MAD-ICP: It Is All About Matching Data – Robust and Informed LiDAR Odometry

Simone Ferrari Luca Di Giammarino Leonardo Brizi Giorgio Grisetti

***Abstract*— LiDAR odometry is the task of estimating the ego-motion of the sensor from sequential laser scans. This problem has been addressed by the community for more than two decades, and many effective solutions are available nowadays. Most of these systems implicitly rely on assumptions about the operating environment, the sensor used, and motion pattern. When these assumptions are violated, several well-known systems tend to perform poorly.**

This paper presents a LiDAR odometry system that can overcome these limitations and operate well under different operating conditions while achieving performance comparable with domain-specific methods. Our algorithm follows the well-known ICP paradigm that leverages a PCA-based kd-tree implementation that is used to extract structural information about the clouds being registered and to compute the minimization metric for the alignment. The drift is bound by managing the local map based on the estimated uncertainty of the tracked pose. To benefit the community, we release an open-source C++ anytime real-time implementation at `https://github.com/rvp-group/mad-icp`.

I. Introduction

LiDAR odometry (LO) involves continually estimating the motion of a moving laser scanner within its environment. It is a fundamental component in various robotic applications, including autonomous vehicles and survey systems. Over the past decade, 3D laser scanners have become more affordable while increasing measurement accuracy and resolution.

A reliable LO system is an essential building block of more complex applications since it supports fundamental skills such as estimating a model of the environment and interacting with the operating scene. With LO becoming a pivotal element in more sophisticated systems, recent advancements emphasize qualities like robustness, simplicity, and ease of use. *Robustness* implies accurate functionality across diverse operating conditions without losing track, encompassing various environments, motion profiles, and LiDAR types. *Simplicity* and ease of use pertain to both the algorithm and the number of parameters involved.

This paper introduces a novel LO method leveraging an efficient and versatile kd-tree data structure and estimated pose uncertainty to dynamically maintain a robust environment model. The tree recursively divides data points based on Principal Component Analysis (PCA) [1], reducing a point cloud into leaves representing small planar patches with surface normals. The kd-tree supports isometric transformations and efficient searches with linear and logarithmic worst-case time complexities. The scan-to-model registration is done through Iterative Closest Point (ICP) using a *point-to-plane* error metric that compares corresponding leaves, hence performing a data reduction. The uncertainty of the pose, emerging as a byproduct of the pose estimation, is used to adaptively select which past frame to use to augment the model (Fig. 1). This is crucial for accurate and resilient estimations.

All authors are with the Department of Computer, Control, and Management Engineering "Antonio Ruberti", Sapienza University of Rome, Italy, Email: {s.ferrari, digiammarino, brizi, grisetti}@diag.uniroma1.it.

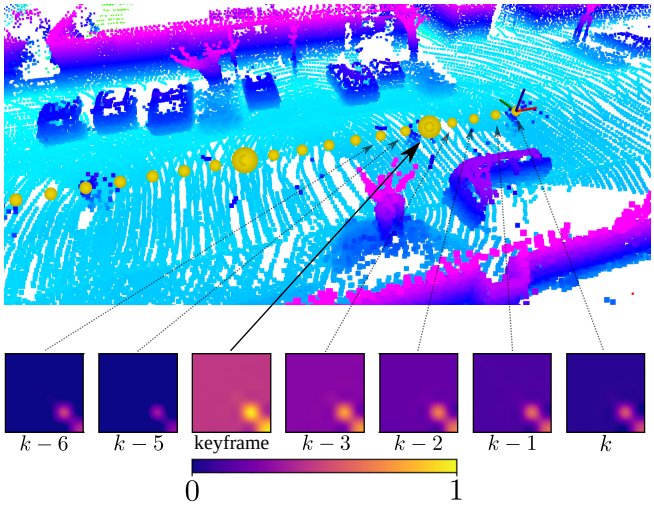


Fig. 1: Above is the point cloud generated by our LiDAR odometry processing a sequence of KITTI [2] dataset. The small yellow spheres represent the registered poses corresponding to each sensor measurement. The larger ones are poses related to the point clouds which form the model. Below, the normalized information 6×6 matrices are used to select the keyframe poses. The point cloud relative to the pose with the highest information is selected to be part of the model.

We conducted extensive experiments on a broad range of public data, acquired in heterogeneous environments (urban, highway, vegetation) with different motion profiles (handheld, car-like, drone). The comparative analysis with other SOTA methods reported that our approach has accuracy comparable to or better than other methods. At the same time, it is the only one that never fails in any sequence. We used the same set of parameters for all datasets acquired with the same LiDAR model.

We release an open-source C++ any-time implementation that can dynamically trade off the accuracy and run-time based on the available computational resources.

II. Related Work

Most of the LO systems combine a registration algorithm with a local map management. Each time a new scan (frame) becomes available, it is registered against the local

map (model) to compute the position of maximum overlap. Subsequently, the local map is augmented with the new measurement, and updated by removing old information. The model can consist of either a single aggregated global map, or in the union of different local maps/scans. Prominent methods like LeGO-LOAM [3], MULLS [4] and KISS-ICP [5] rely on the former scheme, while other SOTA approaches such as in DLO [6] use a set of frames. In contrast, SuMa [7] and MD-SLAM [8] anchor on a single frame as a local map and switch to the next frame when the registration support decreases.

The most prominent registration methods in LO rely on the ICP [9] paradigm. In its basic form, ICP iteratively aligns two point clouds by estimating a transformation that minimizes the distance between corresponding entities. This method originally emerged exploiting points, hence employing *point-to-point* distance as error metric. While accurate, this metric is very sensitive to data association, and wrong matches may lead to wrong registrations. For this reason, other approaches adopt different error functions. Chen *et al.* [10] propose to exploit the *point-to-plane* error metric. This metric, which considers the surface information, allows for sliding along planar patches. With sufficiently dense data, this approach is more robust and less prone to errors from incorrect data matching [11].

Methods that operate on raw data do not assume any particular structure in the environment, however when some structure is present (e.g. urban car-like environments), a point cloud can be segmented in geometric features such as sharp edges or planar surfaces. LOAM [12] has been the root of a family of approaches leveraging geometric feature extraction. Their major drawback is that they resort to 2D projections, which introduce many parameters that are strictly related to the sensor characteristics. In addition, strong assumptions like ground segmentation [3] make estimates degrading in non-planar environment (i.e.vertical motion involved).

All previously introduced methods estimate a single pose from each scan, treating point clouds as individual entities. However, unconventional LO approaches, such as CT-ICP [13] and LoLa-SLAM [14], focus on modeling a continuous trajectory rather than estimating discrete odometry. These methods additionally estimate velocities of intra-scan motion, incorporating on-the-fly point cloud undistortion (deskewing).

With the notable exception of KISS-ICP, all algorithms mentioned so far require the tuning of several parameters that need manual configuration, making them difficult to adapt to a particular context.

The data structure used to support the correspondence search plays a crucial role in LO. Common choices are voxel grids or kd-trees. KISS-ICP [5] performs multiple layers of voxelization, keeping for each voxel only a few points. This has the twofold effect of injecting some isotropic noise in the point distribution and de-biasing the point-to-point metric. CT-ICP [13] keeps the centroids of the voxels, hence denoising the cloud through averaging. The downside of decimating the data is the potential loss of support during scan registration, which might hinder robustness.

A kd-tree is a binary tree where each node represents an axis-aligned hyperplane that splits the data points into two sections. Typically, the splitting node is chosen as the median point along the largest dimension, leading to a balanced space division [15]. In mapping applications, where data often is 3-dimensional, comparative studies show that kd-trees perform excellently in solving k-nearest neighbor problems [16].

Aligning the normal of the splitting plane with the direction of maximum spread computed through PCA according to [17] is the optimal choice since it leads to lower the *depth*. Differently, common kd-tree implementations such as FLANN [18] avoid the PCA computational burden by using planes aligned with the axes of the initial reference system. This trades off computational complexity for the accuracy of the partitioning but has a negative impact on the correctness and completeness of the search. One downside of kd-tree is that each time a new point is added to the representation, the whole tree needs to be changed. FAST-LIO2 [19], addresses the problem of incrementally updating an axis-aligned kd-tree when points are added, removed or re-inserted with small computational overhead, leveraging on concepts similar to the ones of RB-trees [20] used to maintain sorted sets.

On modern computers performing a 3D PCA has a negligible cost, hence it does not make sense to drop the optimal space partitioning for computation. Furthermore, local surface characteristics such as normal (as the axis of minimal variation) are a natural outcome of PCA when computed on a densely distributed set of points.

Relying on these considerations, we designed a PCA kd-tree specifically for ICP, supporting isometric transformations and accurate normal extraction. We represent the local map as a forest of kd-trees, one per keyframe. The keyframes are selected among the most recent scans to maximize the Fisher information of the pose estimated during local map updates.

In summary, the primary contribution of this work is a novel LO that provides:

- a versatile and efficient data structure, capable of encoding all the necessary data in a single representation. This kd-tree is designed to facilitate every required operation, such as plane segmentation, incremental data association, and local map management;
- an information-aware criterion for adaptive local map updates, based on maximizing the pose information with the aim of limiting the noise injected in the local map;
- a robust anytime real-time C++ implementation that never loses track among different environments, without requiring parameter tuning.

III. MAD-ICP

Our pipeline builds on a minimal set of components. The main idea of our technique is to employ a kd-tree for all relevant operations in LO. For each new cloud $\mathcal{C}_k$ delivered by the LiDAR, a kd-tree $\mathcal{T}_k$ is built. The result of this preprocess is a data structure that encodes plane segmentation of the cloud (Sec. III-A), and allows nearest-neighbor queries. Each leaf of $\mathcal{T}_k$ will contain a small subset of the point cloud. The task is to match all the terminal nodes against the local map, until the whole tree (representing $\mathcal{C}_k$) is registered. This process is carried out by a point-to-plane ICP with an incremental data association strategy described in Sec. III-B. The uniqueness of

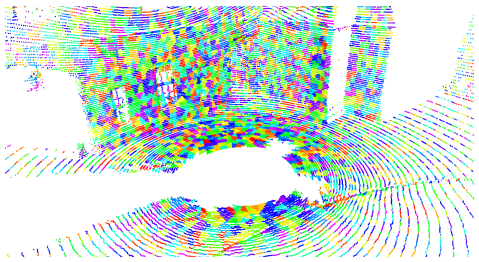


Fig. 2: Plane segmentation of a point cloud with our kd-tree explained in Sec. III-A. Every leaf is depicted with a different color.

our kd-tree relies on the possibility of transforming the entire data structure without altering the search capabilities, thus avoiding rebuilding after registration. The transformed kd-tree is maintained in a local map, which, thanks to the information aware update criterion, always remains reliable (Sec. III-C). Finally, velocities are estimated for point cloud undistortion and to provide a good initial guess for ICP (Sec. III-D). The full algorithm is outlined in Alg. 1.

Algorithm 1 MAD-ICP

local: local map $\mathcal{M}$, keyframe candidates $\mathcal{Q}$, past odometry poses $\mathcal{P}$, translational velocity $\mathbf{v}_{k-1}$, rotational velocity $\boldsymbol{\omega}_{k-1}$
input: point cloud $\mathcal{C}_k$

if $k = 1$ **then** ▷ initialization
  $\mathcal{T}_k \leftarrow$ kd-tree($\mathcal{C}_k$), $\mathcal{M} \leftarrow \{\mathcal{T}_k\}$
  $P \leftarrow \{\mathbf{I_{4x4}}\}$, $\mathbf{v}_{k-1} \leftarrow \mathbf{0_{3x1}}$, $\boldsymbol{\omega}_{k-1} \leftarrow \mathbf{0_{3x1}}$
  return

$\mathcal{C}_k \leftarrow$ deskew($\mathcal{C}_k$, $\mathbf{v}_{k-1}$, $\boldsymbol{\omega}_{k-1}$) ▷ preprocess
$\mathcal{T}_k \leftarrow$ kd-tree($\mathcal{C}_k$)

${}^w\mathbf{X}_{k,\text{guess}} \leftarrow$ integration(${}^w\mathbf{X}_{k-1}$, $\mathbf{v}_{k-1}$, $\boldsymbol{\omega}_{k-1}$) ▷ registration
${}^w\mathbf{X}_k \leftarrow$ ICP($\mathcal{M}$, $\mathcal{T}_k$, ${}^w\mathbf{X}_{k,\text{guess}}$)

$\mathcal{T}_k$.transform(${}^w\mathbf{X}_k$) ▷ tree transformation
$\mathcal{Q}$.push($\mathcal{T}_k$)
$\mathcal{P}$.append(${}^w\mathbf{X}_k$) ▷ velocity estimation
$\mathbf{v}_k$, $\boldsymbol{\omega}_k \leftarrow$ estimateVelocity($\mathcal{P}$)

$p \leftarrow T_k$.leavesPercentage() ▷ local map update
if $p < p_{\text{th}}$ **then**
  M.update($\mathcal{Q}$)
return

A. Point cloud surface normal segmentation

Each input cloud $\mathcal{C} = \mathbf{c}_{1:N}$ is converted into a kd-tree $\mathcal{T} = \langle \boldsymbol{\mu}, \mathbf{n}, \mathbf{d}, \mathcal{T}_l, \mathcal{T}_r \rangle$. Here $\boldsymbol{\mu}$ is the mean of the points, $\mathbf{n}$ is the surface normal and $\mathbf{d}$ is the direction of maximum spread. $\mathcal{T}_l$ and $\mathcal{T}_r$ represent respectively the left and right children of the node, and are set to $\emptyset$ for the leaves.

The construction procedure is recursive, and works as follows:

- compute mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Lambda}$ from the cloud $\mathcal{C}$;
- compute the eigenvectors $\mathbf{W} = \begin{bmatrix}\mathbf{w}_0 & \mathbf{w}_1 & \mathbf{w}_2\end{bmatrix}$ of $\boldsymbol{\Lambda}$;
- set the direction of maximum variation as the eigenvector corresponding to the largest eigenvalue $\mathbf{d} = \mathbf{w}_2$, and the direction of the normal as the eigenvector associated with the shortest eigenvalue $\mathbf{n} = \mathbf{w}_0$;
- compute the oriented bounding box $\langle b_0, b_1, b_2 \rangle$ by applying the rotation $\mathbf{W}$ to the cloud $\mathcal{C}$, and scanning for minimum and maximum along the three axes;
- if the largest dimension b_2 of the bounding box is lower than a threshold $b_{\max} = 0.2$ meters, it means that the points are concentrated in the space; hence recursion is stopped;
- in the case instead the node is not a leaf and spreads over a vast region ($b_2 > b_{\max}$), the points $\mathcal{C}$ are split into two sets to the left and the right of the splitting plane, based on the predicate:

$$\mathbf{d}^T(\mathbf{c}_i - \boldsymbol{\mu}) > 0. \tag{1}$$

The algorithm continues by recursing on these two sets $\mathcal{C}_l$ and $\mathcal{C}_r$, to compute the left and right children $\mathcal{T}_l$ and $\mathcal{T}_r$.

An example of the final plane segmentation can be seen in Fig. 2. This procedure leads to accurate surface normals where the LiDAR pattern is dense enough. Differently, in sparser regions of the cloud, leaves may be defined by only a few noisy points, making their normals unreliable. To address this, in the tree-building process, the normal of a node is propagated to its children when it is sufficiently flat, namely when the length of its shortest dimension is lower than the parameter $b_{\min}$. This technique allows us to estimate surface normals in most cases without adding further computational complexity to the tree construction. In our experiments, we fix $b_{\min} = 0.1$ meters. For the pseudocode of the kd-tree construction, we refer the reader to Alg. 2 and Alg. 3 in the supplementary material.

B. Data association and ICP estimate

Our alignment procedure follows an ICP schema that seeks a transformation $\mathbf{X} \in \mathbb{SE}(3)$ resulting in the best alignment between a reference model $\mathcal{M}$ and the current measurement.

In our system, the model consists of a forest of kd-trees, expressed in the world frame w, while the measurement at time k is represented by the kd-tree $\mathcal{T}_k$, expressed in LiDAR frame.

ICP alternates two steps: data matching and optimization. During data association, each leaf of $\mathcal{T}_k$ is transformed in w, based on the current transformation estimate, and its nearest neighbor in $\mathcal{M}$ is found. Let ${}^w\mathbf{X}_k$ be the sensor pose at instant k expressed in the world frame w and let $q = \langle \boldsymbol{\mu}_q, \mathbf{n}_q, \mathbf{d}_q, \emptyset, \emptyset \rangle$ be a generic leaf in $\mathcal{T}_k$. The search within a tree $\mathcal{T} \in \mathcal{M}$ is a recursive procedure that follows the same predicate of Eq. (1) used during the tree-building process:

$$\mathbf{d}^T({}^w\mathbf{X}_k\boldsymbol{\mu}_q - \boldsymbol{\mu}) > 0. \tag{2}$$

The search starts from the root of each tree in the model and iteratively continues by traversing the child indicated by Eq. (2). The recursion stops when a leaf $l = \langle \boldsymbol{\mu}_l, \mathbf{n}_l, \mathbf{d}_l, \emptyset, \emptyset \rangle$ is reached, thus q and l form a patching pair $\langle l, q \rangle$.

At this point, it is necessary to eliminate outliers. This is accomplished by verifying if the Euclidean distance between $\boldsymbol{\mu}_q$ and $\boldsymbol{\mu}_l$ is greater than the search radius r, which is computed as:

$$r = b_{\max} + \|\boldsymbol{\mu}_q\| \; b_{\text{ratio}}. \tag{3}$$

Here, $b_{\max}$ denotes the maximum size of the leaves, and b_{ratio} represents a parameter that increases the search size radius as the norm of the query $\boldsymbol{\mu}_q$ grows.

Generally, 3D ICP seeks to estimate a transformation matrix $\mathbf{X} \in \mathbb{SE}(3)$ composed by a rotation matrix $\mathbf{R} \in \mathbb{SO}(3)$ and translation vector $\mathbf{t} \in \mathbb{R}^3$. In our case, we estimate the isometry ${}^w\mathbf{X}_k$ denoting the sensor pose at instant k expressed in the world frame w. This is computed by iteratively minimizing a point-to-plane error term using Gauss-Newton. Given a match $\langle l, q \rangle$, the point-to-plane error is computed as:

$$e = \mathbf{n}_l^T \left({}^w\mathbf{X}_k \ \boldsymbol{\mu}_q - \boldsymbol{\mu}_l \right). \tag{4}$$

This shows the difference between ${}^w\boldsymbol{\mu}_q$ and $\boldsymbol{\mu}_l$, projected onto the normal $\mathbf{n}_l$. In addition, we employ the Huber robust loss and in our experiments we set its weighting parameter to $\rho_{\text{ker}} = 0.1$.

Upon convergence, a Gaussian covariance approximation of the solution can be found according to [21]. More specifically, since our approach carries on the optimization of the Lie Algebra $\mathfrak{se}(3)$, the inverse covariance of the estimate $\boldsymbol{\Sigma}^{-1} \in \mathbb{R}^{6\times6}$ is the system matrix of the Gauss-Newton algorithm.

Thanks to the density offered by modern LiDAR, the point-to-plane error exhibits robust performances, steering the optimization process toward explicit surfaces defined in the local map.

Our versatile kd-tree composed of splitting planes and surface normals can be directly transformed by the current estimate. Specifically, this happens multiple times: during optimization, where leaves are transformed over iterations to carry on the next correspondence search, and at the end, when the whole kd-tree $\mathcal{T}_k$, with its splitting planes, is transformed in the world frame through the final estimate ${}^w\mathbf{X}_k$. Tree transformation does not alter data association capabilities and is performed with limited computational effort. Finally, the transformed kd-tree is pushed into a queue $\mathcal{Q}$ of candidate frames for local map updates.

The pseudocode is available in Alg. 4, in the supplementary material.

C. Local map representation and update

Our local map consists of a forest $\mathcal{M}$ of kd-trees, each of them corresponding to a keyframe. This model, composed of independent kd-trees, preserves the accuracy and reliability of each point cloud, leaving the surface normals unchanged. Moreover, compared to either incremental kd-trees or voxel grids, our approach is simpler to implement because updating the local map is accomplished by merely pushing a new tree into it.

Keeping a reliable representation of the environment is crucial for effective registration. Many works in this field update the local map with every new observation [3, 4, 5, 13]. Instead, we believe that updating only when needed saves us from continuously injecting noise into our model. Furthermore, our local map update strategy is information aware, meaning that only the least uncertainty match is considered, avoiding inserting in $\mathcal{M}$ trees that are poorly matched during the optimization phase.

In principle, the local map update could be triggered by a pose change or by a change in the observed space. The first option is challenging to tune and lacks generalization, as it depends on the specific characteristics of the LiDAR sensor. For instance, sensors with an omnidirectional FoV should not trigger a local map update if they are rotated, as the portion of the observed space remains invariant to rotation. For this reason, we trigger local map update when $p < p_{\text{th}}$, where p is the percentage of leaves in $\mathcal{T}_k$ that matched with $\mathcal{M}$, and p_{th} a parameter that we set to 0.8.

When this happens the frame with the lowest uncertainty $\boldsymbol{\Sigma}$ (presented in Sec. III-B) is extracted from the list of frame candidates $\mathcal{Q}$. In order to compare covariance matrices associated with different candidates, several functions — known as *optimality criteria* — have been proposed [22]; in our algorithm, we employ the matrix determinant also known as *D*-optimality criterion [23]. This candidate becomes our new keyframe and is pushed inside the local map $\mathcal{M}$.

The determinant of poses uncertainty is considered also in [24], but with the aim of triggering a new keyframe selection. In contrast, we perform a minimization over $\boldsymbol{\Sigma}$ for choosing the new keyframe, and we trigger this event based on the support of the current observation.

We need to point out that when revisiting an already short-term explored area, our algorithm behavior degenerates to a localization process, anchoring to previously created keyframes without the need for new updates.

D. Velocity estimation and initial guess refinement

Velocity estimation is demanded for two purposes: first, to compensate for motion distortion by deskewing the current point cloud, and second, to predict the next pose, enabling a good initial guess for the ICP algorithm.

The full point cloud acquisition of a LiDAR is rather continuous. For this reason, motion during data capture reflects in point cloud distortion, leading to misaligned points, given the scanning frequency of the sensor. Motion compensation is essential to correct this effect, ensuring accurate spatial representation in the 3D environment [25]. As in [26], given the high rate at which each fragment of a point cloud is acquired, for deskewing, we employ a classic constant motion model.

Differently, given the lower rate at which each point cloud is fully measured, we adopt a smoothing technique in order to provide a good initial guess to our odometry estimation process. Our solution involves computing velocity using Gauss-Newton as in ICP (Sec. III-B). We smooth the translational and rotational velocities, respectively defined as $\mathbf{v}_k \in \mathbb{R}^3$ and $\boldsymbol{\omega}_k \in \mathbb{R}^3$ for the current time k using the last n poses. In our experiments, we set $n = 10$. Specifically, given the transformation ${}^i\mathbf{X}_k$ from the current time instant k to a previous one i (with $\Delta t_i = k - i$), we compute the errors as follows:

$$\mathbf{e}_{\mathbf{v}} = \Delta t_i \ \mathbf{v}_k - {}^i\mathbf{t}_k, \tag{5}$$

$$\mathbf{e}_{\boldsymbol{\omega}} = \Delta t_i \ \boldsymbol{\omega}_k - \text{Log}({}^i\mathbf{R}_k), \tag{6}$$

where the operator $\text{Log}(\cdot) \in \mathbb{R}^3$ is the logarithmic map of the rotation matrix evaluated at the identity.

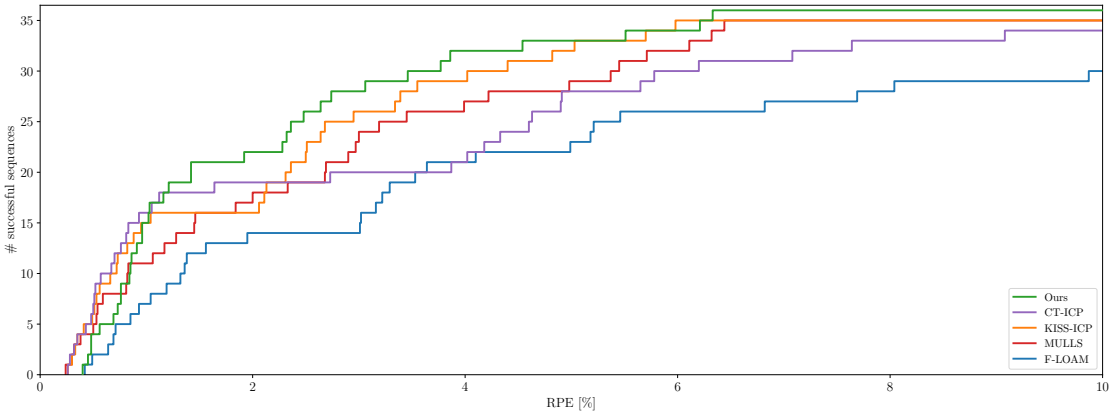


Fig. 3: The plot shows the cumulative error curve for each of the compared approaches, computed using all the sequences RPE [%] of Tab. II. For a given error value on the horizontal axis, the vertical axis shows how many sequences a method achieves a lower error.

IV. Experimental Evaluation

In this section, we report the results of our method extensively tested on several public datasets. In order to quantitatively compare our algorithm with other SOTA approaches, we employ the KITTI benchmark [2] and the VBR benchmark [27] to asses, respectively, local accuracy and robustness of odometry estimates (Sec. IV-B), exploiting the Relative Pose Error (RPE). We conducted all the experiments on a machine with CPU i9-13900kf with 16 physical cores. Our system consists of a minimal and robust LO accompanied by a single set of parameters (Sec. IV-A). Our quantitative results, presented in Sec. IV-C, show that our LO performs on par or better than other SOTA approaches. Experimental evaluation supports our claim in a wide range of heterogeneous environments, including highways, narrow buildings, stairs, static and dynamic settings. We also consider various LiDAR sensors and arbitrary motion profiles (encompassing different types of robots, cars, and handheld devices).

We report an extensive experimental campaign on the following datasets: KITTI [2], Mulran [28], Newer College [29], Hilti 2021 [30]. This experimental evaluation includes the comparison with four other LiDAR odometries released as open-source: F-LOAM [31], MULLS [4], CT-ICP [13] and KISS-ICP [5]. Our quantitative results are complemented by qualitative plots for global consistency of the compared trajectories (Fig. 4). Tab. I quantifies the enhancements of our information-aware strategy compared with a naive selection. For runtimes analysis of our system (Fig. 6, Fig. 7), further qualitative comparisons (Fig. 8, Fig. 9), and tests with RBG-D data (Fig. 10), we refer the reader to our supplementary material.

A. Parameters

Our approach relies on a set of six parameters that do not require tuning when transitioning between different LiDAR sensors or environments. This configuration demonstrates versatility, as it performs consistently across various scenarios. The parameters are already presented over Sec. III, but we compactly list them here for ease of reading:

- $b_{\max} = 0.2$: the maximum size (in meters) of kd-tree leaves;
- $b_{\min} = 0.1$: (in meters) in the tree-building process, when a node is flatter than this parameter, its normal is propagated and assigned to its children;
- $b_{\text{ratio}} = 0.02$: the increase factor of the search radius r needed in the data association;
- $p_{\text{th}} = 0.8$: the threshold needed to trigger the update of the local map M, ensuring an update when the current point cloud is registered for less than 80%;
- $\rho_{\text{ker}} = 0.1$: a parameter of the Huber robust estimator;
- $n = 10$: the number of last poses used to estimate the smoothed velocity.

B. Evaluation setup

For evaluating the accuracy, we employed the most common and widely used ranking method, namely the KITTI benchmark [2], which involves computing the Relative Pose Error (RPE) over a set of subsequences. The length of these subsequences is set to (100, 200, 300, ..., 800) meters for long trajectories like those in KITTI [2], Mulran [28] and Newer College with OS1 [29]. Instead, for relatively short recordings like Newer College with OS0 and Hilti 2021 [30] we opted to reduce these lengths to (10, 20, 30, ..., 80) meters. The evaluation ranks methods based on the average of these error values, measured in percentage. The average of each dataset is computed over all the errors over all possible subsequences, while the final score at the end of the table is computed simply as the mean of the average of each dataset. It is important to note that the total error reported here for each pipeline includes only the sequences where the LO does not fail, ensuring a fair comparison.

For evaluating the robustness, we employed the VBR benchmark [27], which consists of computing a cumulative error curve from all the sequences RPE [%]. The method ranking is determined as the area under the curve up to 10 %, so the larger the better. This metric rewards the robustness of evaluated methods, since a successful result on a sequence

usually adds much more area under the curve than slightly improving the accuracy on many sequences.

C. Comparison with State-of-the-art

For comparison, we explicitly selected the most recent, best-performing SOTA LO, appropriately configuring them for specific motion profiles, sensors, and environmental characteristics when needed.

The first work is F-LOAM [31], which aims to decrease the computational burden of previous LOAMs approaches by transforming the iterative processes into a two-stage distortion compensation method. It uses special features, such as edges with higher local smoothness and planar features with lower smoothness, to improve matching, by projecting the cloud into a range image.

A recent approach that instead relies on the basics of ICP is MULLS (Multi-metric Linear Least Square) [4]. It is designed to be independent of LiDAR specifications, eliminating the need for converting LiDAR data to rings or range images. It extracts geometric feature points, categorizes them (ground, facade, pillars), and performs ego-motion estimation using multi-metric linear Least Squares ICP.

CT-ICP [13] incorporates the motion compensation into registration by adding intra-scan velocities to the variables being estimated within ICP and uses a point-to-plane metric. It relies on a consistent voxel-based local map that is updated with the new scan after registration and deskewing.

KISS-ICP [5] is another modern approach that leverages ICP. This pipeline adopts the constant velocity model for motion compensation. New scans are first subsampled and then matched with the local map to find correspondences. This process is accompanied by an adaptive thresholding scheme. Then, ICP is performed with a point-to-point metric, and the local map is updated as in CT-ICP.

Results using the KITTI benchmark (explained in Sec. IV-B) are shown in Tab. II.

The first section of the table is about the eleven training sequences (00-10) of KITTI [2], recorded by car. These first results exhibit only small differences in performance among the five pipelines. This dataset is recorded with a Velodyne HDL-64. However, it is important to note that KITTI is still the most famous vision benchmark, but the inconsistencies in the baselines are on the order of meters [27], since their ground truth system relies only on RTK-GPS measurements. This ensures good global consistency but not local, which, unfortunately, is required when evaluating odometry systems.

The second section includes the Mulran [28] data collection. This is a car dataset recorded with Ouster OS1-64. In some of these sequences, our method performs slightly better than the others. However, even in this case, the public ground truth trajectories are of poor quality, even worse than KITTI, always relying only on RTK-GPS. Still, they provide some insights into the behavior of these LO in large outdoor scenarios.

In the third and fourth sections, we conducted tests on the Newer College [29] dataset, which is divided into sequences recorded through Ouster OS0-128 and OS1-64 sensors. This collection allowed us to rank the pipelines when dealing with handheld data. This time, the baselines are highly accurate and allow for a fair comparison, given the ground truth processed by registering LiDAR scans to a high-definition map built using a 3D Leica laser scanner. It is evident that our method, KISS-ICP and CT-ICP perform quite similarly in the OS1 sequences. This result was expected because, despite a change in motion profile, OS1 data still represents large and open environments, similar to those found in outdoor KITTI and Mulran streets.

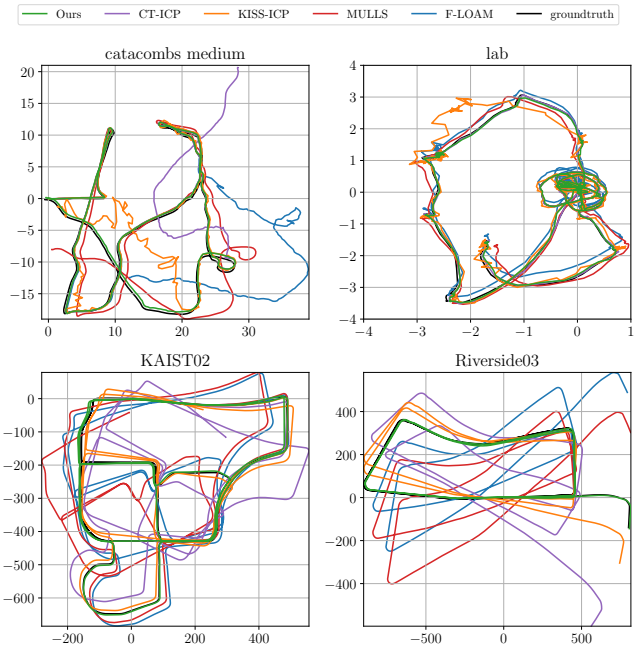


Fig. 4: The plots illustrate the qualitative absolute consistency of the compared approaches within some sequences of the employed datasets.

The situation is entirely different with the OS0 data, where we outperform all the compared methods, demonstrating the versatility and accuracy of our approach. Very similar results can be found in the last table section, showing the only sequences of the Hilti 2021 [30] dataset where the ground truth is provided. The two sequences are acquired using a quadrotor and handled, respectively, with an Ouster OS0-64.

At the end of the table, we present the average score from all the datasets, indicating that our approach performs slightly better than the compared methods. It is important to note that the total error reported here for each pipeline includes only the sequences where other LO do not fail, ensuring a fair comparison. Finally, Tab. III reports for each method the area under the curve depicted in Fig. 3. Our method achieves the highest ranking. This suggests that our system is not only more accurate on average but also exhibits robustness in challenging scenarios where other current SOTA methods lose track.

V. Conclusion

The key of our LO is the clever reuse of previously computed information in many stages: the PCA used to partition the space and construct the search structure is also used to

	catacombs easy	catacombs medium	cloister	math easy	math medium	quad easy	quad medium	stairs	avg
Ours	**1.16**	**1.42**	**1.42**	**0.40**	**0.56**	**2.64**	**5.51**	**0.91**	**1.84**
Ours w/o IA	1.31	2.13	1.45	0.45	0.79	**2.64**	**5.51**	1.06	1.97

TABLE I: Quantitative results obtained from Newer College dataset [29] to show the impact of our information aware local map update strategy discussed in Sec. III-C. Applying our algorithm in conjunction with this information-aware strategy improves accuracy. Notably, in sequences characterized by a confined motion within four walls, such as *quad* sequences, there is no discernible difference as the redundant motion renders all information equally effective.

		Ours	KISS-ICP	F-LOAM	MULLS	CT-ICP
KITTI	0	0.73	0.51	0.69	0.53	**0.50**
	1	0.85	0.72	1.95	0.81	**0.67**
	2	0.76	0.53	1.04	0.59	**0.51**
	3	0.84	**0.66**	1.32	1.84	0.70
	4	0.69	**0.35**	0.71	0.38	**0.35**
	5	0.48	0.30	3.64	0.32	**0.26**
	6	0.48	0.26	0.49	**0.24**	0.28
	7	0.45	0.33	0.42	**0.30**	0.32
	8	1.21	0.82	0.93	0.83	**0.81**
	9	1.02	0.49	0.64	0.54	**0.48**
	10	0.96	0.56	1.19	0.82	**0.52**
	avg	0.82	0.54	1.25	0.60	**0.53**
Mulran	DCC01	3.06	**2.95**	4.10	3.19	4.91
	DCC02	**2.28**	2.50	3.53	2.33	4.18
	DCC03	**1.92**	2.11	3.16	2.90	4.63
	KAIST01	**2.36**	**2.36**	3.22	2.68	4.33
	KAIST02	2.32	**2.31**	3.01	3.45	4.02
	KAIST03	2.74	**2.68**	3.29	2.97	4.60
	River.01	3.77	**3.55**	5.21	4.98	7.08
	River.02	3.46	**3.34**	4.99	5.45	6.20
	River.03	**2.48**	2.51	5.18	3.99	5.78
	Sejong01	4.54	**4.40**	13.59	6.44	9.08
	Sejong02	6.21	**4.82**	7.69	5.37	7.64
	Sejong03	6.33	5.03	6.82	6.32	**4.90**
	avg	4.34	**3.82**	6.97	4.93	6.25
NC0	cat. easy	1.16	2.06	1.36	2.00	**1.12**
	cat. med.	**1.42**	5.70	11.02	3.00	10.42
	cloister	**1.42**	3.39	5.46	1.45	1.64
	m. easy	**0.40**	0.41	0.85	0.50	0.43
	m. med.	**0.56**	0.73	18.05	1.06	0.57
	quad easy	**2.64**	**2.64**	3.02	2.69	2.73
	quad med.	**5.51**	5.98	31.06	5.71	5.65
	stairs	**0.91**	17903.09	8.04	23.84	29.69
	avg	**1.84**	2.60	3.34	2.24	2.02
NC1	short	0.86	0.88	1.56	1.28	**0.83**
	long	0.96	0.95	9.87	1.17	**0.93**
	parkland	**1.03**	1.04	14.79	6.11	1.05
	avg	0.93	0.94	1.56	1.21	**0.90**
Hilti	drone	**3.86**	4.02	19.49	4.22	3.87
	lab	**0.76**	2.13	1.38	1.46	**0.76**
	avg	2.79	3.37	**1.38**	3.26	2.79
	tot avg	**2.14**	2.25	2.90	2.45	2.50

TABLE II: Results obtained using the KITTI benchmark over different public datasets. The table shows the Relative Pose Error (RPE), which is reported in %. For each dataset we show the average, at the end we show the total average. In order to ensure a fair comparison, failure results for the average calculations (the one highlighted in red) are not included.

	Ours	KISS-ICP	F-LOAM	MULLS	CT-ICP
AUC	**288.57**	275.98	205.24	262.08	247.67

TABLE III: Area under the curve for each approach compared in Fig. 3. The higher, the better.

classify the surface and to assign the normals. Similarly, the Fisher information matrix, resulting from registration, is used for adaptive local map updates that limit the drift. Most of the calculations are controlled by the input data, through the implicit statistics computed along the different stages. This results in a robust, simple, and adaptive LO that addresses the limitations of most of the existing solutions overfitted to specific data domains. Our claims are backed up by an extensive set of comparative experiments. To contribute to the community, we provide an open-source C++ anytime real-time implementation. As future works, to make our system even more robust, we envision an odometry that fuses an inertial sensor in a continuous way.

REFERENCES

[1] A. Maćkiewicz and W. Ratajczak, "Principal components analysis (pca)," *Computers & Geosciences*, vol. 19, no. 3, pp. 303–342, 1993. [Online]. Available: https://www.sciencedirect.com/science/article/pii/009830049390090R

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

[3] T. Shan and B. Englot, "Lego-loam: Lightweight and ground-optimized lidar odometry and mapping on variable terrain," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2018, pp. 4758–4765.

[4] Y. Pan, P. Xiao, Y. He, Z. Shao, and Z. Li, "Mulls: Versatile lidar slam via multi-metric linear least square," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*. IEEE, 2021, pp. 11 633–11 640.

[5] I. Vizzo, T. Guadagnino, B. Mersch, L. Wiesmann, J. Behley, and C. Stachniss, "Kiss-icp: In defense of point-to-point icp–simple, accurate, and robust registration if done the right way," *IEEE Robotics and Automation Letters (RA-L)*, vol. 8, no. 2, pp. 1029–1036, 2023.

[6] K. Chen, B. T. Lopez, A. Agha-mohammadi, and A. Mehta, "Direct lidar odometry: Fast localization with dense point clouds," *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 2, pp. 2000–2007, 2022.

[7] J. Behley and C. Stachniss, "Efficient surfel-based slam using 3d laser range data in urban environments." in *Proc. of Robotics: Science and Systems (RSS)*, 2018.

[8] L. D. Giammarino, L. Brizi, T. Guadagnino, C. Stachniss, and G. Grisetti, "Md-slam: Multi-cue direct slam," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2022, pp. 11 047–11 054.

[9] P. Besl and N. McKay, "Method for registration of 3-d

shapes," in *Sensor fusion IV: control paradigms and data structures*, vol. 1611. Spie, 1992, pp. 586–606.

[10] Y. Chen and G. Medioni, "Object modelling by registration of multiple range images," *Image and Vision Computing*, vol. 10, no. 3, pp. 145–155, 1992.

[11] F. Pomerleau, F. Colas, R. Siegwart, and S. Magnenat, "Comparing icp variants on real-world data sets: Open-source library and experimental protocol," *Autonomous robots*, vol. 34, pp. 133–148, 2013.

[12] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time." in *Proc. of Robotics: Science and Systems (RSS)*, 2014.

[13] P. Dellenbach, J. Deschaud, B. Jacquet, and F. Goulette, "Ct-icp: Real-time elastic lidar odometry with loop closure," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*. IEEE, 2022, pp. 5580–5586.

[14] M. Karimi, M. Oelsch, O. Stengel, E. Babaians, and E. Steinbach, "Lola-slam: Low-latency lidar slam using continuous scan slicing," *IEEE Robotics and Automation Letters (RA-L)*, vol. 6, no. 2, pp. 2248–2255, 2021.

[15] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 209–226, 1977.

[16] J. Elseberg, S. Magnenat, R. Siegwart, and A. Nüchter, "Comparison of nearest-neighbor-search strategies and implementations for efficient shape registration," *Journal of Software Engineering for Robotics*, vol. 3, no. 1, pp. 2–12, 2012.

[17] J. McNames, "A fast nearest-neighbor algorithm based on a principal axis search tree," *IEEE Trans. on Pattern Analysis and Machine Intelligence (TPAMI)*, vol. 23, no. 9, pp. 964–976, 2001.

[18] M. Muja and D. Lowe, "Flann-fast library for approximate nearest neighbors user manual," *Computer Science Department, University of British Columbia, Vancouver, BC, Canada*, vol. 5, p. 6, 2009.

[19] W. Xu, Y. Cai, D. He, J. Lin, and F. Zhang, "Fast-lio2: Fast direct lidar-inertial odometry," *IEEE Transactions on Robotics*, vol. 38, no. 4, pp. 2053–2073, 2022.

[20] C. Okasaki, "Red-black trees in a functional setting," *Journal of functional programming*, vol. 9, no. 4, pp. 471–477, 1999.

[21] A. Censi, "An accurate closed-form estimate of icp's covariance," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*. IEEE, 2007, pp. 3167–3172.

[22] J. A. Placed, J. Strader, H. Carrillo, N. Atanasov, V. Indelman, L. Carlone, and J. Castellanos, "A survey on active simultaneous localization and mapping: State of the art and new frontiers," *IEEE Trans. on Robotics (TRO)*, 2023.

[23] A. Wald, "On the efficient design of statistical investigations," *The Annals of Mathematical Statistics*, vol. 14, no. 2, pp. 134–140, 1943.

[24] J. Kuo, M. Muglikar, Z. Zhang, and D. Scaramuzza, "Redesigning slam for arbitrary multi-camera systems," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*. IEEE, 2020, pp. 2116–2122.

[25] O. Salem, E. Giacomini, L. Brizi, L. D. Giammarino, and G. Grisetti, "Enhancing lidar performance: Robust de-skewing exclusively relying on range measurements," in *Intl. Conf. of the Italian Association for Artificial Intelligence (AIxIA)*, vol. 14318. Springer Nature, 2023, p. 310.

[26] S. Anderson and T. D. Barfoot, "Ransac for motion-distorted 3d visual sensors," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2013, pp. 2093–2099.

[27] L. Brizi, E. Giacomini, L. D. Giammarino, S. Ferrari, O. Salem, L. D. Rebotti, and G. Grisetti, "Vbr: A vision benchmark in rome," *arXiv preprint arXiv:2404.11322*, 2024.

[28] G. Kim, Y. S. Park, , Y. Cho, J. Jeong, and A. Kim, "Mulran: Multimodal range dataset for urban place recognition," in *Proc. of the IEEE Intl. Conf. on Robotics & Automation (ICRA)*. IEEE, 2020, pp. 6246–6253.

[29] M. Ramezani, Y. Wang, M. Camurri, D. Wisth, M. Mattamala, and M. Fallon, "The newer college dataset: Handheld lidar, inertial and vision with ground truth," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2020, pp. 4353–4360.

[30] M. Helmberger, K. Morin, B. Berner, N. Kumar, G. Cioffi, and D. Scaramuzza, "The hilti slam challenge dataset," *IEEE Robotics and Automation Letters (RA-L)*, vol. 7, no. 3, pp. 7518–7525, 2022.

[31] H. Wang, C. Wang, C. Chen, and L. Xie, "F-loam : Fast lidar odometry and mapping," in *Proc. of the IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2021, pp. 4390–4396.

[32] T. Schops, T. Sattler, and M. Pollefeys, "Bad slam: Bundle adjusted direct rgbd-d slam," in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 134–144.

VI. Supplementary Material

In this section we expand the main procedures that were not fully detailed.

A. Algorithms

In Alg. 2 and Alg. 3 we provide a formalization of the kd-tree building process. Additionally, in Alg. 4, we offer a comprehensive explanation of how this data structure is utilized for identifying matches and estimating the transformation.

Algorithm 2 kd-tree

input: point cloud $\mathcal{C}$
output: kd-tree $\mathcal{T}$
$\mathcal{T} \leftarrow$ kd-node($\mathcal{C}$, $false$, $\mathbf{0_{3x1}}$)
return $\mathcal{T}$

B. Runtimes

Our implementation is *anytime* because it can provide a valid estimate in real-time, scaling on the hardware. This

Algorithm 3 kd-node

input: point cloud $\mathcal{C}$, boolean flag `inherits_normal`, inherited normal $\mathbf{n}_p$
output: node $\mathcal{T}$

$\boldsymbol{\mu}$, $\boldsymbol{\Lambda} \leftarrow$ meanAndCovariance($\mathcal{C}$) ▷ PCA
$\mathbf{W} \leftarrow$ eigenvectors($\boldsymbol{\Lambda}$)
$\mathbf{w}_0$, $\mathbf{w}_1$, $\mathbf{w}_2 \leftarrow$ sortEigenvectors($\mathbf{W}$)

$\mathcal{T} \leftarrow$ emptyKdNode() ▷ assign node attributes
$\mathcal{T}.\boldsymbol{\mu} \leftarrow \boldsymbol{\mu}$
$\mathcal{T}.\mathbf{n} \leftarrow \mathbf{w}_0$
$\mathcal{T}.\mathbf{d} \leftarrow \mathbf{w}_2$
$\mathcal{T}.\mathbf{b} \leftarrow$ computeBoundingBox($\mathcal{C}$, $\boldsymbol{\mu}$, $\mathbf{W}$)

if $\mathcal{T}.b_2 < b_{\max}$ **then** ▷ base case
  if `inherits_normal` **then**
    $\mathcal{T}.\mathbf{n} \leftarrow \mathbf{n}_p$ ▷ enhance normal
  return $\mathcal{T}$

if ***NOT*** `inherits_normal` ***AND*** $\mathcal{T}.b_0 < b_{\min}$ **then**
  `inherits_normal` $\leftarrow$ *true*
  $\mathbf{n}_p \leftarrow \mathcal{T}.\mathbf{n}$ ▷ propagate normal if node is flat

$\mathcal{C}_l$, $\mathcal{C}_r \leftarrow$ splitCloud($\mathcal{C}$, $\mathcal{T}.\boldsymbol{\mu}$, $\mathcal{T}.\mathbf{d}$) ▷ recursive case
$\mathcal{T}.\mathcal{T}_l \leftarrow$ kd-node($\mathcal{C}_l$, `inherits_normal`, $\mathbf{n}_p$)
$\mathcal{T}.\mathcal{T}_r \leftarrow$ kd-node($\mathcal{C}_r$, `inherits_normal`, $\mathbf{n}_p$)
return $\mathcal{T}$

Algorithm 4 ICP

input: local map $\mathcal{M}$, current tree $\mathcal{T}_k$, prediction ${}^w\mathbf{X}_{k,\text{guess}}$
output: estimated pose ${}^w\mathbf{X}_k$

${}^w\mathbf{X}_k \leftarrow {}^w\mathbf{X}_{k,\text{guess}}$ ▷ initialization

while timeNotElapsed() **do** ▷ anytime ICP rounds
  $\mathbf{H} \leftarrow \mathbf{0_{6x6}}$
  $\mathbf{b} \leftarrow \mathbf{0_{6x1}}$

  for $q \in \mathcal{T}_k$.getLeaves() **do**
    for $\mathcal{T} \in \mathcal{M}$ **do**
      ${}^wq \leftarrow q$.transform(${}^w\mathbf{X}_k$) ▷ incremental
      $l \leftarrow \mathcal{T}$.nearestNeighbor(${}^wq$) ▷ data association

      e, $\mathbf{J} \leftarrow$ firstOrderApprox(q, l) ▷ system determination
      $h \leftarrow$ Huber(e)
      $\mathbf{H} \leftarrow \mathbf{H} + h\,\mathbf{J}^T\mathbf{J}$
      $\mathbf{b} \leftarrow \mathbf{b} + h\,\mathbf{J}^T e$

  ${}^w\mathbf{X}_k \leftarrow$ solve($\mathbf{H}$, $\mathbf{b}$) ▷ system resolution

return ${}^w\mathbf{X}_k$

means that we can return a valid solution even if computation is interrupted before it ends (i.e. less than maximum ICP iterations) to satisfy the real-time constraints. The algorithm is expected to find a more accurate solution the longer it keeps running. In order to speed up computation while maintaining good accuracy, we provide a multi-thread implementation, where each physical core manages a kd-tree of the local map $\mathcal{M}$. Being our approach scan-to-model, enough amount of information needs to be encoded in the local map for higher accuracy. How the number of threads or kd-tree in the local map impacts the final result is showed in Fig. 6. In Fig. 7

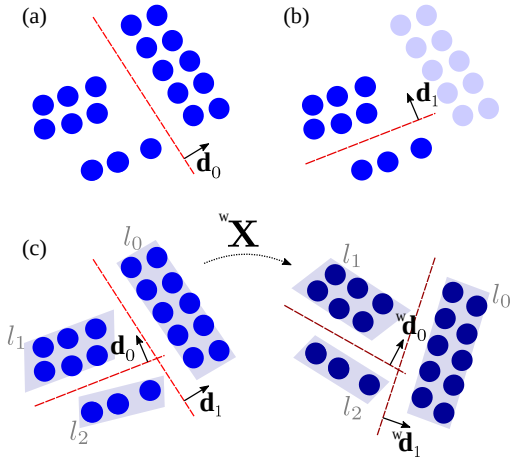


Fig. 5: A 2D illustration of the kd-tree building and transformation processes. Our splitting criteria recursively divides the cloud with a plane whose normal is the direction $\mathbf{d}$ of maximum spread (a, b), identified by PCA. The kd-tree is transformed in the world frame w through an isometry ${}^w\mathbf{X}$ (c). Our transformed kd-tree maintains the search properties.

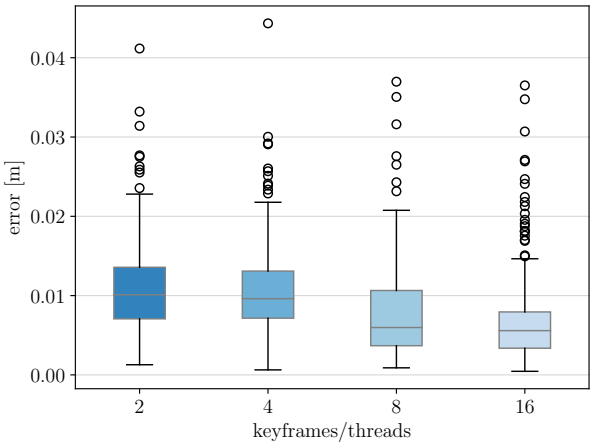


Fig. 6: The image shows the RPE error in meters against the number of threads, thus the number of kd-trees in the forest/local map. We adopt this technique, in order to speed up computation while keeping good accuracy.

instead, we show how our runtimes are related to the spatial extension of the observation (the farther the points of the point cloud the bigger the size of the observation). While this impacts the scan registration procedure, does not make substantial differences during kd-tree building.

C. Qualitative results

To complement our quantitative results, in Fig. 8 we provide further plots for global consistency of compared trajectories. Moreover, in Fig. 9 and Fig. 10 we report qualitative plots for 3D mapping of the *stairs* sequence and of two RGB-D sequences, respectively. Fig. 9 shows that our pipeline is the only one employable in vertical environments, while Fig. 10 demonstrates that our approach works also with point clouds generated with a different sensor.

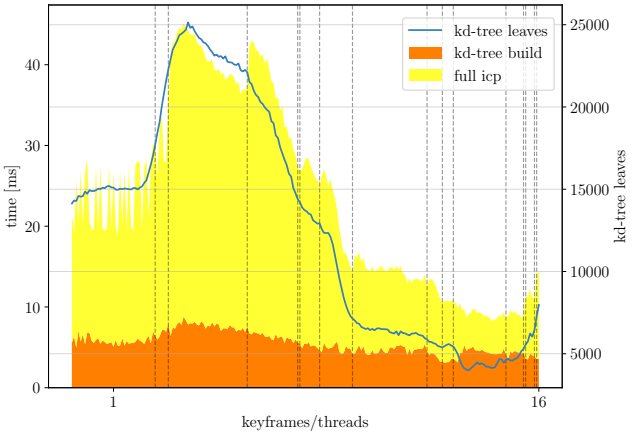


Fig. 7: The image shows our runtimes in milliseconds for each point cloud. Specifically, we show kd-tree building and full registration process. The horizontal-axis highlights the number of keyframe from 1 to 16 (which is equivalent to the number of threads). Moreover, we show the correlation between ICP runtimes and number of leaves contained in the kd-trees.

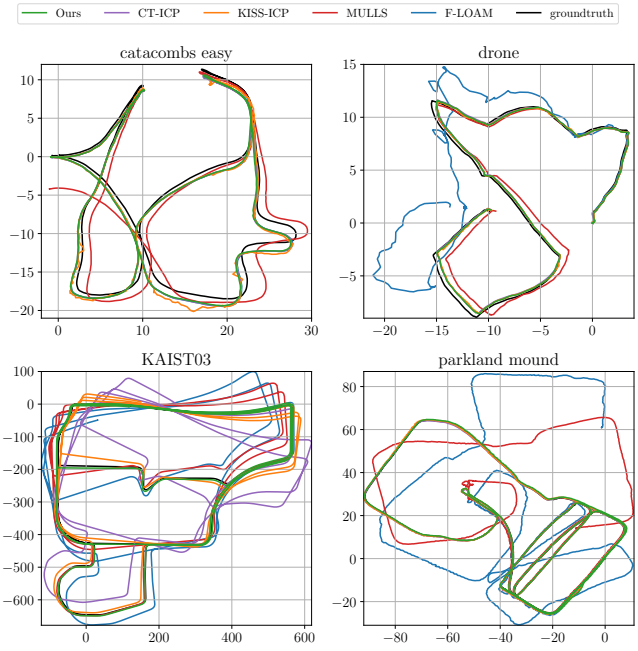
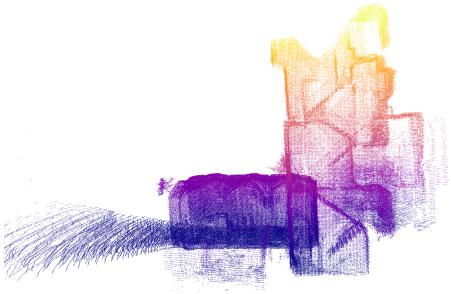
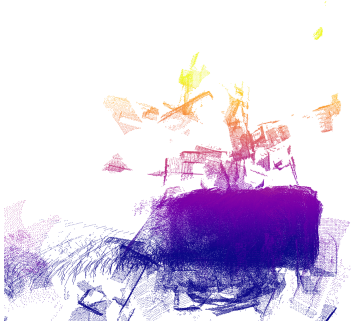


Fig. 8: The plots illustrate the qualitative absolute consistency of the compared approaches within some sequences of the employed datasets.



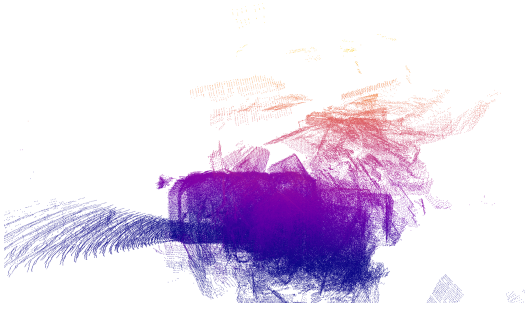
(a) Ours



(b) KISS-ICP [5]

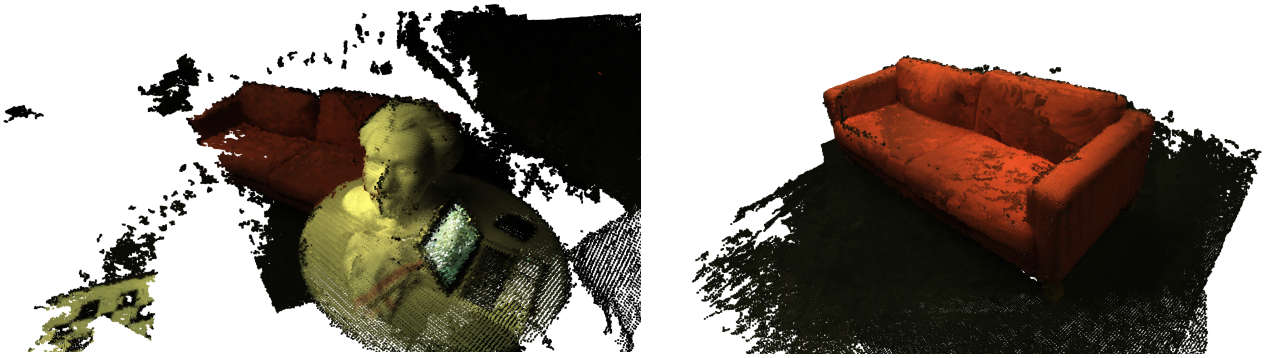
(c) F-LOAM [31]

(d) MULLS [4]



(e) CT-ICP [13]

Fig. 9: Qualitative results on the *stairs* sequence of Newer College dataset [29]. Our approach is the only one that robustly handle vertical motion.



(a) *einstein_1* sequence (b) *sofa_1* sequence

Fig. 10: Qualitative results of our pipeline on two sequences of ETH3D SLAM dataset [32]. Our approach is employable also with RGB-D clouds. Our registration schema does not refine any kind of structure. Something usually done with noisy RGB-D point clouds.