoor area or handheld sensors with high dynamics). In order to further improve the adaptability of the whole system, we propose using a closed-loop proportional negative feedback controller to cope with sudden perturbations in the quality of ICP alignments, as sketched in Figure 5. This controller generates a multiplicative term c that makes δ to grow ($\sigma' = c\delta$) whenever ICP quality starts to drop, promoting the search for additional pairings in the next time step by increasing the search radius for nearest neighbors. A first-order low pass filter, identical to equation (3), is then applied to obtain the actual threshold σ. As it is well known in control theory, a proportional controller always exhibits an offset error, hence the obtained ICP quality will never reach the set-point, but in practice most often remains in the range [0.9, 1].

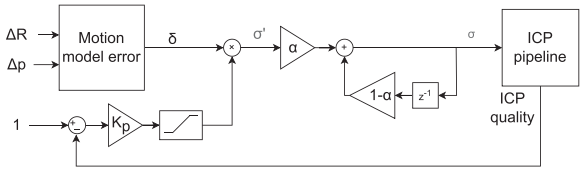


Figure 5. The closed-loop proportional feedback controller strategy to dynamically adapt the ICP parameters, scaled by a threshold σ, based on both, errors in motion model predictions, and the overall ICP quality (in the range [0,1]). The calculation of the motion model error δ is given in equation (4). Note that the input set-point for the controller is 1, the ideal quality.

3.6. ICP optimization loop

One of the central blocks of the LO pipeline is the flexible ICP algorithm, comprising blocks #11 and #12 in Figure 4. As the rest of the LO system, those blocks are dynamically created from a plain text configuration file describing the algorithms to use for data association, optimization, what metric map layers to use, and so on.

Our implementation is "ICP-like" in the sense that the main components of a classical ICP loop remain (i.e., data association, correction of the estimated transformation, and repeat until convergence) but many key differences exist:

- Nearest neighbor search for each input point is generalized into a set of matching algorithms (see Section 3.6.1).
- Several options are available as optimal pose solver algorithms from a set of predefined correspondences (see Section 3.6.2).
- An optional metric map pipeline algorithm can be applied inside the ICP loop itself (see Section 3.6.3).
- Dynamic variables (discussed in Section 3.5), including the ICP iteration count, can be used to parameterize any of the components above. This allows, for example, modifying the search radius over ICP iterations within one single time step (useful for loop closure).
- An optional probabilistic prior distribution of the sought pose is accepted as input, allowing prior knowledge to constraint the otherwise unbounded distance between the initial guess and the final result.
- Generation of log records to debug and visually inspect the internals of the algorithm during the different ICP iterations has been a central feature kept in mind during our implementation. As far as we know, no other public ICP implementation allows inspecting the evolution of the pairings and all other variables over ICP iterations in a systematic way.
- Importance is given to quality evaluators with a placeholder for such algorithms (see Section 3.6.4), fundamental in assessing whether a map registration is successful or not.

Keep in mind that the same ICP structure is to be used during online mapping (LiDAR odometry), localization, and for loop closure (see Section 3.12), although with different parameters and metric map layers.

A step-by-step description of the proposed ICP method is sketched in Algorithm 1 and is explained next, where many low-level details have been omitted for clarity. Overall, the "I" for *iterative* from the original ICP method remains as a fundamental feature required to solve such a strongly nonlinear problem as registering two metric maps with unknown pairings. The loop is repeated until one of two

conditions is reached: a maximum number of iterations N_{max} (line 9 in the pseudocode) or convergence (line 23). Successful registrations most often end up with convergence, hence N_{max} is set to a large enough number ($N_{max} \geq 300$) to avoid undesired search interruptions. Convergence is defined as small enough changes of the translational and rotational parts of the estimated transformation $\mathbf{T} = \{\mathbf{p}, \mathbf{R}\}$ with respect to the previous step $\mathbf{T}' = \{\mathbf{p}', \mathbf{R}'\}$, that is, whether $\delta p < \varepsilon_p$ and $\delta R < \varepsilon_R$ with:

$$\delta p = \|\mathbf{p} - \mathbf{p}\| \tag{5a}$$

$$\delta R = \left\|\log\left(\mathbf{R}^{-1}\mathbf{R}'\right)^{\vee}\right\| \tag{5b}$$

and ε_p and ε_R the corresponding thresholds, which are set to 10^{-4} and $5 \cdot 10^{-5}$, respectively, in all presented experiments. For each iteration, all matching algorithms are sequentially invoked over the two input metric maps to populate a list of potential pairings p (lines 11–14), then solvers are invoked using this list (lines 15–20) until a successful result $\mathbf{T}'$ is obtained (see Section 3.6.2). The optional input hook for invoking a metric map updating pipeline is invoked next (line 21), which enables our claimed new feature of fluidly deskewing the input point cloud as the estimated pose is updated within the ICP iterations (see Section 3.6.3). The loop is run until convergence, then quality evaluation algorithms are applied (lines 29–35) and their result is returned together with the estimated pose.

Algorithm 1 Generic ICP-like optimization algorithm

```
1: function ALIGN(m_l, m_g, T_0, T_p = {T̄_p, Σ_p}, {M_i}, {S_i}, {Q_i}, p)
2:     ▷ m_l and m_g are the local and global metric maps
3:     ▷ T_0 is the initial guess for the SE(3) transformation from local to global
4:     ▷ T_p = {T̄_p, Σ_p} is an optional prior Gaussian for sought transformation
5:     ▷ {M_i}, {S_i}, {Q_i}: sets of matchers, solvers, and quality evaluators.
6:     ▷ p is an optional metric map pipeline
7:     T ← T_0                                          ▷ Initialize optimal SE(3) transform
8:     k ← 0                                            ▷ Initialize iteration counter
9:     while k < N_max do
10:        UPDATEDYNAMICVARIABLES()                     ▷ For matchers and solvers
11:        p ← {}                                       ▷ Initialize pairings with empty set
12:        for each M ∈ {M_i} do                        ▷ For each matcher
13:            p ← p ∪ M.MATCH(m_l, m_g, T)             ▷ Data association
14:        end for
15:        for each S ∈ {S_i} do                        ▷ For each solver
16:            {T', V} ← S.SOLVE(p, T_p)                ▷ Optimal SE(3) solvers
17:            if V then                                ▷ Valid solution?
18:                break                                ▷ No need to try more solvers
19:            end if
20:        end for
21:        p.APPLY(m_l)                                 ▷ Apply observation pipeline (2/2)
22:        C ← CHECKCONVERGENCE(T, T')                  ▷ See Eq. (5)
23:        T ← T'                                       ▷ Update estimation
24:        if C then                                    ▷ Convergence?
25:            break
26:        end if
27:        k ← k + 1                                    ▷ Increment iteration counter
28:    end while
29:    q ← 0  , W ← 0  , bad ← False                    ▷ Initialize quality metric
30:    for each {Q, w_i} ∈ {Q_i} do                     ▷ For each quality algorithm
31:        q_i, bad_i ← Q.EVALUATE(T, m_l, m_g)         ▷ Evaluate metrics
32:        q ← q + w_i q_i                              ▷ Accumulate
33:        bad ← bad or bad_i                           ▷ Logic or
34:    end for
35:    q ← bad ? 0 : q/W                                ▷ Average
36:    return {T, q}                                    ▷ Return estimated transformation and quality
37: end function
```

3.6.1. Matchers. Different kinds of geometric pairs can conceivable be identified between two metric maps: point-to-point, point-to-plane, point-to-line, plane-to-plane, etc. Furthermore, the input local and global maps to be registered can be of different types: point clouds, grid maps, voxel maps, etc. Therefore, our framework provides a set of different match search algorithms depending on the kind of geometric features to find, while the map variety is handled via a generic nearest neighbor (NN) abstract interface. It makes sense to have more than one matching algorithm in multiple situations: (i) looking for successive pairings for each local map point in a prioritized list (e.g., first, try to find a point-to-plane pairing, but if it is not possible, then a point-to-point match), (ii) when several metric map layers exist and each local map layer is to be matched against a particular layer in the global map, and (iii) when different such layers represent differentiated geometric features (e.g., planes, edges, or pole-like objects).

Regarding efficient parallelized implementation of matchers, we follow the findings reported in KISS-ICP (Vizzo et al., 2023) and SIMPLE (Bhandari et al., 2024) about the superiority of *Intel Threading Building Blocks* (TBB) (Reinders, 2007) over other alternatives to exploit multi-core CPUs, hence our implementation also employs that library to parallelize inner loops of most algorithms handling point clouds. A more detailed description of all available matchers and benchmarking them against different metric map data structures is out of the scope of this paper and is left for future works.

3.6.2. Solvers. Having multiple solvers is in order due to the diversity of geometric entity combinations that can be found by matchers. In particular, our implementation offers the classic Horn's method in Horn (1987) (apt for point-to-point pairings) and a generic Gauss–Newton solver suitable for different types of geometries. In the following we are focusing on point-to-point pairings since they have been shown to lead to good results (Vizzo et al., 2023). Point-to-plane correspondences are also used for our 3D-NDT system configuration, but their equations are left out for space reasons and will be described somewhere else. More diverse geometric primitives are worth further investigation, for example, see MULLS (Pan et al., 2021).

Therefore, assume p in Algorithm 1 contains a set of N local $\{\mathbf{l}_i\}_{i=1}^N$ and global $\{\mathbf{g}_i\}_{i=1}^N$ points in $\mathbb{R}^3$ that are supposed to correspond to each other. Solving for the optimal pose $\mathbf{T}^\star \in SE(3)$ that minimizes the overall L2 registration error:

$$\mathbf{T}^\star = \arg\min_{\mathbf{T}} \sum_{i=1}^{N} \|\underbrace{\mathbf{T}\,\mathbf{l}_i - \mathbf{g}_i}_{\mathbf{e}_i(\mathbf{T})}\|^2 = \arg\min_{\mathbf{T}} \sum_{i=1}^{N} \|\mathbf{e}_i(\mathbf{T})\|^2 \tag{6}$$

is a classic problem with well-known closed-form solutions such as the quaternion method in Horn (1987). Unfortunately, in practice all pairings are not to be trusted equally due to the uneven and discrete nature of LiDAR sampling, sensor noise, and the existence of dynamic objects. Although our system offers Horn's algorithm among the possible solvers, it is far from being the optimal choice (see Section 7.4). Recent research has proven that robust (non least-squares) problem formulations lead to superior performance in the presence of pairing outliers and local minima, either using truncated least-squares and semi-definite programming (SDP) relaxation (Briales and Gonzalez-Jimenez, 2017; Yang and Carlone, 2019, 2020) or by simply applying a robust kernel $\rho(\cdot)$ (e.g., Huber, Geman-McClure,…) to equation (6) (Chebrolu et al., 2021):

$$\mathbf{T}^{\star} = \arg\min_{\mathbf{T}} \sum_{i=1}^{N} \rho(\|\mathbf{e}_i(\mathbf{T})\|) \tag{7}$$

In this work, we follow this latter approach for its simplicity and superior efficiency in terms of running time. Furthermore, we propose adding an additional term to the cost function integrating an optional probability distribution reflecting any prior knowledge about the relative map poses. In the context of LO, this term incorporates vehicle pose predictions from the kinematic state estimator (block #2 in Figure 4, see Section 3.8), while in the context of loop closure, it reflects uncertainty built up along a given topological loop (see Section 3.12). In any case, such prior information is modeled as a Gaussian distribution with mean $\overline{\mathbf{T}}_p \in SE(3)$ and covariance matrix $\mathbf{\Sigma}_p$. This prior term, denoted as $\mathbf{e}_p$, is not affected by the robust kernel and is introduced as a Mahalanobis distance of the *vee*-operator applied to the Lie group logarithm map of the pose mismatch, that is:

$$\mathbf{e}_p(\mathbf{T}) = \left(\log \overline{\mathbf{T}}_p^{-1}\mathbf{T}\right)^{\vee} \tag{8}$$

leading to the final robustified least-squares cost function:

$$\mathbf{T}^{\star} = \arg\min_{\mathbf{T}} c(\mathbf{T}) \tag{9}$$

$$c(\mathbf{T}) = \left\|\mathbf{e}_p(\mathbf{T})\right\|^2_{\mathbf{\Sigma}_p} + \sum_{i=1}^{N} \rho(\|\mathbf{e}_i(\mathbf{T})\|) \tag{10}$$

with $\mathbf{e}_i(\mathbf{T})$ the point-to-point pair costs defined in equation (6).

Solving equations (9) and (10) is a nonlinear optimization problem, hence we propose using the iterative Gauss–Newton (GN) algorithm. Since the search space is non-Euclidean, the Lie group perturbation-model approach is employed (Blanco-Claraco, 2021; Sola et al., 2018) where the unknown becomes an increment $\boldsymbol{\xi} \in \mathfrak{se}(3)$ (SE(3) Lie group tangent Euclidean space):

$$\boldsymbol{\xi}^{\star} = \arg\min_{\boldsymbol{\xi}} c(\mathbf{T} \boxplus \boldsymbol{\xi}) \tag{11}$$

$$\mathbf{T}^{\star} = \mathbf{T} \boxplus \boldsymbol{\xi}^{\star} \tag{12}$$

which using the right-hand perturbation convention:

$$\mathbf{T} \boxplus \boldsymbol{\xi} = \mathbf{T}\ e^{\boldsymbol{\xi}} \tag{13}$$

allows us finding the required Jacobians to solve the normal equation of each GN iteration:

$$\mathbf{H}\boldsymbol{\xi}^{\star} = -\mathbf{g} \tag{14}$$

with $\mathbf{H}$ and $\mathbf{g}$ the 6×6 Hessian matrix and 6×1 gradient vector, respectively:

$$\mathbf{H} = \frac{\partial \mathbf{e}_p}{\partial \boldsymbol{\xi}}^{\top} \mathbf{\Sigma}_p^{-1} \frac{\partial \mathbf{e}_p}{\partial \boldsymbol{\xi}} + \sum_{i=1}^{N} \sqrt{w(\|\mathbf{e}_i\|^2)} \frac{\partial \mathbf{e}_i}{\partial \boldsymbol{\xi}}^{\top} \frac{\partial \mathbf{e}_i}{\partial \boldsymbol{\xi}} \tag{15}$$

$$\mathbf{g} = \frac{\partial \mathbf{e}_p}{\partial \boldsymbol{\xi}}^{\top} \mathbf{\Sigma}_p^{-1} \mathbf{e}_p + \sum_{i=1}^{N} \sqrt{w(\|\mathbf{e}_i\|^2)} \frac{\partial \mathbf{e}_i}{\partial \boldsymbol{\xi}}^{\top} \mathbf{e}_i \tag{16}$$

with $\partial \mathbf{e}_p / \partial \boldsymbol{\xi}$ given by Blanco-Claraco (2021), equation (10.19), $w(\cdot)$ the *weight* function of the employed robust kernel, and:

$$\frac{\partial \mathbf{e}_i}{\partial \boldsymbol{\xi}} = \left(\begin{bmatrix} \mathbf{e}_i^{\top} & 1 \end{bmatrix} \otimes \mathbf{I}_3\right) \frac{\partial \mathbf{T} e^{\boldsymbol{\xi}}}{\partial \boldsymbol{\xi}} \tag{17}$$

where the expression for $\partial \mathbf{T} e^{\boldsymbol{\xi}} / \partial \boldsymbol{\xi}$ can be found in Blanco-Claraco (2021), equation (10.19) and $\otimes$ is the Kronecker product. Our implementation actually allows matchers to include individual weights for each pairing, which scale the corresponding Jacobians in the two equations above, but in the present work we just assigned all pairings equal weights. Past works, such as Dellenbach et al. (2022), have exploited such weighting to give higher priority to more reliable points, hence this feature deserves further research. The cost function and associated Jacobians for other geometric entities in our implementation (point-to-plane, point-to-line, etc.) are left out of this work for space reasons and shall be described in future works.

A configurable number of GN iterations can be run, but note that GN is invoked inside an outer ICP loop, hence running just one or two iterations is normally enough: new data association for the next ICP iteration means the cost function is likely to change so it is not worth wasting time iterating to find what will only become partial solutions.

3.6.3. Local map update pipeline. As shown in line 21 of Algorithm 1, an optional metric map pipeline (Section 3.3) hook can be invoked *within* each ICP loop iteration. This feature is not used for loop-closure, but it is key for sequential processing of scans in odometry. In the default configuration for 3D LiDAR odometry (Section 3.7) this hook is used for two operations: (i) refining the estimated vehicle linear and angular velocities, and (ii) using such new estimated velocities to apply the deskewing filter (Section 3.3.1) to the final layers of the

metric map built from the current LiDAR scan (refer to "Observation pipeline (2/2)" in Figure 7).

Linear ${}_s\boldsymbol{v}_s$ and angular velocity ${}_s\boldsymbol{\omega}_s$ vectors of the sensor, both in the sensor frame "s," are estimated under the assumption of constant velocity from the current sensor pose[2] estimation $\mathbf{T}_{sk}$ in the ith ICP iteration and the sensor pose from last time step $(\mathbf{T}_s)_{k-1}$ with a sensor period T:

$$((\mathbf{T}_s)_{k-1})^{-1}\mathbf{T}_{sk} = \begin{bmatrix} & \Delta\mathbf{R} & & \Delta\mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix}_{4\times4} \tag{18}$$

$$\boldsymbol{v}_s = \frac{1}{T}\Delta\mathbf{p} \qquad {}_s\boldsymbol{v}_s = \mathbf{T}_{s_k}^{-1}\boldsymbol{v}_s \tag{19}$$

$$\boldsymbol{\omega}_s = \frac{1}{T}(\log\Delta\mathbf{R})^{\vee} \qquad {}_s\boldsymbol{\omega}_s = \mathbf{T}_{s_k}{}^{-1}\boldsymbol{\omega}_s \tag{20}$$

In practice, this simple method makes the velocity vectors to be estimated alongside with the vehicle incremental pose, enabling fluidly de-skewing points during ICP iterations so they better match the reference local map. As shown in the experimental results and in the ablation study in Section 7.3, this allows the effective tracking of abrupt motion profiles (see, e.g., Section 4.9).

Our approach has a similar aspiration than Continuous-Time ICP (Dellenbach et al., 2022), but there are two differences: (i) ours does not introduce pose discontinuities between the end of a scan and the beginning of the next one, and (ii) ours does not need to use heuristic weights for the different parts of the optimization (point registration errors, final pose, and velocity vectors) since velocities are implicitly defined from the iteratively refined final pose, and the latter is unconstrained.

3.6.4. Quality estimators. Except in degenerate cases, all solvers discussed above such as the classical quaternion method (Horn, 1987) or the GN iterative solver will always produce a result, correct or wrong depending on the existence of outliers in the input pairings. It hence becomes essential to assess whether the output relative pose between the two metric maps is plausible or should otherwise be discarded. In this latter case, our framework discards those key-frames with an unacceptable ICP quality. If this happens right at the beginning of mapping, the metric map is discarded and started from scratch again. Otherwise, amid navigation, the kinematic state estimator extrapolates the predicted vehicle motion to try to recover a good pose tracking in subsequent frames.

As seen in lines 29–35 of Algorithm 1, we propose using the weighted average of a configurable set of quality evaluation algorithms. The abstract interface of such algorithms takes as input the two metric maps that has been aligned and the tentative optimal relative pose found by solvers, and returns two elements: (i) a quality metric $q_i \in [0, 1]$, and (ii) a short-circuit logic discard flag, which shall be activated when the alignment is clearly wrong. In that latter case, the q_i values returned by other quality evaluators are ignored and the registration is discarded.

We have implemented three such algorithms:

- **Point-wise paired ratio:** This simple method just uses the ratio of local map points that were assigned a pairing during data association in the last ICP iteration:

$$q = \frac{N_{pairings}}{N_{local\_points}} \tag{21}$$

This method is normally adequate for LO, although not informative enough for loop closure assessment.

- **Range reprojection:** An implementation of the method proposed in Bogoslavskyi and Stachniss (2017), based on re-projecting 3D points into a range image for comparison between the two maps.
- **Voxel occupancy metric:** A novel algorithm that takes into account all volumetric occupancy information, and its mismatch between both metric maps. This method requires both maps to have an occupancy voxel layer, which is straightforward to add in our framework by adding such requirement to the metric map pipelines. The key insight is that voxels that are either free or occupied in both maps "vote" for a good quality, but those with contradictory information "vote" for a bad quality with a stronger weight. Inspired by Kullback-Leibler distance, we define a heuristic loss function $L(p_i, p_j)$ taking the occupancy probabilities p_i and p_j of each pair of voxels (in the range [0,1]) corresponding to the same global coordinates for the two maps, with the shape illustrated in Figure 6: similar occupancies have the highest score, any of the two map voxels being unobserved ($p \sim .5$) has a smaller score, and contradictory information has a strong negative score. All N voxels are integrated into a single scalar quality q value as:

$$q = \frac{1}{1 + e^{-\kappa d/N}} \quad \text{with:} \quad d = \sum_{k=0}^{N} L(p_i^k, p_j^k) \tag{22}$$

where κ is a heuristic scale parameter.

Apart of these quality metrics, note that the Gauss–Newton optimizer described in Section 3.6.2 also provides an uncertainty estimation for the solution, that is, the Hessian matrix of the last iteration. Although normally overconfident, its eigenvalues and eigenvectors can be also used to assess whether a given relative pose is well-conditioned.

3.7. Default configuration for 3D LiDARs

Once the generic architecture of the LO module and the ICP algorithm have been shown in Figure 4 and Algorithm 1, respectively, we introduce a particular system configuration

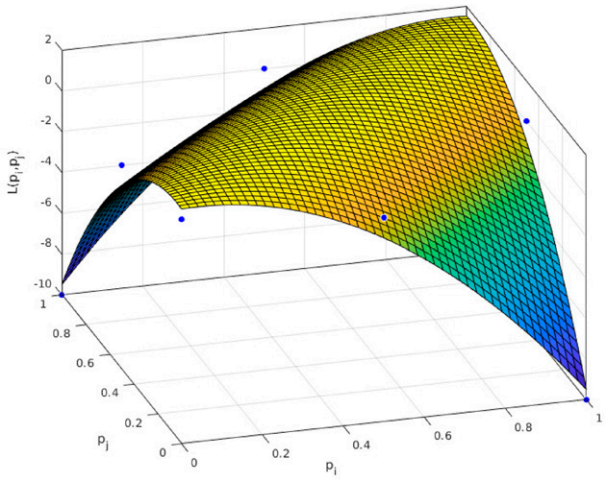


Figure 6. Heuristic loss function to compare voxel occupancies p_i, $p_j \in [0, 1]$ from two maps i and j in the proposed alignment quality metric in Section 3.6.4. The exact equation represented here is $L(p_i, p_j) = 1.5 + p_i + p_j - 12p_i^2 + 22p_ip_j - 12p_j^2$.

which is dubbed lidar3d-default in the implementation and which has been used in most experimental results. A *configuration* comprises the definition of what blocks populate the internals of generic blocks in Figure 4, along with their parameters.

This configuration has been designed to successfully run in as many situations as possible, hence it is not particularly optimized for minimum tracking error but for robustness and for its ability to run at sensor rates or faster. Configurations with better performance for particular sensors and environments could be devised, but this is future work.

Observing Figure 4, there are three ICP-related blocks to be defined (blocks #11, #12, #13) and four metric map pipeline blocks (blocks #3, #5, #6, #8). Next we describe how they are set-up in this configuration.

3.7.1. Local map creation pipeline. A single map layer is created using a hashed voxel map holding point clouds, with a maximum of 20 points per voxel, only (X,Y,Z) attributes per point, and a voxel resolution of 1.5% of the estimated sensor maximum range with a minimum and maximum of 0.5 m and 1.0 m, respectively. No minimum distance between points stored in each voxel is imposed.

3.7.2. Observation pipelines. Split in two parts, (1/2) and (2/2), with contents detailed in Figure 7. Recall that the pipeline is split to allow the first part to be invoked only once per incoming scan, and the latter once for each ICP iteration. As sketched in the figure, a default generator first takes the raw sensor data and produces a point cloud map layer ("raw") with contiguous memory layout (see Section 3.2). If per-point timestamps are present, they are then adjusted, with the particular convention of using the middle of the scan as the time origin. Next, the cloud is down-sampled and nearby points removed (`FilterByRange` in the figure) to avoid polluting the map with vehicle or robot body measurements. Next, a bounding box filter is applied to remove nearby ceiling points, which have been shown to lead to divergence in some scenarios. Then, this point cloud, which underwent decimation once, is decimated once again so we have two layers: a denser one to update the local map, and the sparser one to be used for registration in the ICP algorithm. This idea of using dual-resolution maps comes from its successful implementation in KISS-ICP (Vizzo et al., 2023). Finally, note that de-skewing happens at the end, once for each down-sampled clouds, whereas all former works perform this step at the beginning instead. The reason for this alternative placement is twofold: to enable the fluid de-skewing discussed in Section 3.6.3, and increasing the efficiency since far less points are de-skewed in comparison to performing this operation at the very beginning of the pipeline. The result of first down-sampling then de-skewing is not exactly the same than switching the order of both operations, but in practice both methods lead to equivalent overall trajectory quality metrics.

3.7.3. Local map update pipeline. This pipeline comprises just one map insertion action (Section 3.3.5), taking one of the observation map layer (the denser cloud) and merging it into the local metric map, effectively making the map to grow as the vehicle explores the environment.

3.7.4. ICP-related blocks. One matcher algorithm is defined to find out point-to-point correspondences between the local map and one of the observation layers (the sparser cloud). Then, the Gauss–Newton solver described in Section 3.6.2 is defined as the unique solver. Following KISS-ICP (Vizzo et al., 2023), the matcher threshold and the solver robust kernel are parameterized with a dynamic threshold, although estimated in a different way (see Section 3.5.3). Finally, quality for LO is assessed by using the matching ratio criterion (Section 3.6.4).

3.8. Kinematic state prediction

This role of this algorithm in the LO system (block #2 in Figure 4) is to generate short-term pose and velocity predictions for the vehicle for each incoming LiDAR scan, based on the history of past sensor observations and former ICP registration outcomes. Poses are needed to make the nonlinear iterative search within ICP to start as close to the solution as possible, and velocities are needed to perform the initial scan de-skewing.

We have experimented with two alternative methods: (i) simple kinematic extrapolation using only the two last vehicle poses, and (ii) a full probabilistic formulation based on factor graphs optimizing over a sliding window of past vehicle poses. Initial results showed that the simple method gave more accurate trajectories and was less prone to divergence under abrupt motions, hence we will only discuss the former method here and will leave the latter for future analysis.

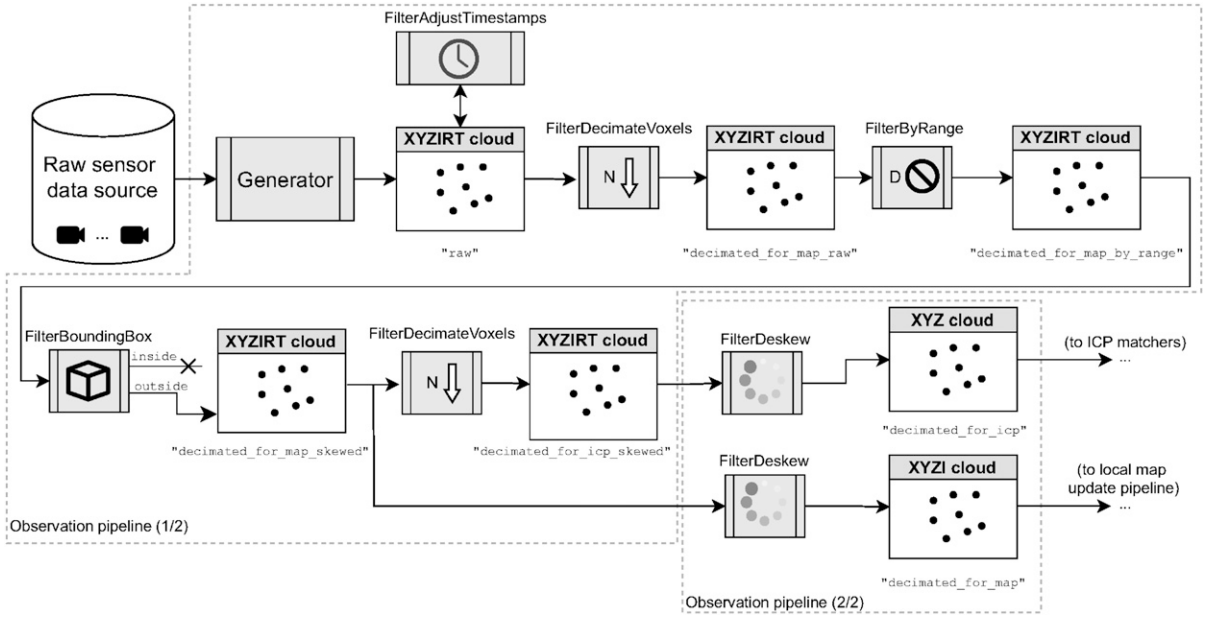


Figure 7. Detailed view of the "observation pipeline" blocks in Figure 4 for the "default 3D LiDAR" LO system configuration. Refer to discussion on pipelines in general in Section 3.3 and to the rationale behind this particular pipeline in Section 3.7.

The implemented method takes only two inputs: estimated poses from ICP optimization, and, optionally if available, wheels odometry readings. Given the two last poses $\mathbf{T}_{s_k}$ and $(\mathbf{T}_s)_{k-1}$ with elapsed time between them of T seconds, equations (18)–(20) give us the estimated linear ${}_s\boldsymbol{v}_s$ and angular ${}_s\boldsymbol{\omega}_s$ velocities, in the local sensor frame "s." Then, when a kinematic state prediction is needed for an instant δ_t ahead of frame $\mathbf{T}_{s_k}$, velocities are left unmodified (constant velocity model assumption) and the predicted pose $\widehat{\mathbf{T}}_{s_{k+1}}$ is evaluated as follows. In the absence of wheel odometry, the former pose is extrapolated assuming constant velocities, that is:

$$\widehat{\mathbf{T}}_{s_{k+1}} = \mathbf{T}_{s_k} \begin{bmatrix} (\delta_t \, {}_s\boldsymbol{\omega}_s)^\wedge & \delta_t \, {}_s\boldsymbol{v}_s \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{23}$$

while, if wheel odometry is present, its values over time are used as a good approximation of short-term incremental motion and such increment ${}_k\mathbf{T}_{k+1}$ is just applied to the last frame, that is, $\widehat{\mathbf{T}}_{s_{k+1}} = \mathbf{T}_{s_k} {}_k\mathbf{T}_{k+1}$.

Note that an output from this prediction method is not available until at least two LiDAR scans have been processed by the LO pipeline. In this initial time step, plus when there is a long period without incoming data (either intentional or due to temporary sensor failure), there will be no kinematic state prediction. Unless the LO system is started while already moving at high speed, this does not present a particular challenge to the ICP-like optimizer. However, our framework offers the option to define an alternative ICP pipeline for time steps without state prediction, for example, to use a much larger matching threshold. Experimental results presented in this paper did not need to exploit such possibility, though.

3.9. Implementation of view-based maps

This section provides some insights into how view-based maps (see Section 3.1) are implemented in our framework. As stated above, these maps are the output of LO, hence they must include all the required information to build actual metric maps from them in the most flexible way. Thus, each such map $\mathfrak{m}$ is defined as a sequence of N keyframes $\mathbf{k}_i$:

$$\mathfrak{m} = \{\mathbf{k}_i\}_{i=1}^{N} \tag{24}$$

$$\mathbf{k}_i = \left(\mathbf{T}_i, (\boldsymbol{v}_i, \boldsymbol{\omega}_i), \{o_{i,j}\}_{j=1}^{M_i}\right) \tag{25}$$

with key-frames being tuples of three elements: (i) the vehicle pose as a Gaussian distribution $\mathbf{T}_i = \{\overline{\mathbf{T}}_i, \boldsymbol{\Sigma}_i\}$, (ii) the estimated vehicle linear ($\boldsymbol{v}_i$) and angular ($\boldsymbol{\omega}_i$) velocities, and (iii) a set of M_i raw sensor observations $o_{i,j}$. Vehicle poses and velocities are given in global coordinates with an arbitrary origin, while observations include both, raw sensor data, and the sensor pose within the vehicle.

Within observations we include an additional metadata observation which, instead of coming from a real sensor, is filled during LO. It includes information such as the bounding box, in robocentric coordinates, of all sensed points for each keyframe. This enables efficient determination of unfeasible loop-closures without having to analyze the raw sensor data (see Section 3.12). Furthermore, given that these maps may make use of the lazy-load feature described in Section 3.13 to make them efficient to load and parse, this metadata enables loading from disk only those sensor frames that actually are needed for loop-closure hypothesis checking.

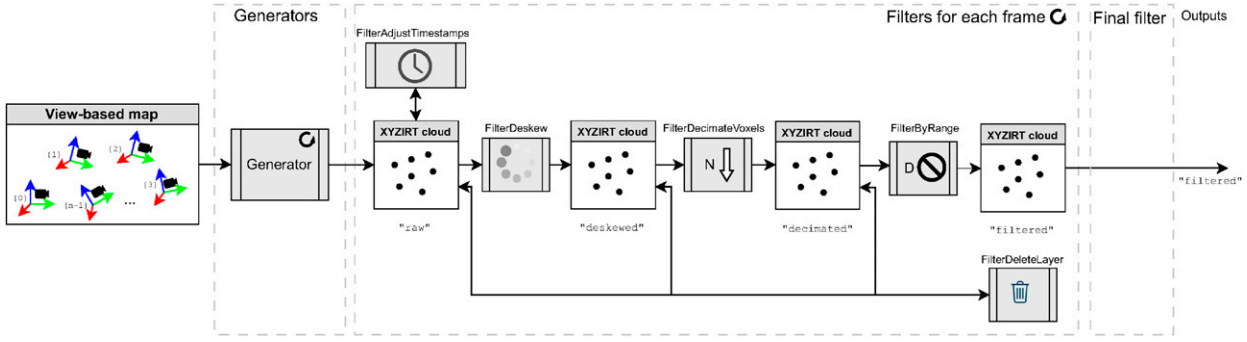


Figure 8. Example configuration for building metric maps from view-based maps (using the application sm2mm): a basic pipeline for building one single point cloud by accumulating decimated versions of de-skewed LiDAR scans. Refer to discussion in Section 3.10.

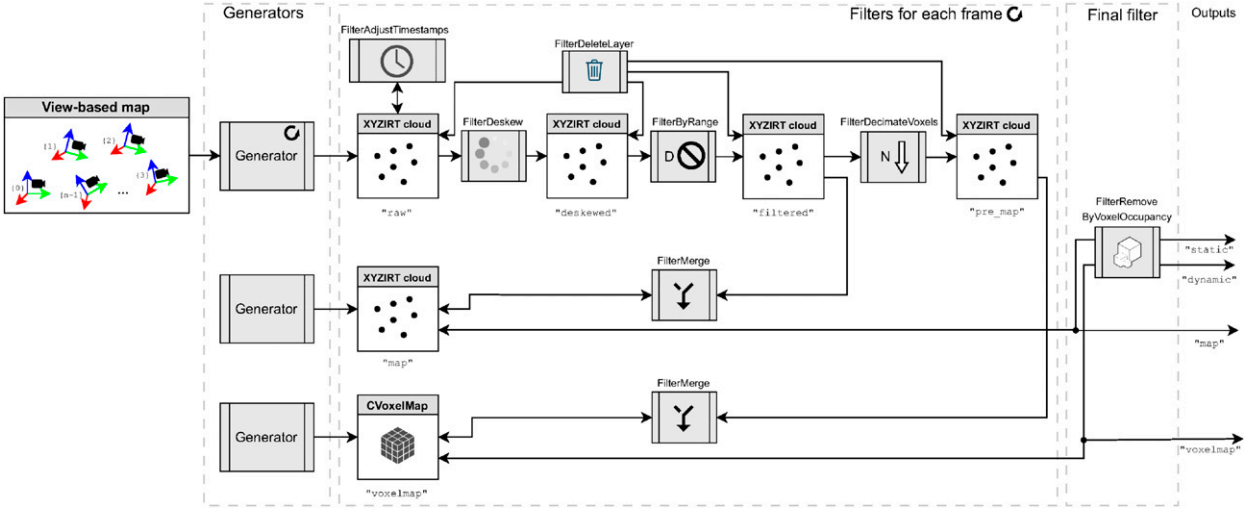


Figure 9. Example configuration for building metric maps from view-based maps (using the application sm2mm): a more advanced pipeline for classification of points into static or dynamic via volumetric probabilistic occupancy. See output examples in Figure 10 and discussion in Section 3.10.

3.10. Converting view-based map to metric maps

Once a view-based map is populated with key-frames, one can use it to build a particular kind of metric map from it; this corresponds to the task represented in Figure 2(c) while discussing the framework overview.

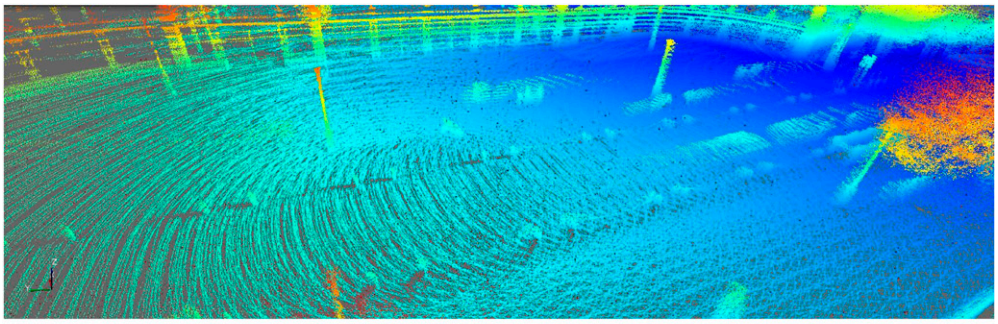
Pipelines comprise three stages:

- **Generators:** Can be evaluated once to create empty metric maps of a given type with a particular set of parameters, or evaluated for each incoming frame in the view-based map to populate a map layer from its raw sensory data.
- **Per-frame filters:** The main part, where all filters in a sequence are evaluated in a predefined order, once per incoming map frame.
- **Final filters:** This optional stage can be used to post-process the maps, further filtering them, etc.

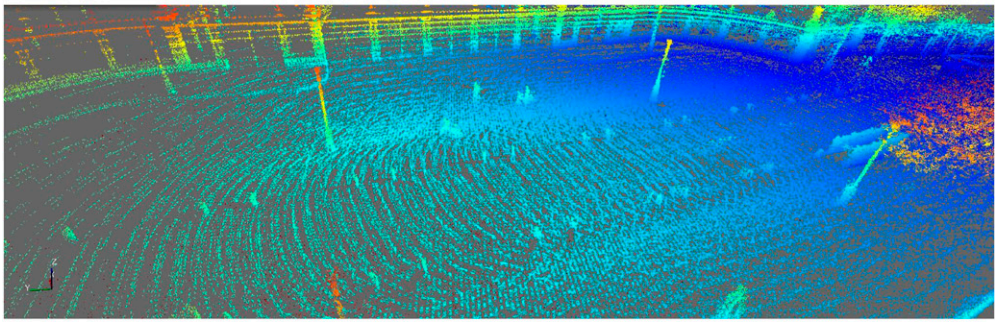
The utility of such flexible design is illustrated with two example metric map pipelines[3]. Possibly the simplest pipeline for 3D LiDAR datasets, sketched in Figure 8, uses a generator converting raw observations into a "raw" point cloud, which is then de-skewed after adjusting per-point timestamps to a given fixed convention. Recall de-skew in this context is possible since estimated velocities are stored within map key-frames (see Section 3.9). Next, scan points are down-sampled and spatially filtered to remove observations from the vehicle body. Finally, all layers are cleared so they start empty for the next map frame, except the map layer used as output, where all points accumulate.

A more elaborate pipeline is presented in Figure 9, with the goal of generating point clouds where mobile objects are removed, which is accomplished as follows. On the left, we have three generators: one to convert raw sensory data into point clouds (just as in the former pipeline), plus other two generators, each only invoked once to create two of the final metric maps. Note that each generator will create metric

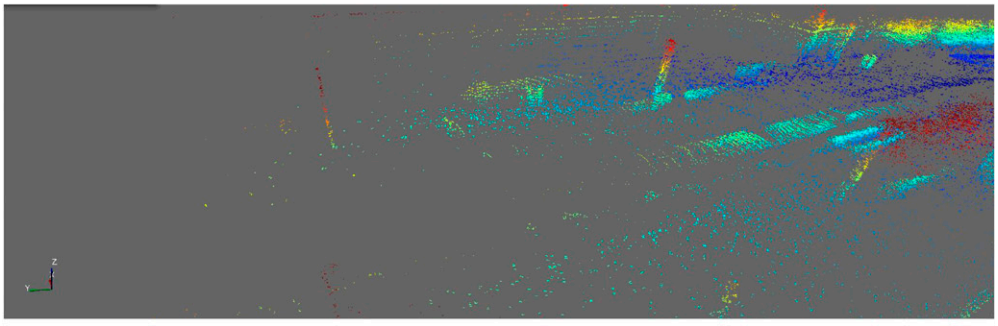
map layers of different types and with a particular sets of parameters (e.g., resolution and log-odds occupancy update weights). These two maps are a point cloud (layer "map") and a 3D occupancy voxel map (Section 3.2). Raw points are de-skewed and spatially filtered to remove self-body interferences. That version of the cloud is inserted, for each frame, into the final point cloud "map," whereas its down-sampled version (for efficiency reasons) is used to perform ray-tracing and update the occupancy value of the voxel map. Once all map frames are processed, the final filtering stage makes use of the filter described in Section 3.3.6 to split the map point cloud into those likely to be static and



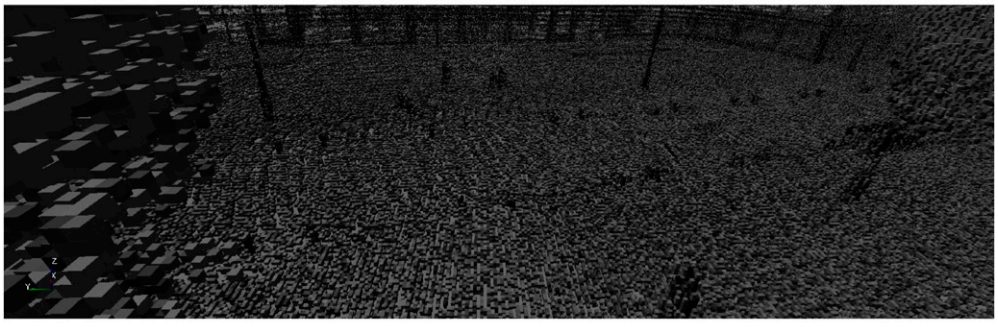
(a) Layer: "map"



(b) Layer: "static"



(c) Layer: "dynamic"



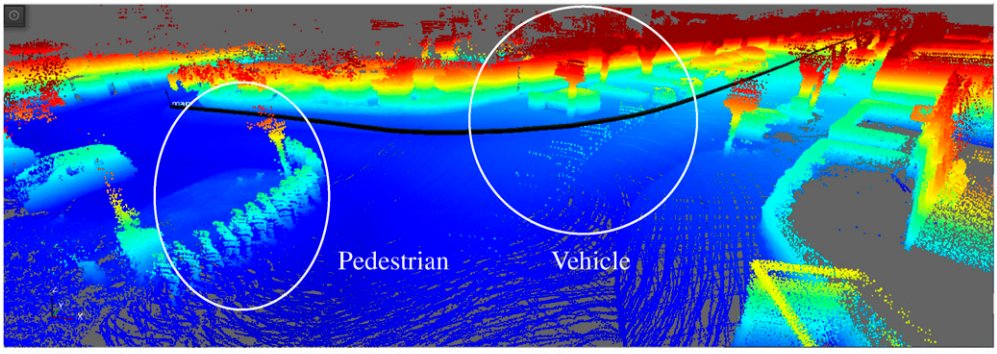
(d) Layer: "voxelmap"

Figure 10. Layers of the metric map built using the pipeline of Figure 9 from the Luxembourg Garden automotive dataset in Dellenbach et al. (2022). Note how most traces of moving pedestrians and vehicles in (a) have been correctly removed in (b). See discussion in Section 3.10.

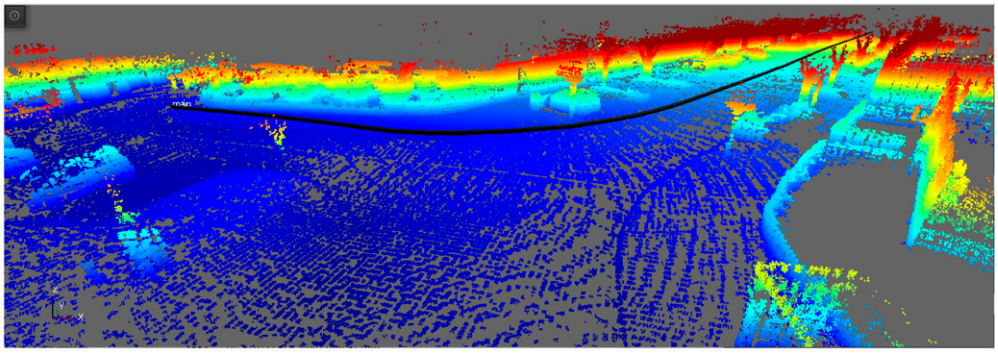
dynamic. As a first example of the attainable results, Figure 10 shows the output map layers for a small fragment of the Paris LuCo dataset (Dellenbach et al., 2022), where it can be seen how most 3D points corresponding to pedestrians, vehicles, or noisy points (e.g., near the edges of light poles), are removed from the static layer. Note that since there is no filter implementing learning-based semantic segmentation yet, vehicles and pedestrians that remain still are classified as static objects.

As an additional demonstration of this dynamic object removal pipeline we have tested it against one of the sequences of the ERASOR dataset introduced in Lim et al. (2021). This dataset is specifically designed to capture situations with dynamic objects such as pedestrians or vehicles in motion. The same parameters were used than in the Paris LuCo dataset above, and results are shown in Figure 11. It can be verified how all traces of vehicles and pedestrians are correctly removed in the "static" layer.

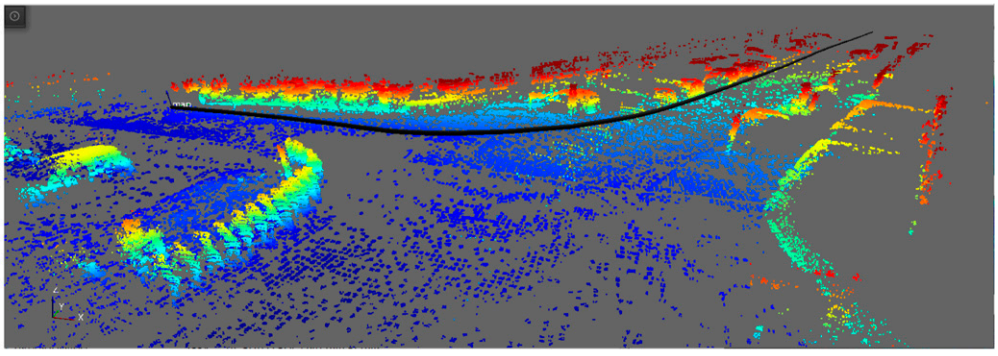
To sum up, the flexibility of our framework enables generators or filters to be combined in arbitrary configurations to easily manipulate and filter metric maps in ways that have not been possible before, and all this without coding.



(a) Layer: "map" (original map)



(b) Layer: "static"



(c) Layer: "dynamic"

Figure 11. Layers of the metric map built using the pipeline of Figure 9 from the sequence 00_4390_to_4530 in the ERASOR dataset (Lim et al., 2021). The estimated vehicle trajectory is shown as a thick black solid line. Note how traces of moving pedestrians and vehicles in (a) have been correctly removed in (b). The "dynamic" layer in (c) contains those moving objects, together with noisy points from environment voxels that were not consistently observed as occupied with a level of confidence high enough. See discussion in Section 3.10.

3.11. Georeferencing view-based maps

Georeferencing a map consists in establishing the correspondence between coordinates in the map frame and either geodetic coordinates or any other global coordinate system such as UTM (Universal Transverse Mercator).

In particular, we address the problem of georeferencing view-based maps $\mathfrak{m}$ (see equations (24) and (25)), or segments of such maps, that contain frames with observations $o_{i,j}$ from low-cost GNSS receivers with positioning errors in the range of meters. Apart of its utility on itself, georeferencing segments of large maps become a valuable tool for loop-closure detection outdoors (see Section 3.12).

Following equation (25), we assume without loss of generality that there are N map frames with global poses with respect to the map origin being $\mathbf{T}_i \equiv {}_{map}\mathbf{T}_i \in SE(3)$ and where all frames contain exactly one GNSS positioning observation o_i with geodetic coordinates, that is, latitude, longitude, and ellipsoid height. Let the first such observation, o_0, be used to determine the East-North-Up (ENU) frame of reference with respect to which all other observations have Euclidean coordinates ${}_{enu}\boldsymbol{t}_i \in \mathbb{R}^3$; see (Blanco et al., 2009, §4.1) for the geodetic transformation formulas.

Therefore, note that once we have run LO and have GNSS observations, two coordinate origins coexist: the LO metric map origin ("map") and that for ENU coordinates ("enu"). Georeferencing the map hence becomes finding out the rigid transformation ${}_{enu}\mathbf{T}_{map} \in SE(3)$ that best explains all observations, as sketched in Figure 12(a). The placement of the ENU frame may be initialized from the very first GNSS observation in the view-based map, or may be given already by another past observation in the context of submapping (see LC in Section 3.12). We propose using factor graphs (Murphy, 2012; Ni et al., 2007), a popular kind of bipartite graphical model, as the ideal tool to solve such estimation problem from noisy, possibly containing spurious, GNSS readings. As shown in Figure 12(b), the graph unknowns are the relative pose ${}_{enu}\mathbf{T}_{map}$ and the vehicle trajectory poses ${}_{enu}\mathbf{T}_i$, in the ENU frame. The relative pose between the latter ones will normally not change significantly, since they were estimated using LO with a small uncertainty, but their values are left as free variables such as, for long trajectories, global GNSS observations are able to correct the LO drift, up to the GNSS receiver accuracy. In particular, GNSS observations are specially beneficial to reduce the larger error that exists in the vertical axis due to LO drift in long trajectories. Two types of factors are defined. First, unary GNSS factors (f_G), whose error vector $\mathbf{e}_G({}_{enu}\mathbf{T}_i)$ is the mismatch between the measured GNSS ENU coordinates ${}_{enu}\boldsymbol{t}_i$ and the predicted position of the GNSS antenna:

$$\mathbf{e}_G = ({}_{enu}\mathbf{T}_i \oplus {}_{v}\boldsymbol{t}_g) - {}_{enu}\boldsymbol{t}_i \tag{26}$$

with ${}_{v}\boldsymbol{t}_g \in \mathbb{R}^3$ the GNSS antenna coordinates on the vehicle ("v"). These factors use: (i) robust kernels to cope with large errors in areas with satellite signal occlusions, and (ii) a zero-mean Gaussian noise model based on the GNSS self-reported accuracy. The other type of factor in Figure 12(b) is the relative pose binary factor (f_R) whose error function $\mathbf{e}_R({}_{enu}\mathbf{T}_{map}, {}_{enu}\mathbf{T}_i)$ is proportional to the pose mismatch between key-frame pose variables ${}_{enu}\mathbf{T}_i$ and their pose as stored in the simple-map:

$$\mathbf{e}_R = \log\left(\left({}_{map}\mathbf{T}_i\right)^{-1}\left({}_{enu}\mathbf{T}_{map}\right)^{-1}{}_{enu}\mathbf{T}_i\right)^{\vee} \tag{27}$$

which follows from pose compositions through the two possible paths between the frame "map" and the ith key-frame in Figure 12(a).

We use GTSAM (Dellaert, 2021) to implement and solve the optimization problem defined by this factor graph. The quality of the resulting georeferencing is assessed via the square roots of the diagonal of the marginal covariance matrix for ${}_{enu}\mathbf{T}_{map}$, which reflect whether all six degrees of freedom in SE(3) were properly constrained by observations or not, for example, a common caveat is the "roll" angle not be estimated accurately when processing a map segment with an almost perfect straight line trajectory. Adding multiple GNSS sensors with a large baseline could be interesting for large vehicles. Experimental results regarding georeferencing maps are shown in Section 5.2, while this feature is also a building block of LC, addressed next.

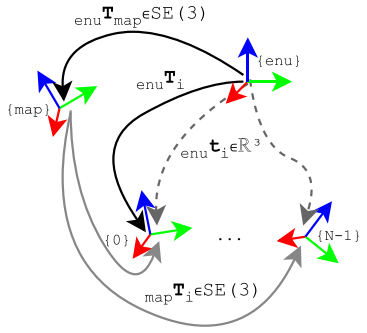
3.12. Loop closure algorithm

Having loop closing (LC) mechanisms is what makes the difference between an odometry and a SLAM system, capable of creating large, globally-consistent maps. LC in our framework is not run concurrently to odometry but as a post-processing stage. This could change in future versions, but the fundamentals would remain the same.

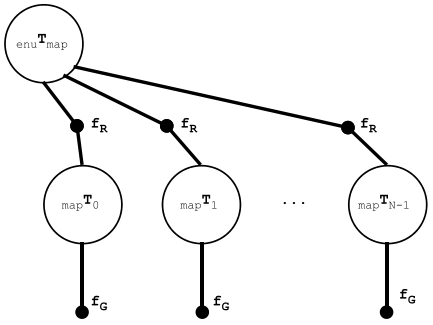
Algorithm 2 Overview of the loop closure algorithm

```
function LOOP_CLOSURE(m)
    ▷ m is the input/output simple-map
    S = {s_i}_{i=1}^M ← DIVIDESUBMAPS(m)                ▷ Split into sub-maps
    for each s_i ∈ S do                                  ▷ For each sub-map
        b_i ← BOUNDINGBOX(s_i)                           ▷ Evaluate bounding box
        g_i ← GEOREFERENCING(s_i)                        ▷ Geo-reference sub-map, if possible
    end for
    G_s ← BUILDSUBMAPGRAPH(S, {g_i})
    G_k ← BUILDKEYFRAMESGRAPH(S, {g_i})
    if any {g_i} then                                    ▷ Optimize poses from GNSS
        {G_k, G_s, m} ← OPTIMIZEGRAPH(G_k, G_s, m)
    end if
    while true do
        L ← LC_CANDIDATES(G_s, {b_i})
        if |L| = 0 then
            break                                        ▷ No more potential loop closures
        end if
        for each L_i ∈ L do                              ▷ For each loop-closure candidate
            {G_k, G_s, m} ← PROCESS_LC(L_i, G_k, G_s, m)
        end for
        {G_k, G_s, m} ← OPTIMIZEGRAPH(G_k, G_s, m)
    end while
    return m                                             ▷ Return map with optimized key-frame poses
end function
```

In the following, please check Algorithm 2 to follow the discussion on the top-level structure of the LC procedure.



(a) Coordinate frames involved in georeferencing



(b) Georeferencing as a factor graph

Figure 12. The problem of georeferencing a simple-map involves (a) finding the unknown relative pose between the ENU and metric map frames of reference, which can be turned up into optimizing the factor graph in (b). See discussion in Section 3.11.

First of all, the input and output of LC are both the view-based map $\mathfrak{m}$ (see definition in Section 3.9), comprising key-frames with SE(3) poses with respect to some arbitrary origin and raw sensory data (i.e., GNSS observations and LiDAR scans). The goal of LC is to detect when key-frames correspond to observations of the same place, even if their global coordinates from LO are far apart, and to correct all poses so the trajectory, and consequently the associated metric maps, are globally consistent. The problem is attacked from a hierarchical view-point by first splitting the input map into sub-maps (line 3 in the pseudocode), for which their local metric map bounding box (line 5) and georeferenced location from GNSS data (line 6, see Section 3.11) are evaluated. Note that GNSS is totally optional, but it will indeed help constraining the posterior search for LC candidates by reducing the uncertainty of relative poses between key-frames that are topologically far apart. Then, we build a two-level graph structure: on the bottom-level, a key-frame graph (G_k) has one node per map key-frame, while on a higher level, a sub-map graph (G_s) has one node per sub-map, as sketched in Figure 13. This approach follows a tradition of splitting large metric maps into submaps, such as done with Tectonic SAM in Ni et al. (2007) or with condensed factors in Grisetti et al. (2012). Two relevant differences of our approach with respect to those past works are: (i) the explicit distinction between condensed factors coming from LiDAR odometry and those from GNSS observations, and (ii) the usage of each of the two graphs for different purposes. In particular, the low-level graph is tightly coupled with a corresponding factor-graph in charge of actually optimizing the global poses for the entire map, while the higher-level sub-map graph is used to efficiently search for LC candidates. Initially, the low-level (key-frames) graph has a linear connection pattern, that is, each node i is connected via a pose constraint (${}_{i\text{-}1}\mathbf{T}_i$) to the immediately previous node $i-1$. Exactly the same happens on the sub-map (higher) level, where we use the notation ${}_{i\text{-}1}\mathbf{P}_i$ to denote relative poses between sub-maps. These two graphs are built in lines 8–9 of Algorithm 2. Then, if there is more than one sub-map where georeferencing was successful, we can exploit that information by means of additional connections between the reference key-frames of the first ever sub-map with GNSS data and all others; those additional edges are labeled as ${}_b\mathbf{T}_a^{GNSS}$ and ${}_b\mathbf{P}_a^{GNSS}$ in Figure 13 for the lower and higher levels, respectively. If there are such additional GNSS constraints, the key-frame graph is optimized using a factor graph (lines 10–12 of the pseudocode) to reduce absolute coordinate errors from those accumulated by pure LO (unbounded), to those of the GNSS receivers (maybe several meters, but bounded). It is worth mentioning that optimization is run in two passes: a first one without robust kernels, then a second robust optimization. Note that this is the opposite of what is typically done with SLAM frontends, where robust optimization comes first to reject outliers. However, if we want to handle loop closures with a large initial mismatch between the global poses of the two involved submaps (e.g., thanks to GNSS data, if available) robust kernels cannot be applied at first, or the error would be so large that it will be ignored as if it was an outlier.

Then, lines 13–22 comprise the loop to search for potential LC candidates, evaluate and apply them if found plausible, re-optimize all poses, and repeat until no further feasible candidates are found. A LC candidate is a pair of sub-maps that may correspond to the same physical place, at least, partially. Potential candidates are determined by evaluating the volumetric intersection ratio between the bounding boxes of every pair of sub-maps and accepting those above a given threshold (line 14). Given the potential large uncertainty in the relative poses of two submaps, metrically close to each other after a long topological loop, we use a Monte Carlo method to draw relative pose samples to evaluate the mathematical expectation of the intersection ratio. To find out the probabilistic relative pose between all sub-maps, we apply Dijkstra's algorithm to the higher-level graph to obtain, for each sub-map, a tree reflecting the shortest paths to all other sub-maps. Relative poses are then

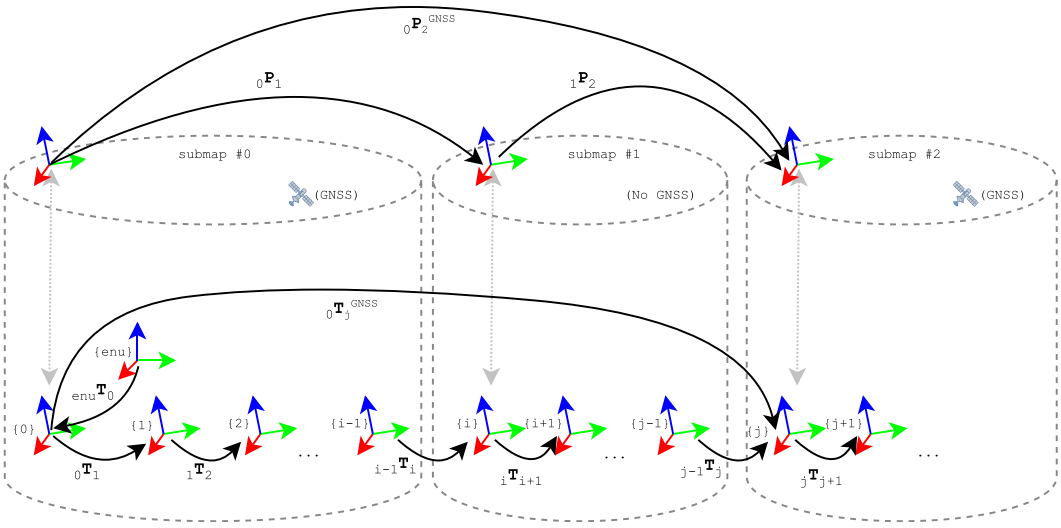


Figure 13. The two-level hierarchical graph used to determine loop closures and optimize the resulting key-frames. On the bottom, all map key-frames; on a higher level, one single frame per sub-map. Some sub-maps may include GNSS observations, allowing the determination of their coordinates in an ENU frame of reference. Refer to discussion in Section 3.12.

computed by composing transformations following the tree edges. Note that adding GNSS edges makes that sub-maps that initially may be far from each other (large uncertainty in their relative pose) now are topologically closer, hence their relative pose based on GNSS will be used instead of the pose composition chain throughout the lower key-frame level. This graph-based procedure is similar to many past works, such as Bosse et al. (2004) or Blanco et al. (2008). Note that an alternative way to determine potential loop closure candidates would be using descriptors for local maps, as classically done in visual place recognition or, in the context of LiDAR SLAM, in methods such as Scan Context (Kim and Kim, 2018) or ScanContext++ (Kim et al., 2021).

Each such potential LC candidate is then actually evaluated (line 19). Our approach is based on running a configurable ICP pipeline on the metric maps built from the view-based maps of the sub-maps, including the final evaluation of the registration quality (see Section 3.6.4) to decide whether the LC seems plausible or not. If it does, new edges are added to both graph levels with the obtained relative pose.

A few words are in order regarding this ICP pipeline, which is different than that in Figure 7 for LO. The first difference is the use of two point-cloud map layers: one for all objects near the ground plane, and another for everything else. The ICP pipeline then includes two instances of point-to-point matchers such that "ground" and "non-ground" points are only attempted to be paired against points belonging to the same category in the other sub-maps. Even a naive method to classify the point cloud into these two categories such as split by z (vertical) coordinates leads to good results, as long as local maps do not have significant tilt and the ground floor can be successfully segmented. More advanced methods have been used in the literature to segment the ground points in the context of loop closure, such as in Quatro++ (Lim et al., 2024), and could be used instead of our naive classifier.

Regarding ICP quality assessment, here we include the voxel occupancy-based method described in Section 3.6.4. Furthermore, the point-cloud metric maps used for LC have a different implementation than that used during LO: instead of the hashed-voxels maps used for LO, we use simple clouds with contiguous memory layout. The main motivation for this is that search for nearest neighbors (NN) (one of the dominating costs in ICP algorithms) with large search radii is more efficient using KD-Trees (the implementation of NN for the latter map type) than searching neighboring voxels in hashed maps.

3.13. Dataset sources

Finally, it is worth mentioning that the former work in Blanco (2019) already introduced data source MOLA modules as an abstraction of either, real live sensors or offline datasets. Such abstraction aims at enabling algorithm implementations in MOLA to immediately work with any of the supported datasets, without worrying about the details of each dataset file and data structures, sensor intrinsic and extrinsic calibration, ground truth (GT) axes conventions, etc.

This work extends the former modules in two substantial ways. First, new dataset sources have been defined, now offering access to:

- **KITTI dataset:** Read LiDAR scans (with per-point intensity field), camera images, and ground truth of this popular dataset introduced in Geiger et al. (2013).
- **KITTI-360 dataset:** Read LiDAR scans (with intensity field and interpolated timestamps), camera images, and

ground truth of the dataset, introduced in Liao et al. (2023).

- **MulRan dataset:** Read LiDAR scans (with intensity and timestamp fields), GNSS, IMU, and ground truth of Kim et al. (2020).
- **ParisLuco dataset:** Read LiDAR scans (with timestamps, and reconstructed per-point ring field), and ground truth of Dellenbach et al. (2022).
- **EuROC MAV dataset:** Read camera images and IMU for Burri et al. (2016). This dataset is not benchmarked in the present work since it does not contain LiDAR.
- **ROS 2:** Support is given for reading from both, a live system, or from *bag* files. Sensor messages for LiDAR point clouds, camera images, IMU readings, wheels odometry, and GNSS are all parsed for MOLA modules to process them.
- **MRPT rawlog files:** Rawlog files are an equivalent to ROS bag files, introduced in the MRPT framework in 2005. It has been used in public datasets, for example, Blanco-Claraco et al. (2014, 2019). Backwards compatibility and Operative System independence has been ensured during the whole life of the project, hence software written today using the latest MRPT version is able to read all past datasets, even if generated in a different OS or processor architecture. There exists an import tool from ROS 1 bag files to rawlogs, hence this MOLA input module also offers easy access to many of the ROS 1 public datasets, for example, HILTI challenge datasets (Helmberger et al., 2022) or the Newer College dataset (Zhang et al., 2021).

Second, two new abstract interfaces have been implemented in all the dataset sources above:

- **Replay control interface:** If used to replay datasets within an asynchronous network of processing nodes, this interface allows dynamically setting the replay speed, pausing and resuming, or fast-forwarding to a particular time step. The application GUI makes use of this generic interface to allow users to control the replay of any dataset.
- **Random-access interface:** For offline, batch processing, this interface allows randomly accessing to the raw sensory observations in the dataset. This enables, for example, running a given SLAM algorithm on a dataset as fast as possible while ensuring no frame is lost for SLAM methods running slower than the sensor rate.

Table 1. Summary of all datasets on which the proposed system has been experimentally validated.

Dataset	LiDAR sensor(s)	LiDAR rings	GNSS	Motion	Environment	Tested sequences	Total scans	Total length (km)
Newer college dataset → Section 4.9 (Zhang et al., 2021)	OS0-128	128	-	Handheld	College	12	78.2k	9.8
Newer college dataset → Section 4.9 (Ramezani et al., 2020)	OS1-64		-	Handheld	College	2	41.9k	4.7
MulRan → Section 4.1 (Kim et al., 2020)	OS1-64		✓	Vehicle	Campus/city	12	127.7k	123.4
KITTI → Section 4.2 (Geiger et al., 2013)	HDL-64E	64	-	Vehicle	Road/urban	22	43.5k	44.24
Voxgraph → Section 4.7 (Reijgwart et al., 2019)	OS1-64		-	Drone	Urban	1	2.2k	0.24
HILTI 2021 → Section 4.6 (Helmberger et al., 2022)	OS0-64		-	Drone	Industrial unit	1	0.9k	0.08
KITTI-360 → Section 4.3 (Liao et al., 2023)	HDL-64E		-	Vehicle	Urban	12	70.1k	58.7
ParisLuco → Section 4.4 (Dellenbach et al., 2022)	HDL32	32	-	Vehicle	City	1	12.7k	4.0
Almería forests → Section 5.3 (Aguilar et al., 2024)	OS0-32		-	Backpack	Forests	6	19.9k	1.6
NTU-VIRAL → Section 4.5 (Nguyen et al., 2022)	2 × OS1-16		-	Drone	Campus	9	32.6k	2.2
DARPA subterranean → Section 4.8 (Tranzatto et al., 2022)	VLP16	16	-	ANYmal C	Caves	4	143.1k	2.0
UAL campus → Section 4.10 (Blanco-Claraco et al., 2019)	VLP16		✓	Vehicle	Campus	1	15.5k	3.4
fr079 → Section 5.1 (Howard and Roy, 2003)	SICK LRF	1	-	PowerBot	Offices	1	1.2k	0.1
Málaga CS faculty → Section 5.1 (Blanco-Claraco, 2024)	SICK LRF		-	Wheelchair	Campus	1	4.7k	1.9
					Overall:	**85**	**594.2k**	**256.4**

Finally, a word is in order regarding memory management for dataset input modules. Loading and keeping in memory whole datasets may become unfeasible for a typical desktop computer, for example, some individual sequences of the MulRan dataset (Kim et al., 2020) take 29 GiB. The present framework exploits the usage of MRPT lazy-load mechanism in different areas, including dataset sources. Observations that tend to be heavy in memory requirements (i.e., images, 3D LiDAR scans, and RGB + D frames) can be automatically loaded to memory when accessed, to be unloaded once a fixed number of additional frames have been processed. Most MOLA dataset input modules also feature read-ahead in a multi-thread fashion to minimize the performance impact of such lazy-load mechanism. In practice, this creates a cache of recently-used and about-to-be-used sensory data quickly accessible in RAM while SLAM is processing them.

4. LiDAR odometry quantitative experimental validation

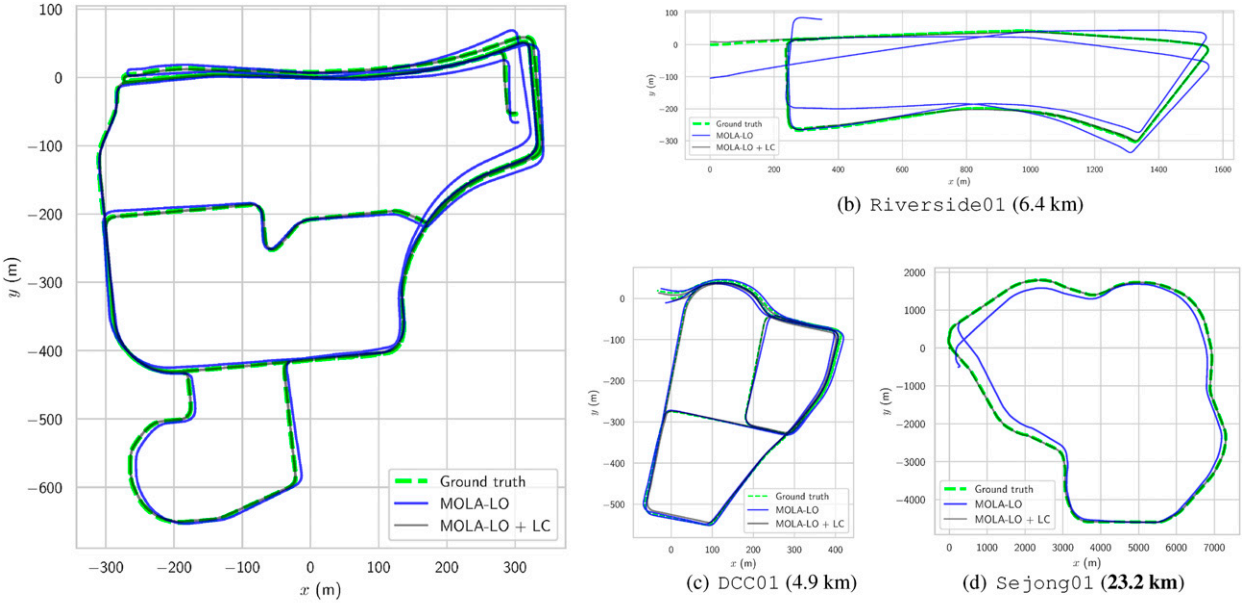
To measure the accuracy and robustness of the proposed system as a whole against different LiDAR scanning and vehicle motion patterns, we performed an exhaustive benchmarking against several public datasets that include 3D LiDAR and ground truth trajectories.

Please refer to Table 1 for an overview of all the used datasets. Note that we selected an extensive list of public datasets in order to cover a large variety of motion models (vehicles, drones, hand-held, etc.) and sensor models and resolutions. In particular, it is noteworthy that we used exactly the same system configuration (called "default configuration" in the following) for all 3D LiDAR datasets, from 16 to 128 beams. In all cases, our system provided accurate trajectory estimations without diverging, while keeping execution times fast enough for running significantly faster than the real sensor rate. That said, these additional configurations have been also evaluated for selected datasets with the aim of serving for ablation studies in Section 7, to demonstrate a particular feature, or to illustrate the flexibility of the proposed system:

- A 2D-LiDAR configuration, addressed in Section 5.1.
- A configuration where the local map update decider (see Section 3.4) is not used and the map is updated for all time steps.
- A configuration using an alternative 3D-NDT metric map instead of plain point clouds.

In order to evaluate our solution against existing state of the art LO methods, the next subsections provide these quantitative error metrics:

- Relative translational error (RTE) (%) and relative rotational error (d/hm, that is, degrees per 100 m): these popular metrics were defined in Geiger et al. (2013), and measure relative pose errors, averaging for poses that are a given distance apart. These metrics have been evaluated using a derived work from the original reference metrics implementation provided by Geiger et al. (2013), which assumed that LiDAR scans and ground-truth are perfectly synchronized and given at the same rate. This is not possible for all dataset sources, hence these metrics will not be available for them.



(a) KAIST01 (6.1 km long)

(b) Riverside01 (6.4 km)

(c) DCC01 (4.9 km)

(d) Sejong01 (**23.2 km**)

Figure 14. Estimated trajectories for the proposed LiDAR odometry system (in its default configuration) applied to the **MulRan dataset**, compared to ground truth. Only one sequence is shown for each of the four locations. Estimated paths including loop closure (Section 3.12) are denoted with legend "MOLA-LO + LC."

Table 2. Relative translational error (RTE) (%), relative rotational error (RRE) (d/hm, that is, degrees per 100 m) (as defined in Geiger et al. (2013)), and absolute translational error (ATE) (rmse values in meters as reported by "`evo_ape -a`") for the **MulRan dataset**. All sequences have loop closures. Smaller numbers are better. Bold means best accuracy in each sequence and category.

Method	KAIST			DCC			Riverside			Sejong			Avr. time per frame
	01	02	03	01	02	03	01	02	03	01	02	03	
KISS-ICP	2.22%	2.15%	2.43%	2.62%	2.32%	1.92%	3.33%	3.15%	2.55%	4.28%	4.79%	4.93%	23 ms
Vizzo et al. (2023)	0.66 d/hm	0.68 d/hm	0.70 d/hm	0.65 d/hm	0.63 d/hm1	0.62 d/hm	0.74 d/hm	0.68 d/hm	0.53 d/hm	0.62 d/hm	0.68 d/hm	0.82 d/hm	
	13.36 m	11.76 m	12.32 m	12.56 m	0.13 m	10.54 m	41.50 m	35.96 m	29.32 m	373.53 m	417.26 m	344.05 m	
SiMpLE (MulRan)	**2.03**%	**1.97**%	**2.22**%	**2.45**%	**1.99**%	**1.75**%	**3.10**%	**2.97**%	**2.40**%	4.15%	5.06%	5.26%	32 ms
Bhandari et al. (2024)	**0.58** d/hm	**0.60** d/hm	**0.65** d/hm	**0.60** d/hm	**0.53** d/hm	**0.56** d/hm	**0.61** d/hm	**0.58** d/hm	0.51 d/hm	**0.47** d/hm	**0.55** d/hm	**0.70** d/hm	
	13.29 m	9.56 m	10.10 m	12.17 m	8.04 m	7.38 m	**23.90** m	30.63 m	50.84 m	636.11 m	690.61 m	566.00 m	
MOLA-LO	2.13%	2.12%	2.38%	2.56%	2.34%	1.88%	3.27%	3.10%	2.52%	4.17%	4.67%	4.83%	24 ms
Config.: default (ours)	0.65 d/hm	0.70 d/hm	0.71 d/hm	0.63 d/hm	0.62 d/hm	0.65 d/hm	0.72 d/hm	0.66 d/hm	0.53 d/hm	0.60 d/hm	0.67 d/hm	0.81 d/hm	
	10.23 m	10.51 m	10.79 m	10.90 m	9.48 m	8.74 m	38.05 m	32.89 m	30.27 m	429.76 m	455.47 m	333.63 m	
MOLA-LO	2.11%	2.09%	2.33%	2.53%	2.27%	1.87%	3.25%	3.08%	2.50%	4.06%	4.58%	4.85%	24 ms
Config.: always update map (ours)	0.65 d/hm	0.69 d/hm	0.71 d/hm	0.63 d/hm	0.61 d/hm	0.66 d/hm	0.71 d/hm	0.65 d/hm	0.52 d/hm	0.58 d/hm	0.66 d/hm	0.81 d/hm	
	10.58 m	9.65 m	10.18 m	10.88 m	9.18 m	9.14 m	35.13 m	31.25 m	29.55 m	466.81 m	512.25 m	413.81 m	
MOLA-LO	2.09%	2.05%	2.30%	2.53%	2.18%	1.89%	3.19%	3.08%	2.49%	**3.91**%	**4.45**%	**4.68**%	39 ms
Config.: 3D-NDT (ours)	0.64 d/hm	0.68 d/hm	0.71 d/hm	0.61 d/hm	0.61 d/hm	0.65 d/hm	0.69 d/hm	0.66 d/hm	**0.50** d/hm	0.56 d/hm	0.63 d/hm	0.78 d/hm	
	8.77 m	**7.44** m	**7.79** m	**9.20** m	**6.74** m	**6.83** m	24.91 m	**21.51** m	**20.30** m	**317.56** m	**337.94** m	**199.25** m	
MOLA-LO + LC (ours) (default LO + GNSS)	**2.04**%	**1.91**%	**2.15**%	**2.34**%	2.27%	**1.70**%	**2.66**%	**2.44**%	**2.08**%	**3.20**%	**3.67**%	**4.21**%	40 ms
	0.64 d/hm	**0.64** d/hm	**0.69** d/hm	**0.61** d/hm	0.67 d/hm	**0.60** d/hm	**0.62** d/hm	**0.57** d/hm	**0.42** d/hm	**0.50** d/hm	**0.57** d/hm	**0.74** d/hm	
	2.42 m	**2.13** m	**1.93** m	**5.14** m	4.21 m	**1.90** m	**3.98** m	**2.92** m	**3.56** m	**36.64** m	**41.90** m	**26.55** m	
MOLA-LO + LC (ours) (default LO, no GNSS)	2.08%	1.98%	2.34%	2.50%	**2.05**%	1.76%	3.79%	3.21%	2.52%	4.08%	4.61%	4.80%	42 ms
	0.63 d/hm	0.66 d/hm	0.72 d/hm	0.65 d/hm	**0.58** d/hm	0.61 d/hm	0.87 d/hm	0.65 d/hm	0.49 d/hm	0.57 d/hm	0.65 d/hm	0.80 d/hm	
	2.57 m	2.56 m	2.76 m	5.31 m	**2.75** m	1.91 m	22.76 m	9.09 m	6.93 m	381.54 m	419.82 m	315.43 m	

- Absolute Trajectory Error (ATE) (m): We use the RMSE value of the distances between time-aligned ground-truth and estimated trajectories, after spatially aligning them using Umeyama'a method (Umeyama, 1991). Evaluation is performed using the open-sourced tool `evo_traj` (Grupp, 2017).

Unless noted otherwise, metrics for alternative methods have been taken from their original publications. The exception is KISS-ICP (Vizzo et al., 2023), which due to its simplicity of integration into third-party systems, has been integrated as another MOLA LO module, hence enabling running all datasets on it and obtaining trajectories in the same format than our proposed system, becoming an excellent baseline SOTA reference for all the datasets.

Sometimes a particular LO method *diverges*, which is represented in the presented metrics as ×. Divergence means a significant drift, large enough to render the overall trajectory (and map) useless. In particular, our criteria for divergence means that at least one of the following conditions hold: (i) trajectory gets "trapped" into a circular orbit, and (ii) relative angular errors larger than 45° for short distances (e.g., less than 10 m).

All results are exactly reproducible by interested readers following the software instructions on the project website. Since all of our proposed methods and software implementations are deterministic, there will be no differences in the results disregarding different CPU speed or number of parallel cores.

4.1. MulRan dataset

This multimodal dataset comprises Ouster OS1-64 LiDAR data, consumer-grade GNSS, and precise ground truth for 12 driving sequences (Kim et al., 2020). All sequences include several loop closures, except the longest ones (`Sejong{01,02,03}`) which only include one, although the longest loop in all the benchmarked datasets (>20 km).

We have compared our LO system in two configurations, (i) its default one and (ii) its 3D-NDT version, against two other state-of-the-art LO methods: KISS-ICP (Vizzo et al., 2023) using its default settings, and SiMpLE (Bhandari et al., 2024) using their parameter set optimized for this dataset. Note that only SiMpLE has parameters optimized for this particular dataset, while our two configurations and KISS-ICP use the same parameters for all datasets. MOLA wrappers have been provided for both methods to provide a fair comparison in terms of identical API to provide input raw data and to collect estimated trajectories. The MOLA dataset source (see Section 3.13) for MulRan takes care of removing LiDAR scans at the beginning or end of each sequence that have no associated ground truth. In this way, we can use the KITTI evaluation metrics (RTE and RRE) apart of APE (see metrics definitions in Section 4). Consumer-grade GNSS observations are not used by our LO system, but they are while applying the loop-closure post-processing stage (explained in Section 3.12).

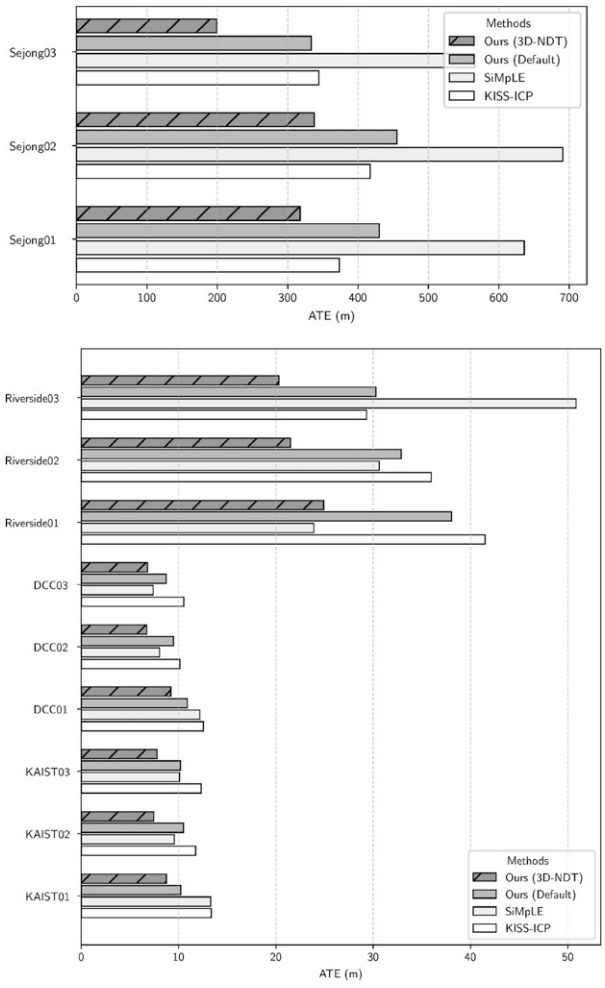


Figure 15. Summary of absolute translational error (ATE) for the **MulRan dataset**.

Sample trajectories from our method are shown in Figure 14, compared to GT. It can be seen that the output from loop closure is nearly indistinguishable from GT. A quantitative benchmark is provided in Table 2, where all three metrics (RTE, RRE, APE) are provided, together with average run times per LiDAR scan (measured on an Intel i7-8700 at 3.20 GHz). A graphical summary of ATE metrics is also provided in Figure 15. All LO methods are faster than the sensor rate (10 Hz), hence all of them would be able to run onboard in real-time. An interesting observation is that there is not always a best performing method for a given sequence, since different methods may have the lowest values for one given metric only: LO methods would rank in a different order depending on the metric of interest. That said, for this dataset we can see how our 3D-NDT configuration clearly outperforms the others, despite not being specifically tuned for this dataset as SiMpLE is. Our SLAM version (reflected as "MOLA LO + LC" in the table) achieves, obviously, better results than the pure LO

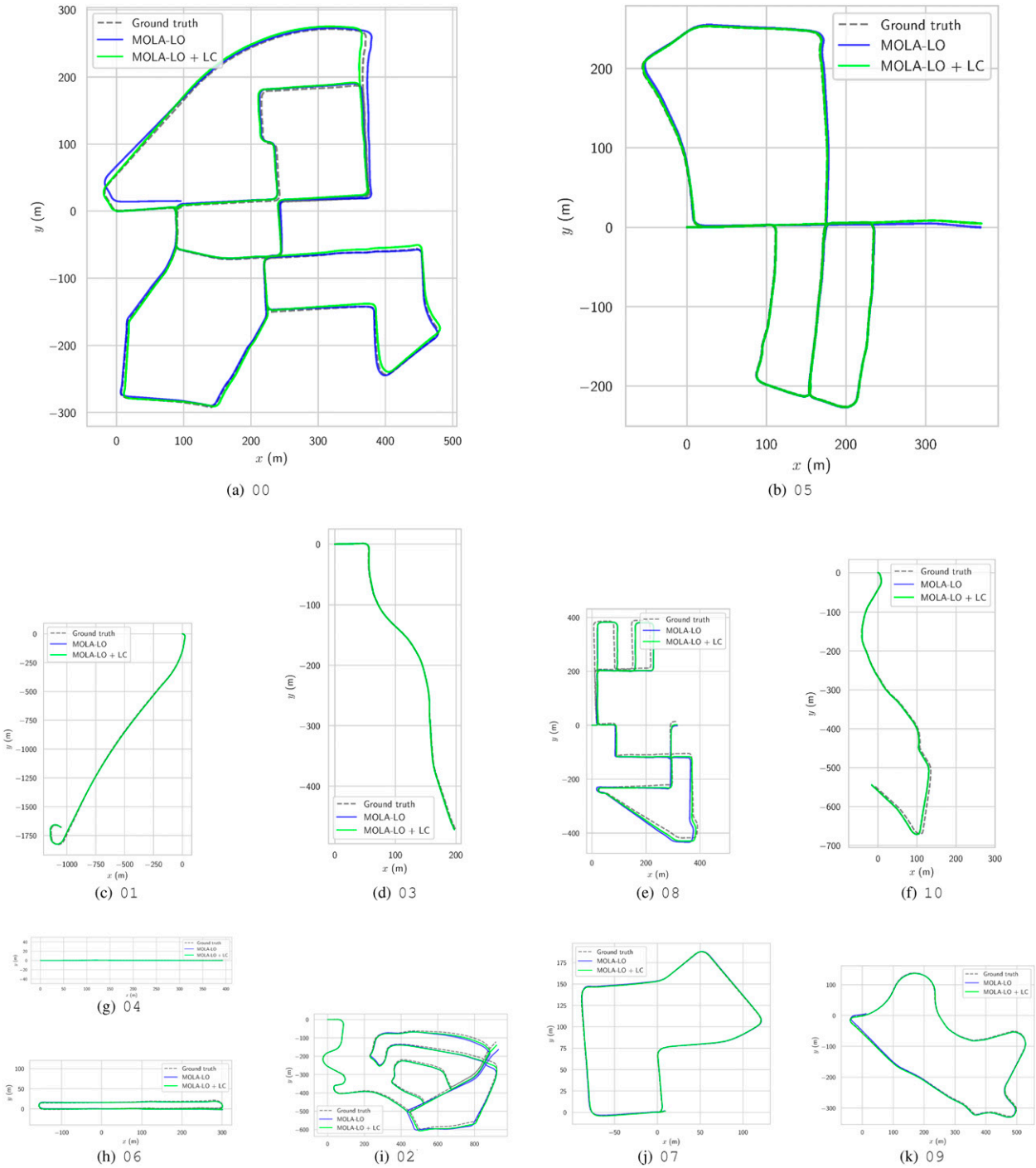


Figure 16. Estimated trajectories for the proposed LiDAR odometry system ("MOLA-LO") applied to the **KITTI odometry dataset**, compared to ground truth. Trajectories denoted as "MOLA-LO + LC" include loop-closure. See discussion in Section 4.2.

methods, hence their values are not marked in bold since the method is qualitatively different.

4.2. KITTI-odometry dataset

The KITTI odometry dataset (Geiger et al., 2013) was the first widely spread benchmark for LiDAR and visual odometry and SLAM methods, hence its popularity and importance. It comprises 11 training sequences (numbered 00–10) with public ground truth and other 11 evaluation sequences with undisclosed ground truth trajectories. Available sensors in this driving dataset include 3D LiDAR (a Velodyne HDL-64E) and two pairs of forward-facing cameras.

The trajectories estimated by our system (with its default configuration) for the 11 training sequences are shown in

Table 3. Relative translational error (RTE) (%), relative rotational error (RRE) (d/hm, that is, degrees per 100 m) (as defined in Geiger et al. (2013)), and absolute translational error (ATE) (rmse values in meters as reported by "evo_ape -a") for the training and evaluation sets of sequences in the **KITTI visual-LiDAR odometry dataset**. Bold figures mark either the absolute best for each sequence if it is pure odometry method (no loop closure), or the best ones for each category (with and without loop closure). Average RTE is also shown for the whole training (00–10) and validating (11–21) KITTI sequence subsets, with the latter being publicly available on the KITTI odometry leaderboard website. Sequences denoted with * numbers are those having loop closures. Results for IMLS-SLAM, MULLS, and CT-ICP2 have been taken from their respective publications (hence the missing RRE and ATE values). The rest have been evaluated manually from result trajectories.

Method	00★	01	02★	03	04	05★	06★	07★	08★	09★	10	Avr.00–10 (%)	Avr.11–21	Avr. time per frame
IMLS-SLAM Deschaud (2018)	**0.50**% 	0.82% 	0.53% 	0.68% 	**0.33%** 	0.32% 	0.33% 	0.33% 	**0.80%** 	0.55% 	0.53% 	**0.55**	0.69% 0.18 d/hm 	~1000 ms
KISS-ICP Vizzo et al. (2023)	0.51% 0.19 d/hm 3.74 m	**0.72**% 0.11 d/hm 3.72 m	**0.52**% 0.15 d/hm 6.60 m	0.66% 0.16 d/hm **0.43** m	0.36% 0.14 d/hm 0.27 m	**0.30**% 0.14 d/hm 1.47 m	**0.26**% 0.08 d/hm 0.39 m	0.32% **0.17** d/hm **0.30** m	0.82% 0.18 d/hm 2.23 m	0.49% 0.13 d/hm 1.45 m	0.57% 0.19 d/hm 0.81 m	**0.55**	0.61% 0.17 d/hm 	**13** ms
SiMpLE (offline) Bhandari et al. (2024)	0.51% **0.18** d/hm **3.67** m	0.86% **0.09** d/hm **2.50** m	0.53% **0.13** d/hm **4.74** m	0.70% **0.14** d/hm 0.45 m	0.40% **0.10** d/hm 0.27 m	**0.30**% **0.13** d/hm **1.23** m	0.28% 0.08 d/hm 0.34 m	**0.30**% 0.18 d/hm 0.38 m	**0.80**% **0.16** d/hm **1.97** m	0.55% **0.12** d/hm **1.30** m	0.51% **0.15** d/hm 0.76 m	**0.55**	0.62%	545 ms
SiMpLE (online) Bhandari et al. (2024)	0.95% 0.45 d/hm 5.45 m	1.48% 0.36 d/hm 12.25 m	1.07% 0.40 d/hm 7.11 m	1.34% 0.41 d/hm 0.59 m	0.80% 0.53 d/hm 0.48 m	0.61% 0.41 d/hm 1.95 m	0.59% 0.31 d/hm 0.92 m	0.50% 0.57 d/hm 0.81 m	1.13% 0.37 d/hm 2.66 m	1.19% 0.41 d/hm 2.15 m	1.43% 0.55 d/hm 1.98 m	0.62	N/A	126 ms
MOLA-LO (default) (ours)	0.51% 0.19 d/hm 3.85 m	0.74% 0.14 d/hm 5.08 m	**0.52**% 0.15 d/hm 6.91 m	0.63% 0.17 d/hm **0.43** m	0.37% 0.13 d/hm 0.28 m	0.33% 0.15 d/hm 1.34 m	0.29% **0.07** d/hm **0.29** m	0.36% 0.18 d/hm 0.37 m	0.81% 0.19 d/hm 2.35 m	0.49% 0.14 d/hm 1.83 m	0.57% 0.18 d/hm 0.81 m	**0.55**	0.62% **0.17** d/hm 	23 ms
MOLA-LO (3D-NDT) (ours)	0.56% 0.21 d/hm 3.74 m	0.87% 0.12 d/hm 3.61 m	0.57% 0.17 d/hm 8.34 m	**0.60**%0.18 d/hm0.50 m	0.41% 0.12 d/hm **0.26** m	0.35% 0.15 d/hm 1.24 m	0.31% 0.08 d/hm 0.35 m	0.33% **0.17** d/hm 0.39 m	0.82% 0.20 d/hm 2.36 m	**0.47**% 0.16 d/hm 1.40 m	**0.50**% 0.21 d/hm **0.75** m	0.58	N/A	32 ms
MOLA-LO (Horn's) (ours)	0.69% 0.29 d/hm 4.88 m	0.73% 0.14 d/hm 4.97 m	0.62% 0.21 d/hm 9.17 m	0.64% 0.17 d/hm 0.45 m	0.37% 0.14 d/hm 0.28 m	0.42% 0.22 d/hm 1.65 m	0.28% 0.08 d/hm 0.31 m	0.47% 0.29 d/hm 0.51 m	0.87% 0.20 d/hm 2.57 m	0.68% 0.18 d/hm 2.43 m	0.65% 0.22 d/hm 0.97 m	0.65	N/A	45 ms
MULLS (LC) Pan et al. (2021)	0.54% 	**0.62%** 	0.69% 	**0.61**% 	**0.35**% 	0.29% 	0.29% 	**0.27**% 	0.83% 	0.51% 	0.61% 	**0.52**	0.65% 0.19 d/hm 	100 ms
CT-ICP2 (LC) Dellenbach et al. (2022)	**0.49%** 	0.76% 	**0.52**% 	0.72% 	0.39% 	**0.25%** 	**0.27**% 	0.31% 	**0.81**% 	**0.49**% 	0.48% 	0.53	**0.58**% **0.12** d/hm 	60 ms
MOLA-LO (default)+ LC (ours)	0.60% 0.17 d/hm 0.81 m	0.74% 0.14 d/hm 5.08 m	0.55% 0.13 d/hm 2.90 m	0.63% 0.17 d/hm 0.43 m	0.37% 0.13 d/hm 0.28 m	0.34% 0.13 d/hm 0.34 m	0.33% 0.13 d/hm 0.34 m	0.39% 0.16 d/hm 0.28 m	**0.81**% 0.17 d/hm 2.05 m	**0.49**% 0.14 d/hm 1.83 m	0.57% 0.18 d/hm 0.81 m	0.58	0.66% 0.16 d/hm 	**31** ms

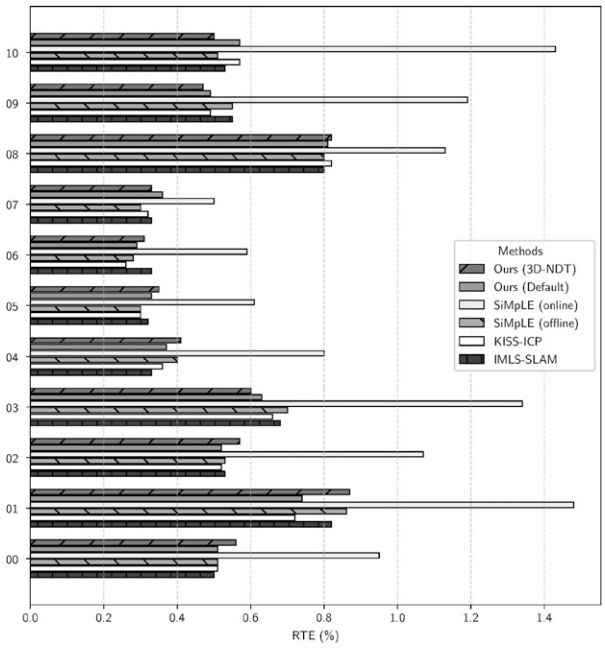


Figure 17. Relative translational error (RTE) (%) for the training sequences in the **KITTI visual-LiDAR odometry dataset**.

Figure 16 for the LO-only module and for the SLAM system (including loop closure), compared with ground truth. Quantitative results are shown in Table 3 and graphically in Figure 17. A larger number of alternative methods are compared in this case due to the popularity of this particular dataset. Note that two configurations of SiMpLE (Bhandari et al., 2024) have been included ("online" and "offline"), with one of them being faster and the other more accurate. However, note that none would be able to run at sensor rate (10 Hz). Excepting IMLS-SLAM and SiMpLE, all others are fast enough to run in real-time at the sensor rate or faster.

Regarding the accuracy of the estimated trajectories for LO methods, it can be seen in Table 3 that the slower and more precise configuration of SiMpLE is slightly better than the rest, although accuracy metrics for these SOTA methods are all quite similar, as demonstrated by an almost perfect tie in the average RTE metric used to evaluate all training (RTE $\sim$ 0.55%) and all testing sequences (RTE $\sim$ 0.61–0.62%).

Note that all shown results for KISS-ICP, SiMpLE, and our system, have been run using the same MOLA KITTI dataset source (see Section 3.13), which includes the 0.205° vertical angle correction of all original dataset scans due to a miscalibration; see Deschaud (2018). Another well-known issue with KITTI is the wrong ground truth in part of sequence `08`, which explains the relative elevated apparent errors of all methods in that sequence.

Regarding the results on the evaluation sequences (11–21), for which ground truth is not publicly available, at the time of writing our system ranks 13th among all LiDAR-only methods (out of 140 total submissions), and 6th among those with an open source implementation, according to the project website[4].

4.3. KITTI-360 dataset

This automotive dataset is a follow-up of the first KITTI dataset, including more sensors (fish-eye cameras, 2D LiDAR, and IMU) and longer driving sequences in suburban scenarios (Liao et al., 2023).

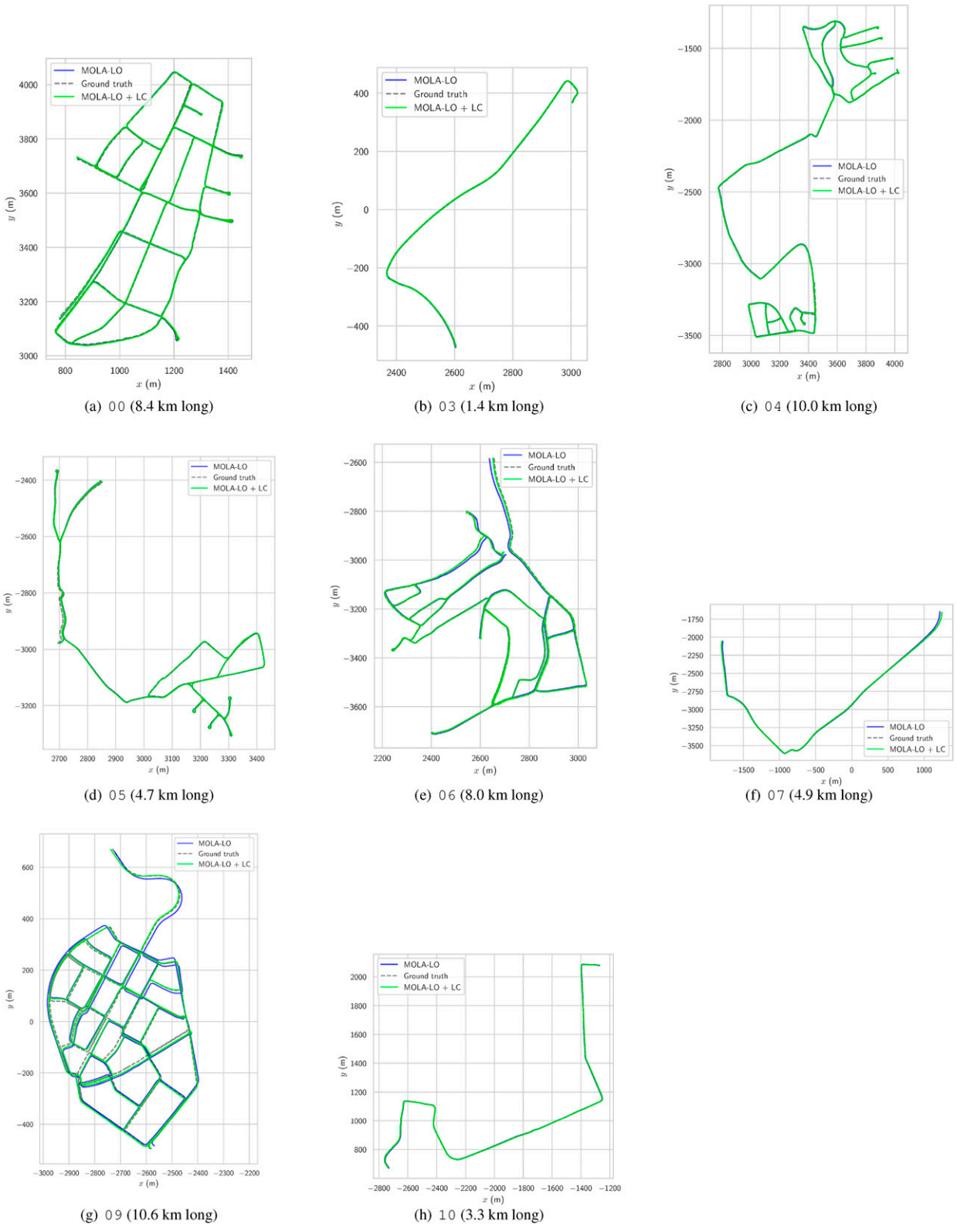
Trajectories estimated by our system are shown in Figure 18, while quantitative performance results, compared to KISS-ICP as SOTA baseline, are summarized in Table 4. As can be seen in the table, where we evaluated the ATE metric, our system achieves better accuracy than KISS-ICP in 6 out of 8 sequences. Regarding our SLAM method with loop closures, and given that most KITTI-360 sequences, except for `03`, `07`, and `10`, have multiple loop-closures, obtained ATE is further reduced. This dataset also includes 4 evaluation sequences without public ground truth for benchmarking on the "trajectory estimation" public leaderboard website[5], where our SLAM solution ranks 3rd out of 5 submissions at the time of writing. In this leaderboard, more modern than the original KITTI, the metric that determines the classification is APE, and the results available online are: CT-ICP2 with LiDAR SLAM in Dellenbach et al. (2022) (0.50 m), SOFT2 with stereo visual SLAM in Cvišić et al. (2022) (0.70 m), our SLAM system (0.72 m), ORB-SLAM2 with stereo visual SLAM in Mur-Artal and Tardos (2017) (1.92 m), and SUMA++ with LiDAR SLAM in Chen et al. (2019) (3.13 m).

4.4. ParisLuco dataset

This automotive dataset was presented in Deschaud (2018) and comprises one single continuous driving sequence with two loops around the Luxembourg Garden in Paris, while grabbing 3D LiDAR scans from a Velodyne HDL-32. Figure 19(a) shows the trajectories estimated by the reference SOTA method KISS-ICP, our LO method, and our full SLAM solution including loop closure detection, along with ground truth. It can be seen that both LO methods perform very similarly, with the SLAM solution being, as expected, much closer to ground truth. Looking at the quantitative ATE metrics, in Table 5, it can be seen that both LO methods have similar accuracy, with KISS-ICP having a small advantage in this case. It is interesting to analyze where these trajectory errors tend to accumulate. As it typically occurs with 3D LiDAR LO, errors accumulate faster in the vertical direction (perpendicular to the ground plane), as can be clearly seen in the trajectory component plots in Figure 19(b), where absolute horizontal errors are in the order of 1 m while vertical errors reach a maximum of 100 m. Loop closures, naturally, bound these vertical errors, reducing them to a maximum of $\sim 45\ m$.

4.5. NTU VIRAL: two 3D LiDARs

This airborne dataset was presented in Nguyen et al. (2022), and comprises nine sequences (three in each of three scenarios) of visual, inertial, and LiDAR data as a drone flights within the range of a motion capture system that serves as



(a) 00 (8.4 km long) (b) 03 (1.4 km long) (c) 04 (10.0 km long)

(d) 05 (4.7 km long) (e) 06 (8.0 km long) (f) 07 (4.9 km long)

(g) 09 (10.6 km long) (h) 10 (3.3 km long)

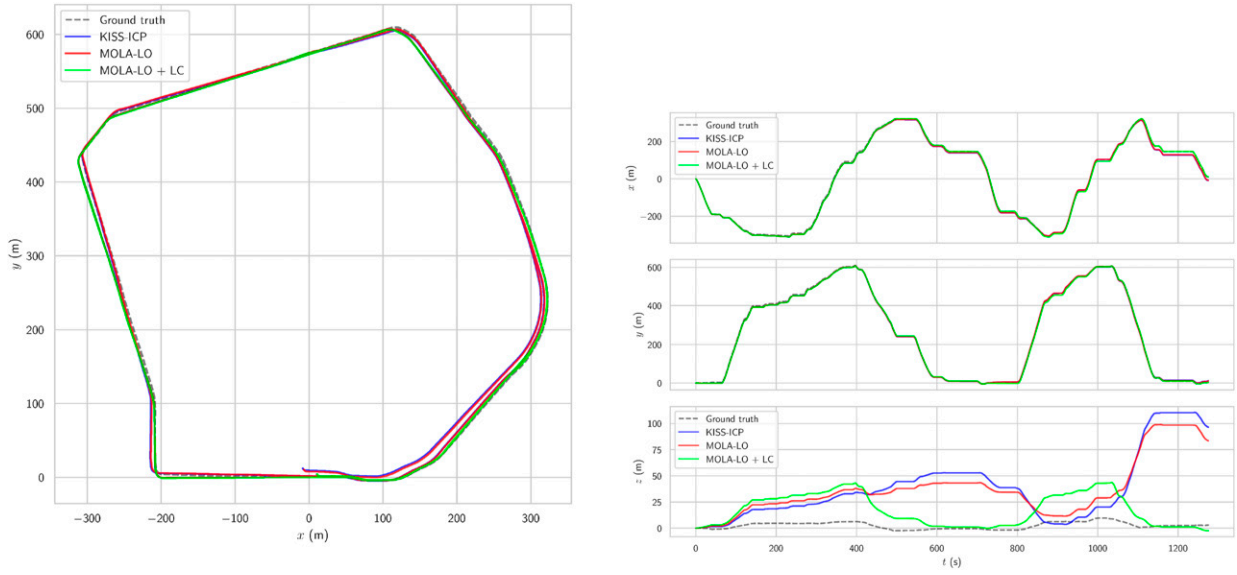
Figure 18. Estimated trajectories for the proposed LiDAR odometry system ("MOLA-LO") applied to the **KITTI-360 odometry dataset**, compared to ground truth. Trajectories denoted as "MOLA-LO + LC" include the post-processing loop-closure stage. See discussion in Section 4.3.

accurate ground truth. Two Velodyne VLP-16 scanners are installed in the drone, one in an horizontal position (as typically placed in automotive datasets) and another one facing vertically towards the ground. Hence, this dataset is ideal to benchmark visual or LiDAR odometry algorithms, possibly making use of high-rate inertial data, in the challenging conditions of drone flight, where linear and angular accelerations tend to be more abrupt than in automotive datasets. The dataset was used in recent works like Nguyen et al. (2024) to evaluate tightly-coupled LiDAR-inertial methods. Table 6 compares the performance of SOTA visual-LiDAR-inertial methods reported in Nguyen et al. (2024) to our solution and KISS-ICP, although they do not make use of neither inertial data, nor images. Loop-closure was not evaluated in this dataset since the range of motion is in the order of magnitude of the LiDAR range, hence there are no loop closures.

From the accuracy metrics, it is clear that methods using inertial sensors (LIOSAM and VIRAL SLAM) have the best performance, although we believe that the smarter sampling of LiDAR scans using edge and plane key points (Nguyen et al., 2021b) plays a more significant role in achieving better accuracy than using an IMU; this topic would require further research. In contrast, LO methods KISS-ICP and ours, using only one (horizontal) LiDAR, have trajectory errors at least one order of magnitude larger than the aforementioned LIO methods, with KISS-ICP performing poorer in this case. Analyzing the spatial trajectories, it becomes clear that vertical movements of the drone are where these two methods perform worst, which can be explained by the simplistic uniform sampling of point clouds (instead of searching for specific features), which makes hard to distinguish vertical motions from remaining static. With this insight in mind, we provide an example of the flexibility of the present framework by adding two additional experiments evaluating MOLA-LO (the last two rows in Table 6). First, we put at test the capability of our system to handle several LiDARs, by enabling both vertical and horizontal LiDARs as inputs, with the idea of helping determining vertical motions. Our system (recall Figure 4 and the default pipeline in Figure 7) then waits for two consecutive scans before populating the observation metric map, taking into account the time offset between both scans to accurately de-skew both in a spatially coherent way. The rest remains exactly the same for the "MOLA-LO (2 LiDARs)" experiment shown in the table. As can be seen, using this vertical scanner indeed improves accuracy, but it is still far from LIO techniques. Note that all these results for MOLA-LO use the same configuration than in all other automotive datasets, with an automatic determination of the minimum points range to be filtered to prevent observations of the vehicle body. In this particular dataset, the smaller size of the drone makes that these rules actually prevent using valid information

Table 4. Absolute translational error (ATE) (rmse values in meters as reported by "`evo_ape -a`") for the **KITTI-360** dataset. Bold means best accuracy in each sequence and category.

Method	00	03	04	05	06	07	09	10
KISS-ICP Vizzo et al. (2023)	5.50 m	**0.62** m	5.85 m	3.05 m	10.45 m	**4.22** m	12.30 m	5.90 m
MOLA-LO (ours)	**2.81** m	0.72 m	**4.89** m	**2.39** m	**6.37** m	8.22 m	**10.78** m	**1.75** m
MOLA-LO + LC (ours)	0.72 m	0.72 m	4.89 m	1.02 m	4.03 m	8.22 m	4.67 m	1.75 m



(a) Trajectories (b) Trajectory components

Figure 19. Estimated trajectories for the proposed LiDAR odometry system ("MOLA-LO") applied to the **Paris LuCo** dataset, compared to KISS-ICP, and to ground truth. Trajectories denoted as "MOLA-LO + LC" include loop-closure. See discussion in Section 4.4.

about nearby objects, but we did not manually tune this filter to ensure all datasets are run with the same configuration. Then, in a second experiment we modified the default pipeline in Figure 7 to create two local maps: "near map" and "far map," for nearby and distant points, respectively. ICP pipeline blocks are then configured to search for potential pairings between nearby LiDAR scan points and the "near map," and between all LiDAR points and the "far map" (interested readers can check the details in the configuration YAML file `lidar3d-near-far.yaml` available online). This configuration corresponds to the last row in the table. As can be seen, this method now has errors in the order of magnitude of LIO techniques, even surpassing LIOSAM on some particular sequences. Finally, using a smarter sampling of feature points would probably further improve the results, but this point is left for future works.

4.6. HILTI 2021 dataset

This dataset, presented in Helmberger et al. (2022), comprises several challenging sequences indoors and mixed indoor-outdoor scenarios recorded with an Ouster OS0-64 LiDAR. From the published sequences, only two of them have full SE(3) ground truth. From those two, we selected the largest sequence (the Drone Testing Area) for benchmarking our LO system. The resulting trajectories are illustrated in Figure 20. Quantitatively, we measured the rmse ATE for MOLA-LO against ground truth obtaining 0.18 m, slightly better than KISS-ICP (Vizzo et al., 2023) which obtains an ATE of 0.21 m. It is noteworthy that our system fails to converge for some of the other sequences in this dataset (whose ground truth is not publicly released), where feature-extraction or multi-modality seem essential for robustly cope with scenarios like narrow, featureless indoor spaces.

Table 5. Absolute translational error (ATE) (rmse values in meters as reported by "`evo_ape -a`") for the **Paris LuCo** dataset, in which there is only one sequence. Bold means best accuracy in each sequence and category.

Method	Paris LuCo (m)	Avr. time per frame (ms)
KISS-ICP Vizzo et al. (2023)	21.2	24
MOLA-LO (default) (ours)	22.1	23
MOLA-LO (3D-NDT) (ours)	**12.3**	88
MOLA-LO (default) + LC (ours)	2.38	50

4.7. Voxgraph dataset

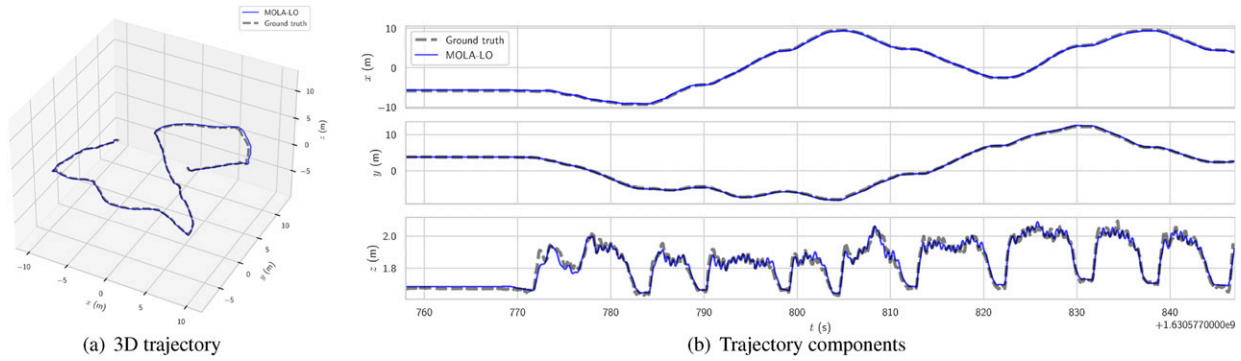
This aerial dataset, presented in Reijgwart et al. (2019), comprises one sequence of a drone flight outdoors, equipped with an Ouster OS1-64 LiDAR and featuring RTK-based ground truth. We compare the ATE metric achieved by several LO SOTA methods in Table 7, where it can be seen that both KISS-ICP and ours achieve a significant better accuracy, with KISS-ICP having a small advantage in this case. The estimated trajectory for our method and the partially-available ground truth are illustrated in Figure 21. Note that ATE metrics have been only evaluated for those segments of the trajectory with corresponding ground truth. As in the HILTI Drone Testing Arena sequence (Section 4.6), loop closure is not required due to the small size of the environment, hence we only evaluate LO methods.

4.8. DARPA subterranean dataset

We now evaluate our LO method against the dataset presented in Tranzatto et al. (2022) by Team Cerberus, winners of the subterranean 2021 DARPA challenge. The public

Table 6. Absolute translational error (ATE) (rmse values in meters as reported by "evo_ape -a") for the **NTU Viral** dataset. Bold means best accuracy in each sequence.

	EEE			NYA			SBS			Avr. time
Method	01	02	03	01	02	03	01	02	03	per frame
LIOSAM (1 LiDAR + IMU) Shan et al. (2020)	0.075 m	0.069 m	0.101 m	0.076 m	0.090 m	0.137 m	0.089 m	0.083 m	0.140 m	< 100 ms
VIRAL SLAM (2 LiDARs + IMU) Nguyen et al. (2021a)	**0.060** m	**0.058** m	**0.037** m	**0.051** m	0.043 m	**0.032** m	**0.048** m	**0.062** m	**0.054** m	< 100 ms
KISS-ICP (1 LiDAR) Vizzo et al. (2023)	2.383 m	1.586 m	1.055 m	0.359 m	×	1.389 m	1.353 m	1.435 m	1.037 m	31.0 ms
MOLA-LO (ours) (1 LiDAR)	1.484 m	1.639 m	1.046 m	0.746 m	1.256 m	0.424 m	0.361 m	1.015 m	1.066 m	31.6 ms
MOLA-LO (ours) (2 LiDARs)	0.780 m	1.574 m	0.725 m	0.755 m	0.595 m	0.427 m	0.344 m	1.000 m	0.914 m	40.0 ms
MOLA-LO (ours) (2 LiDARs +2 maps)	0.100 m	0.362 m	0.175 m	0.220 m	0.220 m	0.121 m	0.118 m	0.123 m	0.115 m	69.4 ms



(a) 3D trajectory (b) Trajectory components

Figure 20. Estimated trajectories for the proposed LiDAR odometry system ("MOLA-LO") applied to the RPG drone testing arena sequence of drone testing area **HILTI-2021** dataset, compared to ground truth.

Table 7. Absolute translational error (ATE) (rmse values in meters as reported by "`evo_ape -a`") for the **Voxgraph** dataset. Bold means best accuracy.

Method	Voxgraph (sequence: `t0`)
LOAM Zhang and Singh (2017)	2.64 m
Voxgraph Reijgwart et al. (2019)	0.83 m
KISS-ICP Vizzo et al. (2023)	0.253 m
MOLA-LO (default) (ours)	0.265 m
MOLA-LO (3D-NDT) (ours)	**0.250** m

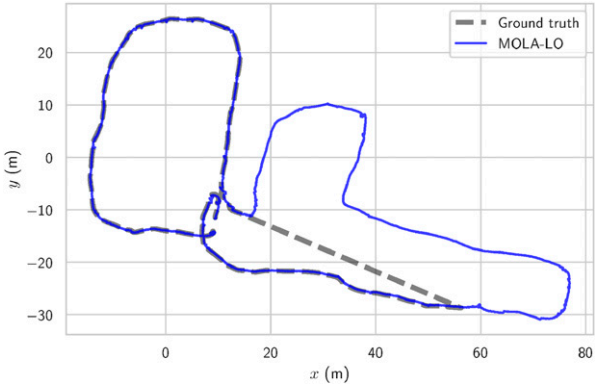


Figure 21. Estimated trajectory by our LO system for the **VoxGraph** dataset, together with the (partial) ground truth. See discussion in Section 4.7.

dataset comprises 3D LiDAR, vision, and IMU readings from four legged robots as they faced the final event of the challenge. Robots in these sequences alternate periods of inactivity with walking as they autonomously explore a series of cave-like scenarios. ANYmal C legged robots were used, equipped with one Velodyne VLP-16 each. We only used LiDAR scans from these dataset, which contains four sequences, one per robot, and ground truth trajectories extracted by scan matching against a ground truth environment point cloud from survey-quality scanners.

Since loop closures are not relevant in these scenarios, only LO methods were evaluated. Two SOTA representative methods, KISS-ICP (Vizzo et al., 2023) and SiMpLE (Bhandari et al., 2024), have been run side by side with our system by using MOLA wrapper modules to ensure identical conditions for input data. KISS-ICP was run with its default parameters, just like system, run with its default configuration and pipelines (Section 3.7). In turn, for SiMpLE we used its slowest and most accurate configuration ("offline"), already evaluted for KITTI in Section 4.2.

The quantitative results are summarized in Table 8. KISS-ICP diverges in all sequences. SiMpLE diverges in the `anymal_4` sequence, achieving good accuracy in the others. Our system does not diverge in any of the sequences, performs faster than SiMpLE, and achieves better ATE trajectory metrics in 3 out of 4 sequences, demonstrating its robustness and viability for non-automotive, unstructured scenarios. As can be seen in the 3D trajectory views and component analysis in Figure 22, our method reconstructs accurate trajectories, with the only weak point worth mentioning being the drift accumulated in the vertical component (z axis), especially for sequences `anymal_1` and `anymal_4`; in fact, the largest part of RMSE ATE for those sequences comes from such z component. Smarter selection of down-sampled points would probably alleviate this drift, a topic that shall be studied in future works.

4.9. Newer college dataset

This dataset consists of several sequences acquired in New College (Oxford) with a handheld device comprising a 3D LiDAR, a high-frequency IMU, and cameras, with the intention of becoming a benchmark for visual-inertial odometry and LiDAR-inertial odometry techniques. The first two sequences (`01` and `02`) used an Ouster OS0-64 and were presented in Ramezani et al. (2020), with walks covering the college Great Quad, Garden Quad, and the Bowling Green, long enough to justify the use of a full SLAM solution with loop closures. Later on, 12 additional sequences using an Ouster OS1-128 sensor were presented in Zhang et al. (2021), with more diverse scenarios and difficulty levels, organized by intentionally-introduced high dynamic translations and rotations to create challenging conditions.

Table 8. Absolute translational error (ATE) (rmse values in meters as reported by "`evo_ape -a`") for the **DARPA Subterranean Final Event** dataset. Bold means best accuracy in each sequence. Divergence is shown as ×.

Method	anymal_1	anymal_2	anymal_3	anymal_4	Avr. time
KISS-ICP Vizzo et al. (2023)	×(88.57 m)	×(4918.14 m)	×(2148.04 m)	×(22.45 m)	1.4 ms
SiMpLE (offline) Bhandari et al. (2024)	**0.44** m	0.48 m	0.28 m	×(18.46 m)	54.2 ms
MOLA-LO (ours)	2.80 m	**0.24** m	**0.16** m	**3.15** m	13.1 ms

Our LO method has been evaluated against the SOTA method KISS-ICP (Vizzo et al., 2023), with quantitative results summarized in Table 9, where it can be seen that our system has a better accuracy in 11 out of the 14 sequences, 8 of them with ATE reductions larger than 50%. Moreover, KISS-ICP diverges in 3 of the hardest sequences ("Maths Hard" (MH), "Underground Hard" (UH), "Stairs" (ST)) while MOLA-LO achieves ATE better than 0.12 m in all of them. It is worth mentioning that none of the two benchmarked LO methods make use of the IMU data. Other works, like Pfreundschuh et al. (2024), have recently reported even better accuracy in novel methods exploiting the intensity channel of LiDAR points together with inertial measurements, but for the sake of fair benchmarking we limit the present comparison to methods relying solely on 3D points.

Trajectories obtained by our LO method, and by the loop-closure post-processing stage, are illustrated in Figures 23 and 24, together with ground truth. In order to understand why the sequences labeled as "hard" are harder to cope with, we show the translational (x, y, z) and orientation (yaw, pitch, roll) components of the trajectory estimated for one such sequence ("Maths hard"), along with ground truth, in Figure 25. Observe how, in time steps corresponding to $t \approx 140$ s, $t \approx 180$ s, and $t \approx 225$ s, the sensor is rotated at a high speed in different combinations of orientations and translations. The introduction of the joint estimation of the velocity vector within the ICP loop itself (see Section 3.6.3) has revealed essential to cope with these sudden disruptions of the constant velocity assumption.

4.10. UAL VLP-16 campus dataset

Finally, the proposed LO system, the full SLAM system including loop closure, and the SOTA method KISS-ICP (Vizzo et al., 2023) have been benchmarked in the automotive dataset reported in Blanco-Claraco et al. (2019), where an electric vehicle is driven around the University of Almeria campus while grabbing scans from a Velodyne VLP-16. Ground truth trajectory is available from RTK GNSS 3D positioning. Estimated trajectories are shown in Figure 26, where it can be seen that, as already explained above, the largest part of the obtained ATE values is caused by accumulation of errors in the vehicle altitude above the initial ground level. Such error is drastically reduced by handling loop closures, as can be seen in Figure 26(b). Regarding a quantitative analysis of errors, they are available in Table 10, with our system exhibiting $\sim 30\%$ less ATE than the baseline method (KISS-ICP), while still being able to run LO ~ 6 times faster than the sensor rate.

5. Qualitative demonstrations

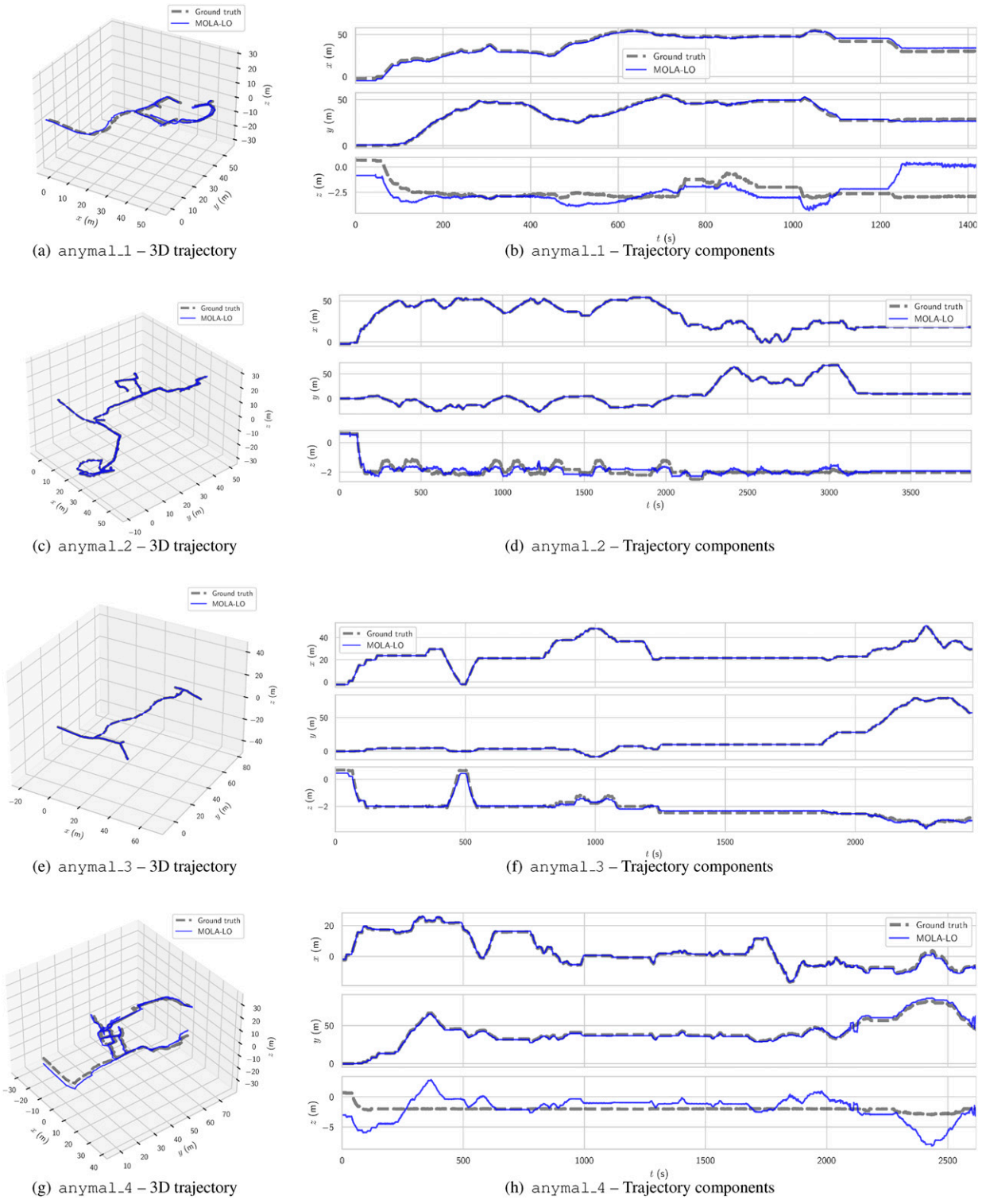
After experimentally measuring the accuracy of our proposal for several public 3D LiDAR datasets with known ground truth, we now provide evidence of other qualitative features illustrating different claims of this work. Further claims will be also analyzed later on in a *quantitative* way in Section 7 with ablation studies.

5.1. 2D LiDAR configuration

The flexibility of our framework to cope with different sensors is here illustrated with an alternative configuration, dubbed `lidar-2d`, suitable for mapping with 2D range finders. While all experiments shown so far comprise building a 3D point cloud local map from 3D scans, just two configuration changes enable the architecture in Figure 4 to build maps from 2D sensors. First, the local map generator (block #5 in Figure 4) is set to create an occupancy voxel map. A plain 2D grid map would also work but, as discussed in Section 3.2, the sparse data structure of voxel maps avoids the need for frequent memory reallocations as the robot explores. Second, the observation pipeline (block #3 in Figure 4) is simplified to that shown in Figure 27, that is, there is no need to down-sample the raw sensor points twice (for registration and for map update) due to the reduced size of 2D scans.

This alternative configuration has been validated with the datasets exposed below, all of them from wheeled robots equipped with encoders from which incremental odometry is taken as an input for the kinematic state prediction module (Section 3.8). First, we applied the LO system to the Freiburg building 079 (`fr079`) dataset (Howard and Roy, 2003), recorded by Cyrill Stachniss in 2010 with a Pioneer2 mobile robot equipped with a SICK LMS range finder. The resulting voxel map (which is only populated at one fixed height) is illustrated in Figure 28(a). Our method took an average of 10.2 ms to process each scan. Note that loop closures were not required here.

Second, we have applied the SLAM solution (LO plus loop-closure detection) to the larger Málaga CS faculty dataset, comprising a 1.9 km trajectory of a robotic



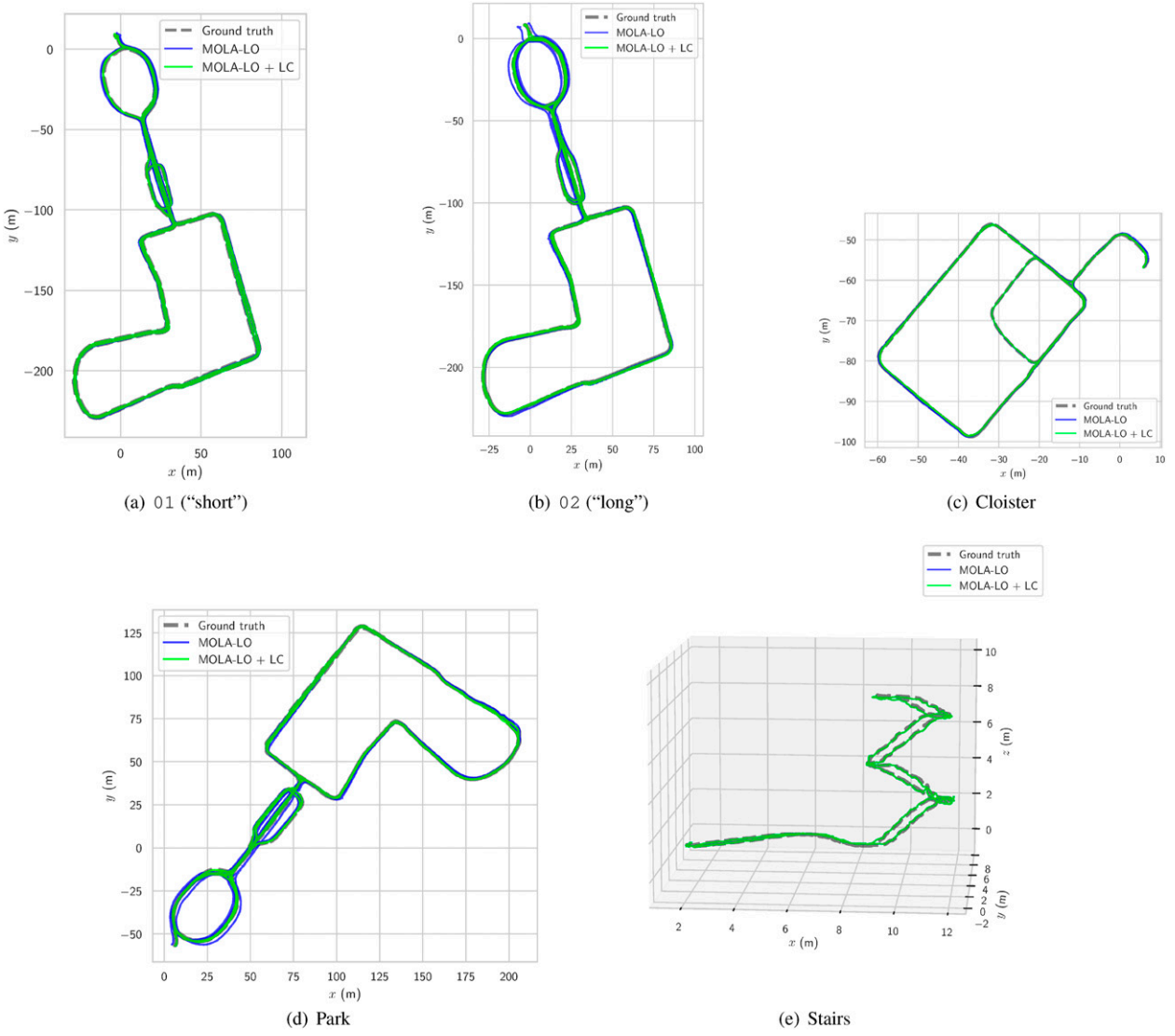
(a) anymal_1 – 3D trajectory
(b) anymal_1 – Trajectory components
(c) anymal_2 – 3D trajectory
(d) anymal_2 – Trajectory components
(e) anymal_3 – 3D trajectory
(f) anymal_3 – Trajectory components
(g) anymal_4 – 3D trajectory
(h) anymal_4 – Trajectory components

Figure 22. Estimated trajectories for the proposed LiDAR odometry system ("MOLA-LO") applied to the **DARPA Subterranean Final Event** dataset, compared to ground truth. See discussion in Section 4.8.

wheelchair equipped with encoders and a 2D SICK LMS range finder. It was recorded in 2006 and is available for download in Blanco-Claraco (2024). The resulting consistent global map is shown in Figure 28(b). As with the former dataset, there is no ground truth for quantitative quality assessment. Our system takes an average of 68.8 ms per scan, including loop closure detection and global optimization.

Table 9. Absolute translational error (ATE) (rmse values in meters as reported by "evo_ape -a") for the **NewerCollege dataset**. Bold means best accuracy in each sequence and category. Divergence is shown as ×. The version of MOLA-LO that always updates the local map is discussed in Section 7.2.

Method	01	02	Cloister	MathsEasy	MathsMedium	MathsHard	Park	QuadEasy	QuadMedium	QuadHard	Undergr.Easy	Undergr.Medium	Undergr.Hard	Stairs
KISS-ICP Vizzo et al. (2023)	**0.61** m	1.78 m	0.95 m	0.07 m	**0.12** m	×(29.6 m)	1.54 m	0.10 m	0.19 m	0.65 m	0.26 m	0.45 m	×(7.98 m)	×(3407 m)
MOLA-LO (ours)	0.68 m	**0.54** m	**0.12** m	**0.06** m	**0.12** m	**0.11** m	**0.90** m	**0.08** m	**0.08** m	**0.12** m	**0.07** m	**0.08** m	**0.09** m	**0.11** m
MOLA-LO (always updateslocal map)	×(5.11 m)	×(84.37 m)	0.39 m	0.09 m	×(4.93 m)	0.78 m	1.38 m	0.08 m	2.99 m	0.10 m	0.08 m	0.46 m	×(8.06 m)	0.34 m
MOLA-LO + LC (ours)	0.31 m	0.40 m	0.19 m	0.09 m	0.13 m	0.31 m	0.31 m	0.09 m	0.12 m	0.16 m	0.13 m	0.13 m	0.25 m	0.42 m



(a) 01 ("short") (b) 02 ("long") (c) Cloister

(d) Park (e) Stairs

Figure 23. Estimated trajectories for the proposed LiDAR odometry system ("MOLA-LO") applied to the **NewerCollege dataset**, compared to ground truth. Trajectories denoted as "MOLA-LO + LC" include the post-processing loop-closure stage discussed in Section 3.12. Continues in Figure 24. See discussion in Section 4.9.

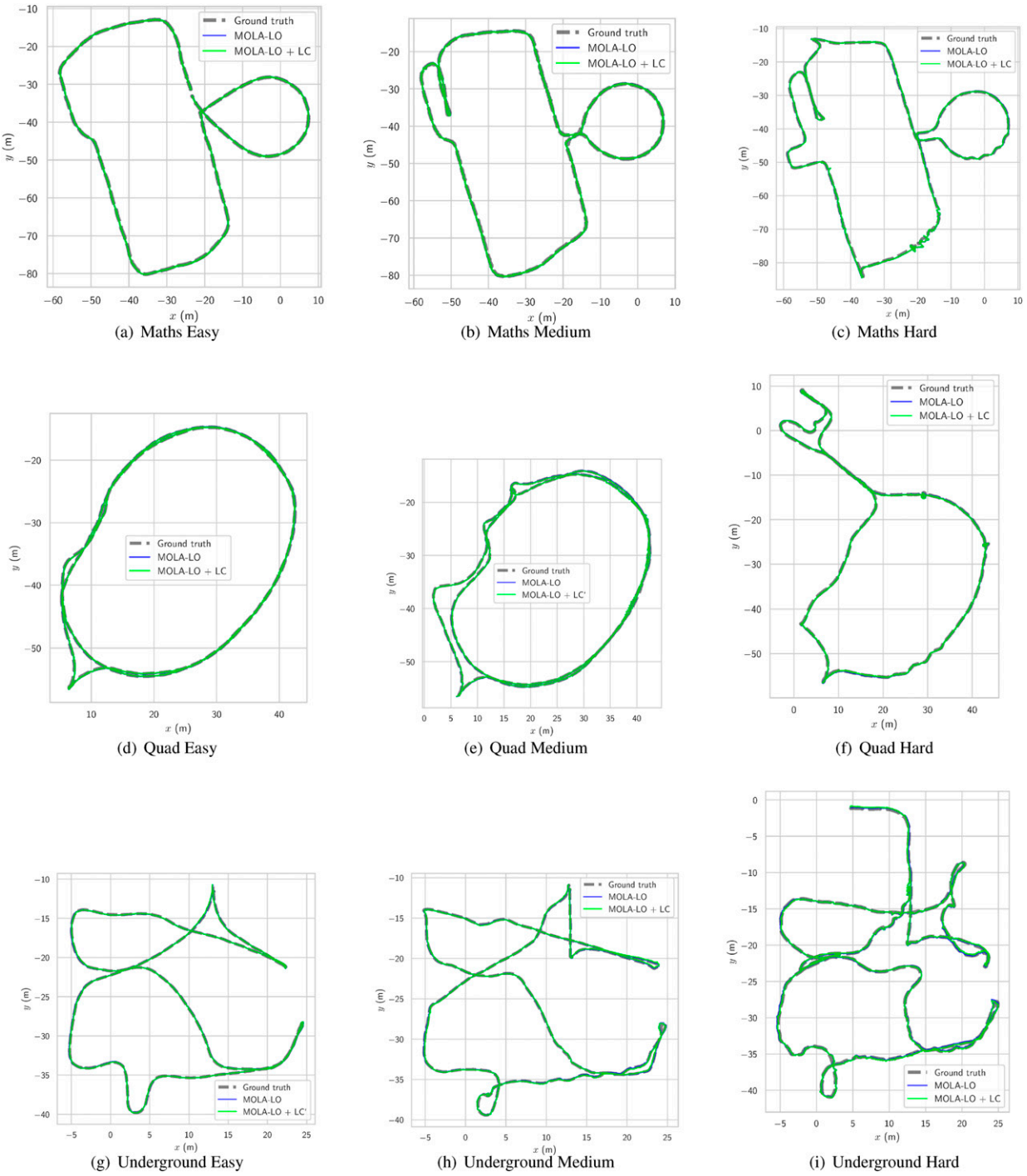


Figure 24. Continuation of Figure 23 with more results for the **Newer College dataset**. See discussion in Section 4.9.

5.2. Georeferencing metric maps

Finding the geodetic coordinates of metric maps is possible from a low-cost GNSS receiver given observations enough, following the method described in Section 3.11. In this section, we show experimental results in georeferencing maps, based on the MulRan dataset (see Section 4.1) since it includes both, accurate ground truth, and readings from a low-cost GNSS device.

To illustrate georeferenciation, we took estimated trajectories in the local map frame, convert them to the ENU frame (refer to Figure 12(a)), then to Earth geocentric coordinates, and finally to geodetic datums for the WGS84 standard geoid. The estimated paths for a subset of four MulRan dataset sequences were exported to KML (Keyhole Markup Language) and displayed on Google Earth for visual inspection. Figure 29 shows the final results, automatically generated by tools in the presented framework.

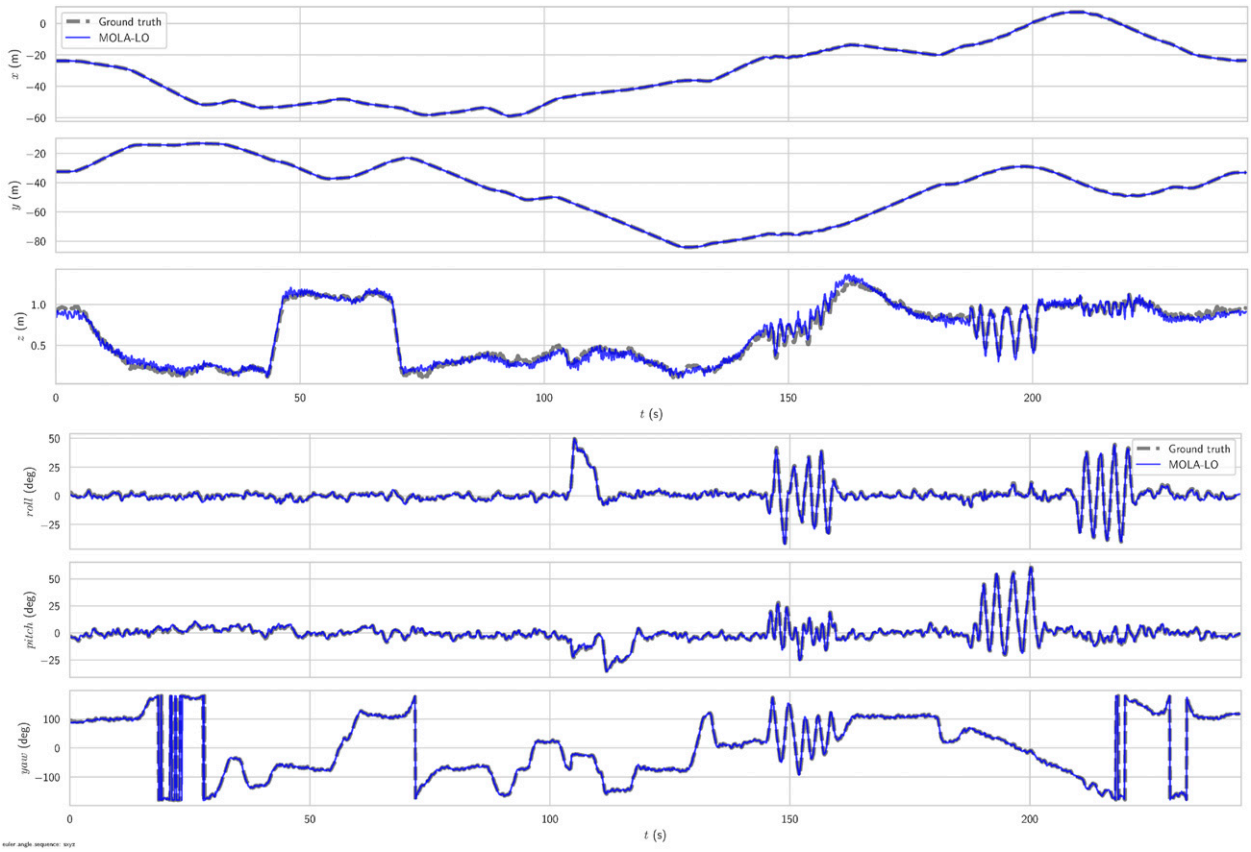


Figure 25. Detailed view of the maths hard (MH) sequence (**Newer College dataset**) in Figure 24. Observe the highly dynamic rotations imposed to the sensor in all six degrees of freedom with the intention of making this sequence a challenge for LO methods, and how our method keeps its estimation close to ground truth at all times. See discussion in Section 4.9.

5.3. Use as a backpack mapping system

As detailed in Aguilar et al. (2024), the proposed LO system has been used for mapping forest areas from datasets grabbed with a backpack kit comprising an Ouster OS0-32 as the unique sensor. Note that the system configuration was not modified with respect to all tested datasets in Section 4 and, although these experiments did not have ground truth for quantitative assessment, the final metric maps are consistent, as illustrated in Figure 30 with views of 2 out of the 6 tested sequences. This experiment shows that the LO system is able to cope with walking patterns in uneven terrains while mapping unstructured, natural environments.

5.4. Building different metric maps

One of the claims of this work is that different robotic tasks are better performed using different map representations, thus the mapping framework should be able to provide such flexibility. To illustrate this feature, we show next how easy is to create metric maps of different types, apart of the commonly-used point clouds. Focusing on the post-processing stage, once mapping is done and there is a view-based map from the LO or SLAM system, it was explained in Section 3.10 how such map can be converted into metric maps. In particular, Figure 9 illustrated how to build several map layers by accumulating 3D LiDAR scans into a global map.

Here, we use a modification of such pipeline, shown in Figure 31, where only one final map layer exists of a custom type which will be changed between experiments. The final map obtained from the Voxgraph dataset (Section 4.7) was fed into that pipeline with two map types: an occupancy voxel map, and a 2D digital elevation model (DEM). The resulting maps are illustrated in Figure 32(a) and (b). Changing a few lines in a configuration YAML file is all that it takes to build one map type or the other.

6. Localization experiments

Localization without mapping can be performed in the present framework in two ways: (i) with particle filtering (as in Blanco-Claraco et al. (2019)) using the `mrpt_pf_localization` software package, or (ii) directly with the LO pipeline introduced in Section 3.4, by disabling map updates. In both cases, the reference map can be given from converting a view-based map from a mapping session into a metric map, as described in Section 3.10. Robots moving slowly and equipped with wheels encoders may be good fits for the particle filtering approach. For

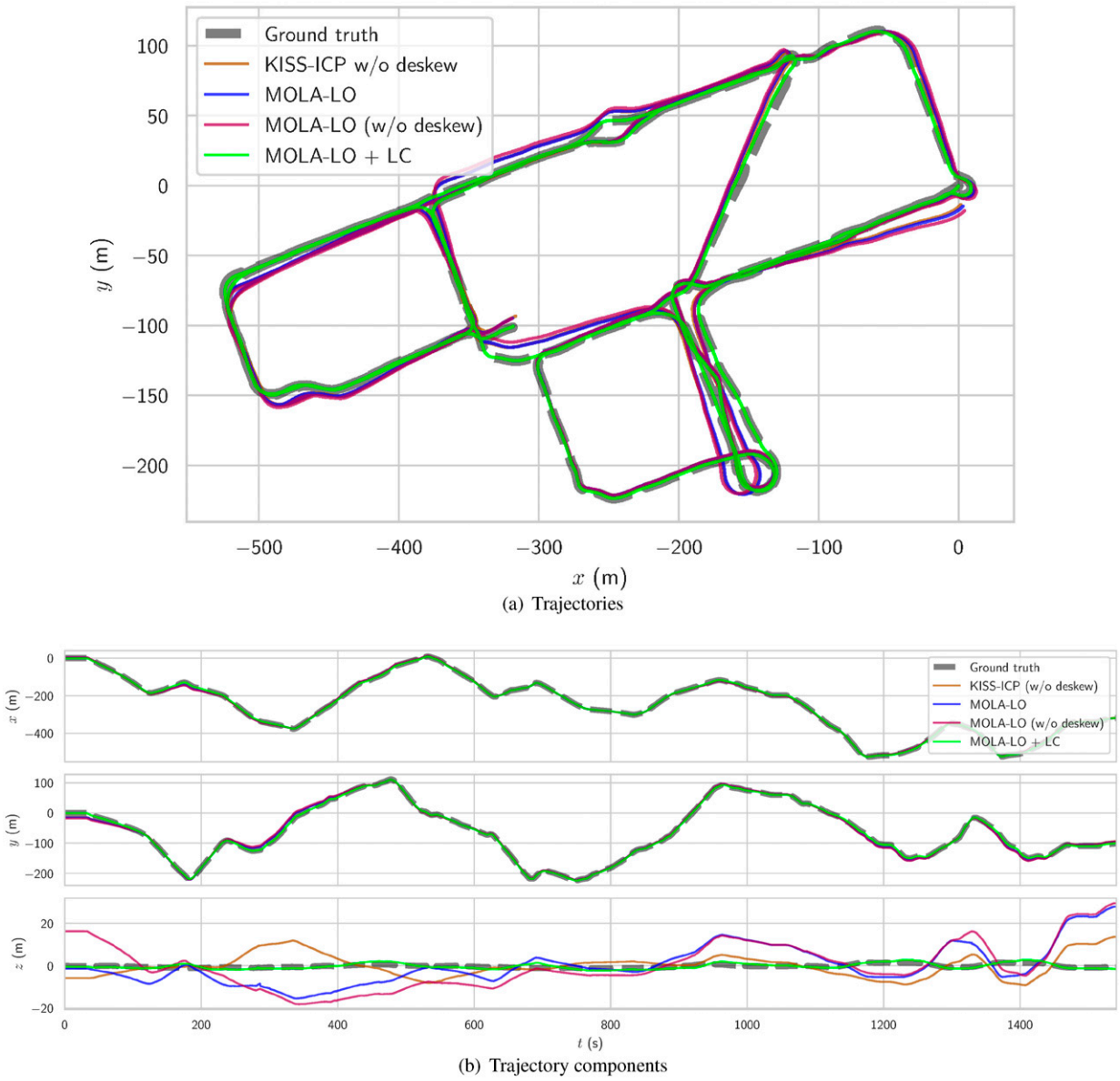


Figure 26. Estimated trajectories for the proposed LiDAR odometry system ("MOLA-LO") applied to the **UAL VLP-16 Campus** dataset, compared to ground truth. Trajectories denoted as "MOLA-LO + LC" include the post-processing loop-closure. See discussion in Section 4.10.

Table 10. Absolute translational error (ATE) (rmse values in meters as reported by "`evo_ape -a`") for the **UAL Campus** dataset, in which there is only one sequence. Bold means best accuracy in each sequence and category.

Method	UAL campus	Avr. timeper frame
KISS-ICP (w/o deskew) Vizzo et al. (2023)	13.21 m	12.0 ms
MOLA-LO (w/o deskew) (ours)	**7.97** m	14.7 ms
KISS-ICP Vizzo et al. (2023)	10.06 m	11.2 ms
MOLA-LO (ours)	**6.65** m	14.5 ms
MOLA-LO + LC (ours)	**1.00** m	21.3 ms

faster vehicles and drones, the LO pipeline is probably a better option due to the more accurate initial predictions for each time step given by the kinematic state prediction module (Section 3.4).

In this section, we provide experimental results for localization using the MulRan dataset, which perfectly fits this task since three driving sequences exist for each location. Therefore, a metric map from the SLAM output for one such sequence (`KAIST02`) is built and used as reference map for localizing the other two sequences of the same location (`KAIST01` and `KAIST03`). The reference global map was built with the metric map building pipeline in Figure 8, using a point cloud as global map. In particular, from all point cloud maps described in Section 3.2, we selected the

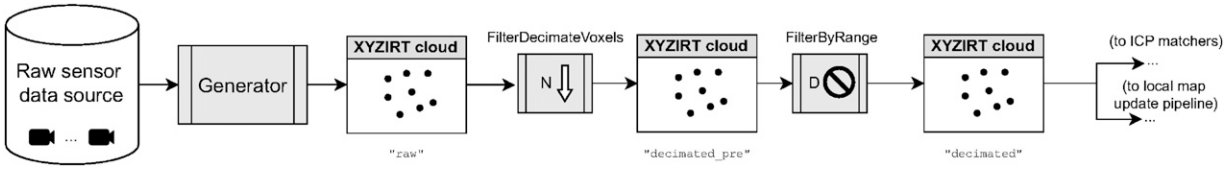
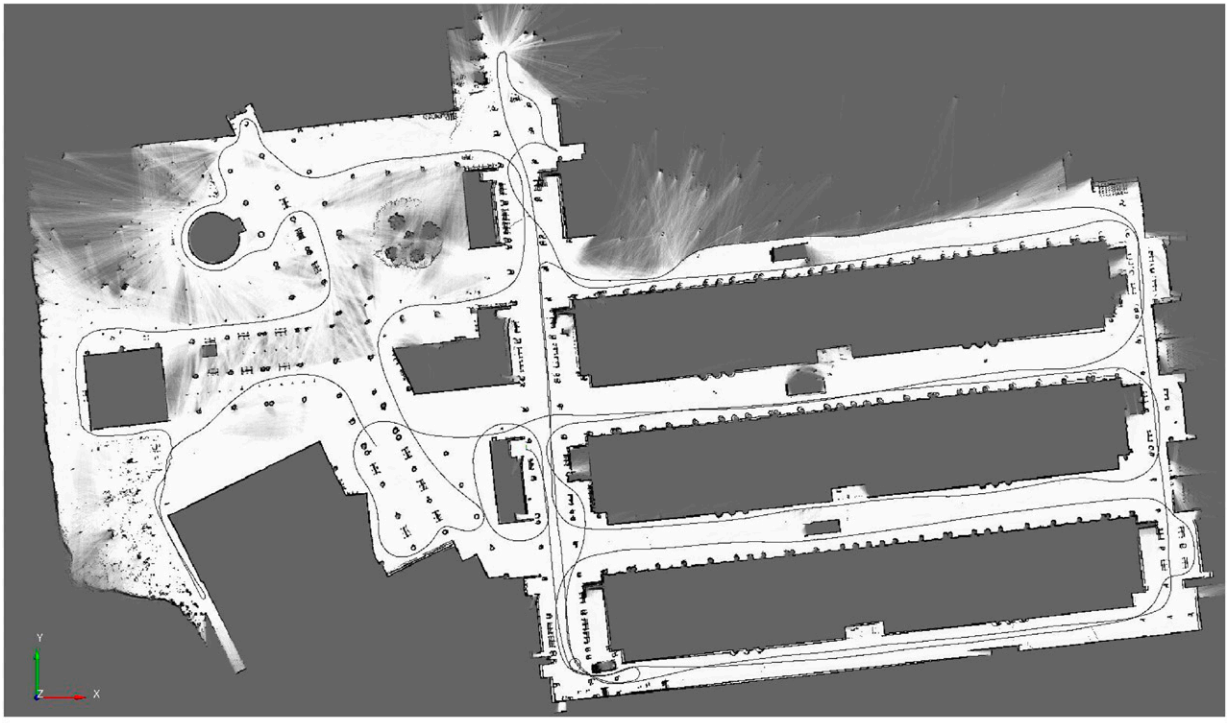


Figure 27. Contents of the "observation pipeline" blocks in Figure 4 for the "2D LiDAR" LO system configuration. Refer to discussion on pipelines in general in Section 3.3 and to Section 5.1 for this particular pipeline.

(a) `fr079` final map

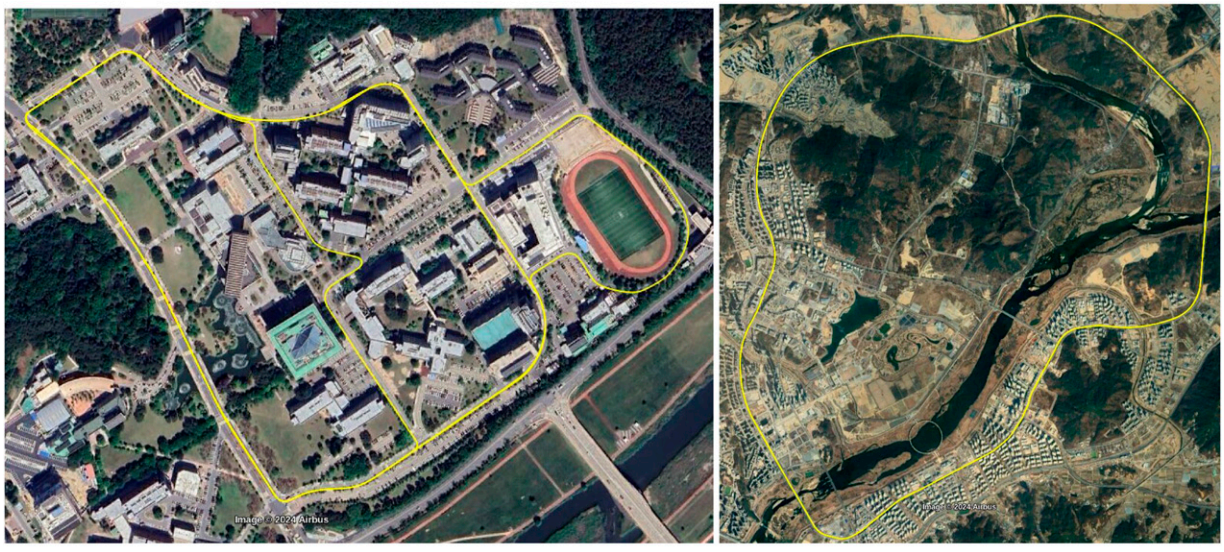


(b) Málaga CS faculty final map

Figure 28. Results for the 2D LiDAR configuration. Refer to discussion in Section 5.1.

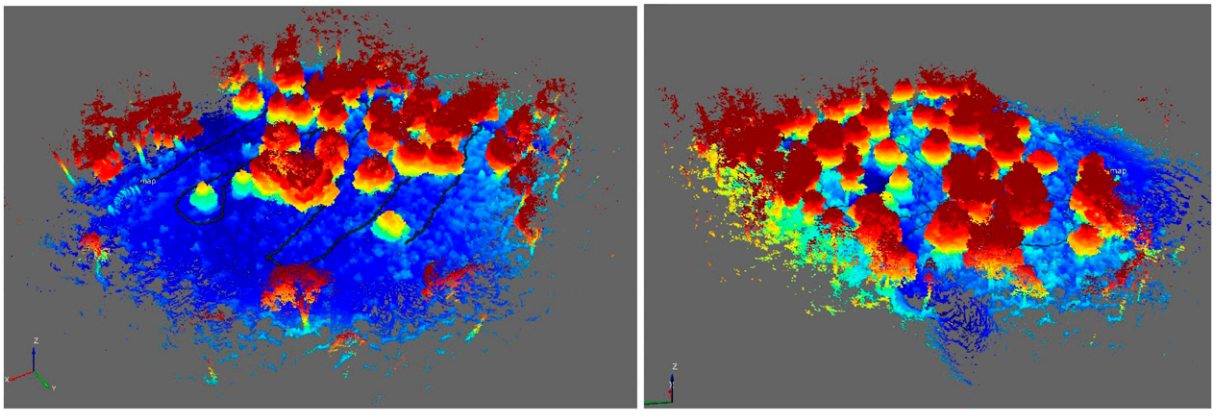
one with hashed voxels as underlying data structure, with a 2 m resolution and a maximum of 20 points per voxel. Localization errors have been estimated using `evo_ape` (Grupp, 2017) between the estimated trajectories and ground truth, as illustrated in Figure 33. In this case, the initial approximated pose was given manually for each sequence, with an error of several meters, hence the error peak at the beginning of each sequence. Automatically

(a) DCC01 (b) Riverside01



(c) KAIST03 (d) Sejong02

Figure 29. Illustrative results for metric map georeferenciation for sequences estimated by the SLAM solution and automatically georeferenced as described in Section 5.2 from consumer-grade GNSS data (Screenshot from Google Earth v7.3.6.9326, images by: Maxar Technologies, 2024; Airbus, 2024).



(a) Forest parcel #2 (b) Forest parcel #3

Figure 30. Final 3D maps from datasets collected in Almería forests with a LiDAR backpack. See discussion in Section 5.3.

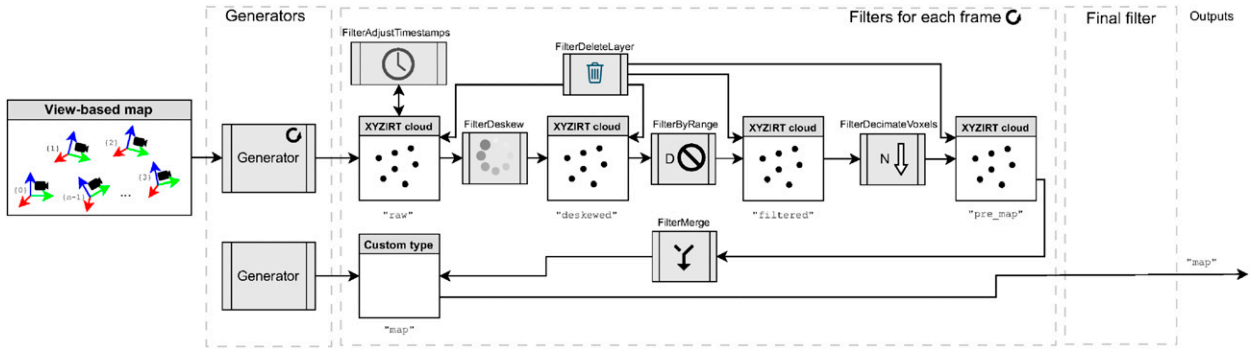
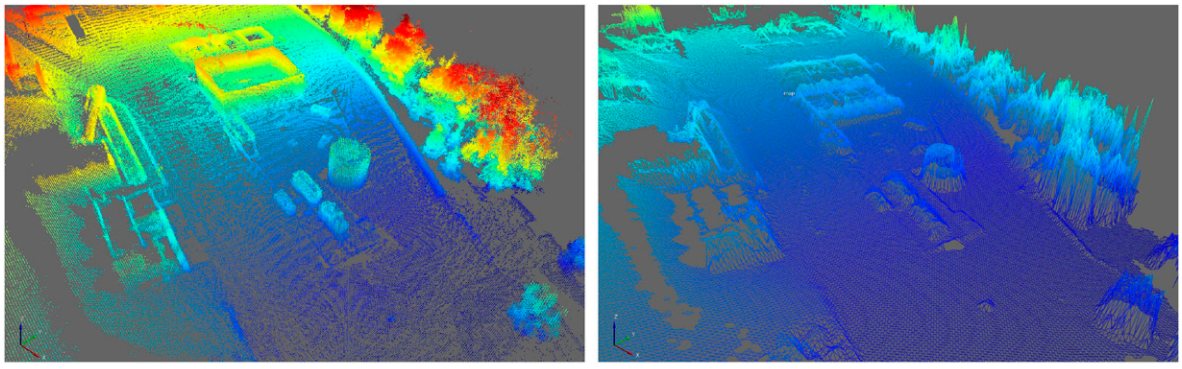


Figure 31. Configuration for building metric maps of arbitrary types from view-based maps using the application sm2mm. The type of the final map is solely determined by the "Generator" block on the bottom, without any other required change in the rest. Refer to discussion in Section 5.4.



(a) Occupancy voxel map (b) Digital terrain elevation model

Figure 32. Example final maps built from the Voxgraph dataset. The voxel map in (a) has been visualized as one point per voxel to favor a clearer visualization. Refer to discussion in Section 5.4.

re-localizing from GNSS readings is possible as a more practical alternative, but at the time of writing, this feature is only implemented in the particle filter-based solution. The localization RMSE with respect to ground truth in these two experiments were 2.21 m and 2.11 m, respectively. Note, however, that virtually all this error is attributable to accumulated residual drift in the map used as reference, since 2.13 m is precisely the RMSE ATE of the full SLAM solution for sequence `KAIST02` (refer to row "MOLA-LO + LC" in Table 2). Basically, localization accuracy is limited by how accurate the reference map is. Note that the tested sequences include dynamic objects (e.g., vehicle overtaking), starts and stops due to traffic, etc.

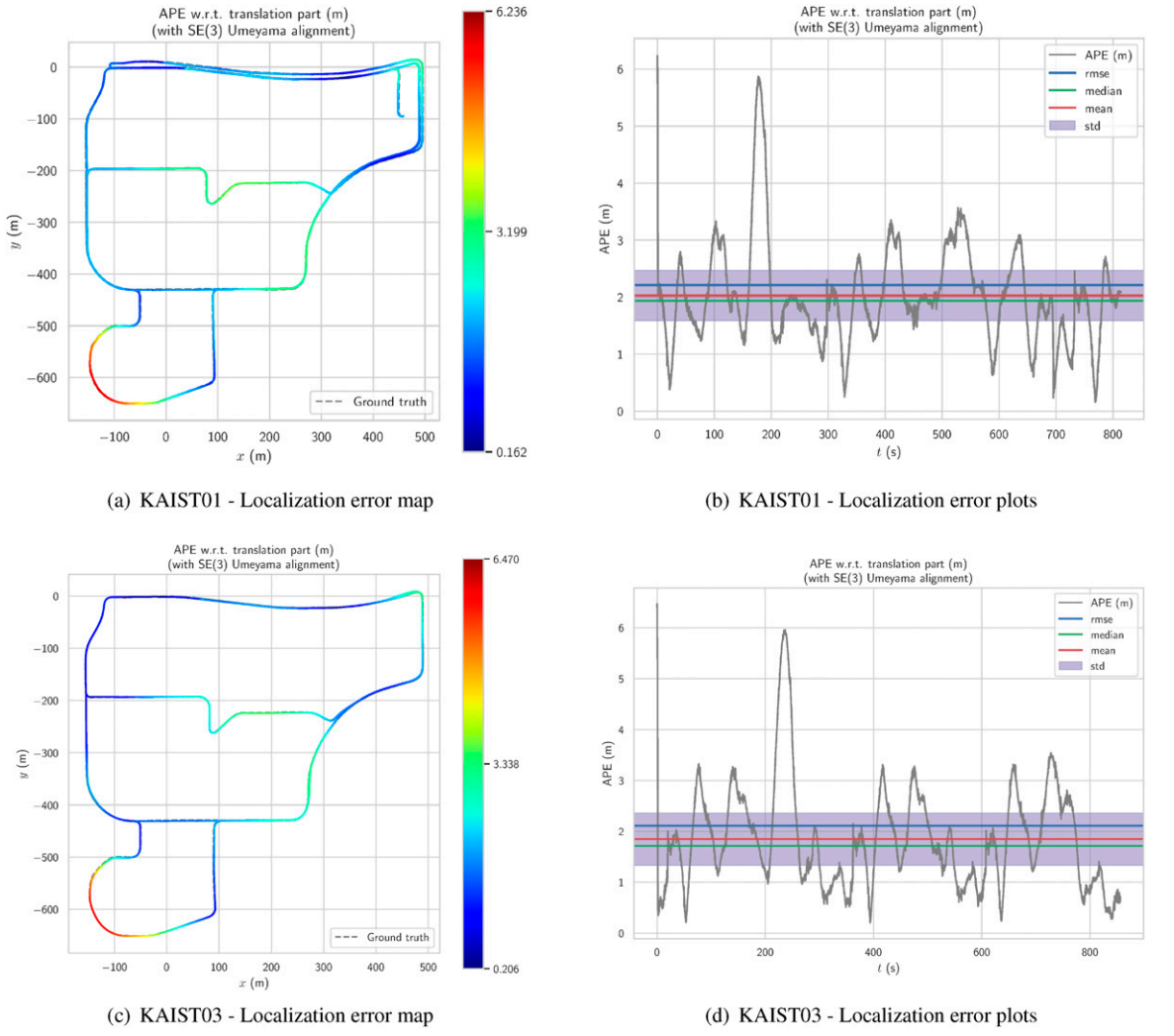
7. Ablation studies

7.1. Scan deskewing

First, we want to validate that scan deskewing by means of trajectory interpolation on SE(3) per equation (2) is good enough and effectively leads to more accurate results. As a benchmark, we tested our LiDAR odometry system (without loop closure) on the UAL campus dataset in Blanco-Claraco et al. (2019) with and without deskewing the input scans, obtaining the results in Table 10. The obtained ATE values with respect to ground truth with and without deskewing are 6.65 m and 7.97 m, respectively. Therefore, undistorting the scans reduced the ATE in a 16.6%, despite the fact that driving speed was relatively slow in this dataset, an average of $\sim 3\ m/s$ (10.8 km/h or 6.7 mph). For comparison, enabling deskewing in KISS-ICP also reduced the ATE from 13.21 m to 10.06 m, a 23.8% improvement. Therefore, we can conclude that deskewing is absolutely a must for accurate localization, even at reduced speeds.

7.2. Local map updates

As mentioned in Section 3.4, our system differs from others in that we do not always update the local map for every single time step. The reason behind this decision is that small localization errors and imperfect scan de-skewing, specially during abrupt angular accelerations would lead to map insertion of misplaced points which would then serve as (incorrect) matches for successive scans, typically leading to unbounded error growth in localization. We postulate that such misplaced points are more likely in high-dynamics datasets than in those with smoother trajectories; e.g. handheld versus automotive. This effect is expected when input scans are subsampled before registration against



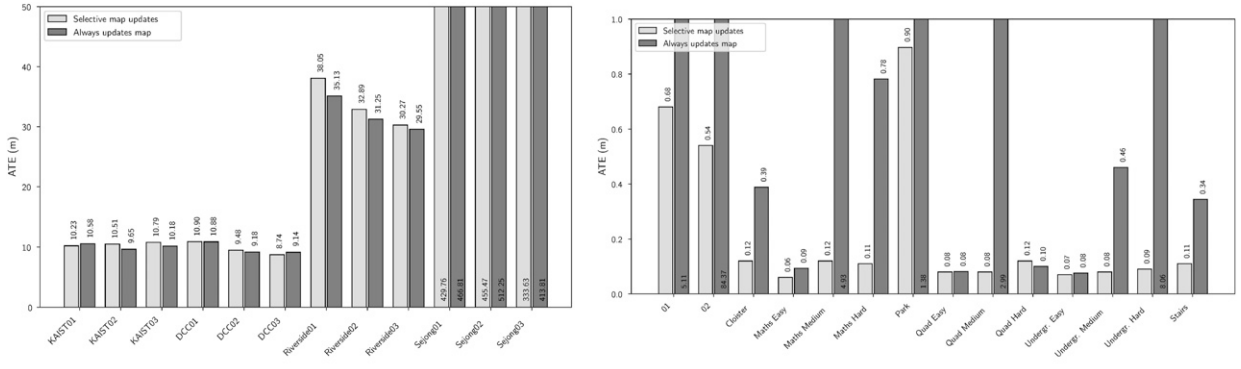
(a) KAIST01 - Localization error map

(b) KAIST01 - Localization error plots

(c) KAIST03 - Localization error map

(d) KAIST03 - Localization error plots

Figure 33. Results of the **localization** experiments performed with **MulRan dataset** sequences KAIST01 and KAIST03 by using as reference the map built from KAIST02. Refer to discussion in Section 6.



(a) Mulran (automotive)

(b) Newer College Dataset (handheld)

Figure 34. Results of the ablation experiments to evaluate the proposed selective map update feature, showing how it becomes relevant for datasets with irregular motion profiles (e.g., handheld sensors). Refer to discussion in Section 7.2.

a local metric maps that do not fuse information from different scans, such as the point clouds used in our default system configuration and in many recent LiDAR odometry approaches. Probabilistic map representations or the 3D-NDT map described in Section 3.2 are expected to be more stable against frequent updates.

To support our hypothesis, we performed the following ablation experiment: we compared the trajectory error with and without the proposed selective map update in an automotive (Mulran) and a handheld (Newer College) dataset. The quantitative results were already reported in Tables 2 and 9, respectively, but for the sake of a clearer visual comparison we have represented them in Figure 34. As can be observed, updating the map for every single time step is not problematic for the automotive dataset, with trajectory errors slightly better in some sequences and worse on others, but not leading to drastic differences. In turn, not using our proposed selective map update strategy for the Newer College dataset leads to a dramatic increase of (at least) 50% for the ATE in 11 out of 14 sequences, hence supporting the introduction of our selective update strategy for robustness against high-dynamic motion profiles.

7.3. Tightly-coupled estimation of vehicle velocities

Next, we want to evaluate the impact of the tightly-coupled estimation of linear and angular velocities within the optimization loop (refer to Section 3.6.3). As baseline, we use the proposal of keeping the velocity vectors as estimated from the last time step, as done in the state-of-the-art method KISS-ICP Vizzo et al. (2023). Comparison will be done using two Newer College Dataset sequences with high dynamics: "Maths hard" (MH) and "Underground hard" (UH). The trajectory ATE for MH increases from 0.11 m using the default system configuration to 0.15 m if the feature at test is disabled. A more drastic increase of the error occurs for the UH sequence, where an ATE of 0.086 m becomes 2.79 m. By visualizing the estimated paths in detail, shown in Figure 35, it can be seen that the error comes from two sources: divergence of the estimation at multiple points, and jerky trajectories for the rest. These experiments demonstrate that this feature alone has a great potential to increase the stability of LiDAR-only odometry methods.

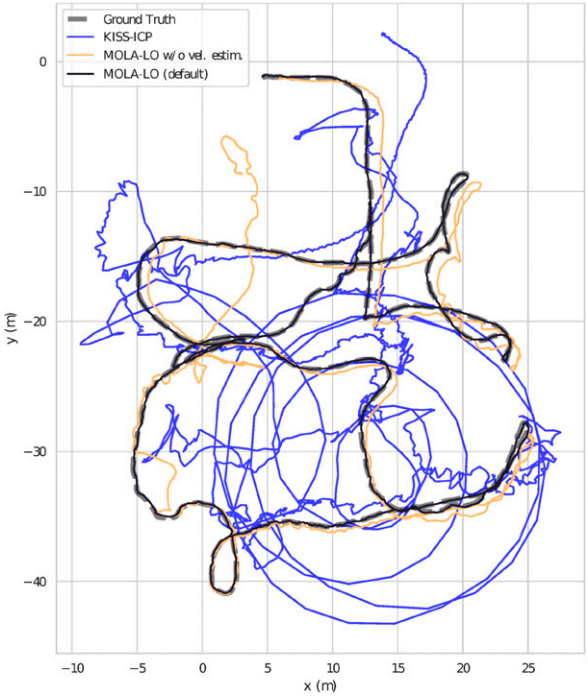


Figure 35. Evaluation of the impact of the tightly-coupled estimation of velocity vectors as part of the ablation study in Section 7.3 for the "Underground hard" sequence of the Newer College Dataset. The picture shows a bird-eye view of the estimated trajectories. In the legend, "MOLA-LO" refers to the default system as introduced in the manuscript, while "MOLA-LO without velocity estimation" is a version where tightly-coupled estimation of this vector is disabled.

7.4. Horn's method

As mentioned in Section 3.6.2, the closed-form Horn's solution to pointwise pairings would seem to be an ideal alternative to iterative methods such as Gauss–Newton due to its simplicity and efficiency. However, as we demonstrate in this section, there are two problems that render iterative nonlinear methods as better practical solutions. We have compared the accuracy of our default LiDAR odometry system with another version where the Gauss–Newton optimizer is replaced with Horn's closed form solution (a change that only takes adding three lines in a YAML configuration file). The dataset on which the study has been done is KITTI, with results already shown in Table 3 in the rows "MOLA-LO (default)" and "MOLA-LO (Horn's)." In all sequences, excepting `01` and `04`, the Horn's method leads to worse results. Overall, the average RTE worsens from 0.55% to 0.65%. The reason for these results is the impossibility for this method to cope with outliers, something easy to integrate in nonlinear optimizers. Another interesting observation is that, despite the solver itself runs ~ 4 times faster than Gauss–Newton, the overall time cost is ~ 2 times slower due to the need to spend more ICP iterations until convergence.

7.5. Loop-closure with and without GNSS

Finally, the capability of the proposed loop closure algorithm to incorporate optional GNSS readings is put at test with the Mulran dataset. Results are shown in the two last rows of Table 2. Overall, the conclusion here is that the presence of GNSS: (i) improves the global accuracy of the trajectories (reduced ATE values in the version with GNSS in 11 out of 12 sequences), and (ii) enables identifying potential loop closures that would not be found by our metric uncertainty-based hypothesis generator in the very long loops (> 20 *km*) of Sejong sequences.

8. Conclusions

This paper introduced a whole framework that aims at filling a gap in the robotics community's needs for flexible map building and editing from 3D LiDAR. It has been shown how the proposed LO and SLAM systems compare well or outperform other SOTA methods regarding estimated trajectory accuracy and robustness against divergence. Excepting georeferencing and loop-closure, all other components of the framework are available as open-source software. The present work leaves plenty of research topics open for future works: (i) benchmarking different metric map data structures to find out which ones suit best to each problem (e.g., real-time LO vs localization without mapping), (ii) designing new pipeline blocks for smarter sampling of point clouds to achieve both, faster and more accurate LO, (iii) alternative pipeline designs suitable for efficient depth camera odometry, or (iv) adding support for tightly-integrated measurements from LiDAR intensity images or inertial sensors.

Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.

ORCID iD

Jose Luis Blanco-Claraco https://orcid.org/0000-0002-9745-285X

Notes

1. The actual implementation left-multiplies these transformations with ${}_b\mathbf{T}_s$, the sensor frame "s" pose in the vehicle body frame "b" in order to give estimates of the vehicle trajectory and to enable data fusion from several sensors.
2. Again, the actual implementation takes into account the sensor pose within the vehicle, but that change of coordinates is neglected here for the sake of clarity.
3. Their corresponding YAML file descriptions can be checked out online under demos in the mp2p_icp project.
4. See: https://www.cvlibs.net/datasets/kitti/eval_odometry.php.
5. See: https://www.cvlibs.net/datasets/kitti-360/leaderboard_semantic_slam.php?task=trajectory.

References

Agarwal S, Mierle K and The Ceres Solver Team (2022) Ceres solver. https://github.com/ceres-solver/ceres-solver.

Aguilar FJ, Blanco JL, Nemmaoui A, et al. (2024) Preliminary results of a low-cost portable terrestrial lidar based on icp-slam algorithms application to automatic forest digital inventory. In: 6th Euro-mediterranean conference for environmental integration (EMCEI-2024), Marrakech, Morocco, 15–18 May 2024.

Arshad S and Kim GW (2021) Role of deep learning in loop closure detection for visual and lidar slam: a survey. *Sensors* 21(4): 1243.

Bailey T and Durrant-Whyte H (2006) Simultaneous localization and mapping (SLAM): Part II. *IEEE Robotics and Automation Magazine* 13(3): 108–117.

Besl PJ and McKay ND (1992) Method for registration of 3-D shapes. *Sensor fusion IV: Control Paradigms and Data Structures* 1611: 586–606, Spie.

Bhandari V, Phillips TG and McAree PR (2024) Minimal configuration point cloud odometry and mapping. *The International Journal of Robotics Research* 43(11): 02783649241235325.

Blanco JL (2019) A modular optimization framework for localization and mapping. *Robotics: Science and Systems*. Freiburg im Breisgau, June 22-26, 2019.

Blanco JL, Fernández-Madrigal JA and Gonzalez J (2008) Toward a unified bayesian approach to hybrid metric–topological slam. *IEEE Transactions on Robotics* 24(2): 259–270.

Blanco JL, Moreno FA and Gonzalez J (2009) A collection of outdoor robotic datasets with centimeter-accuracy ground truth. *Autonomous Robots* 27: 327–351.

Blanco-Claraco JL (2021) A tutorial on *SE*(3) transformation parameterizations and on-manifold optimization. arXiv preprint arXiv:2103.15980.

Blanco-Claraco JL (2024) The malaga computer science faculty 2D lidar mobile robot dataset (2006, Jan 21st). *Zenodo*. DOI: 10.1109/TRO.2008.918049. https://zenodo.org/records/10673253.

Blanco-Claraco JL, Moreno-Duenas FA and González-Jiménez J (2014) The málaga urban dataset: high-rate stereo and lidar in a realistic urban scenario. *The International Journal of Robotics Research* 33(2): 207–214.

Blanco-Claraco JL, Mañas-Alvarez F, Torres-Moreno JL, et al. (2019) Benchmarking particle filter algorithms for efficient velodyne-based vehicle localization. *Sensors* 19(14): 3155.

Bogoslavskyi I and Stachniss C (2017) Analyzing the quality of matched 3d point clouds of objects. In: 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS), Vancouver, Canada, 24–28 September 2017. IEEE, 6685–6690.

Bosse M, Newman P, Leonard J, et al. (2004) Simultaneous localization and map building in large-scale cyclic environments using the atlas framework. *The International Journal of Robotics Research* 23(12): 1113–1139.

Bresson G, Alsayed Z, Yu L, et al. (2017) Simultaneous localization and mapping: a survey of current trends in autonomous driving. *IEEE Transactions on Intelligent Vehicles* 2(3): 194–220.

Briales J and Gonzalez-Jimenez J (2017) Convex global 3d registration with Lagrangian duality. In: Proceedings of the IEEE conference on computer vision and pattern recognition, Honolulu, HI, USA, 21–26 July 2017, 4960–4969.

Burri M, Nikolic J, Gohl P, et al. (2016) The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research* 35(10): 1157–1163.

Cadena C, Carlone L, Carrillo H, et al. (2016) Past, present, and future of simultaneous localization and mapping: toward the robust-perception age. *IEEE Transactions on Robotics* 32(6): 1309–1332.

Campos C, Elvira R, Rodríguez JJG, et al. (2021) ORB-SLAM3: an accurate open-source library for visual, visual–inertial, and multimap SLAM. *IEEE Transactions on Robotics* 37(6): 1874–1890.

Carlone L, Tron R, Daniilidis K, et al. (2015) Initialization techniques for 3d slam: a survey on rotation estimation and its use in pose graph optimization. In: 2015 IEEE international conference on robotics and automation (ICRA), Seattle, Washington, 26–30 May 2015. IEEE, 4597–4604.

Chebrolu N, Läbe T, Vysotska O, et al. (2021) Adaptive robust kernels for non-linear least squares problems. *IEEE Robotics and Automation Letters* 6(2): 2240–2247.

Chen X, Milioto A, Palazzolo E, et al. (2019) Suma++: efficient lidar-based semantic SLAM. *IROS*. Macau, China: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 4530–4537.

Cvišić I, Marković I and Petrović I (2022) Soft2: stereo visual odometry for road vehicles based on a point-to-epipolar-line metric. *IEEE Transactions on Robotics* 39(1): 273–288. DOI: 10.1109/TRO.2022.3188121.

Dellaert F (2021) Factor graphs: exploiting structure in robotics. *Annual Review of Control, Robotics, and Autonomous Systems* 4: 141–166.

Dellenbach P, Deschaud JE, Jacquet B, et al. (2022) CT-ICP: real-time elastic LiDAR odometry with loop closure. In: 2022 international conference on robotics and automation (ICRA), Philadelphia, Pennsylvania, USA, 23–27 May 2022. IEEE, 5580–5586.

Deschaud JE (2018) Imls-slam: scan-to-model matching based on 3d data. In: 2018 IEEE international conference on robotics and automation (ICRA), Brisbane, Australia, 21–25 May 2018. IEEE, 2480–2485.

Durrant-Whyte H and Bailey T (2006) Simultaneous localization and mapping: part I. *IEEE Robotics and Automation Magazine* 13(2): 99–110.

Elfes A (1987) Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation* 3(3): 249–265.

Elhousni M and Huang X (2020) A survey on 3D LiDAR localization for autonomous vehicles. In: 2020 IEEE intelligent vehicles symposium (IV), Las Vegas, NV, USA, 19 October – 13 November 2020. IEEE, 1879–1884.

Eustice R, Singh H, Leonard JJ, et al. (2005) Visually navigating the rms titanic with slam information filters. *Robotics: Science and Systems* 2005: 57–64.

Evans C, Ben-Kiki O and döt Net I (2017) YAML Ain't Markup Language (YAML™) version 1.2.

Faconti D (2023) Bonxai. https://github.com/facontidavide/Bonxai.

Fernandez-Cortizas M, Bavle H, Perez-Saura D, et al. (2024) Multi S-Graphs: an efficient real-time distributed semantic-relational collaborative SLAM. *IEEE Robotics and Automation Letters* 9(6): 6004–6011. DOI: 10.1109/LRA.2024.3399997.

Ferrari S, Di Giammarino L, Brizi L, et al. (2024) Mad-icp: it is all about matching data–robust and informed lidar odometry. arXiv preprint arXiv:2405.05828.

Forster C, Carlone L, Dellaert F, et al. (2016) On-manifold preintegration for real-time visual–inertial odometry. *IEEE Transactions on Robotics* 33(1): 1–21.

Geiger A, Lenz P, Stiller C, et al. (2013) Vision meets robotics: the kitti dataset. *The International Journal of Robotics Research* 32(11): 1231–1237.

Grisetti G, Grzonka S, Stachniss C, et al. (2007a) Efficient estimation of accurate maximum likelihood maps in 3d. In: 2007 IEEE/RSJ international conference on intelligent robots and systems, San Diego, California, 29 October – 2 November 2007. IEEE, 3472–3478.

Grisetti G, Stachniss C, Grzonka S, et al. (2007b) A tree parameterization for efficiently computing maximum likelihood maps using gradient descent. *Robotics: Science and Systems* 3: 9.

Grisetti G, Stachniss C and Burgard W (2009) Nonlinear constraint network optimization for efficient map learning. *IEEE Transactions on Intelligent Transportation Systems* 10(3): 428–439.

Grisetti G, Kümmerle R, Stachniss C, et al. (2010) A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine* 2(4): 31–43.

Grisetti G, Kümmerle R and Ni K (2012) Robust optimization of factor graphs by using condensed measurements. In: Proceedings of the 2012 IEEE/RSJ international conference on intelligent robots and systems (IROS), Vilamoura, Algarve [Portugal], 7–12 October 2012, pp. 581–588.

Grupp M (2017) Evo: python package for the evaluation of odometry and slam. https://github.com/MichaelGrupp/evo.

Gupta S, Guadagnino T, Mersch B, et al. (2024) Effectively detecting loop closures using point cloud density maps. In: Proceedings of the IEEE international conference on robotics and automation (ICRA), Yokohama, Japan, 13th – 17th May 2024.

Helmberger M, Morin K, Berner B, et al. (2022) The Hilti SLAM challenge dataset. *IEEE Robotics and Automation Letters* 7(3): 7518–7525.

Hess W, Kohler D, Rapp H, et al. (2016) Real-time loop closure in 2d lidar slam. In: 2016 IEEE international conference on robotics and automation (ICRA), Stockholm, Sweden, 16–21 May 2016. IEEE, 1271–1278.

Horn BK (1987) Closed-form solution of absolute orientation using unit quaternions. *Journal of the Optical Society of America* 4(4): 629–642.

Howard A and Roy N (2003) The robotics data set repository (radish). https://ais.informatik.uni-freiburg.de/slamevaluation/datasets.php.

Huang G (2019) Visual-inertial navigation: a concise review. In: 2019 international conference on robotics and automation (ICRA), Montreal, Canada, 20–24 May 2019. IEEE, 9572–9582.

Kim G and Kim A (2018) Scan context: egocentric spatial descriptor for place recognition within 3D point cloud map. In: Proceedings of the 2018 IEEE/RSJ international conference on intelligent robots and systems (IROS), Madrid, Spain, 1-5 October 2018, pp. 4802–4809.

Kim G, Park YS, Cho Y, et al. (2020) Mulran: multimodal range dataset for urban place recognition. In: Proceedings of the IEEE international conference on robotics and automation (ICRA). Paris, France, 31 May – August 31, 2020, 6246–6253.

Kim G, Choi S and Kim A (2021) Scan context++: structural place recognition robust to rotation and lateral variations in urban environments. *IEEE Transactions on Robotics* 38(3): 1856–1874. Accepted. To appear.

Klein G and Murray D (2007) Parallel tracking and mapping for small ar workspaces. In: 2007 6th IEEE and ACM international symposium on mixed and augmented reality, Nara, Japan, 3–16 November 2007. IEEE, 225–234.

Konolige K, Bowman J, Chen J, et al. (2010) View-based maps. *The International Journal of Robotics Research* 29(8): 941–957.

Kümmerle R, Grisetti G, Strasdat H, et al. (2011) G2O: a general framework for graph optimization. In: 2011 IEEE international conference on robotics and automation, Shanghai, China, 9–13 May 2011. IEEE, 3607–3613.

Labbe M and Michaud F (2014) Online global loop closure detection for large-scale multi-session graph-based slam. In: 2014 IEEE/RSJ international conference on intelligent robots and systems, Chicago, Illinois, USA, 14–18 September 2014. IEEE, 2661–2666.

Labussière M, Laconte J and Pomerleau F (2020) Geometry preserving sampling method based on spectral decomposition for large-scale environments. *Frontiers in Robotics and AI* 7: 572054.

Lang AH, Vora S, Caesar H, et al. (2019) Pointpillars: fast encoders for object detection from point clouds. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, Long Beach, CA, USA, 16-17 June 2019, 12697–12705.

Lee S and Ryu JH (2024) Autonomous vehicle localization without prior high-definition map. *IEEE Transactions on Robotics* 40: 2888–2906. DOI: 10.1109/TRO.2024.3392149.

Lee D, Jung M, Yang W, et al. (2023) Lidar odometry survey: recent advancements and remaining challenges. *Intel Serv Robotics* 17: 95–118. Available at: https://doi.org/10.1007/s11370-024-00515-8.

Leutenegger S, Lynen S, Bosse M, et al. (2015) Keyframe-based visual–inertial odometry using nonlinear optimization. *The International Journal of Robotics Research* 34(3): 314–334.

Liao Y, Xie J and Geiger A (2023) KITTI-360: a novel dataset and benchmarks for urban scene understanding in 2D and 3D. *Pattern Analysis and Machine Intelligence (PAMI)* 45(3): 3292–3310.

Lim H, Hwang S and Myung H (2021) ERASOR: egocentric ratio of pseudo occupancy-based dynamic object removal for static 3D point cloud map building. *IEEE Robotics and Automation Letters* 6(2): 2272–2279.

Lim H, Kim B, Kim D, et al. (2024) Quatro++: robust global registration exploiting ground segmentation for loop closing in LiDAR SLAM. *The International Journal of Robotics Research* 43(5): 685–715.

Lu F and Milios E (1997) Globally consistent range scan alignment for environment mapping. *Autonomous Robots* 4: 333–349.

Magnusson M, Lilienthal A and Duckett T (2007) Scan registration for autonomous mining vehicles using 3D-NDT. *Journal of Field Robotics* 24(10): 803–827.

Mur-Artal R and Tardos JD (2017) ORB-SLAM2: an open-source SLAM system for monocular, stereo, and RGB-D cameras. *TRO* 33(5): 1255–1262.

Murphy KP (2012) *Machine Learning: A Probabilistic Perspective*. Cambridge, MA: MIT press.

Museth K (2013) VDB: high-resolution sparse volumes with dynamic topology. *ACM Transactions on Graphics* 32(3): 1–22.

Nadeau P (2024) A standard rigid transformation notation convention for robotics research.

Nguyen TM, Yuan S, Cao M, et al. (2021a) VIRAL SLAM: tightly coupled camera-IMU-UWB-lidar SLAM.

Nguyen TM, Yuan S, Cao M, et al. (2021b) Miliom: tightly coupled multi-input LiDAR-inertia odometry and mapping. *IEEE Robotics and Automation Letters* 6(3): 5573–5580.

Nguyen TM, Yuan S, Cao M, et al. (2022) NTU VIRAL: a visual-inertial-ranging-lidar dataset, from an aerial vehicle viewpoint. *The International Journal of Robotics Research* 41(3): 270–280.

Nguyen TM, Xu X, Jin T, et al. (2024) Eigen is all you need: efficient LiDAR-inertial continuous-time odometry with internal association. *IEEE Robotics and Automation Letters* 9(6): 5330–5337. DOI: 10.1109/LRA.2024.3391049.

Ni K, Steedly D and Dellaert F (2007) Tectonic SAM: exact, out-of-core, submap-based SLAM. In: Proceedings of the 2007 IEEE international conference on robotics and automation (ICRA), Roma, Italy, 10 – 14 April 2007, 1678–1685.

Pan Y, Xiao P, He Y, et al. (2021) MULLS: versatile LiDAR SLAM via multi-metric linear least square. In: 2021 IEEE international conference on robotics and automation (ICRA), Xian, China, 30 May – 5 June 2021. IEEE, 11633–11640.

Partow A (2015) C++ mathematical expression toolkit library (exprtk).

Pfreundschuh P, Oleynikova H, Cadena C, et al. (2024) Coin-lio: complementary intensity-augmented lidar inertial odometry. *IEEE International Conference on Robotics and Automation (ICRA),* 1730–1737. DOI: 10.1109/ICRA57147.2024.10610938.

Pomerleau F, Colas F, Siegwart R, et al. (2013) Comparing ICP variants on real-world data sets. *Autonomous Robots* 34(3): 133–148.

Ramezani M, Wang Y, Camurri M, et al. (2020) The newer college dataset: handheld lidar, inertial and vision with ground truth. In: 2020 IEEE/RSJ International conference on intelligent robots and systems (IROS), Las Vegas, Nevada, USA, 25–29 October 2020. IEEE, 4353–4360.

Reijgwart V, Millane A, Oleynikova H, et al. (2019) Voxgraph: globally consistent, volumetric mapping using signed distance function submaps. *IEEE Robotics and Automation Letters* 5(1): 227–234.

Reinders J (2007) *Intel Threading Building Blocks: Outfitting C++ for Multi-Core Processor Parallelism*. Sebastopol, CA: O'Reilly Media, Inc.

Reinke A, Palieri M, Morrell B, et al. (2022) LOCUS 2.0: robust and computationally efficient lidar odometry for real-time 3D mapping. *IEEE Robotics and Automation Letters* 7(4): 9043–9050. DOI: 10.1109/LRA.2022.3181357.

Roriz R, Cabral J and Gomes T (2022) Automotive LiDAR technology: a survey. *IEEE Transactions on Intelligent Transportation Systems* 23(7): 6282–6297. DOI: 10.1109/TITS.2021.3086804.

Rusinkiewicz S and Levoy M (2001) Efficient variants of the ICP algorithm. In: Proceedings third international conference on 3-D digital imaging and modeling, Quebec City, Canada, 28 May 2001 – 1 June 2001. IEEE, 145–152.

Schaefer A, Büscher D, Vertens J, et al. (2021) Long-term vehicle localization in urban environments based on pole landmarks extracted from 3-D lidar scans. *Robotics and Autonomous Systems* 136: 103709.

Seo D-U, Lim H, Lee S, et al. (2022) PaGO-LOAM: robust ground-optimized LiDAR odometry. In: Proceedings of the 2022 19th international conference on ubiquitous robots (UR), Jeju, South Korea, 4–6 July 2022, 1–7.

Shan T and Englot B (2018) Lego-LOAM: lightweight and ground-optimized lidar odometry and mapping on variable terrain. In: Proceedings of the 2018 IEEE/RSJ international conference on intelligent robots and systems (IROS), Madrid, Spain, 1–5 October 2018, 4758–4765.

Shan T, Englot B, Meyers D, et al. (2020) LIO-SAM: tightly-coupled lidar inertial odometry via smoothing and mapping. In: IEEE/RSJ international conference on intelligent robots and systems (IROS), Las Vegas, Nevada, USA, 25–29 October 2020. IEEE, 5135–5142.

Shan T, Englot B, Ratti C, et al. (2021) LVI-SAM: tightly-coupled lidar-visual-inertial odometry via smoothing and mapping. In: 2021 IEEE international conference on robotics and automation (ICRA), Xian, China, 30 May – 5 June 2021. IEEE, 5692–5698.

Sola J, Deray J and Atchuthan D (2018) A micro Lie theory for state estimation in robotics. arXiv preprint arXiv:1812.01537.

Strasdat H, Montiel JM and Davison AJ (2012) Visual SLAM: why filter? *Image and Vision Computing* 30(2): 65–77.

Teschner M, Heidelberger B, Müller M, et al. (2003) Optimized spatial hashing for collision detection of deformable objects. *Vmv* 3: 47–54.

Thrun S, Fox D, Burgard W, et al. (2001) Robust Monte Carlo localization for mobile robots. *Artificial Intelligence* 128(1-2): 99–141.

Tian Y, Khosoussi K, Rosen DM, et al. (2021) Distributed certifiably correct pose-graph optimization. *IEEE Transactions on Robotics* 37(6): 2137–2156.

Tranzatto M, Miki T, Dharmadhikari M, et al. (2022) Cerberus in the darpa subterranean challenge. *Science Robotics* 7(66): eabp9742.

Triggs B, McLauchlan PF, Hartley RI, et al. (2000) Bundle adjustment—a modern synthesis. In: Vision algorithms: theory and practice: international workshop on vision algorithms, Corfu, Greece, 21–22 September 1999. Springer, 298–372.

Tsintotas KA, Bampis L and Gasteratos A (2022) The revisiting problem in simultaneous localization and mapping: a survey on visual loop closure detection. *IEEE Transactions on Intelligent Transportation Systems* 23(11): 19929–19953.

Umeyama S (1991) Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 13(04): 376–380.

Vizzo I, Guadagnino T, Mersch B, et al. (2023) Kiss-icp: in defense of point-to-point icp–simple, accurate, and robust registration if done the right way. *IEEE Robotics and Automation Letters* 8(2): 1029–1036.

Wang L and Singer A (2013) Exact and stable recovery of rotations for robust synchronization. *Information and Inference: A Journal of the IMA* 2(2): 145–193.

Wang Z, Shen Y, Cai B, et al. (2019) A brief review on loop closure detection with 3d point cloud. In: 2019 IEEE international conference on real-time computing and robotics (RCAR), Irkutsk, Russia, 4–9 August 2019. IEEE, 929–934.

Wang Y, Sun Z, Xu CZ, et al. (2020) Lidar iris for loop-closure detection. In: 2020 IEEE/RSJ international conference on intelligent robots and systems (IROS), Las Vegas, Nevada, 25–29 October 2020. IEEE, 5769–5775.

Xu W, Cai Y, He D, et al. (2022a) Fast-LIO2: fast direct lidar-inertial odometry. *IEEE Transactions on Robotics* 38(4): 2053–2073.

Xu X, Zhang L, Yang J, et al. (2022b) A review of multi-sensor fusion slam systems based on 3D lidar. *Remote Sensing* 14(12): 2835.

Yang H and Carlone L (2019) A polynomial-time solution for robust registration with extreme outlier rates. *Robotics: Science and Systems*. Freiburg im Breisgau, June 22-26, 2019.

Yang H and Carlone L (2020) One ring to rule them all: certifiably robust geometric perception with outliers. *Advances in Neural Information Processing Systems* 33: 18846–18859.

Zhang J and Singh S (2014) Loam: lidar odometry and mapping in real-time. *Robotics: Science and Systems*. Berkeley, CA: University of California, 2, 1–9.

Zhang J and Singh S (2017) Low-drift and real-time lidar odometry and mapping. *Autonomous Robots* 41: 401–416.

Zhang L, Camurri M, Wisth D, et al. (2021) Multi-camera lidar inertial extension to the newer college dataset. arXiv preprint arXiv:2112.08854.