ART-SLAM: Accurate Real-Time 6DoF LiDAR SLAM

Matteo Frosi[1], Matteo Matteucci[2] *Member, IEEE*

***Abstract*—Real-time six degrees-of-freedom pose estimation with ground vehicles represents a relevant and well-studied topic in robotics due to its many applications such as autonomous driving and 3D mapping. Although some systems already exist, they are either not accurate or they struggle in real-time settings. In this paper, we propose a fast, accurate and modular LiDAR SLAM system for both batch and online estimation. We first apply downsampling and outlier removal, to filter out noise and reduce the size of the input point clouds. Filtered clouds are then used for pose tracking, possibly aided by a pre-tracking module, and floor detection, to ground optimize the estimated trajectory. Efficient multi-steps loop closure and pose optimization, achieved through a g2o pose graph, are the last steps of the proposed SLAM pipeline. We compare the performance of our system with state-of-the-art point cloud-based methods, LOAM, LeGO-LOAM, A-LOAM, LeGO-LOAM-BOR, LIO-SAM and HDL, and show that the proposed system achieves equal or better accuracy and can easily handle even cases without loops. The comparison is done evaluating the estimated trajectory displacement using the KITTI (urban driving) and Chilean (underground mine) datasets.**

***Index Terms*—SLAM, Range Sensing, Localization, Mapping.**

I. Introduction

TRAJECTORY estimation and map building represent core aspects of many applications in robotics, such as autonomous driving. A great amount of simultaneous localization and mapping (SLAM) systems with 6 degrees-of-freedom (6 DoF) have been proposed in literature in the last decades, with the goal of estimating accurate trajectories with real-time performance. These methods can be grouped in two main categories, vision-based and point cloud-based systems, depending on the main sensor used.

Point cloud-based systems, can capture and represent the environment with a high level of detail, due to the density of the clouds, and they are not afflicted by the issues of vision-based methods, such as illumination and viewpoint changes. Moreover, tracking performed with point clouds is more accurate and stable than its visual counterpart, and it is generally preferred when LiDAR data is available. However, achieving real-time performance, while keeping high accuracy, still remains an open quest in point cloud-based systems.

Manuscript received: September 9, 2021; Revised: December 3, 2021; Accepted: December 28, 2021.

This paper was recommended for publication by Editor Javier Civera upon evaluation of the Associate Editor and Reviewers' comments.

[1]Matteo Frosi is a Ph.D. student at the Dipartimento di Elettronica, Informazione e Bioingegneria of Politecnico di Milano, Milan, Italy matteo.frosi@polimi.it

[2]Matteo Matteucci is Full Professor at the Dipartimento di Elettronica, Informazione e Bioingegneria of Politecnico di Milano, Milan, Italy matteo.matteucci@polimi.it

Digital Object Identifier (DOI): see top of this page.

The most critical aspect which hinders real-time point cloud SLAM is the alignment of LiDAR scans. During the last decades, many algorithms have been created to find the relative motion between two point clouds, operation also known as scan matching. The most used and known methods to perform scan matching are Iterative Closest Point (ICP) [1] and its many variants. The idea behind these algorithms is to align two point clouds iteratively, until convergence or a stopping criterion is satisfied. Although ICP suffers from high computational cost, Generalized ICP [2] and more recent parallel versions (e.g., VGICP [3]) are faster and more accurate, and can be used as alternatives to the original algorithm.

To overcome the computational shortcomings of full point clouds scan matching, feature-based approaches have been proposed. These methods work similarly to standard scan matching, but require fewer resources. They achieve this by extracting 3D features from the clouds, such as edges or planes, and then match them. A low-drift and real-time LiDAR odometry and mapping (LOAM) method is proposed in [4]. LOAM performs 3D point feature to edge, and plane, scan matching to find correspondences between point clouds. The performance of LOAM deteriorates when resources are limited and no loop closure is performed, leading to large estimation errors, as we show in Section III. Improving LOAM, LeGO-LOAM [5] has been proposed, being a lightweight real-time pose estimation and mapping system, composed by five modules: segmentation, feature extraction, LiDAR odometry, LiDAR mapping and transform integration. Speedup is achieved by filtering the input clouds through image-based segmentation, performed on the 2D range projection of each scan. More recent variants of LOAM and LeGO-LOAM have been proposed, optimizing it, namely A-LOAM and LeGO-LOAM-BOR, respectively. Another improved system, w.r.t. LOAM, is LIO-SAM [6], which couples mandatory IMU data and the registration method of LOAM, achieving better performance than the other systems.

Feature-based systems are, in general, less accurate than methods which perform scan matching on whole clouds. For this reason, loop closure and trajectory optimization are mandatory steps in their pipeline. These tasks can easily become computationally demanding as the size of a trajectory increases. To overcome this problem, graph SLAM systems have been proposed, such as [7] and [8], where the trajectory of the robot, estimated via scan matching, is modeled as a graph. There are multiple advantages of this approach, as described by Grisetti et. al. in [9], such as the ability to introduce relationships between sensor data and/or observations from the environment, or a great availability of frameworks for efficient graph optimization, which translates in the optimization of the

corresponding trajectory.

A recent point cloud-based system that relies on a graph structure is HDL [10], which consists of four steps. First, laser scans are pre-processed and filtered to reduce their size. Then, the filtered clouds are used to simultaneously perform tracking and to possibly detect the ground plane. Poses estimated through tracking, and floor coefficients extracted from the point clouds, are used to build a graph of the trajectory, i.e., a pose graph, which is later optimized. The system achieves superior performance, but it is slow, especially when dealing with large point clouds (e.g., more than 100K points).

All algorithms available in literature, being based on feature matching or full scan matching, either achieve high accuracy at the cost of computational time, or sacrifice the quality of the trajectory to obtain real-time performance. Moreover, these systems are monolithic and difficult to modify and adapt, and they are usually bound to some existing framework (e.g., ROS [11]), often hindering portability on different operating systems and integrability with other software. Nevertheless, full point cloud systems are preferable, as the maps (or submaps) created can be used to efficiently obtain accurate 3D reconstructions of the environment, as described in [12], useful for many applications, such as autonomous driving, virtual reality and augmented reality.

For these reasons, in this paper we propose a new system, *ART-SLAM*, to perform point cloud-based graph SLAM, inspired by HDL, with multiple contributions. ART-SLAM is able to achieve real-time performance, retaining high accuracy, even in scenarios without loops. The proposed system is also able to efficiently detect and close loops in the trajectory, using a three-phased algorithm. ART-SLAM presents a high degree of modularity, due to its architecture, described in Section II, and can be easily integrated and improved, being also a zero-copy software. Additionally, it is not ROS-dependent, although a wrapper for integration with ROS is also provided. ART-SLAM is available open source at https://github.com/MatteoF94/ARTSLAM.

II. ART-SLAM

A. System overview

An overview of the proposed framework is represented in Fig. 1. The system is composed by multiple distinct modules, which can be grouped into two main blocks. The first, mandatory (colored in gray), is the core of ART-SLAM, and it is formed by all the modules that perform SLAM on the input point clouds (orange, in figure). The other blocks of the proposed framework are optional, as they can be used to integrate the main system with data coming from different sensors or with pre-processed inputs.

Given an incoming laser scan, the first step is to process it, in the *pre-filterer*, to reduce its size and remove noisy points. The filtered cloud is then sent simultaneously to two modules. The most important one, the *tracker*, estimates the current displacement of the robot by performing scan-to-scan matching with previous filtered scans. The other, *floor detector*, finds the robot pose w.r.t. the ground, adding height and rotational consistency to the trajectory. The current pose

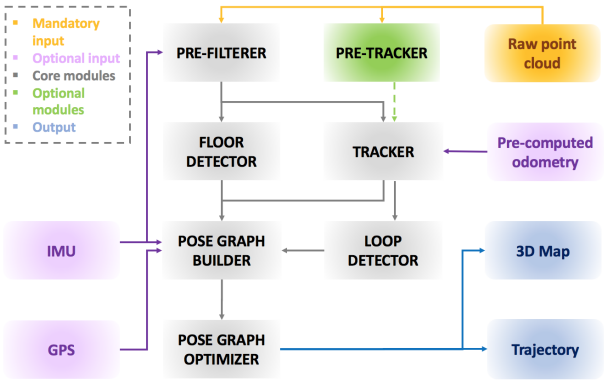

Fig. 1. Architecture of the proposed system, ART-SLAM. Core modules are the mandatory blocks of the pipeline, used to estimate an accurate trajectory and to build a 3D map of the environment, given a sequence of input point clouds.

estimate is sent, along with its corresponding point cloud, to the *loop detector* module, which tries to find loops between new and previous point clouds, again performing scan-to-scan matching. Moreover, poses, loops, and floor coefficients (estimated by the floor detector module) are used to build the pose graph, representing the trajectory of the robot. Lastly, the pose graph is optimized to increase the poses accuracy.

IMU and GPS data (pink in Fig. 1) can be integrated in the *pose graph builder* module, to increase the accuracy of the estimated trajectory. IMU data can also be used in the pre-filterer to de-skew point clouds. Moreover, *pre-computed odometry* (e.g., through a different sensor or system) can be fed to the tracker as initial guess for the scan matching. The *pre-tracker* module (green in Fig. 1) performs multi-level scan-to-scan matching, to quickly obtain a rough estimate of the motion of the robot, before the tracking step: this estimate is broadcasted to the tracker, as initial guess of the scan-to-scan matching, to boost the performance of the module.

Differently from the majority of systems available in literature, our proposed framework is fully modular and its components work independently one from another. This is possible thanks to the *register and dispatch* technique used to create the system, leading to the creation of modules formed by the following elements. Observers allow a module to capture data as soon as it is available, independently of the type, which is then put into one or multiple dispatch queues, i.e., FIFO structures with the purpose of avoiding the loss of incoming data. The core of a module is its main characteristic: it elaborates one datum per queue at a time, extracting it from the relative dispatch queue. As soon as the core finishes its task, it gives the byproducts of the module to the notifier, which broadcasts them to the modules in need.

There are three main advantages of using this architecture. First, input data, for each module, is safely stored for later usage, independently from its processing rate, meaning that it cannot be lost. Then, new modules can be easily integrated in the system, being only dependent on the type of data needed. Lastly, the same core task can be performed in parallel, on multiple threads, if it does not require temporal coherence.

B. Pre-filterer

The pre-filterer module has the purpose of reducing the size of the input point cloud and to remove noise and outliers. Data reduction, or downsampling, is essential because, as stated in the introduction, scan matching on full point clouds can become computationally demanding if the size of the cloud is large enough. Downsampling can reduce point clouds by a factor 5, or even more, if needed, while retaining the spatial structure and density of the initial scan.

The clouds are then filtered to remove outliers and noise points. This operation is more costly w.r.t. the downsampling task. To improve performance w.r.t. HDL [10], we split the cloud into four pairs of octants and perform filtering on each separately, in parallel, obtaining a speedup linear with the number of threads used (up to more than 50%). After that, all the smaller clouds are combined to form a larger, filtered point cloud, ready to be broadcasted to other modules.

C. Tracker

Short term data association, also known as pose tracking, establishes the motion between consecutive poses. The tracker adopts a keyframe-based approach to estimate the trajectory of the robot, performing scan-to-scan matching, using state-of-the-art algorithms (e.g., ICP [1], GICP [2], VGICP [3] and NDT [13]), depending on the user choice and the type of environment the robot is navigating.

Keyframes are data structures describing the motion of the robot in selected locations of its trajectory. They are represented by multiple variables, used to collect data associated to the various poses. In ART-SLAM, each keyframe contains a point cloud and the pose (odometry) estimated by the tracker, data which is also used for loop closure detection, pose graph construction and map creation. Other useful information contained in a keyframe are the timestamp associated to the point cloud, the estimated accumulated distance from the beginning of the trajectory and, if available, acceleration and orientation coming from other sensors.

To reduce the computational resources needed to efficiently perform SLAM, not all the filtered point clouds in input to the tracker become keyframes. Except for the first keyframe, which corresponds to the first point cloud received by the system, other keyframes must satisfy at least one of the following criteria:

- Be distant from the previous keyframe of a user-defined translation $\Delta trans$, in meters
- Be rotated from the previous keyframe of a user-defined angle $\Delta orientation$, in radians
- Have a difference in timestamps of a user-defined interval ΔT, in seconds

The thresholds $\Delta trans$, $\Delta orientation$ and ΔT depend on the dataset considered (e.g., length, type or complexity) and the type of trajectory to be estimated, and should be tuned accordingly to obtain a reasonable number of keyframes, as too few would decrease the accuracy of the SLAM system, and too many would degrade its performance. In indoor scenes, for example, $\Delta trans$ could be set to 0.2 meters, while in large scale urban environments $\Delta trans > 5$ meters.

Given the cloud corresponding to the current keyframe K_n and the available new filtered point cloud in input c_t, scan-to-scan matching is performed between them, to find their relative motion. The algorithm requires an initial guess, to boost performance and accuracy, which can be chosen in two ways: either it is available through other means (e.g., from odometry estimated from another sensor), or a constant motion model is assumed, and the previous relative transformation is used (the one computed between the point cloud of the current keyframe K_n and the previous filtered point cloud c_{t-1}).

Usually, algorithms for point cloud-based tracking find the relative motion between consecutive clouds, c_{t-1} and c_t, and then compose this transformation with the previous ones, to estimate the current odometry. This method may seem more accurate, but it accumulates error the more distant the clouds are from the current keyframe. In ART-SLAM, instead, the motion of the robot is always referred w.r.t. the keyframe closest in time, and the previous motion is taken into consideration only to estimate the guess for scan matching.

This approach, which is unique to ART-SLAM, also allows the system to skip the whole scan matching procedure if pre-computed odometry is available. The latter, even if not completely accurate, allows indeed to immediately check if the current point cloud is a candidate for the selection of a new keyframe. If it is not, the tracker does not perform scan matching, and the relative transformation between the current keyframe and the pre-computed odometry is saved to be used as motion guess in the next iteration. Skipping the scan matching step greatly benefits the performance of the tracker, while retaining the same accuracy.

Once the tracker has detected a point cloud which satisfies any of the keyframe creation criteria described previously, a new keyframe is built and it is broadcasted to the loop closure detection and pose graph builder modules.

D. Pre-tracker

Aligning two full scale clouds would result in the best transformation estimate, as all the 3D points are accounted for. However, this approach is often unsuitable for real-time applications, especially on low-end devices. For this reason, scan matching should be aided with a rough initial guess. The computation of the latter is the purpose of our pre-tracker module, which performs multi-scale scan matching, while working in parallel to the pre-filterer.

First, the same point cloud given in input to the pre-filterer is fed to this module, where it is heavily downsampled. The reduced point cloud is then used to perform scan-to-scan matching with a previously downsampled cloud. This alignment is fast, due to the reduced size of the inputs, even if not as accurate as if it was done with non-downsampled clouds. This procedure can be repeated using the same point clouds, but downsampled at a different scale, lower than the one used in the first phase. The relative motion resulting from these steps is broadcasted to the tracker, to be used as initial guess in the current scan matching, allowing it to possibly skip the frame and reduce the computational resources needed.

E. Floor detection

To enforce height and orientation consistency in the trajectory, filtered point clouds are processed to find the ground plane in them. This can be modeled as a four dimensional vector $GP(a, b, c, d)$ representing the plane equation $a * x + b * y + c * z + d = 0$.

Floor detection should handle multiple scenarios, such as planar or planar-like motion (e.g., urban road), rough terrains (e.g., rocky paths) and environments with ascents and descents. While HDL [10] deals only with the planar motion, in ART-SLAM all scenarios are considered.

In the first case, i.e., planar or planar-like motion, the floor detector module takes a point cloud and manipulates it in the following way. As the ground can be found within a small region of the input scan, the first step performed is clipping the cloud within an acceptable range of search. This step greatly reduces the cloud size, boosting performance when searching for the floor. Then, the clipped output is filtered to eliminate points whose normal is highly non-vertical. This is done to avoid mistakes due to planar-like surfaces in the environment, such as walls. Lastly, Random Sample Consensus (RANSAC) for plane detection is done on the filtered laser scan, to detect and estimate the ground plane coefficients.

When dealing with rough terrains, a floor cannot be found in the previous way, as no planar structures can be detected with RANSAC. The input scan is further clipped, this time horizontally: only the 3D points within a threshold distance from the center of the cloud are kept. This is done to trim the cloud to be as close as possible to the robot, to remove outlier objects such as rocks, logs, or anything which is not planar-like. The few remaining points are then used to perform closed form plane fitting with the least squares method. If the parameters $\{a, b, c, d\}$ are found, they are broadcasted, together with the timestamp associated to the corresponding point cloud, to the pose graph builder module.

The last scenario is trickier to identify just by using a point cloud, as inclined planes are parallel to the robot wheels and cannot be distinguished from non inclined planes by just using point clouds for detection. The process of discovering inclined planes takes place in the pose graph builder module. When a set of floor coefficients $\{a, b, c, d\}$ is associated to a keyframe, the builder checks if there is a noticeable change (user-defined) in vertical orientation w.r.t. the previous keyframe. If affirmative, it means that there has been a change in slope in the trajectory of the robot, and an inclined ground plane has been detected.

F. Loop closure

While moving, the robot may return to a place which was previously visited, forming a loop in its trajectory. Finding loops adds motion constraints in the estimated robot poses, correcting drift and estimation errors. The hard part about loop closure is not asserting the presence of a loop, which can be accomplished via simple scan matching, but detecting when loop closure is even a possibility. To do this, we need to decide when and where to look. In ART-SLAM, detection is performed in three consecutive steps, to efficiently search for loops within the collected keyframes.

First, each time a keyframe K_{query} is available, it is compared against all previous existing keyframes. Instead of performing scan-to-scan matching between the possible pairs $\{K_{query}, K_{candidate}\}$, an odometry based selection is performed. If K_{query} and $K_{candidate}$ are too close in terms of trajectory, meaning that they have a low accumulated distance, they cannot be considered candidates, as it is unlikely that two keyframes, corresponding to point clouds acquired shortly one from the other, would result in a useful loop. Moreover, the loop detector checks if the position, estimated through tracking, of $K_{candidate}$ is in the neighborhood of the pose corresponding K_{query}, within a threshold range, which accounts for drift errors induced by the tracker module. If K_{query} and $K_{candidate}$ satisfy these constraints, meaning that they are sufficiently close in space and far in time, they can be considered a loop closure candidate pair.

Once all candidate pairs have been found, they must be further thinned down to avoid unnecessary computation. The approach proposed in [14] converts point clouds in 2D polar grids, and efficiently compares them using a KD-tree to select the k most similar ones to a given point cloud query. The second phase for efficient loop closure detection in ART-SLAM adopts this method, by comparing the 2D polar grid of the point cloud associated to the query keyframe with the 2D polar grids corresponding to the candidate keyframes. At the end of this step, only k candidate pairs for loop closure remain, ready to be used in the last step.

The few number of candidates allows for scan-to-scan matching on each pair of point clouds, to obtain a set of relative motions. All transformations are then compared to find the best one, i.e., the one with the highest accuracy, and corresponding to the smallest Euclidean distance between all the pairs K_{query} and $K_{candidate}$. If a best match is found, it means that a new loop has been efficiently detected, and it is added to the pose graph as a new constraint.

Differently from HDL [10], where only the first and last steps are performed, in ART-SLAM, the addition of the Scan Context method allows for scalable and efficient loop closure. Indeed, as the length of the trajectory to be estimated increases, the number of pairs to be checked for loop closures also grows in size, as more and more keyframes are added. The first two steps are very fast operations, with the former consisting mainly in a matrix multiplication and the latter being proved to be scalable [14]. Moreover, the 2D polar grids are pre-computed when inserting the keyframes in the pose graph, further decreasing the computation time needed by the loop closure module. At the end of the second phase, there will always be at most k candidate pairs, independently from the number of keyframes to check, making this three-phased approach suitable for efficient loop closure detection.

G. Pose graph building and optimization

As mentioned in the introduction and in the description of the system, our framework is a form of graph SLAM [9]. In graph SLAM, the poses of the robot are modeled as nodes in

TABLE I
PARAMETERS USED FOR THE EXPERIMENTAL VALIDATION, FOR REPRODUCIBILITY, AS DESCRIBED IN [15].

Module	Parameter name	Value
pre-filterer	Downsample method	VOXELGRID
	Downsample resolution	0.25 [m]
	Outlier removal method	RADIUS
	Radius	0.4 [m]
tracker	ΔTrans keyframes	5.0 [m]
	ΔAngle keyframes	0.25 [rad]
	ΔTime keyframes	1.0 [sec]
loop detector	Loop closure search radius	35.0 [m]
	Loop closure min. distance	25.0 [m]
scan matching	Registration method	FAST_GICP
	Max. iterations	64
	Transformation epsilon	0.01

a graph, named *pose graph*, and labeled with their position in the environment. The nodes are connected with edges representing spatial constraints between poses, resulting from sensor measurements (e.g., IMU or GPS) or scene elements, as the floor coefficients in our case.

Each node in the pose graph represents a robot position and at least one measurement (the point cloud is mandatory) acquired at that position; moreover, each node is associated to the corresponding keyframe. An edge between two nodes consists in a probability distribution over the relative transformation of the robot poses corresponding to the nodes. These transformations are either odometry measurements between consecutive positions, or are determined by aligning the sensor measurements acquired between two keyframes, via tracking. Because of the noise corrupting the sensors and the drift in the robot odometry, the associated edges represent soft constraints and are not fixed, to be later optimized. It is, however, possible to insert absolute constraints, which cannot be modified in any way. Examples of hard constraints are floor coefficients, GPS and IMU data, although they can also be set as non absolute elements, to account for the uncertainty of the sensors or the measurements. Finally, edges can be added when performing loop closure, between non consecutive nodes in the graph, forming a ring-like structure.

The structure of the pose graph is given to optimization algorithms to compute the optimal trajectory which satisfies all sensors and motion constraints, giving high accuracy estimates, while elaborating a large number of poses. In our implementation, we use the g2o optimization framework [16], as it proves to be fast and accurate over long trajectories. Moreover g2o allows for the insertion of custom elements in the pose graph, and as such, it is an optimal solution for our modular system, allowing future upgrades.

III. EXPERIMENTAL VALIDATION OF THE SYSTEM

The proposed system is compared with other methods for point cloud-based SLAM: LOAM [4], LeGO-LOAM [5], A-LOAM, LeGO-LOAM-BOR, LIO-SAM [6] and HDL [10], with A-LOAM and LeGO-LOAM-BOR being two advanced versions of LOAM and LeGO-LOAM (code improvement and re-engineering), respectively. We also include, in the comparison, different variants of our system: ART-SLAM without Scan Context, ART-SLAM with Scan Context, ART-SLAM with IMU (for de-skewing and orientation correction in the pose graph), and ART-SLAM with GPS. We evaluate these systems in three scenarios coming from the KITTI dataset [17] [18], corresponding to a short, medium and long sequences, respectively. Lastly, we perform a brief study on the Chilean underground mine dataset [19], to test ART-SLAM in rough environments, typical of long tunnels with very few distinctive features.

LOAM, LeGO-LOAM, A-LOAM, LeGO-LOAM-BOR and LIO-SAM do not require particular parameter tuning, although they need a custom implementation of the point cloud projection module, depending on the laser sensor used. Hence, in our tests, we changed such parameters accordingly. On the other hand, HDL and ART-SLAM share the same configuration parameters, e.g., keyframe selection thresholds and pre-filtering method. Table I shows the most important parameters used in the experiments, for both HDL and ART-SLAM, to allow reproducibility, as described in [15].

It should be noticed that in the KITTI dataset, both IMU and GPS data are acquired at very low frequency, the same as for LiDAR point clouds (about 10 Hz). For this reason, we consider them unreliable, giving low weights in the pose graph, meaning they have only a minor contribution during the graph optimization phase. LIO-SAM, to work, obligatorily requires IMU data at high frequency, and for this reason, to evaluate it, we used the unsynchronized version of the KITTI dataset, having IMU data taken at 100 Hz.

Experiments are tested on a 2021 XMG 64-bit laptop with Intel(R) Core(TM) i7-11800H CPU @ 2.30GHz x 8 cores, with 24576 KB of cache size.

A. Comparison and results

To evaluate the systems we compute the absolute trajectory error (ATE). This metric measures the difference between points of the true and the estimated trajectory. As a preprocessing step, we associate the estimated poses with ground truth poses using timestamps and point cloud indices. We also include a visual evaluation of the estimated trajectory and show the reconstructed 3D map in the long sequences (KITTI 00 and Chilean). Finally, we also show the processing time of the mandatory modules of the proposed system, to prove its real-time performance.

Fig. 2 shows the estimated trajectories of Sequence 07 of the KITTI odometry dataset [17]. All the methods considered for comparison, except for LOAM, accurately follow the ground truth, correctly finding the loop and optimizing the poses. LOAM, instead, quickly drifts apart from the true trajectory: this is caused by the fact that no loop closure is performed. Table II further details the obtained results, as it represents the mean, root mean square error (RMSE), and standard deviation (STD) of the absolute trajectory error, in meters. The highest accuracy is achieved by ART-SLAM with IMU data. This was expected, as this method combines both the advantages of scan-to-scan matching, to track de-skewed point clouds, and orientation integration in the pose graph, to correct

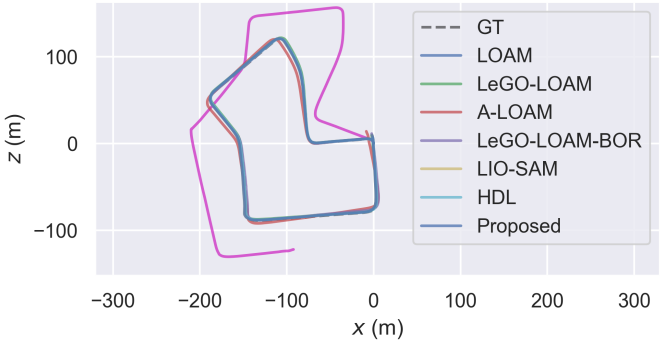

Fig. 2. Comparison between the trajectories estimated by LOAM [4], LeGO-LOAM [5], A-LOAM, LeGO-LOAM-BOR, LIO-SAM [6], HDL [10] and the proposed system, without Scan Context, on Sequence 07 of the KITTI odometry dataset [17].

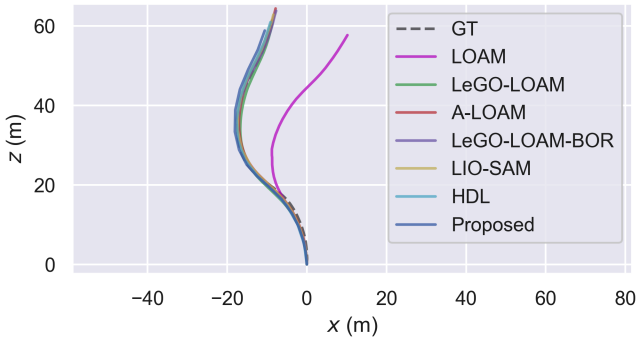

Fig. 3. Comparison between the trajectories estimated by LOAM [4], LeGO-LOAM [5], A-LOAM, LeGO-LOAM-BOR, LIO-SAM [6], HDL [10] and the proposed system, without Scan Context, on city Sequence 05 of the KITTI raw dataset [18].

TABLE II
ATE ON SEQUENCE 07 OF THE KITTI ODOMETRY DATASET [17].

ATE [m]	MEAN	RMSE	STD
LOAM	>10	>10	>10
LeGO-LOAM	1.191	1.309	0.546
A-LOAM	2.467	2.741	1.195
LeGO-LOAM-BOR	1.604	1.807	0.832
LIO-SAM	0.509	0.675	0.351
HDL	0.954	1.253	0.767
ART-SLAM	0.698	0.777	0.341
ART-SLAM (SC)	0.730	0.813	0.358
ART-SLAM (IMU)	**0.343**	**0.366**	**0.127**
ART-SLAM (GPS)	0.782	0.869	0.382

TABLE III
ATE ON CITY SEQUENCE 05 OF THE KITTI RAW DATASET [18].

ATE [m]	MEAN	RMSE	STD
LOAM	>5	>5	>5
LeGO-LOAM	0.707	0.768	0.300
A-LOAM	0.938	1.044	0.459
LeGO-LOAM-BOR	1.094	1.169	0.409
LIO-SAM	**0.493**	**0.338**	**0.280**
HDL	0.893	0.912	0.476
ART-SLAM	0.742	0.812	0.331
ART-SLAM (SC)	0.742	0.812	0.331
ART-SLAM (IMU)	0.746	0.814	0.326
ART-SLAM (GPS)	0.343	0.588	0.477

the front-end estimates. It should be noticed that LIO-SAM comes in second place, proving that, with the same sensors, tracking relying on full point clouds is the best choice for accurate results. Moreover, the slightly worse results of the other variants of ART-SLAM are also expected. The quality of ART-SLAM with Scan Context depends on the loops identified by the Scan Context method, which are not necessarily the best ones, hence reducing the overall accuracy. ART-SLAM with GPS shows a higher ATE because, as stated before, we consider GPS data as unreliable (for the KITTI dataset) and this influences negatively the optimization of the pose graph, leading to a trajectory farther from the ground truth.

After having dealt with a large sequence with the presence of loops, we also evaluated the systems on a shorter sequence. As short datasets do not have a ground truth, we use, instead, raw GPS data, provided along with the point clouds. Fig. 3 shows the estimated trajectories of city Sequence 05 of the KITTI raw dataset [18]. As before, all methods, except for LOAM, accurately represent the ground truth, with small errors in the trajectory. It should not come as a surprise that the results are more or less the same, as, for short trajectories, tracking is performed a limited amount of times, and there is not enough distance to accumulate errors. Table III shows the ATE statistics, in meters. As before, all the systems but LOAM show good results, accurately following the GPS signal, here used as ground truth due to its relatively high accuracy. This is the reason ART-SLAM with GPS is the most accurate method, even though, for a fair comparison, LIO-SAM was highlighted as the most accurate system.

In Fig. 4, we show the behavior of ART-SLAM on one of the most complex sequences of the KITTI odometry dataset, i.e, Sequence 00. In the visual comparison, we did not include A-LOAM and LIO-SAM, which estimated trajectories are far off the ground truth, and would have cluttered the image. From Table IV, one can see the high degree of accuracy achieved by the proposed system, reaching low translation error. Once again, the best accuracy is achieved by ART-SLAM with IMU, followed by ART-SLAM without Scan Context. Fig. 5 and Fig. 6 show the map reconstructed by ART-SLAM and a detailed area of it, respectively.

Table V shows the average processing times, per frame, of the various ART-SLAM variants. Intuitively, independently from the sequence considered, pre-filtering, tracking and floor detection take the same time (aside from the ART-SLAM with IMU case, where de-skewing is also performed, hence the higher pre-filtering time). It is important to notice, instead, the different times associated to loop detection and graph optimization, which clearly depend on the size of the trajectories to estimate. Moreover, the table clearly shows the importance of Scan Context when performing loop closure: in Sequence 00, the loop detection time, using Scan Context, is half the case without it. Considering that data is acquired at 10 Hz and looking at the system architecture of Fig. 1, one can clearly see that ART-SLAM shows real-time performance. All

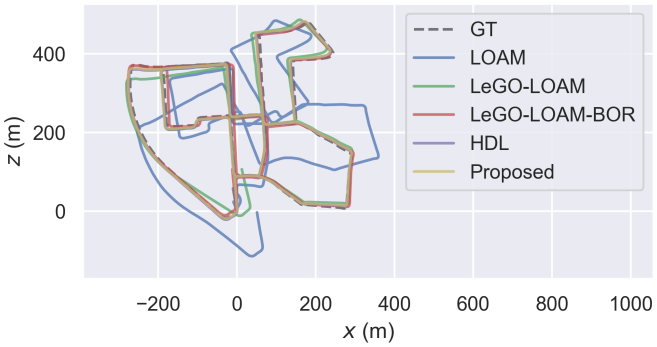

Fig. 4. Comparison between the trajectories estimated by LOAM [4], LeGO-LOAM [5], LeGO-LOAM-BOR, HDL [10] and the proposed system, without Scan Context, on sequence 00 of the KITTI odometry dataset [17].

TABLE IV
ATE ON SEQUENCE 00 OF KITTI ODOMETRY DATASET [17].

ATE [m]	MEAN	RMSE	STD
LOAM	>10	>10	>10
LeGO-LOAM	9.537	11.666	6.718
A-LOAM	>10	>10	>10
LeGO-LOAM-BOR	6.240	6.613	2.188
LIO-SAM	>10	>10	>10
HDL	1.378	1.424	0.779
ART-SLAM	0.981	1.092	0.478
ART-SLAM (SC)	1.232	1.409	0.684
ART-SLAM (IMU)	**0.907**	**1.014**	**0.454**
ART-SLAM (GPS)	1.092	1.156	0.380

methods, except HDL, are feature-based, and written to run in real-time. Nevertheless, from Table V, one can see that the proposed system, even if full point cloud-based, achieves the same processing performance.

Lastly, we show a visual evaluation of the accuracy achieved by ART-SLAM on the Chilean underground mine dataset [19]. This sequence is relatively long and contains multiple loop closures. The dataset is rather peculiar, as there are only 44 LiDAR measurements, taken at 30 to 40 meters of distance each and with large rotations, of around 25M points each (while a cloud in KITTI has around 130K points). Due to this huge gap between the data acquisition poses, any scan matching would fail without proper initialization. For this reason, we gave to the tracker module the ground truth, corrupted by noise, as initial guess. The given noise is uniformly distributed in the x and y directions, within the range $\pm 1cm$. We achieved translation error of about 1 to 5 millimeters, with a processing time much faster than each LiDAR scan period (10 seconds of pre-filtering and 10 seconds of tracking against 152.5 seconds needed to acquire each scan). Moreover, for each scan we correctly identified the ground plane, which is distant around 1 meter from the LiDAR used. The map reconstructed with ART-SLAM can be seen in Fig 7.

IV. CONCLUSIONS AND FUTURE WORKS

We have proposed ART-SLAM, a fast and ground-optimized LiDAR odometry and mapping method, able to perform pose estimation of moving robots in complex environments. Its

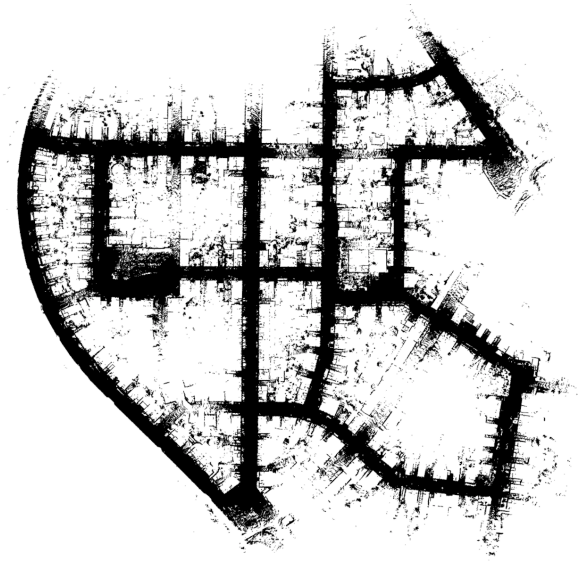

Fig. 5. Map corresponding to Sequence 00 of the KITTI odometry dataset [17], built by the proposed algorithm.

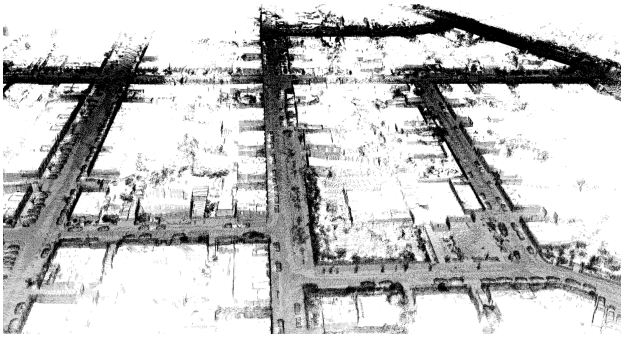

Fig. 6. Detailed area of the map corresponding to Sequence 00 of the KITTI odometry dataset [17], built by the proposed algorithm.

modular design allows it to include many upgrades w.r.t. existing similar systems, such as pre-tracking, smart loop closure, and optimized loop detection. The proposed method is evaluated on a series of datasets corresponding to multiple environments, of different sizes and complexities. Results show that ART-SLAM can achieve similar or better accuracy when compared with the state-of-the-art, with reduced computational cost w.r.t. high accuracy systems. The proposed method proves to achieve real-time performance, while being as accurate as full point clouds scan matching methods.

Due to its nature, ART-SLAM can be easily improved and extended, allowing for multiple future works. Tracking can be performed in a more robust way by enhancing the pre-filtering process to remove moving objects in the scene, such as cars or bicycles, using the method proposed by Postica et. al. [20]. Moreover, the whole system can be aided by using additional data, such as already available 2D maps or existing submaps (e.g., coming from SLAM on overlapping trajectories), to be integrated in the pose graph. Lastly, the system can be modified to handle input point clouds coming from multiple sensors.

TABLE V
COMPARISON OF THE PROCESSING TIME [MS], PER FRAME, OF THE VARIOUS MODULES, BETWEEN THE VARIANTS OF ART-SLAM.

Processing time (per frame) [ms]	Sequence	Pre-filterer	Tracker	Floor detector	Loop detection	Graph optimization
ART-SLAM	KITTI 00	18.627	39.462	25.976	19.301	16.330
ART-SLAM (SC)		18.627	39.462	25.976	9.380	17.791
ART-SLAM (IMU)		21.667	39.462	25.976	13.747	14.486
ART-SLAM (GPS)		18.627	39.462	25.976	14.681	12.733
ART-SLAM	KITTI 07	18.627	39.462	25.976	10.226	1.320
ART-SLAM (SC)		18.627	39.462	25.976	7.239	1.598
ART-SLAM (IMU)		21.667	39.462	25.976	9.808	1.710
ART-SLAM (GPS)		18.627	39.462	25.976	10.407	1.331
ART-SLAM	KITTI SHORT	18.627	39.462	25.976	6.819	0.132
ART-SLAM (SC)		18.627	39.462	25.976	6.819	0.132
ART-SLAM (IMU)		21.667	39.462	25.976	6.720	0.091
ART-SLAM (GPS)		18.627	39.462	25.976	9.085	0.165

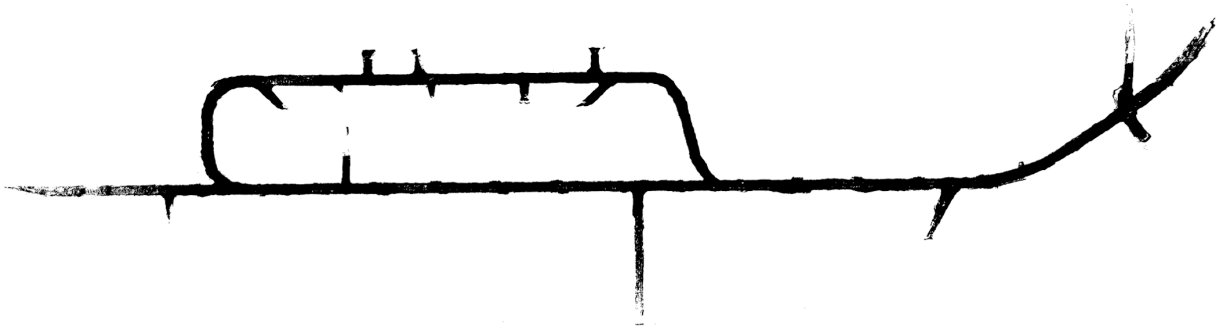

Fig. 7. Map corresponding to the Chilean underground mine dataset [19], built with the proposed algorithm. The density of the map is related to the high number of 3D points acquired by each LiDAR scan (around 25M points each).

REFERENCES

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

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

[3] K. Koide, M. Yokozuka, S. Oishi, and A. Banno, "Voxelized gicp for fast and accurate 3d point cloud registration," *EasyChair Preprint*, no. 2703, 2020.

[4] J. Zhang and S. Singh, "Loam: Lidar odometry and mapping in real-time." in *Robotics: Science and Systems*, vol. 2, no. 9, 2014.

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

[6] T. Shan, B. Englot, D. Meyers, W. Wang, C. Ratti, and D. Rus, "Lio-sam: Tightly-coupled lidar inertial odometry via smoothing and mapping," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2020, pp. 5135–5142.

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

[8] M. Pierzchała, P. Giguère, and R. Astrup, "Mapping forests using an unmanned ground vehicle with 3d lidar and graph-slam," *Computers and Electronics in Agriculture*, vol. 145, pp. 217–225, 2018.

[9] G. Grisetti, R. Kümmerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, no. 4, pp. 31–43, 2010.

[10] K. Koide, J. Miura, and E. Menegatti, "A portable 3d lidar-based system for long-term and wide-area people behavior measurement," *IEEE Trans. Hum. Mach. Syst*, 2018.

[11] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, no. 3.2. Kobe, Japan, 2009, p. 5.

[12] E. Piazza, A. Romanoni, and M. Matteucci, "Real-time cpu-based large-scale three-dimensional mesh reconstruction," *IEEE Robotics and Automation Letters*, vol. 3, no. 3, pp. 1584–1591, 2018.

[13] P. Biber and W. Straßer, "The normal distributions transform: A new approach to laser scan matching," in *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, vol. 3. IEEE, 2003, pp. 2743–2748.

[14] G. Kim and A. Kim, "Scan context: Egocentric spatial descriptor for place recognition within 3D point cloud map," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, Madrid, Oct. 2018.

[15] F. Amigoni, M. Reggiani, and V. Schiaffonati, "An insightful comparison between experiments in mobile robotics and in science," *Autonomous Robots*, vol. 27, no. 4, pp. 313–325, 2009.

[16] G. Grisetti, R. Kümmerle, H. Strasdat, and K. Konolige, "g2o: A general framework for (hyper) graph optimization," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China*, 2011, pp. 9–13.

[17] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the kitti vision benchmark suite," in *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[18] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, "Vision meets robotics: The kitti dataset," *International Journal of Robotics Research (IJRR)*, 2013.

[19] K. Leung, D. Lühr, H. Houshiar, F. Inostroza, D. Borrmann, M. Adams, A. Nüchter, and J. Ruiz del Solar, "Chilean underground mine dataset," *The International Journal of Robotics Research*, vol. 36, no. 1, pp. 16–23, 2017.

[20] G. Postica, A. Romanoni, and M. Matteucci, "Robust moving objects detection in lidar data exploiting visual cues," in *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2016, pp. 1093–1098.