2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)
October 25-29, 2020, Las Vegas, NV, USA (Virtual)

Plug-and-Play SLAM: A Unified SLAM Architecture for Modularity and Ease of Use

Mirco Colosi Irvin Aloise Tiziano Guadagnino
Dominik Schlegel Bartolomeo Della Corte Kai O. Arras Giorgio Grisetti

***Abstract*—Simultaneous Localization and Mapping (SLAM) is considered a mature research field with numerous applications and publicly available open-source systems. Despite this maturity, existing SLAM systems often rely on ad-hoc implementations or are tailored to predefined sensor setups. In this work, we tackle these issues, proposing a novel unified SLAM architecture specifically designed to standardize the SLAM problem and to address heterogeneous sensor configurations. Thanks to its modularity and design patterns, the presented framework is easy to extend, maximizes code reuse and improves computational efficiency. We show in our experiments with a variety of typical sensor configurations that these advantages come without compromising state-of-the-art SLAM performance. The result demonstrates the architecture's relevance for facilitating further research in (multi-sensor) SLAM and its transfer into practical applications.**

I. Introduction

Simultaneous Localization and Mapping (SLAM) has become a mature research field and enabling technology for many applications ranging from autonomous vehicles to augmented reality. While there are robust solutions for well-posed use-cases such as laser-based localization of wheeled robots in planar environments [1], [2], there are scenarios in which either the robot, the environment or the operating conditions are so challenging that a large amount of further fundamental research is needed, as also pointed out by Cadena *et al.* [3].

SLAM with multiple sensors is one approach to address such challenging scenarios as it leverages redundant or complementary information about the environment. Typical examples include the combination of image with range data from cameras, laser range finders or RGB-D sensors and the fusion with propioceptive sensors such as Visual-Inertial Odometry (VIO). Multi-sensor SLAM has been studied in several research communities and typical state-of-the-art systems support two or more sensors at the same time. The majority of these systems, however, are meant to be used with a predefined combination of sensors and are not designed for easy extensibility with other sensors or systems. This lack of flexibility and modularization makes it difficult to analyze, for example, the performance impact of individual sensors, different sensor combinations, alternative components within a SLAM system and to compare state-of-the-art (multi-sensor) SLAM systems with each other. Such questions, however, are key for researchers to further advance the field as well as for practitioners who seek the best cost-benefit solution given a use-case.

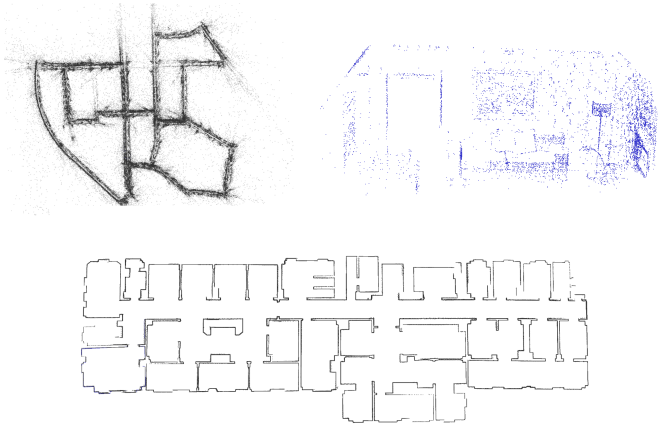


Fig. 1: Result of three different SLAM pipeline configurations set up using our architecture. On the top row, from left to right, a stereo camera Visual SLAM system running on `kitti-00`, and a RGB-D Visual SLAM pipeline performing SLAM on `icl-lr-0`. Bottom row shows the map produced by the 2D LiDAR pipeline on simulated data.

In this paper, we propose a novel unified SLAM architecture that overcomes these limitations and aims at standardizing single-sensor and multi-sensor SLAM. Building upon past work [4] in which we introduced a taxonomy of building blocks of graph-based SLAM, we extend the architecture to multiple sensors and enable users to deploy and combine sensory modalities in a plug-and-play fashion. Thanks to its modularity and separation of the core SLAM modules, the architecture improves code reuse, efficiency, and usability. New sensory cues, for example, can be integrated by simply editing a configuration file. We evaluate our architecture in several experiments using multiple 2D LiDARs, wheel odometry, stereo and RGB-D cameras, and achieve performance results similar to those of state-of-the-art systems (see also Fig. 1). The framework is open-source and written in C++[1].

The remainder of this paper is organized as follows: Sec. II discusses the literature on multi-sensor SLAM systems, Sec. III summarizes our taxonomy of SLAM building blocks which is then extended in Sec. IV to multiple sensors. Sec. V contains the experiment and results, Sec. VI concludes the paper.

Mirco Colosi, Irvin Aloise, Tiziano Guadagnino, Dominik Schlegel, Bartolomeo Della Corte, Giorgio Grisetti are with the Department of Computer, Control, and Management Engineering "Antonio Ruberti", Sapienza University of Rome, Rome, Italy, Email: {`colosi, ialoise, guadagnino, schlegel, dellacorte, grisetti`}`@diag.uniroma1.it`.

Mirco Colosi and Kai O. Arras are with Robert Bosch Corporate Research, Stuttgart, Germany. {`mirco.colosi, kaioliver.arras`}`@de.bosch.com`.

This work has been partially supported by Robert Bosch GmbH.

[1]Source code at http://srrg.gitlab.io/srrg2.html

II. Related Work

In the context of SLAM, sensor fusion indicates the capability of a system of processing multiple cues at the same time. Multi-sensor SLAM could dramatically improve the system performances in various scenarios, especially when those are highly dynamic. In the past years, the community addressed this topic, investigating ways of integrating multiple cues in the same system. A possible way of exploiting multiple cues is to have a main sensor and a supplementary one. The latter supports the system initialization or provides specific cues such as the scale. In the context of Visual-Inertial SLAM, this scenario is very common nowadays. Many state-of-the-art systems combine the use of a monocular camera and an Inertial Measurement Unit (IMU) to perform SLAM [5]–[8]. The IMU data is integrated over time [9] to produce a coarse estimate of the relative motion between two frames and to infer the scale not observable by a monocular camera. Similarly, in the work of Pire *et al.* [10], the wheel odometry computed from encoder readings might be used to provide a prior in the registration of two frames when using stereo cameras. Lately, Rosinol *et al.* [11] developed Kimera, a SLAM framework which combines camera images (either from a monocular or stereo setup) with IMU data to build 3D metric-semantic maps. In the context of LiDAR-based SLAM, Zhang *et al.* [12] proposed to integrate range measurement and RGB data to estimate the sensor motion. More specifically, the system initially computes the ego-motion through Visual Odometry (VO) (high frequency but low accuracy) and then refines it exploiting scan-matching based LiDAR Odometry (LO) (low frequency, high accuracy). Newman *et al.* [13], instead, used the additional cue coming from RGB camera to compute loop-closure through feature-based Visual Place Recognition (VPR).

Currently, the maturity of SLAM as a research field has motivated the research community to also explore system and software aspects such as standardization and modularization of SLAM systems. In this sense, highly coupled architectures entailed to specific sensor settings and operating scenarios leave room to dynamic multi-sensor systems. Our work investigates along this research direction. In this context, Schneider *et al.* [14] proposed *maplab*, a framework to manage VIO in every aspect. Therefore, *maplab* is a Visual-Inertial Mapping and Localization framework which unifies state-of-the-art VIO implementations and map management or localization routines, allowing multi-mission sessions - *i.e.* a single map is generated and refined from multiple physical data acquisitions. The authors offer various off-the-shelf implementations of state-of-the-art algorithms and provide an architecture that allows the user to integrate his own package in the framework. In particular, *maplab* allows to create a single open-loop map for every mission in VIO mode, then stores the map and performs its refinement using efficient off-line algorithms. As in our case, the user can interact with *maplab* through a console and provide its own configuration. Still, this framework is not intended to deal with multiple sensors other than a camera and an IMU.

More recently, Blanco-Claraco proposed *MOLA* [15], a modular, flexible and fully extensible SLAM architecture. MOLA combines in a single system multi-sensor capabilities and large map management while being completely customizable by the user. Examples of configuration parameters can be the type of variable that represents the system state or the *back-end* in charge of performing global optimization. MOLA has different types of independent sub-modules, each of which has a specific role. In this sense, *input* modules process raw sensor readings, and act as data sources for *front-end* modules. The latter exploit standard SLAM algorithms to create nodes and edges of the factor graph, while the *back-end* creates a unified interface to the underlying global optimization framework - that can be chosen arbitrarily. Finally, *map-storage* modules are in charge of storing and managing the map. These modules can also dynamically serialize part of the total map to reduce memory usage. MOLA allows the user to define the front-end module, by overriding handlers of keyframe and factor creation. In our work, instead, we detected some "atomic" modules and their connections to generate expected behaviors, resulting in a more structured architecture that encourages the reuse of sub-modules.

Similarly, Labbé *et al.* [16] proposed a multi-sensor graph SLAM system called RTAB-Map. The modularity is intrinsically granted by the use of Robot Operating System (ROS), by which every processing module is a ROS node. RTAB-Map was originally designed to be an appearance-based loop closure detection approach [17], that was focused on memory management to deal with long-term mapping sessions. Subsequently, RTAB-Map has been highly expanded, resulting now in a Visual/LiDAR SLAM open-source library. RTAB-Map can be used in two modalities. The first one consists of a "passive" map manager, which takes as input odometry measurements - generated by some external system - along with raw visual information. In this case, the system maintains the map, detects loop closures and provides highly efficient memory management. In the "active" modality, RTAB-Map is able to generate itself the odometry information, processing LiDAR or Visual data. In this sense, a great variety of cues can be simultaneously used by a single framework. Still, to extend the system, one has to completely develop a processing module that given raw sensor readings provides ego-motion estimation.

In Colosi *et al.* [4] we introduced a taxonomy of a generic graph-based SLAM system which identifies and defines building blocks or components that are clearly specified by their inputs, outputs and task. Here, we extend this work to multiple sensors. The partition of a SLAM system into components is also investigated in the survey of Younes *et al.* [18]. The authors propose a generic Keyframe-based SLAM (K-SLAM) flowchart made up by several building blocks. For each of them, they specify the expected functionalities and give currently available state-of-the-art implementations. However, this work is only restricted to monocular camera systems although the idea behind the architecture is reasonably general

and might be extensible to a more generic graph-based SLAM system as proposed in this paper.

III. Taxonomy of a Graph-Based SLAM System

A modern SLAM system is generally composed of a collection of modules that process a set of shared data structures. Each module is in charge of performing a relatively isolated task that takes some input data, processes them and produces some output quantities. For a graph-based SLAM approach, the outcome of a SLAM system can be represented through a factor graph [19]. In this approach, the estimated trajectory of the robot is represented through a pose graph, a specialization of a generic factor graph in which each variable represents a robot pose, while factors encode spatial constraints between a pair of poses. To avoid unbounded growth of the factor graph, nodes are generally spawned according to some heuristic, *e.g.* when the distance between poses is higher than a threshold. Variables in the graph correspond to the robot poses in these *key frames*. The system can "attach" additional information about the structure of the environment to each key frame forming a so called *landmarks*. Thus, each variable in the graph represents a rigid body that we also refer to as *local map*.

In Colosi *et al.* [4] we analyzed how a generic single-sensor SLAM system can be composed of recurring building blocks. In the remainder of this section, we review these concepts, while in the next section we extend them to multiple sensors.

A. Core Modules

The workflow of a generic SLAM system should *i)* process raw sensor readings and generate data in a canonical format for the rest of the system, *ii)* estimate the relative motion between two sensor readings, *iii)* generate a trajectory and manage landmarks to create a consistent map, and finally, *iv)* detect loop closures and perform global optimization on the factor graph. Specifically, the core modules for these tasks are as follows:

Raw Data Pre-processor: as the name suggests, this module takes as input a raw sensor reading and extracts suitable data structures that can be used in the other modules. For example, given a RGB-D image, its output may consist of 3D points with visual features attached. We use the term *measurement* for the output of this module.

Aligner: this module computes the relative motion between two comparable entities, *e.g.* two measurements or a measurement and a local map. The Aligner is agnostic to the current system state since it operates only on the two entities given as input and possibly an initial guess of their offset. An example implementation is an ICP-based algorithm for point clouds registration, *e.g.* using as fixed the one extracted from the current sensor reading and as moving the previous one or the current local map.

Tracker: is in charge of managing and updating the current local map and generating the robot pose estimate. Methods like VO or scan matching are typical instances of this module.

Graph-SLAM: arranges local maps in a factor graph, detects loop-closures and eventually triggers global optimization.

B. Support Modules

Each core module owns and uses submodules to break down its task into sets of isolated subtasks, enhancing code reusability and modularity. A core module's capabilities or behavior can vary by using support modules with different algorithms to solve their respective subtasks. The input-output definition ensures that a module's overall role in the system is unaffected when switching between different support modules. In our taxonomy, the support modules of a generic SLAM system are as follows:

Correspondence Finder: given two compatible entities, it computes associations between them. Many implementations of this module have been proposed, either based on their appearance [20], [21] or geometry [22], [23].

Merger: its task is to incorporate new measurements extracted from the current sensor reading in a local map. Depending on the inputs, different mapping approaches can be exploited also in this case, *e.g.* for scan matching algorithms a merging method is proposed in [24].

Loop Detector and Validator: these two modules are in charge of detecting loop-closures and run additional checks to reject false associations. Each accepted loop-closure will be turned to a new factor in the graph.

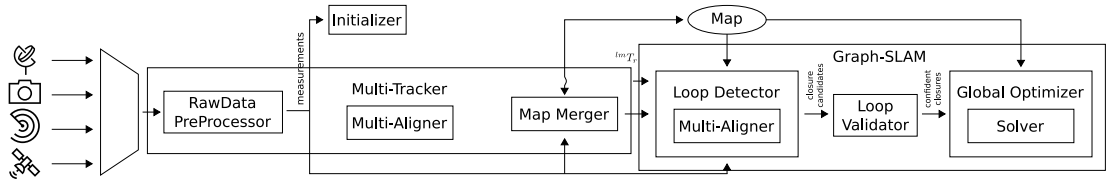
Global Optimizer: it is in charge of performing non-linear optimization on the generated graph and to compute the variables configuration that best explains the factors. Note that this could also be extended to pose-landmark configuration to accomplish map refinement. This module can be implemented using any optimization library available. Our implementation embeds the graphical solver presented in [25].

Many other smaller components can be isolated in a SLAM system. Examples of them are the Map Clipper, which generates a local view of the input map, the Local Map Splitter, that triggers the creation of a new local map, or specific components to bootstrap the SLAM system - *e.g.* a module that performs Structure from Motion (SfM) in monocular configuration.

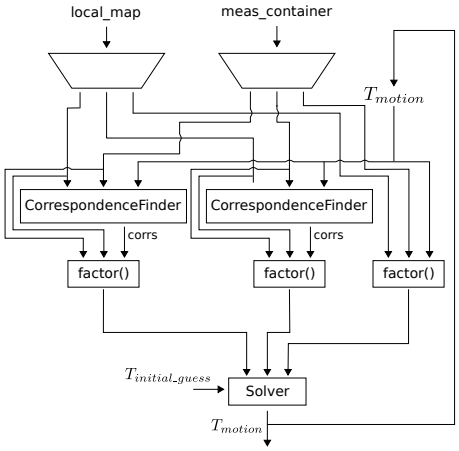
Given this overview of the generic taxonomy of SLAM systems, in the next section we propose our design to seamlessly embed multiple heterogeneous cues in a unified architecture.

IV. A Unified Multi-Sensor SLAM Architecture: Our Approach

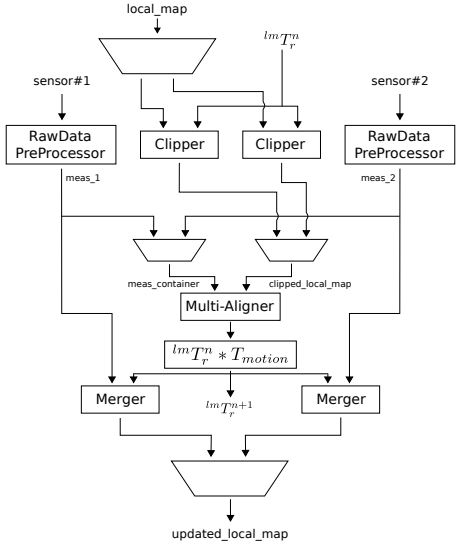
The core idea behind our approach is that the computation performed with the data coming from all the sensors contributes in estimating a single quantity: the current robot pose. Therefore, referring to Sec. III-A, the only core modules involved in this paradigm shift are the Aligner and the Tracker. Still, to allow heterogeneous sensors to coexist in a unique architecture, one should also define an appropriate data structure to represent multi-sensor measurements and local maps. Note that all the support modules are relatively agnostic to the fact



(a) Multi-sensor architecture.



(b) Multi-aligner.



(c) Multi-tracker.

Fig. 2: Top image: blueprint of our multi-sensor SLAM architecture. Each sensor will contribute to populate the *measurement property container*; this is fed into the Multi-Aligner to compute the relative motion of the robot; lately, the Multi-Tracker properly embeds each cue of the *measurement property container* into the *scene property container*; finally, the Graph-SLAM module arranges the local map into a factor graph, detects loop closures and optimizes the graph. Fig. 2b and Fig. 2c, instead, show a close-up of the secondary modules involved in the Multi-Aligner and Multi-Tracker, respectively.

that the system works in a single or multi-sensor configuration. Hence, in the remainder of this section, we first provide our design to store and manage heterogeneous measurements and local maps, then we address the changes in the aforementioned core modules.

A. Multi-Sensor Data Structures

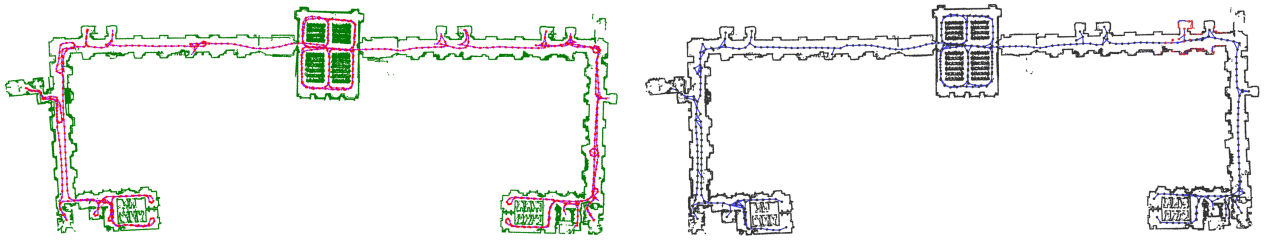
At the basis of our architecture, we have the concept of *Property*. A Property is an introspectable, serializable data element which is characterized by a data type, a value and a name within a containing structure called Dynamic Property Container (DPC). Properties can represent anything, from Plain Old Data (POD) structures - *e.g.* a number or a point cloud - to entire modules - named *Configurables*. Thanks to the introspection, accessing at runtime a specific Property in a DPC requires only the knowledge of its name within the container. Hence, we can store different types of measurement in a single DPC, resulting in what we call a *measurement property container*. This will contain the output of all Data Pre-processors currently instantiated in the pipeline. We can reuse the same machinery to store a multi-sensor local map - here indicated as *scene property container*. In this way, we are able to isolate different cues in a modular fashion, while at the same time, we can provide a single input/output data structure to the core modules.

B. Multi-Sensor Core Modules

Once addressed how to store dynamic and multi-sensor data, we tackle in this section the problem of processing them. In this scenario, the foundation of our approach relies on the concept of *slice*. A *slice* is a partial processing module, in charge of treating a specific cue of the architecture. Therefore, the Aligner and Tracker modules can cope with different sensor readings types by adding multiple slices to their design.

More in detail, a Multi-Sensor Aligner, called Multi-Aligner for brevity, is composed by a set of *aligner-slices* and a single Iterative Least-Squares (ILS) solver. These slices are in charge of producing factors out of sensor readings. Factors are embedded into a single factor graph that is optimized by the aforementioned ILS solver. As in the single-sensor case, the only variable to estimate is the robot relative motion. Still, all factors concerning different sensors will concurrently affect the estimation process.

The same reasoning is applied to the Multi-Sensor Tracker - shortened as Multi-Tracker. It is composed of a single *scene property container* as local map and multiple *tracker-slices*. Given a *measurement property container* and the local map, each tracker-slice will process and manage the appropriate cue from the containers. Fig. 2 provides a schematic illustration of the entire multi-sensor workflow.



(a) [24] (b) Our.

Fig. 3: Qualitative result of a real mapping session. Laser scans are recorded at 40Hz while odometry measurements at 50Hz. The output of system [24] is reported on the left image; the mean processing frame-rate is 34Hz. On the right image is reported the result obtained with our approach; in this case, the mean frame-rate is 256Hz - more than 5 times faster compared to sensors' frame-rates.

APPROACH	STAGE-0 $l = 43.191$ [m]	STAGE-1 $l = 79.146$ [m]	STAGE-2 $l = 480.741$ [m]	STAGE-3 $l = 627.709$ [m]
SRRG_MAPPER2D	ATE = 0.258 [m] RPE = 0.656 [m] 71.57 [Hz]	ATE = 0.295 [m] RPE = 0.777 [m] 67.28 [Hz]	ATE = 0.287 [m] RPE = 1.860 [m] 39.56 [Hz]	ATE = 0.251 [m] RPE = 1.529 [m] 28.35 [Hz]
OUR - SINGLE LIDAR	ATE = **0.037** [m] RPE = **0.059** [m] **289.51** [Hz]	ATE = 0.041 [m] RPE = **0.073** [m] **278.07** [Hz]	ATE = 0.109 [m] RPE = 0.298 [m] **251.51** [Hz]	ATE = 0.110 [m] RPE = 0.245 [m] **189.58** [Hz]
OUR - DOUBLE LIDAR	ATE = 0.047 [m] RPE = 0.084 [m] 153.88 [Hz]	ATE = **0.032** [m] RPE = 0.077 [m] 149.94 [Hz]	ATE = **0.072** [m] RPE = **0.256** [m] 144.42 [Hz]	ATE = **0.047** [m] RPE = **0.127** [m] 125.65 [Hz]

TABLE I: Comparison of Absolute Trajectory Error (ATE) and Relative Pose Error (RPE) between [24] and our approach. All approaches exploit also wheel odometry together with LiDAR data. For each approach, the last row reports the mean processing frame-rate.

To give a more practical insight of the architecture, we can address the case of having a 2D LiDAR pipeline with two rangefinders - one front-facing and the other one rear-facing placed at the same height. Therefore, the system will have two separate Data Pre-processors. Accordingly, the *measurement property container* will hold the two produced measurements and the local map will be composed of a single point cloud where to merge both the measurements. The Multi-Aligner is composed of two *slices*, one for the front rangefinder and the other for the rear one. Both of them will compute the data association between their current measurements and the local map. Each of the two *slices* will expose a set of constraints coming from the associations, that will be used by the Multi-Aligner to estimate the motion. The same applies to the Multi-Tracker. Both *slices* will take a point cloud from the *measurement property container* and will integrate it in the unique point cloud of the local map.

Note that thanks to this design, every other module in the architecture remains agnostic to the number of sensors involved in the pipeline. This means that one can potentially mix-up different modules in a plug-and-play fashion, without the need of further modification to the architecture.

C. Complementary Features

Besides the SLAM related benefits of such architecture, the design pattern that we employed brings advantages to other contexts. In this sense, thanks to the native serialization of each Property, we can automatically store the graph and the local map on disk. This is carried on by our custom-built library, which supports format-independent serialization of arbitrary data structures - called Basic Object Serialization System (BOSS). Furthermore, each processing module - named *Configurable* - exposes its parameters through Properties. This allows us to use BOSS to write a module configuration automatically on storage. Note that since Configurables can be stored in Properties, we are able to *instantiate* an entire pipeline reading from the configuration file. Although such a file is written in a human-readable format - based on JSON - editing a complex configuration by hand might result difficult. Still, thanks to the native introspection of the Property, we developed a graphical editor to edit BOSS configuration files on-the-go. We exploit the same Property features to provide the user with a shell to load, edit and run configurations. Finally, Configurables allow us to expose, via command-line, specific *actions* that can be triggered at runtime. For example, one can pause the pipeline and save the factor graph on storage at runtime, simply typing commands in our shell. Note that the proposed architecture embeds our custom optimization framework as back-end [25] that shares the same core functionalities - *e.g.* configuration management, serialization library. In this sense, also the graph-optimization module remains consistent with the rest of the architecture.

Furthermore, we provide also a unified viewing system based on OpenGL, that decouples processing and viewing. Using this API, one can either run the SLAM pipeline and its visualization on the same machine in two separate threads or run a SLAM pipeline on a cheap embedded system and stream the visual information to a more powerful machine that will act as a passive rendering viewport. Switching between

APPROACH	KITTI-00 $l = 3724.187$ [m]	ICL-LR-0 $l = 6.534$ [m]	EUROC MH-01
PROSLAM	ATE = 1.390 [m] RPE = 0.036 [m] **53.58** [Hz]	ATE = 0.049 [m] RPE = 0.003 [m] **62.97** [Hz]	ATE = 0.138 [m] RPE = 0.051 [m] **88.11** [Hz]
ORB-SLAM2	ATE = **1.256** [m] RPE = **0.029** [m] 14.26 [Hz] (4 threads)	ATE = **0.007** [m] RPE = 0.004 [m] 45.28 [Hz] (4 threads)	ATE = **0.038** [m] RPE = **0.050** [m] 16.62 [Hz] (4 threads)
OUR	ATE = 1.654 [m] RPE = 0.032 [m] 44.06 [Hz]	ATE = 0.016 [m] RPE = **0.002** [m] 49.98 [Hz]	ATE = 0.140 [m] RPE = 0.051 [m] 35.66 [Hz]

TABLE II: Comparison of ATE and RPE between ProSLAM [26], ORB-SLAM2 [27] and our approach. For each approach, the last row reports the mean processing frame-rate.

these two modalities does not require changes in the code. Currently, the multi-process viewing system relies on the ROS communication infrastructure to share data. The attached video shows the use of some of the features present in this section.

V. EXPERIMENTS

In this section, we provide both qualitative and quantitative results obtained though the *instantiation* of different SLAM pipelines embedded in our architecture. In particular, we show that completely different pipelines are able not only to coexist together, but they also achieve competitive results, and thus, using our architecture there is no apparent negative impact on the system performances. Thanks to the modular design of the proposed approach, the system natively allows us to combine heterogeneous sensors - *e.g.* 2D LiDAR and RGB-D - in a unique multi-sensor pipeline. In the remainder of this section, we show the results obtained with 2D LiDAR, Stereo and RGB-D SLAM pipeline instantiations, respectively. All the experiments have been performed on a desktop computer with Ubuntu 16.04 and GCC 7, equipped with an Intel i7-7700K CPU @ 4.20GHz and 32GB of RAM. Note that all the processing in our architecture is *single-threaded*.

A. 2D LiDAR

The 2D laser rangefinder is a very common sensor in SLAM and it has been employed since many years. Nowadays, it can be considered a cheap sensor and, thus, it is now spreading in consumer Robotics. Many open-source systems are available in this context [1], [2], [24], however, the majority of those are designed to be used with only one sensor cue and their extension might require a lot of effort. Our approach, instead, can be arranged to work with multiple rangefinders and/or in combination with wheel odometry by simply editing a configuration file.

To check the performance of our pipeline, we used simulated data, gathered using ROS Stage [28]. We recorded multiple sessions with different path lengths. The simulated differential-drive platform is equipped with 2 laser rangefinders (front and rear-facing, horizontally mounted) and wheel encoders that provide wheel odometry - all streaming data at 10 Hz. In Tab. I we reported a comparison between the approach of Lazaro *et al.* [24] - referred to as `srrg_mapper2d` - and our approach on the different sessions. Both the ATE and RPE are lower than the one obtained with the reference system. As expected, using the information coming from the second rangefinder increases the accuracy, while system speed remains more than 10 times faster than the platform sensor frame-rate. Finally, in Fig. 3 we report a qualitative result of a mapping session in our department. The map and the estimated trajectory obtained with [24] and our approach are comparable, however, our system has a higher frame-rate.

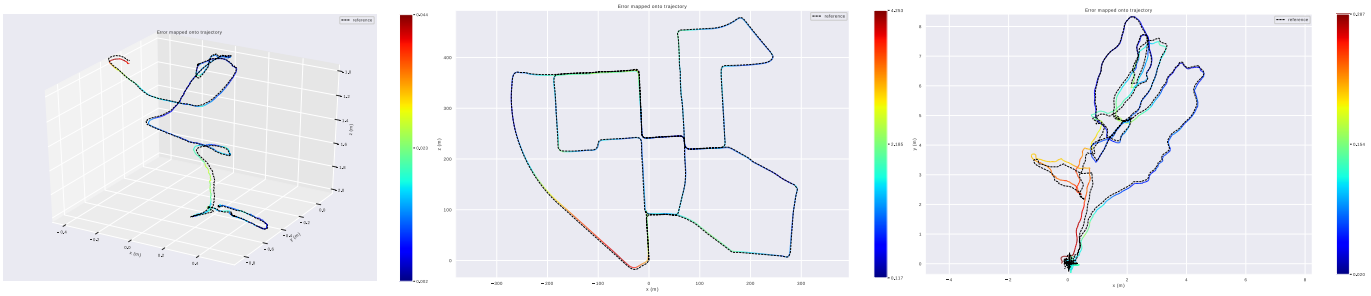
B. Visual SLAM: RGB-D and Stereo

Cameras are among the most frequently used sensors for SLAM, in particular stereo and depth cameras (*e.g.* IR and ToF) are increasingly being used in a variety of domains from robotics, automated driving to consumer electronics. In the proposed architecture, we addressed both Stereo and RGB-D data in a single-sensor fashion. To evaluate the performance of our framework we compare the results obtained on the KITTI dataset [29], on the ICL-NUIM dataset [30] and on the EuRoC MAV dataset [31] with two state-of-the-art systems namely ProSLAM [26] and ORB-SLAM2 [27]. In Tab. II we reported the values of the ATE and RPE on the considered sequences, while in Fig. 4 we show the plots of the trajectories computed with our system in three different scenarios. The comparison confirms that our architecture is able to achieve results comparable with other more mature state-of-the-art systems. For further comparisons and results of our experiments, we refer the reader to the repository wiki page[2].

VI. CONCLUSIONS

In this paper, we present a novel architecture that aims to standardize single- and multi-sensor SLAM. To achieve this goal, we analyzed the recurrent patterns in the context of SLAM systems, generating a taxonomy of sub-modules. Then, we exploited such taxonomy to provide the user with the ability to easily integrate heterogeneous sensors in a single unified pipeline. The paper exposes the design patterns and the data structures used in our implementation, presenting a modular and easy-to-extend architecture. In our opinion, the latter could also have a major educational impact.

[2]Wiki: https://github.com/srrg-sapienza/srrg2_slam_interfaces/wiki



(a) ICL `lr-0`. (b) KITTI `00`. (c) EuRoC `MH-01`.

Fig. 4: Trajectories generated by our architecture using visual SLAM with Stereo and RGB-D.

Finally, we conducted a set of comparative experiments to show that our architecture has no drawbacks on the accuracy of the estimation nor on the runtime performances. In fact, even if pure benchmarking is out of the scope of this work, the results obtained are comparable with ad-hoc state-of-the-art implementations of 2D LiDAR, RGB-D and Stereo SLAM.

REFERENCES

[1] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Trans. on Robotics (TRO)*, 23(1):34–46, 2007.

[2] Wolfgang Hess, Damon Kohler, Holger Rapp, and Daniel Andor. Real-time loop closure in 2d lidar slam. In *IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1271–1278. IEEE, 2016.

[3] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Trans. on Robotics (TRO)*, 32(6):1309–1332, 2016.

[4] Mirco Colosi, Sebastian Haug, Peter Biber, Kai O Arras, and Giorgio Grisetti. Better Lost in Transition Than Lost in Space: SLAM State Machine. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*. IEEE, 2019.

[5] Simon Lynen, Markus W Achtelik, Stephan Weiss, Margarita Chli, and Roland Siegwart. A robust and modular multi-sensor fusion approach applied to mav navigation. In *2013 IEEE/RSJ international conference on intelligent robots and systems*, pages 3923–3929. IEEE, 2013.

[6] Agostino Martinelli. Closed-form solution of visual-inertial structure from motion. *Int. J. of Computer Vision*, 106(2):138–152, 2014.

[7] Raúl Mur-Artal and Juan D Tardós. Visual-inertial monocular slam with map reuse. *IEEE Robotics and Automation Letters (RA-L)*, 2(2), 2017.

[8] Tong Qin, Peiliang Li, and Shaojie Shen. Vins-mono: A robust and versatile monocular visual-inertial state estimator. *IEEE Trans. on Robotics (TRO)*, 34(4):1004–1020, 2018.

[9] Christian Forster, Luca Carlone, Frank Dellaert, and Davide Scaramuzza. Imu preintegration on manifold for efficient visual-inertial maximum-a-posteriori estimation. Georgia Institute of Technology, 2015.

[10] Taihú Pire, Thomas Fischer, Gastón Castro, Pablo De Cristóforis, Javier Civera, and Julio Jacobo Berlles. S-ptam: Stereo parallel tracking and mapping. *Robotics and Autonomous Systems*, 93:27–42, 2017.

[11] Antoni Rosinol, Marcus Abate, Yun Chang, and Luca Carlone. Kimera: an open-source library for real-time metric-semantic localization and mapping. *IEEE Intl. Conf. on Robotics & Automation (ICRA)*, 2020.

[12] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. In *IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 2174–2181. IEEE, 2015.

[13] Paul Newman, David Cole, and Kin Ho. Outdoor SLAM using visual appearance and laser ranging. In *IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1180–1187. IEEE, 2006.

[14] T. Schneider, M. T. Dymczyk, M. Fehr, K. Egger, S. Lynen, I. Gilitschenski, and R. Siegwart. maplab: An open framework for research in visual-inertial mapping and localization. *IEEE Robotics and Automation Letters (RA-L)*, 2018.

[15] Blanco-Claraco J.L. A Modular Optimization Framework for Localization and Mapping. In *Proc. of Robotics: Science and Systems (RSS)*, FreiburgimBreisgau, Germany, June 2019.

[16] Labbé, Mathieu and Michaud, François. RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation. *Journal of Field Robotics (JFR)*, 36(2):416–446, 2019.

[17] Mathieu Labbe and Francois Michaud. Appearance-based loop closure detection for online large-scale and long-term operation. *IEEE Trans. on Robotics (TRO)*, 29(3):734–745, 2013.

[18] Georges Younes, Daniel Asmar, Elie Shammas, and John Zelek. Keyframe-based monocular slam: design, survey, and future directions. *Journal on Robotics and Autonomous Systems (RAS)*, 98:67–88, 2017.

[19] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based SLAM. *IEEE Trans. on Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.

[20] Dominik Schlegel and Giorgio Grisetti. HBST: A hamming distance embedding binary search tree for feature-based visual place recognition. *IEEE Robotics and Automation Letters (RA-L)*, 3(4):3741–3748, 2018.

[21] Gálvez-López, Dorian and Tardós, J. D. Bags of Binary Words for Fast Place Recognition in Image Sequences. *IEEE Trans. on Robotics (TRO)*, 28(5):1188–1197, October 2012.

[22] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Comm. of the ACM*, 18(9):509–517, 1975.

[23] Belur V Dasarathy. Nearest neighbor (nn) norms: Nn pattern classification techniques. *IEEE Computer Society Tutorial*, 1991.

[24] María T Lázaro, Roberto Capobianco, and Giorgio Grisetti. Efficient long-term mapping in dynamic environments. In *IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, pages 153–160. IEEE, 2018.

[25] Giorgio Grisetti, Tiziano Guadagnino, Irvin Aloise, Mirco Colosi, Bartolomeo Della Corte, and Dominik Schlegel. Least squares optimization: from theory to practice. *Robotics*, 9(3):51, July 2020.

[26] Dominik Schlegel, Mirco Colosi, and Giorgio Grisetti. Proslam: Graph slam from a programmer's perspective. In *IEEE Intl. Conf. on Robotics & Automation (ICRA)*, pages 1–9. IEEE, 2018.

[27] Raul Mur-Artal and Juan D Tardós. Orb-slam2: An open-source slam system for monocular, stereo, and rgb-d cameras. *IEEE Trans. on Robotics (TRO)*, 33(5):1255–1262, 2017.

[28] R. Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2-4):189–208, 2008.

[29] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *Intl. Journal of Robotics Research (IJRR)*, 32(11):1231–1237, 2013.

[30] A. Handa, T. Whelan, J.B. McDonald, and A.J. Davison. A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM. In *IEEE Intl. Conf. on Robotics & Automation (ICRA)*, Hong Kong, China, May 2014.

[31] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Joern Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The euroc micro aerial vehicle datasets. *The International Journal of Robotics Research*, 2016.