MCS-SLAM: Multi-Cues Multi-Sensors Fusion SLAM

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

***Abstract*— Simultaneous localization and mapping (SLAM) is one fundamental topic in robotics due to its applications in autonomous driving. Over the last decades, many systems have been proposed, working on data coming from different sensors, such as cameras or LiDARs. Although excellent results were reached, the majority of these methods exploit the data as is, without extracting additional information or considering multiple sensors simultaneously. In this paper, we present MCS-SLAM, a Graph SLAM system that performs sensor fusion by exploiting multi-cues extracted from sensor data: color/intensity, depth/range and normal information. For each sensor, motion estimation is achieved through minimization of the pixel-wise difference between two multi-cue images. All estimates are then collectively optimized to achieve a coherent transformation. Point clouds received as input are also used to perform loop detection and closure. We compare the performance of the proposed system with state-of-the-art point cloud-based methods, LeGO-LOAM-BOR, LIO-SAM, HDL and ART-SLAM, and show that the proposed algorithm achieves less accuracy than the state-of-the-art, while needing much less computational time. The comparison is made by evaluating the estimated trajectory displacement, using the KITTI dataset.**

I. Introduction

To accurately navigate through the environment, robots must perform trajectory estimation and map construction, as these are essential tasks in many real-world applications, such as autonomous driving or re-localization. Over the last decades, many systems have been proposed, to solve the simultaneous localization and mapping (SLAM) problem. These methods can be classified into three categories, depending on the main sensor(s) used: vision-based, if the considered sensor is a camera; point cloud-based, or LiDAR-based, if data comes from a laser rangefinder; and hybrid systems, if the input has multiple origins.

Vision-based systems exploit visual information, either sparse or dense, contained in consecutive images, to track the position of the robot and easily detect loops in the estimated trajectory. Among them, one can find DSO-SLAM [1], ProSLAM [2] or the very recent ORB-SLAM3 [3], which handles data coming from monocular, stereo, or RGB-D cameras, using pin-hole and fish-eye lens models. Although Visual SLAM systems provide very good results, they are prone to errors due to their sensitivity to light changes, low textured environments and weather conditions.

*This work was not supported by any organization

[1]Matteo Frosi (*corresponding author*) 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

On the other hand, LiDAR-based methods can capture and represent the environment with a high level of detail, because of the density of collected points, and they are not affected by the issues of vision-based systems. Tracking performed with point clouds is more accurate and stable than its visual counterpart and it consists in matching two consecutive point clouds to find the best alignment between them, procedure also known as scan matching. Some systems, such as LeGO-LOAM [4] and LIO-SAM [5], perform scan matching by comparing 3D features extracted from point clouds, including edges, planes or clusters, with the latter also requiring high frequency IMU data. Other methods, including HDL [6] and the very recent ART-SLAM [7], perform, instead, scan matching on full point clouds (or a downsampled version), evaluating the point-to-point alignment. Feature-based systems are less accurate than full point cloud-based methods, although they are faster. Nevertheless, achieving real-time performance in LiDAR-based SLAM algorithms, while maintaining high accuracy, remains an open quest.

To combine the advantages of Visual and LiDAR SLAM, the focus shifted to hybrid systems, which couple the data coming from different sensors, with the goal of achieving accurate results in real-time. The work in [9] presents an RGB-D camera with LiDAR EKF SLAM, with the purpose of tackling the issue of unsuccessful visual tracking. If visual tracking fails, the LiDAR pose is used to localize the point cloud data of the RGB-D camera, to build a 3D map. In Limo [10], LiDAR measurements are used for depth extraction, as 3D points are projected on the corresponding RGB images, which are employed later in keyframe-based bundle adjustment. Zhu et al. [11] developed a 3D laser SLAM system associated with a visual method to perform loop detection through a keyframe-based technique, using visual bags-of-words. The work in [12] proposed to use both visual and LiDAR measurements by first running in parallel SLAM for each modality, and then coupling the data.

Hybrid systems give accurate results, but they do not truly target sensor fusion, as one type of data is often used to improve the quality of an existing system based on another type of data (e.g., point clouds to aid Visual SLAM methods or images to improve LiDAR SLAM efficiency). The closest work to sensor fusion is [13], where graph optimization is performed using a specific cost function, considering both laser and feature constraints. Graph-based approaches are widely used in SLAM literature, due to their advantages. The estimated trajectory is modeled as a graph, as described by Grisetti et. al. in [14], where relationships between sensor data and/or observations from the environment can be modeled as edges. Moreover, a great number of frameworks for

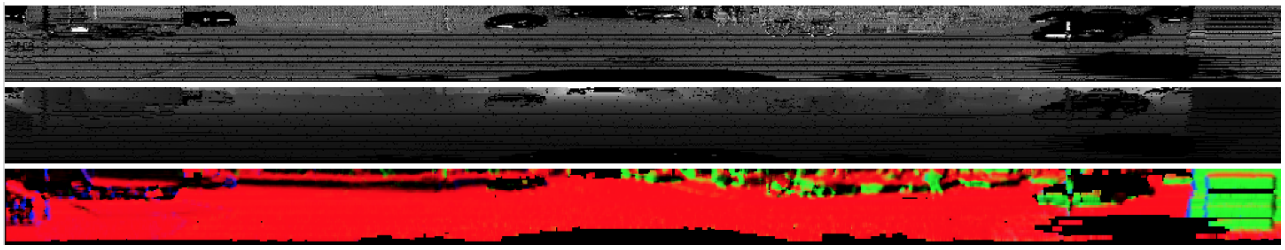

Fig. 1. From top to bottom: intensity, range and normal images extracted from a point cloud of the KITTI odometry dataset [8]. The images cover the field of view of the LiDAR sensor used to scan the environment, i.e., 360 degrees horizontally and 26.8 degrees vertically.

efficient graph optimization currently exist, which translates in the optimization of the estimated trajectory.

Not only sensor fusion is hardly achieved in literature, but tracking, i.e., data association, is most of the time explicit, as it heavily depends on the type of data used: feature detection and extraction for images and 3D point or feature scan matching for point clouds. Della Corte and Bogoslavskyi et al. [15] recently proposed a general algorithm for multi-cue photometric registration of 3D point clouds, designed without considering a specific sensor, nor a particular cue, which represents information about sensor data. The method they proposed, to perform odometry estimation rather than SLAM, uses multiple cues from input data (either RGB-D images or LiDAR point clouds), namely color, range or depth, and normals direction, as in Fig. 1. By doing this, the algorithm avoids an explicit point-to-point data association and is able to compute the transformation between view points under realistic disturbances from an initial motion guess. However, as the length of the trajectory increases, the method drifts, being for odometry estimation only. Moreover, the approach can only handle one sensor at a time (e.g., RBG-D camera or LiDAR), and it is not able to exploit common sensor suites installed on autonomous vehicles.

For these reasons, in this paper we propose an extension of the work in [15] to multiple sensors, and we integrate it into a Graph SLAM framework to perform fast and accurate multi-sensor and multi-cue SLAM, with the following contributions. The proposed system, named MCS-SLAM (Multi-Cues Multi-Sensors Fusion SLAM) performs tracking by integrating data coming from multiple sensors, while avoiding explicit data association. The method is also capable of efficiently detecting and closing loops, using a multiple step algorithm, and it optimizes the estimated trajectory through the g2o optimization framework, allowing for the inclusion of corrective data, such as GPS or IMU. MCS-SLAM presents a high degree of modularity, due to its architecture, described in Section II, which can be easily integrated and improved.

II. MCS-SLAM

A high-level overview of the system architecture of the proposed system can be seen in Fig. 2. MCS-SLAM is made up of distinct modules, the gray boxes in the figure, which represent the core of the system. The current implementation

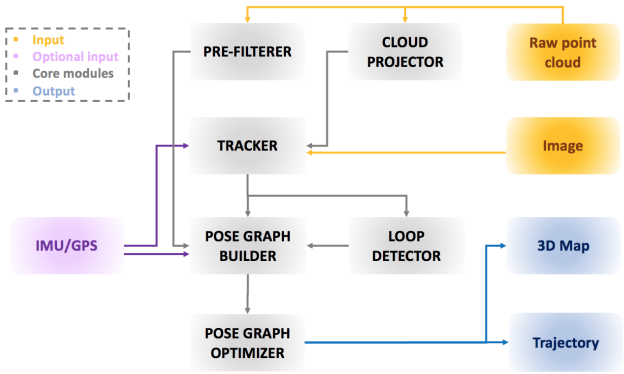

Fig. 2. High-level architecture of the proposed system.

of MCS-SLAM requires point clouds as mandatory input, as loop closure involves scan matching between a pair of clouds. Nevertheless, the proposed method can be extended to perform a different type of loop detection, thus removing the need for LiDAR data as a whole.

Given an incoming laser scan, the first step is to process it, in the *pre-filterer* and in the *cloud projector*. The first reduces the size of the cloud, downsampling it, and it removes noisy elements. The filtered point cloud is used later for efficient loop detection, to correct the estimated trajectory. The cloud projector, instead, converts the 3D point cloud into two 2D images. One represents the ranges of all elements of the point cloud, while the other displays their intensities.

The two images are used by the core components of the system, the *tracker*, which estimates the current displacement of the robot by performing two consecutive steps. First, pairs of images derived from one or multiple sensors (e.g., two LiDARs, a LiDAR and a camera, an RGB-D camera and a LiDAR) are processed in parallel and independently one from the other, to obtain a rough estimate of the current motion. Then, these estimates are jointly optimized to get the transformation which best satisfies the constraints imposed by the multiple sensors.

The current pose estimate is sent, along with its corresponding filtered point cloud, to the *loop detector* module, which tries to efficiently find loops between new and previous point clouds, via scan-to-scan matching. Poses and loops are used to build the pose graph, which represents

the estimated trajectory of the robot. Lastly, the pose graph is optimized to increase the poses accuracy. *IMU and GPS* data (pink boxes in Fig. 2) can also be integrated both in the tracker module, to give it an initial guess for improving the estimated motion, and in the *pose graph builder* module, to increase the accuracy of the estimated trajectory of the robot.

A. Pre-filtering

The purpose of the pre-filterer module is to reduce the number of elements of a point cloud, while also removing noisy data and outliers. In LiDAR SLAM systems, pre-filtering is almost a mandatory step, performed to avoid scan matching between large point clouds, which would be computationally intensive. Downsampling can reduce the input size by a factor of five or even more, if necessary, while retaining the spatial structure of the initial scan. In MCS-SLAM, pre-filtering is not used for tracking, where having dense and large point clouds is, instead, a benefit, but for loop closure. Indeed, scan-to-scan matching is one of the steps followed to detect loops, and it is essential to have reduced clouds, to achieve detection in reasonable time.

B. Cloud projection

As the name implies, the cloud projector module has the purpose of converting a point cloud in 2D images, representing range and intensity. Projection is achieved through the spherical projection model, which best captures the characteristics of laser rangefinder sensors, such as LiDARs.

Let K be a camera matrix in the following form:

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{1}$$

where f_x and f_y specify, respectively, the resolution of azimuth and elevation, and c_x and c_y their offset in pixels. Then, the spherical projection of a 3D point $P = [p_x, p_y, p_z]^T$ is given by:

$$proj(P) = K \begin{bmatrix} atan2(p_y, p_x) \\ atan2(p_z, \sqrt{p_y^2 + p_x^2}) \\ 1 \end{bmatrix} \tag{2}$$

C. Tracking

Tracking, also known as short term data association, is the task of finding the relative motion between two consecutive poses of the robot. The tracker module operates in two consecutive phases: standalone motion estimation and collective motion optimization.

The *standalone motion estimation* phase is performed on all data coming from multiple sensors, independently from each other. In particular, for each sensor, the pipeline described in [15] is executed, to obtain multiple, independent, rough estimates of the robot motion. As a consequence, considering S sensors, we are able to obtain S estimates by running the algorithm presented in [15], in parallel.

This approach seeks to register either two observations with respect to each other or an observation against a 3D model. Sensor observations are normalized into a 2D representation, e.g., an image from a regular camera, or a range image from a LiDAR (as explained previously in Subsection II-B). This representation consists in a multi-cue image, where each pixel contains different types of information, i.e., light intensity, depth information of surface normals, as represented in Fig. 1.

Each measurement is aligned to a model $M = \{P_i, i = 0 \dots N\}$, for which 3D information is available in the form of a point cloud, enriched with data associated to the different cues. As in photometric error minimization approaches, the method tries to iteratively minimize the pixel-wise difference between the current multi-cue image I and the predicted image $\hat{I}(M, X)$, the latter being a multi-cue image obtained by projecting the model M onto a virtual camera located at the estimated pose P_X. X is the transformation matrix that transforms the points of M from the global into the local camera coordinate system.

The goal is to find the best transformation

$$\begin{aligned} X^* &= \underset{X}{argmin} \sum_{u,v,c} \left\| \hat{I}^c_{u,v}(M, X) - I^c_{u,v} \right\|^2_{\Omega^c} \\ &= \underset{X}{argmin} \sum_{u,v,c} e^c_{u,v}(M, X)^T \, \Omega^c \, e^c_{u,v}(M, X) \end{aligned} \tag{3}$$

where $e^c_{u,v}$ is the error at pixel (u, v) between the predicted value $\hat{I}^c_{u,v}$ and the measured value $I^c_{u,v}$ for a particular index c associated to a cue, and $\Omega = diag(\{\Omega^c\})$ is a block diagonal information matrix used to weight the different cues of the image. Instead of solving the problem in Eq. 3, the approach in [15] re-formulates it as a linear system, where, instead of finding the transformation X^* that yields the best result, a perturbation Δx is calculated iteratively, such that

$$X^* = X \oplus \Delta x \tag{4}$$

where X is an initial guess of the transformation and Δx is a vector with six elements, corresponding to the difference in translation and orientation.

This formulation comes from the Taylor expansion of the error described above, combined with Eq. 4:

$$e^c_{u,v}(X \oplus \Delta x) \backsimeq e^c_{u,v}(X) + \left. \frac{\delta e^c_{u,v}(X \oplus \Delta x)}{\delta x} \right|_{\text{x=0}} \Delta x \tag{5}$$

$$= e^c_{u,v}(X) + J^c_{u,v}(X) \Delta x \tag{6}$$

The minimization problem then becomes

$$\Delta x^* = \underset{\Delta x}{argmin} \sum_{u,v} w_{u,v} \sum_{c} \left\| e^c_{u,v}(X) + J^c_{u,v}(X) \Delta x \right\|^2_{\Omega^c} \tag{7}$$

with $w_{u,v}$ being a regularization weight. This, in turn, is equivalent to solve the linear system $H \Delta x^* = b$, with the terms H and b, respectively given by

$$H = \sum_{u,v} w_{u,v} \sum_{c} J^c_{u,v}(X)^T \Omega^c J^c_{u,v}(X) \tag{8}$$

$$b = \sum_{u,v} w_{u,v} \sum_{c} J^c_{u,v}(X)^T \Omega^c e^c_{u,v}(X) \tag{9}$$

This procedure is done incrementally and on multiple scales of the input 2D images, to accurately find the best Δx^*, while avoiding falling into local minima.

As stated in the introduction of this subsection, in MCS-SLAM, this pipeline is executed once for each sensor, in parallel, such that data coming from sensor i, e.g. a LiDAR, generates a motion estimate $trans_i$. Once all estimates are computed, the tracker proceeds to the next phase, namely the *collective motion optimization*, taking also into account the rotational offsets of the various sensors. We do not consider the translation part of each offset, as we are working in the field of autonomous driving and all sensors are rigidly attached to the robot/vehicle (meaning that the motion is the same for all sensors).

First, all motion estimates associated to the multiple sensors are weighted and summed, with each weight w_i being inversely proportional to the re-projection error previously computed as part of the procedure to find transformation $trans_i$, described above. In this way, a unique motion of the robot is obtained, combining the tracking information associated to each sensor. This value is then fed again to each standalone motion estimation pipeline, as initial guess, and optimized only for one iteration. These two steps form a single iteration of the collective motion optimization, and are continuously repeated, until convergence, or up to N times.

The standard approach to perform a sort of sensor fusion is to adopt a Kalman Filter-based solution. Although easy to implement, a Kalman Filter does not perform any kind of post-optimization. This is usually needed, as the poses estimated, associated to each sensor, may differ from each other, especially if coming from data gathered by unreliable sensors. A Kalman Filter would give more weight to the reliable estimates, and it would compute a new pose (and relative motion), which, fed to the tracking modules of the unreliable sensors, would lead to convergence problem and increasing inaccuracies. In MCS-SLAM, instead, the combination of standalone motion estimation and collective motion optimization allows to obtain motion estimates smoothed among all sensors, taking into account their overall accuracy and also avoiding abrupt changes in the estimated poses, independently from the sensor considered.

The tracker of MCS-SLAM adopts a keyframe-based approach to estimate the trajectory of the robot. *Keyframes* are data structures that describe the motion of the robot in particular locations of its trajectory, and they contains multiple useful quantities. MCS-SLAM, differently from the majority of SLAM systems in literature, works using two types of keyframes: one for tracking and one to be used for loop closure and pose graph construction and optimization.

Keyframes associated to the tracker hold the current estimated position of the robot and the corresponding multi-cue images. For each sensor i, we keep track of the current keyframe $K_i^{tracker}$, as it has no other purpose than to avoid useless computations. Keyframes related to the other modules (i.e., loop closure detection and pose graph building and optimization) contain a point cloud and the pose (odometry) estimated by the tracker, along with timestamp of the cloud, accumulated distance from the beginning and, if available, camera images. We refer to these keyframes with K_j^{graph}, since they are computed following the collective optimization phase and are mainly used in the pose graph.

The first tracker keyframe, for each sensor, and the first graph keyframe, correspond to the first point cloud received by the system. Consecutive keyframes (this applies independently from the type) must satisfy at least one of the following criteria:

- The distance between the frames is greater than a user-defined threshold, $\Delta trans$, in meters
- The rotation between the frames is greater than a user-defined angle, $\Delta orientation$, in radians

The thresholds $\Delta trans$ and $\Delta orientation$ depend on the dataset and the type of trajectory to be estimated. Keyframes $K_i^{tracker}$ correspond to very low thresholds, as we want to compare images close in time, to avoid accumulating initialization errors. In our experiments we use $\Delta trans$ and $\Delta orientation$ of, respectively, 50 centimeters and 1 degree. The poses obtained by the tracker after the collective optimization are then filtered through higher thresholds, to obtain multiple graph keyframes K_j^{graph}, which are stored in the pose graph. Again, in our experiments, we set $\Delta trans$ to 5 meters and $\Delta orientation$ to 5 degrees.

All these parameters should be tuned accordingly to the scenario considered. For example, to handle an indoor environment, one would use low thresholds, as robots moving through it are relatively slow. On the other hand, an outdoor scenario would be associated to parameters with high values, as robots navigate it at higher speed than the indoor case.

D. Loop detection

When the robot passes through a location that has been previously visited, it forms a loop in its trajectory. Loops are additional useful constraints to insert into the pose graph, allowing the correction of drifts and estimation errors. In the current MCS-SLAM implementation, detection of loops involves a multiple step method, strictly associated to the availability of point clouds, which are mandatory inputs in the current implementation of the system, and it involves only the K^{graph} keyframes. In the following, we drop the suffix $graph$, referring to keyframes only as K_j.

Whenever a keyframe K_{query} is created, it is compared with all the other keyframes, which are candidates for loop closure. To make loop detection scalable, an odometry-based filtering is performed. A pair K_{query} and $K_{candidate}$ is kept only if the two keyframes correspond to estimated odometries far in time but close in location (within a threshold range). If K_{query} and $K_{candidate}$ satisfy these constraints, they probably correspond to a loop closure.

Once all the loop candidates have been discovered, they are further compared using the approach described in [17], which converts the corresponding point clouds in a bird-eye view grid and selects the pairs with most similar cells. At the end of this step, only k candidate pairs for loop closure remains, to be used in the next step. This few number of candidates is then compared using scan-to-scan matching,

Fig. 3. Range images obtained through the spherical projection of a point cloud split in two, from City Sequence 05 of the KITTI raw dataset [16]. The image on the left corresponds to the environment in front of the LiDAR acquiring the scan (180 degrees), while the image on the right represents the part of the scene behind the sensor (again, 180 degrees).

Fig. 4. From left to right, RGB image, sparse range image obtained through LiDAR-to-camera projection and enhanced range image, from city Sequence 05 of the KITTI raw dataset [16]. The enhancement is achieved through a simple bilinear filtering, along with manual completion.

to find the alignment between the point clouds of each pair. Then, all the transformations obtained are compared. If found, the transformation corresponding to the smallest distance and highest accuracy represents a loop, which is then added to the pose graph as a new constraint.

E. Pose graph building and optimization

MCS-SLAM is a Graph SLAM [14] system, where the poses of the robot are modeled as nodes in a graph, named pose graph, and edges represent spatial constraints resulting from tracking or measurements coming from different sensors, e.g., IMU or GPS. Moreover, each node j is associated to the corresponding keyframe K_j^{graph} and edges can be added also when performing loop detection and closure, between non-consecutive nodes in the graph. Periodically, the pose graph is optimized to best satisfy the constraints provided by the measurements associated to each edge.

III. Experimental validation of the system

The proposed system has been compared against four methods for point cloud-based SLAM: LeGO-LOAM-BOR (which is an improved variant of LeGO-LOAM [4]), LIO-SAM [5], HDL [6] and ART-SLAM [7]. IMU data, which is mandatory in LIO-SAM, is also used in ART-SLAM, to deskew point clouds and to enforce rotational constraints in the pose graph. We evaluate MCS-SLAM in two scenarios coming from the KITTI dataset [8] [16], corresponding to a short path without loops and a medium sequence with one closure at the end of the trajectory.

To test the proposed system, we run it under five different conditions. In the first case, from a single point cloud covering all the surrounding environment (360 degrees horizontal field of view), we generate a single multi-cue image, having no color information (meaning that we consider only range and normal cues). In the second and third scenarios, we split the point cloud in two and use only the points, respectively, in front and behind the LiDAR, covering a horizontal field of view of 180 degrees each, obtaining the range images represented in Fig. 3. It should be noticed that the first three experimental campaigns involve only a single sensor, and they are used to evaluate the SLAM part of MCS-SLAM.

The fourth case considers the point clouds, from the second and third scenarios, as coming from two different sensors. This way, for each input, the standalone motion estimation is performed, followed by the collective motion optimization, described in Subsection II-C (differently from the first three cases, where only the standalone motion estimation is needed). Lastly, in the fifth scenario, we project LiDAR data onto the corresponding RGB image, obtaining a large multi-cue image (color and range). As it can be seen in the central picture of Fig. 4, the projected cloud is sparse w.r.t. the colored image. For this reason, we first apply windowed bilateral filtering on the sparse range image, then we fill each column with the highest range value in them, obtaining the right picture of Fig. 4. The fourth and fifth experimental campaigns involve two sensors (two pseudo LiDAR and a LiDAR coupled with a camera, respectively), and they are used to evaluate the data and sensor fusion capabilities of MCS-SLAM.

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

A. Comparison and results

To evaluate the systems, we compute the absolute trajectory error (ATE) and show the processing time, per frame, of the core modules of MCS-SLAM. In particular, the ATE measures the difference between coordinates of the points belonging to the true and the estimated trajectory. Because Graph SLAM systems correct only the poses associated to keyframes, we perform a further processing step before evaluation: we associate the keyframes to the ground truth positions using timestamps and data indices.

Fig. 5 shows the estimated trajectories on Sequence 07 of the KITTI odometry dataset [8]. All the evaluated methods present a high degree of accuracy, following the ground truth trajectory and easily detecting the loop at the end of the path. Table I further details the obtained results, as it represents the mean, root mean squared error (RMSE) and standard deviation (STD) of the absolute trajectory error, in meters. The proposed system presents an accuracy within an acceptable threshold of about 1.8 meters, which is similar to the LeGO-LOAM-BOR. This result was expected, as

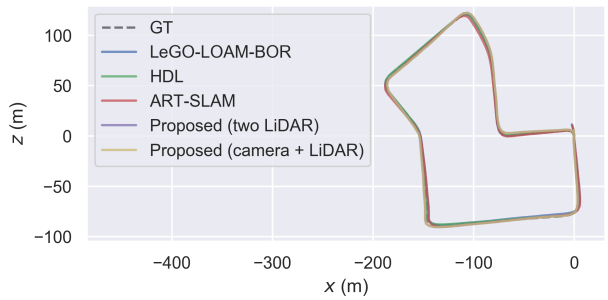

Fig. 5. Comparison between the trajectories estimated by LeGO-LOAM-BOR (derived from LeGO-LOAM [4]), HDL [6], ART-SLAM [7] and the proposed system (in the multiple sensors scenarios), on Sequence 07 of the KITTI odometry dataset [8]. The other methods considered in the evaluation are not included, to avoid further overlapping.

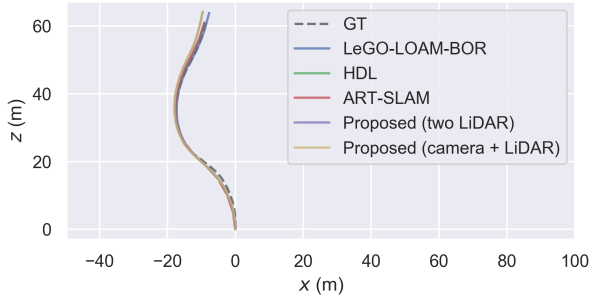

Fig. 6. Comparison between the trajectories estimated by LeGO-LOAM-BOR (derived from LeGO-LOAM [4]), HDL [6], ART-SLAM [7] and the proposed system (in the multiple sensors scenarios), on City Sequence 05 of the KITTI raw dataset [16]. The other methods considered in the evaluation are not included, to avoid further overlapping.

TABLE I

ATE ON SEQUENCE 07 OF THE KITTI ODOMETRY DATASET [8].

ATE [m]	MEAN	RMSE	STD
LeGO-LOAM-BOR	1.604	1.807	0.832
LIO-SAM (mandatory IMU)	0.509	0.675	0.351
HDL	0.954	1.253	0.767
ART-SLAM (with IMU)	**0.343**	**0.366**	**0.127**
MCS-SLAM (one LiDAR)	1.679	2.219	1.137
MCS-SLAM (one LiDAR, front)	1.894	2.392	1.215
MCS-SLAM (one LiDAR, back)	1.917	2.421	1.054
MCS-SLAM (two LiDAR)	1.714	1.892	0.798
MCS-SLAM (camera + LiDAR)	2.822	2.924	1.957

TABLE II

ATE ON CITY SEQUENCE 05 OF THE KITTI RAW DATASET [16].

ATE [m]	MEAN	RMSE	STD
LeGO-LOAM-BOR	1.094	1.169	0.409
LIO-SAM (mandatory IMU)	**0.493**	**0.338**	**0.280**
HDL	0.893	0.912	0.476
ART-SLAM (with IMU)	0.746	0.814	0.326
MCS-SLAM (one LiDAR)	0.679	0.807	0.437
MCS-SLAM (one LiDAR, front)	0.561	0.639	0.305
MCS-SLAM (one LiDAR, back)	0.708	0.806	0.383
MCS-SLAM (two LiDAR)	0.578	0.659	0.317
MCS-SLAM (camera + LiDAR)	1.341	1.551	0.892

tracking performed through MCS-SLAM loses precision due to the projection of point clouds in 2D images, which are later back-projected in 3D clouds, as described in [15]. Moreover, all results associated to the five considered scenarios behave as expected, with the one LiDAR case being the most accurate, followed by the fusion of front and back. The camera plus LiDAR setup is less accurate, as range image densification is achieved through a simple bilinear filtering and manual completion.

To evaluate the accuracy when loop closures are not available, we also considered a short sequence. As the dataset corresponding to raw odometries does not have a ground truth, we use, instead, GPS data, provided along with the point clouds and RGB images. Fig. 6 represents the estimated trajectories on City Sequence 05 of the KITTI raw dataset [16]. As for the medium-length sequence, all the methods considered for evaluation tightly follow the trajectory, even if no loop closure is available to correct drifts and tracking problems, even though for shorter sequences there is not enough room to accumulate such errors. Table II shows the ATE statistics, in meters. As before, all systems show good results, with the proposed method performing almost as well as the method with best accuracy (with a difference of about ten centimeters). These results are also motivated by the fact that the trajectory is relatively simple, almost straight with very smooth turns.

Among all the systems used for comparison, only HDL is not able to run real-time, being two to three times slower than the data acquisition rate (which is one frame every 100 milliseconds). LeGO-LOAM-BOR and LIO-SAM are designed to perform real-time SLAM, with the latter being even faster (the average processing time ranges from 35 to 50 milliseconds). Moreover, as described in [7], also ART-SLAM is able to achieve real-time results, even on long sequences. Table III shows the average processing time, per frame, of MCS-SLAM, in all five scenarios considered in the evaluation, for both sequences. MCS-SLAM, despite being less or as accurate as the other methods, is able to run more than five times faster than the data acquisition rate. It should be noticed that tracking is performed in parallel to the whole loop detection and graph construction and optimization procedure, proving, once again, that MCS-SLAM can be faster than real-time.

Intuitively, the split single-sensor scenarios (back and front) are associated to the lowest runtime, as we are dealing with small images (half width than the image obtained in the one LiDAR case). The processing time associated to the camera plus LiDAR setup may surprise, being the worst of all, but it is due to the densification of the projected point cloud onto the RGB image, operation that is quite intensive due to the size of the image to fill (1242x375 pixels) and the approach adopted (bilinear filtering). This step is not optimized and can be sped up using other, more efficient, algorithms. Nevertheless, it is clear that MCS-SLAM is

TABLE III

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

Processing time (per frame) [ms]	Sequence	Pre-processing	Tracking	Loop detection	Graph optimization
MCS-SLAM (one LiDAR)	KITTI 07	10.172	22.648	10.226	1.320
MCS-SLAM (one LiDAR, front)		8.907	13.893	10.226	1.320
MCS-SLAM (one LiDAR, back)		8.907	13.753	10.226	1.320
MCS-SLAM (two LiDAR)		9.371	16.393	10.226	1.320
MCS-SLAM (camera + LiDAR)		149.823	19.712	10.226	1.320
MCS-SLAM (one LiDAR)	KITTI SHORT	10.172	22.648	6.819	0.132
MCS-SLAM (one LiDAR, front)		8.907	13.597	6.819	0.132
MCS-SLAM (one LiDAR, back)		8.907	13.642	6.819	0.132
MCS-SLAM (two LiDAR)		9.371	17.241	6.819	0.132
MCS-SLAM (camera + LiDAR)		155.712	21.712	6.819	0.132

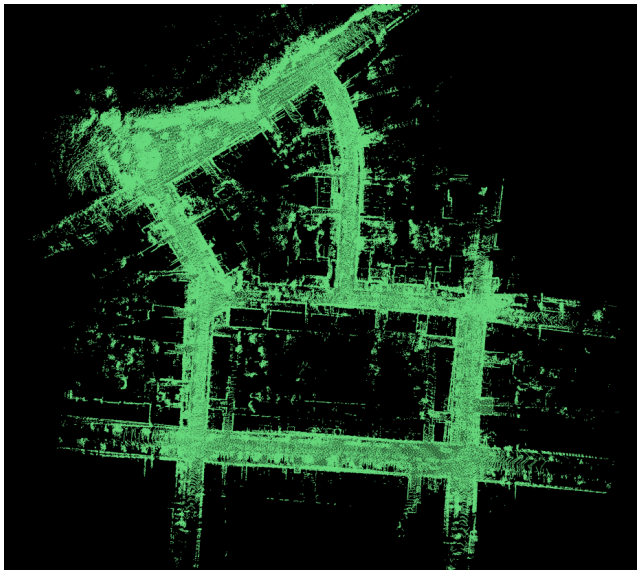

Fig. 7. Map obtained by MCS-SLAM with two LiDAR, on Sequence 07 of the KITTI odometry dataset [8].

definitely faster than the other methods.

Lastly, we include a visual evaluation of the results obtained. Fig. 7 shows the map obtained by MCS-SLAM, in the two LiDAR scenario, of Sequence 07 of the KITTI odometry dataset [8].

IV. CONCLUSIONS

In this paper, we have proposed MCS-SLAM, a Graph SLAM system that is able to perform multi-cue and multi-sensor pose tracking faster than real-time (meaning faster than the data acquisition rate). Differently from systems available in literature, MCS-SLAM is able to effectively fuse data coming from multiple sensors, and to exploit different cues extracted from data itself, i.e., intensity, depth or range, and normal to the surface. This is achieved through two steps: standalone motion estimation and collective motion optimization, involving all sensors considered. The proposed system is improvable and extendable, due to the independent nature of its modules. Lastly, it shows a remarkable speed up w.r.t. state-of-the-art algorithms, making it suitable for real-time applications.

REFERENCES

[1] J. Engel, V. Koltun, and D. Cremers, "Direct sparse odometry," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 3, pp. 611–625, 2017.

[2] D. Schlegel, M. Colosi, and G. Grisetti, "Proslam: graph slam from a programmer's perspective," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 3833–3840.

[3] C. Campos, R. Elvira, J. J. G. Rodríguez, J. M. Montiel, and J. D. Tardós, "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam," *IEEE Transactions on Robotics*, 2021.

[4] 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.

[5] 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.

[6] 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.

[7] M. Frosi and M. Matteucci, "Art-slam: Accurate real-time 6dof lidar slam," *IEEE Robotics and Automation Letters*, 2022.

[8] 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.

[9] Y. Xu, Y. Ou, and T. Xu, "Slam of robot based on the fusion of vision and lidar," in *2018 IEEE International Conference on Cyborg and Bionic Systems (CBS)*. IEEE, 2018, pp. 121–126.

[10] J. Graeter, A. Wilczynski, and M. Lauer, "Limo: Lidar-monocular visual odometry," in *2018 IEEE/RSJ international conference on intelligent robots and systems (IROS)*. IEEE, 2018, pp. 7872–7879.

[11] Z. Zhu, S. Yang, H. Dai, and F. Li, "Loop detection and correction of 3d laser-based slam with visual information," in *Proceedings of the 31st International Conference on Computer Animation and Social Agents*, 2018, pp. 53–58.

[12] Y. Seo and C.-C. Chou, "A tight coupling of vision-lidar measurements for an effective odometry," in *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2019, pp. 1118–1123.

[13] G. Jiang, L. Yin, S. Jin, C. Tian, X. Ma, and Y. Ou, "A simultaneous localization and mapping (slam) framework for 2.5 d map building based on low-cost lidar and vision fusion," *Applied Sciences*, vol. 9, no. 10, p. 2105, 2019.

[14] 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.

[15] B. Della Corte, I. Bogoslavskyi, C. Stachniss, and G. Grisetti, "A general framework for flexible multi-cue photometric point cloud registration," in *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2018, pp. 4969–4976.

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

[17] G. Kim and A. Kim, "Scan context: Egocentric spatial descriptor for place recognition within 3d point cloud map," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2018, pp. 4802–4809.